

**MapReduce Service (MRS)**

**3.3.1-LTS**

# **Development Guide**

**Issue** 02

**Date** 2024-11-01



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

## Trademarks and Permissions



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

# Huawei Cloud Computing Technologies Co., Ltd.

Address:      Huawei Cloud Data Center Jiaoxinggong Road  
                  Qianzhong Avenue  
                  Gui'an New District  
                  Gui Zhou 550029  
                  People's Republic of China

Website:      <https://www.huaweicloud.com/intl/en-us/>

# Contents

---

<b>1 Security Mode.....</b>	<b>1</b>
1.1 Description.....	1
1.2 Obtaining Sample Projects from Huawei Mirrors.....	1
1.3 Sample Projects of MRS Components.....	5
1.4 Security Authentication.....	14
1.4.1 Security Authentication Principles and Mechanisms.....	14
1.4.2 Preparing a Developer Account.....	18
1.4.3 Handling an Authentication Failure.....	25
1.5 ClickHouse Development Guide.....	27
1.5.1 Overview.....	27
1.5.1.1 Introduction to ClickHouse.....	27
1.5.1.2 Basic Concepts.....	28
1.5.1.3 Development Process.....	28
1.5.2 Environment Preparations.....	30
1.5.2.1 Preparing the Development and Operating Environment.....	30
1.5.2.2 Configuring and Importing a Sample Project.....	33
1.5.2.3 Configuring and Importing a Transaction Sample Project.....	38
1.5.2.4 Configuring and Importing the Spring Boot Sample Project.....	41
1.5.3 Application Development.....	42
1.5.3.1 Typical Application Scenario.....	42
1.5.3.2 Development Guideline.....	43
1.5.4 Sample Code.....	43
1.5.4.1 Setting Properties.....	43
1.5.4.2 Establishing a Connection.....	43
1.5.4.3 Creating a Database.....	44
1.5.4.4 Creating a Table.....	44
1.5.4.5 Inserting Data.....	44
1.5.4.6 Querying Data.....	45
1.5.4.7 Deleting a Table.....	45
1.5.4.8 Using BalanceDataSource.....	45
1.5.5 Connecting the Python Driver to the ClickHouse in Security Mode.....	48
1.5.6 Application Commissioning.....	54
1.5.6.1 Commissioning Applications on Windows.....	54

1.5.6.2 Commissioning Applications on Linux.....	59
1.5.6.3 Commissioning the Spring Boot Sample Project.....	64
1.5.6.3.1 Commissioning the Spring Boot Project in Windows.....	64
1.5.6.3.2 Commissioning the Spring Boot Project in Linux.....	68
1.6 Doris Development Guide.....	69
1.6.1 Overview.....	69
1.6.1.1 Doris Introduction.....	69
1.6.1.2 Common Concepts.....	70
1.6.1.3 Development Process.....	72
1.6.1.4 Doris Sample Project Introduction.....	75
1.6.2 Environment Preparation.....	76
1.6.2.1 Preparing for Development Environment.....	76
1.6.2.2 Preparing the Configuration Files for Connecting to the Cluster.....	78
1.6.2.3 Configuring and Importing JDBC or Stream Load Sample Projects.....	80
1.6.2.4 Configuring and Importing SpringBoot Sample Projects.....	83
1.6.3 Doris Application Development.....	85
1.6.3.1 Using Doris Through JDBC or DBalancer.....	85
1.6.3.1.1 Service Scenario Description.....	85
1.6.3.1.2 Application Development Approach.....	85
1.6.3.1.3 Setting Up a Connection.....	86
1.6.3.1.4 Creating a Database.....	87
1.6.3.1.5 Creating a Table.....	87
1.6.3.1.6 Inserting Data.....	87
1.6.3.1.7 Querying Data.....	88
1.6.3.1.8 Deleting a Table.....	88
1.6.3.1.9 Deleting a Database.....	88
1.6.3.2 Loading Data to a Doris Table with Stream Load.....	89
1.6.4 Application Commissioning.....	91
1.6.4.1 Commissioning an Application in Windows.....	91
1.6.4.1.1 Compiling and Running an Application.....	91
1.6.4.1.2 Viewing Windows Commissioning Results.....	98
1.6.4.2 Commissioning an Application in Linux.....	101
1.6.4.2.1 Compiling and Running Applications.....	101
1.6.4.2.2 Viewing Linux Commissioning Results.....	104
1.7 Elasticsearch Development Guide .....	107
1.7.1 Overview.....	107
1.7.1.1 Application Development Overview.....	107
1.7.1.2 Concepts.....	107
1.7.1.3 Application Development Process.....	108
1.7.2 Environment Preparation.....	110
1.7.2.1 Preparing Development and Operating Environment.....	110
1.7.2.2 Configuring and Importing Sample Projects.....	113

1.7.2.2.1 Overview.....	114
1.7.2.2.2 Importing a RestClient Sample Project.....	114
1.7.2.2.3 Importing a SpringBoot Sample Project.....	116
1.7.2.3 Preparing for Security Authentication.....	118
1.7.3 Development Process.....	119
1.7.3.1 Typical Application Scenarios.....	119
1.7.3.2 Low Level Rest Client Sample Code.....	119
1.7.3.2.1 Connecting the Elasticsearch Client to an Elasticsearch Cluster.....	120
1.7.3.2.2 Querying Cluster Information.....	120
1.7.3.2.3 Checking Whether the Specified Index Exists.....	121
1.7.3.2.4 Creating an Index with the Desired Number of Shards.....	121
1.7.3.2.5 Writing Index Data.....	122
1.7.3.2.6 Writing Data in Batches.....	123
1.7.3.2.7 Batch write data to specified routes.....	123
1.7.3.2.8 Querying Index Information.....	124
1.7.3.2.9 Deleting an Index.....	125
1.7.3.2.10 Deleting Some Documents in the Index.....	125
1.7.3.2.11 Deleting All Documents in the Index.....	126
1.7.3.2.12 Refreshing an Index.....	126
1.7.3.2.13 Multi-Thread Example.....	127
1.7.3.2.14 Pre-sorting Indices.....	128
1.7.3.3 High Level Rest Client Sample Code.....	130
1.7.3.3.1 Connecting the Client to a Cluster.....	130
1.7.3.3.2 Querying Cluster Status Information.....	131
1.7.3.3.3 Writing Index Data.....	131
1.7.3.3.4 Operations in Batches.....	133
1.7.3.3.5 Batch write data to specified routes.....	133
1.7.3.3.6 Updating Index Information.....	134
1.7.3.3.7 Querying Index Information.....	135
1.7.3.3.8 Searching for Document Information.....	135
1.7.3.3.9 Searching for Document Information by Using a Cursor.....	136
1.7.3.3.10 Clearing a Cursor.....	136
1.7.3.3.11 Deleting an Index.....	137
1.7.3.3.12 Multi-Thread Request.....	137
1.7.3.3.13 BulkProcessor Batch Import Example.....	138
1.7.3.3.14 Pre-sorting Indices.....	139
1.7.3.4 OpenDistro JDBC Sample Code.....	141
1.7.3.4.1 Connecting the Client to a Cluster.....	141
1.7.3.4.2 Querying an Index.....	142
1.7.3.4.3 Deleting Index Information.....	143
1.7.3.4.4 Functions.....	144
1.7.4 Commissioning Process.....	150

1.7.4.1 Commissioning Applications on Windows.....	150
1.7.4.1.1 Commissioning a RestClient Application.....	150
1.7.4.1.2 Commissioning a SpringBoot Program.....	159
1.7.4.2 Commissioning Applications in Linux.....	162
1.7.4.2.1 Commissioning a RestClient Application.....	162
1.7.4.2.2 Commissioning a SpringBoot Program.....	169
1.8 Flink Development Guide.....	172
1.8.1 Overview.....	172
1.8.1.1 Application Development.....	172
1.8.1.2 Basic Concepts.....	174
1.8.1.3 Development Process.....	175
1.8.2 Environment Preparation.....	176
1.8.2.1 Preparing Development and Operating Environment.....	177
1.8.2.2 Configuring and Importing Sample Projects.....	180
1.8.2.3 Creating a Project (Optional).....	199
1.8.2.4 Preparing for Security Authentication.....	202
1.8.2.5 Configuring a Spring Boot Sample Project.....	209
1.8.3 Developing an Application.....	213
1.8.3.1 DataStream Application.....	213
1.8.3.1.1 Scenarios.....	213
1.8.3.1.2 Java Sample Code.....	214
1.8.3.1.3 Scala Sample Code.....	217
1.8.3.2 Interconnecting with Kafka.....	219
1.8.3.2.1 Scenario Description.....	219
1.8.3.2.2 Java Sample Code.....	222
1.8.3.2.3 Scala Sample Code.....	224
1.8.3.3 Asynchronous Checkpoint Mechanism.....	226
1.8.3.3.1 Scenarios.....	226
1.8.3.3.2 Java Sample Code.....	227
1.8.3.3.3 Scala Sample Code.....	230
1.8.3.4 Job Pipeline Program.....	232
1.8.3.4.1 Scenario.....	232
1.8.3.4.2 Java Sample Code.....	234
1.8.3.4.3 Scala Sample Code.....	237
1.8.3.5 JOIN Operation between Configuration Tables and Streams.....	239
1.8.3.5.1 Scenario Description.....	239
1.8.3.5.2 Java Sample Code.....	242
1.8.3.5.3 Scala Sample Code.....	252
1.8.3.6 Stream SQL Join Program.....	260
1.8.3.6.1 Scenario.....	260
1.8.3.6.2 Java Sample Code.....	261
1.8.3.6.3 Scala Sample Code.....	263

1.8.3.7 Submitting a SQL Job Using Flink Jar.....	266
1.8.3.7.1 Scenario Description.....	266
1.8.3.7.2 Java Sample Code.....	266
1.8.3.8 FlinkServer REST API JavaExample.....	267
1.8.3.8.1 Scenario Description.....	267
1.8.3.8.2 Java Sample Code.....	267
1.8.3.8.3 Accessing Flinkserver RESTful API as a Proxy User.....	268
1.8.3.9 Flink Reading Data from and Writing Data to HBase.....	269
1.8.3.9.1 Scenario Description.....	269
1.8.3.9.2 Java Sample Code.....	269
1.8.3.10 Flink Reading Data from and Writing Data to Hudi.....	274
1.8.3.10.1 Scenario Description.....	274
1.8.3.10.2 Java Sample Code.....	275
1.8.3.11 Python Development Examples.....	277
1.8.3.11.1 Submitting a Regular Job Using Python.....	277
1.8.3.11.2 Submitting a SQL Job Using Python.....	281
1.8.4 Debugging the Application.....	284
1.8.4.1 Compiling and Running the Application.....	284
1.8.4.2 Viewing the Debugging Result.....	294
1.8.4.3 Running a Spring Boot Sample Project and Viewing Results.....	300
1.8.5 More Information.....	301
1.8.5.1 Introduction to Common APIs.....	301
1.8.5.1.1 Java.....	301
1.8.5.1.2 Scala.....	318
1.8.5.2 Overview of RESTful APIs.....	332
1.8.5.3 Overview of Savepoints CLI.....	335
1.8.5.4 Introduction to Flink Client CLI.....	337
1.8.5.5 FAQ.....	338
1.8.5.5.1 Savepoints-related Problems.....	338
1.8.5.5.2 What If the Chrome Browser Cannot Display the Title.....	339
1.8.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11.....	340
1.8.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large.....	341
1.8.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home.....	343
1.8.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?.....	344
1.8.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?.....	345
1.8.5.5.8 Why Cannot I Access the Apache Flink Dashboard?.....	345
1.8.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?.....	346
1.8.5.5.10 Incorrect GLIBC Version.....	347
1.9 GraphBase Development Guide.....	348
1.9.1 Overview.....	348

1.9.1.1 Application Development Overview.....	348
1.9.1.2 Basic Concepts.....	349
1.9.1.3 Development Process.....	349
1.9.2 Environment Preparation.....	352
1.9.2.1 REST API Development Environment Preparation.....	352
1.9.2.1.1 Preparing Development and Operating Environment.....	352
1.9.2.1.2 Configuring and Importing Sample Projects.....	356
1.9.2.1.3 Preparing for Security Authentication.....	359
1.9.2.2 Gremlin API Development Environment Preparation.....	361
1.9.2.2.1 Preparing Development and Operating Environment.....	361
1.9.2.2.2 Configuring and Importing a Piece of Sample Code.....	364
1.9.2.2.3 Preparing for Security Authentication.....	364
1.9.3 Developing an Application.....	365
1.9.3.1 Typical Application Scenario.....	365
1.9.3.2 Preparing Data Files.....	368
1.9.3.2.1 Compiling a Schema File (XML).....	368
1.9.3.2.2 Compiling a Data File (CSV).....	371
1.9.3.2.3 Compiling a Data Description File (DESC).....	372
1.9.3.2.4 Compiling a Graph Mapping Rule File (.mapper).....	373
1.9.3.3 Importing Data.....	386
1.9.3.3.1 Importing Data in Batches Using Tools.....	387
1.9.3.3.2 Importing Data in Real Time Using Tool.....	389
1.9.3.3.3 Using the Hive Data Import Tool.....	394
1.9.3.3.4 Data Export.....	396
1.9.3.5 REST API Development.....	411
1.9.3.6 REST API Invocation Examples and Running Results.....	412
1.9.3.6.1 Example.....	412
1.9.3.6.2 Creating or Deleting a Graph.....	413
1.9.3.6.3 Creating a Schema by Uploading an XML File.....	414
1.9.3.6.4 Creating a Schema.....	414
1.9.3.6.5 Creating an Index.....	418
1.9.3.6.6 Creating and Querying a Vertex or an Edge.....	421
1.9.3.6.7 Performing a Full Graph Query.....	427
1.9.3.6.8 Performing a Path Query.....	430
1.9.3.6.9 Performing a Line Expansion Query.....	438
1.9.3.6.10 Analyzing the Typical Application Scenario.....	440
1.9.3.7 Gremlin Console.....	444
1.9.3.8 Gremlin Java.....	445
1.9.3.8.1 Connecting Gremlin Servers and Clients.....	445
1.9.3.8.2 Running Gremlin Java API Example Code.....	446
1.9.3.9 Gremlin Application Scenarios.....	446
1.9.3.9.1 Querying a Specified Graph Object.....	446

1.9.3.9.2 Queries One or More Properties.....	447
1.9.3.9.3 Traversing a Graph Object.....	447
1.9.3.9.4 Traversing a Path.....	448
1.9.3.9.5 Filtering by Condition.....	449
1.9.3.9.6 Sorting the Result.....	451
1.9.3.9.7 Displaying the Result in Multiple Pages.....	451
1.9.3.9.8 Calculating the Intersection Between Two Data Sets.....	451
1.9.3.9.9 Calculating the Union of Two Data Sets.....	451
1.9.3.9.10 Using Statistical Functions.....	451
1.9.4 Application Commissioning.....	452
1.9.4.1 Commissioning an Application in Windows.....	452
1.9.4.1.1 Compiling and Running an Application.....	452
1.9.4.1.2 Viewing Windows Commissioning Results.....	452
1.9.4.2 Commissioning an Application in Linux.....	453
1.9.4.2.1 Compiling and Running an Application.....	453
1.9.4.2.2 Viewing Linux Commissioning Results.....	455
1.9.5 More Information.....	456
1.9.5.1 Common APIs.....	456
1.9.5.1.1 REST API.....	456
1.9.5.1.2 Gremlin API.....	457
1.10 HBase Development Guide.....	457
1.10.1 Overview.....	457
1.10.1.1 Application Development Overview.....	457
1.10.1.2 Common Concepts.....	458
1.10.1.3 Development Process.....	458
1.10.1.4 HBase Sample Project Introduction.....	461
1.10.2 Environment Preparation.....	463
1.10.2.1 Preparing for Development Environment.....	463
1.10.2.2 Preparing the Connection Cluster Configuration File.....	465
1.10.2.3 Configuring and Importing Sample Projects.....	469
1.10.2.4 Preparing for Security Authentication.....	481
1.10.2.4.1 Security Authentication for HBase Data Read and Write and Global Secondary Index Example (Single-Cluster Scenario).....	481
1.10.2.4.2 Security Authentication for HBase Data Read and Write and Global Secondary Index Example (Mutual Trust Scenarios).....	483
1.10.2.4.3 Accessing HBase REST Service Security Authentication.....	485
1.10.2.4.4 Security Authentication for Accessing the ThriftServer Service.....	486
1.10.2.4.5 Security Authentication for Accessing Multiple ZooKeepers.....	488
1.10.2.4.6 Security Authentication for Interconnecting HBase/Phoenix with Spring Boot.....	489
1.10.3 Developing an Application.....	490
1.10.3.1 HBase Data Read/Write Sample Program.....	490
1.10.3.1.1 Service Scenario Description.....	490
1.10.3.1.2 Application Development Approach.....	491

1.10.3.1.3 Creating Configuration.....	492
1.10.3.1.4 Creating Connection.....	493
1.10.3.1.5 Creating a Table.....	493
1.10.3.1.6 Deleting a Table.....	495
1.10.3.1.7 Modifying a Table.....	496
1.10.3.1.8 Inserting Data.....	497
1.10.3.1.9 Deleting Data.....	498
1.10.3.1.10 Reading Data Using Get.....	499
1.10.3.1.11 Reading Data Using Scan.....	500
1.10.3.1.12 Filtering Data.....	501
1.10.3.1.13 Creating a Secondary Index.....	502
1.10.3.1.14 Deleting an Index.....	505
1.10.3.1.15 Secondary Index-based Query.....	505
1.10.3.1.16 Multi-Point Region Division.....	508
1.10.3.1.17 Creating a Phoenix Table.....	509
1.10.3.1.18 Writing Data to the PhoenixTable.....	510
1.10.3.1.19 Reading the PhoenixTable.....	511
1.10.3.1.20 Using HBase Dual-Read Capability.....	512
1.10.3.1.21 Configuring Log4j Log Output.....	515
1.10.3.2 HBase Global Secondary Index Sample Program.....	516
1.10.3.2.1 Service Scenario Description.....	516
1.10.3.2.2 Creating HBase Global Secondary Indexes.....	516
1.10.3.2.3 Querying Global Secondary Indexes.....	518
1.10.3.2.4 Querying Based on Global Secondary Indexes.....	518
1.10.3.2.5 Disabling Global Secondary Indexes.....	519
1.10.3.2.6 Deleting Global Secondary Indexes.....	520
1.10.3.3 Sample Program for Preloading Meta Tables When Request Concurrency Is High.....	520
1.10.3.3.1 Service Scenario Description.....	520
1.10.3.3.2 Preloading Meta Tables When Request Concurrency Is High.....	521
1.10.3.4 HBase Rest API Invoking Sample Program.....	523
1.10.3.4.1 Querying Cluster Information Using REST.....	523
1.10.3.4.2 Obtaining All Tables Using REST.....	523
1.10.3.4.3 Operate Namespaces Using REST.....	524
1.10.3.4.4 Operate Tables Using REST.....	525
1.10.3.5 Accessing the HBase ThriftServer Sample Program.....	526
1.10.3.5.1 Accessing the ThriftServer Operation Table.....	526
1.10.3.5.2 Accessing ThriftServer to Write Data.....	527
1.10.3.5.3 Accessing ThriftServer to Read Data.....	528
1.10.3.6 Sample Program for HBase to Access Multiple ZooKeepers.....	530
1.10.3.6.1 Accessing Multiple ZooKeepers.....	530
1.10.4 Application Commissioning.....	531
1.10.4.1 Commissioning an Application in Windows.....	531

1.10.4.1.1 Compiling and Running an Application.....	531
1.10.4.1.2 Viewing Windows Commissioning Results.....	536
1.10.4.2 Commissioning an Application in Linux.....	538
1.10.4.2.1 Compiling and Running an Application When a Client Is Installed.....	538
1.10.4.2.2 Compiling and Running an Application When No Client Is Installed.....	541
1.10.4.2.3 Viewing Linux Commissioning Results.....	542
1.10.5 More Information.....	543
1.10.5.1 SQL Query.....	543
1.10.5.2 HBase Dual-Read Configuration Items.....	545
1.10.5.3 External Interfaces.....	548
1.10.5.3.1 Shell.....	548
1.10.5.3.2 Java API.....	549
1.10.5.3.3 SQLLine.....	553
1.10.5.3.4 JDBC APIs.....	560
1.10.5.3.5 Web UI.....	560
1.10.5.4 Phoenix Command Line.....	563
1.10.5.5 FAQs.....	564
1.10.5.5.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?.....	564
1.10.5.5.2 What Are the Application Scenarios of the Bulkload and put Data-loading Modes?.....	565
1.10.5.5.3 An Error Occurred When Building a JAR Package.....	565
1.10.5.5.4 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "java.lang.OutOfMemoryError: Direct buffer memory" Is Displayed in the Error Information?.....	566
1.10.5.5.5 Why Does the Error Information Contain "GSSEException" When an IBM JDK Is Used on a Client to Connect to an HBase Cluster?.....	566
1.11 HDFS Development Guide.....	567
1.11.1 Overview.....	567
1.11.1.1 Introduction to HDFS.....	567
1.11.1.2 Basic Concepts.....	568
1.11.1.3 Development Process.....	569
1.11.2 Environment Preparation.....	571
1.11.2.1 Preparing Development and Operating Environment.....	571
1.11.2.2 Configuring and Importing Sample Projects.....	574
1.11.2.3 Preparing the Authentication Mechanism.....	581
1.11.3 Developing the Project.....	583
1.11.3.1 Scenario.....	583
1.11.3.2 Development Idea.....	583
1.11.3.3 Declare the Example Codes.....	584
1.11.3.3.1 Initializing the HDFS.....	584
1.11.3.3.2 Creating Directories.....	586
1.11.3.3.3 Writing Data into a File.....	587
1.11.3.3.4 Appending Data to a File.....	588

1.11.3.3.5 Reading Data from a File.....	589
1.11.3.3.6 Deleting a File.....	589
1.11.3.3.7 Deleting Directories.....	590
1.11.3.3.8 Multi-Thread Tasks.....	591
1.11.3.3.9 Setting Storage Policies.....	592
1.11.3.3.10 Colocation.....	593
1.11.4 Commissioning the Application.....	596
1.11.4.1 Commissioning an Application in the Windows Environment.....	596
1.11.4.1.1 Compiling and Running an Application.....	596
1.11.4.1.2 Checking the Commissioning Result.....	598
1.11.4.2 Commissioning an Application in the Linux Environment.....	600
1.11.4.2.1 Compiling and Running an Application With the Client Installed.....	600
1.11.4.2.2 Compiling and Running an Application With the Client Not Installed.....	601
1.11.4.2.3 Checking the Commissioning Result.....	602
1.11.5 More Information.....	603
1.11.5.1 Common API Introduction.....	603
1.11.5.1.1 Java API.....	603
1.11.5.1.2 C API.....	607
1.11.5.1.3 HTTP REST API.....	611
1.11.5.2 Shell Command Introduce.....	620
1.12 HetuEngine Development Guide.....	622
1.12.1 Overview.....	622
1.12.1.1 Introduction to HetuEngine.....	623
1.12.1.2 Concepts.....	623
1.12.1.3 Connection Modes.....	624
1.12.1.4 Development Process.....	624
1.12.2 Environment Preparation.....	626
1.12.2.1 Preparing the Development and Running Environment.....	626
1.12.2.2 Configuring and Importing Sample ProjectsConfiguring and Importing Sample Projects.....	631
1.12.2.3 Setting Up a Python3 Project.....	632
1.12.2.4 Setting Up a Spring Boot Sample Project.....	633
1.12.2.5 Preparing for Security Authentication.....	634
1.12.2.5.1 KeyTab File Authentication Using HSFabric.....	634
1.12.2.5.2 Username and Password Authentication Using HSFabric.....	635
1.12.2.5.3 Username and Password Authentication Using HSBroker.....	635
1.12.3 Application Development.....	635
1.12.3.1 Typical Application Scenario.....	635
1.12.3.2 Java Sample Code.....	635
1.12.3.2.1 KeyTab File Authentication Using HSFabric.....	636
1.12.3.2.2 Username and Password Authentication Using HSFabric.....	638
1.12.3.2.3 Querying SQL Tasks using JDBC.....	640
1.12.3.2.4 Username and Password Authentication Using HSBroker.....	643

1.12.3.3 Python3 Sample Code.....	644
1.12.3.3.1 Username and Password Authentication Using HSBroker.....	644
1.12.3.3.2 Username and Password Authentication Using HSFabric.....	645
1.12.3.3.3 KeyTab File Authentication Using HSFabric.....	647
1.12.4 Application Commissioning.....	648
1.12.4.1 Commissioning Applications on Windows.....	649
1.12.4.2 Commissioning Applications on Linux.....	650
1.12.4.3 Commissioning a Python3 Application.....	652
1.12.4.4 Commissioning a Spring Boot Application.....	653
1.13 Hive Development Guide.....	655
1.13.1 Overview.....	655
1.13.1.1 Application Development Overview.....	655
1.13.1.2 Common Concepts.....	656
1.13.1.3 Required Permissions.....	656
1.13.1.4 Development Process.....	660
1.13.2 Preparing the Environment.....	661
1.13.2.1 Preparing Development and Operating Environment.....	661
1.13.2.2 Configuring the JDBC Sample Project.....	666
1.13.2.3 Configuring the Hcatalog Sample Project.....	670
1.13.2.4 Configuring the Python Sample Project.....	673
1.13.2.5 Configuring the Python3 Sample Project.....	674
1.13.2.6 Configuring a Spring Boot Sample Project.....	676
1.13.3 Developing an Application.....	680
1.13.3.1 Typical Scenario Description.....	680
1.13.3.2 Example Codes.....	682
1.13.3.2.1 Creating a Table.....	682
1.13.3.2.2 Loading Data.....	684
1.13.3.2.3 Querying Data.....	685
1.13.3.2.4 UDF.....	686
1.13.3.2.5 Example Program Guide.....	688
1.13.3.2.6 Accessing Multiple ZooKeepers.....	692
1.13.3.4 Commissioning Applications.....	693
1.13.4.1 Running JDBC and Viewing Results.....	693
1.13.4.2 Running HCatalog and Viewing Results.....	695
1.13.4.3 Running Python and Viewing Results.....	697
1.13.4.4 Running Python3 and Viewing Results.....	698
1.13.4.5 Running a Spring Boot Sample Project and Viewing Results.....	699
1.13.5 More Information.....	701
1.13.5.1 Interface Reference.....	701
1.13.5.1.1 JDBC.....	701
1.13.5.1.2 Hive SQL.....	701
1.13.5.1.3 WebHCat.....	705

1.13.5.2 FAQ.....	729
1.13.5.2.1 A Message Is Displayed Stating "Unable to read HiveServer2 configs from ZooKeeper" During the Use of the Secondary Development Program.....	729
1.13.5.2.2 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions.....	729
1.13.5.2.3 Hive SQL Is Incompatible with SQL2003 Standards.....	730
1.13.5.2.4 "version 'GLIBCXX_3.4.21' not found" Exception Occurs When the Python 3 Secondary Development Program Is Used to Access Hive.....	734
1.14 IoTDB Development Guide.....	734
1.14.1 Overview.....	734
1.14.1.1 Application Development Overview.....	734
1.14.1.2 Basic Concepts.....	734
1.14.1.3 Development Process.....	736
1.14.1.4 IoTDB Sample Project.....	737
1.14.2 Environment Preparations.....	738
1.14.2.1 Preparing the Environment.....	738
1.14.2.2 Preparing the Configuration Files for Connecting to the Cluster.....	740
1.14.2.3 Configuring and Importing a Sample Project.....	742
1.14.3 Application Development.....	749
1.14.3.1 IoTDB JDBC.....	749
1.14.3.1.1 Java Example Code.....	749
1.14.3.1.2 Using the keytab File for JDBC Authentication.....	750
1.14.3.2 IoTDB Session.....	752
1.14.3.2.1 Java Example Code.....	752
1.14.3.2.2 Using the Keytab File for Session Authentication.....	754
1.14.3.3 IoTDB Flink.....	756
1.14.3.3.1 FlinkIoTDBSink.....	756
1.14.3.3.2 FlinkIoTDBSource.....	758
1.14.3.4 IoTDB Kafka.....	760
1.14.3.4.1 Java Example Code.....	760
1.14.3.5 IoTDB UDF Program.....	762
1.14.3.5.1 IoTDB UDF Sample Code.....	762
1.14.4 Application Commissioning.....	763
1.14.4.1 Commissioning Applications on Windows.....	763
1.14.4.1.1 Compiling and Running Applications.....	763
1.14.4.1.2 Viewing Commissioning Results.....	767
1.14.4.2 Commissioning JDBC and Session Applications on Linux.....	769
1.14.4.2.1 Compiling and Running Applications.....	769
1.14.4.2.2 Viewing Commissioning Results.....	771
1.14.4.3 Commissioning Flink Applications on Flink Web UI and Linux.....	771
1.14.4.3.1 Compiling and Running Applications.....	771
1.14.4.3.2 Viewing Commissioning Results.....	778
1.14.4.4 Commissioning Kafka Applications on Linux.....	780
1.14.4.4.1 Compiling and Running Applications.....	780

1.14.4.4.2 Viewing Commissioning Results.....	782
1.14.4.5 Using a UDF.....	782
1.14.4.5.1 Registering a UDF.....	782
1.14.4.5.2 Querying a UDF.....	784
1.14.4.5.3 Deregistering a UDF.....	784
1.14.5 More Information.....	785
1.14.5.1 Common APIs.....	785
1.14.5.1.1 Java API.....	785
1.15 Kafka Development Guide.....	788
1.15.1 Overview.....	788
1.15.1.1 Development Environment Preparation.....	788
1.15.1.2 Common Concepts.....	788
1.15.1.3 Development Process.....	789
1.15.2 Environment Preparation.....	791
1.15.2.1 Preparing for Development and Operating Environment.....	792
1.15.2.2 Configuring and Importing Sample Projects.....	795
1.15.2.3 Preparing for Security Authentication.....	800
1.15.2.3.1 Sasl Kerberos authentication.....	800
1.15.2.3.2 SASL/PLAINTEXT Authentication.....	801
1.15.3 Developing an Application.....	802
1.15.3.1 Typical Scenario Description.....	802
1.15.3.2 Typical Scenario Sample Code Description.....	802
1.15.3.2.1 Producer API Usage Sample.....	802
1.15.3.2.2 Consumer API Usage Sample.....	803
1.15.3.2.3 Multi-thread Producer Sample.....	803
1.15.3.2.4 Multi-thread Consumer Sample.....	804
1.15.3.3 Kafka Streams Scenario Description.....	805
1.15.3.4 Kafka Streams Sample Code Description.....	805
1.15.3.4.1 High level KafkaStreams API Usage Sample.....	806
1.15.3.4.2 Low level KafkaStreams API Usage Sample.....	806
1.15.3.5 Kafka Token Authentication Mechanism Scenario.....	807
1.15.3.6 Sample Code for Kafka Token Authentication.....	807
1.15.3.7 Sample Code for Connecting Kafka to Spring Boot.....	809
1.15.4 Application Commissioning.....	812
1.15.4.1 Commissioning an Application in Windows.....	812
1.15.4.2 Commissioning an Application in Linux.....	813
1.15.4.3 Kafka Streams Sample Running Guide.....	816
1.15.4.3.1 High level Streams API Sample Usage Guide.....	816
1.15.4.3.2 Low level Streams API Sample Usage Guide.....	817
1.15.4.4 Sample Code Running Guide for the Kafka Token Authentication Mechanism.....	818
1.15.5 More Information.....	819
1.15.5.1 External Interfaces.....	819

1.15.5.1.1 Shell.....	819
1.15.5.1.2 Java API.....	820
1.15.5.1.3 Security Ports.....	823
1.15.5.1.4 SSL Encryption Function Used by a Client.....	823
1.15.5.2 FAQ.....	824
1.15.5.2.1 Topic Authentication Fails During Sample Running and "example-metric1=TOPIC_AUTHORIZATION_FAILED" Is Displayed.....	824
1.15.5.2.2 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic.....	824
1.16 MapReduce Development Guide.....	825
1.16.1 Overview.....	825
1.16.1.1 MapReduce Overview.....	825
1.16.1.2 Basic Concepts.....	825
1.16.1.3 Development Process.....	826
1.16.2 Environment Preparation.....	828
1.16.2.1 Preparing for Development and Operating Environment.....	828
1.16.2.2 Configuring and Importing Sample Projects.....	831
1.16.2.3 Creating a New Project (Optional).....	834
1.16.2.4 Preparing the Authentication Mechanism.....	837
1.16.3 Developing the Project.....	838
1.16.3.1 MapReduce Statistics Sample Project.....	838
1.16.3.1.1 Typical Scenarios.....	838
1.16.3.1.2 Example Code.....	839
1.16.3.2 MapReduce Accessing Multi-Component Example Project.....	842
1.16.3.2.1 Instance.....	843
1.16.3.2.2 Example Code.....	844
1.16.4 Commissioning the Application.....	849
1.16.4.1 Commissioning the Application in the Windows Environment.....	849
1.16.4.1.1 Compiling and Running the Application.....	850
1.16.4.1.2 Checking the Commissioning Result.....	851
1.16.4.2 Commissioning an Application in the Linux Environment.....	853
1.16.4.2.1 Compiling and Running the Application.....	853
1.16.4.2.2 Checking the Commissioning Result.....	854
1.16.5 More Information.....	856
1.16.5.1 Cross-Platform Compatibility of JAR Packages.....	856
1.16.5.2 Common APIs.....	858
1.16.5.2.1 Java API.....	858
1.16.5.2.2 REST API.....	861
1.16.5.3 FAQ.....	863
1.16.5.3.1 No Response from the Client When Submitting the MapReduce Application.....	863
1.16.5.3.2 When an Application Is Run, An Abnormality Occurs Due to Network Faults.....	864
1.16.5.3.3 How to Perform Remote Debugging During MapReduce Secondary Development?.....	864
1.17 Oozie Development Guide.....	866

1.17.1 Overview.....	866
1.17.1.1 Application Development Overview.....	866
1.17.1.2 Common Concepts.....	867
1.17.1.3 Development Process.....	867
1.17.2 Environment Preparation.....	869
1.17.2.1 Preparing Development and Operating Environment.....	869
1.17.2.2 Downloading and Importing Sample Projects.....	871
1.17.2.3 Preparing Authentication Mechanism Code.....	872
1.17.3 Developing the Project.....	873
1.17.3.1 Development of Configuration Files.....	873
1.17.3.1.1 Description.....	873
1.17.3.1.2 Development Procedure.....	874
1.17.3.2 Example Codes.....	877
1.17.3.2.1 job.properties.....	877
1.17.3.2.2 workflow.xml.....	878
1.17.3.2.3 Start Action.....	878
1.17.3.2.4 End Action.....	879
1.17.3.2.5 Kill Action.....	879
1.17.3.2.6 FS Action.....	880
1.17.3.2.7 MapReduce Action.....	881
1.17.3.2.8 coordinator.xml.....	882
1.17.3.3 Development of Java.....	883
1.17.3.3.1 Description.....	883
1.17.3.3.2 Sample Code.....	884
1.17.3.4 Scheduling Spark to Access HBase and Hive Using Oozie.....	885
1.17.4 Commissioning the Application.....	888
1.17.4.1 Commissioning an Application in the Windows Environment.....	888
1.17.4.1.1 Compiling and Running Applications.....	888
1.17.4.1.2 Checking the Commissioning Result.....	889
1.17.5 More Information.....	890
1.17.5.1 Common API Introduce.....	890
1.17.5.1.1 Shell.....	890
1.17.5.1.2 Java.....	891
1.17.5.1.3 REST.....	891
1.18 Redis Development Guide.....	891
1.18.1 Overview.....	892
1.18.1.1 Application Development Overview.....	892
1.18.1.2 Common Concepts.....	893
1.18.1.3 Development Process.....	893
1.18.2 Environment Preparation.....	895
1.18.2.1 Development and Operating Environment.....	895
1.18.2.2 Configuring and Importing Sample Projects.....	900

1.18.2.3 Preparing for Security Authentication.....	903
1.18.2.3.1 Preparing Authentication Mechanism Code.....	903
1.18.3 Developing an Application.....	903
1.18.3.1 Typical Scenario Description.....	904
1.18.3.2 Example Code Description.....	904
1.18.3.2.1 Redis Cluster Initialization.....	904
1.18.3.2.2 String Type Access.....	905
1.18.3.2.3 List Type Access.....	906
1.18.3.2.4 Hash Type Access.....	906
1.18.3.2.5 Set Type Access.....	907
1.18.3.2.6 Sorted Set Type Access.....	908
1.18.3.2.7 Usage of Basic GEO APIs.....	909
1.18.3.2.8 Use of Pipelines in the Single-thread Scenario.....	909
1.18.3.2.9 Use of Pipelines in the Multi-thread Scenario.....	910
1.18.3.2.10 Use of Pipelines in the bytes data Scenario.....	911
1.18.3.2.11 Connection to a Redis Logical ClusterUsing Spring Boot.....	912
1.18.3.3 Redis Client Authentication Mode.....	914
1.18.3.3.1 Usage Instruction.....	914
1.18.3.3.2 Authentication Though the auth.conf File.....	914
1.18.3.3.3 Global Authentication Through APIs.....	915
1.18.3.3.4 Authentication Though the jaas.conf File.....	916
1.18.3.3.5 Independent Authentication Through APIs.....	917
1.18.3.4 Connection to Redis in Security Mode Using Other Languages.....	918
1.18.3.4.1 Connection to Redis in Security Mode Using Python 3.x.....	918
1.18.3.4.2 Connection to Redis in Security Mode Using C and C++.....	921
1.18.3.4.3 Connection to Redis in Security Mode Using Node.js.....	925
1.18.3.4.4 Connection to Redis in Security Mode Using the Go Language.....	928
1.18.4 Application Commissioning.....	940
1.18.4.1 Commissioning an Application in Windows.....	940
1.18.4.1.1 Compiling and Running an Application.....	940
1.18.4.1.2 Viewing Commissioning Results.....	945
1.18.4.1.3 Commissioning a Spring Boot Program.....	947
1.18.4.2 Commissioning an Application in Linux.....	949
1.18.4.2.1 Compiling and Running an Application.....	949
1.18.4.2.2 Viewing Commissioning Results.....	952
1.18.4.2.3 Commissioning the Spring Boot Program.....	955
1.18.5 More Information.....	957
1.18.5.1 External Interfaces.....	957
1.18.5.1.1 Shell.....	957
1.18.5.1.2 Java API.....	958
1.18.5.2 Performance Optimization.....	958
1.18.5.2.1 Optimized Redis Write/Read Performance in Security Mode.....	958

1.19 RTD Development Guide.....	959
1.19.1 Overview.....	959
1.19.1.1 Application Development Overview.....	959
1.19.1.2 Common Concepts.....	959
1.19.1.3 Development Process.....	959
1.19.2 Environment Preparation.....	961
1.19.2.1 Preparing the Development and Operating Environment.....	961
1.19.2.2 Configuring and Importing Sample Projects.....	962
1.19.3 Developing an Application.....	965
1.19.3.1 Event Source Custom Plug-in Program.....	965
1.19.3.1.1 Scenario.....	965
1.19.3.1.2 Development Guidelines.....	966
1.19.3.1.3 Sample Code.....	966
1.19.3.2 Decision Engine Custom Program.....	973
1.19.3.2.1 Scenario.....	973
1.19.3.2.2 Development Guidelines.....	973
1.19.3.2.3 Sample Code.....	974
1.19.4 Commissioning the Application.....	977
1.19.4.1 Compiling and Packaging.....	977
1.20 Solr Development Guide.....	984
1.20.1 Overview.....	984
1.20.1.1 Application Development Overview.....	984
1.20.1.2 Common Concepts.....	985
1.20.1.3 Development Process.....	986
1.20.2 Environment Preparation.....	988
1.20.2.1 Preparing for Development and Operating Environment.....	988
1.20.2.2 Configuring and Importing Sample Projects.....	992
1.20.2.3 Preparing for Security Authentication.....	993
1.20.3 Developing an Application.....	994
1.20.3.1 Typical Scenario Description.....	994
1.20.3.2 Development Idea.....	994
1.20.3.3 Example Code Description.....	995
1.20.3.3.1 Initializing Solr.....	995
1.20.3.3.2 Querying a Collection.....	996
1.20.3.3.3 Deleting a Collection.....	997
1.20.3.3.4 Creating a Collection.....	997
1.20.3.3.5 Adding a Document.....	998
1.20.3.3.6 Querying a Document.....	999
1.20.3.3.7 Deleting a Document.....	1000
1.20.4 Application Commissioning.....	1000
1.20.4.1 Commissioning an Application in Windows.....	1000
1.20.4.1.1 Compiling and Running an Application.....	1000

1.20.4.1.2 Viewing Commissioning Results.....	1003
1.20.4.2 Commissioning an Application in Linux.....	1005
1.20.4.2.1 Compiling and Running an Application When a Client Is Installed.....	1005
1.20.4.2.2 Compiling and Running an Application When No Client Is Installed.....	1008
1.20.4.2.3 Viewing Commissioning Results.....	1009
1.20.5 More Information.....	1010
1.20.5.1 External Interfaces.....	1010
1.20.5.1.1 Shell.....	1010
1.20.5.1.2 Java API.....	1011
1.20.5.1.3 Web UI.....	1012
1.21 Spark Development Guide.....	1013
1.21.1 Overview.....	1013
1.21.1.1 Application Development Overview.....	1013
1.21.1.2 Basic Concepts.....	1015
1.21.1.3 Development Process.....	1021
1.21.2 Preparing for the Environment.....	1024
1.21.2.1 Preparing for Development and Operating Environment.....	1024
1.21.2.2 Configuring and Importing Sample Projects.....	1029
1.21.2.3 Creating a New Project (Optional).....	1046
1.21.2.4 Preparing for Security Authentication.....	1048
1.21.2.5 Configuring the Python3 Sample Project.....	1051
1.21.3 Developing the Project.....	1052
1.21.3.1 Spark Core Project.....	1052
1.21.3.1.1 Overview.....	1052
1.21.3.1.2 Java Sample Code.....	1054
1.21.3.1.3 Scala Sample Code.....	1056
1.21.3.1.4 Python Sample Code.....	1056
1.21.3.2 Spark SQL Project.....	1057
1.21.3.2.1 Overview.....	1057
1.21.3.2.2 Java Sample Code.....	1060
1.21.3.2.3 Scala Sample Code.....	1060
1.21.3.2.4 Python Sample Code.....	1061
1.21.3.3 Accessing the Spark SQL Through JDBC.....	1062
1.21.3.3.1 Overview.....	1062
1.21.3.3.2 Java Sample Code.....	1064
1.21.3.3.3 Scala Sample Code.....	1066
1.21.3.4 Spark on HBase.....	1068
1.21.3.4.1 Performing Operations on Data in Avro Format.....	1068
1.21.3.4.2 Performing Operations on the HBase Data Source.....	1072
1.21.3.4.3 Using the BulkPut Interface.....	1075
1.21.3.4.4 Using the BulkGet Interface.....	1078
1.21.3.4.5 Using the BulkDelete Interface.....	1082

1.21.3.4.6 Using the BulkLoad Interface.....	1085
1.21.3.4.7 Using the foreachPartition Interface.....	1088
1.21.3.4.8 Distributively Scanning HBase Tables.....	1091
1.21.3.4.9 Using the mapPartition Interface.....	1094
1.21.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming.....	1098
1.21.3.5 Reading Data from HBase and Write It Back to HBase.....	1101
1.21.3.5.1 Overview.....	1101
1.21.3.5.2 Java Example Code.....	1104
1.21.3.5.3 Scala Example Code.....	1106
1.21.3.5.4 Python Example Code.....	1108
1.21.3.6 Reading Data from Hive and Write It to HBase.....	1109
1.21.3.6.1 Overview.....	1109
1.21.3.6.2 Java Sample Code.....	1112
1.21.3.6.3 Scala Example Code.....	1114
1.21.3.6.4 Python Example Code.....	1115
1.21.3.7 Streaming Connecting to Kafka0-10.....	1116
1.21.3.7.1 Overview.....	1116
1.21.3.7.2 Java Example Code.....	1119
1.21.3.7.3 Scala Example Code.....	1122
1.21.3.8 Structured Streaming Project.....	1124
1.21.3.8.1 Overview.....	1124
1.21.3.8.2 Java Sample Code.....	1127
1.21.3.8.3 Scala Sample Code.....	1128
1.21.3.8.4 Python Sample Code.....	1129
1.21.3.9 Structured Streaming Stream-Stream Join.....	1130
1.21.3.9.1 Overview.....	1130
1.21.3.9.2 Scala Example Code.....	1134
1.21.3.10 Spark on Elasticsearch.....	1136
1.21.3.10.1 Overview.....	1136
1.21.3.10.2 Java Sample Code.....	1140
1.21.3.10.3 Scala Sample Code.....	1141
1.21.3.10.4 Python Sample Code.....	1142
1.21.3.11 Spark on Solr.....	1143
1.21.3.11.1 Overview.....	1143
1.21.3.11.2 Java Sample Code.....	1145
1.21.3.11.3 Scala Sample Code.....	1146
1.21.3.11.4 Python Sample Code.....	1147
1.21.3.12 Structured Streaming Status Operation.....	1148
1.21.3.12.1 Overview.....	1148
1.21.3.12.2 Scala Sample Code.....	1151
1.21.3.13 Concurrent Access from Spark to HBase in Two Clusters.....	1153
1.21.3.13.1 Overview.....	1153

1.21.3.13.2 Scala Sample Code.....	1153
1.21.3.14 Synchronizing HBase Data from Spark to CarbonData.....	1154
1.21.3.14.1 Overview.....	1154
1.21.3.14.2 Java Example Code.....	1156
1.21.3.15 Using Spark to Perform Basic Hudi Operations.....	1157
1.21.3.15.1 Overview.....	1157
1.21.3.15.2 Java Example Code.....	1158
1.21.3.15.3 Scala Example Code.....	1159
1.21.3.15.4 Python Example Code.....	1160
1.21.3.16 Compiling User-defined Configuration Items for Hudi.....	1162
1.21.3.16.1 HoodieDeltaStreamer.....	1162
1.21.3.16.2 User-defined Partitioner.....	1163
1.21.3.17 Using Spark to Write Data to ClickHouse.....	1164
1.21.3.17.1 Overview.....	1164
1.21.3.17.2 Java Sample Code.....	1166
1.21.3.17.3 Scala Sample Code.....	1167
1.21.3.17.4 Python Sample Code.....	1169
1.21.3.17.5 SQL Example.....	1171
1.21.4 Commissioning the Application.....	1171
1.21.4.1 Commissioning Applications on Windows.....	1171
1.21.4.1.1 Compiling and Running Applications.....	1172
1.21.4.1.2 View Debugging Results.....	1174
1.21.4.2 Commissioning an Application in Linux.....	1174
1.21.4.2.1 Compiling and Running the Application.....	1175
1.21.4.2.2 Checking the Commissioning Result.....	1178
1.21.5 More Information.....	1179
1.21.5.1 Common APIs.....	1179
1.21.5.1.1 Java.....	1179
1.21.5.1.2 Scala.....	1184
1.21.5.1.3 Python.....	1190
1.21.5.1.4 REST API.....	1194
1.21.5.2 Common CLIs.....	1202
1.21.5.3 JDBCServer Interface.....	1203
1.21.5.4 Structured Streaming Functions and Reliability.....	1205
1.21.5.5 FAQ.....	1210
1.21.5.5.1 How to Add a User-Defined Library.....	1210
1.21.5.5.2 How to Automatically Load Jars Packages?.....	1213
1.21.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStresmingKafka Project Is Running?.....	1213
1.21.5.5.4 Privilege Control Mechanism of SparkSQL UDF Feature.....	1214
1.21.5.5.5 Why Does Kafka Fail to Receive the Data Written Back by SLog in to the node where the client is installed as the client installation user.park Streaming?.....	1214

1.21.5.5.6 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?.....	1215
1.21.5.5.7 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?.....	1217
1.21.5.5.8 How to Perform Remote Debugging Using IDEA?.....	1217
1.21.5.5.9 How to Submit the Spark Application Using Java Commands?.....	1220
1.21.5.5.10 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used... 1222	1222
1.21.5.5.11 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming.....	1223
1.21.5.5.12 Restrictions on Restoring the Spark Application from the checkpoint.....	1223
1.21.5.5.13 Support for Third-party JAR Packages on x86 and TaiShan Platforms.....	1224
1.21.5.5.14 What Should I Do If a Large Number of Directories Whose Names Start with blockmgr- or spark- Exist in the /tmp Directory on the Client Installation Node?.....	1226
1.21.5.5.15 Error Code 139 Is Reported When Python Pipeline Runs in the Arm Environment.....	1227
1.21.5.5.16 What Should I Do If the Structured Streaming Task Submission Way Is Changed?.....	1227
1.21.5.5.17 Migrating Spark Streaming's Interconnection from Kafka 0.8 to Kafka 0.10.....	1228
1.22 YARN Development Guide.....	1236
1.22.1 Overview.....	1237
1.22.2 Interfaces.....	1238
1.22.2.1 Command.....	1238
1.22.2.2 Java API.....	1244
1.22.2.3 REST API.....	1247
1.22.2.4 REST APIs of Superior Scheduler.....	1251

## **2 Normal Mode.....** **1267**

2.1 ClickHouse Development Guide.....	1267
2.1.1 Overview.....	1267
2.1.1.1 Introduction to ClickHouse.....	1267
2.1.1.2 Basic Concepts.....	1268
2.1.1.3 Development Process.....	1268
2.1.2 Environment Preparations.....	1270
2.1.2.1 Preparing the Development and Operating Environment.....	1270
2.1.2.2 Configuring and Importing a Sample Project.....	1273
2.1.2.3 Configuring and Importing a Transaction Sample Project.....	1278
2.1.2.4 Configuring and Importing the Spring Boot Sample Project.....	1281
2.1.3 Application Development.....	1282
2.1.3.1 Typical Application Scenario.....	1282
2.1.3.2 Development Guideline.....	1283
2.1.4 Sample Code.....	1283
2.1.4.1 Setting Properties.....	1283
2.1.4.2 Establishing a Connection.....	1283
2.1.4.3 Creating a Database.....	1284
2.1.4.4 Creating a Table.....	1284
2.1.4.5 Inserting Data.....	1284

2.1.4.6 Querying Data.....	1285
2.1.4.7 Deleting a Table.....	1285
2.1.4.8 Using BalanceDataSource.....	1285
2.1.5 Application Commissioning.....	1288
2.1.5.1 Commissioning Applications on Windows.....	1288
2.1.5.2 Commissioning Applications on Linux.....	1293
2.1.5.3 Commissioning the Spring Boot Sample Project.....	1297
2.1.5.3.1 Commissioning the Spring Boot Project in Windows.....	1297
2.1.5.3.2 Commissioning the Spring Boot Project in Linux.....	1301
2.2 Doris Development Guide.....	1303
2.2.1 Overview.....	1303
2.2.1.1 Doris Introduction.....	1303
2.2.1.2 Common Concepts.....	1303
2.2.1.3 Development Process.....	1305
2.2.1.4 Doris Sample Project Introduction.....	1308
2.2.2 Environment Preparation.....	1309
2.2.2.1 Preparing for Development Environment.....	1309
2.2.2.2 Preparing the Configuration Files for Connecting to the Cluster.....	1311
2.2.2.3 Configuring and Importing JDBC or Stream Load Sample Projects.....	1313
2.2.2.4 Configuring and Importing SpringBoot Sample Projects.....	1317
2.2.3 Doris Application Development.....	1318
2.2.3.1 Using Doris Through JDBC and DBalancer.....	1318
2.2.3.1.1 Service Scenario Description.....	1319
2.2.3.1.2 Application Development Approach.....	1319
2.2.3.1.3 Setting Up a Connection.....	1319
2.2.3.1.4 Creating a Database.....	1320
2.2.3.1.5 Creating a Table.....	1321
2.2.3.1.6 Inserting Data.....	1321
2.2.3.1.7 Querying Data.....	1321
2.2.3.1.8 Deleting a table.....	1322
2.2.3.1.9 Deleting a Database.....	1322
2.2.3.2 Loading Data to a Doris Table with Stream Load.....	1322
2.2.4 Application Commissioning.....	1324
2.2.4.1 Commissioning an Application in Windows.....	1325
2.2.4.1.1 Compiling and Running an Application.....	1325
2.2.4.1.2 Viewing Windows Commissioning Results.....	1332
2.2.4.2 Commissioning an Application in Linux.....	1335
2.2.4.2.1 Compiling and Running Applications.....	1335
2.2.4.2.2 Viewing Linux Commissioning Results.....	1338
2.3 Elasticsearch Development Guide .....	1341
2.3.1 Overview.....	1341
2.3.1.1 Application Development Overview.....	1341

2.3.1.2 Concepts.....	1341
2.3.1.3 Application Development Process.....	1342
2.3.2 Environment Preparation.....	1344
2.3.2.1 Preparing the Development Environment and Operating Environment.....	1344
2.3.2.2 Configuring and Importing Sample Projects.....	1347
2.3.2.2.1 Overview.....	1347
2.3.2.2.2 Importing a RestClient Sample Project.....	1348
2.3.2.2.3 Importing a SpringBoot Sample Project.....	1350
2.3.3 Development Process.....	1352
2.3.3.1 Typical Application Scenarios.....	1352
2.3.3.2 Low Level RestClient Sample Java Code.....	1353
2.3.3.2.1 Connecting the Elasticsearch Client to an Elasticsearch Cluster.....	1353
2.3.3.2.2 Querying the Health Status of an Elasticsearch Cluster.....	1353
2.3.3.2.3 Checking Whether the Specified Index Exists.....	1354
2.3.3.2.4 Creating an Index with the Desired Number of Shards.....	1354
2.3.3.2.5 Writing Index Data.....	1355
2.3.3.2.6 Writing Data in Batches.....	1356
2.3.3.2.7 Batch write data to specified routes.....	1357
2.3.3.2.8 Querying Index Information.....	1358
2.3.3.2.9 Deleting an Index.....	1358
2.3.3.2.10 Deleting Some Documents in the Index.....	1359
2.3.3.2.11 Deleting All Documents in the Index.....	1359
2.3.3.2.12 Refreshing an Index.....	1360
2.3.3.2.13 Multi-Thread Example.....	1361
2.3.3.2.14 Pre-sorting Indices.....	1362
2.3.3.3 High Level RestClient Sample Code.....	1363
2.3.3.3.1 Connecting the Client to a Cluster.....	1364
2.3.3.3.2 Querying Cluster Information.....	1364
2.3.3.3.3 Writing Index Data.....	1365
2.3.3.3.4 Operations in Batches.....	1366
2.3.3.3.5 Batch write data to specified routes.....	1367
2.3.3.3.6 Updating Index Information.....	1368
2.3.3.3.7 Querying Index Information.....	1368
2.3.3.3.8 Searching for Document Information.....	1368
2.3.3.3.9 Searching for Document Information by Using a Cursor.....	1369
2.3.3.3.10 Clearing a Cursor.....	1370
2.3.3.3.11 Deleting an Index.....	1370
2.3.3.3.12 Multi-Thread Request.....	1371
2.3.3.3.13 BulkProcessor Batch Import Example.....	1372
2.3.3.3.14 Pre-sorting Indices.....	1373
2.3.4 Commissioning Process.....	1374
2.3.4.1 Commissioning Applications on Windows.....	1374

2.3.4.1.1 Commissioning a RestClient Application.....	1374
2.3.4.1.2 Commissioning a SpringBoot Program.....	1383
2.3.4.2 Commissioning Applications in Linux.....	1386
2.3.4.2.1 Commissioning a RestClient Application.....	1386
2.3.4.2.2 Commissioning a SpringBoot Program.....	1391
2.4 Flink Development Guide.....	1393
2.4.1 Overview.....	1393
2.4.1.1 Application Development.....	1394
2.4.1.2 Basic Concepts.....	1396
2.4.1.3 Development Process.....	1397
2.4.2 Environment Preparation.....	1398
2.4.2.1 Preparing the Development and Operating Environment.....	1398
2.4.2.2 Configuring and Importing Sample Projects.....	1401
2.4.2.3 Creating a Project (Optional).....	1423
2.4.2.4 Configuring a Spring Boot Sample Project.....	1426
2.4.3 Developing an Application.....	1431
2.4.3.1 DataStream Application.....	1431
2.4.3.1.1 Scenarios.....	1431
2.4.3.1.2 Java Sample Code.....	1432
2.4.3.1.3 Scala Sample Code.....	1434
2.4.3.2 Interconnecting with Kafka.....	1436
2.4.3.2.1 Scenarios.....	1436
2.4.3.2.2 Java Sample Code.....	1437
2.4.3.2.3 Scala Sample Code.....	1439
2.4.3.3 Asynchronous Checkpoint Mechanism.....	1440
2.4.3.3.1 Scenarios.....	1440
2.4.3.3.2 Java Sample Code.....	1441
2.4.3.3.3 Scala Sample Code.....	1443
2.4.3.4 Job Pipeline Program.....	1446
2.4.3.4.1 Scenario.....	1446
2.4.3.4.2 Java Sample Code.....	1448
2.4.3.4.3 Scala Sample Code.....	1450
2.4.3.5 JOIN Operation between Configuration Tables and Streams.....	1452
2.4.3.5.1 Scenario Description.....	1452
2.4.3.5.2 Java Sample Code.....	1455
2.4.3.5.3 Scala Sample Code.....	1464
2.4.3.6 Stream SQL Join Program.....	1472
2.4.3.6.1 Scenario.....	1472
2.4.3.6.2 Java Sample Code.....	1473
2.4.3.6.3 Scala Sample Code.....	1475
2.4.3.7 Submitting a SQL Job Using Flink Jar.....	1477
2.4.3.7.1 Scenario Description.....	1478

2.4.3.7.2 Java Sample Code.....	1478
2.4.3.8 FlinkServer REST API JavaExample.....	1479
2.4.3.8.1 Scenario Description.....	1479
2.4.3.8.2 Java sample code for invoking the FlinkServer REST API to create a tenant.....	1479
2.4.3.8.3 Calling a FlinkServer REST API to Start a Job.....	1480
2.4.3.8.4 Accessing Flinkserver RESTful API as a Proxy User.....	1480
2.4.3.9 Flink Reading Data from and Writing Data to HBase.....	1481
2.4.3.9.1 Scenario Description.....	1481
2.4.3.9.2 Java Sample Code.....	1482
2.4.3.10 Flink Reading Data from and Writing Data to Hudi.....	1487
2.4.3.10.1 Scenario Description.....	1487
2.4.3.10.2 Java Sample Code.....	1487
2.4.3.11 Python Development Examples.....	1490
2.4.3.11.1 Submitting a Regular Job Using Python.....	1490
2.4.3.11.2 Submitting a SQL Job Using Python.....	1493
2.4.4 Debugging the Application.....	1497
2.4.4.1 Compiling and Running Applications.....	1497
2.4.4.2 Viewing the Debugging Result.....	1505
2.4.4.3 Running a Spring Boot Sample Project and Viewing Results.....	1511
2.4.5 More Information.....	1512
2.4.5.1 Introduction to Common APIs.....	1512
2.4.5.1.1 Java.....	1512
2.4.5.1.2 Scala.....	1529
2.4.5.2 Overview of RESTful APIs.....	1543
2.4.5.3 Overview of Savepoints CLI.....	1546
2.4.5.4 Introduction to Flink Client CLI.....	1548
2.4.5.5 FAQ.....	1549
2.4.5.5.1 Savepoints-related Problems.....	1549
2.4.5.5.2 What If the Chrome Browser Cannot Display the Title.....	1550
2.4.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11.....	1551
2.4.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large.....	1552
2.4.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home.....	1554
2.4.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?.....	1555
2.4.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?.....	1556
2.4.5.5.8 Why Cannot I Access the Apache Flink Dashboard?.....	1556
2.4.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?.....	1557
2.4.5.5.10 Incorrect GLIBC Version.....	1558
2.5 GraphBase Development Guide.....	1559
2.5.1 Overview.....	1559
2.5.1.1 Application Development Overview.....	1559

2.5.1.2 Basic Concepts.....	1560
2.5.1.3 Development Process.....	1560
2.5.2 Environment Preparation.....	1563
2.5.2.1 REST API Development Environment Preparation.....	1563
2.5.2.1.1 Preparing the Development and Operating Environment.....	1563
2.5.2.1.2 Configuring and Importing Sample Projects.....	1567
2.5.2.1.3 Preparing for Security Authentication.....	1570
2.5.2.2 Gremlin API Development Environment Preparation.....	1571
2.5.2.2.1 Preparing the Development and Operating Environment.....	1572
2.5.2.2.2 Configuring and Importing a Piece of Sample Code.....	1575
2.5.3 Developing an Application.....	1575
2.5.3.1 Typical Application Scenario.....	1575
2.5.3.2 Preparing Data Files.....	1578
2.5.3.2.1 Compiling a Schema File (XML).....	1578
2.5.3.2.2 Compiling a Data File (CSV).....	1581
2.5.3.2.3 Compiling a Data Description File (DESC).....	1582
2.5.3.2.4 Compiling a Graph Mapping Rule File (.mapper).....	1583
2.5.3.3 Importing Data.....	1596
2.5.3.3.1 Importing Data in Batches Using Tools.....	1597
2.5.3.3.2 Importing Data in Real Time Using Tool.....	1598
2.5.3.3.3 Using the Hive Data Import Tool.....	1603
2.5.3.4 Data Export.....	1605
2.5.3.5 REST API Development.....	1620
2.5.3.6 REST API Invocation Examples and Running Results.....	1620
2.5.3.6.1 Example.....	1620
2.5.3.6.2 Creating or Deleting a Graph.....	1622
2.5.3.6.3 Creating a Schema by Uploading an XML File.....	1623
2.5.3.6.4 Creating a Schema.....	1623
2.5.3.6.5 Creating an Index.....	1627
2.5.3.6.6 Creating and Querying a Vertex or an Edge.....	1630
2.5.3.6.7 Performing a Full Graph Query.....	1636
2.5.3.6.8 Performing a Path Query.....	1638
2.5.3.6.9 Performing a Line Expansion Query.....	1646
2.5.3.6.10 Analyzing the Typical Application Scenario.....	1648
2.5.3.7 Gremlin Console.....	1653
2.5.3.8 Gremlin Java.....	1654
2.5.3.8.1 Connecting Gremlin Servers and Clients.....	1654
2.5.3.8.2 Running Gremlin Java API Example Code.....	1655
2.5.3.9 Gremlin Application Scenarios.....	1655
2.5.3.9.1 Querying a Specified Graph Object.....	1655
2.5.3.9.2 Queries One or More Properties.....	1655
2.5.3.9.3 Traversing a Graph Object.....	1655

2.5.3.9.4 Traversing a Path.....	1657
2.5.3.9.5 Filtering by Condition.....	1657
2.5.3.9.6 Sorting the Result.....	1659
2.5.3.9.7 Displaying the Result in Multiple Pages.....	1659
2.5.3.9.8 Calculating the Intersection Between Two Data Sets.....	1659
2.5.3.9.9 Calculating the Union of Two Data Sets.....	1660
2.5.3.9.10 Using Statistical Functions.....	1660
2.5.4 Application Commissioning.....	1660
2.5.4.1 Commissioning an Application in Windows.....	1660
2.5.4.1.1 Compiling and Running an Application.....	1660
2.5.4.1.2 Viewing Windows Commissioning Results.....	1661
2.5.4.2 Commissioning an Application in Linux.....	1662
2.5.4.2.1 Compiling and Running an Application.....	1662
2.5.4.2.2 Viewing Linux Commissioning Results.....	1664
2.5.5 More Information.....	1665
2.5.5.1 Common APIs.....	1665
2.5.5.1.1 REST API.....	1665
2.5.5.1.2 Gremlin API.....	1665
2.6 HBase Development Guide.....	1666
2.6.1 Overview.....	1666
2.6.1.1 Application Development Overview.....	1666
2.6.1.2 Common Concepts.....	1667
2.6.1.3 Development Process.....	1667
2.6.1.4 HBase Sample Project Introduction.....	1669
2.6.2 Environment Preparation.....	1671
2.6.2.1 Preparing for Development and Operating Environment.....	1671
2.6.2.2 Preparing the Connection Cluster Configuration File.....	1673
2.6.2.3 Configuring and Importing Sample Projects.....	1676
2.6.3 Developing an Application.....	1687
2.6.3.1 HBase Data Read/Write Sample Program.....	1687
2.6.3.1.1 Service Scenario Description.....	1687
2.6.3.1.2 Application Development Approach.....	1688
2.6.3.1.3 Configuring Log4j Log Output.....	1689
2.6.3.1.4 Creating Configuration.....	1690
2.6.3.1.5 Creating Connection.....	1690
2.6.3.1.6 Creating a Table.....	1691
2.6.3.1.7 Deleting a Table.....	1693
2.6.3.1.8 Modifying a Table.....	1693
2.6.3.1.9 Inserting Data.....	1694
2.6.3.1.10 Deleting Data.....	1696
2.6.3.1.11 Reading Data Using Get.....	1697
2.6.3.1.12 Reading Data Using Scan.....	1697

2.6.3.1.13 Filtering Data.....	1699
2.6.3.1.14 Creating a Secondary Index.....	1700
2.6.3.1.15 Deleting an Index.....	1702
2.6.3.1.16 Secondary Index-based Query.....	1703
2.6.3.1.17 Multi-Point Region Division.....	1706
2.6.3.1.18 Creating a Phoenix Table.....	1707
2.6.3.1.19 Writing Data to the PhoenixTable.....	1708
2.6.3.1.20 Reading the PhoenixTable.....	1708
2.6.3.1.21 Using HBase Dual-Read Capacity.....	1709
2.6.3.2 HBase Global Secondary Index Sample Program.....	1713
2.6.3.2.1 Service Scenario Description.....	1713
2.6.3.2.2 Creating HBase Global Secondary Indexes.....	1713
2.6.3.2.3 Querying Global Secondary Indexes.....	1715
2.6.3.2.4 Querying Based on Global Secondary Indexes.....	1715
2.6.3.2.5 Disabling Global Secondary Indexes.....	1716
2.6.3.2.6 Deleting Global Secondary Indexes.....	1717
2.6.3.3 Sample Program for Preloading Meta Tables When Request Concurrency Is High.....	1717
2.6.3.3.1 Service Scenario Description.....	1717
2.6.3.3.2 Preloading Meta Tables When Request Concurrency Is High.....	1718
2.6.3.4 HBase Rest API Invoking Sample Program.....	1720
2.6.3.4.1 Querying Cluster Information Using REST.....	1720
2.6.3.4.2 Obtaining All Tables Using REST.....	1721
2.6.3.4.3 Operate Namespaces Using REST.....	1721
2.6.3.4.4 Operate Tables Using REST.....	1722
2.6.3.5 Accessing the HBase ThriftServer Sample Program.....	1724
2.6.3.5.1 Accessing the ThriftServer Operation Table.....	1724
2.6.3.5.2 Accessing ThriftServer to Write Data.....	1726
2.6.3.5.3 Accessing ThriftServer to Read Data.....	1727
2.6.3.6 Sample Program for HBase to Access Multiple ZooKeepers.....	1729
2.6.3.6.1 Accessing Multiple ZooKeepers.....	1729
2.6.3.7 Example Configuration for Interconnecting HBase/Phoenix with SpringBoot.....	1730
2.6.4 Application Commissioning.....	1730
2.6.4.1 Commissioning an Application in Windows.....	1730
2.6.4.1.1 Compiling and Running an Application.....	1731
2.6.4.1.2 Viewing Windows Commissioning Results.....	1735
2.6.4.2 Commissioning an Application in Linux.....	1737
2.6.4.2.1 Compiling and Running an Application When a Client Is Installed.....	1737
2.6.4.2.2 Compiling and Running an Application When No Client Is Installed.....	1740
2.6.4.2.3 Viewing Linux Commissioning Results.....	1741
2.6.5 More Information.....	1742
2.6.5.1 SQL Query.....	1742
2.6.5.2 HBase Dual-Read Configuration Items.....	1744

2.6.5.3 External Interfaces.....	1747
2.6.5.3.1 Shell.....	1748
2.6.5.3.2 Java APIs.....	1749
2.6.5.3.3 SQLLine.....	1753
2.6.5.3.4 JDBC APIs.....	1759
2.6.5.3.5 WebUI.....	1760
2.6.5.4 Phoenix Command Line.....	1763
2.6.5.5 FAQs.....	1764
2.6.5.5.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?.....	1764
2.6.5.5.2 What Are the Application Scenarios of the bulkload and put Data-loading Modes?.....	1765
2.6.5.5.3 An Error Occurred When Building a JAR Package.....	1765
2.6.5.5.4 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "java.lang.OutOfMemoryError: Direct buffer memory" Is Displayed in the Error Information?.....	1766
2.7 HDFS Development Guide.....	1766
2.7.1 Overview.....	1766
2.7.1.1 Introduction to HDFS.....	1767
2.7.1.2 Basic Concepts.....	1767
2.7.1.3 Development Process.....	1768
2.7.2 Environment Preparation.....	1770
2.7.2.1 Development and Operating Environment.....	1770
2.7.2.2 Configuring and Importing Sample Projects.....	1773
2.7.3 Developing the Project.....	1780
2.7.3.1 Scenario.....	1780
2.7.3.2 Development Idea.....	1781
2.7.3.3 Declare the Example Codes.....	1781
2.7.3.3.1 Initializing the HDFS.....	1781
2.7.3.3.2 Creating Directories.....	1782
2.7.3.3.3 Writing Data into a File.....	1783
2.7.3.3.4 Appending Data to a File.....	1784
2.7.3.3.5 Reading Data from a File.....	1784
2.7.3.3.6 Deleting a File.....	1785
2.7.3.3.7 Deleting Directories.....	1786
2.7.3.3.8 Multi-Thread Tasks.....	1786
2.7.3.3.9 Setting Storage Policies.....	1787
2.7.3.3.10 Colocation.....	1788
2.7.4 Commissioning the Application.....	1792
2.7.4.1 Commissioning an Application in the Windows Environment.....	1792
2.7.4.1.1 Compiling and Running an Application.....	1792
2.7.4.1.2 Checking the Commissioning Result.....	1793
2.7.4.2 Commissioning an Application in the Linux Environment.....	1794

2.7.4.2.1 Compiling and Running an Application with the Client Installed.....	1795
2.7.4.2.2 Compiling and Running an Application with the Client Not Installed.....	1796
2.7.4.2.3 Checking the Commissioning Result.....	1797
2.7.5 More Information.....	1798
2.7.5.1 Common API Introduction.....	1798
2.7.5.1.1 Java API.....	1798
2.7.5.1.2 C API.....	1802
2.7.5.1.3 HTTP REST API.....	1806
2.7.5.2 Shell Command Introduce.....	1811
2.8 HetuEngine Development Guide.....	1813
2.8.1 Overview.....	1813
2.8.1.1 Introduction to HetuEngine.....	1813
2.8.1.2 Concepts.....	1814
2.8.1.3 Connection Modes.....	1815
2.8.1.4 Development Process.....	1815
2.8.2 Preparing Environment.....	1817
2.8.2.1 Preparing Development and Operating Environment.....	1817
2.8.2.2 Configuring and Importing Sample Projects.....	1821
2.8.2.3 Setting Up a Python3 Project.....	1822
2.8.2.4 Setting Up a Spring Boot Sample Project.....	1824
2.8.3 Application Development.....	1824
2.8.3.1 Typical Application Scenario.....	1824
2.8.3.2 Java Sample Code.....	1825
2.8.3.2.1 Accessing Hive Data Sources Using HSFabric.....	1825
2.8.3.2.2 Accessing Hive Data Sources Using HSBroker.....	1826
2.8.3.2.3 Querying the Execution Progress and Status of an SQL Statement Using JDBC.....	1828
2.8.3.3 Python3 Sample Code.....	1830
2.8.3.3.1 Accessing Hive Data Sources Using HSBroker.....	1831
2.8.3.3.2 Accessing Hive Data Sources Using HSFabric.....	1832
2.8.4 Application Commissioning.....	1833
2.8.4.1 Commissioning Applications on Windows.....	1833
2.8.4.2 Commissioning Applications on Linux.....	1835
2.8.4.3 Commissioning the Python3 Sample Project.....	1836
2.8.4.4 Commissioning a Spring Boot Application.....	1837
2.9 Hive Development Guide.....	1839
2.9.1 Overview.....	1839
2.9.1.1 Application Development Overview.....	1839
2.9.1.2 Common Concepts.....	1840
2.9.1.3 Development Process.....	1840
2.9.2 Preparing the Environment.....	1842
2.9.2.1 Preparing Development and Operating Environment.....	1842
2.9.2.2 Configuring the JDBC Sample Project.....	1846

2.9.2.3 Configuring the Hcatalog Sample Project.....	1850
2.9.2.4 Configuring the Python Sample Project.....	1854
2.9.2.5 Configuring the Python3 Sample Project.....	1855
2.9.2.6 Configuring a Spring Boot Sample Project.....	1856
2.9.3 Developing an Application.....	1860
2.9.3.1 Typical Scenario Description.....	1860
2.9.3.2 Example Codes.....	1863
2.9.3.2.1 Creating a Table.....	1863
2.9.3.2.2 Loading Data.....	1864
2.9.3.2.3 Querying Data.....	1865
2.9.3.2.4 UDF.....	1866
2.9.3.2.5 Example Program Guide.....	1868
2.9.3.2.6 Accessing Multiple ZooKeepers.....	1872
2.9.4 Commissioning Applications.....	1873
2.9.4.1 Running JDBC and Viewing Results.....	1873
2.9.4.2 Running HCatalog and Viewing Results.....	1877
2.9.4.3 Running Python and Viewing Results.....	1880
2.9.4.4 Running Python3 and Viewing Results.....	1881
2.9.4.5 Running a Spring Boot Sample Project and Viewing Results.....	1882
2.9.5 More Information.....	1883
2.9.5.1 Interface Reference.....	1883
2.9.5.1.1 JDBC.....	1884
2.9.5.1.2 Hive SQL.....	1884
2.9.5.1.3 WebHCat.....	1887
2.9.5.2 FAQ.....	1911
2.9.5.2.1 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions.....	1911
2.9.5.2.2 "version 'GLIBCXX_3.4.21' not found" Exception Occurs When the Python 3 Secondary Development Program Is Used to Access Hive.....	1911
2.10 IoTDB Development Guide.....	1912
2.10.1 Overview.....	1912
2.10.1.1 Application Development Overview.....	1912
2.10.1.2 Basic Concepts.....	1912
2.10.1.3 Development Process.....	1913
2.10.1.4 IoTDB Sample Project.....	1915
2.10.2 Environment Preparations.....	1916
2.10.2.1 Prepare the Development Environment.....	1916
2.10.2.2 Preparing the Configuration Files for Connecting to the Cluster.....	1918
2.10.2.3 Configuring and Importing Sample Projects.....	1919
2.10.3 Application Development.....	1926
2.10.3.1 IoTDB JDBC.....	1926
2.10.3.1.1 Java Example Code.....	1926
2.10.3.2 IoTDB Session.....	1927
2.10.3.2.1 Java Example Code.....	1927

2.10.3.3 IoTDB Flink.....	1929
2.10.3.3.1 FlinkIoTDBSink.....	1929
2.10.3.3.2 FlinkIoTDBSource.....	1930
2.10.3.4 IoTDB Kafka.....	1932
2.10.3.4.1 Java Sample Code.....	1932
2.10.3.5 IoTDB UDF Program.....	1934
2.10.3.5.1 IoTDB UDF Sample Code.....	1934
2.10.4 Application Commissioning.....	1935
2.10.4.1 Commissioning Applications on Windows.....	1935
2.10.4.1.1 Compiling and Running Applications.....	1935
2.10.4.1.2 Viewing Commissioning Results.....	1939
2.10.4.2 Commissioning JDBC and Session Applications on Linux.....	1940
2.10.4.2.1 Compiling and Running Applications.....	1940
2.10.4.2.2 Viewing Commissioning Results.....	1942
2.10.4.3 Commissioning Flink Applications on Flink Web UI and Linux.....	1942
2.10.4.3.1 Compiling and Running Applications.....	1942
2.10.4.3.2 Viewing Commissioning Results.....	1947
2.10.4.4 Commissioning Kafka Applications on Linux.....	1949
2.10.4.4.1 Compiling and Running Applications.....	1949
2.10.4.4.2 Viewing Commissioning Results.....	1951
2.10.4.5 Using a UDF.....	1951
2.10.4.5.1 Registering a UDF.....	1951
2.10.4.5.2 Querying a UDF.....	1953
2.10.4.5.3 Deregistering a UDF.....	1953
2.10.5 More Information.....	1954
2.10.5.1 Common APIs.....	1954
2.10.5.1.1 Java API.....	1954
2.11 Kafka Development Guide.....	1957
2.11.1 Overview.....	1957
2.11.1.1 Development Environment Preparation.....	1957
2.11.1.2 Common Concepts.....	1957
2.11.1.3 Development Process.....	1958
2.11.2 Environment Preparation.....	1959
2.11.2.1 Preparing for Development and Operating Environment.....	1960
2.11.2.2 Configuring and Importing Sample Projects.....	1963
2.11.3 Developing an Application.....	1968
2.11.3.1 Typical Scenario Description.....	1968
2.11.3.2 Example Code Description.....	1969
2.11.3.2.1 Producer API Usage Sample.....	1969
2.11.3.2.2 Consumer API Usage Sample.....	1969
2.11.3.2.3 Multi-thread Producer Sample.....	1970
2.11.3.2.4 Multi-thread Consumer Sample.....	1971

2.11.3.3 Kafka Streams Sample Code Description.....	1972
2.11.3.4 Kafka Streams Sample Code Description.....	1972
2.11.3.4.1 High level KafkaStreams API Usage Sample.....	1972
2.11.3.4.2 Low level KafkaStreams API Usage Sample.....	1973
2.11.3.5 Sample Code for Connecting Kafka to Spring Boot.....	1974
2.11.4 Application Commissioning.....	1976
2.11.4.1 Commissioning an Application in Windows.....	1976
2.11.4.2 Commissioning an Application in Linux.....	1978
2.11.4.3 Kafka Streams Sample Running Guide.....	1981
2.11.4.3.1 High Level Kafka Streams API Sample Usage Guide.....	1981
2.11.4.3.2 Low level Kafka Streams API Sample Usage Guide.....	1982
2.11.5 More Information.....	1982
2.11.5.1 External Interfaces.....	1983
2.11.5.1.1 Shell.....	1983
2.11.5.1.2 Java API.....	1984
2.11.5.2 FAQ.....	1986
2.11.5.2.1 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic.....	1986
2.12 MapReduce Development Guide.....	1986
2.12.1 Overview.....	1986
2.12.1.1 MapReduce Overview.....	1987
2.12.1.2 Basic Concepts.....	1987
2.12.1.3 Development Process.....	1988
2.12.2 Environment Preparation.....	1990
2.12.2.1 Preparing Development and Operating Environment.....	1990
2.12.2.2 Configuring and Importing Sample Projects.....	1993
2.12.2.3 Creating a New Project (Optional).....	1996
2.12.3 Developing the Project.....	1999
2.12.3.1 MapReduce Statistics Sample Project.....	1999
2.12.3.1.1 Typical Scenarios.....	1999
2.12.3.1.2 Example Codes.....	2000
2.12.3.2 MapReduce Accessing Multi-Component Example Project.....	2003
2.12.3.2.1 Instance.....	2003
2.12.3.2.2 Example Code.....	2004
2.12.4 Commissioning the Application.....	2008
2.12.4.1 Commissioning the Application in the Windows Environment.....	2009
2.12.4.1.1 Compiling and Running the Application.....	2009
2.12.4.1.2 Checking the Commissioning Result.....	2009
2.12.4.2 Commissioning the Application in the Linux Environment.....	2012
2.12.4.2.1 Compiling and Running the Application.....	2012
2.12.4.2.2 Checking the Commissioning Result.....	2013
2.12.5 More Information.....	2016
2.12.5.1 Cross-Platform Compatibility of JAR Packages.....	2016

2.12.5.2 Common APIs.....	2017
2.12.5.2.1 Java API.....	2018
2.12.5.2.2 REST API.....	2020
2.12.5.3 FAQ.....	2022
2.12.5.3.1 No Response from the Client When Submitting the MapReduce Application.....	2022
2.12.5.3.2 How to Perform Remote Debugging During MapReduce Secondary Development?.....	2022
2.13 Oozie Development Guide.....	2025
2.13.1 Overview.....	2025
2.13.1.1 Application Development Overview.....	2025
2.13.1.2 Common Concepts.....	2026
2.13.1.3 Development Process.....	2026
2.13.2 Environment Preparation.....	2028
2.13.2.1 Preparing for Development and Operating Environment.....	2028
2.13.2.2 Downloading and Importing Sample Projects.....	2030
2.13.3 Developing the Project.....	2031
2.13.3.1 Development of Configuration Files.....	2031
2.13.3.1.1 Description.....	2031
2.13.3.1.2 Development Procedure.....	2032
2.13.3.2 Example Codes.....	2035
2.13.3.2.1 job.properties.....	2035
2.13.3.2.2 workflow.xml.....	2036
2.13.3.2.3 Start Action.....	2036
2.13.3.2.4 End Action.....	2037
2.13.3.2.5 Kill Action.....	2037
2.13.3.2.6 FS Action.....	2038
2.13.3.2.7 MapReduce Action.....	2039
2.13.3.2.8 coordinator.xml.....	2040
2.13.3.3 Development of Java.....	2041
2.13.3.3.1 Description.....	2041
2.13.3.3.2 Sample Code.....	2041
2.13.3.4 Scheduling Spark to Access HBase and Hive Using Oozie.....	2042
2.13.4 Commissioning the Application.....	2045
2.13.4.1 Commissioning an Application in the Windows Environment.....	2045
2.13.4.1.1 Compiling and Running Applications.....	2046
2.13.4.1.2 Checking the Commissioning Result.....	2046
2.13.5 More Information.....	2047
2.13.5.1 Common API Introduce.....	2047
2.13.5.1.1 Shell.....	2047
2.13.5.1.2 Java.....	2048
2.13.5.1.3 REST.....	2048
2.14 Redis Development Guide.....	2048
2.14.1 Overview.....	2049

2.14.1.1 Application Development Overview.....	2049
2.14.1.2 Common Concepts.....	2050
2.14.1.3 Development Process.....	2050
2.14.2 Environment Preparation.....	2051
2.14.2.1 Development and Operating Environment.....	2051
2.14.2.2 Configuring and Importing Sample Projects.....	2055
2.14.3 Developing an Application.....	2058
2.14.3.1 Typical Scenario Description.....	2058
2.14.3.2 Example Code Description.....	2059
2.14.3.2.1 Redis Cluster Initialization.....	2059
2.14.3.2.2 String Type Access.....	2059
2.14.3.2.3 List Type Access.....	2060
2.14.3.2.4 Hash Type Access.....	2061
2.14.3.2.5 Set Type Access.....	2062
2.14.3.2.6 Sorted Set Type Access.....	2062
2.14.3.2.7 Usage of Basic GEO APIs.....	2063
2.14.3.2.8 Use of Pipelines in the Single-thread Scenario.....	2064
2.14.3.2.9 Use of Pipelines in the Multi-thread Scenario.....	2065
2.14.3.2.10 Use of Pipelines in the bytes data Scenario.....	2066
2.14.3.2.11 Connection to a Redis Logical ClusterUsing Spring Boot.....	2066
2.14.4 Application Commissioning.....	2068
2.14.4.1 Commissioning an Application in Windows.....	2068
2.14.4.1.1 Compiling and Running an Application.....	2068
2.14.4.1.2 Viewing Commissioning Results.....	2073
2.14.4.1.3 Commissioning a Spring Boot Program.....	2075
2.14.4.2 Commissioning an Application in Linux.....	2077
2.14.4.2.1 Compiling and Running an Application.....	2077
2.14.4.2.2 Viewing Commissioning Results.....	2081
2.14.4.2.3 Commissioning the Spring Boot Program.....	2083
2.14.5 More Information.....	2085
2.14.5.1 External Interfaces.....	2085
2.14.5.1.1 Shell.....	2085
2.14.5.1.2 Java API.....	2085
2.15 RTD Development Guide.....	2085
2.15.1 Overview.....	2086
2.15.1.1 Application Development Overview.....	2086
2.15.1.2 Common Concepts.....	2086
2.15.1.3 Development Process.....	2086
2.15.2 Environment Preparation.....	2088
2.15.2.1 Preparing the Development and Operating Environment.....	2088
2.15.2.2 Configuring and Importing Sample Projects.....	2089
2.15.3 Developing an Application.....	2092

2.15.3.1 Event Source Custom Plug-in Program.....	2092
2.15.3.1.1 Scenario.....	2092
2.15.3.1.2 Development Guidelines.....	2093
2.15.3.1.3 Sample Code.....	2093
2.15.3.2 Decision Engine Custom Program.....	2100
2.15.3.2.1 Scenario.....	2100
2.15.3.2.2 Development Guidelines.....	2100
2.15.3.2.3 Sample Code.....	2101
2.15.4 Commissioning the Application.....	2104
2.15.4.1 Compiling and Packaging.....	2104
2.16 Solr Development Guide.....	2111
2.16.1 Overview.....	2111
2.16.1.1 Application Development Overview.....	2111
2.16.1.2 Common Concepts.....	2112
2.16.1.3 Development Process.....	2113
2.16.2 Environment Preparation.....	2115
2.16.2.1 Development and Operating Environment.....	2115
2.16.2.2 Configuring and Importing Sample Projects.....	2118
2.16.3 Developing an Application.....	2120
2.16.3.1 Typical Scenario Description.....	2120
2.16.3.2 Development Idea.....	2120
2.16.3.3 Example Code Description.....	2121
2.16.3.3.1 Initializing Solr.....	2121
2.16.3.3.2 Querying a Collection.....	2122
2.16.3.3.3 Deleting a Collection.....	2123
2.16.3.3.4 Creating a Collection.....	2123
2.16.3.3.5 Adding a Document.....	2124
2.16.3.3.6 Querying a Document.....	2125
2.16.3.3.7 Deleting a Document.....	2126
2.16.4 Application Commissioning.....	2126
2.16.4.1 Commissioning an Application in Windows.....	2126
2.16.4.1.1 Compiling and Running an Application.....	2126
2.16.4.1.2 Viewing Commissioning Results.....	2129
2.16.4.2 Commissioning an Application in Linux.....	2132
2.16.4.2.1 Compiling and Running an Application When a Client Is Installed.....	2132
2.16.4.2.2 Compiling and Running an Application When No Client Is Installed.....	2135
2.16.4.2.3 Viewing Commissioning Results.....	2136
2.16.5 More Information.....	2139
2.16.5.1 External Interfaces.....	2139
2.16.5.1.1 Shell.....	2139
2.16.5.1.2 Java API.....	2140
2.16.5.1.3 Web UI.....	2140

2.17 Spark Development Guide.....	2142
2.17.1 Overview.....	2142
2.17.1.1 Application Development Overview.....	2142
2.17.1.2 Basic Concepts.....	2143
2.17.1.3 Development Process.....	2150
2.17.2 Preparing for the Environment.....	2153
2.17.2.1 Development and Operating Environment.....	2153
2.17.2.2 Configuring and Importing Sample Projects.....	2157
2.17.2.3 Creating a New Project (Optional).....	2173
2.17.2.4 Configuring the Python3 Sample Project.....	2174
2.17.3 Developing the Project.....	2175
2.17.3.1 Spark Core Project.....	2175
2.17.3.1.1 Overview.....	2175
2.17.3.1.2 Java Example Code.....	2177
2.17.3.1.3 Scala Sample Code.....	2179
2.17.3.1.4 Python Example Code.....	2179
2.17.3.2 Spark SQL Project.....	2180
2.17.3.2.1 Overview.....	2180
2.17.3.2.2 Java Sample Code.....	2182
2.17.3.2.3 Scala Sample Code.....	2183
2.17.3.2.4 Python Sample Code.....	2183
2.17.3.3 Accessing the Spark SQL Through JDBC.....	2184
2.17.3.3.1 Overview.....	2184
2.17.3.3.2 Java Sample Code.....	2186
2.17.3.3.3 Scala Sample Code.....	2188
2.17.3.4 Spark on HBase.....	2189
2.17.3.4.1 Performing Operation on Data in Avro Format.....	2189
2.17.3.4.2 Performing Operations on the HBase Data Source.....	2193
2.17.3.4.3 Using the BulkPut Interface.....	2196
2.17.3.4.4 Using the BulkGet Interface.....	2199
2.17.3.4.5 Using the BulkDelete Interface.....	2202
2.17.3.4.6 Using the BulkLoad Interface.....	2205
2.17.3.4.7 Using the foreachPartition Interface.....	2208
2.17.3.4.8 Distributedly Scanning HBase Tables.....	2211
2.17.3.4.9 Using the mapPartition Interface.....	2213
2.17.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming.....	2216
2.17.3.5 Reading Data from HBase and Write It Back to HBase.....	2220
2.17.3.5.1 Overview.....	2220
2.17.3.5.2 Java Example Code.....	2222
2.17.3.5.3 Scala Example Code.....	2224
2.17.3.5.4 Python Example Code.....	2226
2.17.3.6 Reading Data from Hive and Write It to HBase.....	2227

2.17.3.6.1 Overview.....	2227
2.17.3.6.2 Java Sample Code.....	2229
2.17.3.6.3 Scala Example Code.....	2231
2.17.3.6.4 Python Example Code.....	2233
2.17.3.7 Streaming Connecting to Kafka0-10.....	2233
2.17.3.7.1 Overview.....	2233
2.17.3.7.2 Java Example Code.....	2235
2.17.3.7.3 Scala Example Code.....	2238
2.17.3.8 Structured Streaming Project.....	2241
2.17.3.8.1 Overview.....	2241
2.17.3.8.2 Java Sample Code.....	2243
2.17.3.8.3 Scala Sample Code.....	2244
2.17.3.8.4 Python Sample Code.....	2245
2.17.3.9 Structured Streaming Stream-Stream Join.....	2245
2.17.3.9.1 Overview.....	2246
2.17.3.9.2 Scala Example Code.....	2249
2.17.3.10 Spark on Elasticsearch.....	2251
2.17.3.10.1 Overview.....	2251
2.17.3.10.2 Java Sample Code.....	2254
2.17.3.10.3 Scala Sample Code.....	2255
2.17.3.10.4 Python Sample Code.....	2256
2.17.3.11 Spark on Solr.....	2257
2.17.3.11.1 Overview.....	2257
2.17.3.11.2 Java Sample Code.....	2259
2.17.3.11.3 Scala Sample Code.....	2261
2.17.3.11.4 Python Sample Code.....	2262
2.17.3.12 Structured Streaming Status Operation.....	2262
2.17.3.12.1 Overview.....	2263
2.17.3.12.2 Scala Sample Code.....	2264
2.17.3.13 Synchronizing HBase Data from Spark to CarbonData.....	2267
2.17.3.13.1 Overview.....	2268
2.17.3.13.2 Java Example Code.....	2269
2.17.3.14 Using Spark to Perform Basic Hudi Operations.....	2270
2.17.3.14.1 Overview.....	2270
2.17.3.14.2 Java Example Code.....	2271
2.17.3.14.3 Scala Example Code.....	2272
2.17.3.14.4 Python Sample Code.....	2274
2.17.3.15 Compiling User-defined Configuration Items for Hudi.....	2278
2.17.3.15.1 HoodieDeltaStreamer.....	2278
2.17.3.15.2 User-defined Partitioner.....	2279
2.17.3.16 Using Spark to Write Data to ClickHouse.....	2280
2.17.3.16.1 Overview.....	2280

2.17.3.16.2 Java Sample Code.....	2282
2.17.3.16.3 Scala Sample Code.....	2283
2.17.3.16.4 Python Sample Code.....	2285
2.17.3.16.5 SQL Example.....	2286
2.17.4 Commissioning the Application.....	2287
2.17.4.1 Commissioning Applications on Windows.....	2287
2.17.4.1.1 Compiling and Running Applications.....	2287
2.17.4.1.2 View Debugging Results.....	2288
2.17.4.2 Commissioning an Application in Linux.....	2289
2.17.4.2.1 Compiling and Running the Application.....	2289
2.17.4.2.2 Checking the Commissioning Result.....	2294
2.17.5 More Information.....	2294
2.17.5.1 Common APIs.....	2294
2.17.5.1.1 Java.....	2294
2.17.5.1.2 Scala.....	2300
2.17.5.1.3 Python.....	2305
2.17.5.1.4 REST API.....	2309
2.17.5.2 Common CLIs.....	2316
2.17.5.3 JDBCServer Interface.....	2317
2.17.5.4 Structured Streaming Functions and Reliability.....	2319
2.17.5.5 FAQ.....	2324
2.17.5.5.1 How to Add a User-Defined Library.....	2324
2.17.5.5.2 How to Automatically Load Jars Packages?.....	2327
2.17.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?.....	2327
2.17.5.5.4 Why Does Kafka Fail to Receive the Data Written Back by Spark Streaming?.....	2328
2.17.5.5.5 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?.....	2329
2.17.5.5.6 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?.....	2331
2.17.5.5.7 How to Perform Remote Debugging Using IDEA?.....	2331
2.17.5.5.8 How to Submit the Spark Application Using Java Commands?.....	2334
2.17.5.5.9 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used.....	2336
2.17.5.5.10 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming.....	2337
2.17.5.5.11 Restrictions on Restoring the Spark Application from the checkpoint.....	2337
2.17.5.5.12 Support for Third-party JAR Packages on x86 and TaiShan Platforms.....	2338
2.17.5.5.13 What Should I Do If a Large Number of Directories Whose Names Start with blockmgr- or spark- Exist in the /tmp Directory on the Client Installation Node?.....	2340
2.17.5.5.14 Error Code 139 Is Reported When Python Pipeline Runs in the Arm Environment.....	2341
2.17.5.5.15 What Should I Do If the Structured Streaming Task Submission Way Is Changed?.....	2341
2.17.5.5.16 Migrating Spark Streaming's Interconnection from Kafka 0.8 to Kafka 0.10.....	2342
2.18 YARN Development Guide.....	2349
2.18.1 Overview.....	2349

2.18.2 Interfaces.....	2350
2.18.2.1 Command.....	2350
2.18.2.2 Java API.....	2357
2.18.2.3 REST API.....	2360
2.18.2.4 REST APIs of Superior Scheduler.....	2363
<b>3 Development Specifications.....</b>	<b>2379</b>
3.1 Development Environment Construction.....	2379
3.1.1 Rules.....	2379
3.2 Security Authentication.....	2379
3.2.1 Rules.....	2379
3.2.2 Suggestions.....	2380
3.3 ClickHouse.....	2380
3.3.1 Rules.....	2380
3.3.2 Suggestions.....	2382
3.4 Doris.....	2385
3.4.1 Table Creation Rules.....	2385
3.4.2 Data Change.....	2386
3.4.3 Naming Conventions.....	2387
3.4.4 Data Query.....	2387
3.4.5 Data Import.....	2388
3.4.6 UDF Development.....	2389
3.4.7 Connection and Running.....	2390
3.5 Elasticsearch.....	2390
3.5.1 Application Scenarios.....	2390
3.5.2 Rules.....	2391
3.5.3 Suggestions.....	2392
3.6 Flink.....	2393
3.6.1 Applicable Scenarios.....	2393
3.6.2 Rules.....	2393
3.6.3 Suggestions.....	2394
3.7 GraphBase.....	2394
3.7.1 Rules.....	2394
3.7.2 Recommendations.....	2395
3.8 HBase.....	2396
3.8.1 Application Scenarios.....	2396
3.8.2 Rules.....	2396
3.8.3 Suggestions.....	2401
3.8.4 Examples.....	2403
3.8.5 Appendix.....	2409
3.9 HDFS.....	2411
3.9.1 Application Scenarios.....	2411
3.9.2 Rules.....	2411

3.9.3 Suggestions.....	2415
3.10 Hive.....	2416
3.10.1 Application Scenarios.....	2416
3.10.2 Rules.....	2417
3.10.3 Suggestions.....	2421
3.10.4 Examples.....	2422
3.11 Hudi.....	2431
3.11.1 Applicable Scenarios.....	2431
3.11.2 Suggestions.....	2432
3.12 IoTDB.....	2432
3.12.1 Applicable Scenarios.....	2432
3.12.2 Rules.....	2432
3.12.3 Suggestions.....	2433
3.13 Kafka.....	2434
3.13.1 Application Scenarios.....	2434
3.13.2 Rules.....	2434
3.13.3 Suggestions.....	2435
3.14 Mapreduce.....	2435
3.14.1 Application Scenarios.....	2435
3.14.2 Rules.....	2436
3.14.3 Suggestions.....	2437
3.14.4 Examples.....	2438
3.15 Oozie.....	2440
3.15.1 Application Scenarios.....	2440
3.15.2 Rules.....	2440
3.15.3 Suggestions.....	2442
3.16 Redis.....	2443
3.16.1 Application Scenarios.....	2443
3.16.2 Rules.....	2444
3.16.3 Suggestions.....	2445
3.17 Solr.....	2445
3.17.1 Application Scenarios.....	2445
3.17.2 Rules.....	2446
3.17.3 Suggestions.....	2447
3.18 Spark.....	2449
3.18.1 Application Scenarios.....	2449
3.18.2 Rules.....	2450
3.18.3 Suggestions.....	2452
3.19 Yarn.....	2455
3.19.1 Application Scenarios.....	2455
3.19.2 Rules.....	2455
<b>4 Manager Management Development Guide.....</b>	<b>2457</b>

4.1 Overview.....	2457
4.1.1 Application Development Overview.....	2457
4.1.2 Common Concepts.....	2457
4.1.3 Development Process.....	2458
4.2 Environment Preparation.....	2460
4.2.1 Preparing Development and Running Environments.....	2460
4.2.2 Configuring and Importing Sample Projects.....	2461
4.3 Developing an Application.....	2464
4.3.1 Typical Scenario Description.....	2464
4.3.2 Development Guideline.....	2465
4.3.3 Example Code Description.....	2465
4.3.3.1 Login Authentication.....	2465
4.3.3.2 Adding Users.....	2466
4.3.3.3 Searching for Users.....	2466
4.3.3.4 Modifying Users.....	2467
4.3.3.5 Deleting Users.....	2467
4.3.3.6 Exporting a User List.....	2467
4.4 Application Commissioning.....	2468
4.4.1 Commissioning an Application in the Windows OS.....	2468
4.4.1.1 Compiling and Running an Application.....	2468
4.4.1.2 Viewing Windows Commissioning Results.....	2469
4.5 More Information.....	2472
4.5.1 External Interfaces.....	2472
4.5.1.1 Java API.....	2472
4.5.2 FAQ.....	2475
4.5.2.1 JDK1.6 Fails to Connect to the FusionInsight System Using JDK1.8.....	2475
4.5.2.2 Authentication Fails, Error Code "401" Is Returned, and Logs Displayed as "Authorize Failed"....	2476
4.5.2.3 An Operation Fails and "log4j:WARN No appenders could be found for logger(basicAuth.Main)" Is Displayed in Logs.....	2477
4.5.2.4 An Operation Fails and "illegal character in path at index 57" Is Displayed in Logs.....	2477
4.5.2.5 Run the curl Command to Access REST APIs.....	2478
<b>5 Appendix.....</b>	<b>2480</b>
5.1 Accessing FusionInsight Manager of an MRS Cluster.....	2480
5.2 Installing a Client.....	2484
5.3 Open Source Application Reconstruction Guide.....	2492

# 1 Security Mode

## 1.1 Description

### Overview

This document describes the application development process, sample projects, and code of components in the big data cluster. It also provides FAQs and development specifications for developers with Java development experience to develop applications.

### Preparations

- The cluster has been installed and is running properly.
- You have a basic understanding of components in the big data cluster.
- You have a basic understanding of Java.
- You have learned about MRS development components and how to use the Elastic Cloud Server (ECS).
- You have a basic understanding about how to use Maven.

### Obtaining Sample Projects

Log in to GitHub to obtain the files related to the component sample projects. Download the dependency JAR files of the sample projects from Huawei Mirrors, and download the rest open source dependency JAR files from the Maven central repository.

The MRS sample code library provides sample projects of basic functions of each component for users. You can import a sample project for compilation and running based on the Kerberos authentication mode of the cluster.

## 1.2 Obtaining Sample Projects from Huawei Mirrors

### Procedure

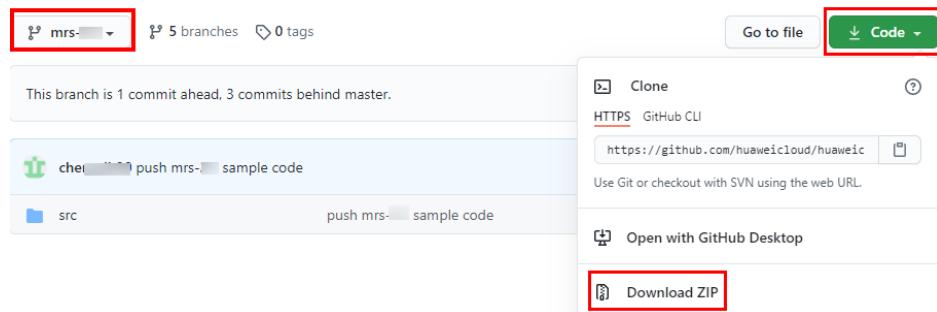
Building a sample project includes the following operations:

1. Download the Maven project source code and configuration files of the sample project. For details, see [Obtaining Sample Projects](#).
2. Configure the Maven mirror repository of the SDK in Huawei Mirrors. For details, see [Configuring Huawei Open-Source Mirrors](#).
3. Build a complete Maven project based on user requirements.

## Obtaining Sample Projects

You can download MRS sample projects at <https://github.com/huaweicloud/huaweicloud-mrs-example>.

**Figure 1-1** Downloading sample code



**Switch the branch to the version that matches the MRS cluster**, download the package to a local directory, and decompress the package to obtain the sample code project of each component. For details about the sample projects of each component of the current version, see [Sample Projects of MRS Components](#).

## Configuring Huawei Open-Source Mirrors

Huawei provides Huawei Mirrors for you to download all dependency JAR files of sample projects. However, you need to download the rest dependency open-source JAR files from the Maven central repository or other custom repository address.

### NOTE

Before using a development tool to download the dependency JAR files in the local environment, ensure that the following conditions are met:

- The local network is normal.  
Uses a browser and visit Huawei Mirrors to check whether the website can be accessed. If the access is abnormal, connect the local network.
- The proxy is disabled for the development tool.

Take the IntelliJ IDEA development tool of version 2020.2 as an example. Choose **File > Settings > Appearance & Behavior > System Settings > HTTP Proxy**, select **No proxy**, and click **OK** to save the configuration.

Perform the following steps to configure the open-source mirror warehouse.

**Step 1** Check that JDK 1.8 or later and Maven 3.0 or later have been installed.

**Step 2** Configure the Maven configuration file.

- To overwrite the Maven configuration file, choose **Huawei SDK > HuaweiCloud SDK** at Huawei Mirrors, download the **settings.xml** file, and

overwrite the < Maven installation directory >/**conf/settings.xml** file with the downloaded file.

- (Optional) If you do not want to overwrite the Maven configuration file, manually modify the **settings.xml** configuration file or the **pom.xml** file of the component sample project to configure the mirror repository address. The configuration methods are as follows:

- **Configuration method 1:** Modify the **settings.xml** configuration file.

Add the following open-source mirror repository address to **mirrors** in the **settings.xml** configuration file.

```
<mirror>
  <id>repo2</id>
  <mirrorOf>central</mirrorOf>
  <url>https://repo1.maven.org/maven2/</url>
</mirror>
```

Add the following mirror repository address to **profiles** in the **settings.xml** configuration file.

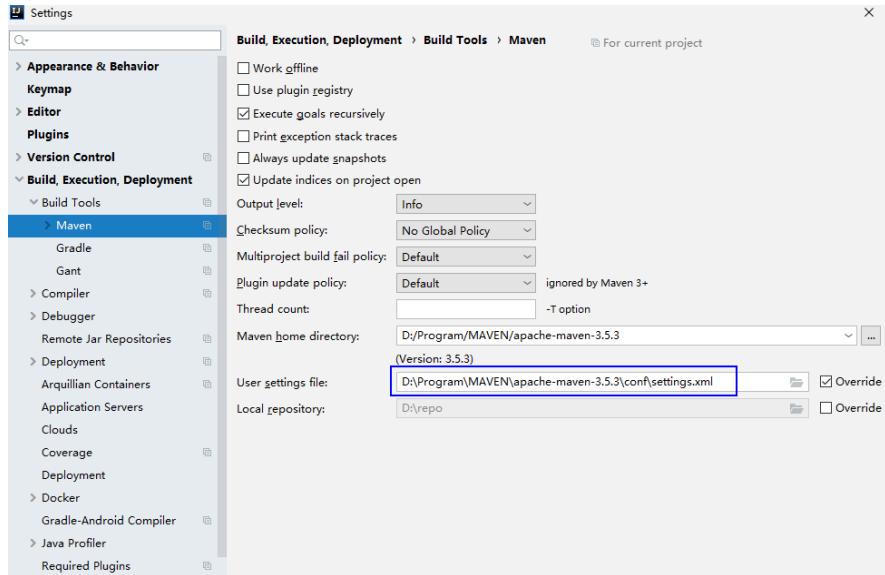
```
<profile>
  <id>huaweicloudsdk</id>
  <repositories>
    <repository>
      <id>huaweicloudsdk</id>
      <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </repository>
  </repositories>
</profile>
```

Add the following mirror repository address to the **activeProfiles** node in the **settings.xml** file.

```
<activeProfile>huaweicloudsdk</activeProfile>
```

 NOTE

- Huawei Mirrors does not provide third-party open-source JAR files. After configuring Huawei open-source mirrors, you need to separately configure third-party Maven mirror repository address.
- When using the IntelliJ IDEA development tool, you can choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven** to view the directory where the **settings.xml** file is stored.



- **Configuration method 2: Modify the `pom.xml` configuration file.**

Add the following mirror repository address directly to the **pom.xml** file in the secondary development sample project.

```
<repositories>

    <repository>
        <id>huaweicloudsdk</id>
        <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
        <releases><enabled>true</enabled></releases>
        <snapshots><enabled>true</enabled></snapshots>
    </repository>

    <repository>
        <id>central</id>
        <name>Maven Central</name>
        <url>https://repo1.maven.org/maven2/</url>
    </repository>

</repositories>
```

**Step 3** Configure the default Maven code and JDK. Add the following information to **profiles** in the **settings.xml** configuration file:

```
<profile>
<id>JDK1.8</id>
<activation>
<activeByDefault>true</activeByDefault>
<jdk>1.8</jdk>
</activation>
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<maven.compiler.encoding>UTF-8</maven.compiler.encoding>
<maven.compiler.source>1.8</maven.compiler.source>
```

```
<maven.compiler.target>1.8</maven.compiler.target>
<maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
</properties>
</profile>
```

----End

## 1.3 Sample Projects of MRS Components

The MRS sample code library provides sample projects of basic functions of each component for users. See [Table 1-1](#) for details about the sample projects of each component of the current version.

For details about how to obtain a sample project, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Table 1-1** Sample projects of each component

Component	Project Name	Project Description
ClickHouse	clickhouse-examples	As an independent DBMS system, ClickHouse allows you to use the SQL language to perform common operations. This sample project describes the implementation of the following common operations: creating connections, databases, and tables, setting attributes, inserting and querying data, and deleting tables.
Elasticsearch	elasticsearch-rest-client-example	Sample code of the Elasticsearch REST APIs. An application is developed to search for all library information, provide information about books related to search keywords, and grade the books by score. The search function can be implemented by Elasticsearch.
Flink (normal/ security)	FlinkCheckpointJavaExample	Sample project of the asynchronous checkpoint mechanism application.
	FlinkCheckpointScalaExample	Assume that you want to collect data volume in a 4-second time window every other second and the status of operators must be strictly consistent. That is, if an application recovers from a failure, the status of all operators must be the same.

Component	Project Name	Project Description
	FlinkConfigtable-JavaExample	Assume that there is a log text file about dwell durations of netizens for shopping online at weekends and a CSV table of netizen information, and you need to develop a Flink application that achieves following functions:
	FlinkConfigtableScalaExample	<ul style="list-style-type: none"> <li>Collect statistics on female netizens who spend more than two hours in total for online shopping in real time. The name fields in the log file and the CSV table can be used as keywords, based on which the two files are joined.</li> <li>The first column in the log file records names, the second column records genders, and the third column records the dwell durations in the unit of minute. Three columns are separated by comma (,).</li> </ul>
	FlinkKafkaJavaExample	Sample code for developing an application to produce and consume data in Kafka.
	FlinkKafkaScalaExample	Assume that a Flink service receives one message record every second. A Flink application is developed to output prefixed message content in real time based on service requirements.
	FlinkPipelineJavaExample	In this sample project, the publisher job generates 10,000 pieces of data each second. Data is sent by the NettySink operator from the publisher job to downstream jobs. The other two jobs are used as subscribers and each of them subscribes a piece of data.
	FlinkPipelineScalaExample	
	FlinkStreamJavaExample	Assume that you need to develop a DataStream application of Flink to perform the following operations on logs about dwell durations of netizens for shopping online at weekends:
	FlinkStreamScalaExample	<ul style="list-style-type: none"> <li>Collect statistics on female netizens who dwell on online shopping for more than two hours in real time.</li> <li>The first column in the log file records names, the second column records genders, and the third column records the dwell durations in the unit of minute. Three columns are separated by comma (,).</li> </ul>

Component	Project Name	Project Description
	FlinkStreamSqlJoinExample	<p>Assume that a Flink service receives a message every second. The message records the basic information about a user, including the name, gender, and age. Another Flink service (service 2) receives a message irregularly, and the message records the name and career information about the user.</p> <p>To meet the requirements of some services, a Flink application is developed to achieve the following function: using the username recorded in the message received by service 2 as a keyword to jointly query service data.</p>
	FlinkHBaseJavaExample	<p>Reads and writes HBase data through Flink API jobs.</p> <ol style="list-style-type: none"><li>1. Writes data to HBase:<ol style="list-style-type: none"><li>a. Specify the parent directory of the <b>hbase-site.xml</b> file. Flink Sink can obtain the HBase connection.</li><li>b. Use the connection to determine whether a table exists. If it does not, create one.</li><li>c. Convert received data into Put objects and writes the Put objects to HBase.</li></ol></li><li>2. Reads data from HBase:<ol style="list-style-type: none"><li>a. Specify the parent directory of the <b>hbase-site.xml</b> file. Flink Source can obtain the HBase connection.</li><li>b. Use the connection to determine whether a table exists. If it does not, the job fails. In this case, you need to create a table in HBase shell or an upstream job.</li><li>c. Read data from HBase, converts result data into Row objects, and sends the Row objects to downstream operators.</li></ol></li></ol>

Component	Project Name	Project Description
	FlinkHudiJavaExample	<p>The job generates one data record per second, writes the data to the Hudi table, and reads and prints the data in the Hudi table.</p> <ol style="list-style-type: none"> <li>1. Write data to Hudi:             <ol style="list-style-type: none"> <li>a. Generate data through a random data generation class.</li> <li>b. Convert the generated data into <b>DataStream&lt;RowData&gt;</b>.</li> <li>c. Write data to the Hudi table.</li> </ol> </li> <li>2. Read data from Hudi:             <ol style="list-style-type: none"> <li>a. Read data from the Hudi table.</li> <li>b. Combine the read data into the JSON format and print the data.</li> </ol> </li> </ol>
	pyflink-example	Uses Python to read and write Kafka and submit SQL jobs.
HBase	hbase-example	Assume that you need to develop an application to manage information about users of service A in an enterprise and provide the following functions: creating user information tables, adding the educational background and title of users to the tables, querying user names and addresses based on user IDs, and collecting and deleting data.
	hbase-rest-example	Sample code for accessing the HBase service using REST APIs.
	hbase-thrift-example	Sample code for accessing the HBase service using Thrift APIs.
	hbase-zk-example	Sample code for connecting to the FusionInsight ZooKeeper and the third-party ZooKeeper to process HBase services.
HDFS	hdfs-example-normal	Service operation objects of HDFS are files. File operations covered in the sample code include creating a folder, writing data to a file, appending file content, reading a file, and deleting a file or folder. Other operations on HDFS include setting file access permissions.
	hdfs-example-security	

Component	Project Name	Project Description
HetuEngine	hetu-examples-normal	Assume that you need to join table A of the Hive data source and table B of the MPPDB data source to develop an application for querying data of the Hive data source using HetuEngine. The process is as follows: connecting to the HetuEngine JDBC Server, assembling SQL statements, running the statements, analyzing results, and disabling the connection.
	hetu-examples-security	
	hetu-examples-python3	
Hive	hcatalog-example	Sample code for obtaining Hive metadata information from MRS.  A Hive data analysis application is developed to manage employee information of an enterprise by analyzing the data in the employee information table and employee contact information table.
	hive-jdbc-example	Sample code for connecting to HiveServer using JDBC.
	hive-jdbc-example-multizk	Sample code for connecting to HiveServer with multiple ZooKeeper IP addresses using JDBC.
	python-examples	Sample code for connecting to HiveServer using Python 2.
	python3-examples	Sample code for connecting to HiveServer using Python 3.
IoTDB	iotdb-flink-example	Sample code for sending data from a Flink job to an IoTDB server. <ul style="list-style-type: none"> <li>A simulated source SensorSource generates a data point every second.</li> <li>Flink uses IoTDBSink to consume produced data and write the data into IoTDB.</li> </ul> Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.
	iotdb-jdbc-example	Sample code for running IoTDB SQL statements in JDBC connection mode.
	iotdb-session-example	Sample code for running IoTDB SQL statements in Session connection mode.

Component	Project Name	Project Description
Kafka	kafka-examples	Kafka is a distributed message system, in which messages can be published or subscribed. A Producer is developed to send a message to a topic of a Kafka cluster every second, and a Consumer is implemented to ensure that the topic is subscribed and that messages of the topic are consumed in real time.
Manager	manager-examples	Assume that you need to work with FusionInsight Manager in non-GUI mode. An HTTP basic authentication-based application is required to provide the following functions: <ul style="list-style-type: none"><li>• Log in to FusionInsight Manager.</li><li>• Log in to FusionInsight Manager for data querying, adding, or deleting.</li></ul>
MapReduce	mapreduce-example-normal	The sample project describes the MapReduce statistics sample application and the sample application for MapReduce to access multiple components.
	mapreduce-example-security	
Oozie (normal/ security)	OozieMapReduceExample	The sample project describes how to submit MapReduce jobs and query job status using Java APIs. The sample code involves only MapReduce jobs.
	OozieSparkHBaseExample	Sample code for scheduling Spark to access HBase and Hive using Oozie.
	OozieSparkHiveExample	
Redis	redis-examples	The service operation object of Redis is key. Operations covered by the sample code include access operations for keys of the String, List, Hash, Set, and Sorted Set types.
RTD (normal/ security)	rtd-example	This example describes how to customize event source plug-ins and decision engine configuration sets.

Component	Project Name	Project Description
Solr	solr-example	<p>An application is developed to manage employees and search for their personal information. Solr can provide the searching service.</p> <p>Operations involved in the sample code include establishing a connection with the Solr server, querying a collection, deleting a collection, creating a collection, adding one or more documents, performing simple search, and deleting a document.</p>
Spark (normal/ security)	SparkHbaseToCarbonJavaExample	<p>Data is written to HBase in real time for point query services and is synchronized to CarbonData tables in batches at a specified interval for analytical query services.</p>
	SparkHbaseToHbaseJavaExample	<p>Assume that table1 of HBase stores a user's consumption amount on the current day and table2 stores the user's history consumption amount.</p> <p>In table1, the <b>key=1,cf:cid=100</b> record indicates that user1's consumption amount on the current day is CNY 100.</p>
	SparkHbaseToHbasePythonExample	<p>In table2, the <b>key=1,cf:cid=1000</b> record indicates that user1's history consumption amount is CNY 1000.</p> <p>Based on some service requirements, a Spark application is developed to implement the following functions:</p> <p>Calculate a user's history consumption amount based on the user name, that is, the user's total consumption amount = CNY 100 (consumption amount of the current day) + CNY 1000 (history consumption amount).</p> <p>In the preceding example, the application run result is that in table2, the total consumption amount of user1 (<b>key=1</b>) is CNY 1100 (<b>cf:cid=1100</b>).</p>
	SparkOnClickHouseJavaExample SparkOnClickHousePythonExample SparkOnClickHouseScalaExample	<p>In Spark applications, the native APIs of ClickHouse JDBC and the Spark JDBC driver can be used to create, query, and insert ClickHouse databases and tables.</p>

Component	Project Name	Project Description
	SparkHivetoHbase JavaExample	Assume that <b>person</b> table of Hive stores a user's consumption amount on the current day and HBase table2 stores the user's history consumption amount data. In the <b>person</b> table, the <b>name=1,account=100</b> record indicates that user1's consumption amount on the current day is CNY 100.
	SparkHivetoHbase PythonExample	In the <b>person</b> table, the <b>name=1,account=100</b> record indicates that user1's consumption amount on the current day is CNY 100.
	SparkHivetoHbase ScalaExample	In table2, the <b>key=1,cf:cid=1000</b> record indicates that user1's history consumption amount is CNY 1000. Based on some service requirements, a Spark application is developed to implement the following functions: Calculate a user's history consumption amount based on the user name, that is, the user's total consumption amount = CNY 100 (consumption amount of the current day) + CNY 1000 (history consumption amount). In the preceding example, the application run result is that in table2, the total consumption amount of user1 ( <b>key=1</b> ) is CNY 1100 ( <b>cf:cid=1100</b> ).
	SparkJavaExample	A Spark application is developed to perform the following operations on logs about netizens' dwell time for online shopping at weekends.
	SparkPythonExample	<ul style="list-style-type: none"><li>Collect statistics on female netizens who dwell on online shopping for more than two hours at weekends.</li><li>The first column in the log file records names, the second column records genders, and the third column records the dwell durations in the unit of minute. Three columns are separated by comma (,).</li></ul>
	SparkLauncherJavaExample	Sample code for submitting Spark applications using the <b>org.apache.spark.launcher.SparkLauncher</b> class through Java commands.
	SparkLauncherScalaExample	Sample code for submitting Spark applications using the <b>org.apache.spark.launcher.SparkLauncher</b> class through Scala commands.

Component	Project Name	Project Description
	SparkOnEsJavaExample	In Spark, you can read, write, and collect statistics on the data in Elasticsearch using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.
	SparkOnEsPythonExample	
	SparkOnEsScalaExample	
	SparkOnHbaseJavaExample	HBase can be used as data sources in Spark applications. In this sample project, data is stored in HBase in Avro format. Data is read from the HBase, and the read data is filtered.
	SparkOnHbasePythonExample	
	SparkOnHbaseScalaExample	
	SparkOnHudiJavaExample	Spark is used to perform operations such as data insertion, query, update, incremental query, query at a specific time point, and data deletion on Hudi.
	SparkOnHudiPythonExample	
	SparkOnHudiScalaExample	
	SparkOnMultiHbaseScalaExample	Sample code for concurrent access from Spark to HBase in two clusters.
	SparkOnSolrJavaExample	In the Spark application, Solr client APIs are used to create indexes, insert data, and search for data.
	SparkOnSolrPythonExample	
	SparkOnSolrScalaExample	
	SparkSQLJavaExample	A Spark application is developed to perform the following operations on logs about netizens' dwell time for online shopping at weekends.
	SparkSQLPythonExample	<ul style="list-style-type: none"><li>Collect statistics on female netizens who dwell on online shopping for more than two hours at weekends.</li></ul>
	SparkSQLScalaExample	<ul style="list-style-type: none"><li>The first column in the log file records names, the second column records genders, and the third column records the dwell durations in the unit of minute. Three columns are separated by comma (,).</li></ul>

Component	Project Name	Project Description
	SparkStreamingKafka010JavaExample	Assume that Kafka receives one word record every second in a service. Based on some service requirements, a Spark application is developed to implement the following function: Calculate the total number of records of each word in real time.
	SparkStreamingKafka010ScalaExample	
	SparkStructuredStreamingJavaExample	In Spark applications, Structured Streaming is used to call Kafka APIs to obtain word records. Word records are classified to obtain the number of records of each word.
	SparkStructuredStreamingPythonExample	
	SparkStructuredStreamingScalaExample	
	SparkThriftServerJavaExample	On the custom JDBCServer client, JDBC connection is used to load, query, and delete data, create tables, and the JDBC APIs are used to submit data analysis tasks and return results.
	SparkThriftServerScalaExample	
	StructuredStreamingADScalaExample	Structured Streaming is used to read advertisement request data, display data, and click data from Kafka, obtain effective display statistics and click statistics in real time, and write the statistics to Kafka.
	StructuredStreamingStateScalaExample	In the Spark structure flow application, the number of events in each session and the start and end timestamp of the sessions are collected in different batches. At the same time, the system exports the sessions that are in the updated state in this batch.

## 1.4 Security Authentication

### 1.4.1 Security Authentication Principles and Mechanisms

#### Function

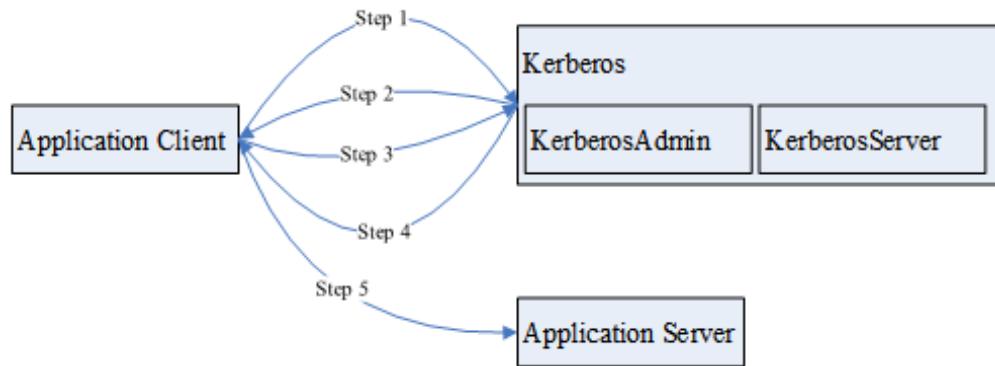
For clusters in security mode with Kerberos authentication enabled, security authentication is required during application development.

Kerberos, named after the ferocious three-headed guard dog of Hades from Greek mythology, is now used to a concept in authentication. The Kerberos protocol adopts a client-server model and cryptographic algorithms such as AES (Advanced Encryption Standard). It provides mutual authentication, that is, both the client and the server can verify each other's identity. Kerberos is used to prevent interception and replay attacks and protect data integrity. It is a system that manages keys by using a symmetric key mechanism.

## Architecture

Kerberos architecture is shown in [Figure 1-2](#) and module description in [Table 1-2](#).

**Figure 1-2** Kerberos architecture



**Table 1-2** Module description

Module	Description
Application Client	An application client, which is usually an application that submits tasks or jobs
Application Server	An application server, which is usually an application that an application client accesses
Kerberos	A service that provides security authentication
KerberosAdmin	A process that provides authentication user management
KerberosServer	A process that provides authentication ticket distribution

The process and principle are described as follows:

An application client can be a service in a cluster or a secondary development application of the customer. An application client can submit tasks or jobs to an application service.

1. Before submitting a task or job, the application client needs to apply for a ticket granting ticket (TGT) from the Kerberos service to establish a secure session with the Kerberos server.
2. After receiving the TGT request, the Kerberos service resolves parameters in the request to generate a TGT, and uses the key of the username specified by the client to encrypt the response.
3. After receiving the TGT response, the application client (based on the underlying RPC) resolves the response and obtains the TGT, and then applies for a server ticket (ST) of the application server from the Kerberos service.
4. After receiving the ST request, the Kerberos service verifies the TGT validity in the request and generates an ST of the application service, and then uses the application service key to encrypt the response.
5. After receiving the ST response, the application client packages the ST into a request and sends the request to the application server.
6. After receiving the request, the application server uses its local application service key to resolve the ST. After successful verification, the request becomes valid.

## Basic Concepts

The following concepts can help users learn the Kerberos architecture quickly and understand the Kerberos service better. The following uses security authentication for HDFS as an example.

### TGT

A TGT is generated by the Kerberos service and used to establish a secure session between an application and the Kerberos server. The validity period of a TGT is 24 hours. After 24 hours, the TGT expires automatically.

The following describes how to apply for a TGT (HDFS is used as an example):

1. Obtain a TGT through an API provided by HDFS.

```
/*
 * login Kerberos to get TGT, if the cluster is in security mode
 * @throws IOException if login is failed
 */
private void login() throws IOException {
    // not security mode, just return
    if (! "kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
        return;
    }

    //security mode
    System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);

    UserGroupInformation.setConfiguration(conf);
    UserGroupInformation.loginUserFromKeytab(PRINCIPAL_NAME, PATH_TO_KEYTAB);
}
```

2. Run shell commands on the client in kinit mode. For details about how to use shell commands, see *Shell O&M Command Description*.

### ST

An ST is generated by the Kerberos service and used to establish a secure session between an application and application service. An ST is valid only once.

In FusionInsight products, the generation of an ST is based on the Hadoop-RPC communication. The underlying RPC submits a request to the Kerberos server and the Kerberos server generates an ST.

## Sample Authentication Code

```
package com.huawei.bigdata.hdfs.examples;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.security.UserGroupInformation;

public class KerberosTest {
    private static String PATH_TO_HDFS_SITE_XML = KerberosTest.class.getClassLoader().getResource("hdfs-site.xml")
        .getPath();
    private static String PATH_TO_CORE_SITE_XML = KerberosTest.class.getClassLoader().getResource("core-site.xml")
        .getPath();
    private static String PATH_TO_KEYTAB =
        KerberosTest.class.getClassLoader().getResource("user.keytab").getPath();
    private static String PATH_TO_KRB5_CONF =
        KerberosTest.class.getClassLoader().getResource("krb5.conf").getPath();
    private static String PRINCIPAL_NAME = "develop";
    private FileSystem fs;
    private Configuration conf;

    /**
     * initialize Configuration
     */
    private void initConf() {
        conf = new Configuration();

        // add configuration files
        conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));
        conf.addResource(new Path(PATH_TO_CORE_SITE_XML));
    }

    /**
     * login Kerberos to get TGT, if the cluster is in security mode
     * @throws IOException if login is failed
     */
    private void login() throws IOException {
        // not security mode, just return
        if (! "kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
            return;
        }

        //security mode
        System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);

        UserGroupInformation.setConfiguration(conf);
        UserGroupInformation.loginUserFromKeytab(PRINCIPAL_NAME, PATH_TO_KEYTAB);
    }

    /**
     * initialize FileSystem, and get ST from Kerberos
     * @throws IOException
     */
    private void initFileSystem() throws IOException {
        fs = FileSystem.get(conf);
    }

    /**
```

```
* An example to access the HDFS
* @throws IOException
*/
private void doSth() throws IOException {
    Path path = new Path("/tmp");
    FileStatus fStatus = fs.getFileStatus(path);
    System.out.println("Status of " + path + " is " + fStatus);
    //other thing
}

public static void main(String[] args) throws Exception {
    KerberosTest test = new KerberosTest();
    test.initConf();
    test.login();
    test.initFileSystem();
    test.doSth();
}
}
```

#### NOTE

1. During Kerberos authentication, you need to configure the file parameters required for configuring the Kerberos authentication, including the keytab path, Kerberos authentication username, and the **krb5.conf** configuration file of the client for Kerberos authentication.
2. Method **login()** indicates calling the Hadoop API to perform Kerberos authentication and generating a TGT.
3. Method **doSth** indicates calling the Hadoop API to access the file system. In this situation, the underlying RPC automatically carries the TGT to Kerberos for verification and then an ST is generated.
4. The preceding code can be used to create **KerberosTest.java** in the HDFS secondary development sample project in security mode and run and view the commissioning result. For details, see [HDFS Development Guide](#).

## 1.4.2 Preparing a Developer Account

### Scenario

A developer account is used to run the sample project. When developing components for different services, you need to assign different user permissions.

### Procedure

**Step 1** Log in to FusionInsight Manager.

**Step 2** Choose **System > Permission > Role > Create Role**.

1. Enter a role name, for example, **developrole**.
2. Check whether Ranger authentication is enabled. For details, see "How Do I Determine Whether the Ranger Authentication Is Used for a Service?" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.
  - If yes, go to **Step 3**.
  - If no, edit the role to add the permissions required for service development based on the permission control type of the service. For details, see [Table 1-3](#).

**Table 1-3** List of permissions

Service	Permissions to Be Granted
HDFS	In <b>Configure Resource Permission</b> , choose <i>Name of the desired cluster</i> > <b>HDFS</b> > <b>File System</b> , select <b>Read</b> , <b>Write</b> , and <b>Execute</b> for <b>hdfs://hacluster</b> , and click <b>OK</b> .
MapReduce/ Yarn	<ol style="list-style-type: none"><li>1. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b>, and select <b>Read</b>, <b>Write</b>, and <b>Execute</b> for the user. Choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b> &gt; <b>user</b>, select <b>Read</b>, <b>Write</b>, and <b>Execute</b> for <b>mapred</b>. To execute multiple component cases, perform the following operations: Choose <i>Name of the desired cluster</i> &gt; <b>HBase</b> &gt; <b>HBase Scope</b> &gt; <b>global</b> and select the <b>default</b> option <b>create</b>. Choose <i>Name of the desired cluster</i> &gt; <b>HBase</b> &gt; <b>HBase Scope</b> &gt; <b>global</b> &gt; <b>hbase</b>, select <b>hbase:meta</b>, and click <b>Execute</b>. Choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b> &gt; <b>user</b>, and select <b>Read</b>, <b>Write</b>, and <b>Execute</b> for <b>Hive</b>. Choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b> &gt; <b>user</b> &gt; <b>hive</b>, and select <b>Read</b>, <b>Write</b>, and <b>Execute</b> for <b>warehouse</b>. Choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b> &gt; <b>tmp</b>, and select <b>Read</b>, <b>Write</b> and <b>Execute</b> for <b>hive-scratch</b>. If examples exist, select <b>Read</b>, <b>Write</b>, <b>Execute</b>, and <b>recursion</b> for <b>example</b>. Choose <i>Name of the desired cluster</i> &gt; <b>Hive</b> &gt; <b>Hive Read Write Privileges</b> and select <b>Query</b>, <b>Insert</b>, <b>Create</b> and <b>recursion</b> for <b>default</b>. Click <b>OK</b>.</li><li>2. Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Yarn</b> &gt; <b>Scheduler Queue</b> &gt; <b>root</b>, select the default option <b>Submit</b>, and click <b>OK</b>.</li></ol>
HBase	In <b>Configure Resource Permission</b> , choose <i>Name of the desired cluster</i> > <b>HBase</b> > <b>HBase Scope</b> > <b>global</b> , select the <b>admin</b> , <b>create</b> , <b>read</b> , <b>write</b> , and <b>execute</b> permissions, and click <b>OK</b> .

Service	Permissions to Be Granted
Spark	<ol style="list-style-type: none"> <li>(Configure this parameter if HBase is installed.) In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>HBase</b> &gt; <b>HBase Scope</b> &gt; <b>global</b>, select the default option <b>create</b>, and click <b>OK</b>.</li> <li>(Configure this parameter if HBase is installed.) Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>HBase</b> &gt; <b>HBase Scope</b> &gt; <b>global</b> &gt; <b>hbase</b>, select <b>execute</b> for <b>hbase:meta</b>, and click <b>OK</b>.</li> <li>Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b> &gt; <b>user</b>, select <b>Execute</b> for <b>hive</b>, and click <b>OK</b>.</li> <li>Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b> &gt; <b>user</b> &gt; <b>hive</b>, select <b>Read</b>, <b>Write</b>, and <b>Execute</b> for <b>warehouse</b>, and click <b>OK</b>.</li> <li>Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Hive</b> &gt; <b>Hive Read Write Privileges</b>, select the default option <b>Create</b>, and click <b>OK</b>.</li> <li>Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Yarn</b> &gt; <b>Scheduler Queue</b> &gt; <b>root</b>, select the default option <b>Submit</b>, and click <b>OK</b>.</li> </ol>
Hive	<p>In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Yarn</b> &gt; <b>Scheduler Queue</b> &gt; <b>root</b>, select the default option <b>Submit</b> and <b>Admin</b>, and click <b>OK</b>.</p> <p><b>NOTE</b> Extra operation permissions required for Hive application development must be obtained from the system administrator.</p>
ClickHouse	<p>In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>ClickHouse</b> &gt; <b>ClickHouse Scope</b> and select <b>Create Privilege</b> for the target database Click the database name, select the read and write permissions of the corresponding table based on the task scenario, and click <b>OK</b>.</p>

Service	Permissions to Be Granted
IoTDB	<ol style="list-style-type: none"><li>In the <b>Configure Resource Permission</b> table, choose <i>Name of the desired cluster</i> &gt; <b>IoTDB</b> &gt; <b>Common User Permission</b>, select <b>Set Database</b> for the <b>root</b> directory, and click <b>OK</b>.</li><li>In the <b>Configure Resource Permission</b> table, choose <i>Name of the desired cluster</i> &gt; <b>IoTDB</b> &gt; <b>Common User Permission</b> &gt; <b>root</b>, select the corresponding storage group, select the <b>Create</b>, <b>Modify</b>, <b>Write</b>, <b>Read</b>, or <b>Delete</b> permission based on the task scenario, and click <b>OK</b>.</li></ol>
HetuEngine	<ol style="list-style-type: none"><li>In the <b>Configure Resource Permission</b> table, choose <i>Name of the desired cluster</i> &gt; <b>Hive</b> and select <b>Hive Admin Privilege</b>.</li><li>In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Hive</b> &gt; <b>Hive Read Write Privileges</b>, select permissions based on task scenarios, and click <b>OK</b>.</li></ol> <p><b>NOTE</b> Scenario<ul style="list-style-type: none"><li>○ In the default database, select <b>Query</b> to query the tables of other users.</li><li>○ In the default database, select <b>Delete</b> and <b>Insert</b> to import data to tables of other users.</li></ul></p>
Flink	<ol style="list-style-type: none"><li>In the <b>Configure Resource Permission</b> table, choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b> &gt; <b>hdfs://hacluster/</b> &gt; <b>flink</b>, select <b>Read</b>, <b>Write</b>, and <b>Execute</b>, and click <b>Service</b> in the <b>Configure Resource Permission</b> table to return.</li><li>In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Yarn</b> &gt; <b>Scheduler Queue</b> &gt; <b>root</b>, select the default option <b>Submit</b>, and click <b>OK</b>.</li></ol> <p><b>NOTE</b> If <b>state backend</b> is set to a path on HDFS, for example, <b>hdfs://hacluster/flink-checkpoint</b>, configure the read, write, and execute permissions on the <b>hdfs://hacluster/flink-checkpoint</b> directory.</p>
Solr	-
Elasticsearch	-
GraphBase	-
Kafka	-
Redis	In <b>Configure Resource Permission</b> , choose <i>Name of the desired cluster</i> > <b>Redis</b> > <b>Redis Access Manage</b> , select <b>Read</b> , <b>Write</b> , and <b>Management</b> , and click <b>OK</b> .

Service	Permissions to Be Granted
Oozie	<ol style="list-style-type: none"><li>1. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Oozie</b> &gt; <b>Common User Privileges</b>, and click <b>OK</b>.</li><li>2. Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>HDFS</b> &gt; <b>File System</b>, select <b>Read</b>, <b>Write</b>, and <b>Execute</b> for <code>hdfs://hacluster</code>, and click <b>OK</b>.</li><li>3. Edit the role. In <b>Configure Resource Permission</b>, choose <i>Name of the desired cluster</i> &gt; <b>Yarn</b>, select <b>Cluster Admin Operations</b>, and click <b>OK</b>.</li></ol>

**Step 3** Choose **System > Permission > User Group > Create User Group** to create a user group for the sample project, for example, **developgroup**.

**Step 4** Choose **System > Permission > User > Create** to create a user for the sample project.

**Step 5** Enter a username, for example, **developuser**, select a user type and user group to which the user is to be added according to [Table 1-4](#), bind the role **developrole** obtain permissions, and click **OK**.

**Table 1-4** User type and user group list

Service	User Type	User Group
HDFS	Machine-Machine	Join the <b>developgroup</b> and <b>supergroup</b> groups. Set the primary group to <b>supergroup</b> .
MapReduce/Yarn	Machine-Machine	Join the <b>developgroup</b> group.
HBase	Machine-Machine	Join the <b>hadoop</b> group.

Service	User Type	User Group
Spark	Machine-Machine/Human-Machine	<p>Join the <b>developgroup</b> group.</p> <p>If the user needs to interconnect with Kafka, add the <b>Kafkaadmin</b> user group.</p> <p>If you need to connect to Elasticsearch, create a <b>Human-Machine</b> user and join the <b>elasticsearch</b> group.</p>
Hive	Machine-Machine/Human-Machine	Join the <b>hive</b> group.
Solr	Machine-Machine	Join the <b>solr</b> group.
Elasticsearch	Machine-Machine	<p>Join the <b>elasticsearch</b> group.</p> <p>To execute the OpenDistro JDBC sample of Elasticsearch, you need to join the <b>supergroup</b> group and set it as the primary group.</p>
Kafka	Machine-Machine	Join the <b>kafkaadmin</b> group.

Service	User Type	User Group
HetuEngine	<b>Human-Machine</b>	Join the <b>hive</b> group. Set the primary group to <b>hive</b> .
Redis	<b>Machine-Machine</b>	Join the <b>developgroup</b> group.
ClickHouse	<b>Machine-Machine/Human-Machine</b>	Join the <b>developgroup</b> and <b>supergroup</b> (primary) groups and add a role with the ClickHouse permission.
Oozie	<b>Human-Machine</b>	Join the <b>hadoop</b> , <b>supergroup</b> , and <b>hive</b> groups If the multi-instance function is enabled for Hive, the user must belong to a specific Hive instance group, for example, <b>hive3</b> .
GraphBase	<b>Human-Machine</b>	Join the <b>graphbaseadmin</b> , <b>graphbasedeveloper</b> , or <b>graphbaseoperator</b> group.
Flink	<b>Human-Machine</b>	Join the <b>developgroup</b> and <b>hadoop</b> groups. Set the primary group to <b>developgroup</b> . <b>NOTE</b> If a user wants to interconnect with Kafka, a hybrid cluster with Flink and Kafka components is required, or cross-cluster mutual trust needs to be configured for the cluster with Flink and the cluster with Kafka components. Additionally, the created Flink user is added to the <b>kafkaadmin</b> user group.

Service	User Type	User Group
IoTDB	Machine-Machine	Join the <b>iotdbgroup</b> group.

- Step 6** If Ranger authentication is enabled for the service, in addition to the permissions of the default user group and role, grant required permissions to the user or its role or user group on the Ranger web UI after the user is created. For details, see "Component Operation Guide" > "Using Ranger" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.
- Step 7** On the homepage of FusionInsight Manager, choose **System** > **Permission** > **User**. Select **developuser** from the user list and click **More** > **Download Authentication Credential** to download authentication credentials. Save the downloaded package and decompress the file to obtain **user.keytab** and **krb5.conf** files. These files are used for security authentication during the sample project. For details, see the corresponding service development guide.

#### NOTE

If the user type is human-machine, you need to change the initial password before downloading the authentication credential file. Otherwise, **Password has expired - change password to reset** is displayed when you use the authentication credential file. As a result, security authentication fails.

----End

### 1.4.3 Handling an Authentication Failure

#### Symptom

An authentication failure occurs during the commissioning and running of an example project.

#### Troubleshooting Process

There are many reasons that will cause an authentication failure. You are advised to perform the following steps for troubleshooting in different scenarios:

- Step 1** Check whether the network connection between the device where this application runs and the cluster is normal, and check whether the TCP and UDP ports required by Kerberos authentication can be accessed.
- Step 2** Check whether each configuration file is correctly read and stored in a correct directory.

- Step 3** Ensure that the username and keytab file are obtained according to the operation guide.
- Step 4** Check whether the configuration information is properly set before initiating an authentication.
- Step 5** Check whether multiple authentication requests are initiated in the same process. That is, check whether the **login()** method is called repeatedly.
- Step 6** If the problem persists, contact Huawei engineers for further analysis.

----End

## Authentication Failure Examples

"clock skew too great" is displayed during authentication.

- Step 1** Check the cluster time.
- Step 2** Check the time of the machine where the development environment is located. The difference between the machine time and the cluster time must be less than 5 minutes.

----End

"(Receive time out) can not connect to kdc server" is displayed during authentication.

- Step 1** Check whether the content of the **krb5.conf** file is correct. That is, check whether the file content is the same as the service IP address configuration of KerberosServer in the cluster.
- Step 2** Check whether the Kerberos service is running properly.
- Step 3** Check whether the firewall is disabled.

----End

An exception occurs when a client application submits a task to the Hadoop cluster, and the message "Failed to find any Kerberos tgt" or "No valid credentials provided" is displayed.

- Step 1** Check whether the **kinit** command is executed. If no, perform kinit authentication before submitting the task.
- Step 2** In the multi-thread scenario, when the process starts, call the **loginFromKeytab** function provided by Hadoop to log in to KDC to obtain TGT. Before submitting tasks, call the **reloginFromKeytab** function to update the TGT.

```
// After the first login succeeds, set user group information.  
UserGroupInformation.loginUserFromKeytab(this.userPrincipal, this.keytabFile);  
// Before the thread submits the task:  
UserGroupInformation.getLoginUser().reloginFromKeytab();
```
- Step 3** If multiple scripts use the **kinit** command to authenticate the same user, run the **export KRB5CCNAME=keytab\_path** command before running the **kinit** command in each script to ensure that the path specified by **KRB5CCNAME** in each script process is inconsistent.

----End

# 1.5 ClickHouse Development Guide

## 1.5.1 Overview

### 1.5.1.1 Introduction to ClickHouse

#### ClickHouse

ClickHouse is a column-based database oriented to online analysis and processing. It supports SQL query and provides good query performance. The aggregation analysis and query performance based on large and wide tables is excellent, which is one order of magnitude faster than other analytical databases.

Advantages for ClickHouse:

- High data compression ratio
- Multi-core parallel computing
- Vectorized computing engine
- Supporting nested data structure
- Supporting sparse indexes
- Supporting INSERT and UPDATE

**ClickHouse application scenarios:**

- Real-time data warehouse

The streaming computing engine (such as Flink) is used to write real-time data to ClickHouse. With the excellent query performance of ClickHouse, Multi-dimensional and multi-mode real-time query and analysis requests can be responded within subseconds.

- Offline query

Large-scale service data is imported to ClickHouse and constructs a large wide table with hundreds of millions to tens of billions of records and hundreds of dimensions. It supports personalized statistics collection and continuous exploratory query and analysis at any time to assist business decision-making and provides excellent query experience.

#### Introduction to the ClickHouse Development Interface

ClickHouse is developed using C++ and positioned as a DBMS. It supports HTTP and Native TCP network interface protocols and multiple driver modes such as JDBC and ODBC. You are advised to use [clickhouse-jdbc](#) of the community version for application development.

### 1.5.1.2 Basic Concepts

#### Concepts

- **cluster**

Cluster: Cluster is a logical concept in ClickHouse. It can be defined by users as required, which is different from the general understanding of cluster. Multiple ClickHouse nodes are loosely coupled and exist independently.

- **shards**

Shard: A shard is a horizontal division of a cluster. A cluster can consist of multiple shards.

- **replicas**

Replica: One shard can contain multiple replicas.

- **partition**

Partition: A partition is used for local replicas and can be considered as a vertical division.

- **MergeTree**

ClickHouse has a huge table engine system. As the basic table engine of the family system, MergeTree provides functions such as data partitioning, primary indexes, and secondary indexes. When creating a table, you need to specify the table engine. Different table engines determine the final character of a data table, for example, the features of a data table, the form how data is stored, and how data is loaded.

### 1.5.1.3 Development Process

[Figure 1-3](#) and [Table 1-5](#) describe the phases in the development process.

Figure 1-3 ClickHouse application development process

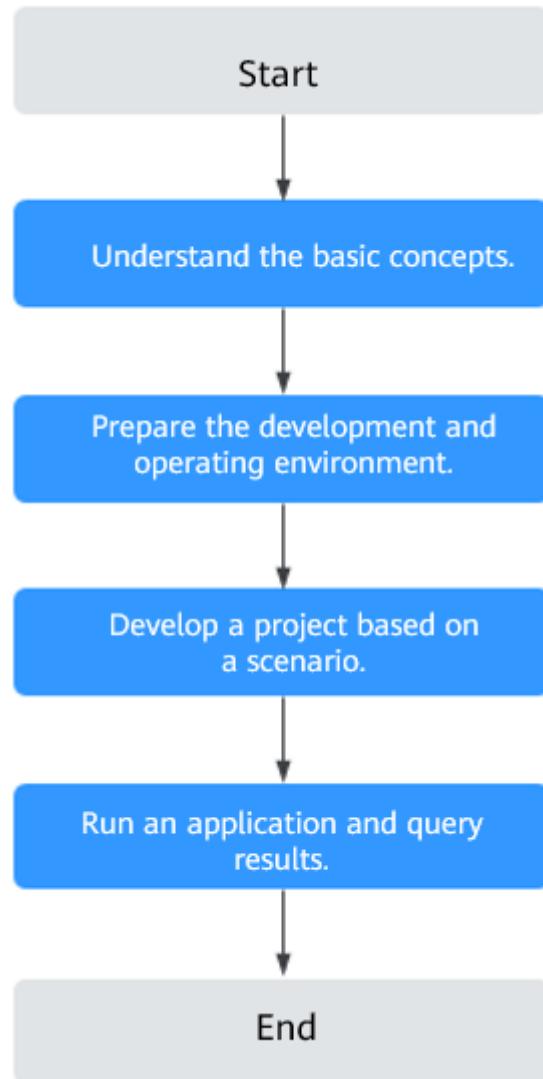


Table 1-5 Description of the ClickHouse application development process

Phase	Description	Reference
Understand the basic concepts.	Before application development, you need to understand basic concepts of ClickHouse.	<a href="#">Concepts</a>
Prepare the development and operating environment.	ClickHouse applications can be developed in multiple languages, mainly Java. The IntelliJ IDEA tool is recommended. Configure the development environment based on the guide.	<a href="#">Preparing the Development and Operating Environment</a>

Phase	Description	Reference
Develop a project based on a scenario.	Sample projects are provided to help you quickly understand APIs of ClickHouse components.	<a href="#">Configuring and Importing a Sample Project</a>
Run application and query results.	You can check the application running status from the running results.	<a href="#">Commissioning Applications on Windows</a> <a href="#">Commissioning Applications on Linux</a>

## 1.5.2 Environment Preparations

### 1.5.2.1 Preparing the Development and Operating Environment

#### Preparing the Development Environment

[Table 1-6](#) describes the environment required for application development.

**Table 1-6** Development environment

Item	Description
Operating System (OS)	<ul style="list-style-type: none"><li>• Development environment: Windows 7 or later.</li><li>• Operating environment: Linux</li></ul> If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.

Item	Description
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
Installing and configuring IntelliJ IDEA	<p>Basic configuration of the development environment. You are advised to use version 2019.1 or other compatible versions.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li><li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li><li>• If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>

Item	Description
Apache Maven installation	Basic configuration of the development environment. This operation is used for project management throughout the lifecycle of software development.
Developer account preparation	Prepare a cluster user for application development and grant permissions to the user. For details, see <a href="#">Preparing a Developer Account</a> .
7-zip	This tool is used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

## Preparing the Operating Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning application; configure the network connection, and commission the application in Windows.
  - a. **Accessing FusionInsight Manager of an MRS Cluster.** In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.
  - b. Copy all items from the hosts file in the decompression directory to the hosts file on the node where the client is installed. Ensure that the network communication between the local PC and hosts listed in the hosts file in the decompression directory is normal.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client in a directory, for example, `/opt/hadoopclient`, on the node.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the `/etc/hosts` file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides. Ensure that the local host can communicate with each host in the cluster on the network.

### 1.5.2.2 Configuring and Importing a Sample Project

#### Context

Obtain the ClickHouse development sample project and import the project to IntelliJ IDEA to learn the sample project.

#### Prerequisites

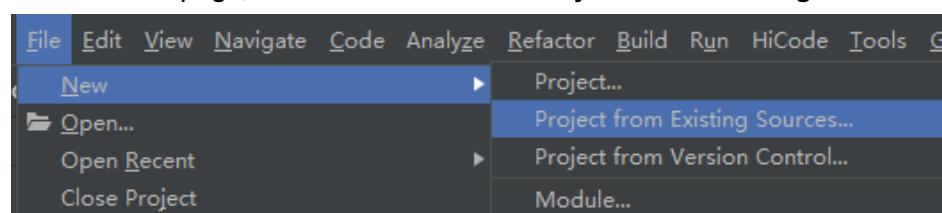
Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the cluster in the lower-right corner on the FusionInsight Manager page.

#### Scenarios

ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

#### Procedure

- Step 1** Obtain the sample project folder **clickhouse-examples** and Maven configurations in the `src` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
  1. On the IDEA page, choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-examples/ClickHouseJDBCJavaExample** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

**Step 3** After the project is imported, modify the **clickhouse-example.properties** file in the **conf** directory of the sample project based on the actual environment information.

```
loadBalancerIPList=
sslUsed=false
loadBalancerHttpPort=21425
loadBalancerHttpsPort=21426
CLICKHOUSE_SECURITY_ENABLED=true
user=
# Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or
environment variables.
password=
isMachineUser=false
isSupportMachineUser=false
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
clickhouse_dataSource_ip_list=
native_dataSource_ip_list=
```

**Table 1-7** Configuration description

Parameter	Default Value	Definition
loadBalancerIPList	-	<p>Mandatory. Set this parameter to the IP address list of LoadBalance.</p> <ul style="list-style-type: none"><li>• Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instance</b> and check the service IP addresses of all ClickHouseBalancer instances.</li><li>• Use commas (,) to separate multiple IP addresses, for example, <b>10.10.10.100,10.10.10.101</b>.</li><li>• If the IP address is an IPv6 address, convert it, for example, from <b>192:168:0:0:0:158:2</b> to <b>192:168::158:2</b>.</li></ul>

Parameter	Default Value	Definition
sslUsed	true	<p>Whether to enable SSL encryption. You are advised to set this parameter to <b>true</b> for clusters in security mode.</p> <p><b>Note:</b> If this parameter is set to <b>false</b>, non-SSL connections are used. Security clusters use SSL connections by default. If non-SSL connections are required, log in to FusionInsight Manager, choose <b>Cluster &gt; Services &gt; ClickHouse</b>, click the <b>Configuration</b> tab then the <b>All Configurations</b> sub-tab. On the page that is displayed, search for the <b>SSL_NONESSL_BOTH_ENABLE</b> configuration item, change its value to <b>true</b>, and restart all ClickHouse service instances.</p>
loadBalancerHttpPort	21425	<p>HTTP port number of LoadBalance. If <b>sslUsed</b> is set to <b>false</b>, this parameter cannot be left blank.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Port</b> in the <b>HTTP Balancer Port</b> column.</p>
loadBalancerHttpsPort	21426	<p>HTTPS port number of LoadBalance. If <b>sslUsed</b> is set to <b>true</b>, this parameter cannot be left blank.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</p>
CLICKHOUSE_SECURITY_ENABLED	true	Whether to enable the security mode for ClickHouse. This parameter has a fixed value of <b>true</b> for a cluster in security mode.
user	N/A	Developer user name prepared in <a href="#">Table 1-3</a> .
password	N/A	<p>Password of a human-machine user. Leave this parameter blank if a machine-machine user is used.</p> <p>Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.</p> <p><b>NOTE</b> If the human-machine user is created on FusionInsight Manager, you need to change the initial password upon the first login.</p>

Parameter	Default Value	Definition
isMachineUser	false	<p>Set this parameter to <b>true</b> if a machine-machine user is used for authentication.</p> <p>If this parameter is set to <b>true</b>, the values of <b>user</b> and <b>password</b> are the username and password of the machine-machine user.</p>
isSupportMachineUser	true	<p>Whether to support machine-machine user authentication. Value options are as follows:</p> <p><b>true</b>: Machine-machine user authentication is supported.</p> <p><b>false</b>: Machine-machine user authentication is not supported.</p> <p>To set this parameter, perform the following operations:</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Configuration</b> then <b>All Configurations</b>. In the navigation pane on the left, choose <b>ClickHouseServer(Role) &gt; Security</b>. On the page that is displayed, check the value of <b>SUPPORT_MACHINE_USER</b>.</p>
clusterName	default_c_luster	ClickHouse logical cluster name. Use the actual one.
databaseName	testdb	Name of the database to be created in the sample code project. You can change the database name based on the site requirements.
tableName	testtb	Name of the table to be created in the sample code project. You can change the table name based on the site requirements.
batchRows	10000	Number of data records written in a batch.
batchNum	10	Total number of batches in which data is written.

Parameter	Default Value	Definition
clickhouse_dataSource_ip_list	-	<p>Mandatory. Set this parameter to the IP address list and HTTP port number of the ClickHouseBalancer instance.</p> <ul style="list-style-type: none"> <li>• Use commas (,) to separate multiple IP addresses. The format is as follows: <i>ip:port,ip:port,ip:port</i>.</li> <li>• IP address: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instance</b> and check the service IP addresses of all ClickHouseBalancer instances.</li> <li>• Port: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</li> <li>• If the IP address is an IPv6 address, convert it to the IP address + Port number format. 192:168:0:0:0:158:2:21425 &gt; [192:168::158:2]:21425</li> </ul>
native_dataSource_ip_list	-	<p>Mandatory. Set this parameter to the IP address list and TCP port number of the ClickHouseBalancer instance.</p> <ul style="list-style-type: none"> <li>• Use commas (,) to separate multiple IP addresses. The format is as follows: <i>ip:port,ip:port,ip:port</i>.</li> <li>• IP address: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instance</b> and check the service IP addresses of all ClickHouseBalancer instances.</li> <li>• Port: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Port</b> in the <b>TCP Balancer Port</b> column.</li> <li>• If the IP address is an IPv6 address, convert it to the IP address + Port number format. 192:168:0:0:0:158:2:21424 &gt; [192:168::158:2]:21424</li> </ul>

 NOTE

ClickHouse uses the LoadBalance-based deployment architecture to automatically distribute user access traffic to multiple backend nodes, expanding service capabilities to external systems and improving fault tolerance. When a client application requests a cluster, Nginx-based ClickHouseBalancer is used to distribute traffic. In this way, data read/write load and high availability of application access are guaranteed.

- Step 4** (Optional) If machine-machine user authentication is used, download the **user.keytab** and **krb5.conf** authentication credential files of the machine-machine user by referring to [Step 7](#), and upload the files to the **conf** directory of the sample project.

----End

### 1.5.2.3 Configuring and Importing a Transaction Sample Project

#### Context

Obtain the ClickHouse development sample project and import the project to IntelliJ IDEA to learn the sample project.

#### Prerequisites

Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the cluster in the lower-right corner on the FusionInsight Manager page.

#### Scenarios

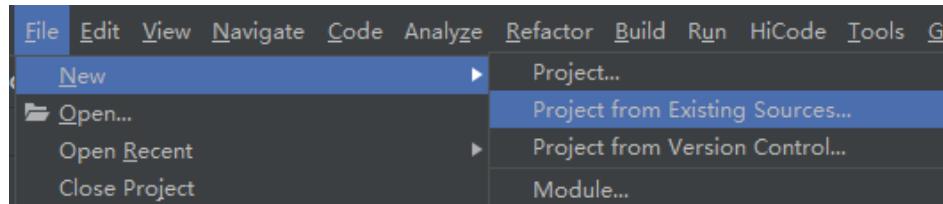
ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

#### Procedure

- Step 1** Obtain the sample project folder **clickhouse-examples** and Maven configurations in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. On the IDEA page, choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-examples/ClickHouseJDBC-Transaction-JavaExample** folder and click **OK**.

3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

**Step 3** After the project is imported, modify the **clickhouse-example.properties** file in the **conf** directory of the sample project based on the actual environment information.

```
loadBalancerIPList=
sslUsed=true
loadBalancerHttpPort=21425
loadBalancerHttpsPort=21426
CLICKHOUSE_SECURITY_ENABLED=true
user=
# Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or
environment variables.
password=
#Whether to use transactions.
useTransaction=true
#Whether to automatically submit SQL statements. The options are true (automatic) and false (manual).
Automatic submission is recommended. Due to the session forwarding mechanism of Balance, SQL
statements cannot be manually submitted by connecting to Balance.
autoCommit=true
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
```

**Table 1-8** Parameter description

Parameter	Default Value	Description
loadBalancerIPList	-	<p>Mandatory. Set this parameter to the IP address list of LoadBalance.</p> <ul style="list-style-type: none"><li>• Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instance</b> and check the service IP addresses of all ClickHouseBalancer instances.</li><li>• Use commas (,) to separate multiple IP addresses, for example, <b>10.10.10.100,10.10.10.101</b>.</li><li>• If the IP address is an IPv6 address, convert it, for example, from <b>192:168:0:0:0:158:2</b> to <b>192:168::158:2</b>.</li></ul>

Parameter	Default Value	Description
sslUsed	true	<p>Whether to enable SSL encryption. You are advised to set this parameter to <b>true</b> for clusters in security mode.</p> <p><b>Note:</b> If this parameter is set to <b>false</b>, non-SSL connections are used. Security clusters use SSL connections by default. If non-SSL connections are required, log in to FusionInsight Manager, choose <b>Cluster &gt; Services &gt; ClickHouse</b>, click the <b>Configuration</b> tab then the <b>All Configurations</b> sub-tab. On the page that is displayed, search for the <b>SSL_NONESSL_BOTH_ENABLE</b> configuration item, change its value to <b>true</b>, and restart all ClickHouse service instances.</p>
loadBalancerHttpPort	21425	<p>HTTP port number of LoadBalance. If <b>sslUsed</b> is set to <b>false</b>, this parameter cannot be left blank.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Port</b> in the <b>HTTP Balancer Port</b> column.</p>
loadBalancerHttpsPort	21426	<p>HTTPS port number of LoadBalance. If <b>sslUsed</b> is set to <b>true</b>, this parameter cannot be left blank.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</p>
CLICKHOUSE_SECURITY_ENABLED	true	Whether to enable the security mode for ClickHouse. This parameter has a fixed value of <b>true</b> for a cluster in security mode.
user	N/A	Developer user name prepared in <a href="#">Table 1-3</a> .
password	N/A	<p>Password of a human-machine user. This parameter is left blank if a machine-machine user is used.</p> <p>Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.</p> <p><b>NOTE</b> If the human-machine user is created on FusionInsight Manager, you need to change the initial password upon the first login.</p>

Parameter	Default Value	Description
useTransaction	true	Whether to use transactions. Set this parameter to <b>true</b> . If this parameter is set to <b>true</b> , SQL statements use transactions.
autoCommit	true	Whether to submit data automatically. The available values are: <b>true</b> : The transactions are automatically submitted, which is equivalent to using implicit transactions. <b>false</b> : The transactions are manually submitted.
clusterName	default_c_luster	ClickHouse logical cluster name. Use the actual one.
databaseName	testdb	Name of the database to be created in the sample code project. You can change the database name based on the site requirements.
tableName	testtb	Name of the table to be created in the sample code project. You can change the table name based on the site requirements.
batchRows	10000	Number of data records written in a batch.
batchNum	10	Total number of batches in which data is written.

#### NOTE

ClickHouse uses the LoadBalance-based deployment architecture to automatically distribute user access traffic to multiple backend nodes, expanding service capabilities to external systems and improving fault tolerance. When a client application requests a cluster, Nginx-based ClickHouseBalancer is used to distribute traffic. In this way, data read/write load and high availability of application access are guaranteed.

- Step 4** (Optional) If machine-machine user authentication is used, download the **user.keytab** and **krb5.conf** authentication credential files of the machine-machine user by referring to [Step 7](#), and upload the files to the **conf** directory of the sample project.

----End

### 1.5.2.4 Configuring and Importing the Spring Boot Sample Project

#### Scenarios

To run the Spring Boot interface sample code of the ClickHouse component of MRS, perform the following operations.

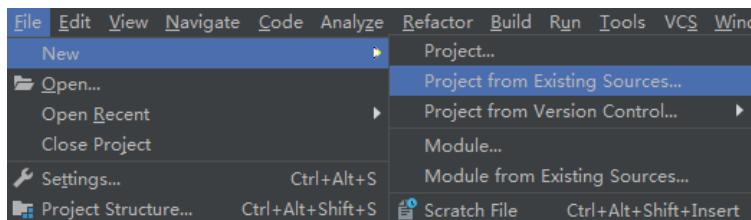
In the following example, an application is developed by compiling a Spring Boot project to connect to ClickHouse in Windows.

## Procedure

**Step 1** Obtain the sample project folder **clickhouse-rest-client-example** in the **src/springboot/clickhouse-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Choose **File > New > Project from Existing Sources....**



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-rest-client-example** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.

Select the recommended JDK version and click **Finish**.

**Step 3** In the IntelliJ IDEA development environment, open the **clickhouse-example.properties** file in the **conf** directory of the sample project, and change the values of the parameters listed in [Table 1-7](#).

**Step 4** (Optional) If machine-machine user authentication is used, download the **user.keytab** and **krb5.conf** authentication credential files of the machine-machine user by referring to [Step 7](#), and upload the files to the **conf** directory of the sample project.

----End

## 1.5.3 Application Development

### 1.5.3.1 Typical Application Scenario

This section describes the application development in a typical scenario, helping you quickly learn and master the ClickHouse development process and know key functions.

#### Scenario

ClickHouse enables you to perform common service operations by executing SQL statements. The SQL operations involved in sample codes include creating a database, creating a table, inserting data into a table, querying table data, and deleting a table.

Sample codes are described in the following sequence:

1. Setting properties
2. Establishing a connection
3. Creating a database
4. Creating a table
5. Inserting data into a table
6. Querying data
7. Deleting a table

### 1.5.3.2 Development Guideline

#### Development Guideline

As an independent DBMS system, ClickHouse allows you to use the SQL language to perform common operations. In the development program example, the clickhouse-jdbc API is used for description. The development process consists of the following parts:

- Setting properties: Sets the parameters for connecting to a ClickHouse service instance.
- Establishing a connection: Establishes a connection to the ClickHouse service instance.
- Creating a database: Creates a ClickHouse database.
- Creating a table: Creates a table in the ClickHouse database.
- Inserting data: Inserts data into a ClickHouse table.
- Querying data: Queries data in ClickHouse tables.
- Deleting a table: Deletes a ClickHouse table.

### 1.5.4 Sample Code

#### 1.5.4.1 Setting Properties

Set connection properties in the examples of creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files. The following code sets the socket timeout interval to 60 seconds:

```
Properties clickHouseProperties = new Properties();
clickHouseProperties.setProperty(ClickHouseClientOption.CONNECTION_TIMEOUT.getKey(),
Integer.toString(60000));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL.getKey(), Boolean.toString(true));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL_MODE.getKey(), "none");
```

If **sslUsed** in the **clickhouse-example.properties** configuration file in [Configuring and Importing a Sample Project](#) is set to **true**, set the following connection properties in the examples for creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files:

```
clickHouseProperties.setSsl(true);
clickHouseProperties.setSslMode("none");
```

#### 1.5.4.2 Establishing a Connection

The following code snippets are provided in the **initConnection** method of the **ClickhouseJDBCHaDemo** class. During connection creation, the user and password configured in [Table 1-7](#) are used as authentication credentials. ClickHouse performs security authentication on the server with the user and password.

```
clickHouseProperties.setProperty(ClickHouseDefaults.USER.getKey(), userName);
clickHouseProperties.setProperty(ClickHouseDefaults.PASSWORD.getKey(), userPass);
try {
    clickHouseProperties.setProperty(ClickHouseClientOption.FAILOVER.getKey(), "21");
    clickHouseProperties.setProperty(ClickHouseClientOption.LOAD_BALANCING_POLICY.getKey(),
"roundRobin");
    balancedClickhouseDataSource = new ClickHouseDataSource(JDBC_PREFIX + UriList,
clickHouseProperties);
} catch (Exception e) {
    LOG.error("Failed to create balancedClickHouseProperties.");
    throw e;
}
```

### 1.5.4.3 Creating a Database

Run the **on cluster** statement to create a database whose name is the value of **databaseName** in [Table 1-7](#) in the cluster.

```
private void createDatabase(String databaseName, String clusterName) throws Exception {
    String createDbSql = "create database if not exists " + databaseName + " on cluster " + clusterName;
    util.exeSql(createDbSql);
}
```

### 1.5.4.4 Creating a Table

Run the **on cluster** statement to create the ReplicatedMerge and Distributed tables whose names are the same as the values of **tableName** in [Table 1-7](#).

#### NOTE

If multiple ClickHouse services are installed in the cluster, for example, **clickhouse-1**, the ZooKeeper path is **/clickhouse-1/tables/{shard}/**.

```
private void createTable(String databaseName, String tableName, String clusterName) throws Exception {
    String createSql = "create table " + databaseName + "." + tableName + " on cluster " + clusterName +
" (name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/' +
databaseName + "." + tableName + "", "+ "{replica}") partition by toYYYYMM(date) order by age";
    String createDisSql = "create table " + databaseName + "." + tableName + "_all" + " on cluster " +
clusterName + " as " + databaseName + "." + tableName + " ENGINE = Distributed(default_cluster," +
databaseName + "," + tableName + ", rand());" ; ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(createSql);
    sqlList.add(createDisSql);
    util.exeSql(sqlList);
}
```

### 1.5.4.5 Inserting Data

The table created in [Creating a Table](#) has three fields of the String, UInt8, and Date types.

```
String insertSql = "insert into " + databaseName + "." + tableName + " values (?, ?, ?)";
PreparedStatement preparedStatement = connection.prepareStatement(insertSql);
long allBatchBegin = System.currentTimeMillis();
for (int j = 0; j < batchNum; j++) {
    for (int i = 0; i < batchRows; i++) {
        preparedStatement.setString(1, "huawei_" + (i + j * 10));
        preparedStatement.setInt(2, ((int) (Math.random() * 100)));
        preparedStatement.setDate(3, generateRandomDate("2018-01-01", "2021-12-31"));
        preparedStatement.addBatch();
    }
}
```

```
        }
        long begin = System.currentTimeMillis();
        preparedStatement.executeBatch();
        long end = System.currentTimeMillis();
        log.info("Inert batch time is {} ms", end - begin);
    }
long allBatchEnd = System.currentTimeMillis();
log.info("Inert all batch time is {} ms", allBatchEnd - allBatchBegin);
```

#### 1.5.4.6 Querying Data

Query statement 1 **querySql1** queries any 10 records in the **tableName** table created in [Creating a Table](#). Query statement 2 **querySql2** uses a built-in function to combine the year and month fields in the **tableName** table created in [Creating a Table](#).

```
private void queryData(String databaseName, String tableName) throws Exception {
    String querySql1 = "select * from " + databaseName + "." + tableName + "_all" + " order by age limit 10";
    String querySql2 = "select toYYYYMM(date),count(1) from " + databaseName + "." + tableName + "_all"
+ " group by toYYYYMM(date) order by count(1) DESC limit 10";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(querySql1);
    sqlList.add(querySql2);
    ArrayList<ArrayList<ArrayList<String>>> result = util.exeSql(sqlList);
    for (ArrayList<ArrayList<String>> singleResult : result) {
        for (ArrayList<String> strings : singleResult) {
            StringBuilder stringBuilder = new StringBuilder();
            for (String string : strings) {
                stringBuilder.append(string).append("\t");
            }
            log.info(stringBuilder.toString());
        }
    }
}
```

#### 1.5.4.7 Deleting a Table

Delete the replica table and distributed table created in [Creating a Table](#).

```
private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " +
clusterName;
    String dropDisTableSql = "drop table if exists " + databaseName + "." + tableName + "_all" + " on cluster "
+ clusterName;
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(dropLocalTableSql);
    sqlList.add(dropDisTableSql);
    util.exeSql(sqlList);
}
```

#### 1.5.4.8 Using BalanceDataSource

BalanceDataSource provides high-availability access connection APIs for applications. When detecting that a ClickHouseBalancer service is abnormal, applications can access the service by using other normal service instances. The following sample code provides examples of ClickHouse JDBC and Native JDBC using BalanceDataSource.

### NOTE

- ClickHouse JDBC uses HTTP to access ClickHouse, and Native JDBC uses TCP to access ClickHouse. For details about the access configurations, see the configuration description of [clickhouse.dataSource\\_ip\\_list](#) and [native.dataSource\\_ip\\_list](#) in [Configuring and Importing a Sample Project](#).
- Native JDBC does not support SSL-encrypted access.

## ClickHouse JDBC Example

Initial configuration and connection.

```
public void initConnection() throws Exception {
    Properties properties = new Properties();
    String proPath = System.getProperty("user.dir") + File.separator + "conf"
        + File.separator + "clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(proPath));
    } catch (IOException e) {
        LOG.error("Failed to load properties file.");
        throw e;
    }
    Properties clickHouseProperties = new Properties();
    boolean isSec = Boolean.parseBoolean(properties.getProperty("CLICKHOUSE_SECURITY_ENABLED"));
    String UriList = properties.getProperty("clickhouse.dataSource_ip_list");
    String userName = properties.getProperty("user");
    boolean isMachineUser = Boolean.parseBoolean(properties.getProperty("isMachineUser"));
    if (isSec) {
        if (isMachineUser) {
            clickHouseProperties.setProperty("isMachineUser", "true");
            clickHouseProperties.setProperty("keytabPath", System.getProperty("user.dir") + File.separator +
                "conf" + File.separator + "user.keytab");
        }
        String userPass = properties.getProperty("password");
        clickHouseProperties.setProperty(ClickHouseDefaults.PASSWORD.getKey(), userPass);
        clickHouseProperties.setProperty(ClickHouseClientOption.SSL.getKey(), Boolean.toString(true));
        clickHouseProperties.setProperty(ClickHouseClientOption.SSL_MODE.getKey(), "none");
    }
    clickHouseProperties.setProperty(ClickHouseDefaults.USER.getKey(), userName);
    try {
        clickHouseProperties.setProperty(ClickHouseClientOption.FAILOVER.getKey(), "21");
        clickHouseProperties.setProperty(ClickHouseClientOption.LOAD_BALANCING_POLICY.getKey(),
            "roundRobin");
        balancedClickhouseDataSource = new ClickHouseDataSource(JDBC_PREFIX + UriList,
        clickHouseProperties);
    } catch (Exception e) {
        LOG.error("Failed to create balancedClickHouseProperties.");
        throw e;
    }
}
```

Obtain the connection and send an SQL request.

```
public void queryData(String databaseName, String tableName) {
    String querySql1 = "select name,age from " + databaseName + "." + tableName + "_all" + " order by age
limit 10";
    try (ClickHouseConnection connection = balancedClickhouseDataSource.getConnection();
        ClickHouseStatement statement = connection.createStatement();
        ResultSet set = statement.executeQuery(querySql1)){
        while (set.next()) {
            LOG.info("Name is: " + set.getString(1) + ", age is: " + set.getString(2));
        }
    } catch (SQLException e) {
        //If the return code is 210, the LB server may be disconnected. Check whether the connection is
        available and try again.
        if (e.getErrorCode() == ClickHouseErrorCode.NETWORK_ERROR.code) {
            balancedClickhouseDataSource.actualize();
            try (ClickHouseConnection con = balancedClickhouseDataSource.getConnection());

```

```
        ClickHouseStatement st = con.createStatement();
        ResultSet rs = st.executeQuery(querySql1)) {
        while (rs.next()) {
            LOG.info("Name is: " + rs.getString(1) + ", age is: " + rs.getString(2));
        }
    } catch (SQLException exception) {
        LOG.error("Query data failed.");
    }
} else {
    LOG.error("Query data failed.", e);
}
}
```

## Native JDBC Example

Initial configuration and connection.

```
public void initConnection() throws IOException {
    Properties properties = new Properties();
    String proPath = System.getProperty("user.dir") + File.separator + "conf"
        + File.separator + "clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(proPath));
    } catch (IOException e) {
        LOG.error("Failed to load properties file.");
        throw e;
    }
    String UriList = properties.getProperty("native.dataSource.ip.List");
    //Whether to check connection availability during connection initialization.
    properties.put("is_check_connection", "true");
    balancedClickhouseDataSource = new BalancedClickhouseDataSource(JDBC_PREFIX + UriList, properties)
        //This method periodically checks whether the connection is available in the background and
        maintains a list of available connections. The following configuration indicates that the check is performed
        every 30 seconds.
        .scheduleActualization(30, TimeUnit.SECONDS);
}
```

Obtain the connection and send an SQL request.

```
public void queryData(String databaseName, String tableName) {
    String querySql1 = "select name,age from " + databaseName + "." + tableName + "_all" + " order by age
limit 10";
    try (ClickHouseConnection connection = balancedClickhouseDataSource.getConnection();
        ClickHouseStatement statement = connection.createStatement();
        ResultSet set = statement.executeQuery(querySql1)){
        while (set.next()) {
            LOG.info("Name is: " + set.getString(1) + ", age is: " + set.getString(2));
        }
    } catch (SQLException e) {
        //If the return code is 210, the LB server may be disconnected. Check whether the connection is
        available and try again.
        if (e.getErrorCode() == ClickHouseErrCode.NETWORK_ERROR.code()) {
            balancedClickhouseDataSource.actualize();
            try (Connection con = balancedClickhouseDataSource.getConnection();
                Statement statement = con.createStatement();
                ResultSet rs = st.executeQuery(querySql1)) {
                while (rs.next()) {
                    LOG.info("Name is: " + rs.getString(1) + ", age is: " + rs.getString(2));
                }
            } catch (SQLException exception) {
                LOG.error("Query data failed.");
            }
        } else {
            LOG.error("Query data failed.", e);
        }
    }
}
```

## 1.5.5 Connecting the Python Driver to the ClickHouse in Security Mode

### Overview

This topic describes how to use the open-source Python driver to connect to ClickHouse in security mode.

Machine-machine users and Kerberos authentication are available in ClickHouse for secure access. ClickHouse is also compatible with the human-machine user authentication of the open-source driver.

- Human-Machine Users  
You can simply use the username and password to connect to ClickHouse in security mode.
- Machine-Machine Users  
You need to adapt the Kerberos authentication to machine-machine users based on the source code of the open-source driver to connect to ClickHouse in security mode.

### Driver Adaptation

#### Driver Source Code

To support machine-machine users and be compatible with open-source capabilities, add the optional parameter **X-ClickHouse-MachineUser** to the header of the message sent from the driver source code to the ClickHouse server. The data type of the value is string.

The options are as follows:

**true**: A machine-machine user is used and needs to be authenticated.

**false** or not added: A human-machine user is used and needs to be authenticated.

#### Driver Usage

- Initialize the connection.

To use a human-machine user, use it in the same way as using the open-source driver, or set **X-ClickHouse-MachineUser** to **false**.

To use a machine-machine user, set **X-ClickHouse-MachineUser** to **true**.

- Password

For a human-machine user, use the password set when the user is created.

For a machine-machine user, parse the keytab file, obtain the file content, and pass it as the password.

### Prerequisites

- A ClickHouse user, for example, **cksuser** has been created on FusionInsight Manager. For details, see [Preparing the Development and Operating Environment](#). The authentication file of the user has been downloaded. For details, see [Step 7](#).

- If you are using an HTTP port, log in to FusionInsight Manager, choose **Cluster > Services > ClickHouse > Configurations > All Configurations > ClickHouse > Network**, and set **SSL\_NONESSL\_BOTH\_ENABLE** to **true**, save the configuration and restart the ClickHouse service to apply the changes.
- Python3 has been deployed in the running environment.
- You have prepared the ClickHouse open-source Python driver package. You can download the package from <https://github.com/ClickHouse/clickhouse-connect/archive/refs/tags/v0.6.21.zip>.
- You have prepared and installed the dependent package related to the Python driver.

You can run the following command to install the dependency package:

**pip3 install certifi lz4 pytz urllib3 zstandard**

 NOTE

- If the versions of dependency packages do not match, install the matched versions, for example:

**pip3 install urllib3==1.26.6**

- You can also manually download the dependency packages, upload them to the runtime environment of the Python driver, and install them. The dependency packages and their versions are as follows:

certifi-2020.4.5.1

lz4-3.1.3

pytz-2020.1

urllib3-1.26.6

zstandard-0.21.0

## Procedure

The following uses the **clickhouse-connect-0.6.21** driver as an example to describe how to perform source code adaptation and connect to ClickHouse in security mode through HTTP.

### Modifying Driver Source Code

**Step 1** Add the **support\_machine\_user** parameter to the initialization function of **clickhouse-connect-0.6.21\clickhouse\_connect\driver\httpclient.py**. The parameter is optional and its value is a string. The default value is **None**. During initialization, the value is saved to the **self.support\_machine\_user** variable.

```
apply_server_timezone: Optional[Union[str, bool]] = True,  
support_machine_user: Optional[str] = None):
```

```

46     # pylint: disable=too-many-arguments,too-many-locals,too-many-branches,too-many-statements,
47     def __init__(self,
48         interface: str,
49         host: str,
50         port: int,
51         username: str,
52         password: str,
53         database: str,
54         compress: Union[bool, str] = True,
55         query_limit: int = 0,
56         query_retries: int = 2,
57         connect_timeout: int = 10,
58         send_receive_timeout: int = 300,
59         client_name: Optional[str] = None,
60         verify: bool = True,
61         ca_cert: Optional[str] = None,
62         client_cert: Optional[str] = None,
63         client_cert_key: Optional[str] = None,
64         session_id: Optional[str] = None,
65         settings: Optional[Dict[str, Any]] = None,
66         pool_mgr: Optional[PoolManager] = None,
67         http_proxy: Optional[str] = None,
68         https_proxy: Optional[str] = None,
69         server_host_name: Optional[str] = None,
70         apply_server_timezone: Optional[Union[str, bool]] = True,
71         support_machine_user: Optional[str] = None):
72     """
73     Create an HTTP ClickHouse Connect client
74     See clickhouse_connect.get_client for parameters

```

```

self.support_machine_user = None
if support_machine_user:
    self.support_machine_user = support_machine_user

```

```

124
125     if session_id:
126         ch_settings['session_id'] = session_id
127     elif 'session_id' not in ch_settings and common.get_setting('autogenerate_session_id'):
128         ch_settings['session_id'] = str(uuid.uuid4())
129
130     self.support_machine_user = None
131     if support_machine_user:
132         self.support_machine_user = support_machine_user
133
134     if coerce_bool(compress):
135         compression = '.'.join(available_compression)
136         self.write_compression = available_compression[0]
137     elif compress and compress not in ('False', 'false', '0'):

```

**Step 2** In the SQL command execution function of `clickhouse-connect-0.6.21\clickhouse_connect\driver\httpclient.py`, check whether the `self.support_machine_user` parameter is empty. If this parameter is not left blank, the `X-ClickHouse-MachineUser` parameter must be contained in the message header.

```

if self.support_machine_user:
    headers['X-ClickHouse-MachineUser'] = self.support_machine_user

```

```

177     def _query_with_context(self, context: QueryContext) -> QueryResult:
178         headers = {}
179         params = {}
180         if self.database:
181             params['database'] = self.database
182         if self.protocol_version:
183             params['client_protocol_version'] = self.protocol_version
184             context.block_info = True
185         if self.support_machine_user:
186             headers['X-ClickHouse-MachineUser'] = self.support_machine_user
187         params.update(context.bind_params)
188         params.update(self._validate_settings(context.settings))
189         if columns_only_re.search(context.uncommented_query):

```

```

305     def command(self,
306         cmd,
307         parameters: Optional[Union[Sequence, Dict[str, Any]]] = None,
308         data: Union[str, bytes] = None,
309         settings: Optional[Dict] = None,
310         use_database: int = True,
311         external_data: Optional[ExternalData] = None) -> Union[str, int, Sequence[str], QuerySummary]:
312         """
313             See BaseClient doc_string for this method
314         """
315         cmd, params = bind_query(cmd, parameters, self.server_tz)
316         headers = {}
317         payload = None
318         fields = None
319         if external_data:
320             ...
321         elif isinstance(data, str):
322             ...
323         elif isinstance(data, bytes):
324             ...
325         if payload is None and not cmd:
326             raise ProgrammingError('Command sent without query or recognized data') from None
327         if payload or fields:
328             params['query'] = cmd
329         else:
330             payload = cmd
331         if use_database and self.database:
332             params['database'] = self.database
333         params.update(self._validate_settings(settings or {}))
334
335         if self.support_machine_user:
336             headers['X-ClickHouse-MachineUser'] = self.support_machine_user
337
338         method = 'POST' if payload or fields else 'GET'
339         response = self._raw_request(payload, params, headers, method, fields=fields)
340         if response.data:
341             try:
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366

```

----End

## Using the Driver

Modify the main function code of the **clickhouse-connect-0.6.21\examples\insert\_examples.py** sample to verify the use of the human-machine and machine-machine, respectively.

**Step 1** Add the driver package path to the beginning of **insert\_examples.py**.

```

import sys
sys.path.append('..')
import base64

```

**Step 2** Human-machine user: Create a client. Same as using the open-source driver, you do not need to specify the **support\_machine\_user** parameter. Set other parameters based on your environment. Verify the process of creating a connection, deleting a table, creating a table, inserting data, and querying data.

Use the following sample code to replace all code in the red box of the main function:

```

71     usage
72     def main():
73         global client # pylint: disable=global-statement
74         client = clickhouse_connect.get_client()
75         print('Nested example flatten_nested = 1 (Default)')
76         inserted_nested_flat()
77         print('\n\nNested example flatten_nested = 0')
78         insert_nested_not_flat()

```

The sample code is as follows:

```

global client # pylint: disable=global-statement
print('start test demo')

```

```
client = clickhouse_connect.get_client(host='xx.xx.xx.xx', port=xxx, username='xxx', password='xxx')
client.command('DROP TABLE IF EXISTS default.test')
client.command(
    """
    CREATE TABLE default.test
    (
        `key` UInt32,
        `value` String
    )
    ENGINE = MergeTree
    ORDER BY key
    """
)
result = client.query('insert into default.test values(1,\'test\')')
result = client.query('select * from default.test')
print(result.column_names[0:2])
print(result.result_columns[0:2])
```

### NOTE

Refer to the following to configure the parameters:

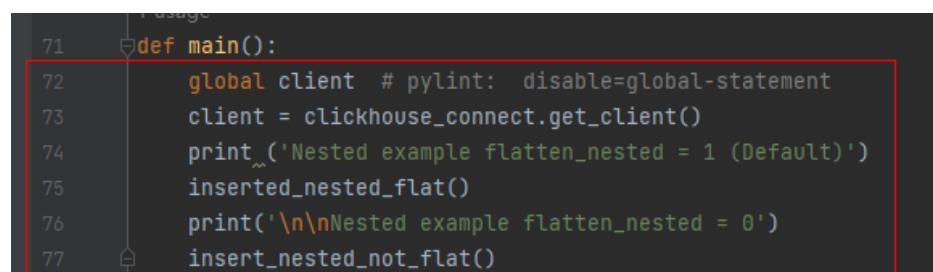
- **host**: IP address of the ClickHouse instance. To obtain the service IP address of the ClickHouseServer instance, log in to FusionInsight Manager, choose **Cluster > Services > ClickHouse**, and click the **Instances** tab.
- **port**: HTTP port of the ClickHouse server. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > ClickHouse > Configurations > All Configurations > ClickHouseServer > Network**, and check the value of **http\_port**.
- **username**: human-machine username created on FusionInsight Manager
- **password**: password set when the human-machine user is created on FusionInsight Manager

If the sample code contains authentication passwords, there are security risks. Delete the information about the passwords after using the code or store them securely.

### Step 3 Machine-machine user:

1. Parse the keytab file and obtain the content.
2. When the client is initialized, the **support\_machine\_user** parameter is passed and set to **true**. The user password is the file content parsed in step 1.
3. Verify the process of creating a connection, deleting a table, creating a table, inserting data, and querying data.

Use the following sample code to replace all code in the red box of the main function:



```
71 def main():
72     global client # pylint: disable=global-statement
73     client = clickhouse_connect.get_client()
74     print('Nested example flatten_nested = 1 (Default)')
75     inserted_nested_flat()
76     print('\n\nNested example flatten_nested = 0')
77     insert_nested_not_flat()
```

The sample code is as follows:

```
global client # pylint: disable=global-statement
print('start test demo')
file = open('user.keytab', 'rb')
byte_array = file.read()
b64_str = base64.b64encode(byte_array)
b64_str = b64_str.decode('utf8')

client = clickhouse_connect.get_client(host='xx.xx.xx.xx', port=xxx,username='xxx', password=b64_str,
support_machine_user='true')
```

```
client.command('DROP TABLE IF EXISTS default.test')
client.command(
    """
CREATE TABLE default.test
(
    `key` UInt32,
    `value` String
)
ENGINE = MergeTree
ORDER BY key
""")
result = client.query('insert into default.test values(1,\'test\')')
result = client.query('select * from default.test')
print(result.column_names[0:2])
print(result.result_columns[0:2])
```

#### NOTE

- Parse the keytab file. **user.keytab** is the keytab file of the user downloaded from FusionInsight Manager. Replace it with the one you need.
- Create a connection and configure the parameters by referring the following:
  - **host**: IP address of the ClickHouse instance. To obtain the service IP address of the ClickHouseServer instance, log in to FusionInsight Manager, choose **Cluster > Services > ClickHouse**, and click the **Instances** tab.
  - **port**: HTTP port of the ClickHouse server. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > ClickHouse > Configurations > All Configurations > ClickHouseServer > Network**, and check the value of **http\_port**.
  - **username**: machine-machine username created on FusionInsight Manager
  - **password**: variable for storing the keytab file content
  - **support\_machine\_user**: whether the user is a machine-machine user. The value is **true**.

----End

## Commissioning

**Step 1** Upload the modified driver code on the server, for example, to the **/opt/tmp** directory.

**Step 2** For a machine-machine user, upload the keytab file downloaded from to the **/opt/tmp/clickhouse-connect-0.6.21/examples** directory.

**Step 3** Run the following command to go to the example path:

```
cd /opt/tmp/clickhouse-connect-0.6.21/examples
```

**Step 4** Run the following command to execute the sample code:

```
python3 insert_examples.py
```

The result is as follows.

```
[root@XXXXXX examples]# python3 insert_examples.py
Unable to connect optimized C data functions [No module named 'clickhouse_connect.driverc.buffer'], falling back to pure Python
start test demo
('key', 'value')
[[1], ['test']]
```

----End

## 1.5.6 Application Commissioning

### 1.5.6.1 Commissioning Applications on Windows

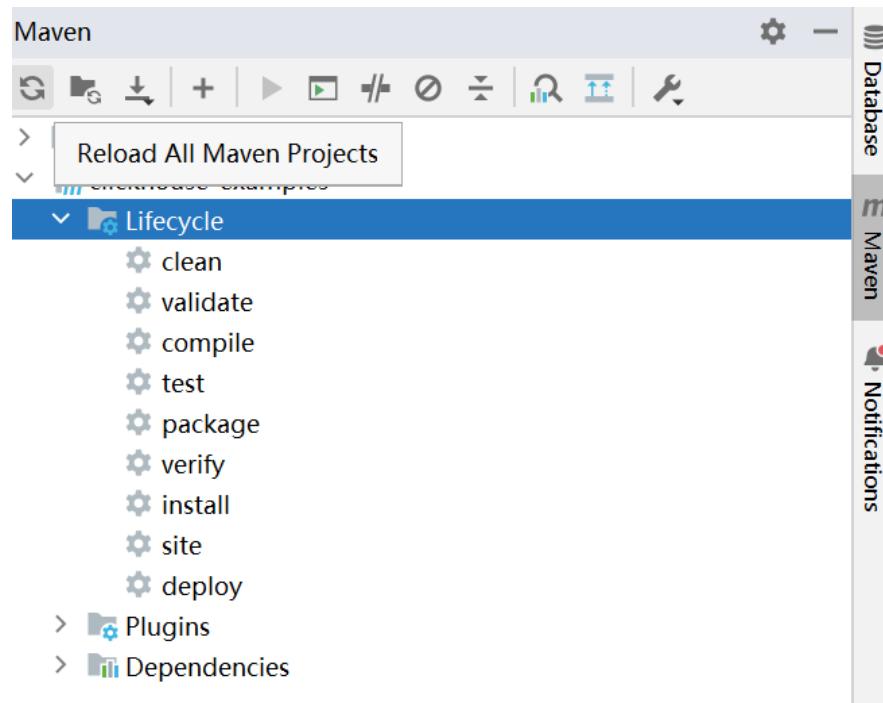
#### Writing and Running An Application

You can run an application in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

#### Procedure

- Step 1** Click **Reload All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

**Figure 1-4** reload projects

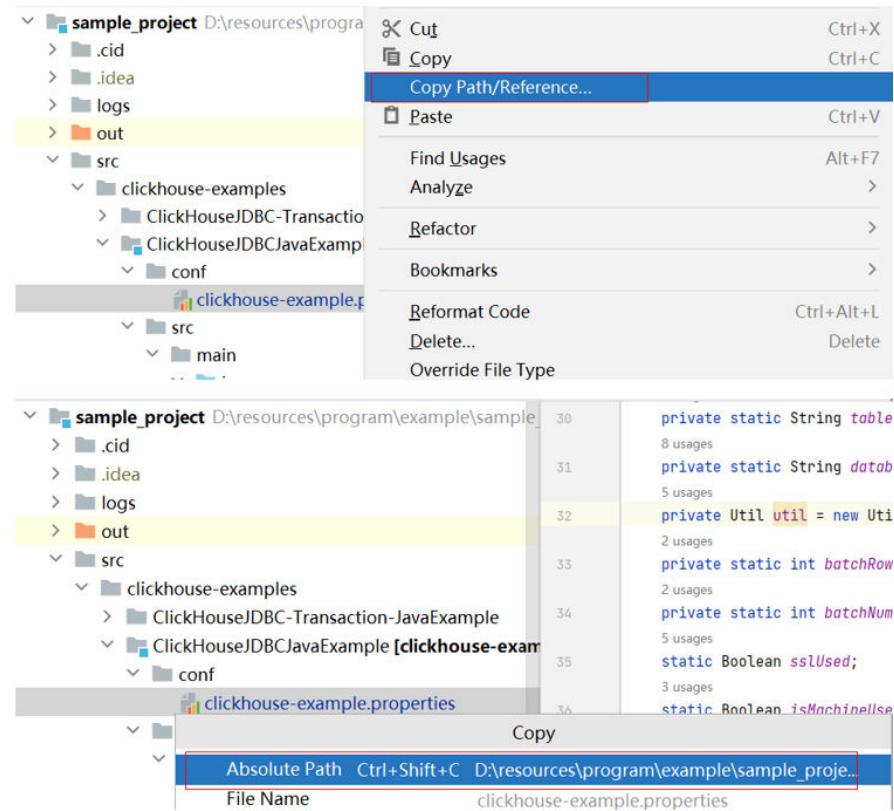


- Step 2** Copy the **clickhouse-example.properties** path on the IDEA page. Right-click **clickhouse-example.properties** and choose **Copy Path/Reference > Absolute Path** from the shortcut menu.

**NOTE**

Skip this step if you are commissioning the transaction sample project.

**Figure 1-5** Copying absolute path of the configuration file



**Step 3** In **Demo.java**, replace the path of **proPath** in the **getProperties()** method with **clickhouse-example.properties**.

**Figure 1-6** Replacing the path

```
private void getProperties() throws Exception {
    Properties properties = new Properties();
    String proPath = "D:\\resources\\program\\example\\sample_project\\src\\clickhouse-examples\\ClickHouseJDBCJavaExample\\conf\\clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(new File(proPath)));
    } catch (IOException e) {
        log.error("Failed to load properties file.");
        throw e;
    }
}
```

**Step 4** In **ClickhouseJDBCHaDemo.java**, replace the path of **proPath** in the **initConnection()** method with **clickhouse-example.properties**.

#### NOTE

Skip this step if you are commissioning the transaction sample project.

**Figure 1-7** Replacing the path

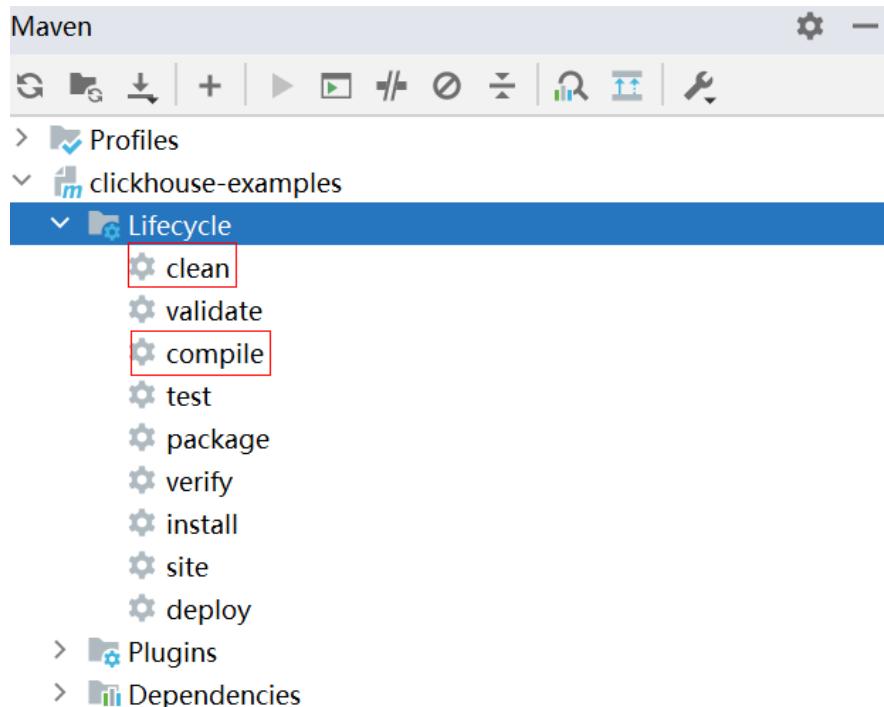
```
public void initConnection() throws Exception {
    Properties properties = new Properties();
    String proPath = "D:\\resources\\program\\example\\sample_project\\src\\clickhouse-examples\\ClickHouseJDBCJavaExample\\conf\\clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(proPath));
    } catch (IOException e) {
        LOG.error("Failed to load properties file.");
        throw e;
    }
}
```

**Step 5** Compile the application.

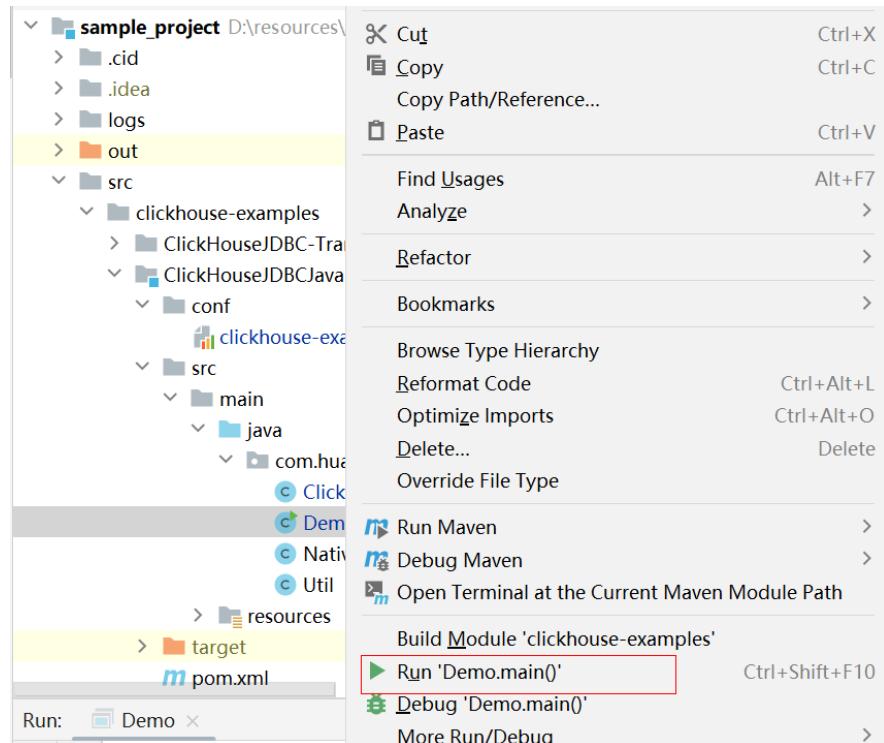
- Select **Maven > clickhouse-examples > Lifecycle > clean** and double-click **clean** to run the **clean** command of Maven.

- Select **Maven > clickhouse-examples > Lifecycle > compile** and double-click **compile** to run the **compile** command of Maven.

**Figure 1-8** Maven commands clean and compile



**Step 6** Click `Run'Demo.main()'` to run the application.

**Figure 1-9** Running the program

----End

## Viewing the Commissioning Result

After a ClickHouse application finishes running, you can use one of the following methods to view the running result:

- Viewing the command output
- View the **clickhouse-example.log** file in the **logs** directory to obtain the application running status.

After the complete sample of the **clickhouse-examples** finishes running, the following information is displayed on the console:

```
2023-09-19 16:20:48,344 | INFO | main | loadBalancerIPList is 192.168.5.132, loadBalancerHttpPort is 21422, user is ck_user, clusterName is default_cluster, isSec is true, password is xxx. | com.huawei.clickhouse.examples.Demo.main(Demo.java:42)
2023-09-19 16:20:48,350 | INFO | main | ckLbServerList current member is 0, ClickhouseBalancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Demo.getCKLbServerList(Demo.java:110)
2023-09-19 16:20:48,436 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:50,781 | INFO | main | Execute query:drop table if exists testdb.testtb on cluster default_cluster no delay | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:51,504 | INFO | main | Execute time is 723 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:51,511 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:51,897 | INFO | main | Execute query:drop table if exists testdb.testtb_all on cluster default_cluster no delay | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:52,421 | INFO | main | Execute time is 524 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:52,422 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:52,946 | INFO | main | Execute query:create database if not exists testdb on cluster default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
```

```
2023-09-19 16:20:53,405 | INFO | main | Execute time is 458 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:53,406 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:53,757 | INFO | main | Execute query:create table testdb.testtb on cluster default_cluster
(name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/'
testdb.testtb','{replica}') partition by toYYYYMM(date) order by age |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:54,243 | INFO | main | Execute time is 485 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:54,244 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:54,640 | INFO | main | Execute query:create table testdb.testtb_all on cluster
default_cluster as testdb.testtb ENGINE = Distributed(default_cluster,testdb,testtb, rand()); |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:55,175 | INFO | main | Execute time is 535 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:55,175 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.insertData(Util.java:143)
2023-09-19 16:20:58,868 | INFO | main | Insert batch time is 503 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:160)
2023-09-19 16:21:01,015 | INFO | main | Insert batch time is 631 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:160)
2023-09-19 16:21:02,521 | INFO | main | Inert all batch time is 4163 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:164)
2023-09-19 16:21:02,522 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:21:03,051 | INFO | main | Execute query:select * from testdb.testtb_all order by age limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:21:03,430 | INFO | main | Execute time is 379 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:21:03,433 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:21:03,760 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
testdb.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:21:04,361 | INFO | main | Execute time is 600 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:21:04,362 | INFO | main | name age date |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,362 | INFO | main | huawei_9 12 2021-04-20 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,362 | INFO | main | huawei_17 15 2021-05-23 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_5 24 2021-04-15 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_13 39 2020-07-04 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_3 49 2021-06-27 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_15 50 2020-06-26 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_11 53 2020-08-14 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_12 56 2021-12-19 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_19 57 2021-10-31 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_0 57 2020-03-01 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202105 3 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202110 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202104 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202008 2 |
```

```
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202007 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202106 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202012 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202109 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202003 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,365 | INFO | main | 202011 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:05,044 | INFO | main | Name is: huawei_9, age is: 12 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,044 | INFO | main | Name is: huawei_17, age is: 15 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_5, age is: 24 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_13, age is: 39 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_3, age is: 49 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_15, age is: 50 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_11, age is: 53 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_12, age is: 56 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_19, age is: 57 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,046 | INFO | main | Name is: huawei_0, age is: 57 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
```

Process finished with exit code 0

### 1.5.6.2 Commissioning Applications on Linux

ClickHouse applications can run in a Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

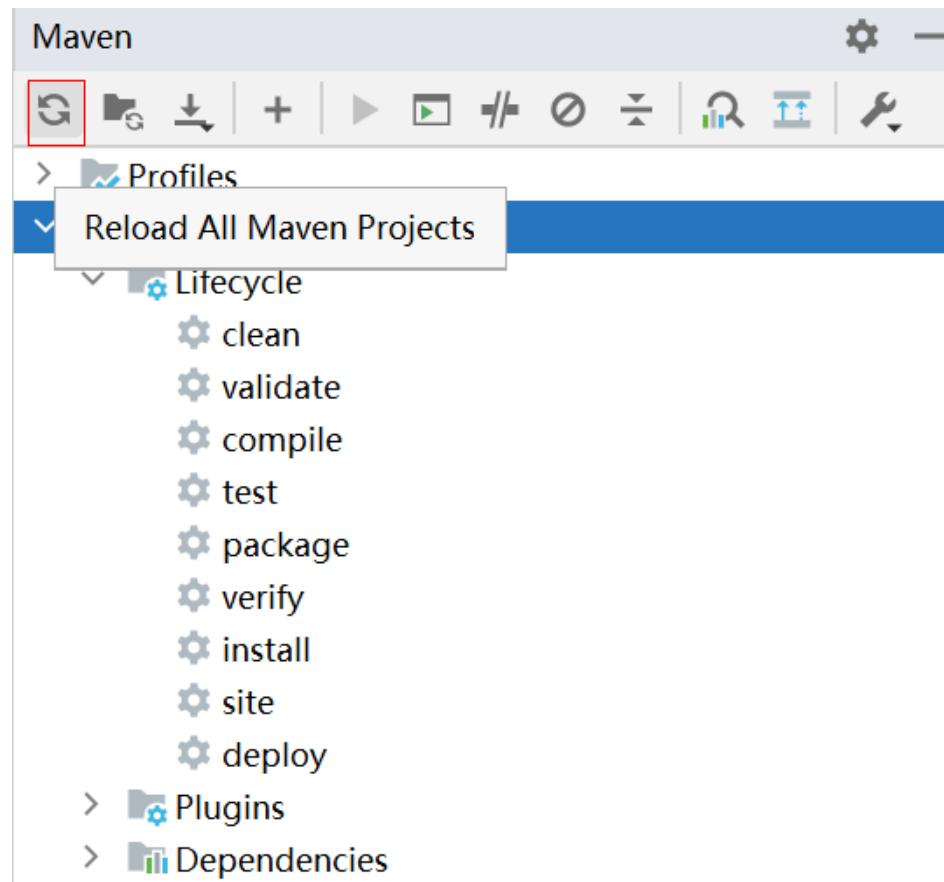
#### Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

#### Compile and run applications.

**Step 1** Click **Reload All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

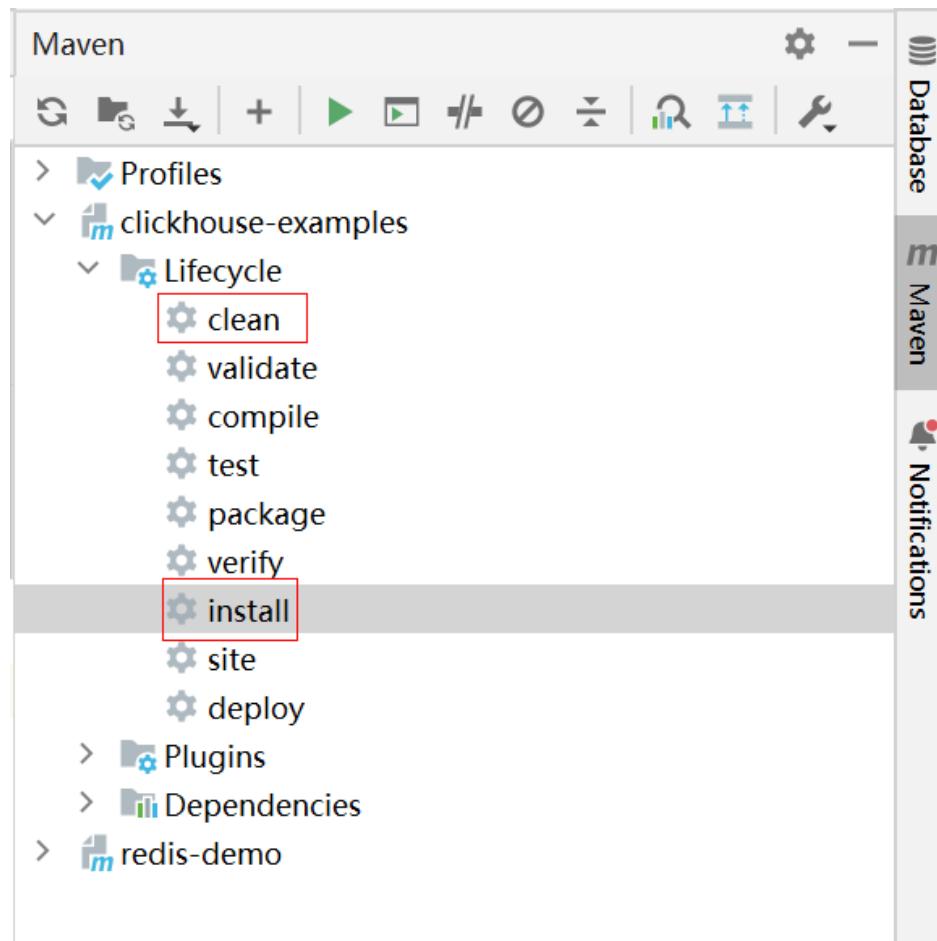
Figure 1-10 reload projects



**Step 2** Exporting the JAR file

- Select **Maven > clickhouse-examples > Lifecycle > clean** and double-click **clean** to run the **clean** command of Maven.
- Select **Maven > clickhouse-examples > Lifecycle > install** and double-click **install** to run the **install** command of Maven.

Figure 1-11 Maven clean and install commands



- Step 3** Copy the **clickhouse-examples-\*jar** file from the **target** directory and the **conf** folder from the **clickhouse-examples** directory to the ClickHouse client installation directory, for example, *Client installation directory/JDBC* or *Client installation directory/JDBCTransaction*.

NOTE

The *Client installation directory/JDBC* directory is used to commission JDBC secondary samples.

The *Client installation directory/JDBCTransaction* directory is used to commission secondary transaction samples.

- Step 4** Log in to the client node, go to the directory where the JAR file is uploaded, and change the file permission to 700.

**cd** *Client installation directory/JDBC*

Or **cd** *Client installation directory/JDBCTransaction*

**chmod 700 clickhouse-examples-\*jar**

- Step 5** In the client directory where **clickhouse-examples-\*jar** is stored, run the following commands to run the JAR file:

**source** *Client installation directory/bigdata\_env*

**cd** *Client installation directory/JDBC*

Or **cd Client installation directory/JDBCTransaction**

**java -jar clickhouse-examples-\*.jar**

----End

## Viewing Commissioning Results

After a ClickHouse application is run, you can use one of the following methods to view the running result:

- Viewing the command output
- Viewing ClickHouse logs

View the **logs clickhouse-example.log** file in the directory where the current JAR file is located, for example, *Client installation directory/JDBC/logs/ clickhouse-example.log* or *Client installation directory/JDBCTransaction/ logs/ clickhouse-example.log*.

The execution result of the JAR file is as follows:

```
2023-09-21 09:08:38,944 | INFO | main | loadBalancerIPList is 192.168.5.132, loadBalancerHttpPort is 21422, user is ck_user, clusterName is default_cluster, isSec is true, password is xxx. | com.huawei.clickhouse.examples.Demo.main(Demo.java:42)
2023-09-21 09:08:38,949 | INFO | main | ckLbServerList current member is 0, ClickhouseBalancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Demo.getCKLBServerList(Demo.java:111)
2023-09-21 09:08:38,982 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-21 09:08:39,356 | INFO | main | Execute query:drop table if exists testdb.testtb on cluster default_cluster no delay | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-21 09:08:39,537 | INFO | main | Execute time is 181 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-21 09:08:39,545 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-21 09:08:39,580 | INFO | main | Execute query:drop table if exists testdb.testtb_all on cluster default_cluster no delay | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-21 09:08:39,717 | INFO | main | Execute time is 136 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-21 09:08:39,718 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-21 09:08:39,752 | INFO | main | Execute query:create database if not exists testdb on cluster default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-21 09:08:39,887 | INFO | main | Execute time is 134 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-21 09:08:39,888 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-21 09:08:39,920 | INFO | main | Execute query:create table testdb.testtb on cluster default_cluster (name String, age UInt8, date Date)ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/testdb.testtb','{replica}') partition by toYYYYMM(date) order by age | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-21 09:08:40,064 | INFO | main | Execute time is 143 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-21 09:08:40,065 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-21 09:08:40,096 | INFO | main | Execute query:create table testdb.testtb_all on cluster default_cluster as testdb.testtb ENGINE = Distributed(default_cluster,testdb,testtb, rand()); | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-21 09:08:40,231 | INFO | main | Execute time is 134 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-21 09:08:40,232 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.insertData(Util.java:143)
2023-09-21 09:08:40,447 | INFO | main | Insert batch time is 101 ms | com.huawei.clickhouse.examples.Util.insertData(Util.java:160)
2023-09-21 09:08:42,063 | INFO | main | Insert batch time is 113 ms | com.huawei.clickhouse.examples.Util.insertData(Util.java:160)
2023-09-21 09:08:43,564 | INFO | main | Inert all batch time is 3224 ms |
```

```
com.huawei.clickhouse.examples.Util.insertData(Util.java:164)
2023-09-21 09:08:43,564 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-21 09:08:43,596 | INFO | main | Execute query:select * from testdb.testtb_all order by age limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-21 09:08:43,619 | INFO | main | Execute time is 23 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-21 09:08:43,620 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-21 09:08:43,652 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
testdb.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-21 09:08:43,675 | INFO | main | Execute time is 23 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-21 09:08:43,677 | INFO | main | name age date |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,677 | INFO | main | huawei_7 4 2020-06-26 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,677 | INFO | main | huawei_1 10 2021-01-05 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,677 | INFO | main | huawei_12 17 2021-04-02 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,677 | INFO | main | huawei_6 21 2021-12-02 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,678 | INFO | main | huawei_8 22 2021-01-04 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,678 | INFO | main | huawei_4 26 2020-05-14 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,678 | INFO | main | huawei_11 30 2021-10-06 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,678 | INFO | main | huawei_18 32 2020-12-15 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,678 | INFO | main | huawei_0 33 2020-06-23 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,678 | INFO | main | huawei_15 35 2020-07-13 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202006 4 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202110 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202101 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202104 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202001 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202005 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202012 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,679 | INFO | main | 202108 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,680 | INFO | main | 202107 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,680 | INFO | main | 202102 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:159)
2023-09-21 09:08:43,732 | INFO | main | Name is: huawei_7, age is: 4 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,732 | INFO | main | Name is: huawei_1, age is: 10 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,732 | INFO | main | Name is: huawei_12, age is: 17 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,733 | INFO | main | Name is: huawei_6, age is: 21 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,733 | INFO | main | Name is: huawei_8, age is: 22 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,733 | INFO | main | Name is: huawei_4, age is: 26 |
```

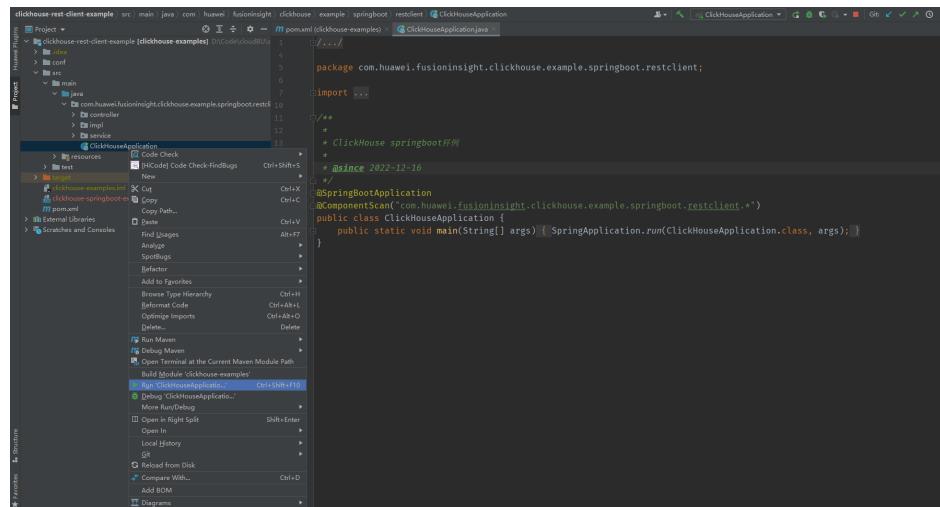
```
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,733 | INFO  | main | Name is: huawei_11, age is: 30 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,734 | INFO  | main | Name is: huawei_18, age is: 32 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,734 | INFO  | main | Name is: huawei_0, age is: 33 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
2023-09-21 09:08:43,734 | INFO  | main | Name is: huawei_15, age is: 35 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:79)
```

### 1.5.6.3 Commissioning the Spring Boot Sample Project

### 1.5.6.3.1 Commissioning the Spring Boot Project in Windows

# Compiling and Running Applications

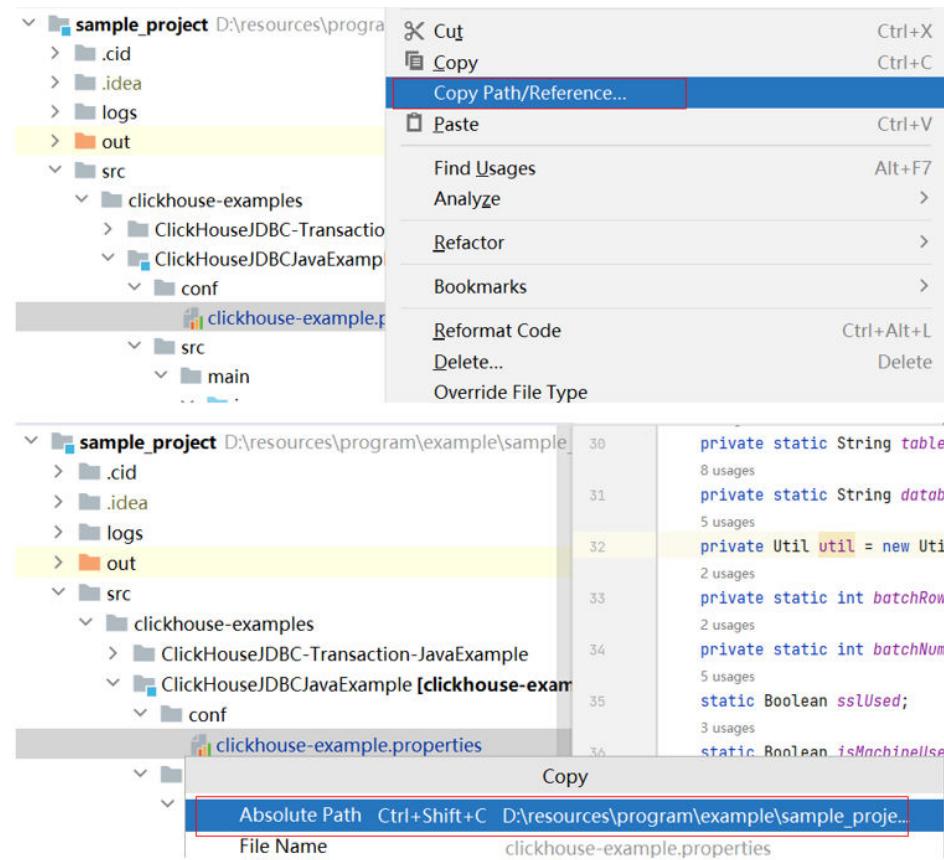
You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host. Right-click **ClickHouseApplication** in IntelliJ IDEA project **clickhouse-rest-client-examples** in the development environment, and then choose **Run ClickHouseApplication** from the shortcut menu to run the application project.



## Procedure

- Step 1** Copy the `clickhouse-example.properties` path on the IDEA page. Right-click `clickhouse-example.properties` and choose **Copy Path/Reference > Absolute Path** from the shortcut menu.

**Figure 1-12** Copying absolute path of the configuration file



**Step 2** In `ClickHouseFunc.java`, replace the path of `proPath` in the `getProperties()` method with `clickhouse-example.properties`.

**Figure 1-13** Replacing the path



----End

## Viewing Commissioning Results

- After the ClickHouse Spring Boot service is started, the ClickHouse sample interface is used to trigger the running of the sample code. Enter the link of the specific operation to be performed in the address box of the browser, for example, <http://localhost:8080/clickhouse/executeQuery>. The following result is returned:  
ClickHouse springboot client runs normally.
- View the `clickhouse-springboot-example.log` file to obtain the application running status.

After the `clickhouse-springboot` sample is run, the following information is displayed on the console:

```
.\clickhouse-springboot-example
((()__))|_|_|_|`V`|\\\\\\
```

```
\W ____| |_)| | | | | |(| | ) ) )
' |____| .|_| | \_ | \_, | | | |
=====|_|=====|_|/_=/_/_/
:: Spring Boot ::           (v2.7.0)

2023-02-06 17:58:22.144 INFO 20556 --- [      main] c.h.f.c.e.s.r.ClickHouseApplication   :
Starting ClickHouseApplication using Java 1.8.0_181 on DESKTOP-64PQBSD with PID 20556 (D:\Code
\cloudBU\sample_project\src\springboot\clickhouse-examples\clickhouse-rest-client-example\target
\classes started by l00467039 in D:\Code\cloudBU\sample_project\src\springboot\clickhouse-examples
\clickhouse-rest-client-example)
2023-02-06 17:58:22.146 INFO 20556 --- [      main] c.h.f.c.e.s.r.ClickHouseApplication   : No
active profile set, falling back to 1 default profile: "default"
2023-02-06 17:58:22.815 INFO 20556 --- [      main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-02-06 17:58:22.820 INFO 20556 --- [      main] o.apache.catalina.core.StandardService  :
Starting service [Tomcat]
2023-02-06 17:58:22.821 INFO 20556 --- [      main] org.apache.catalina.core.StandardEngine :
Starting Servlet engine: [Apache Tomcat/9.0.63]
2023-02-06 17:58:22.928 INFO 20556 --- [      main] o.a.c.c.C.[Tomcat].[localhost]. [/]   :
Initializing Spring embedded WebApplicationContext
2023-02-06 17:58:22.928 INFO 20556 --- [      main] w.s.c.ServletWebServerApplicationContext :
Root WebApplicationContext: initialization completed in 750 ms
2023-02-06 17:58:23.175 INFO 20556 --- [      main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context
path "
2023-02-06 17:58:23.181 INFO 20556 --- [      main] c.h.f.c.e.s.r.ClickHouseApplication   : Started
ClickHouseApplication in 1.388 seconds (JVM running for 3.989)
2023-02-06 17:58:32.640 INFO 20556 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost]. [/]   :
Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-02-06 17:58:32.640 INFO 20556 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet   :
Initializing Servlet 'dispatcherServlet'
2023-02-06 17:58:32.641 INFO 20556 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet   :
Completed initialization in 1 ms
2023-02-06 17:58:32.656 INFO 20556 --- [nio-8080-exec-1] .f.c.e.s.r.c.ClickHouseExampleController :
Begin to execute query in clickhouse...
2023-02-06 17:58:32.658 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc   :
loadBalancerIPList is 192.168.6.24, loadBalancerHttpPort is 21422, user is ckuser, clusterName is
default_cluster, isSec is true.
2023-02-06 17:58:32.659 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc   :
ckLBServerList current member is 0, ClickhouseBalancer is 192.168.6.24:21422
2023-02-06 17:58:32.664 INFO 20556 --- [nio-8080-exec-1] ru.yandex.clickhouse.ClickHouseDriver   :
Driver registered
2023-02-06 17:58:32.665 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:35.494 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   :
Execute sql drop table if exists testdb.testtb on cluster default_cluster no delay, time is 216 ms
2023-02-06 17:58:35.495 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:36.074 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   :
Execute sql drop table if exists testdb.testtb_all on cluster default_cluster no delay, time is 196 ms
2023-02-06 17:58:36.074 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:36.765 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   :
Execute sql create database if not exists testdb on cluster default_cluster, time is 218 ms
2023-02-06 17:58:36.765 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:37.364 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   :
Execute sql create table testdb.testtb on cluster default_cluster (name String, age UInt8, date
Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/testdb.testtb','{replica}') partition by
toYYYYMM(date) order by age, time is 198 ms
2023-02-06 17:58:37.366 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:37.872 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   :
Execute sql create table testdb.testtb_all on cluster default_cluster as testdb.testtb ENGINE =
Distributed(default_cluster,testdb,testtb, rand()), time is 160 ms
2023-02-06 17:58:38.959 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   : Insert
batch time is 591 ms
2023-02-06 17:58:41.114 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util   : Insert
batch time is 572 ms
```

```

2023-02-06 17:58:43.095 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 413 ms
2023-02-06 17:58:45.220 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 543 ms
2023-02-06 17:58:47.207 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 406 ms
2023-02-06 17:58:49.236 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 460 ms
2023-02-06 17:58:51.197 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 390 ms
2023-02-06 17:58:53.233 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 466 ms
2023-02-06 17:58:55.355 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 555 ms
2023-02-06 17:58:57.321 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
batch time is 386 ms
2023-02-06 17:58:58.832 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Insert
all batch time is 20564 ms
2023-02-06 17:58:58.832 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:59.256 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util :
Execute sql select * from testdb.testtb_all order by age limit 10, time is 119 ms
2023-02-06 17:58:59.257 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:59.972 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util :
Execute sql select toYYYYMM(date),count(1) from testdb.testtb_all group by toYYYYMM(date) order
by count(1) DESC limit 10, time is 289 ms
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_234 0 2022-12-16
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_1805 0 2022-12-21
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_3359 0 2022-12-13
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_4275 0 2022-12-09
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_4307 0 2022-12-20
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_4586 0 2022-12-02
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_6326 0 2022-12-18
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_9878 0 2022-12-06
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_2482 0 2022-12-18
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
huawei_2904 0 2022-12-25
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202201 8628
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202208 8587
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202205 8558
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202210 8547
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202207 8463
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202203 8379
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202204 8351
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202206 8300
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202209 8297
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :
202212 8228

```

### 1.5.6.3.2 Commissioning the Spring Boot Project in Linux

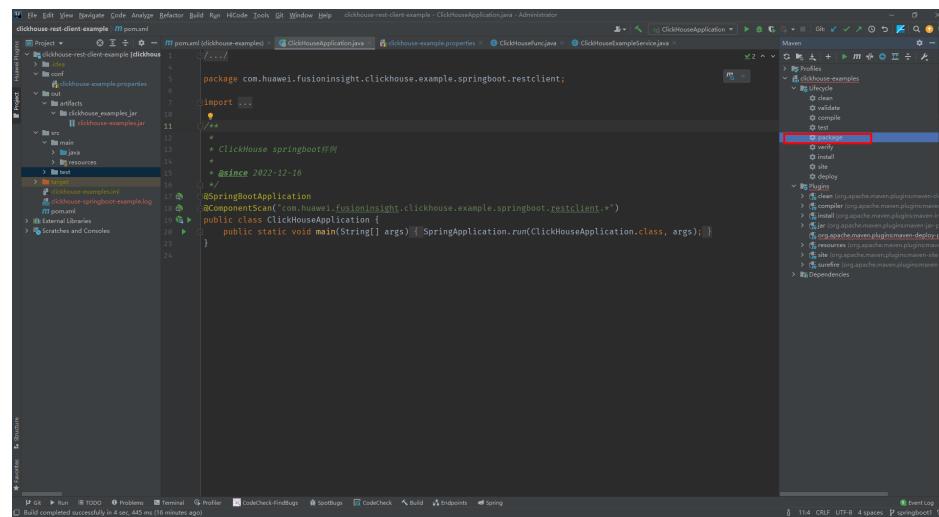
The ClickHouse Spring Boot applications can run in a Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

#### Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

### Compiling and Running Applications

- Step 1** Click **Maven** on the right of IDEA, expand **Lifecycle**, and double-click **package** to package the current project.



- Step 2** Log in to the ClickHouse client node as user **root**, create a running directory, for example, **/opt/test**, and obtain the JAR package whose name contains **-with-dependencies** from the **target** directory of IDEA. Upload the JAR package and the **conf** folder in the IDEA to the **/opt/test** directory, as shown in the following figure.

```
[root@host-192-168-6-24 test]# 
[root@host-192-168-6-24 test]# pwd
/opt/test
[root@host-192-168-6-24 test]# ll
total 25048
-rw-r--r-- 1 root root 25644916 Feb  6 19:24 clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
drwxr-xr-x 2 root root     4096 Feb  6 18:03 conf
[root@host-192-168-6-24 test]# 
```

- Step 3** Run the following commands to set environment variables and run the JAR package:

**cd** *Client installation path*

**source** *bigdata\_env*

**cd** */opt/test*

**java -jar** *clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar*

The command output is displayed as follows:



```
[root@host-192-168-6-24 test]# java -jar clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
2023-02-06 19:42:02.200 INFO 42496 --- [main] c.h.f.c.e.s.r.ClickHouseApplication : Starting ClickHouseApplication using Java 1.8.0_332 on host-192-168-6-24 with PID 24296 [/opt/test	clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar started by root in /opt/test]
2023-02-06 19:42:02.203 INFO 42496 --- [main] c.h.f.c.e.s.r.ClickHouseApplication : No active profile set, falling back to 1 default profile: "default"
2023-02-06 19:42:03.396 INFO 42496 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-02-06 19:42:03.400 INFO 42496 --- [main] o.s.b.w.embedded.tomcat.StandardService : Starting service [Tomcat]
2023-02-06 19:42:03.410 INFO 42496 --- [main] o.s.b.w.embedded.tomcat.StandardService : Starting service [Tomcat]
2023-02-06 19:42:03.476 INFO 42496 --- [main] o.s.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring Embedded WebApplicationContext
2023-02-06 19:42:04.074 INFO 42496 --- [main] o.s.c.c.C.[Tomcat].[localhost].[/] : Root WebApplicationContext: initialization completed in 1188 ms
2023-02-06 19:42:04.074 INFO 42496 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-02-06 19:42:04.079 INFO 42496 --- [main] c.h.f.c.e.s.r.ClickHouseApplication : Started ClickHouseApplication in 2.718 seconds (JVM running for 3.07)
```

#### Step 4 Call the Spring Boot sample API of ClickHouse to trigger running the sample code.

- Running method in Windows:

Open the browser, enter **http://IP address of the ClickHouse client node:8080/clickhouse/executeQuery** in the address box, and check the returned information.

ClickHouse springboot client runs normally.

- Running method in Linux:

Log in to the ClickHouse client node and run the following command to view shell logs and log files in Linux:

**curl http://localhost:8080/clickhouse/executeQuery**  
**vi clickhouse-springboot-example.log**

```
[root@host-192-168-6-24 test]# ll
total 25660
drwxr-xr-x 1 root root 25644916 Feb 6 19:24 clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
drwxr-xr-x 1 root root 9384 Feb 6 19:42 clickhouse-springboot-example.log
drwxr-xr-x 2 root root 4096 Feb 6 18:03 root
[root@host-192-168-6-24 test]# vi clickhouse-springboot-example.log
2023-02-06 19:42:02.200 [main] INFO 42496 --- [main] c.h.f.c.e.s.r.ClickHouseApplication : Starting ClickHouseApplication using Java 1.8.0_332 on host-192-168-6-24 with PID 24296 [/opt/test	clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar started by root in /opt/test]
2023-02-06 19:42:02.203 [main] INFO 42496 --- [main] c.h.f.c.e.s.r.ClickHouseApplication : No active profile set, falling back to 1 default profile: "default"
2023-02-06 19:42:03.396 [main] INFO 42496 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-02-06 19:42:03.400 [main] INFO 42496 --- [main] o.s.b.w.embedded.tomcat.StandardService : Starting service [Tomcat]
2023-02-06 19:42:03.410 [main] INFO 42496 --- [main] o.s.b.w.embedded.tomcat.StandardService : Starting service [Tomcat]
2023-02-06 19:42:03.476 [main] INFO 42496 --- [main] o.s.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring Embedded WebApplicationContext
2023-02-06 19:42:04.074 [main] INFO 42496 --- [main] o.s.c.c.C.[Tomcat].[localhost].[/] : Root WebApplicationContext: initialization completed in 1188 ms
2023-02-06 19:42:04.074 [main] INFO 42496 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-02-06 19:42:04.079 [main] INFO 42496 --- [main] c.h.f.c.e.s.r.ClickHouseApplication : Started ClickHouseApplication in 2.718 seconds (JVM running for 3.07)
2023-02-06 19:43:53.323 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing DispatcherServlet 'dispatcherServlet'
2023-02-06 19:43:53.326 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
2023-02-06 19:43:53.376 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] f.c.e.s.r.c.ClickHouseExampleController : Begin to execute query in clickhouse...
2023-02-06 19:43:53.389 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc : loadBalancerPList is [http://192.168.6.24:8080], loadBalancerPort is 8080
2023-02-06 19:43:53.389 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc : ckLbServerList current member is 0, ClickhouseBalancer is [http://192.168.6.24:8080]
2023-02-06 19:43:53.391 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] ru.yandex.clickhouse.ClickHouseDriver : Driver registered
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Try times is 0, current load balancer is [http://192.168.6.24:8080]
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Execute sql drop table if exists testdb.testtb on cluster default
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Load balancer is [http://192.168.6.24:8080]
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Try times is 0, current load balancer is [http://192.168.6.24:8080]
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Execute sql drop table if exists testdb.testtb_all on cluster default
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Try times is 0, current load balancer is [http://192.168.6.24:8080]
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Execute sql create database if not exists testdb on cluster default
2023-02-06 19:43:53.392 [http-nio-8080-exec-1] INFO 42496 --- [http-nio-8080-exec-1] c.h.f.c.e.s.r.restclient.impl.Util : Try times is 0, current load balancer is [http://192.168.6.24:8080]
```

----End

## 1.6 Doris Development Guide

### 1.6.1 Overview

#### 1.6.1.1 Doris Introduction

Doris is a high-performance and real-time analytical database based on the MPP architecture. Doris is well known for its high speed and ease of use. It can return query results in massive data in sub-second response time and supports high-concurrency point query scenarios. It also supports complex analysis scenarios with high throughput. Based on this, Apache Doris can meet the requirements of report analysis, ad hoc query, unified data warehouse construction, and data lake federated query acceleration. Users can build applications such as user behavior

analysis, A/B experiment platform, log retrieval analysis, user profile analysis, and order analysis.

Doris uses the MPP model, which is executed concurrently between nodes and within nodes. It is applicable to the distributed join of multiple large tables. Supports the vectorized query engine, adaptive query execution (AQE), optimization policy that combines CBO and RBO, and hot data cache query.

### 1.6.1.2 Common Concepts

In Doris, data is logically described in the form of tables.

- **Row&Column**

A table consists of rows and columns:

- Row: indicates a row of user data.
- Column: describes different fields in a row of data.

Columns can be divided into two categories: Key and Value. From the business perspective, Key and Value can correspond to the dimension column and indicator column respectively. From the point of view of the aggregation model, rows with the same key column are aggregated into one row. The aggregation mode of the Value column is specified by the user when creating a table.

- **Tablet&Partition**

In the storage engine of Doris, user data is horizontally divided into several data fragments (tablets, also called data buckets). Each Tablet contains several rows of data. Tablet data has no intersection and is physically stored independently.

Multiple tablets belong to different partitions logically. A tablet belongs to only one partition, and a partition contains several tablets. Because tablets are physically independent, partitions can be considered as physically independent. A tablet is the smallest physical storage unit for operations such as data movement and replication.

Multiple partitions form a table. A partition can be considered as the smallest logical management unit. Data can be imported and deleted only for one partition.

- **Data model**

Doris data models are classified into three types: Aggregate, Unique, and Duplicate.

- **Aggregate model**

When data is imported, rows with the same Key column are aggregated into one row, and the Value column is aggregated based on the AggregationType parameter. Currently, the AggregationType supports the following aggregation modes:

- SUM: Sum of values in multiple lines.
- Replace: Replace the value in the next batch of data.
- MAX: retain the maximum value.

- MIN: minimum value.

- **Unique model**

In some multidimensional analysis scenarios, users focus more on how to ensure the uniqueness of keys, that is, how to obtain the uniqueness constraint of the primary key. Therefore, the Unique data model is introduced.

- **Read-on-Merge**

The read-on-merge implementation of the Unique model can be completely replaced by the Replace mode in the Aggregate model. Its internal implementation and data storage are exactly the same.

- **Combine on write**

The unique model is implemented by combining data on write. The query performance of the unique model is closer to that of the duplicate model. Compared with the aggregate model, the unique model has higher query performance than the aggregate model in scenarios where primary key constraints are required, especially in aggregate queries and queries where a large amount of data needs to be filtered by indexes.

In the Unique table with the merge-on-write option enabled, the overwritten and updated data is marked and deleted during the import phase, and new data is written to a new file. During query, all data marked for deletion is filtered out at the file level, and the read data is the latest data. This eliminates the data aggregation process during read merge and supports pushdown of multiple predicates in many cases. Therefore, the performance can be greatly improved in many scenarios, especially in the case of aggregated queries.

- **Duplicate model**

In some multidimensional analysis scenarios, data has neither primary keys nor aggregation requirements. The Duplicate data model can be introduced to meet such requirements.

This data model is different from the Aggregate and Unique models. The data is stored exactly as the data in the import file, and there is no aggregation. Even if the two rows of data are identical, they are retained. The DUPLICATE KEY specified in the table creation statement is only used to specify the columns by which the underlying data is sorted.

- **Data Model Selection Suggestions**

The data model is determined when the table is created and cannot be modified. Therefore, it is important to choose a suitable data model.

- i. The Aggregate model greatly reduces the amount of data to be scanned and the amount of query calculation during aggregation query through pre-aggregation. It is suitable for the report query scenario with fixed mode. However, this model is not friendly to count(\*) queries. In addition, the aggregation mode of the Value column is fixed. Therefore, the semantic correctness needs to be considered when performing other types of aggregation queries.
- ii. The Unique model ensures the uniqueness of primary key constraints in scenarios where unique primary key constraints are required.

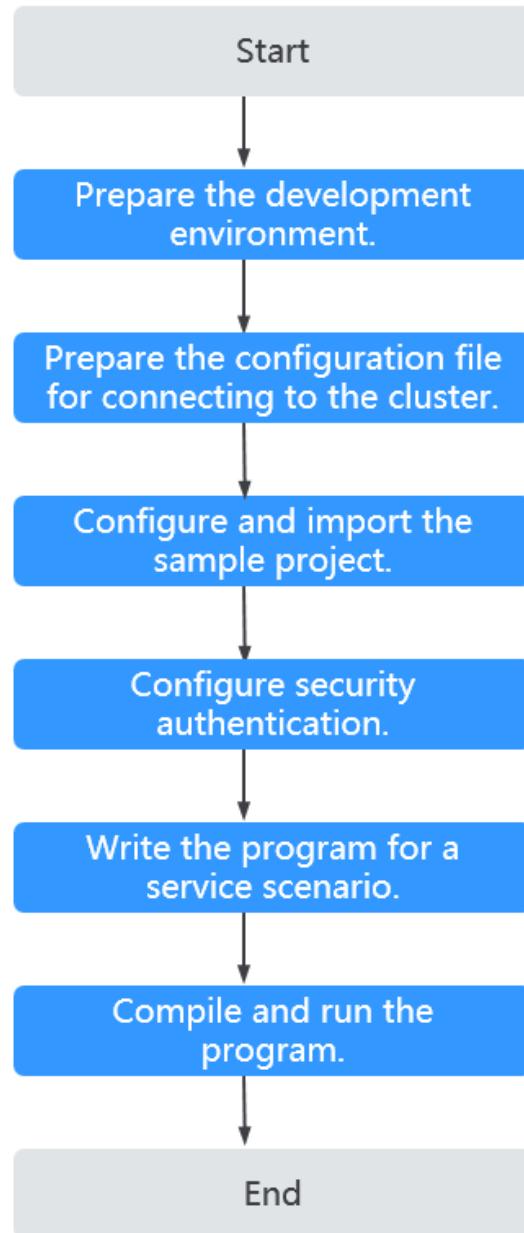
However, the query advantages of pre-aggregation such as ROLLUP cannot be taken advantage of.

- 1) For users who have high performance requirements for aggregated queries, the merge-on-write implementation introduced in version 1.2 is recommended.
- 2) The Unique model supports only the entire row update. If the unique primary key constraint and some columns need to be updated, (e.g., importing multiple source tables into a Doris table), you can use the Aggregate model and set the aggregation type of the non-primary key column to Replaced\_IF\_NOT\_NULL.
- 3) Duplicate is applicable to Ad-hoc queries in any dimension. Although the pre-aggregation feature cannot be used, it is not restricted by the aggregation model and can take advantage of the column-store model. (Only the relevant columns are read, not all the Key columns).

### 1.6.1.3 Development Process

The following figure shows the phases in the development process.

Figure 1-14 Doris application development process



**Table 1-9** HBase application development process description

Phase	Description	Reference Document
Prepare the development environment.	Before developing an application, you need to prepare the development environment. You are advised to use the Java language and IntelliJ IDEA tool for development. In addition, you need to complete the initial configuration of the JDK and Maven.	<a href="#">Preparing for Development Environment</a>
Prepare the Configuration File for Connecting to the Cluster.	During application development or running, you need to connect to the MRS cluster through cluster configuration files. The configuration file contains user files used for security authentication. You can obtain related content from the created MRS cluster.  Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured.	<a href="#">Preparing the Configuration Files for Connecting to the Cluster</a>
Configure and import the sample project.	Doris provides multiple sample programs in different scenarios. You can obtain sample projects and import them to the local development environment for program learning.	<a href="#">Configuring and Importing JDBC or Stream Load Sample Projects</a> <a href="#">Configuring and Importing SpringBoot Sample Projects</a>

Phase	Description	Reference Document
Configure security authentication.	When using the JDBC or Spring Boot interface to connect to Doris, you need to configure a user with Doris administrator rights for security authentication.	<a href="#">Configuring and Importing JDBC or Stream Load Sample Projects</a> <a href="#">Configuring and Importing SpringBoot Sample Projects</a>
Develop programs based on business scenarios.	Develop programs based on actual service scenarios and invoke component interfaces to implement corresponding functions.	<a href="#">Using Doris Through JDBC or DBalancer</a>
Compile and run the application.	You can commission and run the program in the local Windows development environment, or compile the program into a JAR package and submit it to the Linux node for running.	<a href="#">Application Commissioning</a>

#### 1.6.1.4 Doris Sample Project Introduction

To obtain the MRS sample project, visit <https://github.com/huaweicloud/huaweicloud-mrs-example>. Switch to the version branch that matches the MRS cluster, download the package to the local PC, and decompress the package to obtain the sample code project of each component.

Currently, MRS provides the following Doris-related sample projects:

**Table 1-10** Doris-related sample projects

Sample Project Location	Function Description
doris-examples/doris-jdbc-example	Application development example of Doris data read and write operations. You can invoke Doris APIs to create user tables, insert data into tables, query table data, and delete tables. For details, see <a href="#">Using Doris Through JDBC or DBalancer</a> .

Sample Project Location	Function Description
springboot/doris-examples	SpringBoot application development example for Doris data read and write operations. This section provides an example for connecting Doris to SpringBoot. For details, see <a href="#">Configuring and Importing SpringBoot Sample Projects</a> .
doris-examples/doris-stream-load-example	Application development example of the Doris Stream Load API for importing local CSV files. This section provides an example of using the Stream Load API to import data from a local CSV file to a Doris table. For details about the example, see <a href="#">Loading Data to a Doris Table with Stream Load</a> .

## 1.6.2 Environment Preparation

### 1.6.2.1 Preparing for Development Environment

[Table 1-11](#) describes the environment required for secondary development.

**Table 1-11** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>

Item	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
JUnit plug-in installation	Basic configuration for the development environment.

Item	Description
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.  Huawei provides Huawei Mirrors for you to download all dependency JAR files of sample projects. However, you need to download the rest dependency open-source JAR files from the Maven central repository or other custom repository address. For details, see <a href="#">Configuring Huawei Open-Source Mirrors</a> .
7-zip	Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

### 1.6.2.2 Preparing the Configuration Files for Connecting to the Cluster

#### Preparing for User Authentication

For an MRS cluster with Kerberos authentication enabled, you need to prepare a user who has the operation permission on related components for program authentication.

The following Doris permission configuration example is for reference only. You can modify the configuration as you need.

**Step 1** Log in to FusionInsight Manager.

**Step 2** Choose **System > Permission > Role**. On the displayed page, click **Create Role**.

1. Enter a role name, for example, **dorisrole**.
2. In the **Configure Resource Permission** area, select *Desired cluster > Doris*, select **Doris Adm Permission**, and click **OK**.

**Step 3** Choose **User**, and click **Create**. On the displayed page, create a human-machine user, for example, **developuser**, and associate the user to the role created in [Step 2](#).

**Step 4** Log in to FusionInsight Manager again as the **developuser** user and change the initial password of the user.

----End

#### Configuring the Network of the Running Environment

Nodes used for program debugging or running must be able to communicate with nodes within the MRS cluster, and the host domain names must be configured.

- Scenario 1: Configure the network connection between the local Windows environment and MRS cluster nodes.
  - a. Log in to FusionInsight Manager, click **Download Client** in the upper right corner of **Homepage**, select **Configuration Files Only** for **Select**

**Client Type**, and click **OK**. After the client file package is generated, download the client to the local PC as prompted and decompress it.

For example, if the client configuration file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Then, continue to decompress this file.

- b. Copy the **hosts** file content from the decompression directory to the **hosts** file of the local PC.

 NOTE

- If you need to debug the application in the local Windows environment, ensure that the local PC can communicate with the hosts listed in the **hosts** file.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- When configuring IPv6 hosts on the local PC running Windows, add **%** and the interface index of the network interface card (NIC) to the end of the IPv6 address. The interface index can be obtained by running the **ipconfig** command.

The following is an example:

`fec1:0:0:e505:8:99:5:1%10`

- Scenario 2: Configure the network communication between the Linux environment and MRS cluster nodes.
  - a. Install the MRS cluster client on nodes.  
For example, the client installation directory can be **/opt/hadoopclient**.
  - b. Obtain the configuration files.
    - i. Log in to FusionInsight Manager, click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.
    - ii. Log in to the active OMS node as user **root** and go to the directory where the client configuration file is stored. The default directory is **/tmp/FusionInsight-Client/**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:  
`cd /tmp/FusionInsight-Client`  
`tar -xvf FusionInsight_Cluster_1_Services_Client.tar`  
`tar -xvf FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar`  
`cd FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles`
  - c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If they are not contained, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed.

### 1.6.2.3 Configuring and Importing JDBC or Stream Load Sample Projects

#### Background

To run the JDBC or Stream Load API sample code of the Doris component, perform the following operations.

#### Procedure

**Step 1** Obtain the sample project folder **doris-jdbc-example** or **doris-stream-load-example** in the **src\doris-examples** directory in the sample code decompression directory by referring to [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** Import the sample project to the IntelliJ IDEA development environment.

1. On the menu bar of IntelliJ IDEA, choose **File > Open...** to display the Open File or Project dialog box.
2. In the displayed dialog box, select the **doris-jdbc-example** or **doris-stream-load-example** folder and click **OK**. In Windows, the full path of the folder does not contain spaces.

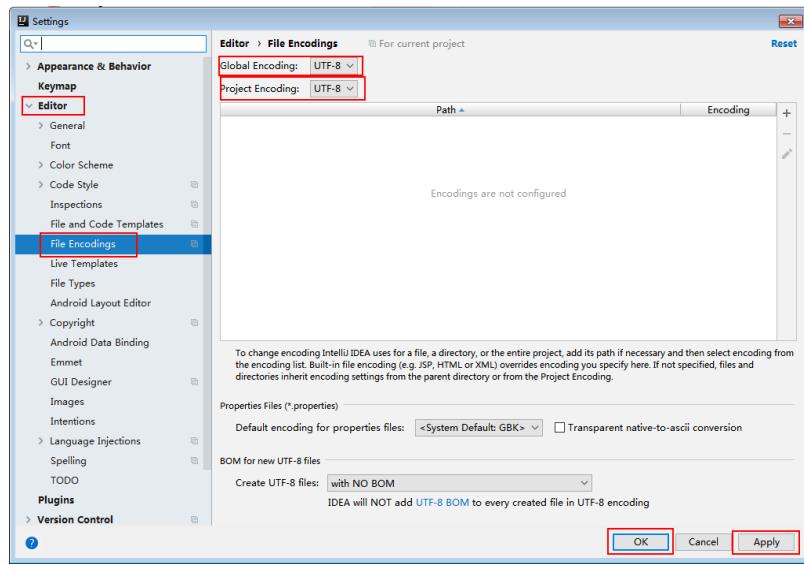
##### NOTE

- This section describes how to develop an application that connects to the Doris service in JDBC mode in Windows.
- Set the **DORIS\_MY\_USER** and **DORIS\_MY\_PASSWORD** environment variables in the local environment variables. Store the environment variables in ciphertext and decrypt the environment variables to ensure security.
  - **DORIS\_MY\_USER** indicates the user name for logging in to the Doris.
  - **DORIS\_MY\_PASSWORD** indicates the password for logging in to the Doris.If the error message "Could not connect to *IP address of the FE instance*.MySQL protocol query connection port" is displayed during the execution of the sample program, the configured local environment variables may not take effect. In this case, restart the computer for the settings to take effect.
- After the **jdb-example** sample project is imported, modify the following parameters:
  - In the code, change *xxx* in **HOST = "xxx"** to the IP address of the master FE node of the Doris. To obtain the IP address of the master FE node, choose **Cluster > Services > Doris** on Manager and view the **Host Where Leader Locates**.
  - In the code, change *xxx* where **PORT = "xxx"** to the MySQL query connection port of Doris. The default port is 29982. To obtain the port, log in to FusionInsight Manager, choose Cluster > Services > Doris > Configurations, and search for **query\_port**.

**Step 3** Set the encoding format of the IntelliJ IDEA text file to solve the problem of garbled characters.

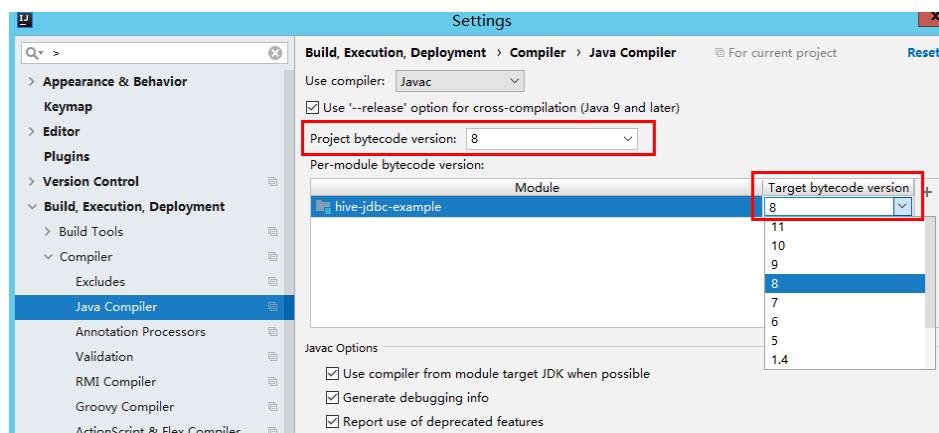
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA. The Settings window is displayed.
2. In the navigation tree on the left, choose **Editor > File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 1-15](#).

**Figure 1-15 Setting the Encoding Format of IntelliJ IDEA**

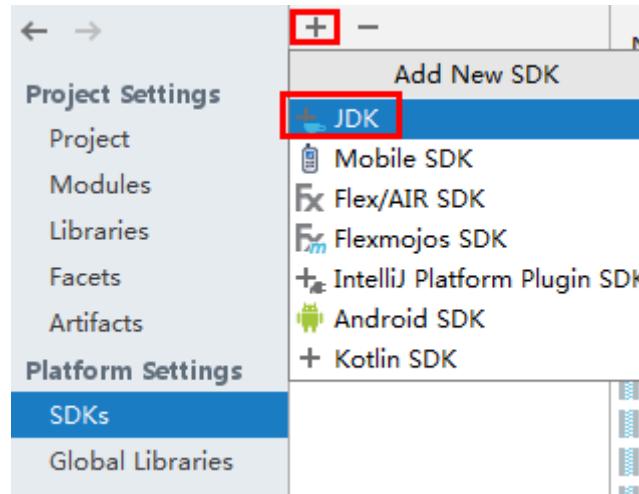


**Step 4** Set the JDK of the project.

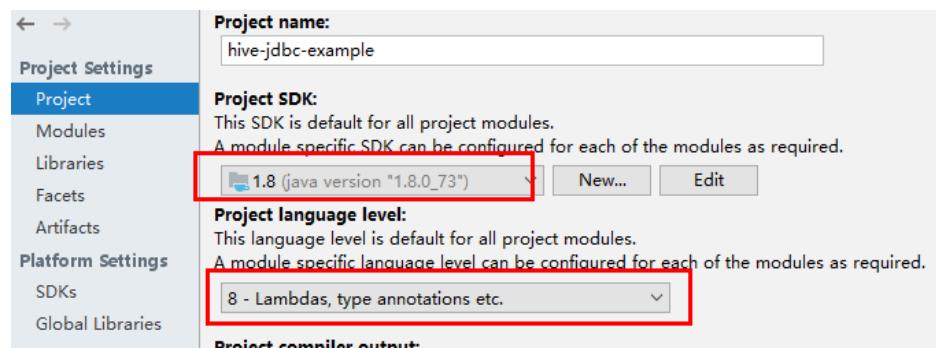
1. On the IntelliJ IDEA menu bar, choose **File > Settings....** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **doris-jdbc-example**.



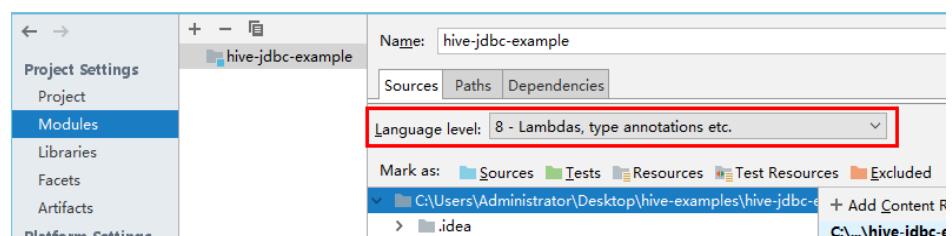
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



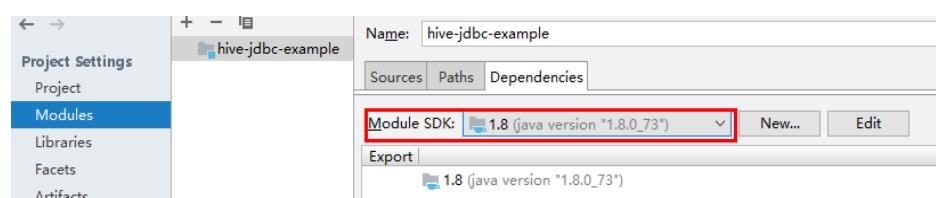
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

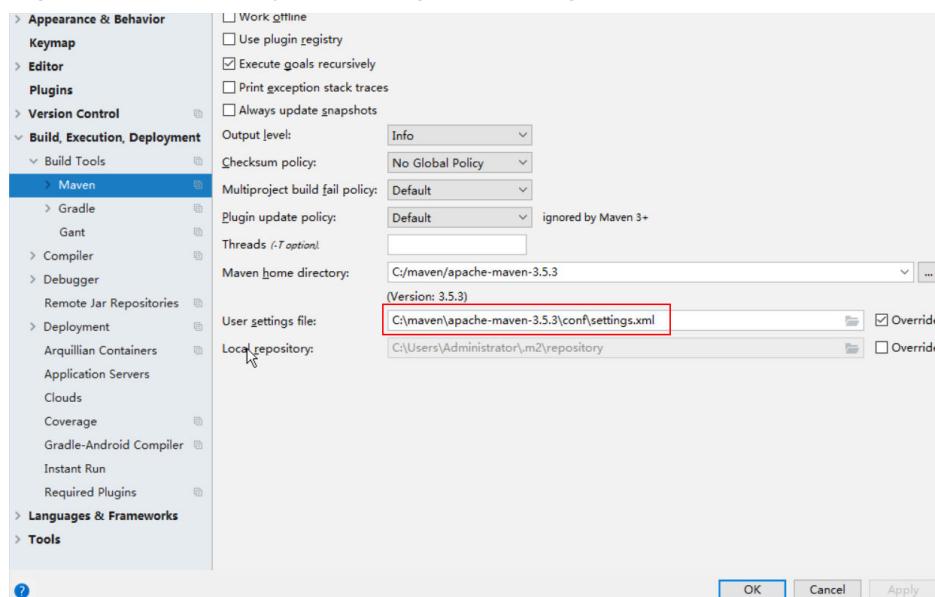


11. Click **Apply** and **OK**.

**Step 5** Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

**Figure 1-16** Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

#### 1.6.2.4 Configuring and Importing SpringBoot Sample Projects

##### Background

To run the SpringBoot API sample code of the Doris component of MRS, perform the following operations:

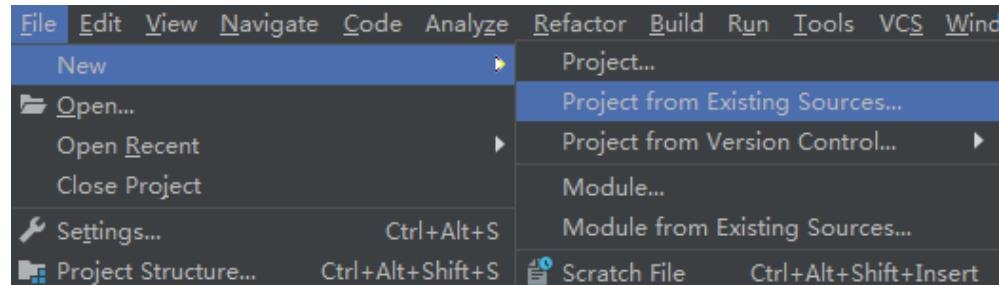
This section describes how to develop an application that connects to the Doris service in SpringBoot mode in Windows.

##### Procedure

- Step 1** Obtain the sample project folder **doris-rest-client-example** in the **src/springboot/doris-examples** directory in the sample code decompression directory by referring to [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Choose **File > New > Project from Existing Sources**.



2. In the **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **doris-rest-client-example** folder and click **OK**.
3. Confirm the subsequent configuration and click **Next**. If there is no special requirement, use the default value.

Select the recommended JDK version and click **Finish**.

**NOTE**

- This section describes how to develop an application that connects to the Doris service in SpringBoot mode in Windows.
- Set the **DORIS\_MY\_USER** and **DORIS\_MY\_PASSWORD** environment variables in the local environment variables. Store the environment variables in ciphertext and decrypt the environment variables to ensure security.
  - **DORIS\_MY\_USER** indicates the user name for logging in to the Doris.
  - **DORIS\_MY\_PASSWORD** indicates the password for logging in to the Doris.If the error message "Could not connect to *IP address of the FE instance*.MySQL protocol query connection port" is displayed during the execution of the sample program, the configured local environment variables may not take effect. In this case, restart the computer for the settings to take effect.
- After the **doris-rest-client-example** sample project is imported, modify the following parameters:
  - In the code, change *xxx* in **HOST = "xxx"** to the IP address of the master FE node of the Doris. To obtain the IP address of the master FE node, choose **Cluster > Services > Doris** on Manager and view the **Host Where Leader Locates**.
  - In the code, change *xxx* where **PORT = "xxx"** to the MySQL query connection port of Doris. The default port is 29982. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **query\_port**.

----End

 NOTE

- This section describes how to develop an application that connects to the Doris service in JDBC mode in Windows.
- After the jdbc-example sample project is imported, modify the following parameters:
- Change the value of xxx under USER = "xxx" in the code to the actual development user, for example, developer.
- Change the value of xxx under PASSWD = "xxx" in the code to the actual password of the development user.
- In the code, change xxx in HOST = "xxx" to the IP address of the master FE node of the Doris. You can obtain the IP address of the master FE node by choosing Cluster > Service > Doris on FusionInsight Manager and checking the host where the leader resides.
- In the code, change xxx where PORT = "xxx" to the MySQL query connection port of Doris. The default port is 29982. To obtain the port, log in to FusionInsight Manager, choose Cluster > Service > Doris > Configuration, and search for query\_port.

## 1.6.3 Doris Application Development

### 1.6.3.1 Using Doris Through JDBC or DBalancer

#### 1.6.3.1.1 Service Scenario Description

This section describes how to quickly learn the Doris development process and understand key interface functions.

#### Scenario

Doris can use SQL statements to perform common service operations. The SQL operations involved in the code example include creating a database, creating a table, inserting table data, querying table data, deleting a table, and deleting a database (DBalancer).

The code sample operation process is as follows:

1. Connect to Doris using JDBC or DBalancer.
2. Create a database.
3. Create a table in the database.
4. Inserts data into a table.
5. Query table data.
6. Delete the table.
7. Delete a database. (DBalancer)

#### 1.6.3.1.2 Application Development Approach

#### Function Decomposition

Doris is compatible with the MySQL protocol. Common operations can be performed by using the SQL language.

The development process consists of the following parts:

- Establish a connection: Establish a connection to the Doris service instance.
- Create Library: Create a Doris database.
- Create Table: Create a table in the Doris database.
- Insert Data: Insert data into the Doris table.
- Querying data: Query the Doris table data.
- Deleting a table: You can delete a created Doris table.
- Deleting a database (DBalancer): You can delete a created Doris database.

### 1.6.3.1.3 Setting Up a Connection

## Connecting Doris Using JDBC

The following code snippet uses the **createConnection** method of the **JDBCExample** class in the **com/huawei/bigdata/doris/example** package.

```
private static Connection createConnection() throws Exception {
    Connection connection = null;
    try {
        Class.forName(JDBC_DRIVER);
        String dbUrl = String.format(DB_URL_PATTERN, HOST, PORT);
        connection = DriverManager.getConnection(dbUrl, USER, PASSWD);
    } catch (Exception e) {
        logger.error("Init doris connection failed.", e);
        throw new Exception(e);
    }
    return connection;
}
```

In the preceding code snippet, HOST and PORT indicate the IP address of the master FE node and the MySQL query connection port for Doris, respectively.

- To obtain the IP address of the master FE node, log in to FusionInsight Manager, choose **Cluster > Services > Doris**, and check the **Host Where Leader Locates**.
- The default MySQL query connection port is 29982. To obtain the port number, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **query\_port**.

```
private static final String HOST = "192.168.67.78";
private static final int PORT = 29982;
```

## Connecting Doris Using DBalancer

The following code snippet uses the **createConnection** method on the **DBalancerJDBCExample** class in the **com/huawei/bigdata/doris/example** package.

```
private static Connection createConnection() throws Exception {
    Connection connection = null;
    try {
        Class.forName(JDBC_DRIVER);
        String dbUrl = String.format(DB_URL_PATTERN, HOST, PORT);
        connection = DriverManager.getConnection(dbUrl, USER, PASSWD);
    } catch (Exception e) {
        logger.error("Init doris connection failed.", e);
        throw new Exception(e);
    }
    return connection;
}
```

In the preceding code snippet, HOST and PORT indicate the IP address of the DBalancer node and DBalancer TCP access port set in the following example.

- To obtain the IP address of the DBalancer node, log in to FusionInsight Manager and choose **Cluster > Services > Doris > Instances**.
- The default TCP access port of DBalancer is 29992. To obtain the port number, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **balancer\_tcp\_port**.

```
private static final String HOST = "192.168.20.37";
private static final int PORT = 29992;
```

#### 1.6.3.1.4 Creating a Database

Run the SQL statement in Java JDBC or Java DBalancer mode to create the database corresponding to the *dbName* variable in the cluster.

```
String createDatabaseSql = "create database if not exists dbName";
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

#### 1.6.3.1.5 Creating a Table

Run the SQL statement in Java JDBC or Java DBalancer mode to create the table corresponding to *tableName* in the database corresponding to the *dbName* variable in the cluster.

```
String createTableSql = "create table if not exists " + dbName + "." + tableName + " (\n" +
    "c1 int not null,\n" +
    "c2 int not null,\n" +
    "c3 string not null\n" +
    ") engine=olap\n" +
    "unique key(c1, c2)\n" +
    "distributed by hash(c1) buckets 1";
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

#### 1.6.3.1.6 Inserting Data

Run the following SQL statement in Java JDBC or Java DBalancer mode to insert data into the *dbName.tableName* table of the cluster.

```
String insertTableSql = "insert into " + dbName + "." + tableName + " values(?, ?, ?)";
private static void insert(Connection connection, String sql) throws Exception {
    int INSERT_BATCH_SIZE = 10;
    try(PreparedStatement stmt = connection.prepareStatement(sql)) {
        for (int i =0; i < INSERT_BATCH_SIZE; i++) {
            stmt.setInt(1, i);
            stmt.setInt(2, i * 10);
            stmt.setString(3, String.valueOf(i * 100));
            stmt.addBatch();
        }
        stmt.executeBatch();
    } catch (Exception e) {
```

```
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

### 1.6.3.1.7 Querying Data

Use Java JDBC or Java DBalancer to run SQL statements to query data in the *dbName.tableName* table in the cluster.

```
String querySql = "select * from " + dbName + "." + tableName + " limit 10";
private static void query(Connection connection, String sql) throws Exception {
    try (Statement stmt = connection.createStatement()) {
        ResultSet resultSet = stmt.executeQuery(sql) {
            ResultSetMetaData md = resultSet.getMetaData();
            int columnCount = md.getColumnCount();
            StringBuffer stringBuffer = new StringBuffer();
            logger.info("Start to print query result.");
            for (int i = 1; i <= columnCount; i++) {
                stringBuffer.append(md.getColumnName(i));
                stringBuffer.append(" ");
            }
            logger.info(stringBuffer.toString());

            while (resultSet.next()) {
                stringBuffer = new StringBuffer();
                for (int i = 1; i <= columnCount; i++) {
                    stringBuffer.append(resultSet.getObject(i));
                    stringBuffer.append(" ");
                }
                logger.info(stringBuffer.toString());
            }
        } catch (Exception e) {
            logger.error("Execute sql {} failed.", sql, e);
            throw new Exception(e);
        }
    }
}
```

### 1.6.3.1.8 Deleting a Table

Run the SQL statement in Java JDBC or Java DBalancer mode to delete the *dbName.tableName* table from the cluster.

```
String dropSql = "drop table " + dbName + "." + tableName;
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

### 1.6.3.1.9 Deleting a Database

Run the SQL statement using Java DBalancer to delete the *dbName* database from the cluster.

```
String dropDb = "drop database " + dbName;
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

### 1.6.3.2 Loading Data to a Doris Table with Stream Load

The Doris Stream Load sample program imports local CSV file data to a Doris table.

#### Example Code

Modify the following parameters in the sample code based on the site requirements:

- **HOST**: IP address of the master FE node of Doris. To obtain the IP address of the master FE node, log in to FusionInsight Manager, choose **Cluster > Services > Doris**, and check **Host Where Leader Locates**.
- **PORT**: HTTPS port of the Doris FE service. The default value is **29991**. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > Doris**, click **Configurations**, and search for **https\_port**.
- **JDBC\_PORT**: The value is the MySQL protocol query connection port of the Doris. The default port is **29982**. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **query\_port**.
- **DATABASE**: database of the table storing imported data
- **TABLE**: name of the Doris table that stores imported data
- **getHttpPost**: path of the CSV file to be imported

```
public class DorisStreamLoader {  
  
    // FE IP Address  
    private final static String HOST = "192.168.13.178";  
    // If Kerberos authentication is enabled for the cluster (the cluster is in security mode), FE port is the value  
    // of https_port; If Kerberos authentication is disabled for the cluster (the cluster is in normal mode), FE port  
    // is the value of http_port.  
    private final static int PORT = 29991;  
  
    private final static int JDBC_PORT = 29982;  
    // db name  
    private final static String DATABASE = "test_2";  
    // table name  
    private final static String TABLE = "doris_test_sink";  
  
    private static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";  
    private static final String DB_URL_PATTERN = "jdbc:mysql://%s:%d?rewriteBatchedStatements=true";  
  
    private static final String USER = System.getenv("DORIS_MY_USER");  
    private static final String PASSWD = System.getenv("DORIS_MY_PASSWORD");  
  
    // If Kerberos authentication is enabled for the cluster (the cluster is in security mode), Start the url with  
    // https; If Kerberos authentication is disabled for the cluster (the cluster is in normal mode), start the url  
    // with http.  
    private final static String loadUrl = String.format("https://%s:%s/api/%s/%s/_stream_load",  
        HOST, PORT, DATABASE, TABLE);  
  
    //Call the Curl method.  
    public static String execCurl(String[] cmd) {  
        ProcessBuilder process = new ProcessBuilder(cmd);  
        Process p;  
        try {  
            p = process.start();  
            BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));  
            StringBuilder builder = new StringBuilder();  
            String line;  
            while ((line = reader.readLine()) != null) {  
                System.out.println(line);  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
builder.append(line);
builder.append(System.getProperty("line.separator"));
}
return builder.toString();

} catch (Exception e) {
System.out.print("error");
}
return null;
}

public static void initTable(){
String createDatabaseSql = "create database if not exists "+DATABASE;

String createTableSql = "create table if not exists " + DATABASE + "." + TABLE + " (\n" +
" `id` int NULL COMMENT '\"',\n" +
" `number` int NULL COMMENT '\"',\n" +
" `price` DECIMAL(12,2) NULL COMMENT '\"',\n" +
" `skuname` varchar(40) NULL COMMENT '\"',\n" +
" `skudesc` varchar(200) NULL COMMENT '\"'\n" +
" ) ENGINE=OLAP\n" +
" DUPLICATE KEY(`id')\n" +
" COMMENT \"Offering information table\"\n" +
" DISTRIBUTED BY HASH(`id`) BUCKETS 1\n" +
" PROPERTIES (\n" +
" \"replication_num\" = \"3\",\n" +
" \"in_memory\" = \"false\",\n" +
" \"storage_format\" = \"V2\"\n" +
" );";
try (Connection connection = createConnection()) {
// Create a database.
System.out.println("Start create database.");
execDDL(connection, createDatabaseSql);
System.out.println("Database created successfully.");
// Create a table.
System.out.println("Start create table.");
execDDL(connection, createTableSql);
System.out.println("Table created successfully.");
} catch (Exception e) {
System.out.println("Execute doris operation failed.");
}
}

private static Connection createConnection() throws Exception {
Connection connection = null;
try {
Class.forName(JDBC_DRIVER);
String dbUrl = String.format(DB_URL_PATTERN, HOST, JDBC_PORT);
connection = DriverManager.getConnection(dbUrl, USER, PASSWD);
} catch (Exception e) {
System.out.println("Init doris connection failed.");
throw new Exception(e);
}
return connection;
}

public static void execDDL(Connection connection, String sql) throws Exception {
try (PreparedStatement statement = connection.prepareStatement(sql)) {
statement.execute();
} catch (Exception e) {
System.out.println("Execute sql {} failed.");
throw new Exception(e);
}
}

// Call the API.
public static String getHttpPost(String csvPath) {

String[] cmdList = {"curl", "-k", "--location-trusted", "-u" + USER + ":" + PASSWD, "-H", "expect:100-
```

```
continue", "-H", "column_separator:", "-T",
csvPath,
loadUrl};

String responseMsg = execCurl(cmdList);
System.out.println("curl" + responseMsg);

return responseMsg;
}

public static void main(String[] args) throws IOException {
initTable();
String path = DorisStreamLoader.class.getClassLoader().getResource("test.csv").getPath();
path = URLDecoder.decode(path, "UTF-8");
File file = new File(path);
String filePath = file.getAbsolutePath();
// In the Linux scenario, upload the test.csv file in the resource directory to the Linux background and
// specify the file path in getPost.
getPost(filePath);
}
}
```

## 1.6.4 Application Commissioning

### 1.6.4.1 Commissioning an Application in Windows

#### 1.6.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

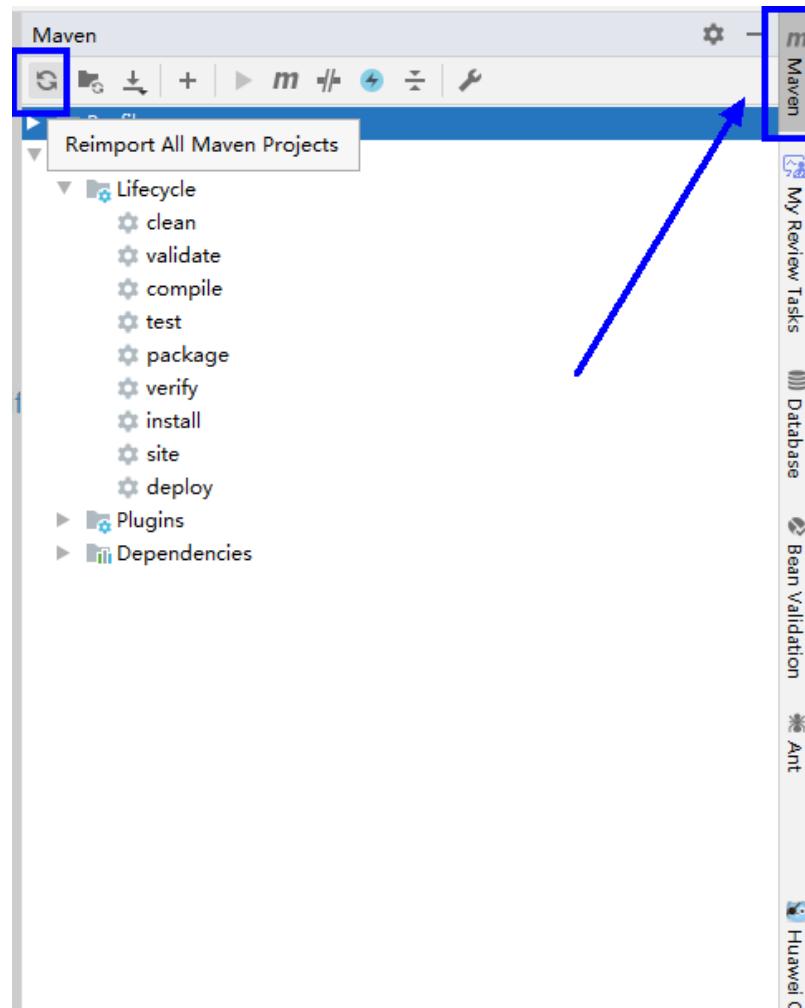
##### NOTE

- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

##### Procedure

**Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

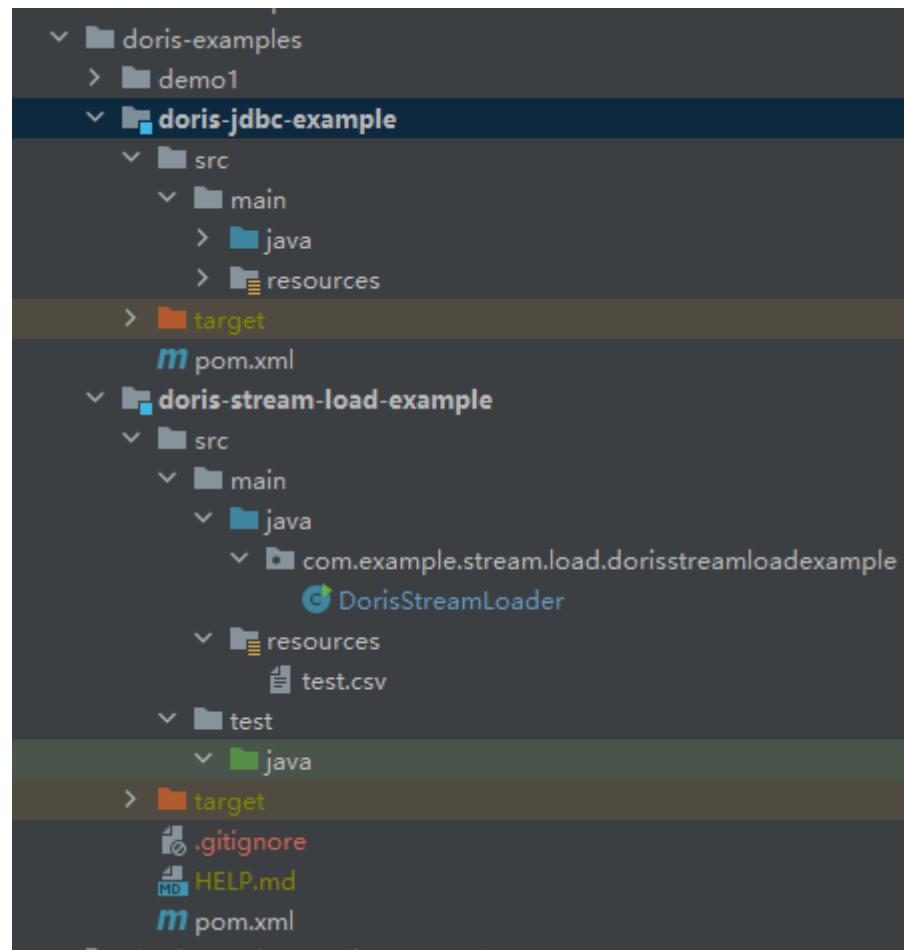
Figure 1-17 reimport projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user.  
See [Figure 1-18](#).

Figure 1-18 Directory list of Doris to be compiled



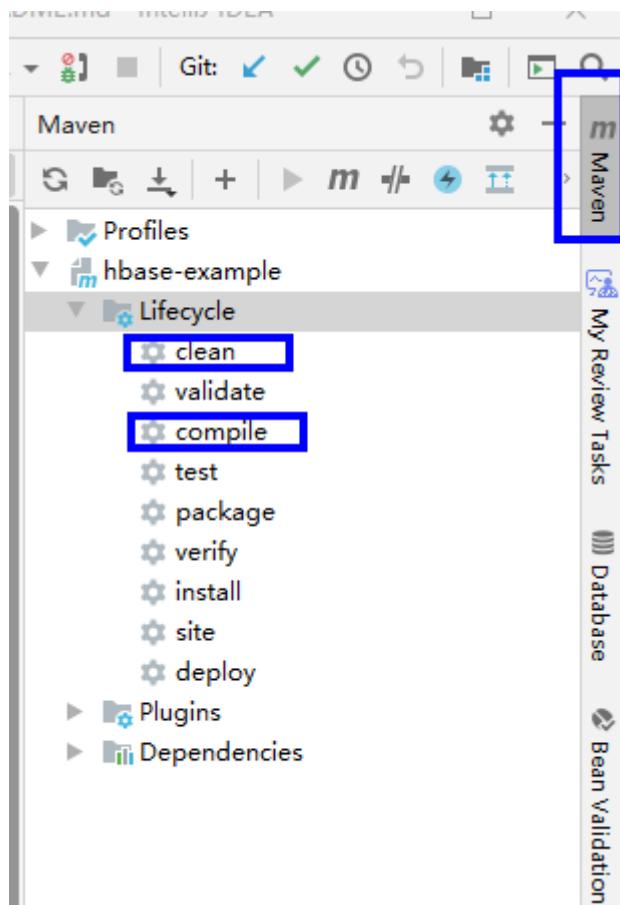
1. Compile an application.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

Figure 1-19 clean and compile tools of Maven



- Method 2:

Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

When the compilation is complete, the **Build Success** message is printed and the target directory is generated.

Figure 1-20 doris-jdbc-example compiled

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.241 s  
[INFO] Finished at: 2023-08-17T23:09:46+08:00  
[INFO] -----  
PS D:\gitSource\sample_project\src\doris-examples\doris-jdbc-example>
```

Figure 1-21 doris-stream-load-example compiled

```
[INFO] ----- [ jar ] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ doris-stream-load-example ---
[INFO] Deleting D:\doris-stream-load-example\target
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time:  0.190 s
[INFO] Finished at: 2024-06-12T11:26:09+08:00
[INFO] -----
```

 NOTE

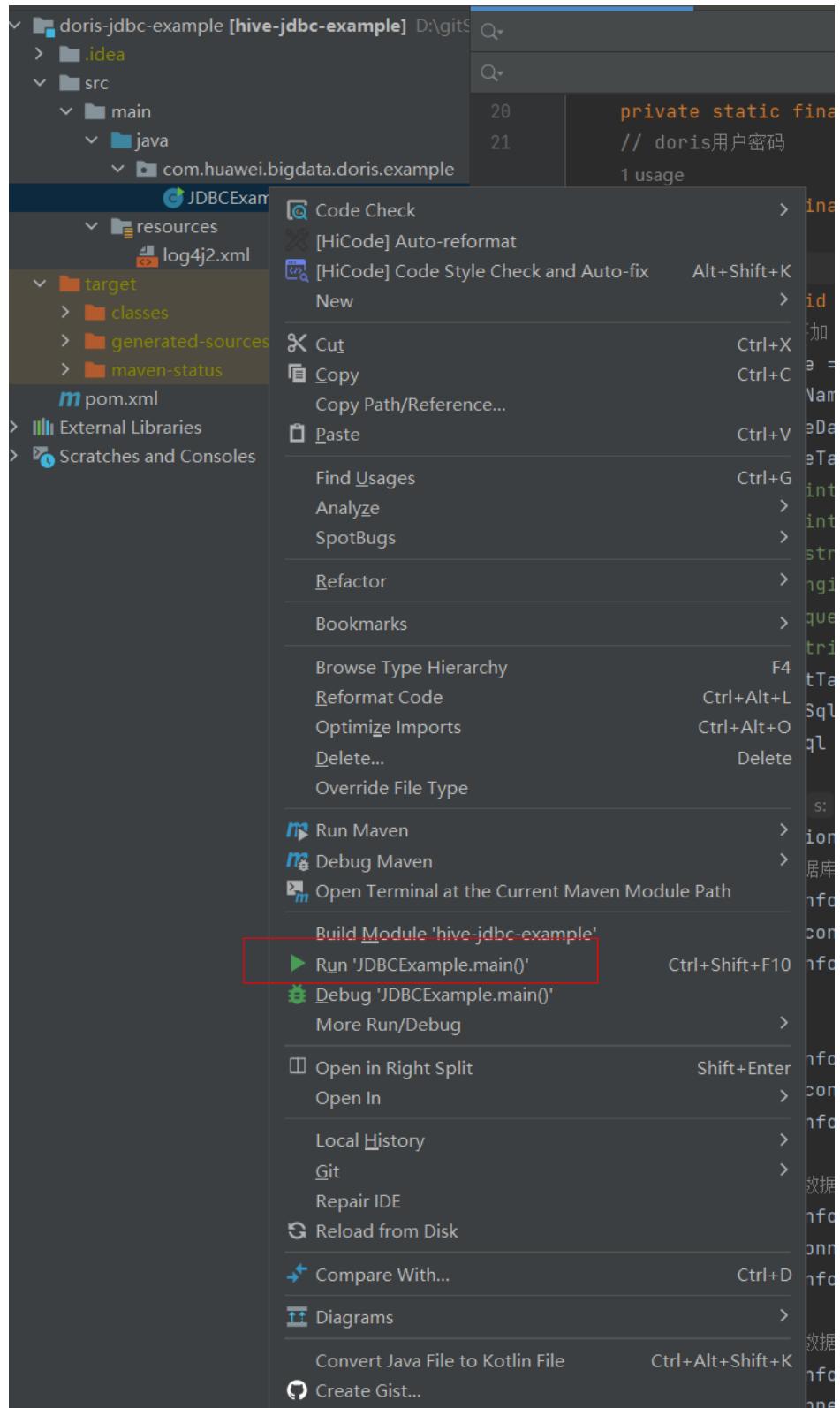
If the error message "Please use -source 7 or later to enable try-with-resources" is displayed during compilation, add the following content to the **properties** tag in the **pom.xml** configuration file:

```
<maven.compiler.source>8</maven.compiler.source>
<maven.compiler.target>8</maven.compiler.target>
```

2. Run the program.

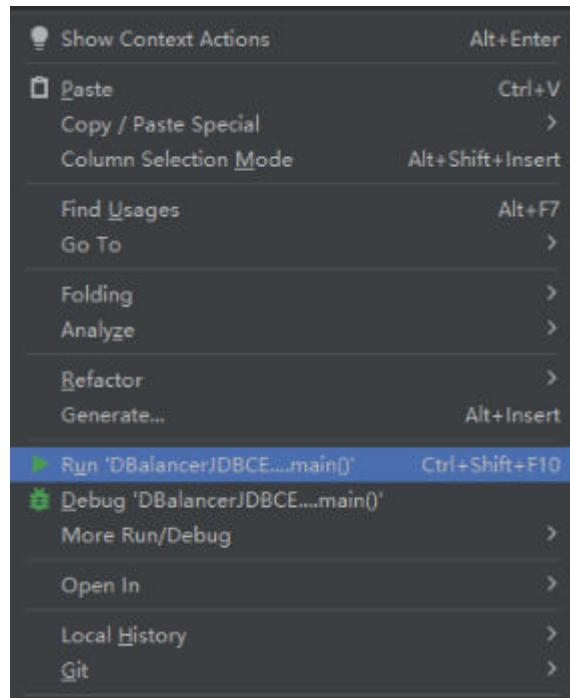
- Right-click the **JDBCExample.java** file and choose **Run 'JDBCExample.main()'**.

Figure 1-22 Run the Doris application.



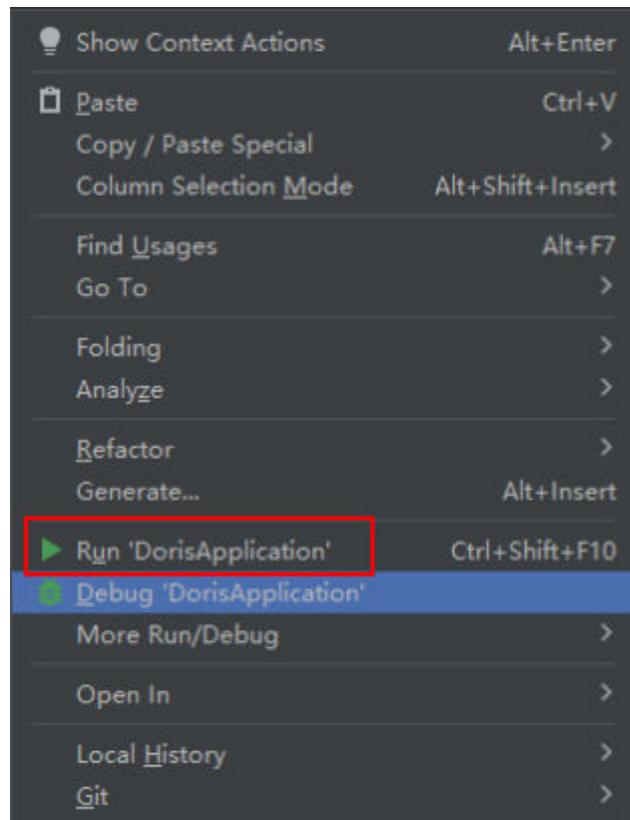
- Right-click the **DBalancerJDBCExample.java** file and choose **Run 'DBalancerJDBCExample.main()'**.

Figure 1-23 Run the DorisDBalancer application.



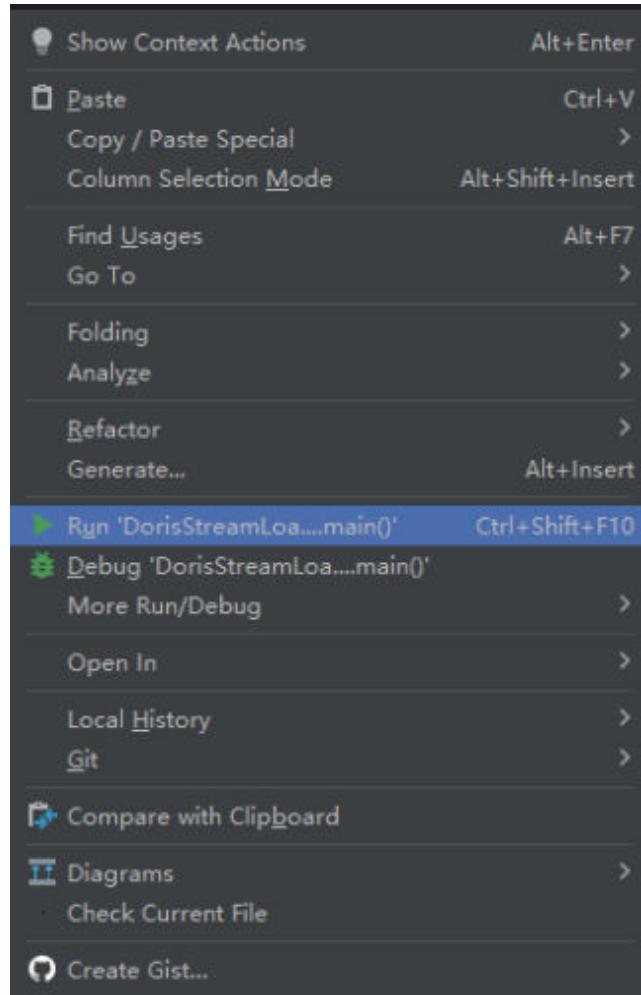
- Right-click the **DorisApplication.java** file and choose **Run 'DorisApplication'**.

Figure 1-24 Run the Doris SpringBoot application.



- Right-click the **DorisStreamLoader.java** file and choose **Run 'DorisStreamLoader.main()'**.

Figure 1-25 Run the Doris Stream Load application.



----End

#### 1.6.4.1.2 Viewing Windows Commissioning Results

##### Operation Scenario

After a Doris application finishes running, you can use one of the following methods to view the running result:

- Check the IntelliJ IDEA.
- Check the Doris log.

##### Procedure

- After the **JDBCExample.java** of **doris-jdbc-example** sample is successfully executed, the following information is displayed:

```
2023-08-17 23:13:13,473 | INFO | main | Start execute doris example. |  
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:41)  
2023-08-17 23:13:13,885 | INFO | main | Start create database. |
```

```
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:44)
2023-08-17 23:13:13,949 | INFO | main | Database created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:46)
2023-08-17 23:13:13,950 | INFO | main | Start create table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:49)
2023-08-17 23:13:14,132 | INFO | main | Table created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:51)
2023-08-17 23:13:14,133 | INFO | main | Start to insert data into the table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:54)
2023-08-17 23:13:14,733 | INFO | main | Inserting data to the table succeeded. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:56)
2023-08-17 23:13:14,733 | INFO | main | Start to query table data. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:59)
2023-08-17 23:13:15,079 | INFO | main | Start to print query result. |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:121)
2023-08-17 23:13:15,079 | INFO | main | c1 c2 c3 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:126)
2023-08-17 23:13:15,079 | INFO | main | 0 0 0 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 1 10 100 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 2 20 200 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 3 30 300 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 4 40 400 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 5 50 500 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 6 60 600 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 7 70 700 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 8 80 800 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 9 90 900 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | Querying table data succeeded. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:61)
2023-08-17 23:13:15,081 | INFO | main | Start to delete the table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:64)
2023-08-17 23:13:15,114 | INFO | main | Table deleted successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:66)
2023-08-17 23:13:15,124 | INFO | main | Doris example execution successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:71)
```

Process finished with exit code 0

- After the **DBalancerJDBCExample.java** of **doris-jdbc-example** sample is successfully executed, the following information is displayed:

```
2024-06-12 14:22:08,716 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:131)
2024-06-12 14:22:08,716 | INFO | main | c1 c2 c3 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:136)
2024-06-12 14:22:08,717 | INFO | main | 0 0 0 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 1 10 100 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 2 20 200 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 3 30 300 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 4 40 400 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 5 50 500 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 6 60 600 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 7 70 700 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 8 80 800 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 9 90 900 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | Querying table data succeeded. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:66)
2024-06-12 14:22:08,718 | INFO | main | Start to delete the table. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:69)
2024-06-12 14:22:08,836 | INFO | main | Table deleted successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:71)
2024-06-12 14:22:08,836 | INFO | main | Start to delete the database. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:74)
2024-06-12 14:22:08,961 | INFO | main | Database deleted successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:76)
2024-06-12 14:22:08,962 | INFO | main | Doris example execution successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:81)

Process finished with exit code 0
```

- Running result of interconnection between Doris and SpringBoot
  - The result of running the Doris JDBC sample application using SpringBoot is as follows:

Enter **http://IP address of the node that runs the sample:8080/doris/example/executesql** in the address box of the browser. IDEA prints logs. The following figure shows the returned information.

**Figure 1-26** Doris JDBC returned running information



```
localhost:8080/doris/example/executesql
=====
Doris Example Start =====
Start create database. Database created successfully. Start create table. Table created successfully. Start to insert data into the table. Inserting data to the table succeeded. Start to query table data. Query result: c1 c2 c3 0 0 0 1 10 100 2.20 200 3 30 300 4 40 400 5 50 500 6 60 600 7 70 700 8 80 800 9 90 900 Querying table data succeeded. Start to delete the table. Table deleted successfully.
=====
Doris Example End =====
```

- The result of running the Doris DBalancer sample application using SpringBoot is as follows:

**Figure 1-27** Doris DBalancer returned running information



```
=====
Doris Example Start =====
10 100 2 20 200 3 30 300 4 40 400 5 50 500 6 60 600 7 70 700 8 80 800 9 90 900 Querying table data succeeded. Start to delete the table. Table deleted successfully. Start to delete the database. Database deleted successfully.
=====
Doris Example End =====
```

- If the **doris-stream-load-example** sample is successfully executed, the following information is displayed:

```
Start create database.
Database created successfully.
Start create table.
Table created successfully.
{
  "TxnId": 27,
  "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
  "Comment": "",
  "TwoPhaseCommit": "false",
  "Status": "Success",
  "Message": "OK",
  "NumberTotalRows": 4,
  "NumberLoadedRows": 4,
  "NumberFilteredRows": 0,
  "NumberUnselectedRows": 0,
  "LoadBytes": 141,
  "LoadTimeMs": 387,
  "BeginTxnTimeMs": 58,
  "StreamLoadPutTimeMs": 137,
  "ReadDataTimeMs": 0,
  "WriteDataTimeMs": 94,
  "CommitAndPublishTimeMs": 95
}
curl{
  "TxnId": 27,
  "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
  "Comment": "",
  "TwoPhaseCommit": "false",
  "Status": "Success",
  "Message": "OK",
  "NumberTotalRows": 4,
  "NumberLoadedRows": 4,
  "NumberFilteredRows": 0,
  "NumberUnselectedRows": 0,
  "LoadBytes": 141,
  "LoadTimeMs": 387,
  "BeginTxnTimeMs": 58,
  "StreamLoadPutTimeMs": 137,
  "ReadDataTimeMs": 0,
  "WriteDataTimeMs": 94,
  "CommitAndPublishTimeMs": 95
}
```

Connect to Doris on the MySQL client and run the following command to view the imported data:

```
select * from doris_test_sink;
```

```
mysql> select * from doris_test_sink;
+-----+-----+-----+-----+
| id   | number | price | skuname | skudesc      |
+-----+-----+-----+-----+
| 10001 |    12  | 13.30 | test1   | this is atest
| 10002 |   100  | 15.30 | test2   | this is atest
| 10003 |   102  | 16.30 | test3   | this is atest
| 10004 |   120  | 17.30 | test4   | this is atest
+-----+-----+-----+-----+
4 rows in set (0.02 sec)
```

## 1.6.4.2 Commissioning an Application in Linux

### 1.6.4.2.1 Compiling and Running Applications

#### Scenario

After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

#### Prerequisites

- You have installed a JDK in the Linux environment. The version of the JDK must be consistent with that of the JDK used by IntelliJ IDEA to export the JAR file.
- If the host where the Linux environment is deployed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the Linux environment is deployed. The host names and IP addresses must be mapped one on one.

## Compiling and Running JDBC, DBalancer and SpringBoot Programs

### Step 1 Export a JAR file.

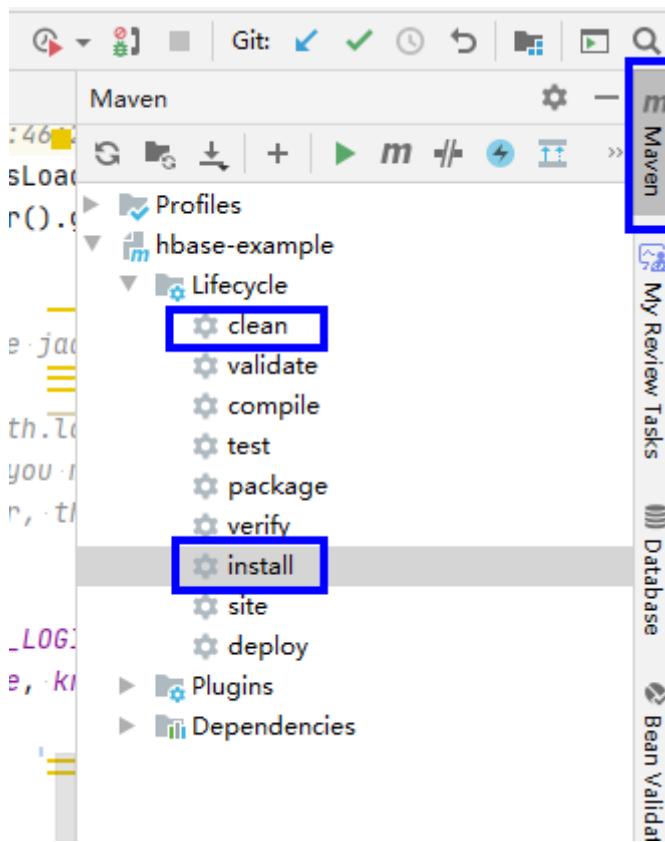
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 1-28 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

**Step 2** Decide whether to run Doris to connect to SpringBoots.

- If yes, perform the following steps to run the sample:
  - Create a running directory in the Linux environment and save the **doris-rest-client-example-\*jar** file in the **target** directory to this directory.
  - Switch to the running directory and run the following command to run the JAR package:  
**java -jar doris-rest-client-example-\*jar-with-dependencies.jar**
- If no, go to **Step 4**.

**Step 3** Export the JAR package on which the sample project depends.

Go to the directory where the **pom.xml** file is stored in the **Terminal** window at the bottom of the IDEA or in other command line tools. Run the following command:

**mvn dependency:copy-dependencies -DoutputDirectory=lib**

The **lib** folder is generated in the directory where the **pom.xml** file is stored. The **lib** folder contains the JAR packages on which the sample project depends.

**Step 4** Prepare the dependency JAR file and configuration file.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create the subdirectory **lib**. Export the JAR package on which the sample project depends, and upload the JAR package generated in **Step 1** and JAR package generated in **Step 3** to the **lib** directory of the Linux operating system.
2. In the **/opt/test** root directory, create the **run.sh** script, modify the following content, and save the file:

```
#!/bin/sh  
BASEDIR=`cd $(dirname $0);pwd`  
cd ${BASEDIR}  
for file in ${BASEDIR}/lib/*.jar  
do  
i_cp=$i_cp:$file  
echo "$file"  
done  
  
java -cp ${i_cp} com.huawei.bigdata.doris.example.JDBCExample
```

**com.huawei.bigdata.doris.example.JDBCExample** is used as an example. Replace it with the actual code.

**Step 5** Configure the environment variables **DORIS\_MY\_USER** and **DORIS\_MY\_PASSWORD**.

1. Add the following content to the **/etc/profile** file and save the file:  
*DORIS\_MY\_USER=Username for accessing Doris*  
*DORIS\_MY\_PASSWORD=Password for accessing Doris*
2. Run the following command to apply the environment variables:  
**source /etc/profile**

**Step 6** Go to **/opt/test** and run the following command to run the JAR file:

**sh run.sh**

----End

## Compiling and Running the Stream Load Program

- Step 1** Create a directory in the Linux environment, for example, **/opt/test**, and upload the **/src/main/resources/test.csv** file prepared in the **doris-stream-load-example** sample project obtained in **Step 1** to the directory.
- Step 2** Change the value of **getHttpPost** in the **doris-stream-load-example** sample code to **/opt/test/test.csv**.
- Step 3** Export a JAR file.

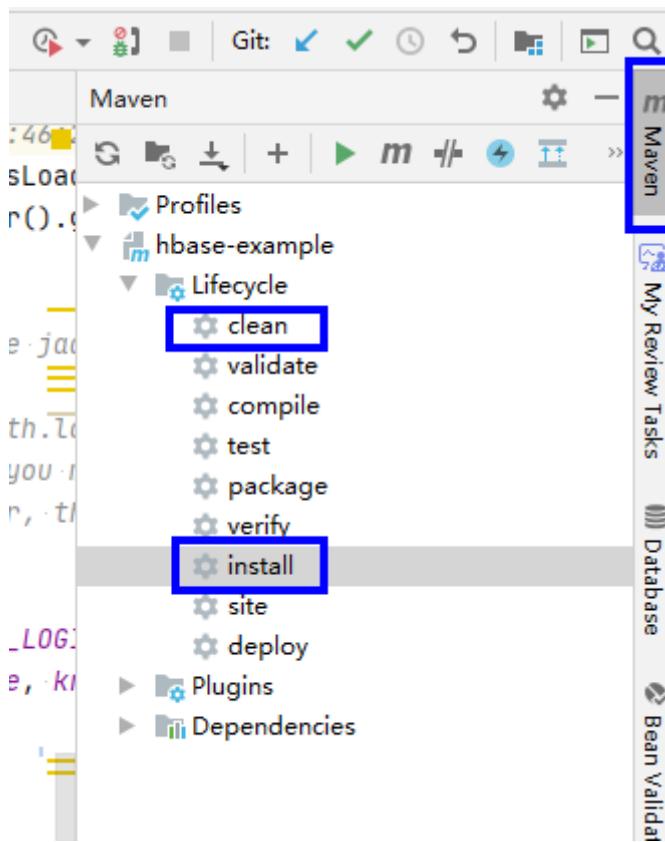
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 1-29 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

**Step 4** Place **doris-stream-load-example-\*jar** in the **target** directory to the directory created in **Step 1**, for example, **/opt/test**.

**Step 5** Switch to the JDK directory and run the JAR package.

```
java -jar doris-stream-load-example-*jar-with-dependencies.jar
```

----End

#### 1.6.4.2.2 Viewing Linux Commissioning Results

- After the **JDBCExample.java** of **doris-jdbc-example** sample is successfully executed, the following information is displayed:

```
2023-08-17 23:13:13,473 | INFO | main | Start execute doris example. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:41)
2023-08-17 23:13:13,885 | INFO | main | Start create database. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:44)
2023-08-17 23:13:13,949 | INFO | main | Database created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:46)
2023-08-17 23:13:13,950 | INFO | main | Start create table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:49)
2023-08-17 23:13:14,132 | INFO | main | Table created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:51)
```

```

2023-08-17 23:13:14,133 | INFO | main | Start to insert data into the table. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:54)
2023-08-17 23:13:14,733 | INFO | main | Inserting data to the table succeeded. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:56)
2023-08-17 23:13:14,733 | INFO | main | Start to query table data. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:59)
2023-08-17 23:13:15,079 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:121)
2023-08-17 23:13:15,079 | INFO | main | c1 c2 c3 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:126)
2023-08-17 23:13:15,079 | INFO | main | 0 0 0 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 1 10 100 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 2 20 200 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 3 30 300 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 4 40 400 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 5 50 500 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 6 60 600 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 7 70 700 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 8 80 800 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 9 90 900 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:61)
2023-08-17 23:13:15,081 | INFO | main | Start to delete the table. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:64)
2023-08-17 23:13:15,114 | INFO | main | Table deleted successfully. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:66)
2023-08-17 23:13:15,124 | INFO | main | Doris example execution successfully. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:71)

```

- After the **DBalancerJDBCExample.java** of **doris-jdbc-example** sample is successfully executed, the following information is displayed:

```

2024-06-12 14:22:08,716 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:131)
2024-06-12 14:22:08,716 | INFO | main | c1 c2 c3 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:136)
2024-06-12 14:22:08,717 | INFO | main | 0 0 0 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:140)
2024-06-12 14:22:08,717 | INFO | main | 1 10 100 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 2 20 200 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 3 30 300 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 4 40 400 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 5 50 500 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 6 60 600 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 7 70 700 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 8 80 800 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 9 90 900 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:66)
2024-06-12 14:22:08,718 | INFO | main | Start to delete the table. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:69)
2024-06-12 14:22:08,836 | INFO | main | Table deleted successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:71)
2024-06-12 14:22:08,836 | INFO | main | Start to delete the database. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:74)
2024-06-12 14:22:08,951 | INFO | main | Database deleted successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:76)
2024-06-12 14:22:08,962 | INFO | main | Doris example execution successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:81)

```

Process finished with exit code 0

- Running result of interconnection between Doris and SpringBoot
  - The result of running the Doris JDBC sample application using SpringBoot is as follows:  
Enter **http://IP address of the node that runs the sample:8080/doris/example/executesql** in the address box of the browser. IDEA prints logs. The following figure shows the returned information.

**Figure 1-30** Doris JDBC returned running information

```

-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----| Doris Example Start -----| -----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----| Start create database. Database created successfully. Start create table. Table created successfully. Start to insert data into the table. Inserting data to the table
-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----| succeeded. Start to query table data. Query result : c1 c2 c3 0 0 1 10 100 2 20 200 3 30 300 4 40 400 5 50 500 6 60 600 7 70 700 8 80 800 9 90 900 Querying table data succeeded. Start to delete the table. Table deleted successfully.
-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----| Doris Example End -----| -----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

- The result of running the Doris DBalancer sample application using SpringBoot is as follows:

**Figure 1-31** Doris DBalancer returned running information

- ```
***** Doris Example Start ***** Start create database. Database created successfully. Start create table. Table created successfully. Start to insert data into the table. Inserting data to the table succeeded. Start to query table data. Query result: c1 c2 c3 0 0 0
10 100 2 20 200 3 30 300 4 40 400 5 50 500 6 60 600 7 70 700 8 80 800 9 90 900 Querying table data succeeded. Start to delete the table. Table deleted successfully. start to delete the database. Database deleted successfully. ***** Doris Example End *****
```
- If the **doris-stream-load-example** sample is successfully executed, the following information is displayed:

```
Start create database.
Database created successfully.
Start create table.
Table created successfully.
{
    "TxnId": 27,
    "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 4,
    "NumberLoadedRows": 4,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 141,
    "LoadTimeMs": 387,
    "BeginTxnTimeMs": 58,
    "StreamLoadPutTimeMs": 137,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 94,
    "CommitAndPublishTimeMs": 95
}
curl{
    "TxnId": 27,
    "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 4,
    "NumberLoadedRows": 4,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 141,
    "LoadTimeMs": 387,
    "BeginTxnTimeMs": 58,
    "StreamLoadPutTimeMs": 137,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 94,
    "CommitAndPublishTimeMs": 95
}
```

Connect to Doris on the MySQL client and run the following command to view the imported data:

```
select * from doris_test_sink;
```

```
mysql> select * from doris_test_sink;
+-----+-----+-----+-----+
| id   | number | price | skuname | skudesc      |
+-----+-----+-----+-----+
| 10001 |    12 | 13.30 | test1   | this is atest
| 10002 |   100 | 15.30 | test2   | this is atest
| 10003 |   102 | 16.30 | test3   | this is atest
| 10004 |   120 | 17.30 | test4   | this is atest
+-----+-----+-----+-----+
4 rows in set (0.02 sec)
```

## 1.7 Elasticsearch Development Guide

### 1.7.1 Overview

#### 1.7.1.1 Application Development Overview

##### Introduction to Elasticsearch

Elasticsearch, a Lucene-powered search server, provides a distributed full-text search and analysis engine with multi-user capabilities. Designed for big data scenarios, Elasticsearch is easy to install and use and supports stable, fast, and reliable real-time search and analysis.

In addition to the basic functions, such as timestamp-based filtering and exact match provided by traditional relational databases, Elasticsearch can perform full-text search, process synonyms, score documents based on the correlation, and generate analysis and aggregation results based on the same data. Data can be processed in real time if a large number of working processes do not exist.

#### 1.7.1.2 Concepts

##### Basic Concepts

- **cluster**

There are multiple nodes in a cluster, one of which is the master node and the others are the slave nodes. Master node and slave node are defined within a cluster. In Elasticsearch, decentralization is posed. That is, there is no "central node" of the Elasticsearch cluster to external systems. Nodes in an Elasticsearch cluster can be logically considered as an entity. In this way, communication with any node in Elasticsearch is equivalent to the communication with the Elasticsearch cluster.

- **shards**

Indicates the index shard. Elasticsearch can divide a complete index into multiple shards. In this way, a large index can be split into multiple shards and distributed to different nodes, implementing distributed search. The number of shards can only be specified before the index is created and cannot be changed after the index is created.

- **replicas**

Indicates the index replica. Elasticsearch allows you to set multiple replicas for an index. Replication is to improve the fault tolerance of the system. When a shard of a node is damaged or lost, the system can restore it using its replica. In addition, replication is to improve the query efficiency of Elasticsearch. Elasticsearch automatically balances the search requests.

- **recovery**

Indicates data restoration or data redistribution. When a node is added or deleted, Elasticsearch redistributes index shards based on the load of the corresponding physical server. When a faulty node is restarted, data restoration is also performed.

- **river**

Indicates a data source of Elasticsearch. It is also a method of synchronizing data from other storage modes (such as the database) to Elasticsearch. **river** is a plug-in Elasticsearch service. The river plug-in reads data from a river data source and indexes it to Elasticsearch. Official rivers include those from CouchDB, RabbitMQ, Twitter, and Wikipedia.

- **gateway**

Indicates the storage mode of an Elasticsearch index snapshot. By default, Elasticsearch stores an index in the memory. When the memory is full, Elasticsearch persistently saves the index to the local hard disk. A gateway stores index snapshots. When the corresponding Elasticsearch cluster is stopped and then restarted, the index backup data is read from the gateway. Elasticsearch supports multiple types of gateways, including local file systems (default), distributed file systems, and Hadoop HDFS.

- **discovery.zen**

Indicates the automatic node discovery mechanism of Elasticsearch. Elasticsearch is a P2P system. It searches for existing nodes through broadcast and then communicates with nodes through multicast protocols. Elasticsearch supports P2P interaction.

- **Index**

Indicates the data storage mode of Elasticsearch and is similar to a table in relational database .

- **Document**

Is similar to the row of a relational database. Each document of the same index has a unique ID.

- **Field**

Is similar to the column in a relational database, which is the smallest unit of Elasticsearch data storage.

### 1.7.1.3 Application Development Process

[Figure 1-32](#) shows the Elasticsearch application development process.

Figure 1-32 Elasticsearch application development process

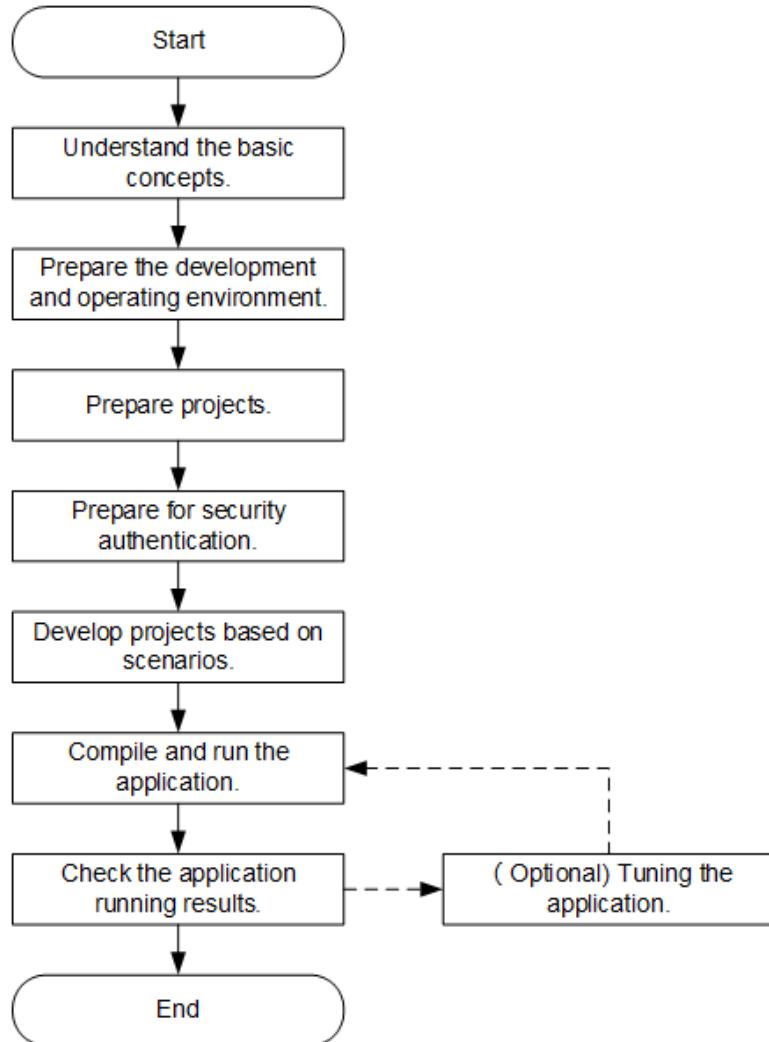


Table 1-12 Description of the Elasticsearch application development process

| Phase                                             | Description                                                                                                                                                                                                                                            | Reference                                                       |
|---------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| Understand basic concepts.                        | Before application development, learn basic concepts of Elasticsearch and understand scenario requirements.                                                                                                                                            | <a href="#">Concepts</a>                                        |
| Development Environment and Operating Environment | Elasticsearch applications support RESTful development using Java. You are advised to use the IDEA tool to configure development environments. Elasticsearch runs on an Elasticsearch client. Install and configure the client according to the guide. | <a href="#">Preparing Development and Operating Environment</a> |

| Phase                                    | Description                                                                                                                                                                                    | Reference                                                 |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| Prepare a project.                       | Elasticsearch provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create an Elasticsearch project according to the guide. | <a href="#">Configuring and Importing Sample Projects</a> |
| Develop a project based on the scenario. | Provide a sample project using Java language, covering an entire process for sample projects from query index to deletion index.                                                               | <a href="#">Development Process</a>                       |
| Compile and run applications .           | You can compile the developed application and submit it for running.                                                                                                                           | <a href="#">Commissioning Process</a>                     |
| View application running results.        | Application running results are stored under a user-defined path. You can also view the application running status on the UI.                                                                  | <a href="#">Commissioning Process</a>                     |

## 1.7.2 Environment Preparation

### 1.7.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

**Table 1-13** describes the development environment and operating environment required for application development.

**Table 1-13** Development environment

| Item | Description                                                                                                                                                                                                                                                                                                           |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OS   | <ul style="list-style-type: none"><li>• Development environment: Windows (Windows 7 or later is supported.)</li><li>• Operating environment: Windows or Linux. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul> |

| Item                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Installing JDK                           | <p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul> |
| Installing and configuring IntelliJ IDEA | Tool used for developing GraphBase applications. The version must be 2019.1 or other compatible version. <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li><li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li><li>• If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Installing the JUnit plug-in             | Basic configurations of the development environment.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

| Item                         | Description                                                                                                                                  |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Installation of Maven        | Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development. |
| User development preparation | See <a href="#">Preparing a Developer Account</a> for configuration.                                                                         |
| 7-zip                        | It is a tool used to decompress *.zip and *.rar files. The 7-Zip 16.04 is supported.                                                         |

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) and choose **Homepage > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Elasticsearch\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the Elasticsearch sample project. The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\ Elasticsearch\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Elasticsearch/config/* root@IP address of the client node:/opt/hadoopclient/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory.
  - c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

### 1.7.2.2 Configuring and Importing Sample Projects

### 1.7.2.2.1 Overview

#### Background

Obtain the Elasticsearch development sample project. Import the project to IntelliJ IDEA for learning.

#### Prerequisites

Ensure that the time difference between the local PC and the cluster is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. Time of the cluster can be viewed in the lower-right corner on FusionInsight Manager.

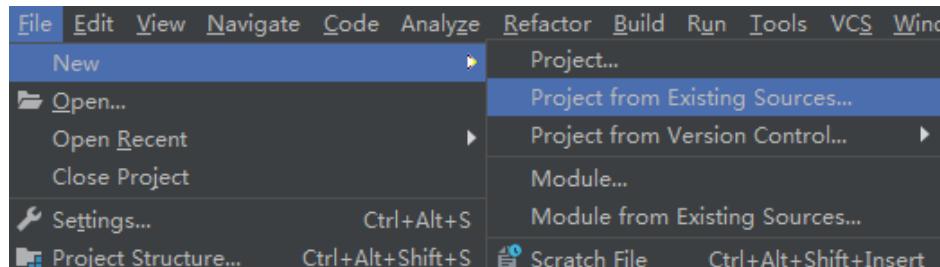
### 1.7.2.2 Importing a RestClient Sample Project

#### Procedure

**Step 1** Obtain the sample project folder **elasticsearch-rest-client-example** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **elasticsearch-rest-client-example** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

**Step 3** Obtain the authentication credential file of the new user. Obtain the **user.keytab** and **krb5.conf** files during [Preparing a Developer Account](#) and copy them to the **conf** directory of the sample project for security authentication in the sample project.

**Step 4** Find the configuration file **es-rest-client-example.properties** exists in the client folder **Elasticsearch\config** directory. Copy the parameter values in the file to the **esParams.properties** file in the **conf** directory of the sample project.

**Step 5** In the IntelliJ IDEA development environment, open the **esParams.properties** file in the **conf** directory of the sample project, and change the values of the parameters listed in [Table 1-14](#) as required.

**Table 1-14** Configuration table

| Parameter                | Default Value                                     | Description                                                                                                                                                                                                                                                    |
|--------------------------|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| esServerHost             | Examples:<br>ip1:port1,ip2:port2<br>,ip3:port3... | Specifies the list of combinations of the IP addresses of nodes in the Elasticsearch cluster and the HTTP ports of the Elasticsearch instances installed on the nodes, except EsMaster instances.                                                              |
| connectTimeout           | 5000                                              | Specifies the timeout interval of the connection between the client and the server, in milliseconds.                                                                                                                                                           |
| socketTimeout            | 60,000                                            | Specifies the server response obtaining timeout interval of the client, in milliseconds.                                                                                                                                                                       |
| connectionRequestTimeout | 100000                                            | Indicates that the connection timeout interval is obtained from the connection pool, in milliseconds.                                                                                                                                                          |
| maxConnPerRoute          | 100                                               | Maximum number of connections per route.                                                                                                                                                                                                                       |
| maxConnTotal             | 1000                                              | Maximum number of connections                                                                                                                                                                                                                                  |
| isSecureMode             | true                                              | Indicates whether the client is in security mode. The value <b>true</b> indicates that the security mode is enabled, and the value <b>false</b> indicates that the normal mode is enabled.                                                                     |
| sslEnabled               | true                                              | Indicates whether SSL encryption is enabled on the client. The value <b>true</b> indicates that SSL encryption is enabled, and the value <b>false</b> indicates that SSL encryption is disabled. The value must be the same as that configured in the cluster. |
| principal                | esuser                                            | Specifies the authentication subject. The recommended format is "username". You can also use the format of "username@system domain name".                                                                                                                      |
| snifferEnable            | true                                              | Indicates whether the Rest client enables the sniffing function. The value <b>true</b> indicates that the sniffing function is enabled, and the value <b>false</b> indicates that the sniffing function is disabled.                                           |
| customJaasPath           | null                                              | User-defined path of the <b>jaas.conf</b> file, which must contain specific file name. If this parameter is not set, the system automatically generates a path.                                                                                                |

**Step 6** Set **isSecureMode** in the **esParams.properties** file in the **conf** directory of the sample project to **true** and enable the security mode.

 NOTE

- If an error is reported during sample code importing, use a later IntelliJ IDEA version.
- The HTTP request function of the EsMaster instance is disabled. The value of the **esServerHost** parameter does not contain the EsMaster instance, that is, the 24148 port. Otherwise, the request fails.
- Set **esServerHost** to the list of combinations. The combinations contain the IP addresses of all nodes in the Elasticsearch cluster and the HTTP ports of all Elasticsearch instances installed on the nodes, for example, ip1: Port1, ip2:port2, ip3:port3. You can query the port number as follows: On FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Elasticsearch > Configurations > All Configurations**. Click **Port** of the corresponding instance and check the value of **SERVER\_PORT**.
- To check whether the Elasticsearch cluster is in security mode or normal mode, log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Elasticsearch > Configurations > All Configurations**, and search for **ELASTICSEARCH\_SECURITY\_ENABLE** parameter. If this parameter can be queried and its value is **true**, the Elasticsearch cluster is in security mode. If this parameter is not found or its value is **false**, the Elasticsearch cluster is in normal mode.
- You can log in to the Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**, which is the current system domain name. For example, if the cluster domain name is **9427068F-6EFA-4833-B43E-60CB641E5B6C.COM**, the value of **principal** is **esuser@9427068F-6EFA-4833-B43E-60CB641E5B6C.COM**.
- You can customize the file path of **jaas.conf** by setting the **customJaasPath** parameter. If there is no such requirement, you can leave this parameter empty. The system automatically generates a path.

----End

### 1.7.2.2.3 Importing a SpringBoot Sample Project

#### Scenario

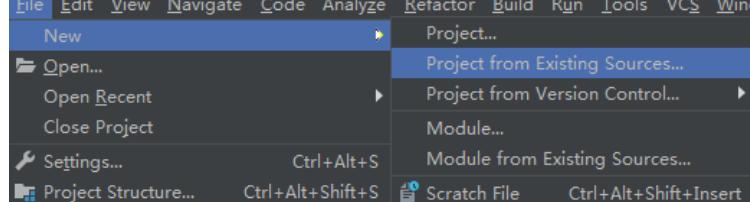
To run the SpringBoot interface sample code of the Elasticsearch component of MRS, perform the following operations.

The following example develops an application that uses SpringBoot to connect to Impala in the Windows environment.

#### Procedure

- Step 1** Obtain the sample project folder **elasticsearch-rest-client-example** in the **src/springboot/elasticsearch-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the sample project to the IntelliJ IDEA development environment in the application development environment.

1. Choose **File > New > Project from Existing Sources....**



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **elasticsearch-rest-client-example** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.

Select the recommended JDK version and click **Finish**.

- Step 3** Obtain the authentication credential file of the new user. Obtain the **user.keytab** and **krb5.conf** files obtained during [Preparing a Developer Account](#) and copy them to the **conf** directory of the sample project for security authentication.
- Step 4** Find configuration file **es-rest-client-example.properties** in the **Elasticsearch \config** directory of the client folder. Copy the parameter values in the file to the **esParams.properties** file in the **conf** directory of the sample project.
- Step 5** In the IntelliJ IDEA development environment, open the **esParams.properties** file in the **conf** directory of the sample project, and change the values of the parameters listed in [Table 1-15](#).

**Table 1-15** Configuration table

| Parameter                | Default Value                    | Description                                                                                                                                                                                                             |
|--------------------------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| esServerHost             | ip1:port1,ip2:port2,ip3:port3... | List of combinations of the IP addresses of nodes in the Elasticsearch cluster and the HTTP ports of the Elasticsearch instances installed on the nodes, except EsMaster instances.                                     |
| connectTimeout           | 5000                             | Timeout interval of the connection between the client and the server, in milliseconds.                                                                                                                                  |
| socketTimeout            | 60000                            | Server response obtaining timeout interval of the client, in milliseconds.                                                                                                                                              |
| connectionRequestTimeout | 100000                           | Connection timeout interval is obtained from the connection pool, in milliseconds.                                                                                                                                      |
| maxConnPerRoute          | 100                              | Maximum number of connections on each route                                                                                                                                                                             |
| maxConnTotal             | 1000                             | Maximum number of connections                                                                                                                                                                                           |
| isSecureMode             | true                             | Whether the client is in security mode. The value <b>true</b> indicates that the security mode is enabled, and the value <b>false</b> indicates that the normal mode is enabled.                                        |
| sslEnabled               | true                             | Whether to enable SSL encryption on the client. <b>true</b> indicates that encryption is enabled, and <b>false</b> indicates that encryption is disabled. The value must be the same as that configured in the cluster. |

| Parameter      | Default Value | Description                                                                                                                                                                                                |
|----------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| principal      | esuser        | Authentication subject. The options are as follows: <ul style="list-style-type: none"><li>• <b>Username</b> (recommended)</li><li>• <i>Username@System domain name</i></li></ul>                           |
| snifferEnable  | true          | Whether the REST Client enables the sniffing function. The value <b>true</b> indicates that the sniffing function is enabled, and the value <b>false</b> indicates that the sniffing function is disabled. |
| customJaasPath | N/A           | User-defined path of the <b>jaas.conf</b> file, which must contain specific file name. If this parameter is not set, the system automatically generates a path.                                            |

**Step 6** Set **isSecureMode** in the **esParams.properties** file in the **conf** directory of the sample project to **true** and enable the security mode.

 **NOTE**

- If an error is reported during sample code importing, use IntelliJ IDEA of a later version.
- The HTTP request function of the EsMaster instance is disabled. The value of the **esServerHost** parameter does not contain the EsMaster instance, that is, the 24148 port. Otherwise, the request fails.
- Set **esServerHost** to a list of combinations of IP addresses of nodes in the installed Elasticsearch cluster and HTTP ports of Elasticsearch instances installed on the nodes, for example, *ip1:port1,ip2:port2,ip3:port3...*. To query the port number, choose **Cluster > Services > Elasticsearch** on FusionInsight Manager, click **Configuration**, click **All Configurations**, click **Port** of the corresponding instance, and check the value of **SERVER\_PORT**.
- To check whether the current Elasticsearch cluster is in security mode or common mode, log in to FusionInsight Manager, choose **Cluster > Services > Elasticsearch**, click **Configuration**, click **All Configurations**, and search for the **ELASTICSEARCH\_SECURITY\_ENABLE** parameter. If the parameter can be queried and its value is **true**, the Elasticsearch cluster is in security mode. If this parameter cannot be queried or the value of this parameter is **false**, the Elasticsearch cluster is in normal mode.
- *System domain name*: log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**. For example, if **Local Domain** is set to *XXX.COM*, the value of **principal** is **esuser@XXX.COM**.
- You can set the **customJaasPath** parameter to customize the path of the **jaas.conf** file. Leave this parameter blank if it is not required, and the system automatically generates a path.

----End

### 1.7.2.3 Preparing for Security Authentication

#### Scenarios

In a security cluster environment, the components must be mutually authenticated before communicating with each other to ensure communication

security. Spnego security authentication is required for Elasticsearch application development. Spnego authentication is a security protocol expending Kerberos and using GSS-API authentication mechanism. Security authentication is typically code-based. The **krb5.conf**, and **user.keytab** files are used together to support the Oracle Java and IBM Java platforms.

**krb5.conf** and **user.keytab** are obtained in **Preparing a Development Account** and are stored in the **conf** directory of the sample project.

 NOTE

On the RestClient client, change the value of **principal** in the **esParams.properties** file in the **conf** directory of the sample project to *new username@<system domain name>*.

## 1.7.3 Development Process

### 1.7.3.1 Typical Application Scenarios

You can quickly learn and master the Elasticsearch development process and know key API functions in a typical application scenario.

#### Scenario Description

Assume that you want to develop an application to search for all library information, provide information about books related to search keywords, and grade the books by score. The search function can be implemented by Elasticsearch. The search process is as follows:

1. Connecting the client to a cluster
2. Querying the health status
3. Checking whether a specified index exists
4. Creating an index for specifying the number of shards
5. Writing index data
6. Writing data in batches
7. Batch write data to specified routes
8. Querying index information
9. Deleting an index
10. Deleting documents in an index
11. Refreshing an index
12. Using a multi-thread sample for query
13. Pre-sorting indices

### 1.7.3.2 Low Level Rest Client Sample Code

### 1.7.3.2.1 Connecting the Elasticsearch Client to an Elasticsearch Cluster

#### Function Description

Before using APIs provided by Elasticsearch, you need to obtain the Elasticsearch client. Connect the Elasticsearch client to the desired Elasticsearch cluster by setting the IP address and port number of the cluster.

##### NOTE

Call **RestClient.close()** to close requested resources after the Elasticsearch operation is complete.

```
public static void main(String[] args) {
    HwRestClient hwRestClient = new HwRestClient();
    RestClient restClient = hwRestClient.getRestClient();
    try {
        /...
    } catch (Exception e) {
        LOG.error("There are exceptions occurred.", e);
    } finally {
        if (restClient != null) {
            try {
                restClient.close();
                LOG.info("Close the client successful.");
            } catch (Exception e1) {
                LOG.error("Close the client failed.", e1);
            }
        }
    }
}
```

##### NOTE

- By default, the HwRestClient reads the following configuration files from the **conf** directory in the code running path: **esParams.properties**, **krb5.conf**, and **user.keytab**.
- The configuration file path can be customized. For example, if the code runs in the Windows operating system, and the configuration files are stored in the root directory of drive D, you can set **HwRestClient hwRestClient** to **new HwRestClient("D:\\")**.

### 1.7.3.2.2 Querying Cluster Information

#### Function

**QueryClusterInfo.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/cluster** directory is used to query information about the Elasticsearch cluster.

```
/*
 * Query the cluster's information
 */
public static void queryClusterInfo(RestClient restClientTest) {

    Response response;
    try {
        Request request=new Request("GET", "/_cluster/health");
        //Process the returned result to display it more intuitively
        request.addParameter("pretty", "true");
        response = restClientTest.performRequest(request);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("QueryClusterInfo successful.");
        } else {
            LOG.error("QueryClusterInfo failed.");
        }
    }
```

```
        }
        LOG.info("QueryClusterInfo response entity is : {}. ",EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
        LOG.error("QueryClusterInfo failed, exception occurred.", e);
    }
}
```

### 1.7.3.2.3 Checking Whether the Specified Index Exists

#### Function

**IndexExist.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/exist** directory is used to check whether the specified index exists in the Elasticsearch cluster.

```
/*
 * Check the existence of the index or not.
 */
public static void boolean(RestClient restClientTest, String index) {

    Response response;
    try {
        Request request = new Request("HEAD", "/" + index);
        response = restClientTest.performRequest(request);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("***** Check index successful,index is exist : " + index + " *****");
        }
        if (HttpStatus.SC_NOT_FOUND == response.getStatusLine().getStatusCode()) {
            LOG.info("Check index successful,index is exist : {}.");
        }
        return true;
    }
    if (HttpStatus.SC_NOT_FOUND == response.getStatusLine().getStatusCode()) {
        LOG.info("Index is not exist : {}.");
    }
    return false;
} catch (Exception e) {
    LOG.error("Check index failed, exception occurred.", e);
}
return false;
}
```

### 1.7.3.2.4 Creating an Index with the Desired Number of Shards

#### Function

**CreateIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/index** directory is used to create an index of numbers of specified primary shards and replica shards.

Set variables *shardNum* and *replicaNum* in the following code, that is, the number of primary shards and the number of replica shards.

```
/*
 * Create one index with customized shard number and replica number.
 */
public static void createIndexWithShardNum(RestClient restClientTest, String index) {

    Response rsp;
    int shardNum = 3;
    int replicaNum = 1;
```

```

String jsonString =
    "{" + "\"settings\":{\":" + "\"number_of_shards\":\"" + shardNum + "\",\"" + "\"number_of_replicas\":\""
+ replicaNum + "\"}}";
    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
try {
    Request request = new Request("PUT", "/" + index);
    request.addParameter("pretty", "true");
    request.setEntity(entity);
    rsp = restClientTest.performRequest(request);
    if (HttpStatus.SC_OK == rsp.getStatusLine().getStatusCode()) {
        LOG.info("CreateIndexWithShardNum successful.");
    } else {
        LOG.error("CreateIndexWithShardNum failed.");
    }
    LOG.info("CreateIndexWithShardNum response entity is : {}.", EntityUtils.toString(rsp.getEntity()));
} catch (Exception e) {
    LOG.error("CreateIndexWithShardNum failed, exception occurred.", e);
}
}

```

### 1.7.3.2.5 Writing Index Data

#### Function

**PutData.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/index** directory is used to insert data into a specified index, to update the index.

Variable **jsonString** of **String** indicates the data to be inserted. The value is user-defined.

```

/**
 * Write one document into the index
 */
public static void putData(RestClient restClientTest, String index, String type, String id) {

    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    esMap.put("name", "Happy");
    esMap.put("author", "Alex Yang");
    esMap.put("pubinfo", "Beijing,China");
    esMap.put("pubtime", "2020-01-01");
    esMap.put("description", "Elasticsearch is a highly scalable open-source full-text search and analytics
engine.");
    String jsonString = gson.toJson(esMap);

    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
    Response rsp;

    try {
        Request request = new Request("POST", "/" + index + "/" + type + "/" + id);
        request.addParameter("pretty", "true");
        request.setEntity(entity);
        rsp = restClientTest.performRequest(request);
        if (HttpStatus.SC_OK == rsp.getStatusLine().getStatusCode() || HttpStatus.SC_CREATED ==
rsp.getStatusLine().getStatusCode()) {
            LOG.info("PutData successful.");
        } else {
            LOG.error("PutData failed.");
        }
        LOG.info("PutData response entity is : {}.", EntityUtils.toString(rsp.getEntity()));
    } catch (Exception e) {
        LOG.error("PutData failed, exception occurred.", e);
    }
}

```

### 1.7.3.2.6 Writing Data in Batches

#### Function

**Bulk.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/bulk` directory is used to insert data into a specified index in batches.

**totalRecordNumber** indicates the total number of documents to be inserted, and **oneCommit** indicates the number of documents to be inserted at a time. Data added to **esMap** is the data to be written into the index.

```
/*
 * Send a bulk request
 */
private static void bulk(RestClient restClientTest, String index) {
    //Total number of documents to be written
    long totalRecordNum = 10000;
    //Number of documents written in a bulk
    long oneCommit = 1000;
    long circleNumber = totalRecordNum / oneCommit;
    StringEntity entity;
    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    String str = "{ \"index\" : { \"_index\" : \"\" + index + "\" } }";
    for (int i = 1; i <= circleNumber; i++) {
        StringBuilder builder = new StringBuilder();
        for (int j = 1; j <= oneCommit; j++) {
            esMap.clear();
            esMap.put("name", "Linda");
            esMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
            esMap.put("height", (float)ThreadLocalRandom.current().nextInt(140, 220));
            esMap.put("weight", (float)ThreadLocalRandom.current().nextInt(70, 200));
            esMap.put("cur_time", System.currentTimeMillis());
            String strJson = gson.toJson(esMap);
            builder.append(str).append("\n");
            builder.append(strJson).append("\n");
        }
        entity = new StringEntity(builder.toString(), ContentType.APPLICATION_JSON);
        entity.setContentType("UTF-8");
        Response response;
        try {
            Request request = new Request("PUT", "/_bulk");
            request.addParameter("pretty", "true");
            request.setEntity(entity);
            response = restClientTest.performRequest(request);
            if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
                LOG.info("Already input documents : " + oneCommit * i);
            } else {
                LOG.error("Bulk failed.");
            }
            LOG.info("Bulk response entity is : " + EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            LOG.error("Bulk failed, exception occurred.", e);
        }
    }
}
```

### 1.7.3.2.7 Batch write data to specified routes

#### Function

**BulkRoutingSample.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/bulk` directory

is used to set routing for each request in the bulk, route documents with the same routing to the same segment, and specify routing during query to narrow the query range and improve the query speed.

```
/*
 * Number of documents to be written.
 */
private static final int NUM_OF_DOC = 100;

/**
 * Use bulk to write data and specify routing.
 */
private static void bulkWithRouting(RestClient restClient, String indexName) {
    StringEntity entity;
    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    String str = "{ \"index\" : { \"_index\" : \"\" + indexName + "\"}";
    StringBuilder sb = new StringBuilder();
    int age;
    for (int i = 1; i <= NUM_OF_DOC; i++) {
        esMap.clear();
        esMap.put("user", "Linda" + i);
        age = ThreadLocalRandom.current().nextInt(18, 100);
        esMap.put("age", age);
        esMap.put("postDate", "2020-01-01");
        esMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
        esMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
        String strJson = gson.toJson(esMap);
        // Route documents of the same age to the same shard based on the age.
        sb.append(str).append("\"routing\":\"").append(age).append("\"\")").append("\n");
        sb.append(strJson).append("\n");
    }
    entity = new StringEntity(sb.toString(), ContentType.APPLICATION_JSON);
    entity.setContentType("UTF-8");
    Response response;
    try {
        Request request = new Request("PUT", "_bulk");
        request.addParameter("pretty", "true");
        request.setEntity(entity);
        response = restClient.performRequest(request);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("Bulk routing is successful.");
        } else {
            LOG.error("Bulk routing is failed.");
        }
    } catch (IOException e) {
        LOG.error("Bulk routing is failed, exception occurred.", e);
    }
}
```

### 1.7.3.2.8 Querying Index Information

#### Function

**QueryData.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/search** directory is used to query information about a document with specified **index**, **type**, and **id**.

```
/*
 * Query all data of one index.
 */
public static void queryData(RestClient restClientTest, String index, String type, String id){

    Response response;
    try {
        Request request = new Request("GET", "/" + index + "/" + type + "/" + id);
        request.addParameter("pretty", "true");
    }
```

```
response = restClientTest.performRequest(request);
if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
    LOG.info("QueryData successful.");
} else {
    LOG.error("QueryData failed.");
}
LOG.info("QueryData response entity is : {}.", EntityUtils.toString(response.getEntity()));
} catch (Exception e) {
    LOG.error("QueryData failed, exception occurred.", e);
}
}
```

### 1.7.3.2.9 Deleting an Index

#### Function

**DeleteIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/delete** directory is used to delete a specified index.

```
/**
 * Delete one index
 */
public static void deleteIndex(RestClient restClientTest, String index) {

    Response response;
    try {
        Request request = new Request("DELETE", "/" + index + "?&pretty=true");
        response = restClientTest.performRequest(request);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("DeleteIndex successful.");
        } else {
            LOG.error("DeleteIndex failed.");
        }
        LOG.info("DeleteIndex response entity is : {}.", EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
        LOG.error("DeleteIndex failed, exception occurred.", e);
    }
}
```

### 1.7.3.2.10 Deleting Some Documents in the Index

#### Function

**DeleteSomeDocumentsInIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/delete** directory is used to refresh a specified index.

```
/**
 * Delete some documents by query in one index
 */
public static void deleteSomeDocumentsInIndex(RestClient restClientTest, String index, String field, String value) {
    String jsonString =
        "\n" +
        "  \"query\": {\n" +
        "    \"match\": { \"" + field + "\": \"" + value + "\"}\n" +
        "  }\n";
    Map<String, String> params = Collections.singletonMap("pretty", "true");
    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
    Response rsp;
    try {
        rsp = restClientTest.performRequest("POST", "/" + index + "/_delete_by_query", params, entity);
        if (HttpStatus.SC_OK == rsp.getStatusLine().getStatusCode()) {
            LOG.info("DeleteSomeDocumentsInIndex successful.");
        }
    }
```

```
        else {
            LOG.error("DeleteSomeDocumentsInIndex failed.");
        }
        LOG.info("DeleteSomeDocumentsInIndex response entity is : " +
EntityUtils.toString(rsp.getEntity()));
    }
    catch (Exception e) {
        LOG.error("DeleteSomeDocumentsInIndex failed, exception occurred.", e);
    }
}
```

### 1.7.3.2.11 Deleting All Documents in the Index

#### Function

**DeleteAllDocumentsInIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/delete** directory is used to refresh a specified index.

```
/**
 * Delete all documents by query in one index
 */
public static void deleteAllDocumentsInIndex(RestClient restClientTest, String index) {

    String jsonString = "{\n" + " \\"query\\": {\n" + " \\"match_all\\": {}\n" + " }\n" + "}";

    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
    Response response;
    try {
        response = restClientTest.performRequest("POST", "/" + index + "/_delete_by_query", params,
entity);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("DeleteAllDocumentsInIndex successful.");
        } else {
            LOG.error("DeleteAllDocumentsInIndex failed.");
        }
        LOG.info("DeleteAllDocumentsInIndex response entity is :
{}.", EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
        LOG.error("DeleteAllDocumentsInIndex failed, exception occurred.", e);
    }
}
```

### 1.7.3.2.12 Refreshing an Index

#### Refreshing a Single Index

**Flush.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/flush** directory is used to refresh a specified index.

```
/**
 * Flush one index data to storage and clearing the internal transaction log
 */
public static void flushOneIndex(RestClient restClientTest, String index) {

    Response flushRsp;
    try {
        Request request=new Request("POST", "/" + index + "/_flush");
        request.addParameter("pretty", "true");
        flushRsp = restClientTest.performRequest(request);
        LOG.info(EntityUtils.toString(flushRsp.getEntity()));
    }
}
```

```
if (HttpStatus.SC_OK == flushRsp.getStatusLine().getStatusCode()) {
    LOG.info("Flush one index successful.");
} else {
    LOG.error("Flush one index failed.");
}
LOG.info("Flush one index response entity is : {}", EntityUtils.toString(flushRsp.getEntity()));
} catch (Exception e) {
    LOG.error("Flush one index failed, exception occurred.", e);
}
```

## Refreshing all Indexes

**Flush.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/flush** directory is used to refresh all indexes.

```
/*
 * Flush all indexes data to storage and clearing the internal transaction log
 * The user who performs this operation must belong to the "supergroup" group.
 */
private static void flushAllIndexes(RestClient restClientTest) {
    Map<String, String> params = Collections.singletonMap("pretty", "true");
    Response flushRsp;
    try {
        flushRsp = restClientTest.performRequest("POST", "/_all/_flush", params);
        if (HttpStatus.SC_OK == flushRsp.getStatusLine().getStatusCode()) {
            LOG.info("Flush all indexes successful.");
        } else {
            LOG.error("Flush all indexes failed.");
        }
        LOG.info("Flush all indexes response entity is : " + EntityUtils.toString(flushRsp.getEntity()));
    } catch (Exception e) {
        LOG.error("Flush all indexes failed, exception occurred.", e);
    }
}
```

### 1.7.3.2.13 Multi-Thread Example

#### Function

**MultithreadRequest.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/multithread** directory is used to create a thread class **sendRequestThread**, rewrite a **run** method, implement a **bulk** request, and write data in batches.

```
/*
 * Thread that sends bulk request
 */
public static class sendRequestThread implements Runnable {
    private RestClient restClientTh;
    private HwRestClient hwRestClient;
    @Override
    public void run() {
        LOG.info("Thread begin.");
        try {
            hwRestClient = new HwRestClient();
            restClientTh = hwRestClient.getRestClient();
            LOG.info("Thread name: " + Thread.currentThread().getName());
            bulk(restClientTh, "huawei1", "type1");
        } catch (Exception e) {
            LOG.error("SendRequestThread has exception.", e);
        } finally {
            if (restClientTh != null) {
```

```
        try {
            restClientTh.close();
            LOG.info("Close the client successful in thread : " + Thread.currentThread().getName());
        } catch (IOException e) {
            LOG.error("Close the client failed.", e);
        }
    }
}
```

In the main method, the **fixedThreadPool** thread pool is created. By specifying **threadPoolSize** and **jobNumber**, multiple threads are supported to send **bulk** requests and data can be concurrently written in batches.

```
public static void main(String[] args) throws Exception {  
    LOG.info("Start to do multithread request !");  
    ExecutorService fixedThreadPool;  
    HwRestClient hwRestClient = new HwRestClient();  
    RestClient restClientThNew;  
    restClientThNew = hwRestClient.getRestClient();  
    int threadPoolSize = 2;  
    int jobNumber = 5;  
    try {  
        fixedThreadPool = Executors.newFixedThreadPool(threadPoolSize);  
        for (int i = 0; i < jobNumber; i++) {  
            fixedThreadPool.execute(new sendRequestThread());  
        }  
        fixedThreadPool.shutdown();  
    } catch (Exception e) {  
        LOG.error("There are exceptions in sendRequestThread.", e);  
    } finally {  
        if (restClientThNew != null) {  
            try {  
                restClientThNew.close();  
                LOG.info("Close the client successful.");  
            } catch (Exception e1) {  
                LOG.error("Close the client failed.", e1);  
            }  
        }  
    }  
}
```

#### 1.7.3.2.14 Pre-sorting Indices

## Function Description

The `IndexSorting.java` file in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/index` directory is used to configure the fields to be sorted when an index is created, implement the `bulk` request, write data in batches, and query the top N documents.

During index creation, specify the fields to be sorted and specify whether each field is sorted in descending or ascending order.

```
private static void createIndexWithSorting(RestClient restClient, String index) {  
    Response rsp;  
    String jsonString;  
    jsonString = "{\"settings\":{\"number_of_shards\":\"3\", \"number_of_replicas\":\"1\",  
        + \"sort.field\": [\"height\",\"weight\"],\"sort.order\": [\"desc\",\"asc\"]},\"mappings\":{  
        + \"properties\":{\"height\":{\"type\":\"float\"},\"weight\":{\"type\":\"float\"}}}}";  
    HttpEntity entity = new StringEntity(jsonString, ContentType.APPLICATION_JSON);  
    try {
```

```

Request request = new Request("PUT", "/" + index);
request.addParameter("pretty", "true");
request.setEntity(entity);
rsp = restClient.performRequest(request);
if (rsp.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
    LOG.info("Create index with sorting successfully.");
} else {
    LOG.error("Create index with sorting failed.");
}
LOG.info("Create index with sorting response entity is {}.", EntityUtils.toString(rsp.getEntity()));
} catch (IOException | ParseException e) {
    LOG.error("Failed to create index with sorting, exception occurred.", e);
}
}

```

Then, write data in batches.

```

public static void bulk(RestClient restClientTest, String index) {
    // Total number of documents to be written
    long totalRecordNum = 10000;
    // Number of documents written in each bulk. The recommended size is 5 MB to 15 MB.
    long oneCommit = 1000;
    long circleNumber = totalRecordNum / oneCommit;
    StringEntity entity;
    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    String str = "{ \"index\" : { \"_index\" : \"" + index + "\" } }";

    for (int i = 1; i <= circleNumber; i++) {
        StringBuilder builder = new StringBuilder();
        for (int j = 1; j <= oneCommit; j++) {
            esMap.clear();
            esMap.put("name", "Linda");
            esMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
            esMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
            esMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
            esMap.put("cur_time", System.currentTimeMillis());

            String strJson = gson.toJson(esMap);
            builder.append(str).append(System.lineSeparator());
            builder.append(strJson).append(System.lineSeparator());
        }
        entity = new StringEntity(builder.toString(), ContentType.APPLICATION_JSON);
        entity.setContentEncoding("UTF-8");
        Response response;
        try {
            Request request = new Request("PUT", "/_bulk");
            request.addParameter("pretty", "true");
            request.setEntity(entity);
            response = restClientTest.performRequest(request);
            if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
                LOG.info("Already input documents : {}.", oneCommit * i);
            } else {
                LOG.error("Bulk failed.");
            }
            LOG.info("Bulk response entity is : {}.", EntityUtils.toString(response.getEntity()));
        } catch (IOException | ParseException e) {
            LOG.error("Bulk failed, exception occurred.", e);
        }
    }
}

```

Perform the refresh operation so that the data can be queried after being written in batches.

```

private static void refresh(RestClient restClient, String index) {
    try {
        Map<String, String> params = Collections.singletonMap("pretty", Boolean.TRUE.toString());
        Response rsp = restClient.performRequest(buildRequest("POST", "/" + index + "/_refresh", params,
null));
        if (rsp.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {

```

```
        LOG.info("Refresh sorting index successfully.");
    }
} catch (IOException e) {
    LOG.error("Failed to refresh sorting index, exception occurred.", e);
}
}
```

Then, you can query the sorting field. By default, the top 10 documents are obtained.

```
private static void queryData(RestClient restClient, String index) {
    Response response;
    String jsonStr;
    jsonStr = "{\"sort\":{\"height\":{“desc”:”desc\\”, “weight”:“asc\\”}},“track_total_hits”: false}”;
    StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
    try {
        Request request = new Request("GET", "/" + index + "/_search");
        request.addParameter("pretty", "true");
        request.setEntity(entity);
        response = restClient.performRequest(request);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            LOG.info("Query sorting index successfully.");
        } else {
            LOG.error("Failed to query sorting index.");
        }
        LOG.info("Query sorting index response entity is {}.", EntityUtils.toString(response.getEntity()));
    } catch (IOException | ParseException e) {
        LOG.error("Failed to query sorting index, exception occurred.", e);
    }
}
```

### 1.7.3.3 High Level Rest Client Sample Code

#### 1.7.3.3.1 Connecting the Client to a Cluster

## Function

You can obtain the client and connect to a specific Elasticsearch cluster by setting the IP address and port. This is necessary before you use the API provided by the Elasticsearch.



Call **RestHighLevelClient.close()** to close requested resources after the Elasticsearch service operation is complete.

```
public static void main(String[] args) {
    RestHighLevelClient highLevelClient = null;
    HwRestClient hwRestClient = new HwRestClient();
    try {
        highLevelClient = new RestHighLevelClient(hwRestClient.getRestClientBuilder());
        /...
    } finally {
        try {
            if (highLevelClient != null) {
                highLevelClient.close();
            }
        } catch (IOException e) {
            LOG.error("Failed to close RestHighLevelClient.", e);
        }
    }
}
```

#### NOTE

- By default, the HwRestClient reads the following configuration files from the **conf** directory in the code running path: **esParams.properties**, **krb5.conf**, and **user.keytab**.
- The configuration file path can be customized. For example, if the code runs in the Windows operating system, and the configuration files are stored in the root directory of drive D, you can set **HwRestClient hwRestClient** to **new HwRestClient("D:\\")**.

### 1.7.3.3.2 Querying Cluster Status Information

#### Function

**QueryClusterInfo.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/cluster** directory is used to query information about the Elasticsearch cluster.

```
/*
 * Get cluster information
 */
public static void queryClusterInfo(RestHighLevelClient highLevelClient) {

    try {
        MainResponse response = highLevelClient.info(RequestOptions.DEFAULT);
        ClusterName clusterName = response.getClusterName();
        LOG.info("ClusterName:[{}], clusterUuid:[{}], nodeName:[{}], version:[{}].", new Object[]
        { clusterName.value(),
            response.getClusterUuid(), response.getNodeName(), response.getVersion().toString() });
    } catch (Exception e) {
        LOG.error("QueryClusterInfo is failed,exception occurred.", e);
    }
}
```

### 1.7.3.3.3 Writing Index Data

Data can be written to a specified index by specifying data sources in different formats, such as a character string in the JSON format, a data source in the Map format, and a data source in the XContentBuilder format.

#### Writing Data by Using the Character String in the JSON Format

**IndexByJson.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index** directory is used to write data into a specified index by using a data source in the JSON format.

The variable *jsonString* is data to be inserted and can be customized.

```
/*
 * Create or update index by json
 */
public static void indexByJson(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        IndexRequest indexRequest = new IndexRequest(index, type, id);
        String jsonString = "{" + "\"user\":\"kimchy1\"," + "\"age\":\"100\"," + "\"postDate\"
        ":"2020-01-01",
            + "\"message\":\"trying out Elasticsearch\"," + "\"reason\":\"daily update\",
            + "\"innerObject1\":\"Object1\"," + "\"innerObject2\":\"Object2\",
            + "\"innerObject3\":\"Object3\"," + "\"uid\":\"11\"}";
        indexRequest.source(jsonString, XContentType.JSON);
        IndexResponse indexResponse = highLevelClient.index(indexRequest, RequestOptions.DEFAULT);

        LOG.info("IndexByJson response is {}.", indexResponse.toString());
    } catch (Exception e) {
```

```
        LOG.error("IndexByJson is failed,exception occurred.", e);
    }
```

## Writing Data in the Map Format

**IndexByMap.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index** directory is used to write data into a specified index by using a data source in the MAP format.

The variable *dataMap* is data to be inserted and can be customized.

```
/*
 * Create or update index by map
 */
public static void indexByMap(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        Map<String, Object> dataMap = new HashMap<>();
        dataMap.put("user", "kimchy2");
        dataMap.put("age", "200");
        dataMap.put("postDate", new Date());
        dataMap.put("message", "trying out Elasticsearch");
        dataMap.put("reason", "daily update");
        dataMap.put("innerObject1", "Object1");
        dataMap.put("innerObject2", "Object2");
        dataMap.put("innerObject3", "Object3");
        dataMap.put("uid", "22");
        IndexRequest indexRequest = new IndexRequest(index, type, id).source(dataMap);
        IndexResponse indexResponse = highLevelClient.index(indexRequest, RequestOptions.DEFAULT));

        LOG.info("IndexByMap response is {}.", indexResponse.toString());
    } catch (Exception e) {
        LOG.error("IndexByMap is failed,exception occurred.", e);
    }
}
```

## Writing Data in the XContentBuilder format

**IndexByXContentBuilder.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index** directory is used to write data into a specified index by using a data source in the XContentBuilder format.

The variable *builder* is data to be inserted and can be customized.

```
/*
 * Create or update index by XContentBuilder
 */
public static void indexByXContentBuilder(RestHighLevelClient highLevelClient, String index, String type, String id) {

    try {
        XContentBuilder builder = XContentFactory.jsonBuilder();
        builder.startObject();
        {
            builder.field("user", "kimchy3");
            builder.field("age", "300");
            builder.field("postDate", "2020-01-01");
            builder.field("message", "trying out Elasticsearch");
            builder.field("reason", "daily update");
            builder.field("innerObject1", "Object1");
            builder.field("innerObject2", "Object2");
            builder.field("innerObject3", "Object3");
            builder.field("uid", "33");
        }
    }
}
```

```
builder.endObject();
IndexRequest indexRequest = new IndexRequest(index, type, id).source(builder);
IndexResponse indexResponse = highLevelClient.index(indexRequest, RequestOptions.DEFAULT);

LOG.info("IndexByXContentBuilder response is {}", indexResponse.toString());
} catch (Exception e) {
    LOG.error("IndexByXContentBuilder is failed,exception occurred.", e);
}
```

#### 1.7.3.3.4 Operations in Batches

### Function

**Bulk.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/bulk** directory is used to perform operations in batches, for example, create an index, update an index, or delete an index.

```
/*
 * Bulk request can be used to execute multiple index,update or delete
 * operations using a single request.
 */
private static void bulk(RestHighLevelClient highLevelClient, String index, String type) {

    try {
        Map<String, Object> jsonMap = new HashMap<>();
        for (int i = 1; i <= 100; i++) {
            BulkRequest request = new BulkRequest();
            for (int j = 1; j <= 1000; j++) {
                jsonMap.clear();
                jsonMap.put("user", "Linda");
                jsonMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
                jsonMap.put("postDate", "2020-01-01");
                jsonMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
                jsonMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
                request.add(new IndexRequest(index, type).source(jsonMap));
            }
            BulkResponse bulkResponse = highLevelClient.bulk(request, RequestOptions.DEFAULT);

            if (RestStatus.OK.equals((bulkResponse.status()))) {
                LOG.info("Bulk is successful");
            } else {
                LOG.info("Bulk is failed");
            }
        }
    } catch (Exception e) {
        LOG.error("Bulk is failed,exception occurred.", e);
    }
}
```

#### 1.7.3.3.5 Batch write data to specified routes

### Function

**BulkRoutingSample.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/bulk** directory is used to set routing for each request in the bulk, route documents with the same routing to the same segment, and specify routing during query to narrow the query range and improve the query speed.

```
/*
 * Number of documents to be written.
 */
```

```
private static final int NUM_OF_DOC = 100;

/**
 * Use bulk to write data and specify routing.
 */
private static void bulkWithRouting(RestHighLevelClient highLevelClient, String indexName) {
    try {
        Map<String, Object> jsonMap = new HashMap<>();
        BulkRequest request = new BulkRequest();
        int age;
        for (int i = 1; i <= NUM_OF_DOC; i++) {
            jsonMap.clear();
            jsonMap.put("user", "Linda" + i);
            age = ThreadLocalRandom.current().nextInt(18, 100);
            jsonMap.put("age", age);
            jsonMap.put("postDate", "2020-01-01");
            jsonMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
            jsonMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
            // Route documents of the same age to the same shard based on the age.
            request.add(new IndexRequest(indexName).source(jsonMap).routing(String.valueOf(age)));
        }
        BulkResponse bulkResponse = highLevelClient.bulk(request, RequestOptions.DEFAULT);
        if (RestStatus.OK.equals(bulkResponse.status())) {
            LOG.info("Bulk routing is successful.");
        } else {
            LOG.info("Bulk routing is failed.");
        }
    } catch (IOException e) {
        LOG.error("Bulk routing is failed, exception occurred.", e);
    }
}
```

### 1.7.3.3.6 Updating Index Information

## Function

**Update.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/update` directory is used to update the index.

```
/**
 * Update index
 */
public static void update(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        XContentBuilder builder = XContentFactory.jsonBuilder();
        builder.startObject();
        { // update information
            builder.field("postDate", new Date());
            builder.field("reason", "update again");
        }
        builder.endObject();
        UpdateRequest request = new UpdateRequest(index, type, id).doc(builder);
        UpdateResponse updateResponse = highLevelClient.update(request, RequestOptions.DEFAULT);

        LOG.info("Update response is {}.", updateResponse.toString());
    } catch (Exception e) {
        LOG.error("Update is failed,exception occurred.", e);
    }
}
```

### 1.7.3.3.7 Querying Index Information

#### Function

**GetIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/search** directory is used to query information about a document with specified **index**, **type**, and **id**.

```
/*
 * Get index information
 */
private static void getIndex(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        GetRequest getRequest = new GetRequest(index, type, id);
        String[] includes = new String[] { "message", "test*" };
        String[] excludes = Strings.EMPTY_ARRAY;
        FetchSourceContext fetchSourceContext = new FetchSourceContext(true, includes, excludes);
        getRequest.fetchSourceContext(fetchSourceContext);
        getRequest.storedFields("message");
        GetResponse getResponse = highLevelClient.get(getRequest, RequestOptions.DEFAULT);

        LOG.info("GetIndex response is {}", getResponse.toString());
    } catch (Exception e) {
        LOG.error("GetIndex is failed,exception occurred.", e);
    }
}
```

### 1.7.3.3.8 Searching for Document Information

#### Function

**Search.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/search** directory is used to query related document information in a specified index. You can customize specified sorting, filtering, highlighting, feedback data range, timeout interval, and the like.

```
/*
 * Search some information in index
 */
public static void search(RestHighLevelClient highLevelClient, String index) {
    try {
        SearchRequest searchRequest = new SearchRequest(index);
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.termQuery("user", "kimchy1"));

        //Specifying Sorting
        searchSourceBuilder.sort(new FieldSortBuilder("_doc").order(SortOrder.ASC));

        // Source filter
        String[] includeFields = new String[] { "message", "user", "innerObject*" };
        String[] excludeFields = new String[] { "postDate" };
        searchSourceBuilder.fetchSource(includeFields, excludeFields);
        // Control which fields get included
        // excluded

        // Request Highlighting
        HighlightBuilder highlightBuilder = new HighlightBuilder();
        HighlightBuilder.Field highlightUser = new HighlightBuilder.Field("user");
        // Create a field highlighter for
        // the user field

        highlightBuilder.field(highlightUser);
    }
```

```
searchSourceBuilder.highlighter(highlightBuilder);
searchSourceBuilder.from(0);
searchSourceBuilder.size(2);
searchSourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS));
searchRequest.source(searchSourceBuilder);
SearchResponse searchResponse = highLevelClient.search(searchRequest, RequestOptions.DEFAULT);

LOG.info("Search response is {}.", searchResponse.toString());
} catch (Exception e) {
    LOG.error("Search is failed,exception occurred.", e);
}
}
```

### 1.7.3.3.9 Searching for Document Information by Using a Cursor

#### Function

**SearchScroll.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/searchscroll` directory is used to query related document information in a specified index by using a cursor. The method returns a **scrollId**. You can use the **scrollId** to clear the cursor. For details about the clearing method, see [Clearing a Cursor](#).

```
/*
 * Send a search scroll request
 */
public static String searchScroll(RestHighLevelClient highLevelClient, String index) {
    String scrollId = null;
    try {
        final Scroll scroll = new Scroll(TimeValue.timeValueMinutes(1L));
        SearchRequest searchRequest = new SearchRequest(index);
        searchRequest.scroll(scroll);
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchQuery("title", "Elasticsearch"));
        searchRequest.source(searchSourceBuilder);

        SearchResponse searchResponse = highLevelClient.search(searchRequest, RequestOptions.DEFAULT);
        scrollId = searchResponse.getScrollId();
        SearchHit[] searchHits = searchResponse.getHits().getHits();
        LOG.info("SearchHits is {}", searchResponse.toString());

        while (searchHits != null && searchHits.length > 0) {
            SearchScrollRequest scrollRequest = new SearchScrollRequest(scrollId);
            scrollRequest.scroll(scroll);
            searchResponse = highLevelClient.searchScroll(scrollRequest, RequestOptions.DEFAULT);
            scrollId = searchResponse.getScrollId();
            searchHits = searchResponse.getHits().getHits();
            LOG.info("SearchHits is {}", searchResponse.toString());
        }
    } catch (Exception e) {
        LOG.error("SearchScroll is failed,exception occurred.", e);
        return null;
    }
    return scrollId;
}
```

### 1.7.3.3.10 Clearing a Cursor

#### Function

**SearchScroll.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/searchscroll` directory includes the method **clearScroll(highLevelClient,scrollId)**, which is used to clear a cursor specified by **scrollId**.

```
/*
 * Clear a search scroll
 */
public static void clearScroll(RestHighLevelClient highLevelClient, String scrollId) {
    ClearScrollRequest clearScrollRequest = new ClearScrollRequest();
    clearScrollRequest.addScrollId(scrollId);
    ClearScrollResponse clearScrollResponse;
    try {
        clearScrollResponse = highLevelClient.clearScroll(clearScrollRequest, RequestOptions.DEFAULT);
        if (clearScrollResponse.isSucceeded()) {
            LOG.info("ClearScroll is successful.");
        } else {
            LOG.error("ClearScroll is failed.");
        }
    } catch (IOException e) {
        LOG.error("ClearScroll is failed,exception occurred.", e);
    }
}
```

### 1.7.3.3.11 Deleting an Index

#### Function

**DeleteIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/delete** directory is used to delete a specified index.

```
/*
 * Delete the index
 */
public static void deleteIndex(RestHighLevelClient highLevelClient, String index) {
    try {
        DeleteIndexRequest request = new DeleteIndexRequest(index);
        DeleteIndexResponse deleteResponse = highLevelClient.indices().deleteIndex(request,
RequestOptions.DEFAULT);

        if (deleteResponse.isAcknowledged()) {
            LOG.info("Delete index is successful");
        } else {
            LOG.info("Delete index is failed");
        }
    } catch (Exception e) {
        LOG.error("Delete index : {} is failed, exception occurred.", index, e);
    }
}
```

### 1.7.3.3.12 Multi-Thread Request

#### Function

**MultithreadRequest.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/multithread** directory is used to create a thread class **sendRequestThread**, rewrite a **run** method, implement a **bulk** request, and write data in batches.

```
/*
 * The thread that send a bulk request
 */
public static class sendRequestThread implements Runnable {

    private RestHighLevelClient highLevelClientTh;
    @Override
    public void run() {
```

```
LOG.info("Thread begin.");
HwRestClient hwRestClient = new HwRestClient();
try {
    highLevelClientTh = new RestHighLevelClient(hwRestClient.getRestClientBuilder());
    LOG.info("Thread name: " + Thread.currentThread().getName());
    bulk(highLevelClientTh,"huawei1","type1");
} catch (Exception e) {
    LOG.error("SendRequestThread had exception.", e);
} finally {
    if (highLevelClientTh != null) {
        try {
            highLevelClientTh.close();
            LOG.info("Close the highLevelClient successful in thread : " +
Thread.currentThread().getName());
        } catch (IOException e) {
            LOG.error("Close the highLevelClient failed.", e);
        }
    }
}
```

In the **main** method, the **fixedThreadPool** thread pool is created. By specifying **threadPoolSize** and **jobNumber**, multiple threads are supported to send bulk requests and data can be concurrently written in batches.

```
public static void main(String[] args) throws Exception {  
    LOG.info("Start to do bulk request !");  
  
    ExecutorService fixedThreadPool;  
    int threadPoolSize = 2;  
    int jobNumber = 5;  
    if (HighLevelUtils.getConfig()) {  
        try {  
            fixedThreadPool = Executors.newFixedThreadPool(threadPoolSize);  
            for (int i = 0; i < jobNumber; i++) {  
                fixedThreadPool.execute(new sendRequestThread());  
            }  
            fixedThreadPool.shutdown();  
        } catch (Exception e) {  
            LOG.error("SendRequestThread is failed,exception occurred.", e);  
        }  
    } else {  
        LOG.error("Failed to get configuration!");  
    }  
}
```

### **1.7.3.3.13 BulkProcessor Batch Import Example**

## Description

The `BulkProcessorSample.java` file in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/bulk` directory is used to instruct users to use BulkProcessor to import data to the database in batches.

## BulkProcessor initialization:

```
private static BulkProcessor getBulkProcessor(RestHighLevelClient highLevelClient) {  
    BulkProcessor.Listener listener = new BulkProcessor.Listener() {  
        @Override  
        public void beforeBulk(long executionId, BulkRequest bulkRequest) {  
            int numberOfActions = bulkRequest.numberOfActions();  
            LOG.info("Executing bulk {} with {} requests.", executionId, numberOfActions);  
        }  
    };  
    return BulkProcessor.builder(bulkRequest::add).setListener(listener).build();  
}
```

```

    }

    @Override
    public void afterBulk(long executionId, BulkRequest bulkRequest, BulkResponse bulkResponse) {
        if (bulkResponse.hasFailures()) {
            LOG.warn("Bulk {} executed with failures.", executionId);
        } else {
            LOG.info("Bulk {} completed in {} milliseconds.", executionId,
                    bulkResponse.getTook().getMillis());
        }
    }

    @Override
    public void afterBulk(long executionId, BulkRequest bulkRequest, Throwable throwable) {
        LOG.error("Failed to execute bulk.", throwable);
    }
};

BiConsumer<BulkRequest, ActionListener<BulkResponse>> bulkConsumer =
    (request, bulkListener) -> highLevelClient.bulkAsync(request, RequestOptions.DEFAULT,
bulkListener);

BulkProcessor bulkProcessor = BulkProcessor.builder(bulkConsumer, listener)
    .setBulkActions(onceBulkMaxNum)
    .setBulkSize(new ByteSizeValue(onceBulkMaxSize, ByteSizeUnit.MB))
    .setConcurrentRequests(concurrentRequestsNum)
    .setFlushInterval(TimeValue.timeValueSeconds(flushTime))
    .setBackoffPolicy(BackoffPolicy.constantBackoff(TimeValue.timeValueSeconds(1L), maxRetry))
    .build();

LOG.info("Init bulkProcess successfully.");
return bulkProcessor;
}

```

**BulkProcessor import example:**

```

private void singleThreadBulk() {
    //single thread
    int bulkTime = 0;
    while (bulkTime++ < totalNumberForThread) {
        Map<String, Object> dataMap = new HashMap<>();
        dataMap.put("date", "2019/12/9");
        dataMap.put("text", "the test text");
        dataMap.put("title", "the title");
        bulkProcessor.add(new IndexRequest(indexName, indexType).source(dataMap));
    }
    LOG.info("This thread bulks successfully, the thread name is {}.", Thread.currentThread().getName());
}

```

#### 1.7.3.3.14 Pre-sorting Indices

### Function Description

The **IndexSorting.java** file in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index** directory is used to configure the fields to be sorted when an index is created, implement the **bulk** request, write data in batches, and query the top N documents.

During index creation, specify the fields to be sorted and specify whether each field is sorted in descending or ascending order.

```

private static void createIndexWithSorting(RestHighLevelClient highLevelClient, String index) {
    try {
        CreateIndexRequest indexRequest = new CreateIndexRequest(index);
        indexRequest.settings(

```

```

Settings.builder().put("index.number_of_shards", 3)
    .put("index.number_of_replicas", 1)
    .putList("index.sort.field", "height", "weight")
    .putList("index.sort.order", SortOrder.DESC.toString(), SortOrder.ASC.toString());
indexRequest.mapping("{\"properties\": {\"height\": {\"type\": \"float\"},\n" +
    + "\"weight\": {\"type\": \"float\"}}}", XContentType.JSON);
CreateIndexResponse response = highLevelClient.indices().create(indexRequest,
RequestOptions.DEFAULT);
if (response.isAcknowledged() || response.isShardsAcknowledged()) {
    LOG.info("Create index with sorting successfully.");
}
} catch (IOException e) {
    LOG.error("Failed to create index with sorting, exception occurred.", e);
}
}

```

Then, write data in batches.

```

public static void bulk(RestHighLevelClient highLevelClient, String index) {
    try {
        Map<String, Object> jsonMap = new HashMap<>();
        for (int i = 1; i <= 10; i++) {
            BulkRequest request = new BulkRequest();
            for (int j = 1; j <= 1000; j++) {
                jsonMap.clear();
                jsonMap.put("user", "Linda");
                jsonMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
                jsonMap.put("postDate", "2020-01-01");
                jsonMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
                jsonMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
                request.add(new IndexRequest(index).source(jsonMap));
            }
            BulkResponse bulkResponse = highLevelClient.bulk(request, RequestOptions.DEFAULT);

            if (RestStatus.OK.equals((bulkResponse.status()))) {
                LOG.info("Bulk is successful");
            } else {
                LOG.info("Bulk is failed");
            }
        }
    } catch (IOException e) {
        LOG.error("Bulk is failed,exception occurred.", e);
    }
}

```

Perform the refresh operation so that the data can be queried after being written in batches.

```

private static void refresh(RestHighLevelClient highLevelClient, String index) {
    try {
        RefreshRequest refRequest = new RefreshRequest(index);
        highLevelClient.indices().refresh(refRequest, RequestOptions.DEFAULT);
        LOG.info("Refresh sorting index successfully.");
    } catch (IOException e) {
        LOG.error("Failed to refresh sorting index, exception occurred.", e);
    }
}

```

Then, you can query the sorting field. By default, the top 10 documents are obtained.

```

private static void queryData(RestHighLevelClient highLevelClient, String index) {
    try {
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.sort(new FieldSortBuilder("height").order(SortOrder.DESC));
        searchSourceBuilder.sort(new FieldSortBuilder("weight").order(SortOrder.ASC));
        // Source filter
        String[] includeFields = new String[]{"weight", "height", "age"};
        String[] excludeFields = new String[]{};
    }
}

```

```
searchSourceBuilder.fetchSource(includeFields, excludeFields);
// Control which fields get included or excluded
// Request Highlighting
HighlightBuilder highlightBuilder = new HighlightBuilder();
HighlightBuilder.Field highlightUser = new HighlightBuilder.Field("height");
highlightBuilder.field(highlightUser);
searchSourceBuilder.highlighter(highlightBuilder);
searchSourceBuilder.timeout(new TimeValue(1, TimeUnit.SECONDS));
SearchRequest searchRequest = new SearchRequest(index);
searchRequest.source(searchSourceBuilder);
SearchResponse searchResponse = highLevelClient.search(searchRequest, RequestOptions.DEFAULT);
LOG.info("Search response is {}", searchResponse.toString());
} catch (IOException e) {
    LOG.error("Failed to query sorting index, exception occurred.", e);
}
}
```

### 1.7.3.4 OpenDistro JDBC Sample Code

#### 1.7.3.4.1 Connecting the Client to a Cluster

##### Function Description

This section describes how to obtain the client and set a cluster name, an IP address, and a port number, to connect to a specific Elasticsearch cluster. This is a prerequisite before using the APIs provided by Elasticsearch.

###### NOTE

- After the Elasticsearch operations are complete, you need to call **Connection.close()** to close the applied resources.
- The OpenDistro JDBC sample does not support IBM JDK.

```
public Connection sqlConnect() {
    Properties properties = initConfig();
    if (properties == null) {
        LOG.error("Get properties error");
        return null;
    }
    try {
        String url = getUrl();
        if (url == null) {
            LOG.error("Get server url error");
            return null;
        }
        return DriverManager.getConnection(url, properties);
    } catch (SQLException e) {
        LOG.error("Connect error.");
        return null;
    }
}
```

###### NOTE

By default, the client reads the following configuration files from the **conf** directory in the code running path: **esParams.properties**, **krb5.conf**, and **user.keytab**.

### 1.7.3.4.2 Querying an Index

#### Function Description

The **SqlSearch.java** file in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/jdbc** directory is used to query index data.

```
private static void basicSearch(Connection connection) {
    if (connection == null) {
        LOG.error("Connection is null.");
        return;
    }
    String sql = "SELECT * FROM example-sql1 limit 10";
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.executeQuery();
        LOG.info("Search succeed.");
    } catch (SQLException e) {
        LOG.info("Search failed.");
    }
}
```

Syntax:

```
SELECT [DISTINCT] (* | EXPRESSION) [[AS] ALIAS] [, ...]
FROM INDEX_NAME
[WHERE PREDICATES]
[GROUP BY EXPRESSION [, ...]]
[HAVING PREDICATES]
[ORDER BY EXPRESSION [IS [NOT] NULL] [ASC | DESC] [, ...]]
[LIMIT [offset, ] size]
```

#### Example SQLs:

- Search for specific fields only:  
**SELECT user, age FROM example-sql1 limit 10**
- Use aliases for a field:  
**SELECT age AS user\_age FROM example-sql1 limit 10**
- Deduplicate the results:  
**SELECT DISTINCT age FROM example-sql1 limit 10**
- Example of the **Where** clause:

```
private static void whereAndLimitSearch(Connection connection) {
    if (connection == null) {
        LOG.error("Connection is null.");
        return;
    }
    String sql = "SELECT user, age FROM example-sql1 WHERE height>? limit 10";
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setInt(1, 100);
        statement.executeQuery();
        LOG.info("Search succeed.");
    } catch (SQLException e) {
        LOG.error("Search failed.");
    }
}
```



#### NOTE

You are advised to use **PreparedStatement** to perform parameterized query, which can effectively prevent SQL injection.

- GroupBy:

- ```
SELECT weight AS user_weight FROM example-sql1 GROUP BY user_weight limit 10
```
- Function use:  

```
SELECT ADD(weight,10) AS user_weight FROM example-sql1 GROUP BY ADD(weight,10) limit 10
```
  - Having clause:  

```
SELECT age, MAX(weight) FROM example-sql1 GROUP BY age HAVING MIN(weight)>20 limit 10
```
  - OrderBy:  

```
SELECT height FROM example-sql1 ORDER BY height DESC limit 10
```
  - Table association:  

```
SELECT l.height,l.weight,i.height,i.weight FROM example-sql1 l JOIN example-sql2 i ON i.age = l.age limit 10
```
  - Subquery:  

```
SELECT a.myHeight, a.myWeight FROM (SELECT height AS myHeight, weight AS myWeight FROM example-sql1 WHERE age > 25 LIMIT 5) AS a
```
  - Use a single field to search for text:  

```
SELECT user, weight, address FROM example-sql1 WHERE MATCH_QUERY(address, 'Holmes') limit 10
```
  - Split texts based on operators:  

```
SELECT user, weight, address FROM example-sql1 WHERE QUERY('address:Lane OR address:Street') limit 10
```

#### 1.7.3.4.3 Deleting Index Information

##### Function Description

The `SqlDelete.java` file in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/jdbc` directory is used to delete index data.

##### NOTICE

If the WHERE clause is not specified, all documents are deleted.

```
public static void deleteData(Connection connection) {  
    if (connection == null) {  
        LOG.error("Statement is null.");  
        return;  
    }  
    String sql = "DELETE FROM example-sql2 WHERE height=? ";  
    try (PreparedStatement statement = connection.prepareStatement(sql)) {  
        statement.setInt(1, 25);  
        statement.executeQuery();  
        LOG.info("Delete data successful.");  
    } catch (SQLException e) {  
        LOG.error("Delete data failed.");  
    }  
}
```

#### 1.7.3.4.4 Functions



The following table lists the functions supported by OpenDistro JDBC sql and their usage and examples, There are six types of functions: mathematical operation functions, trigonometric functions, time and date functions, string functions, summation functions, and advanced functions.

#### Mathematical Operation Functions

**Table 1-16** List of mathematical operation functions

Function	Description	Example
abs	abs(number T) -> T	SELECT abs(0.5) FROM my-index LIMIT 1
add	add(number T, number) -> T	SELECT add(1, 5) FROM my-index LIMIT 1
cbrt	cbrt(number T) -> T	SELECT cbrt(0.5) FROM my-index LIMIT 1
ceil	ceil(number T) -> T	SELECT ceil(0.5) FROM my-index LIMIT 1
conv	conv(string T, int a, int b) -> T	SELECT CONV('12', 10, 16), CONV('2C', 16, 10), CONV(12, 10, 2), CONV(1111, 2, 10) FROM my-index LIMIT 1
crc32	crc32(string T) -> T	SELECT crc32('MySQL') FROM my-index LIMIT 1
divide	divide(number T, number) -> T	SELECT divide(1, 0.5) FROM my-index LIMIT 1
e	e() -> double	SELECT e() FROM my-index LIMIT 1
exp	exp(number T) -> T	SELECT exp(0.5) FROM my-index LIMIT 1
expm1	expm1(number T) -> T	SELECT expm1(0.5) FROM my-index LIMIT 1
floor	floor(number T) -> T	SELECT floor(0.5) AS Rounded_Down FROM my-index LIMIT 1
ln	ln(number T) -> double	SELECT ln(10) FROM my-index LIMIT 1
log	log(number T) -> double or log(number T, number) -> double	SELECT log(10) FROM my-index LIMIT 1
log2	log2(number T) -> double	SELECT log2(10) FROM my-index LIMIT 1

Function	Description	Example
log10	log10(number T) -> double	SELECT log10(10) FROM my-index LIMIT 1
mod	mod(number T, number) -> T	SELECT modulus(2, 3) FROM my-index LIMIT 1
multiply	multiply(number T, number) -> number	SELECT multiply(2, 3) FROM my-index LIMIT 1
pi	pi() -> double	SELECT pi() FROM my-index LIMIT 1
pow	ow(number T) -> T or pow(number T, number) -> T	SELECT pow(2, 3) FROM my-index LIMIT 1
power	power(number T) -> T or power(number T, number) -> T	SELECT power(2, 3) FROM my-index LIMIT 1
rand	rand() -> number or rand(number T) -> T	SELECT rand(0.5) FROM my-index LIMIT 1
rint	rint(number T) -> T	SELECT rint(1.5) FROM my-index LIMIT 1
round	round(number T) -> T	SELECT round(1.5) FROM my-index LIMIT 1
sign	sign(number T) -> T	SELECT sign(1.5) FROM my-index LIMIT 1
signum	signum(number T) -> T	SELECT signum(0.5) FROM my-index LIMIT 1
sqrt	sqrt(number T) -> T	SELECT sqrt(0.5) FROM my-index LIMIT 1
strcmp	strcmp(string T, string T) -> T	SELECT strcmp('hello', 'hello') FROM my-index LIMIT 1
subtract	subtract(number T, number) -> T	SELECT subtract(3, 2) FROM my-index LIMIT 1
truncate	truncate(number T, number T) -> T	SELECT truncate(56.78, 1) FROM my- index LIMIT 1
/	number [op] number -> number	SELECT 1 / 100 FROM my-index LIMIT 1
%	number [op] number -> number	SELECT 1 % 100 FROM my-index LIMIT 1

## Trigonometric Functions

**Table 1-17** List of trigonometric functions

Function	Description	Example
acos	acos(number T) -> double	SELECT acos(0.5) FROM my-index LIMIT 1
asin	asin(number T) -> double	SELECT asin(0.5) FROM my-index LIMIT 1
atan	atan(number T) -> double	SELECT atan(0.5) FROM my-index LIMIT 1
atan2	atan2(number T, number) -> double	SELECT atan2(1, 0.5) FROM my-index LIMIT 1
cos	cos(number T) -> double	SELECT cos(0.5) FROM my-index LIMIT 1
cosh	cosh(number T) -> double	SELECT cosh(0.5) FROM my-index LIMIT 1
cot	cot(number T) -> double	SELECT cot(0.5) FROM my-index LIMIT 1
degrees	degrees(number T) -> double	SELECT degrees(0.5) FROM my-index LIMIT 1
radians	radians(number T) -> double	SELECT radians(0.5) FROM my-index LIMIT 1
sin	sin(number T) -> double	SELECT sin(0.5) FROM my-index LIMIT 1
sinh	sinh(number T) -> double	SELECT sinh(0.5) FROM my-index LIMIT 1
tan	tan(number T) -> double	SELECT tan(0.5) FROM my-index LIMIT 1

## Date and Time Functions

**Table 1-18** List of time and date functions

Function	Description	Example
adddate	adddate(date, INTERVAL expr unit) -> date	SELECT adddate(date('2020-08-26'), INTERVAL 1 hour) FROM my-index LIMIT 1
curdate	curdate() -> date	SELECT curdate() FROM my-index LIMIT 1

Function	Description	Example
date	date(date) -> date	SELECT date() FROM my-index LIMIT 1
date_format	date_format(date, string) -> string or date_format(date, string, string) -> string	SELECT date_format(date, 'Y') FROM my-index LIMIT 1
date_sub	date_sub(date, INTERVAL expr unit) -> date	SELECT date_sub(date('2008-01-02'), INTERVAL 31 day) FROM my-index LIMIT 1
dayofmonth	dayofmonth(date) -> integer	SELECT dayofmonth(date) FROM my-index LIMIT 1
dayname	dayname(date) -> string	SELECT dayname(date('2020-08-26')) FROM my-index LIMIT 1
dayofyear	dayofyear(date) -> integer	SELECT dayofyear(date('2020-08-26')) FROM my-index LIMIT 1
dayofweek	dayofweek(date) -> integer	SELECT dayofweek(date('2020-08-26')) FROM my-index LIMIT 1
from_days	from_days(N) -> integer	SELECT from_days(733687) FROM my-index LIMIT 1
hour	hour(time) -> integer	SELECT hour((time '01:02:03')) FROM my-index LIMIT 1
maketime	maketime(integer, integer, integer) -> date	SELECT maketime(1, 2, 3) FROM my-index LIMIT 1
microsecond	microsecond(expr) -> integer	SELECT microsecond((time '01:02:03.123456')) FROM my-index LIMIT 1
minute	minute(expr) -> integer	SELECT minute((time '01:02:03')) FROM my-index LIMIT 1
month	month(date) -> integer	SELECT month(date) FROM my-index
monthname	monthname(date) -> string	SELECT monthname(date) FROM my-index
now	now() -> date	SELECT now() FROM my-index LIMIT 1
quarter	quarter(date) -> integer	SELECT quarter(date('2020-08-26')) FROM my-index LIMIT 1
second	second(time) -> integer	SELECT second((time '01:02:03')) FROM my-index LIMIT 1

Function	Description	Example
subdate	subdate(date, INTERVAL expr unit) -> date, datetime	SELECT subdate(date('2008-01-02'), INTERVAL 31 day) FROM my-index LIMIT 1
time	time(expr) -> time	SELECT time('13:49:00') FROM my-index LIMIT 1
time_to_sec	time_to_sec(time) -> long	SELECT time_to_sec(time '22:23:00') FROM my-index LIMIT 1
timestamp	timestamp(date) -> date	SELECT timestamp(date) FROM my-index LIMIT 1
to_days	to_days(date) -> long	SELECT to_days(date '2008-10-07') FROM my-index LIMIT 1
week	week(date[mode]) -> integer	SELECT week(date('2008-02-20')) FROM my-index LIMIT 1
year	year(date) -> integer	SELECT year(date) FROM my-index LIMIT 1

## String Functions

**Table 1-19** List of string functions

Function	Description	Example
ascii	ascii(string T) -> integer	SELECT ascii(name.keyword) FROM my-index LIMIT 1
concat	concat(str1, str2) -> string	SELECT concat('hello', 'world') FROM my-index LIMIT 1
concat_ws	concat_ws(separator, string, string...) -> string	SELECT concat_ws("-", "Tutorial", "is", "fun!") FROM my-index LIMIT 1
left	left(string T, integer) -> T	SELECT left('hello', 2) FROM my-index LIMIT 1
length	length(string) -> integer	SELECT length('hello') FROM my-index LIMIT 1
locate	locate(string, string, integer) -> integer or locate(string, string) -> INTEGER	SELECT locate('o', 'hello') FROM my-index LIMIT 1, SELECT locate('l', 'hello', 3) FROM my-index LIMIT 1
replace	replace(string T, string, string) -> T	SELECT replace('hello', 'l', 'x') FROM my-index LIMIT 1

Function	Description	Example
right	right(string T, integer) -> T	SELECT right('hello', 1) FROM my-index LIMIT 1
rtrim	rtrim(string T) -> T	SELECT rtrim(name.keyword) FROM my-index LIMIT 1
substring	substring(string T, integer, integer) -> T	SELECT substring(name.keyword, 2,5) FROM my-index LIMIT 1
trim	trim(string T) -> T	SELECT trim(' hello') FROM my-index LIMIT 1
upper	upper(string T) -> T	SELECT upper('helloworld') FROM my-index LIMIT 1

## Summation Functions

**Table 1-20** List of summation functions

Function	Description	Example
avg	avg(number T) -> T	SELECT avg(2, 3) FROM my-index LIMIT 1
count	count(number T) -> T	SELECT count(date) FROM my-index LIMIT 1
min	min(number T, number) -> T	SELECT min(2, 3) FROM my-index LIMIT 1
show	show(string T) -> T	SHOW TABLES LIKE my-index

## Advanced Functions

**Table 1-21** List of advanced functions

Function	Specification	Example
if	if(boolean, es_type, es_type) -> es_type	SELECT if(false, 0, 1) FROM my-index LIMIT 1, SELECT if(true, 0, 1) FROM my-index LIMIT 1
ifnull	ifnull(es_type, es_type) -> es_type	SELECT ifnull('hello', 1) FROM my-index LIMIT 1, SELECT ifnull(null, 1) FROM my-index LIMIT 1
isnull	isnull(es_type) -> integer	SELECT isnull(null) FROM my-index LIMIT 1, SELECT isnull(1) FROM my-index LIMIT 1

## 1.7.4 Commissioning Process

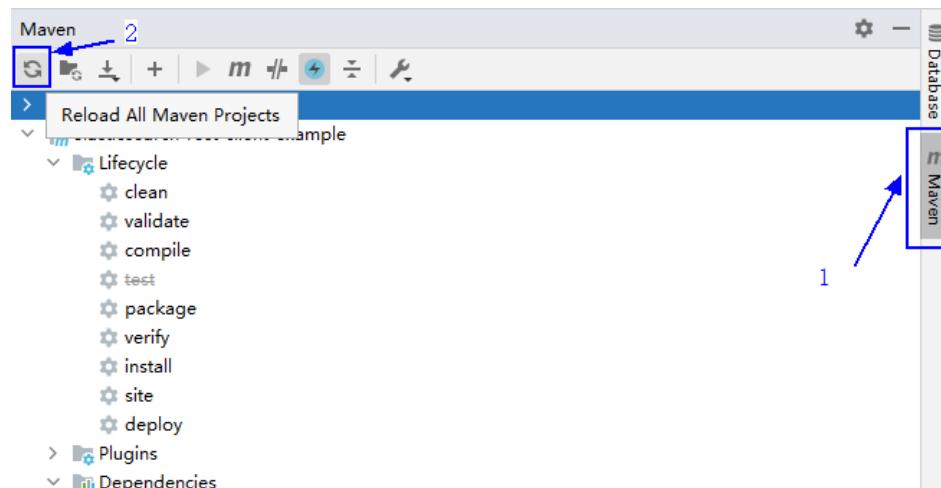
### 1.7.4.1 Commissioning Applications on Windows

#### 1.7.4.1.1 Commissioning a RestClient Application

#### Compiling and Running Applications

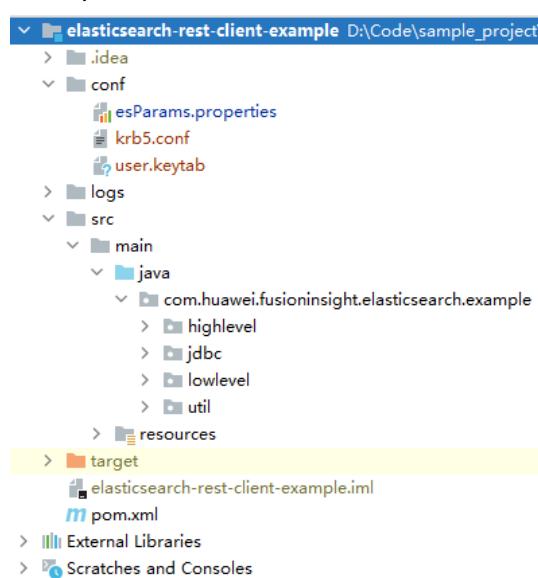
You can run an application on Windows after code development is complete. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

1. Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.



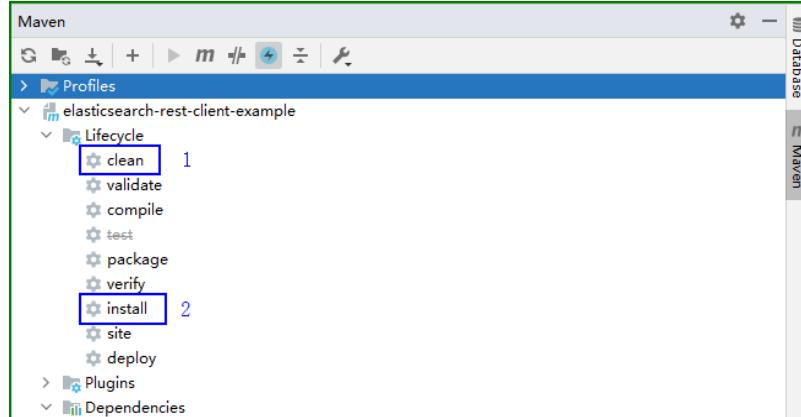
2. Compile and run an application.

- a. The following figure shows the file list after the configuration file has been placed and the code has been modified to match the login user.



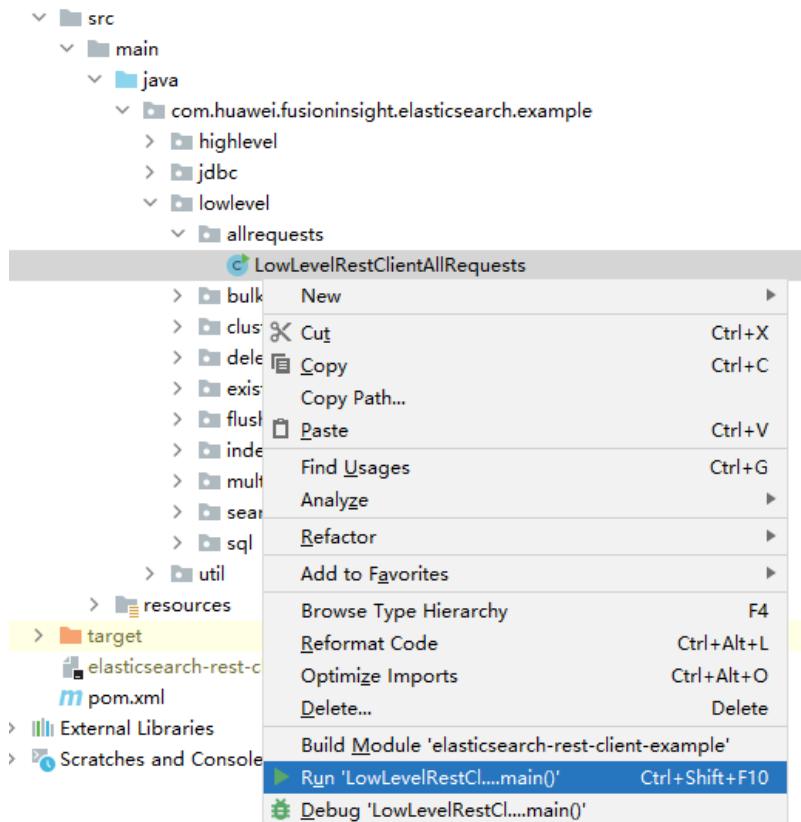
b. Compile an application.

Double-click **clean** and **install** to run the **maven clean** and **maven install** commands. After the compilation is complete, the message "Build Success" is displayed.

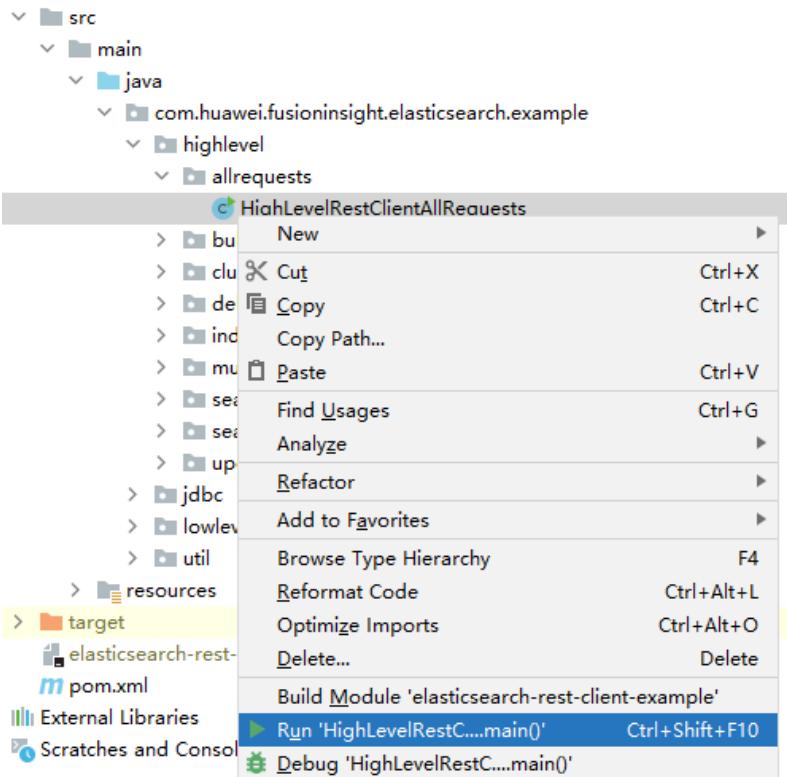


c. Run the program.

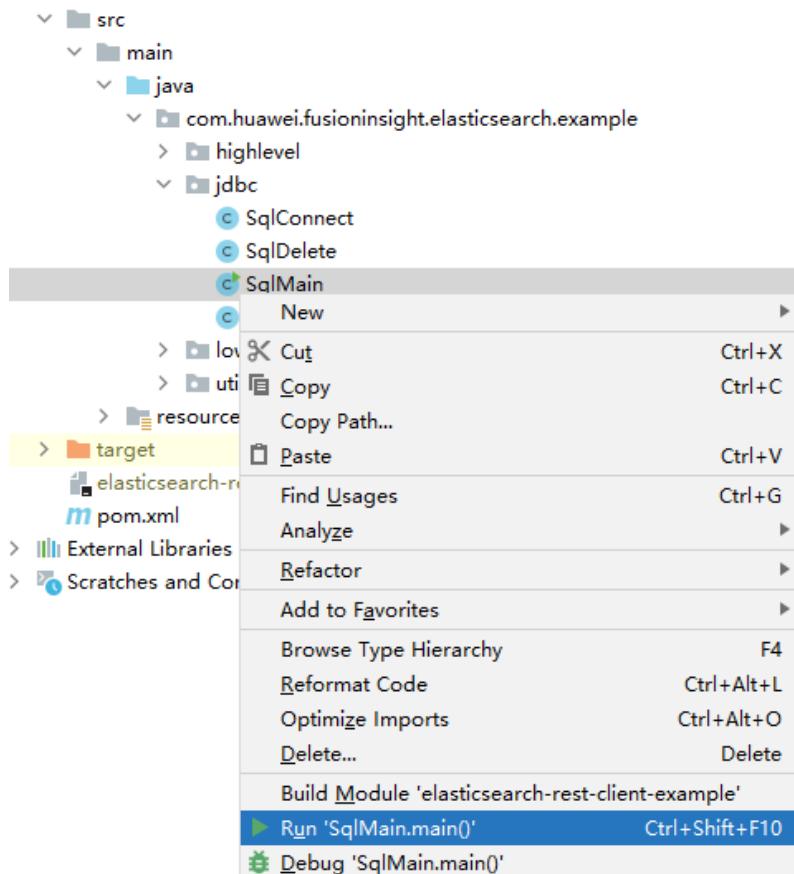
Right-click the **LowLevelRestClientAllRequests** file and choose **Run 'Low....main()'** from the shortcut menu to run all samples of the Low Level Rest Client.



Right-click the **HighLevelRestClientAllRequests** file and choose **Run 'High....main()'** from the shortcut menu to run all samples of the High Level Rest Client.



Right-click the **SqlMain** file and choose **Run 'SqlMain.main()'** from the shortcut menu to run all samples of the JDBC Rest Client.



## Viewing Commissioning Results

After you run an Elasticsearch application, you can use either of the following methods to view the running status:

- Check the running status of the application according to the running result.
- View Elasticsearch logs to learn application running status, that is, the **elasticsearch-rest-client-example/logs/es-example.log** log file in the **logs** directory.

After a complete sample of the Low Level Rest Client is run, some operation results are displayed on the console:

```
2019-01-09 19:08:50,675 | INFO | main | Start to do low level rest client request ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.
main(LowLevelRestClientAllRequests.java:449)
2019-01-09 19:08:50,690 | INFO | main | esServerHost:10.1.1.1:24100 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:73)
2019-01-09 19:08:50,690 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:74)
2019-01-09 19:08:50,690 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:75)
2019-01-09 19:08:50,690 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:76)
2019-01-09 19:08:50,690 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:77)
2019-01-09 19:08:50,690 | INFO | main | shardNum:5 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:78)
2019-01-09 19:08:50,690 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:79)
2019-01-09 19:08:50,690 | INFO | main | isSecureMode:true |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:80)
2019-01-09 19:08:50,690 | INFO | main | oneCommit:5 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:81)
2019-01-09 19:08:50,690 | INFO | main | totalRecordNumber:10 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:82)
2019-01-09 19:08:50,690 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.i
nitProperties(LowLevelRestClientAllRequests.java:83)
2019-01-09 19:08:50,690 | INFO | main | FilePath is D:\idea-workspace5\elasticsearch-rest-client-example
\conf\ |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.
setSecConfig(LowLevelRestClientAllRequests.java:116)
2019-01-09 19:08:50,690 | INFO | main | jaasPath is D:\idea-workspace5\elasticsearch-rest-client-example\
\conf\es.h00423283.jaas.conf |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:170)
2019-01-09 19:08:50,690 | INFO | main | keytabPath is D:\idea-workspace5\elasticsearch-rest-client-
example\conf\user.keytab |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:171)
2019-01-09 19:08:50,706 | INFO | main | es.security.indication is true |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.
setSecConfig(LowLevelRestClientAllRequests.java:123)
2019-01-09 19:08:53,800 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:08:53,800 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:08:53,800 | INFO | main | isSecureMode is true |
```

```
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-09 19:08:53,800 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:08:53,800 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:08:53,800 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is null
isInitiator true KeyTab is D:\idea-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass is
false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:08:54,159 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:08:54,440 | INFO | main | Success to get the service realm elasticsearch/hadoop.<system
domain name>@<system domain name> |
org.elasticsearch.client.RestClientBuilder.getServerRealm(RestClientBuilder.java:576)
2019-01-09 19:08:54,440 | INFO | main | Initialize the client successfully. |
org.elasticsearch.client.RestClientBuilder.authenticate(RestClientBuilder.java:495)
2019-01-09 19:08:54,440 | INFO | main | The Low Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.
getRestClient(LowLevelRestClientAllRequests.java:171)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is null
isInitiator true KeyTab is D:\idea-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass is
false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:08:54,534 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:08:54,815 | INFO | main | QueryClusterInfo successful. |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.
queryClusterInfo(LowLevelRestClientAllRequests.java:188)
2019-01-09 19:08:54,815 | INFO | main | QueryClusterInfo response entity is : {
    "cluster_name" : "elasticsearch_cluster",
    "status" : "yellow",
    "timed_out" : false,
    "number_of_nodes" : 3,
    "number_of_data_nodes" : 1,
    "active_primary_shards" : 15,
    "active_shards" : 15,
    "relocating_shards" : 0,
    "initializing_shards" : 0,
    "unassigned_shards" : 15,
    "delayed_unassigned_shards" : 0,
    "number_of_pending_tasks" : 0,
    "number_of_in_flight_fetch" : 0,
    "task_max_waiting_in_queue_millis" : 0,
    "active_shards_percent_as_number" : 50.0
}
|
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllRequests.
queryClusterInfo(LowLevelRestClientAllRequests.java:192)
```

After a complete sample of the High Level RestClient is run, some operation results are displayed on the console:

```
2019-01-09 19:12:23,206 | INFO | main | Start to do high level rest client request ! |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.main(HighLevelRestClientAllRequests.java:500)
2019-01-09 19:12:23,222 | INFO | main | esServerHost:192.168.61.68:24100 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:94)
2019-01-09 19:12:23,222 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
```

```
s.initProperties(HighLevelRestClientAllRequests.java:95)
2019-01-09 19:12:23,222 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:96)
2019-01-09 19:12:23,222 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:97)
2019-01-09 19:12:23,222 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:98)
2019-01-09 19:12:23,222 | INFO | main | shardNum:5 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:99)
2019-01-09 19:12:23,222 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:100)
2019-01-09 19:12:23,222 | INFO | main | isSecureMode:true |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:101)
2019-01-09 19:12:23,222 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.initProperties(HighLevelRestClientAllRequests.java:102)
2019-01-09 19:12:23,253 | INFO | main | Path is D:\idea-workspace5\elasticsearch-rest-client-example\conf \
|
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.setSecConfig(HighLevelRestClientAllRequests.java:134)
2019-01-09 19:12:23,253 | INFO | main | jaasPath is D:\idea-workspace5\elasticsearch-rest-client-example\
\conf\es.h00423283.jaas.conf |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:170)
2019-01-09 19:12:23,253 | INFO | main | keytabPath is D:\\idea-workspace5\\elasticsearch-rest-client-
example\\conf\\user.keytab |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:171)
2019-01-09 19:12:23,253 | INFO | main | es.security.indication is true |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.setSecConfig(HighLevelRestClientAllRequests.java:139)
2019-01-09 19:12:26,379 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:12:26,379 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:12:26,379 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-09 19:12:26,379 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:12:26,379 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:12:26,379 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is null
isInitiator true KeyTab is D:\\idea-workspace5\\elasticsearch-rest-client-example\\conf\\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass is
false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:12:26,754 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:12:27,035 | INFO | main | Success to get the service realm elasticsearch/hadoop.<system
domain name>@<system domain name> |
org.elasticsearch.client.RestClientBuilder.getServerRealm(RestClientBuilder.java:576)
2019-01-09 19:12:27,035 | INFO | main | Initialize the client successfully. |
org.elasticsearch.client.RestClientBuilder.authenticate(RestClientBuilder.java:495)
2019-01-09 19:12:27,644 | INFO | main | The High Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.getHighLevelRestClient(HighLevelRestClientAllRequests.java:185)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is null
isInitiator true KeyTab is D:\\idea-workspace5\\elasticsearch-rest-client-example\\conf\\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass is
false storePass is false clearPass is false
```

```
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:12:27,738 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:12:28,207 | INFO | main | ClusterName:[elasticsearch_cluster], clusterUuid:
[TTr6K7laQKuY3yZU28QiZg], nodeName:[EsNode1@192.168.61.68], version:[6.1.3]. |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.queryClusterInfo(HighLevelRestClientAllRequests.java:198)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is null
isInitiator true KeyTab is D:\idea-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass is
false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:12:28,316 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:12:28,722 | INFO | main | IndexByJson response is
IndexResponse[index=huawei1,type=type1,id=1,version=1,result=created,seqNo=0,primaryTerm=1,shards={"t
otal":2,"successful":1,"failed":0}]. |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRequest
s.indexByJson(HighLevelRestClientAllRequests.java:222)
```

After a complete sample of the JDBC Rest Client is run, some operation results are displayed on the console:

```
2022-03-31 10:37:05,006 | INFO | main | Success to get the service realm HADOOP.COM. |
org.elasticsearch.hwclient.HwRestClient.getServerRealm(HwRestClient.java:370)
2022-03-31 10:37:05,009 | INFO | main |
esServerHost:8.5.164.12:24100,8.5.165.1:24100,8.5.164.10:24100,8.5.164.12:24102,8.5.165.1:24102,8.5.164.10:
24102 | org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:173)
2022-03-31 10:37:05,010 | INFO | main | connectTimeout:5000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:174)
2022-03-31 10:37:05,010 | INFO | main | socketTimeout:60000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:175)
2022-03-31 10:37:05,010 | INFO | main | connectionRequestTimeout:100000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:176)
2022-03-31 10:37:05,010 | INFO | main | maxConnPerRouteTotal:100 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:177)
2022-03-31 10:37:05,010 | INFO | main | maxConnTotal:1000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:178)
2022-03-31 10:37:05,010 | INFO | main | isSecureMode:true |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:179)
2022-03-31 10:37:05,011 | INFO | main | sslEnabled:true |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:180)
2022-03-31 10:37:05,011 | INFO | main | principal:admintest@HADOOP.COM |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:181)
2022-03-31 10:37:05,011 | INFO | main | Config path is D:\Code\sample_project\src\elasticsearch-rest-client-
example\conf\ | org.elasticsearch.hwclient.HwRestClient.setSecConfig(HwRestClient.java:231)
2022-03-31 10:37:05,044 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:221)
2022-03-31 10:37:05,045 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:222)
2022-03-31 10:37:05,047 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:221)
2022-03-31 10:37:05,048 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:222)
2022-03-31 10:37:05,048 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.<init>(RestClientBuilder.java:245)
2022-03-31 10:37:05,050 | INFO | main | Get application configuration entry use by application name
EsClient. | org.elasticsearch.client.RestClientBuilder.getAppConfigurationEntry(RestClientBuilder.java:569)
2022-03-31 10:37:05,050 | INFO | main | Try to read the jaas configuration entry again, app name is
EsClient. |
org.elasticsearch.client.RestClientBuilder.readAppConfigurationEntryByAppName(RestClientBuilder.java:588)
2022-03-31 10:37:05,050 | INFO | main | Read application configuration entry from Es jaas conf file. |
org.elasticsearch.client.RestClientBuilder.readAppConfigurationEntryByAppName(RestClientBuilder.java:592)
2022-03-31 10:37:05,053 | INFO | main | Get application configuration entry use by application name
```

```
EsClient. |
org.elasticsearch.client.RestClientBuilder.readAppConfigurationEntryFromFile(RestClientBuilder.java:634)
2022-03-31 10:37:05,053 | INFO | main | Complete to read from Es jaas conf file, app name is EsClient. |
org.elasticsearch.client.RestClientBuilder.readAppConfigurationEntryByAppName(RestClientBuilder.java:595)
2022-03-31 10:37:05,054 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:221)
2022-03-31 10:37:05,054 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:222)
2022-03-31 10:37:05,145 | INFO | main | JDK version is SUN |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:526)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is null
isInitiator true KeyTab is D:\Code\sample_project\src\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is admintest@HADOOP.COM tryFirstPass is false useFirstPass is false
storePass is false clearPass is false
principal is admintest@HADOOP.COM
Will use keytab
Commit Succeeded

2022-03-31 10:37:05,201 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:537)
2022-03-31 10:37:05,205 | INFO | RefreshTGTThread | TGT refresh thread started |
org.elasticsearch.client.RestClientBuilder$RefreshTgtThread.run(RestClientBuilder.java:758)
2022-03-31 10:37:05,209 | INFO | RefreshTGTThread | TGT valid starting at: Thu Mar 31 10:34:09 CST
2022 | org.elasticsearch.client.RestClientBuilder.getRefreshTime(RestClientBuilder.java:697)
2022-03-31 10:37:05,210 | INFO | RefreshTGTThread | TGT expires: Fri Apr 01 10:34:09 CST 2022
| org.elasticsearch.client.RestClientBuilder.getRefreshTime(RestClientBuilder.java:698)
2022-03-31 10:37:05,210 | INFO | RefreshTGTThread | TGT refresh sleeping until: Fri Apr 01 05:46:09 CST
2022 | org.elasticsearch.client.RestClientBuilder$RefreshTgtThread.run(RestClientBuilder.java:765)
2022-03-31 10:37:05,262 | INFO | main | Success to get the service realm HADOOP.COM |
org.elasticsearch.client.RestClientBuilder.getRealm(RestClientBuilder.java:899)
2022-03-31 10:37:05,262 | INFO | main | Initialize the client successfully. |
org.elasticsearch.client.RestClientBuilder.authenticate(RestClientBuilder.java:753)
2022-03-31 10:37:06,275 | INFO | main | I/O exception (org.apache.http.NoHttpResponseException) caught
when processing request to {s}->https://8.5.164.12:24100: The target server failed to respond |
org.apache.http.impl.execchain.RetryExec.execute(RetryExec.java:97)
2022-03-31 10:37:06,276 | INFO | main | Retrying request to {s}->https://8.5.164.12:24100 |
org.apache.http.impl.execchain.RetryExec.execute(RetryExec.java:113)
2022-03-31 10:37:10,194 | INFO | main | Create index successful by high level client. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.createIndexForHighLevel(SqlMain.java:77)
2022-03-31 10:37:12,271 | INFO | main | Create index successful by high level client. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.createIndexForHighLevel(SqlMain.java:77)
2022-03-31 10:37:13,065 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:13,771 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:14,366 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:14,798 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:15,230 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:15,715 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:16,353 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:17,008 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:17,495 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:17,991 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:18,973 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:19,348 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:19,699 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:19,947 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
```

```
2022-03-31 10:37:20,207 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:20,431 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:20,716 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:21,006 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:21,229 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:21,470 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:37:22,910 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.basicSearch(SqlSearch.java:30)
2022-03-31 10:37:23,097 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.insteadFieldSearch(SqlSearch.java:57)
2022-03-31 10:37:23,302 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.distinctSearch(SqlSearch.java:71)
2022-03-31 10:37:23,507 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.whereAndLimitSearch(SqlSearch.java:86)
2022-03-31 10:37:23,730 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.groupByAndAliasSearch(SqlSearch.java:100)
2022-03-31 10:37:24,560 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.functionsSearch(SqlSearch.java:114)
2022-03-31 10:37:25,039 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.havingSearch(SqlSearch.java:129)
2022-03-31 10:37:25,130 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.orderBySearch(SqlSearch.java:143)
2022-03-31 10:37:25,554 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.joinSearch(SqlSearch.java:158)
2022-03-31 10:37:25,675 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.subQuerySearch(SqlSearch.java:174)
2022-03-31 10:37:25,752 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.matchSearch(SqlSearch.java:188)
2022-03-31 10:37:25,859 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.querySearch(SqlSearch.java:203)
2022-03-31 10:37:26,282 | INFO | main | Delete data successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlDelete.deleteData(SqlDelete.java:36)
2022-03-31 10:37:26,521 | INFO | main | Delete index is successful. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.delete.DeleteIndex.deleteIndex(DeleteIndex.java:36 )
2022-03-31 10:37:26,736 | INFO | main | Delete index is successful. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.delete.DeleteIndex.deleteIndex(DeleteIndex.java:36 )
```

Process finished with exit code 0

## Log Description

The log level is **INFO** by default. You can view more detailed information by changing the log level (**TRACE**, **DEBUG**, **INFO**, **WARN**, and **ERROR**). You can modify the **log4j.properties** file to change log levels, for example:

```
# This sets the global logging level and specifies the appenders
log4j.rootLogger=INFO, theConsoleAppender, R

# settings for the console appender
log4j.appender.theConsoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.theConsoleAppender.Threshold=INFO
log4j.appender.theConsoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.theConsoleAppender.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t
| %m | %l%n

# R is set to be a File appender using a PatternLayout.
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.Append=true
log4j.appender.R.Threshold=debug
log4j.appender.R.MaxFileSize=10240KB
```

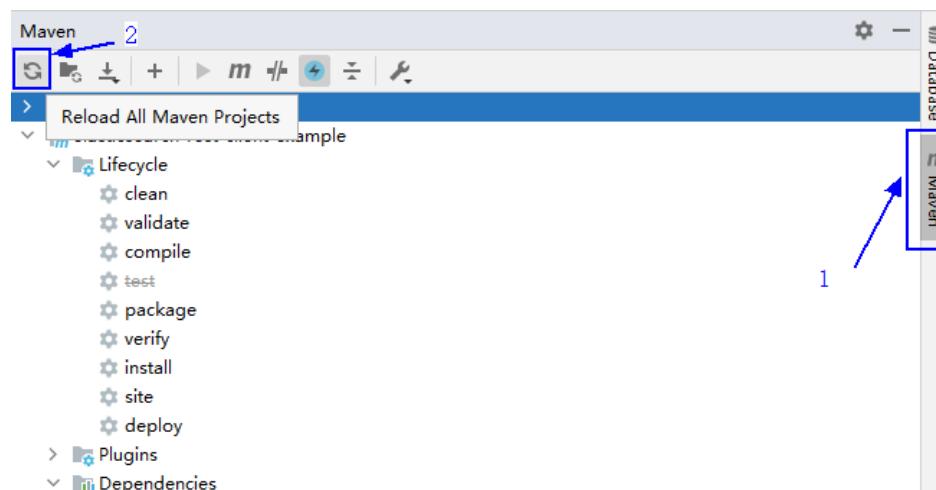
```
log4j.append.R.MaxBackupIndex=10  
log4j.append.R.File=logs/es-example.log  
log4j.append.R.layout=org.apache.log4j.PatternLayout  
log4j.append.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n
```

### 1.7.4.1.2 Commissioning a SpringBoot Program

#### Compiling and Running Applications

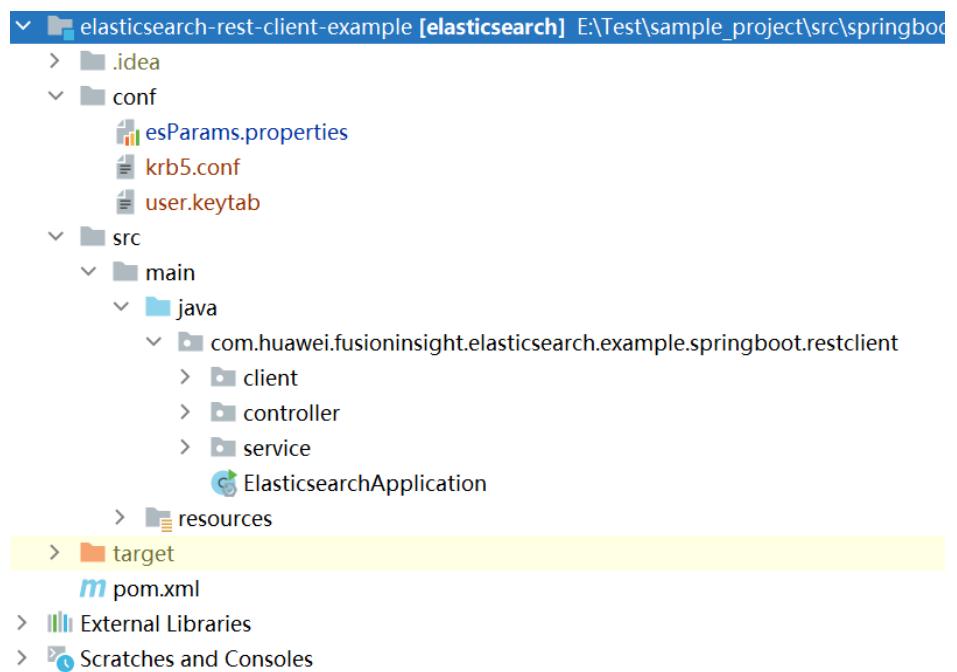
You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

1. Click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependency.



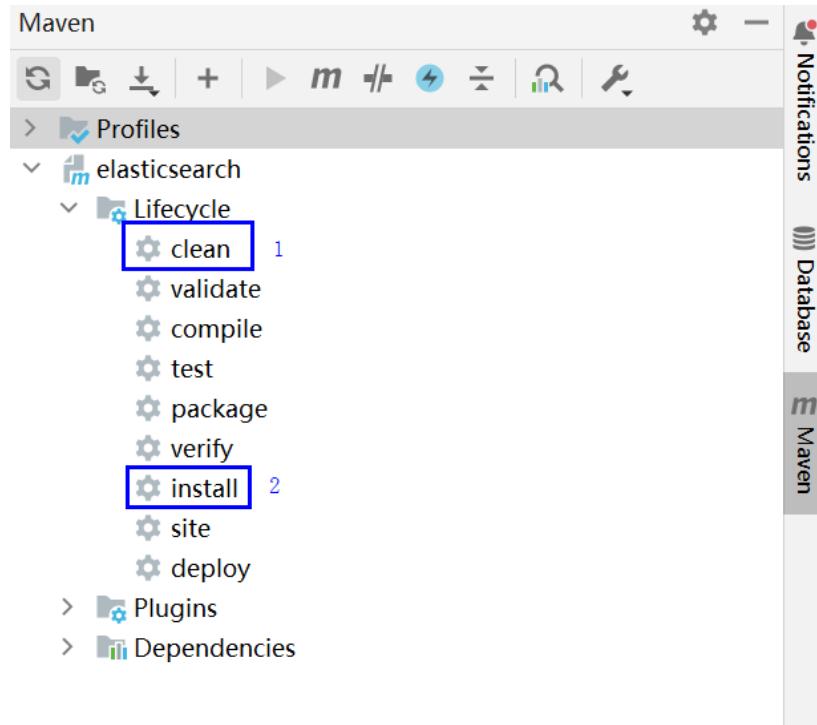
2. Compile and run an application.

- a. The following figure shows the file list after the configuration file has been placed and the code has been modified to match the login user.



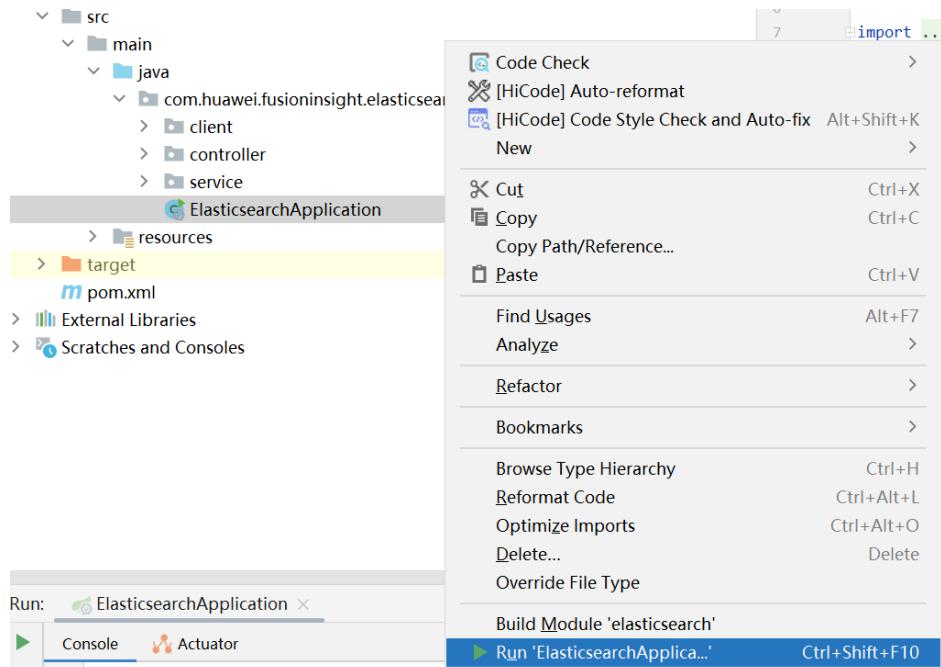
- b. Compile an application.

Double-click **clean** and **install** to run the **maven clean** and **maven install** commands. After the compilation is complete, "Build Success" is displayed.



- c. Run the program.

Right-click **ElasticsearchApplication** and choose **Run'ElasticsearchApplication...'** from the shortcut menu to start the SpringBoot service.



3. Call the SpringBoot sample API of Elasticsearch to trigger the running of the sample code.

After the SpringBoot service is started, enter the link of the specific operation to be performed in the address box of the browser. For example, to query the cluster health status, enter <http://localhost:8080/elasticsearch/cluster/health> in the address box of the browser. The following result is returned:

```
{"cluster_name":"elasticsearch_cluster","status":"green","timed_out":false,"number_of_nodes":6,"number_of_data_nodes":3,"active_primary_shards":11,"active_shards":22,"relocating_shards":0,"initializing_shards":0,"unassigned_shards":0,"delayed_unassigned_shards":0,"number_of_pending_tasks":0,"number_of_in_flight_fetch":0,"task_max_waiting_in_queue_millis":0,"active_shards_percent_as_number":100.0}
```

**Table 1-22** lists the operations that can be performed when Elasticsearch interconnects with SpringBoot.

**Table 1-22** Operations for interconnecting Elasticsearch with SpringBoot

Method	Browser Link	Description
getElasticsearch	<a href="http://localhost:8080/elasticsearch/">http://localhost:8080/elasticsearch/</a>	Obtain cluster information such as clusterName, clusterUuid, and nodeName.
clusterHealth	<a href="http://localhost:8080/elasticsearch/cluster/health">http://localhost:8080/elasticsearch/cluster/health</a>	Obtain the cluster health status.
indexByJson	<a href="http://localhost:8080/elasticsearch/indexByJson?index=/Index name">http://localhost:8080/elasticsearch/indexByJson?index=/Index name</a>	Write data in JSON format.
indexByMap	<a href="http://localhost:8080/elasticsearch/indexByMap?index=/Index name">http://localhost:8080/elasticsearch/indexByMap?index=/Index name</a>	Write data in the Map format.
indexByXContentBuilder	<a href="http://localhost:8080/elasticsearch/indexByXContentBuilder?index=/Index name">http://localhost:8080/elasticsearch/indexByXContentBuilder?index=/Index name</a>	Write data in the XContentBuilder format.
update	<a href="http://localhost:8080/elasticsearch/update?index=/Index name&amp;id=ID value">http://localhost:8080/elasticsearch/update?index=/Index name&amp;id=ID value</a>	Update index data.
bulk	<a href="http://localhost:8080/elasticsearch/bulk?index=/Index name">http://localhost:8080/elasticsearch/bulk?index=/Index name</a>	Write data in batches.
getIndex	<a href="http://localhost:8080/elasticsearch/getIndex?index=/Index name&amp;id=ID value">http://localhost:8080/elasticsearch/getIndex?index=/Index name&amp;id=ID value</a>	Query index information.
search	<a href="http://localhost:8080/elasticsearch/search?index=/Index name">http://localhost:8080/elasticsearch/search?index=/Index name</a>	Search for related document information under a specified index.

Method	Browser Link	Description
scroll	<code>http://localhost:8080/elasticsearch/scroll?index=/index name</code>	Search for document information under a specified index using a cursor. You can disable the cursor by modifying <b>scrollId</b> .
deleteIndex	<code>http://localhost:8080/elasticsearch/deleteIndex?index=/index name</code>	Delete an index.

## 1.7.4.2 Commissioning Applications in Linux

### 1.7.4.2.1 Commissioning a RestClient Application

Elasticsearch application programs can run in a Linux environment. After the code development is complete, you can upload a .jar package to the prepared Linux running environment, to run the applications.

#### Prerequisites

JDK is installed in the Linux environment. The version must be the same as that of the JDK used to export the .jar package from IntelliJ IDEA and Java environment variables have been set.

### Compiling and Running the Program

#### Step 1 Export a JAR package.

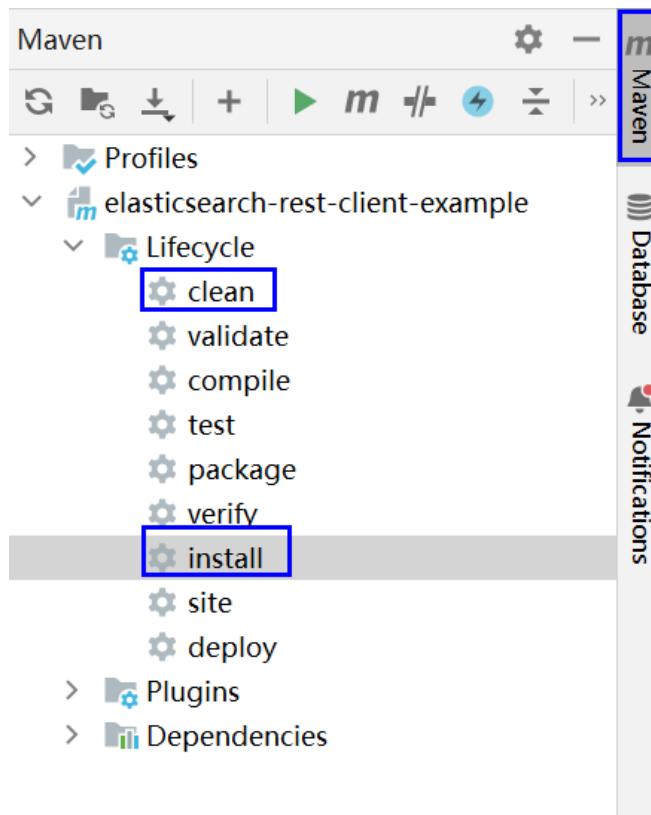
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 1-33 Maven tools: clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 1-34 Enter mvn clean compile in the IDEA Terminal text box.



After the compilation is completed, the message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

### Step 2 Generate the **libs** on which the running depends.

Go to the local root directory of the project and run the following command in the Windows command prompt window to generate the **libs** on which the running depends:

**mvn dependency:copy-dependencies -DoutputDirectory=libs**

### Step 3 Run the .jar package

- Copy the directories **libs** and **conf** required for the development environment to any directory (For example, /opt/elasticsearch-rest-client-example) in the

Linux operating environment. Then, copy the **.jar** package generated in the application environment to **/opt/elasticsearch-rest-client-example/libs/**.

 **NOTE**

Modify the **esParams.properties** file in the **conf** directory based on the site requirements. For details, see [Table 1-14](#).

2. Switch to the code directory `cd /opt/elasticsearch-rest-client-example`.
3. Run the following Java command:

- Run Low Level RestClient code

```
java -cp /opt/elasticsearch-rest-client-example/conf/*:/opt/
elasticsearch-rest-client-example/libs/*
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.L
owLevelRestClientAllRequests
```

- Run High Level RestClient code

```
java -cp /opt/elasticsearch-rest-client-example/conf/*:/opt/
elasticsearch-rest-client-example/libs/*
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.
HighLevelRestClientAllRequests
```

- Run JDBC Rest Client code

```
java -cp /opt/elasticsearch-rest-client-example/conf/*:/opt/
elasticsearch-rest-client-example/libs/*
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain
```

----End

## Viewing the Commissioning Result

After the Elasticsearch ends running, you can view the running status using either of the following methods:

- View the running result.
- Obtain the Elasticsearch log **/opt/elasticsearch-example/logs/es-example.log** in the **logs** directory.

After a complete sample of the Low Level RestClient is run, operation results are as follows:

```
2019-01-09 19:08:50,675 | INFO | main | Start to do low level rest client request ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.main(LowLevelRestClientAllRequests.java:449)
2019-01-09 19:08:50,690 | INFO | main | esServerHost:10.1.1.1:24100 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:73)
2019-01-09 19:08:50,690 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:74)
2019-01-09 19:08:50,690 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:75)
2019-01-09 19:08:50,690 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:76)
2019-01-09 19:08:50,690 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:77)
2019-01-09 19:08:50,690 | INFO | main | shardNum:5 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:78)
```

```
2019-01-09 19:08:50,690 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:79)
2019-01-09 19:08:50,690 | INFO | main | isSecureMode:true |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:80)
2019-01-09 19:08:50,690 | INFO | main | oneCommit:5 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:81)
2019-01-09 19:08:50,690 | INFO | main | totalRecordNumber:10 |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:82)
2019-01-09 19:08:50,690 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.initProperties(LowLevelRestClientAllRequests.java:83)
2019-01-09 19:08:50,690 | INFO | main | FilePath is D:\eclipse-workspace5\elasticsearch-rest-client-
example\conf |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.setSecConfig(LowLevelRestClientAllRequests.java:116)
2019-01-09 19:08:50,690 | INFO | main | jaasPath is D:\eclipse-workspace5\elasticsearch-rest-client-
example\conf\es.h00423283.jaas.conf |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:170)
2019-01-09 19:08:50,690 | INFO | main | keytabPath is D:\eclipse-workspace5\elasticsearch-rest-
client-example\conf\user.keytab |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:171)
2019-01-09 19:08:50,706 | INFO | main | es.security.indication is true |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.setSecConfig(LowLevelRestClientAllRequests.java:123)
2019-01-09 19:08:53,800 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:08:53,800 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:08:53,800 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-09 19:08:53,800 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:08:53,800 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:08:53,800 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is
null isInitiator true KeyTab is D:\eclipse-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass
is false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:08:54,159 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:08:54,440 | INFO | main | Success to get the service realm elasticsearch/
hadoop.<system domain name>@<system domain name> |
org.elasticsearch.client.RestClientBuilder.getServerRealm(RestClientBuilder.java:576)
2019-01-09 19:08:54,440 | INFO | main | Initialize the client successfully. |
org.elasticsearch.client.RestClientBuilder.authenticate(RestClientBuilder.java:495)
2019-01-09 19:08:54,440 | INFO | main | The Low Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.getRestClient(LowLevelRestClientAllRequests.java:171)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is
null isInitiator true KeyTab is D:\eclipse-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass
is false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:08:54,534 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:08:54,815 | INFO | main | QueryClusterInfo successful. |
```

```
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.queryClusterInfo(LowLevelRestClientAllRequests.java:188)
2019-01-09 19:08:54,815 | INFO | main | QueryClusterInfo response entity is :
{
    "cluster_name" : "elasticsearch_cluster",
    "status" : "yellow",
    "timed_out" : false,
    "number_of_nodes" : 3,
    "number_of_data_nodes" : 1,
    "active_primary_shards" : 15,
    "active_shards" : 15,
    "relocating_shards" : 0,
    "initializing_shards" : 0,
    "unassigned_shards" : 15,
    "delayed_unassigned_shards" : 0,
    "number_of_pending_tasks" : 0,
    "number_of_in_flight_fetch" : 0,
    "task_max_waiting_in_queue_millis" : 0,
    "active_shards_percent_as_number" : 50.0
}
|
com.huawei.fusioninsight.elasticsearch.example.lowlevelrestclient.allrequests.LowLevelRestClientAllReq
uests.queryClusterInfo(LowLevelRestClientAllRequests.java:192)
```

After a complete sample of the High Level RestClient is run, some operation results are displayed on the console:

```
2019-01-09 19:12:23,206 | INFO | main | Start to do high level rest client request !
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.main(HighLevelRestClientAllRequests.java:500)
2019-01-09 19:12:23,222 | INFO | main | esServerHost:192.168.61.68:24100 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:94)
2019-01-09 19:12:23,222 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:95)
2019-01-09 19:12:23,222 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:96)
2019-01-09 19:12:23,222 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:97)
2019-01-09 19:12:23,222 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:98)
2019-01-09 19:12:23,222 | INFO | main | shardNum:5 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:99)
2019-01-09 19:12:23,222 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:100)
2019-01-09 19:12:23,222 | INFO | main | isSecureMode:true |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:101)
2019-01-09 19:12:23,222 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.initProperties(HighLevelRestClientAllRequests.java:102)
2019-01-09 19:12:23,253 | INFO | main | Path is D:\eclipse-workspace5\elasticsearch-rest-client-
example\conf |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.setSecConfig(HighLevelRestClientAllRequests.java:134)
2019-01-09 19:12:23,253 | INFO | main | jaasPath is D:\eclipse-workspace5\\elasticsearch-rest-client-
example\\conf\\es.h00423283.jaas.conf |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:170)
2019-01-09 19:12:23,253 | INFO | main | keytabPath is D:\\eclipse-workspace5\\\\elasticsearch-rest-
client-example\\conf\\user.keytab |
com.huawei.fusioninsight.elasticsearch.example.util.LoginUtil.setJaasFile(LoginUtil.java:171)
2019-01-09 19:12:23,253 | INFO | main | es.security.indication is true |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.setSecConfig(HighLevelRestClientAllRequests.java:139)
2019-01-09 19:12:26,379 | INFO | main | esSecConfig is true |
```

```
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:12:26,379 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:12:26,379 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-09 19:12:26,379 | INFO | main | esSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-09 19:12:26,379 | INFO | main | systemSecConfig is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-09 19:12:26,379 | INFO | main | isSecureMode is true |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is
null isInitiator true KeyTab is D:\eclipse-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass
is false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:12:26,754 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:12:27,035 | INFO | main | Success to get the service realm elasticsearch/
hadoop.<system domain name>@<system domain name> |
org.elasticsearch.client.RestClientBuilder.getServerRealm(RestClientBuilder.java:576)
2019-01-09 19:12:27,035 | INFO | main | Initialize the client successfully. |
org.elasticsearch.client.RestClientBuilder.authenticate(RestClientBuilder.java:495)
2019-01-09 19:12:27,644 | INFO | main | The High Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.getHighLevelRestClient(HighLevelRestClientAllRequests.java:185)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is
null isInitiator true KeyTab is D:\eclipse-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass
is false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:12:27,738 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:12:28,207 | INFO | main | ClusterName:[elasticsearch_cluster], clusterUuid:
[TTr6K7laQKuY3yZU28QiZg], nodeName:[EsNode1@192.168.61.68], version:[6.1.3]. |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.queryClusterInfo(HighLevelRestClientAllRequests.java:198)
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is
null isInitiator true KeyTab is D:\eclipse-workspace5\elasticsearch-rest-client-example\conf\user.keytab
refreshKrb5Config is false principal is hmm@<system domain name> tryFirstPass is false useFirstPass
is false storePass is false clearPass is false
principal is hmm@<system domain name>
Will use keytab
Commit Succeeded

2019-01-09 19:12:28,316 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:445)
2019-01-09 19:12:28,722 | INFO | main | IndexByJson response is
IndexResponse[index=huawei1,type=type1,id=1,version=1,result=created,seqNo=0,primaryTerm=1,shard
s={"total":2,"successful":1,"failed":0}]. |
com.huawei.fusioninsight.elasticsearch.example.highlevelrestclient.allrequests.HighLevelRestClientAllRe
quests.indexByJson(HighLevelRestClientAllRequests.java:222)
```

After a complete sample of the JDBC Rest Client is run, some operation results are displayed on the console:

```
2022-03-31 10:37:05,201 | INFO | main | Get kerberos TGT successfully. |
org.elasticsearch.client.RestClientBuilder.getTGT(RestClientBuilder.java:537)
2022-03-31 10:37:05,205 | INFO | RefreshTGTThread | TGT refresh thread started |
org.elasticsearch.client.RestClientBuilder$RefreshTgtThread.run(RestClientBuilder.java:758)
2022-03-31 10:37:05,209 | INFO | RefreshTGTThread | TGT valid starting at: Thu Mar 31 10:34:09
CST 2022 | org.elasticsearch.client.RestClientBuilder.getRefreshTime(RestClientBuilder.java:697)
2022-03-31 10:37:05,210 | INFO | RefreshTGTThread | TGT expires: Fri Apr 01 10:34:09 CST
2022 | org.elasticsearch.client.RestClientBuilder.getRefreshTime(RestClientBuilder.java:698)
```

```
2022-03-31 10:37:05,210 | INFO | RefreshTGTThread | TGT refresh sleeping until: Fri Apr 01 05:46:09  
CST 2022 | org.elasticsearch.client.RestClientBuilder$RefreshTgtThread.run(RestClientBuilder.java:765)  
2022-03-31 10:37:05,262 | INFO | main | Success to get the service realm HADOOP.COM |  
org.elasticsearch.client.RestClientBuilder.getRealm(RestClientBuilder.java:899)  
2022-03-31 10:37:05,262 | INFO | main | Initialize the client successfully. |  
org.elasticsearch.client.RestClientBuilder.authenticate(RestClientBuilder.java:753)  
2022-03-31 10:37:06,275 | INFO | main | I/O exception (org.apache.http.NoHttpResponseException)  
caught when processing request to {s}->https://8.5.164.12:24100: The target server failed to respond |  
org.apache.http.impl.execchain.RetryExec.execute(RetryExec.java:97)  
2022-03-31 10:37:06,276 | INFO | main | Retrying request to {s}->https://8.5.164.12:24100 |  
org.apache.http.impl.execchain.RetryExec.execute(RetryExec.java:113)  
2022-03-31 10:37:10,194 | INFO | main | Create index successful by high level client. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.createIndexForHighLevel(SqlMain.java:77  
)  
2022-03-31 10:37:12,271 | INFO | main | Create index successful by high level client. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.createIndexForHighLevel(SqlMain.java:77  
)  
2022-03-31 10:37:13,065 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:13,771 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:14,366 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:14,798 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:15,230 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:15,715 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:16,353 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:17,008 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:17,495 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:17,991 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:18,973 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:19,348 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:19,699 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:19,947 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:20,207 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:20,431 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:20,716 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:21,006 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:21,229 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:21,470 | INFO | main | Bulk is successful. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)  
2022-03-31 10:37:22,910 | INFO | main | Search succeed. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.basicSearch(SqlSearch.java:30)  
2022-03-31 10:37:23,097 | INFO | main | Search succeed. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.insteadFieldSearch(SqlSearch.java:57)  
2022-03-31 10:37:23,302 | INFO | main | Search succeed. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.distinctSearch(SqlSearch.java:71)  
2022-03-31 10:37:23,507 | INFO | main | Search succeed. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.whereAndLimitSearch(SqlSearch.java:86  
)  
2022-03-31 10:37:23,730 | INFO | main | Search succeed. |  
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.groupByAndAliasSearch(SqlSearch.java:1  
00)
```

```
2022-03-31 10:37:24,560 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.functionsSearch(SqlSearch.java:114)
2022-03-31 10:37:25,039 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.havingSearch(SqlSearch.java:129)
2022-03-31 10:37:25,130 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.orderBySearch(SqlSearch.java:143)
2022-03-31 10:37:25,554 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.joinSearch(SqlSearch.java:158)
2022-03-31 10:37:25,675 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.subQuerySearch(SqlSearch.java:174)
2022-03-31 10:37:25,752 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.matchSearch(SqlSearch.java:188)
2022-03-31 10:37:25,859 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.querySearch(SqlSearch.java:203)
2022-03-31 10:37:26,282 | INFO | main | Delete data successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlDelete.deleteData(SqlDelete.java:36)
2022-03-31 10:37:26,521 | INFO | main | Delete index is successful. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.delete.DeleteIndex.deleteIndex(DeleteIndex.java:36)
2022-03-31 10:37:26,736 | INFO | main | Delete index is successful. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.delete.DeleteIndex.deleteIndex(DeleteIndex.java:36)
```

## Log description

The default log level is **INFO** and more details can be viewed by changing the log level (**TRACE**, **DEBUG**, **INFO**, **WARN**, and **ERROR**). You can modify the **log4j.properties** file to change log levels, for example:

```
# This sets the global logging level and specifies the appenders
log4j.rootLogger=INFO,theConsoleAppender,R

# settings for the console appender
log4j.appender.theConsoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.theConsoleAppender.Threshold=INFO
log4j.appender.theConsoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.theConsoleAppender.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t
| %m | %l%n

# R is set to be a File appender using a PatternLayout.
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.Append=true
log4j.appender.R.Threshold=debug
log4j.appender.R.MaxFileSize=10240KB
log4j.appender.R.MaxBackupIndex=10
log4j.appender.R.File=logs/es-example.log
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n
```

### 1.7.4.2.2 Commissioning a SpringBoot Program

Applications that interconnect Elasticsearch with SpringBoot can run in the Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

## Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

## Compiling and Running Applications

**Step 1** Click Terminal in the lower left corner of the IDEA page to access the terminal. Run the **mvn clean package** command to compile the package.



```
Terminal: Local + v
Windows PowerShell
尝试新的跨平台 PowerShell https://aka.ms/powershell
PS E:\Test\sample_project\src\springboot\elasticsearch-examples\elasticsearch-rest-client-example> mvn clean package
```



If the command output contains "Build Success", the compilation is successful, as shown in the following figure. After the compilation is successful, the es-springboot.jar file is generated in the target directory of the sample project.



```
Terminal: Local + v
[INFO] -- maven-jar-plugin:3.2.2:jar (default-jar) @ elasticsearch ...
[INFO] Building jar: E:\Test\sample_project\src\springboot\elasticsearch-examples\elasticsearch-rest-client-example\target\es-springboot.jar
[INFO]
[INFO] -- spring-boot-maven-plugin:1.3.2.RELEASE:repackage (repackage) @ elasticsearch ...
[INFO]
[INFO] -- spring-boot-maven-plugin:1.3.2.RELEASE:repackage (default) @ elasticsearch ...
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.836 s
[INFO] Finished at: 2023-02-27T10:34:23+08:00
[INFO]
PS E:\Test\sample_project\src\springboot\elasticsearch-examples\elasticsearch-rest-client-example>
```

**Step 2** Create a directory on Linux as the running directory, for example, /opt/es-springboot, and save the es-springboot.jar and conf folders in the target directory to the directory. Modify the esParams.properties file in the conf directory and save the user credential files krb5.conf and user.keytab of Elasticsearch to the conf directory.

#### NOTE

Modify the esParams.properties configuration file in the conf directory based on the site requirements. For details, see [1.2.3.2](#).

**Step 3** Run the JAR file.

1. Switch to the code function directory.

```
cd /opt/es-springboot
```

2. Run the java command to start the SpringBoot service.

```
java -jar es-springboot.jar conf/
```

3. Start another terminal on the current node to access the Elasticsearch client.

```
cd Client installation directory
```

```
source bigdata_env
```

```
kinit esuser
```

4. Perform the curl operation of Elasticsearch. For example, if you run the curl **http://localhost:8080/elasticsearch/cluster/health** command to query the cluster health status, the following information is displayed:

```
{"cluster_name":"elasticsearch_cluster","status":"green","timed_out":false,"number_of_nodes":6,"number_of_data_nodes":3,"active_primary_shards":11,"active_shards":22,"relocating_shards":0,"initializing_shards":0,"unassigned_shards":0,"delayed_unassigned_shards":0,"number_of_pending_tasks":0,"number_of_in_flight_fetch":0,"task_max_waiting_in_queue_millis":0,"active_shards_percent_as_number":100.0}
```

For details about other operations, see [Table 1-23](#).

**Table 1-23** Curl operations for interconnecting Elasticsearch with SpringBoot

Method	Curl Commands on the Linux Client	Description
getElasticsearch	curl 'http://localhost:8080/elasticsearch/'	Obtain cluster information such as <code>clusterName</code> , <code>clusterUuid</code> , and <code>nodeName</code> .
clusterHealth	curl 'http://localhost:8080/elasticsearch/_cluster/health'	Obtain the cluster health status.
indexByJson	curl 'http://localhost:8080/elasticsearch/_indexByJson?index=/Index name'	Write data in JSON format.
indexByMap	curl 'http://localhost:8080/elasticsearch/_indexByMap?index=/Index name'	Write data in the Map format.
indexByXcontentBuilder	curl 'http://localhost:8080/elasticsearch/_indexByXContentBuilder?index=/Index name'	Write data in the XContentBuilder format.
update	curl 'http://localhost:8080/elasticsearch/_update?index=/Index name&id=/D value'	Update index data.
bulk	curl 'http://localhost:8080/elasticsearch/_bulk?index=/Index name'	Write data in batches.
getIndex	curl 'http://localhost:8080/elasticsearch/_getIndex?index=/Index name&id=/D value'	Query index information.
search	curl 'http://localhost:8080/elasticsearch/_search?index=/Index name'	Search for related document information under a specified index.
scroll	curl 'http://localhost:8080/elasticsearch/_scroll?index=/Index name'	Search for document information under a specified index using a cursor. You can disable the cursor by modifying <b>scrollId</b> .
deleteIndex	curl 'http://localhost:8080/elasticsearch/_deleteIndex?index=/Index name'	Delete an index.

**----End**

# 1.8 Flink Development Guide

## 1.8.1 Overview

### 1.8.1.1 Application Development

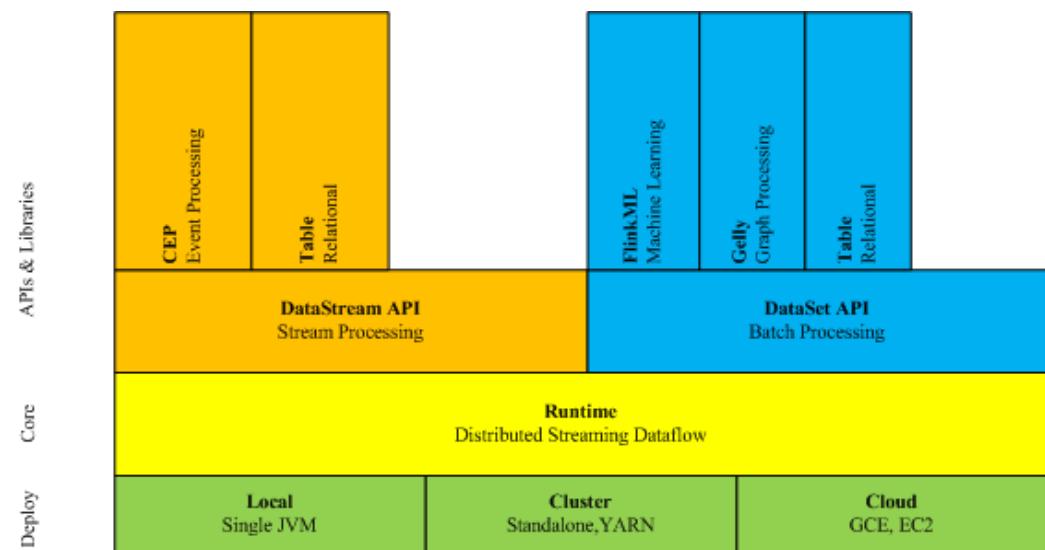
#### Overview

Flink is a unified computing framework that supports both batch processing and stream processing. It provides a stream data processing engine that supports data distribution and parallel computing. Flink features stream processing and is a top open-source stream processing engine in the industry.

Flink provides high-concurrency pipeline data processing, millisecond-level latency, and high reliability, making it suitable for low-latency data processing.

**Figure 1-35** shows the technology stack of Flink.

**Figure 1-35** Technology stack of Flink



The following lists the key features of Flink in the current version:

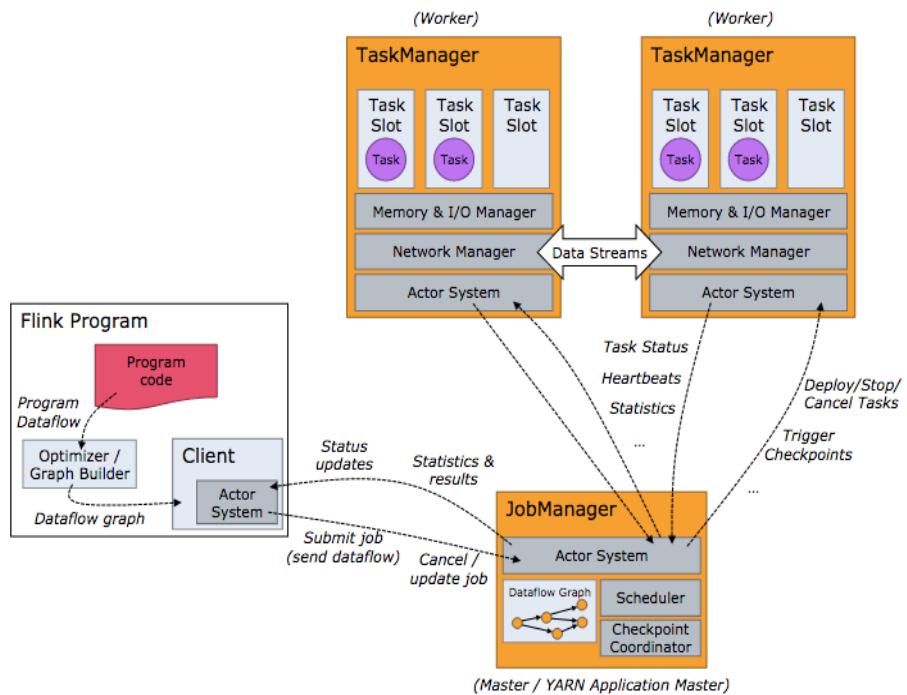
- DataStream
- Checkpoint
- Window
- Job Pipeline
- Configuration Table

For details about other Flink features, see <https://ci.apache.org/projects/flink/flink-docs-release-1.17>.

## Architecture

[Figure 1-36](#) shows the architecture of Flink.

**Figure 1-36** Flink architecture



As shown in [Figure 1-36](#), the entire Flink system consists of three parts:

- **Client**  
Flink client is used to submit jobs (streaming jobs) to Flink.
- **TaskManager**  
TaskManager (also called worker) is a service execution node of Flink. It executes specific tasks. A Flink system could have multiple TaskManagers. These TaskManagers are equivalent to each other.
- **JobManager**  
JobManager (also called master) is a management node of Flink. It manages all TaskManagers and schedules tasks submitted by users to specific TaskManagers. In high-availability (HA) mode, multiple JobManagers are deployed. Among these JobManagers, one of which is selected as the leader, and the others are standby.

Flink provides the following features:

- **Low latency**  
Millisecond-level processing capability.
- **Exactly once**  
Asynchronous snapshot mechanism, ensuring that all data is processed only once.

- High availability  
Leader/Standy JobManagers, preventing single point of failure (SPOF).
- Scale out  
Manual scale out supported by TaskManagers.

## Flink Development APIs

Flink DataStream API can be developed using Scala and Java languages, as shown in [Table 1-24](#).

**Table 1-24** Flink DataStream API

Function	Description
Scala API	API in Scala, which can be used for data processing, such as filtering, joining, windowing, and aggregation.
Java API	API in Java, which can be used for data processing, such as filtering, joining, windowing, and aggregation.

### 1.8.1.2 Basic Concepts

#### Basic Concepts

- **DataStream**  
DataStream is the minimum unit of Flink processing and is one of core concepts of Flink. DataStreams are initially imported from external systems in formats of socket, Kafka, and files. After being processed by Flink, DataStreams are exported to external systems in formats of socket, Kafka, and files.
- **Data Transformation**  
A data transformation is a data processing unit that transforms one or multiple DataStreams into a new DataStream.  
Data transformation can be classified as follows:
  - One-to-one transformation, for example, map.
  - One-to-zero, one-to-one, or one-to-multiple transformation, for example, flatMap.
  - One-to-zero or one-to-one transformation, for example, filter.
  - Multiple-to-one transformation, for example, union.
  - Transformation of multiple aggregations, for example, window and keyby.
- **Checkpoint**  
Checkpoint is the most important Flink mechanism to ensure reliable data processing. Checkpoints ensure that all application statuses can be recovered from a checkpoint in case of failure occurs and data is processed exactly once.
- **Savepoint**  
Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. After the upgrade, you can set the

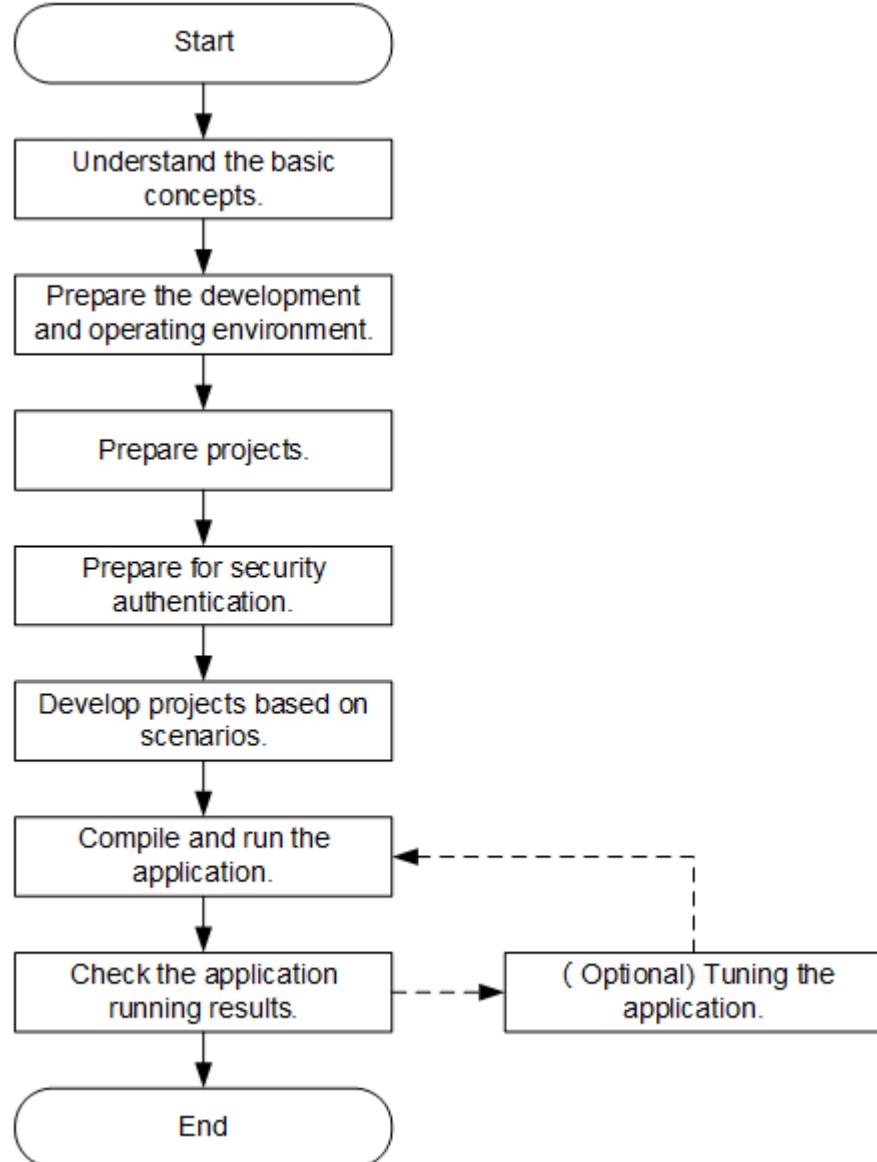
task status to the savepoint storage status and start the restoration, ensuring data continuity.

### 1.8.1.3 Development Process

#### Development Process of a Flink Application

[Figure 1-37](#) shows the Flink development process:

**Figure 1-37** Development process of a Flink application



**Table 1-25** Description of the development process

Phase	Description	Reference
Understand the basic concepts.	Before the development process, you are advised to gain a basic understanding of Flink.	<a href="#">Basic Concepts</a>
Prepare the development and operating environment.	Flink applications can be developed using Scala or Java. You are advised to use IntelliJ IDEA tool to configure the development environment as instructed, based on your development language.  The running environment of Flink is the Flink client. Install and configure the client as instructed.	<a href="#">Preparing Development and Operating Environment</a>
Prepare projects.	Flink provides sample projects for you to import and learn. Alternatively, you can create a Flink project as instructed.	<a href="#">Configuring and Importing Sample Projects</a> <a href="#">Creating a Project (Optional)</a>
Prepare for security authentication.	The security authentication is mandatory if security clusters are used.	<a href="#">Preparing for Security Authentication</a>
Develop the project.	Sample projects in Scala and Java are provided to help you quickly understand programming interfaces of Flink components.	<a href="#">Developing an Application</a>
Compile and run the application.	Guidance is provided for you to compile and run a developed application.	<a href="#">Compiling and Running the Application</a>
Check running results.	Application running results are stored in a path specified by you. You can also view application running status through Apache Flink Dashboard.	<a href="#">Viewing the Debugging Result</a>
Tune the application.	Tune the application to meet certain service requirements.  After the tuning is complete, the application needs to be compiled and run again.	"Flink Performance Tuning" in the Component Operation Guide

## 1.8.2 Environment Preparation

### 1.8.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

**Table 1-26** describes the environment required for application development.

**Table 1-26** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>Development environment: Windows OS. Windows 7 or later is supported.</li><li>Operating environment: Linux OS If the program needs to be commissioned locally, the runtime environment must be able to communicate with the cluster service plane network.</li></ul>
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows: The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IDEA installation and configuration	Used for developing Flink applications. The version must be 2019.1 or other compatible version.
Scala installation	Basic configuration for the Scala development environment. The required version is 2.11.7.

Item	Description
Scala plug-in installation	Basic configuration for the Scala development environment. The required version is 1.5.4.
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
Developer account preparation	Prepare a cluster user for application development and grant permissions to the user by referring to <a href="#">Preparing a Developer Account</a> .
7-Zip	Used to decompress .zip and .rar packages, 7-Zip 16.04 is supported.
Python3	Used to run Flink Python jobs. Python 3.7 or later is required.

## Preparing an Operating Environment

During application development, prepare the code running and commissioning environment to verify that the application is running properly.

To use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

1. Install the client in a directory, for example, `/opt/hadoopclient`, on the node. Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

### NOTE

- Verify that the configuration options in the Flink client configuration file `flink-conf.yaml` are correctly configured. For details, see [Preparing for Security Authentication](#).
  - In security mode, append the service IP address of the node where the client is installed and floating IP address of Manager to the `jobmanager.web.allow-access-address` configuration item in the `flink-conf.yaml` file. Use commas (,) to separate IP addresses.
2. Log in to the FusionInsight Manager portal. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user `root`. Go to the decompression path of the cluster client and copy all configuration files in the `FusionInsight_Cluster_1_Services_ClientConfig\Flink\config` directory to the `conf` directory where the compiled JAR package is stored for subsequent commissioning, for example, `/opt/hadoopclient/conf`.

For example, if the client software package is `FusionInsight_Cluster_1_Services_Client.tar` and the download path is `/tmp/FusionInsight-Client` on the active management node, run the following commands:

```
cd /tmp/FusionInsight-Client
```

```
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Flink/config/* root@IP address of the client node:/opt/hadoopclient/  
conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-27](#) describes the main configuration files.

**Table 1-27** Configuration files

File	Function
core-site.xml	Configures Flink parameters.
hdfs-site.xml	Configures HDFS parameters.
yarn-site.xml	Configures Yarn parameters.
flink-conf.yaml	Configures the Flink client.
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Provides Kerberos server configuration information.

3. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

4. (Optional) To run a Python job, perform the following additional configurations:
  - a. Log in to the node where the Flink client is installed as the **root** user and run the following command to check whether Python 3.7 or a later has been installed:  
**python3 -V**
  - b. Go to the python 3 installation path, for example, **/srv/pyflink-example**, and install the **virtualenv**:  
**cd /srv/pyflink-example**  
**virtualenv venv --python=python3.x**  
**source venv/bin/activate**
  - c. Copy the **Flink/flink/opt/python/apache-flink-\*.tar.gz** file from the client installation directory to **/srv/pyflink-example**:  
**cp /Client installation directory/Flink/flink/opt/python/apache-flink-\*.tar.gz /srv/pyflink-example**
  - d. Install the dependency package. If the following command output is displayed, the installation is successful:

```
python -m pip install apache-flink-libraries-*tar.gz
python -m pip install apache-flink- Version number.tar.gz
...
Successfully built apache-flink
Installing collected packages: apache-flink
Attempting uninstall: apache-flink
  Found existing installation: apache-flink x.xx.x
  Uninstalling apache- flink-x.xx.x.
  Successfully uninstalled apache-flink-x.xx.x
Successfully installed apache-flink-x.xx.x
```

### 1.8.2.2 Configuring and Importing Sample Projects

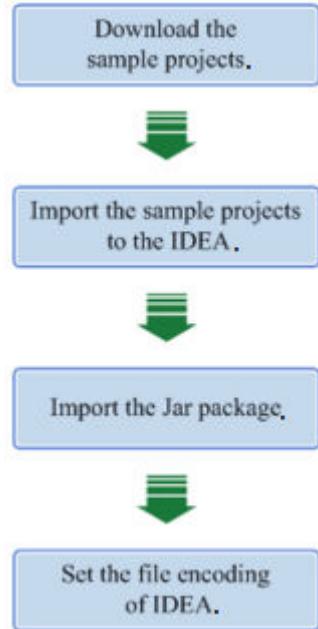
#### Scenarios

Flink provides sample projects for multiple scenarios, including Java and Scala sample projects, to help you quickly learn Flink projects.

Methods to import Java and Scala projects are the same.

The following example describes how to import Java sample code. [Figure 1-38](#) shows the operation process.

**Figure 1-38** Process of importing sample projects



#### Procedure

**Step 1** Obtain the sample project folder **flink-examples-security** in the **src\flink-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

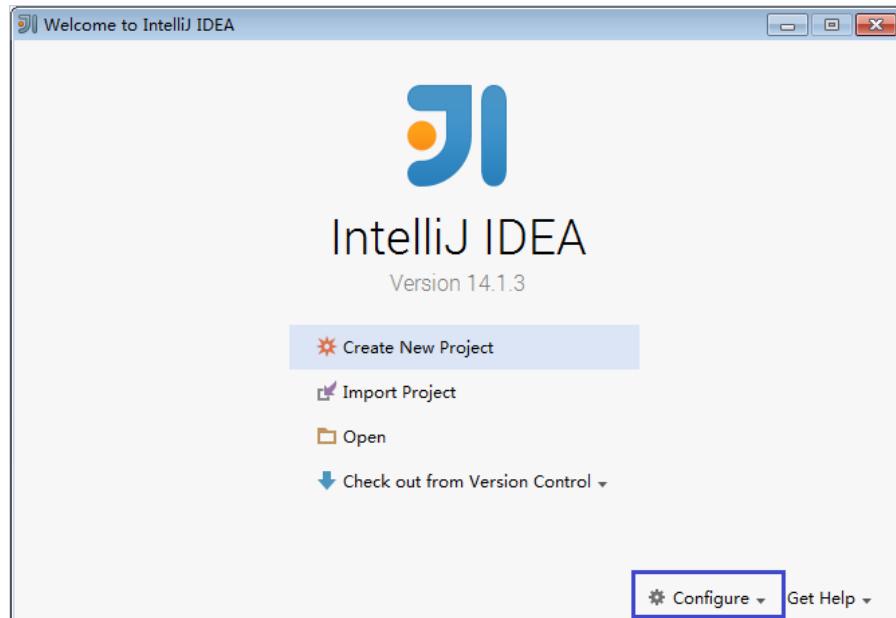
 NOTE

- In security mode, obtain the sample project **flink-examples-security** from the **src\flink-examples** folder.
- In normal mode, obtain the sample project **flink-examples-normal** from the **src\flink-examples** folder.

**Step 2** Before importing the sample project, configure JDK for IntelliJ IDEA.

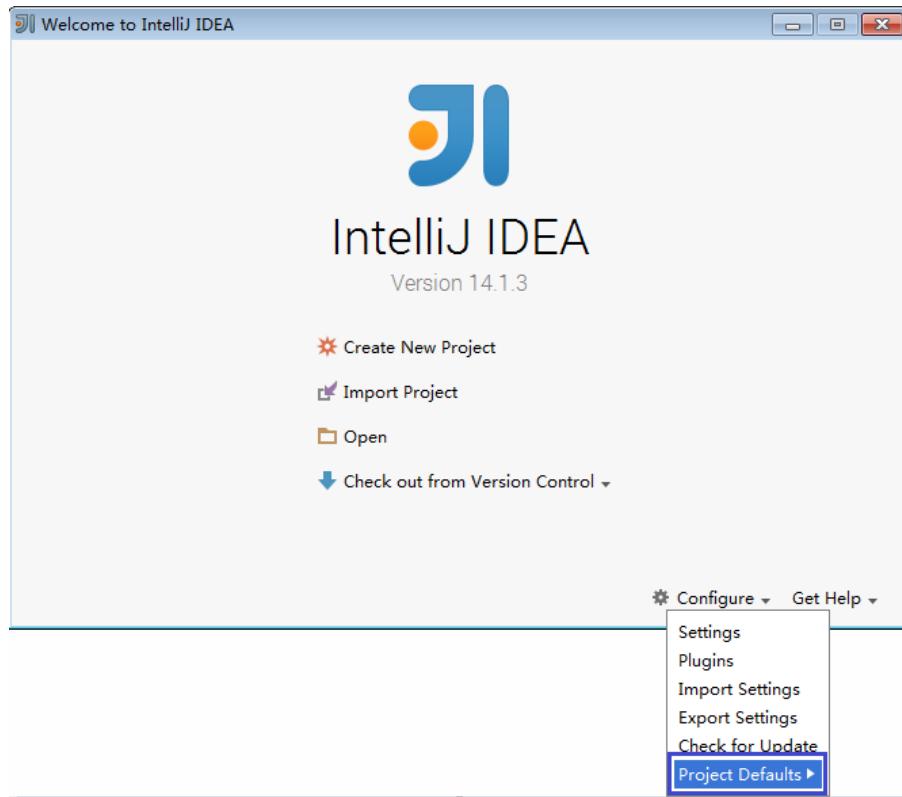
1. Start IntelliJ IDEA and click **Configure**.

**Figure 1-39** Choosing Configure



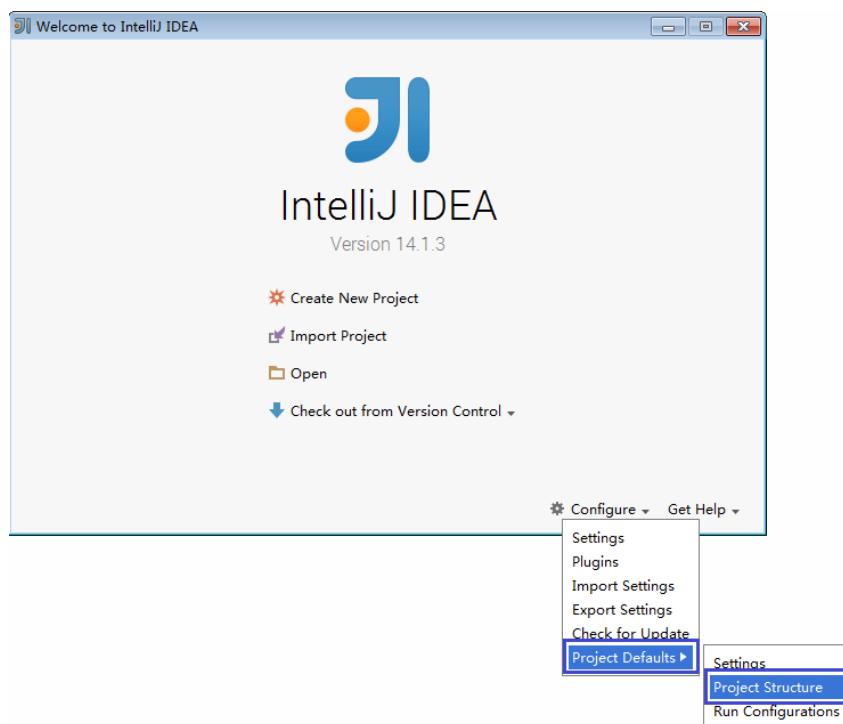
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 1-40 Choosing Project Defaults



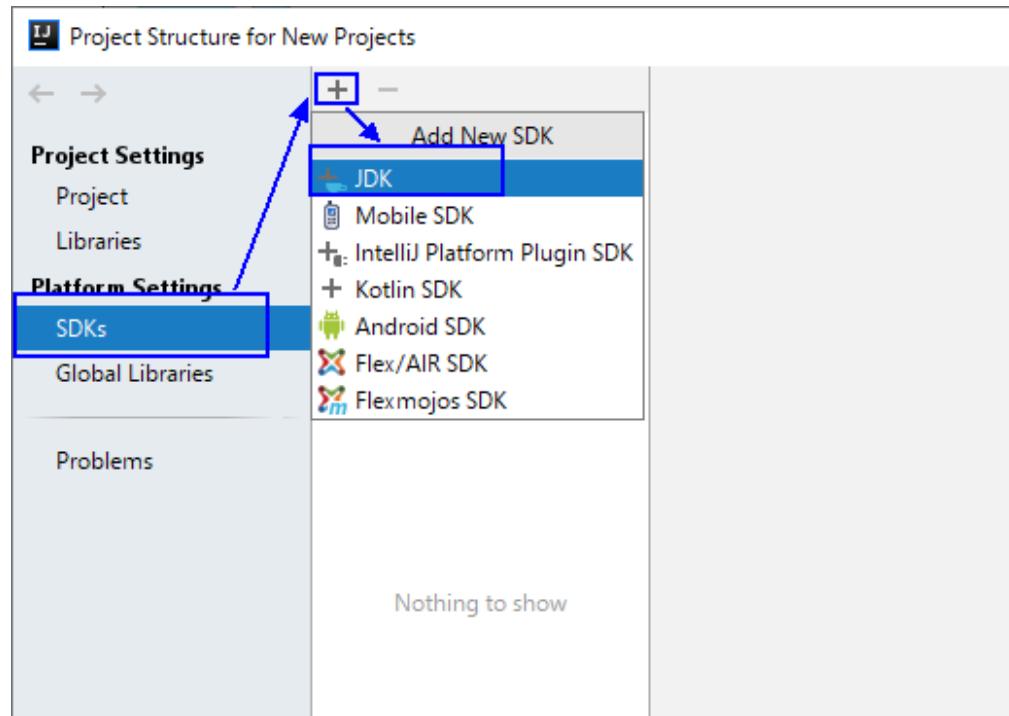
3. Choose **Project Structure** from the **Project Defaults** submenu.

Figure 1-41 Project Defaults



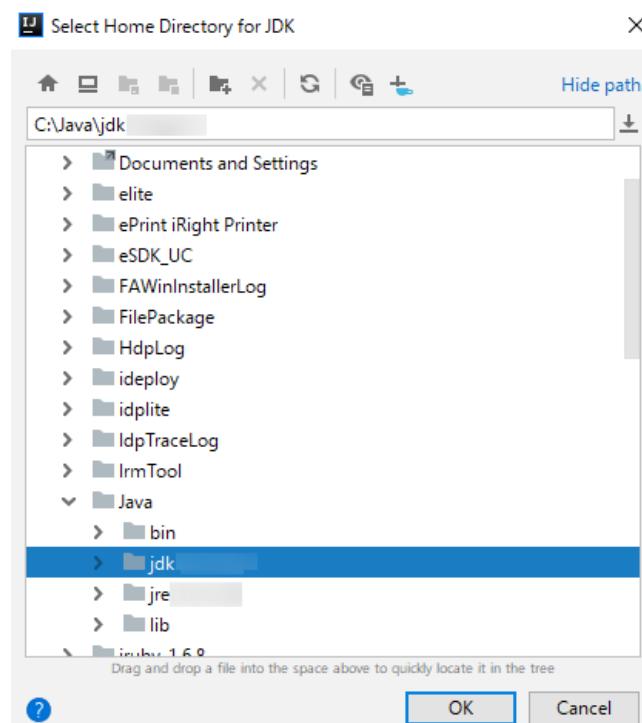
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 1-42 Adding the JDK



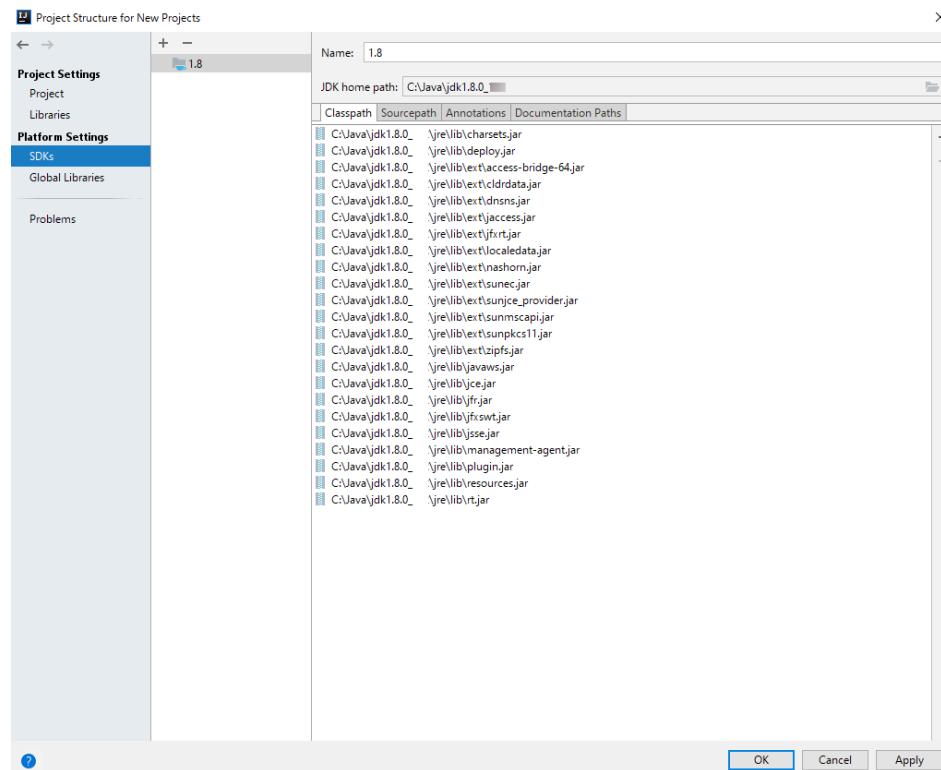
5. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 1-43 Selecting the JDK directory



6. After the JDK is selected, click **OK** to complete the configuration.

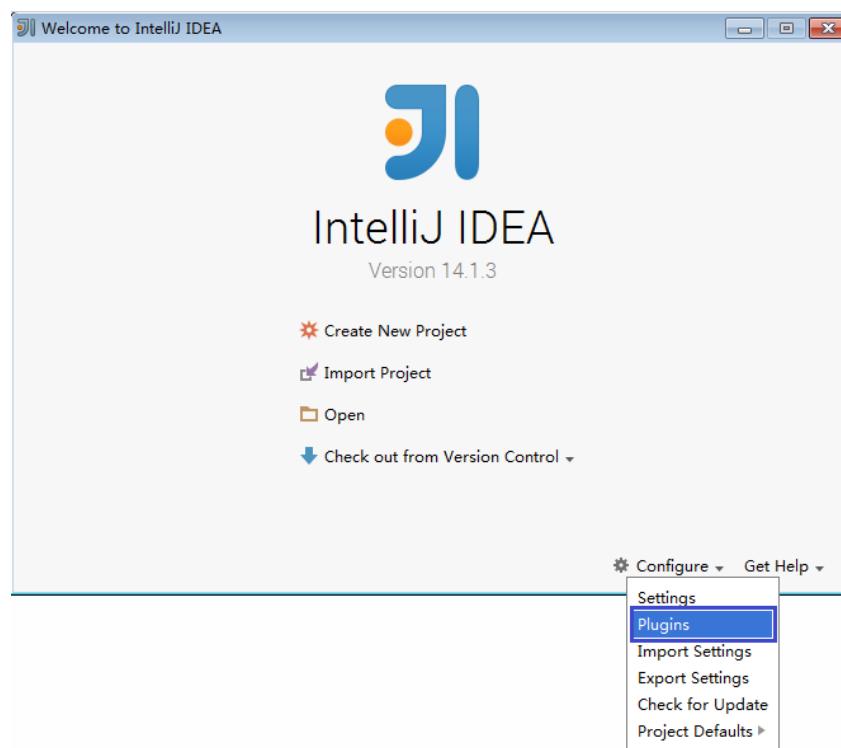
Figure 1-44 Completing the JDK configuration



**Step 3** (Optional) If a Scala sample project is imported, install Scala plug-ins in IntelliJ IDEA.

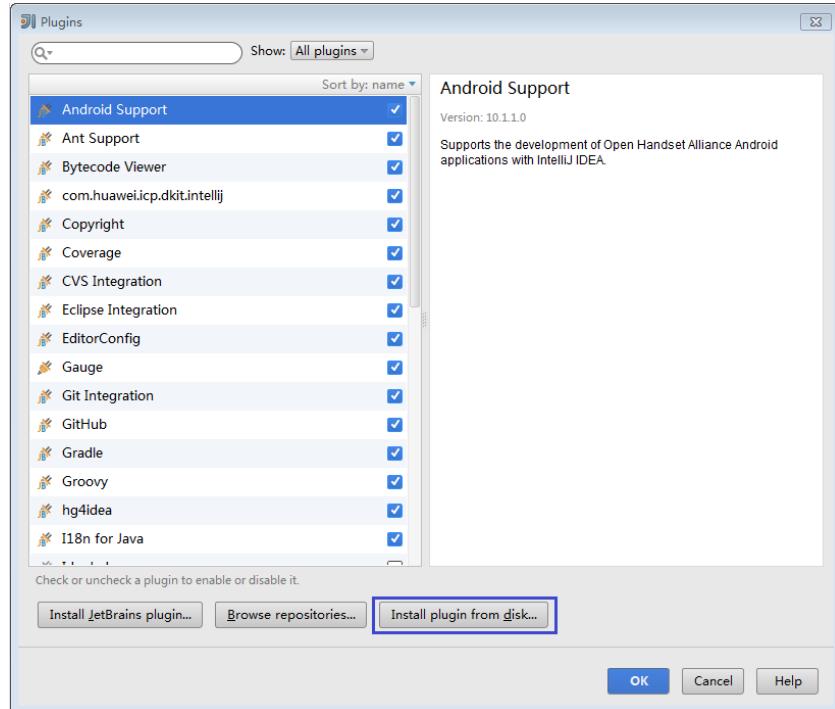
1. Choose **Plugins** from the **Configure** drop-down list.

Figure 1-45 Plugins



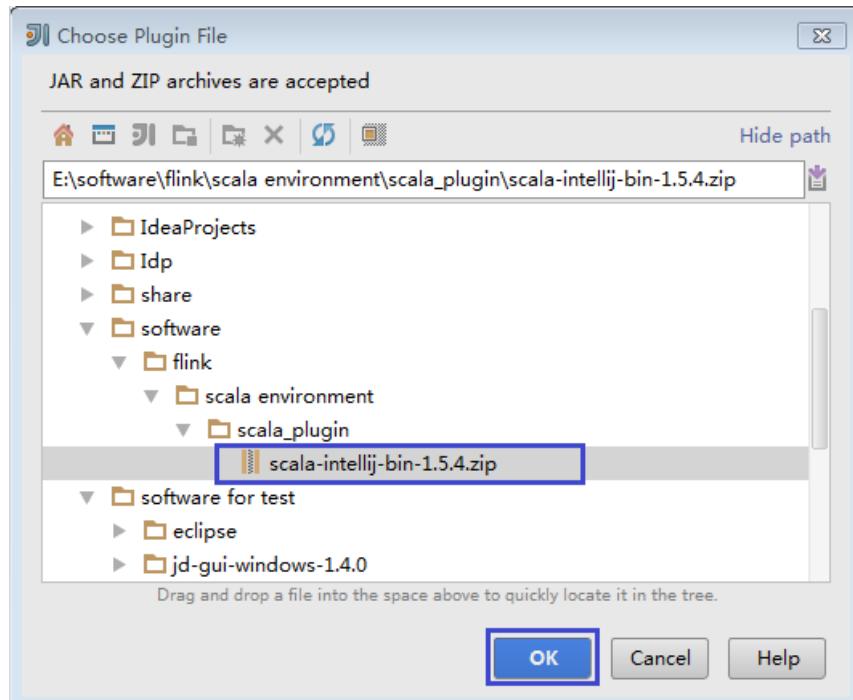
2. On the **Plugins** page, click **Install plugin from disk**.

**Figure 1-46** Install plugin from disk



3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.

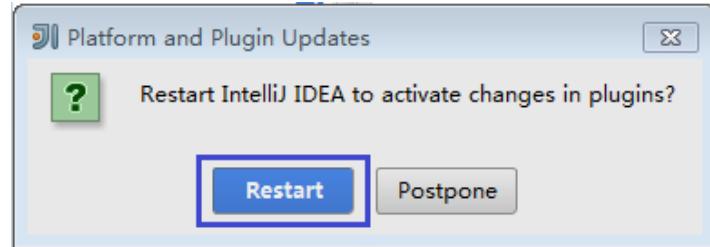
**Figure 1-47** Choose Plugin File



4. On the **Plugins** page, click **Apply** to install the Scala plugins.

5. On the **Platform and Plugin Updates** page that is displayed, click **Restart** to make the configurations take effect.

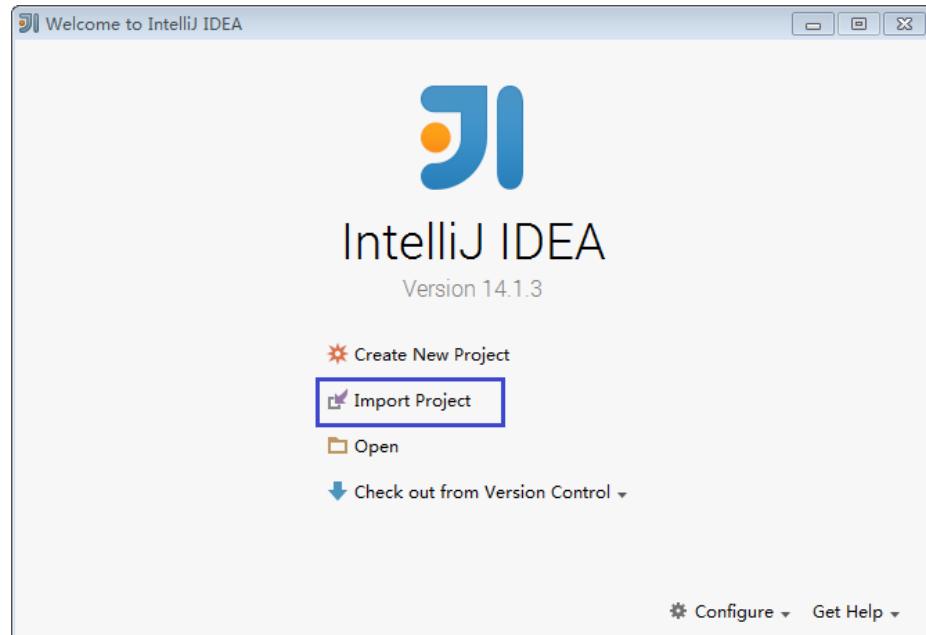
**Figure 1-48** Platform and Plugin Updates



**Step 4** Import the Java sample project to IDEA.

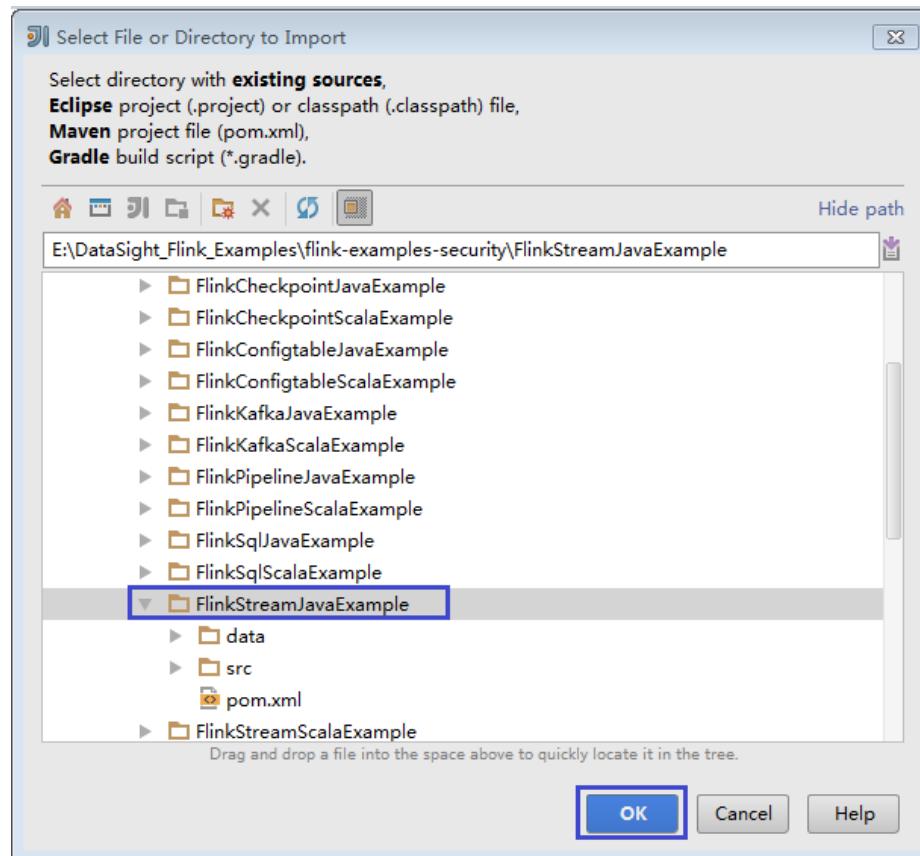
1. Start IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.  
Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File > Import project...** to import a project.

**Figure 1-49** Importing a project (on the Quick Start page)



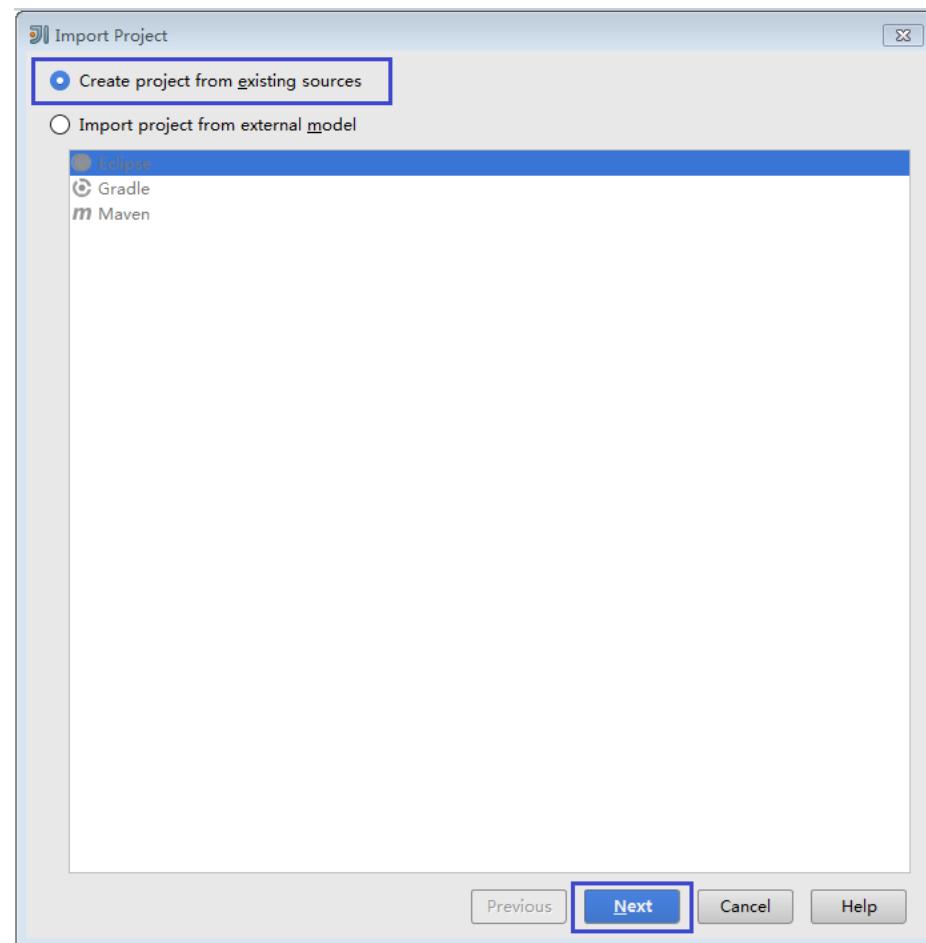
2. Select the directory for storing the imported projects and click **OK**.

Figure 1-50 Select File or Directory to Import



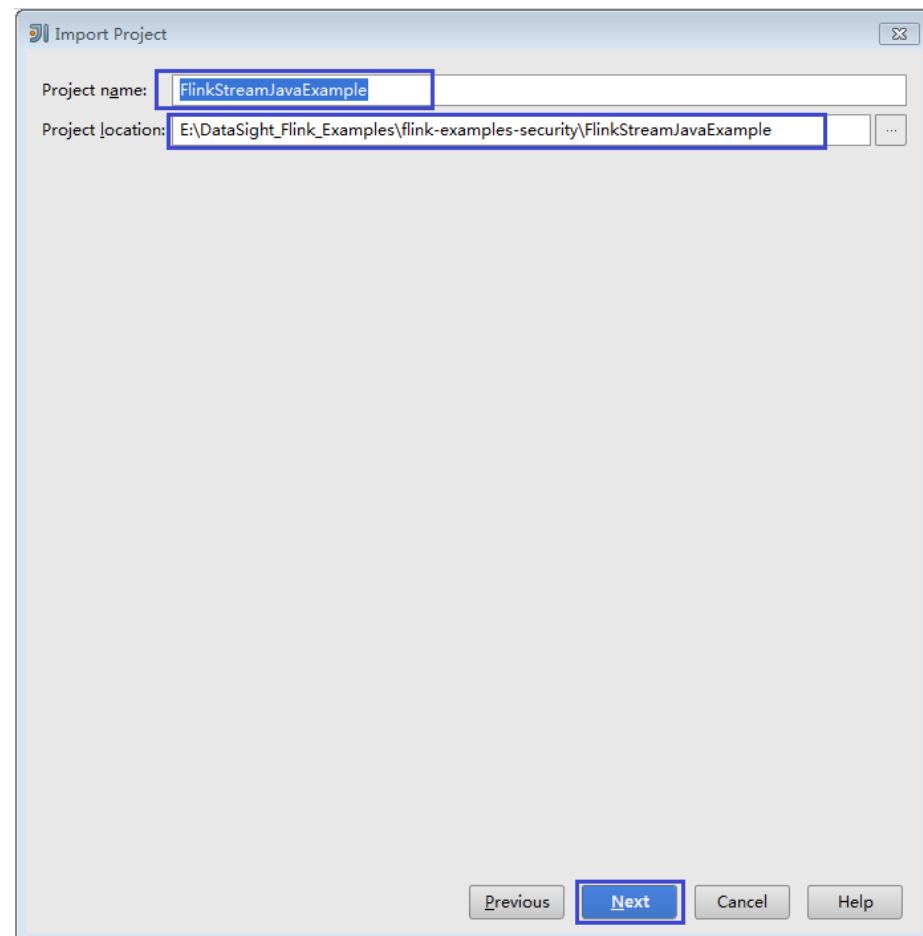
3. Select **Create project from existing sources** and click **Next**.

Figure 1-51 Create project from existing sources



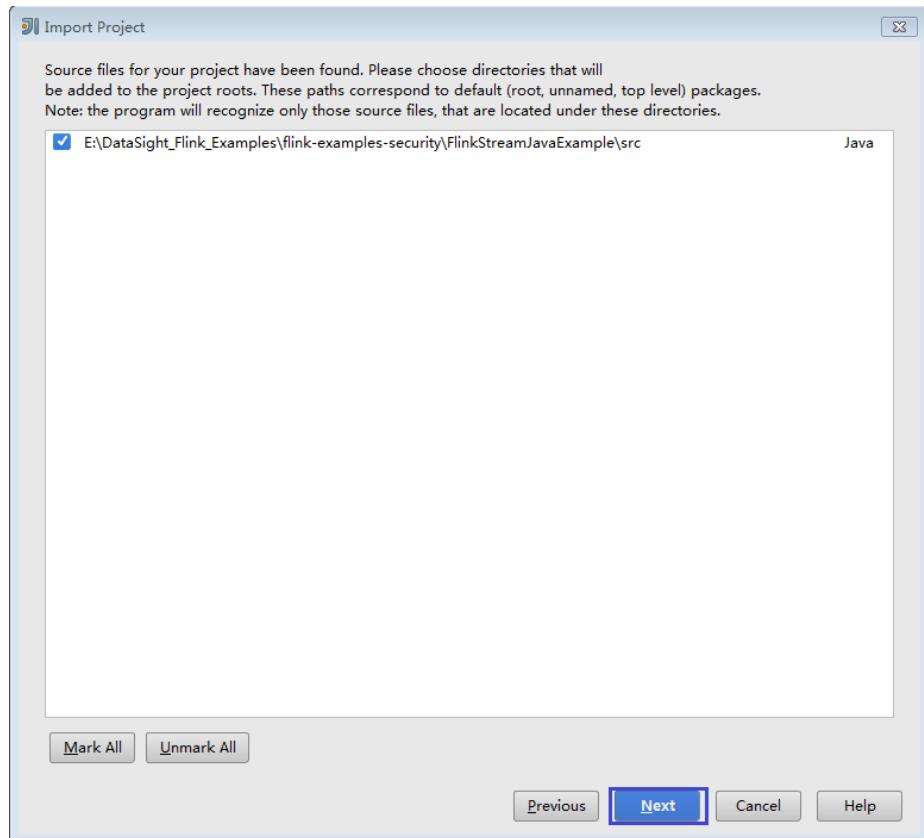
4. Confirm the project location and project name, and click **Next**.

Figure 1-52 Import Project



5. Retain the default value of the **root** directory for the project to be imported and click **Next**.

Figure 1-53 Import Project



6. Retain the default dependency library that is automatically recognized by IDEA and the suggested module structure, and click **Next**.
7. Confirm the JDK to be used by the project and click **Next**.
8. After the import is complete, click **Finish**. The imported sample project is displayed on the IDEA home page.

**Figure 1-54** Completing the import

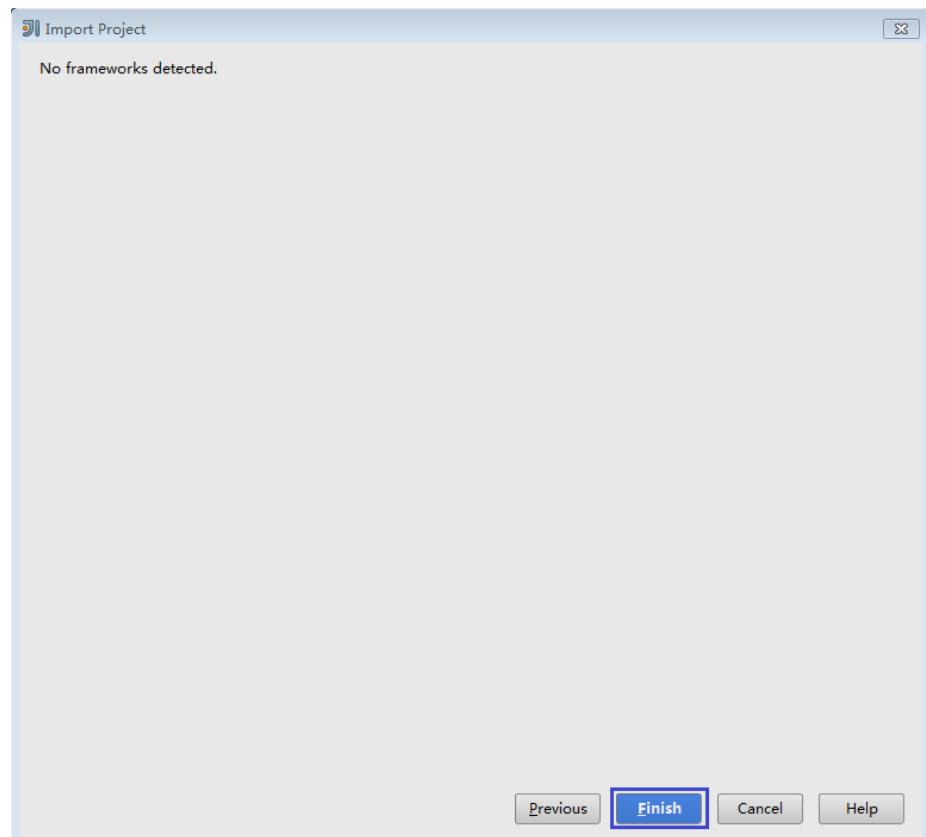
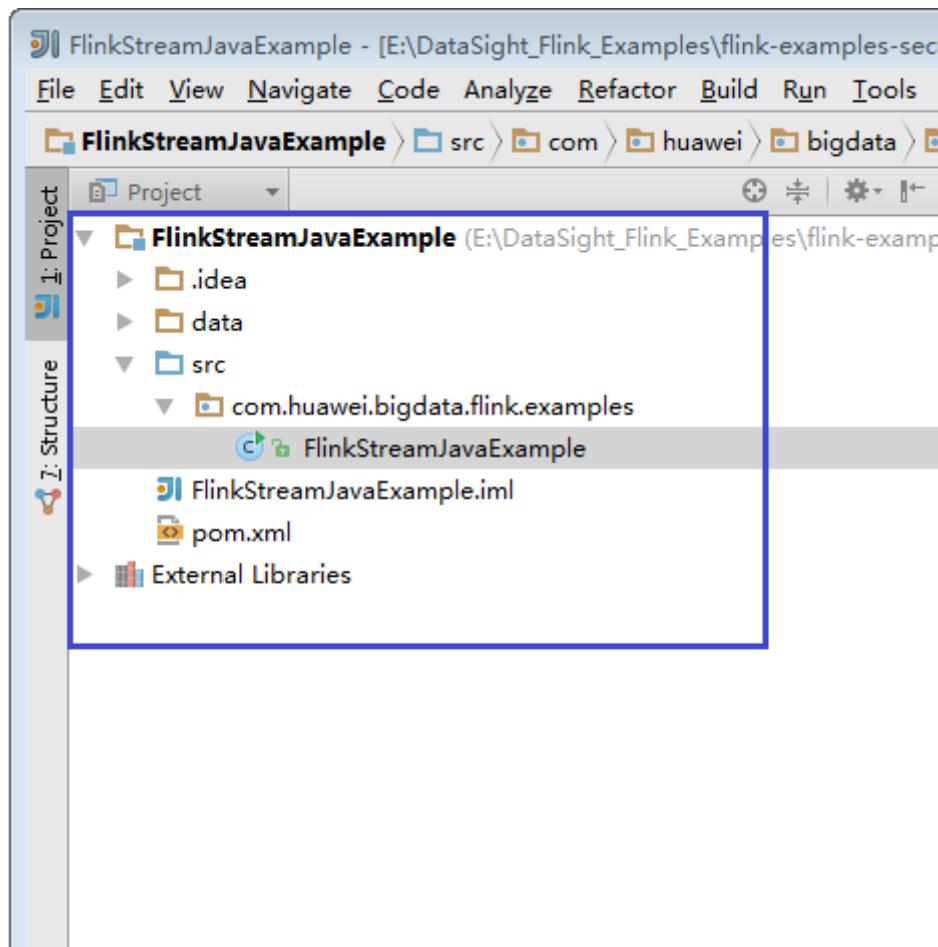


Figure 1-55 Imported project



**Step 5** Import the dependency JAR file for the sample project.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured.

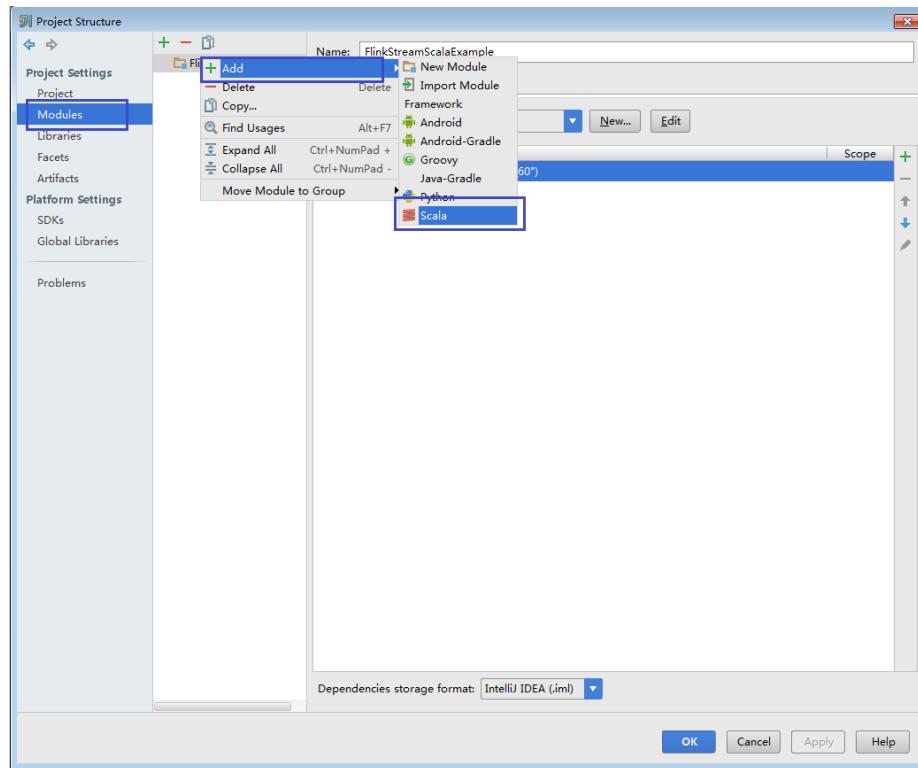
**NOTE**

If other FusionInsight components such as Kafka and Redis are used in the sample code, obtain them from the installation directory of these FusionInsight components. For details about dependency packages of sample projects, see [Reference information about the dependency package for running the sample project](#).

**Step 6** (Optional) If a Scala sample application is imported, configure a language for the project.

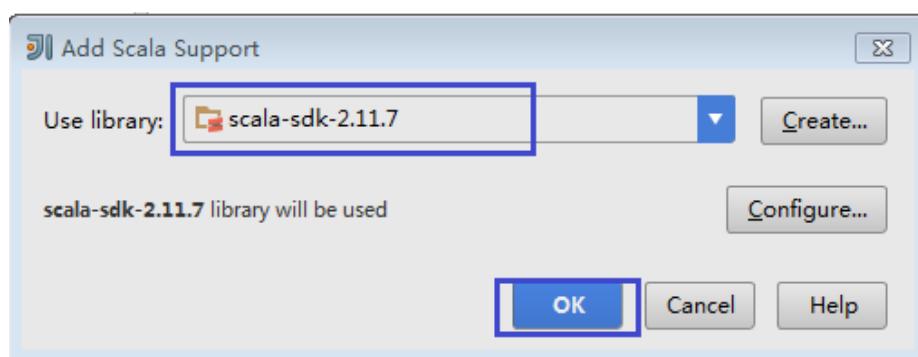
1. On the IDEA home page, choose **File > Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add > Scala**.

Figure 1-56 Selecting Scala



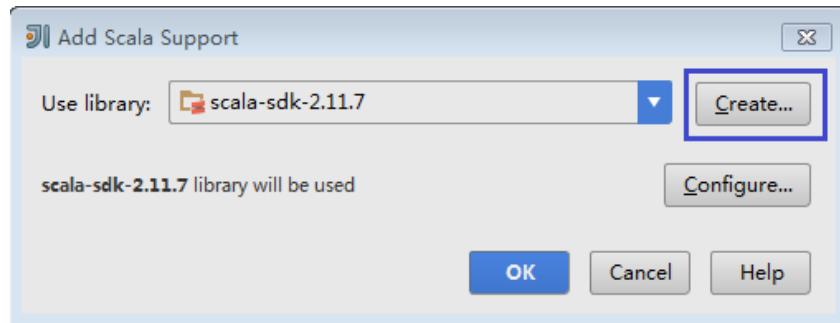
3. Wait until IDEA identifies Scala SDK, select the dependency JAR files in the **Add Scala Support** dialog box, and then click **OK**.

Figure 1-57 Add Scala Support



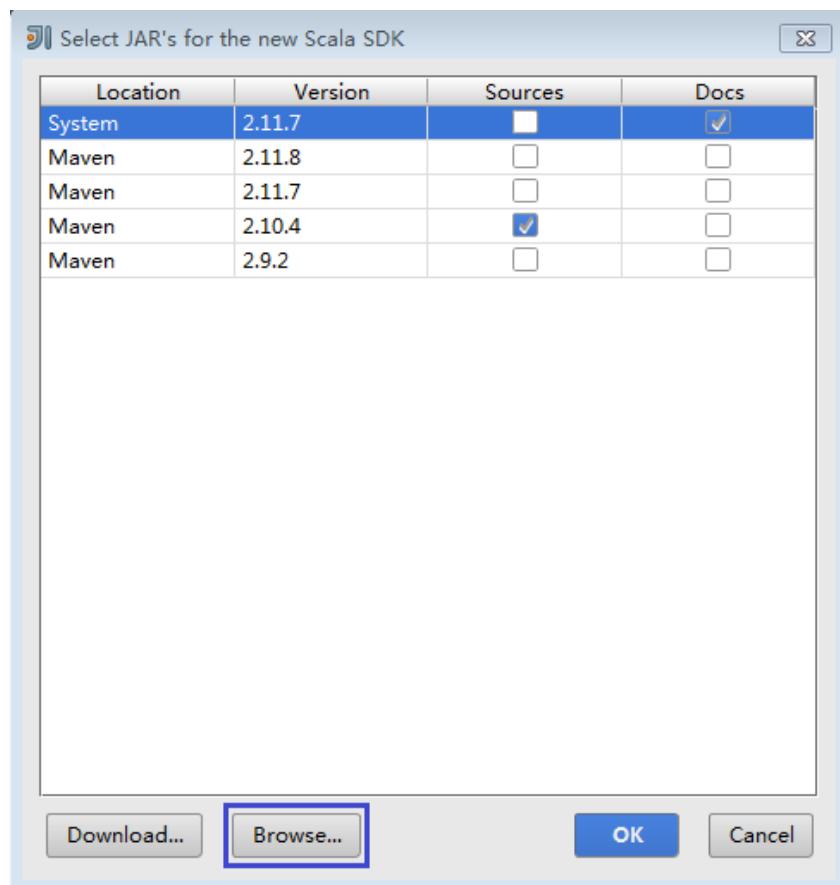
4. If IDEA fails to identify Scala SDK, create a Scala SDK.
  - a. Click **Create....**

Figure 1-58 Create...



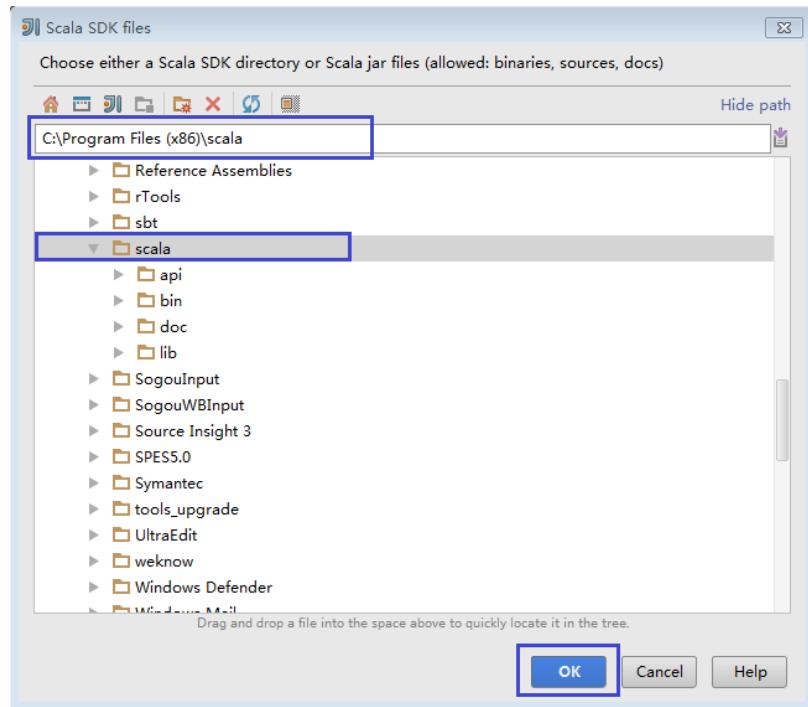
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse....**

Figure 1-59 Select JAR's for the new Scala SDK



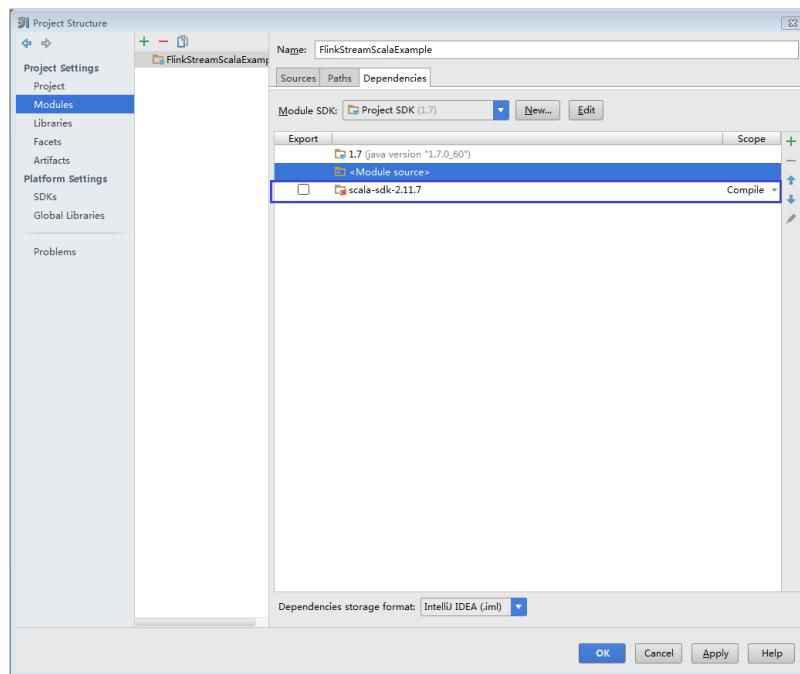
- c. On the **Scala SDK files** page, select the **scala sdk** directory and click **OK**.

Figure 1-60 Scala SDK files



5. Click OK.

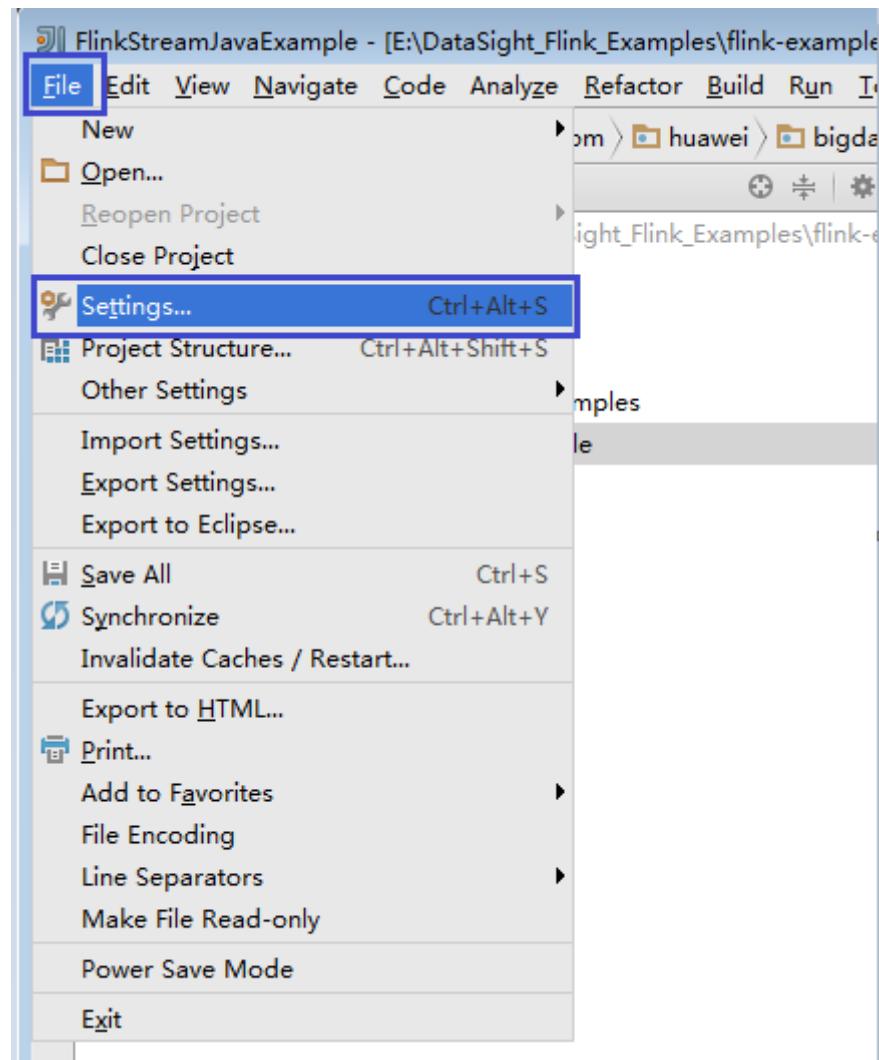
Figure 1-61 The configuration is successful.



**Step 7** Configure the text file encoding format of IDEA to prevent garbled characters.

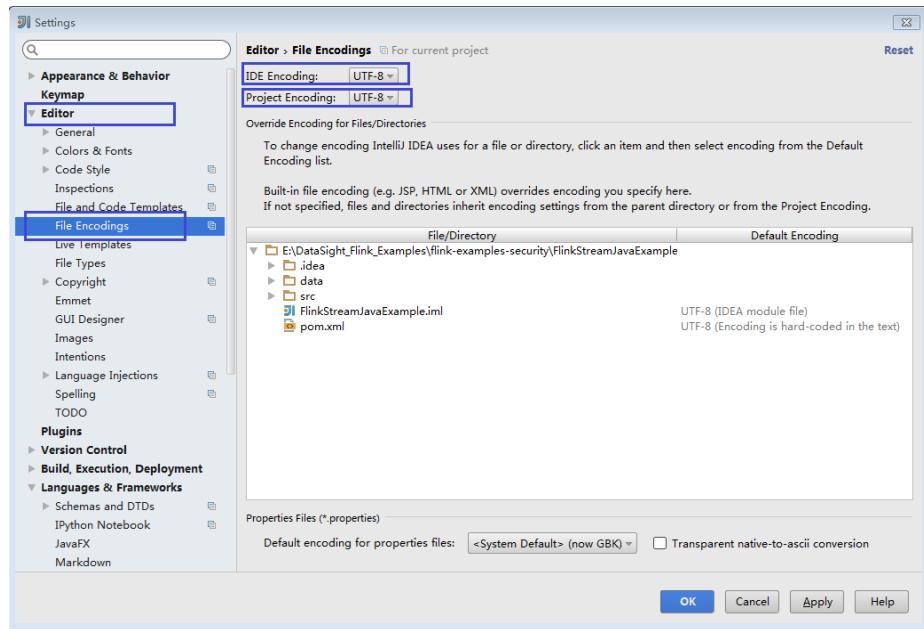
1. On the IDEA home page, choose **File > Settings....**

Figure 1-62 Choosing Settings



2. Configure encoding.
  - a. On the **Settings** page, unfold **Editor**, and choose **File Encodings**.
  - b. Select **UTF-8** from the **IDE Encoding** and **Project Encoding** drop-down lists on the right.
  - c. Click **Apply**.
  - d. Click **OK** to complete the encoding settings.

Figure 1-63 Settings



NOTE

- Obtain related dependency packages from the Flink server installation directory.
- Obtain Kafka dependency packages from the Kafka environment.
- Obtain Redis dependency packages from the Redis environment.
- For details about the dependency packages, see [Reference information about the dependency package for running the sample project](#).

----End

## Reference information about the dependency package for running the sample project

The **lib** and **opt** directories of the Flink client contain Flink JAR files. By default, the **lib** directory contains the Flink core JAR file, and the **opt** directory contains the JAR file (for example, **flink-connector-kafka\*.jar**) for connecting to external components. If it is required during application development, manually copy the related JAR files to the **lib** directory.

The dependency packages of the sample projects provided by Flink are as follows:

**Table 1-28** Dependency package for running the sample project

Sample project	Dependency package	Obtaining Dependency Packages
<ul style="list-style-type: none"><li>• Sample project of the DataStream application</li><li>• Sample project of the asynchronous checkpoint mechanism application</li></ul>	flink-dist_*.jar	Can be obtained from <b>lib</b> in the installation directory of the Flink client or server.
<ul style="list-style-type: none"><li>• Sample projects of Submitting SQL Jobs Using Flink Jar</li><li>• Sample projects of FlinkServer REST API</li></ul>	<ul style="list-style-type: none"><li>• flink-dist_*.jar</li><li>• flink-table_*.jar</li></ul>	Can be obtained from <b>lib</b> in the installation directory of the Flink client or server.
Sample projects of the application for producing and consuming data in Kafka	<ul style="list-style-type: none"><li>• kafka-clients-*.jar</li><li>• flink-connector-kafka_*.jar</li></ul>	<ul style="list-style-type: none"><li>• <b>kafka-clients-*.jar</b> is provided by Kafka and can be obtained from the <b>lib</b> directory in the installation directory of the Kafka client or server.</li><li>• <b>flink-connector-kafka_*.jar</b> can be obtained from <b>opt</b> in the installation directory of the Flink client or server.</li></ul>
Sample projects of pipeline	<ul style="list-style-type: none"><li>• flink-connector-netty_*.jar</li><li>• flink-dist_*.jar</li></ul>	<ul style="list-style-type: none"><li>• <b>flink-connector-netty_*.jar</b> can be obtained from the <b>lib</b> folder generated after the secondary development sample code is compiled.</li><li>• <b>flink-dist_*.jar</b> can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li></ul>

Sample project	Dependency package	Obtaining Dependency Packages
Sample projects of JOIN between configuration tables and streams	<ul style="list-style-type: none"><li>commons-pool2-*jar</li><li>flink-dist-*jar</li><li>jredisclient-*jar</li><li>super-csv-2.4.0.jar</li><li>wcc_krb5-*jar</li></ul>	<ul style="list-style-type: none"><li><b>commons-pool2-*jar, jredisclient-*jar, and wcc_krb5-*jar</b> are provided by Redis and can be obtained from the <b>lib</b> directory in the installation directory of Redis server.</li><li>flink-dist-*jar can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li><li>You can obtain <b>super-csv-2.4.0.jar</b> from the <b>lib</b> directory of the configuration table sample code or download it from <a href="https://github.com/super-csv/super-csv/releases/download/v2.4.0/super-csv-distribution-2.4.0-bin.zip">https://github.com/super-csv/super-csv/releases/download/v2.4.0/super-csv-distribution-2.4.0-bin.zip</a>.</li></ul>
Sample projects of Stream SQL JOIN	<ul style="list-style-type: none"><li>kafka-clients-*jar</li><li>flink-connector-kafka-*jar</li><li>flink-dist-*jar</li><li>flink-table-*jar</li></ul>	<ul style="list-style-type: none"><li><b>kafka-clients-*jar</b> is provided by Kafka and can be obtained from the <b>lib</b> directory in the installation directory of the Kafka client or server.</li><li>flink-connector-kafka-*jar can be obtained from <b>opt</b> in the installation directory of the Flink client or server.</li><li>flink-dist-*jar, flink-table-*jar can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li></ul>
Sample projects of HBase	<ul style="list-style-type: none"><li>flink-connector-hbase-*jar</li><li>flink-dist-*jar</li><li>flink-table-*jar</li><li>hbase-clients-*jar</li></ul>	<ul style="list-style-type: none"><li>flink-connector-hbase-*jar can be obtained from <b>opt</b> in the installation directory of the Flink client or server.</li><li>flink-dist-*jar, flink-table-*jar can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li><li><b>hbase-clients-*jar</b> is provided by HBase and can be obtained from the <b>lib</b> directory in the installation directory of the HBase client or server.</li></ul>
Sample projects of Hudi	hbase-unsafe-*jar	Can be obtained from the <b>lib</b> folder generated after the secondary development sample code is compiled.

### 1.8.2.3 Creating a Project (Optional)

#### Scenarios

IntelliJ IDEA can be used to create a Flink project. A Scala project is used as an example to illustrate how to create a Flink project.

## Procedure

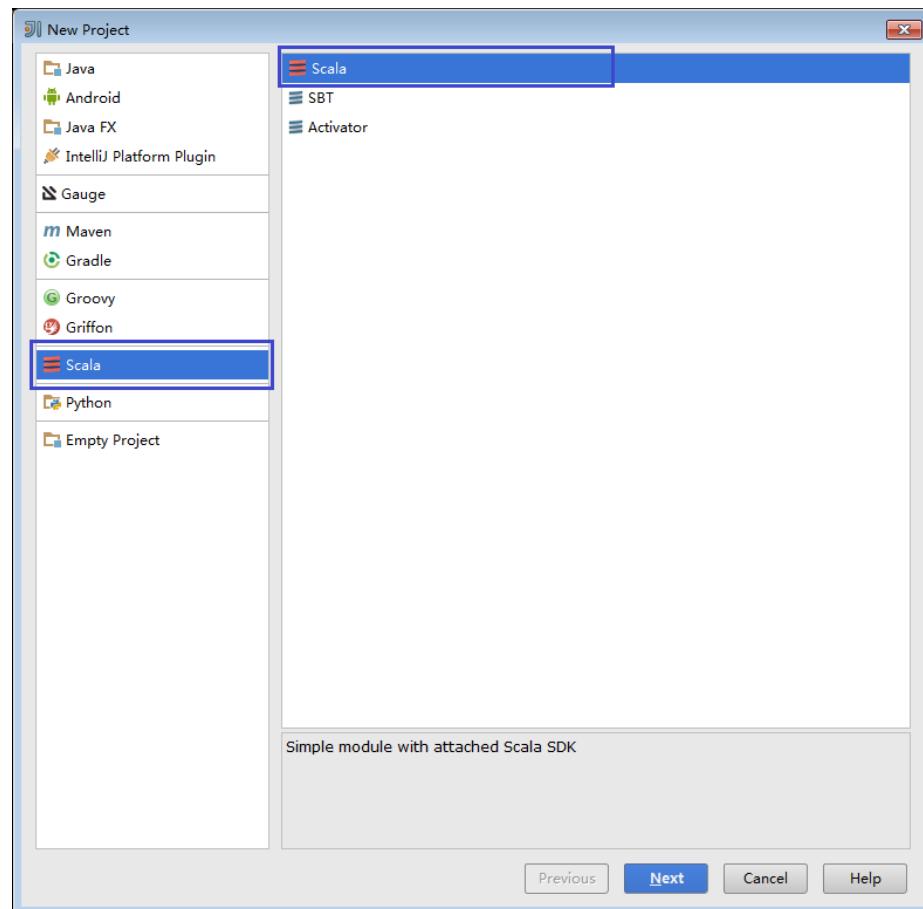
**Step 1** Start the IDEA and choose **Create New Project**.

**Figure 1-64** Creating a project



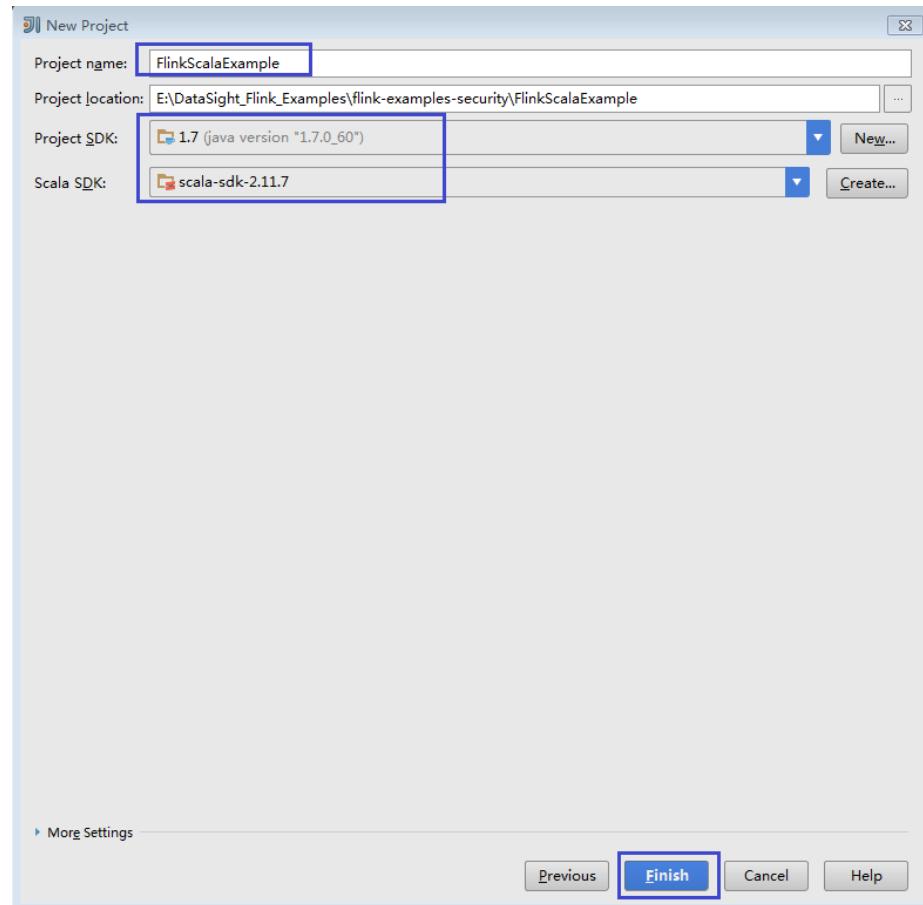
**Step 2** On the **New Project** page, choose a Scala development environment, choose **Scala Module**, and then click **Next**. If you want to create a Java project, choose corresponding parameters of Java.

Figure 1-65 Selecting a development environment



**Step 3** On the New Project page, enter the project name and project location, select the JDK version and Scala SDK, and then click **Finish**.

**Figure 1-66 Filling in project information**



----End

#### 1.8.2.4 Preparing for Security Authentication

##### Description

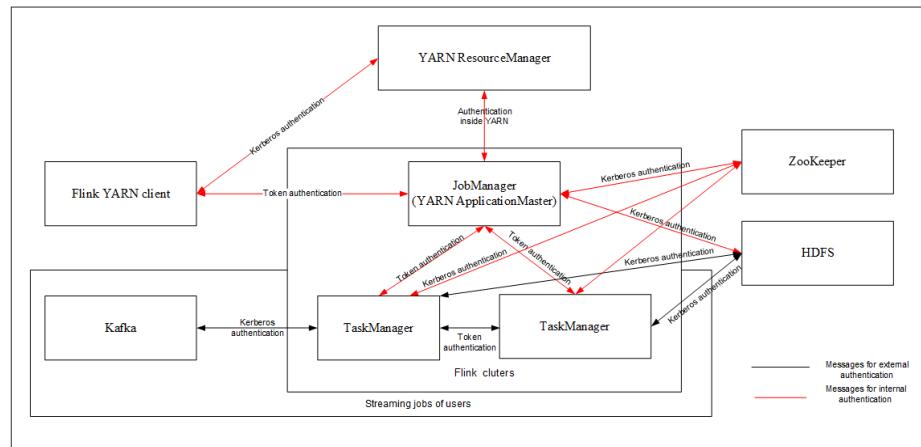
In a cluster with the security mode enabled, the components must be mutually authenticated before communicating with each other to ensure communication security.

To submit a Flink application, you need to ensure that Flink can communicate with Yarn and HDFS. Security authentication needs to be configured for the Flink application to be submitted.

Flink supports authentication and encrypted transmission. This section describes how to prepare for the authentication and encrypted transmission.

## Security Authentication

Figure 1-67 Authentication mode



Flink uses the following three authentication modes:

- Kerberos authentication is used between Flink Yarn client and Yarn ResourceManager, JobManager and ZooKeeper, JobManager and HDFS, TaskManager and HDFS, Kafka and TaskManager, and TaskManager and ZooKeeper.
- Security cookie authentication is used between Flink Yarn client and JobManager, JobManager and TaskManager, and TaskManager and TaskManager.
- Internal authentication of Yarn is used between Yarn ResourceManager and ApplicationMaster (AM).

### NOTE

- Flink JobManager and Yarn ApplicationMaster are in the same process.
- If security mode is enabled, you must use the Kerberos authentication and security cookie authentication.

**Table 1-29** Authentication methods

Authentication Mode	Configuration Method
Kerberos authentication (Currently, only keytab is supported.)	<ol style="list-style-type: none"><li>1. Download the user <b>keytab</b> file from FusionInsight Manager and save it to a directory on the node where the Flink client is deployed.</li><li>2. Configure the <b>flink-conf.yaml</b> as follows:<ol style="list-style-type: none"><li>a. Keytab path <code>security.kerberos.login.keytab: /home/flinkuser/keytab/flinkuser.keytab</code><p><b>NOTE</b> <code>/home/flinkuser/keytab/</code> indicates the directory for storing the <b>keytab</b> file.</p></li><li>b. Username for running the job <code>security.kerberos.login.principal: flinkuser</code></li><li>c. Kerberos authentication configuration required in HA mode when ZooKeeper is configured <code>zookeeper.sasl.disable: false</code> <code>security.kerberos.login.contexts: Client</code></li><li>d. Kerberos authentication (if needed) between the Kafka client and Kafka broker <code>security.kerberos.login.contexts: Client,KafkaClient</code></li></ol></li></ol>

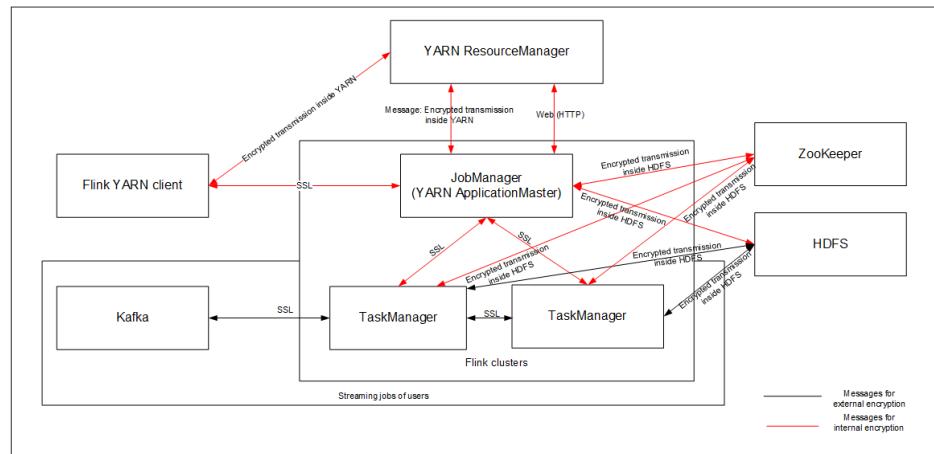
Authentication Mode	Configuration Method
Security cookie authentication	<ol style="list-style-type: none"><li>1. Obtain the SSL certificate and save it to the Flink client.</li><li>2. Set the following configuration items in the <b>flink-conf.yaml</b> file in the <b>conf</b> directory of the Flink client:<ul style="list-style-type: none"><li>• Set the <b>security.ssl.key-password</b>, <b>security.ssl.keystore-password</b>, and <b>security.ssl.truststore-password</b> to <i>&lt;password&gt;</i>.</li><li>• Set <b>security.ssl.keystore</b> to the relative path of the keystore file, for example, <b>ssl/flink.keystore</b>.</li><li>• Set <b>security.ssl.truststore</b> to the relative path of the truststore file, for example, <b>ssl/flink.truststore</b>.</li><li>• Set <b>security.cookie</b> to a random password string.</li></ul></li></ol> <p><b>NOTE</b></p> <ol style="list-style-type: none"><li>1. The generated <b>flink.keystore</b>, <b>flink.truststore</b>, and <b>security.cookie</b> items are automatically filled in the corresponding configuration items in <b>flink-conf.yaml</b>.</li><li>2. The values of <b>security.ssl.key-password</b>, <b>security.ssl.keystore-password</b>, and <b>security.ssl.truststore-password</b> must be obtained using the plaintext encryption API of Manager: <code>curl -k -i -u user name:password -X POST -HContent-type:application/json -d '{"plainText":"password"}' 'https://x.x.x.x:28443/web/api/v2/tools/encrypt'</code>. <i>user name:password</i> indicates the user name and password for logging in to the system. Password of "plainText" is the password for invoking the <b>generate_keystore.sh</b> script. <i>x.x.x.x</i> is the floating IP address of FusionInsight Manager in the cluster.</li><li>3. Set <b>security.enable</b> to true and check whether <b>security.cookie</b> is configured successfully. <code>security.cookie: ae70acc9-9795-4c48-ad35-8b5adc8071744f605d1d-2726-432e-88ae-dd39bfec40a9</code></li></ol>
Internal authentication of Yarn	You do not need to do any configuration for this authentication mode.

 **NOTE**

One Flink cluster supports only one user. One user can create multiple Flink clusters.

## Encrypted Transmission

Figure 1-68 Encrypted transmission of Flink



Flink uses following encrypted transmission modes:

- Encrypted transmission inside Yarn: It is used between the Flink Yarn client and Yarn ResourceManager, as well as Yarn ResourceManager and JobManager.
- SSL transmission is used between Flink Yarn client and JobManager, JobManager and TaskManager, and TaskManager and TaskManager.
- Internal encrypted transmission of Hadoop is used between JobManager and HDFS, TaskManager and HDFS, JobManager and ZooKeeper, and TaskManager and ZooKeeper.

### NOTE

You do not need to do any configurations for internal encryption of Yarn and Hadoop. Only SSL configuration is required.

To configure SSL encrypted transmission, perform the following steps to configure the **flink-conf.yaml** file on the client:

1. Turn on the SSL switch and set SSL encryption algorithms. **Table 1-30** describes the parameters. Set the parameters based on your need.

Table 1-30 Parameter description

Parameter	Example Value	Description
security.ssl.enabled	true	Enabling the SSL function
akka.ssl.enabled	true	Enabling Akka SSL
blob.service.ssl.enabled	true	Enabling SSL for the BLOB channel

Parameter	Example Value	Description
taskmanager.data.ssl.enabled	true	Enabling SSL for communications between TaskManagers
security.ssl.algorithms	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Setting SSL encryption algorithms

 NOTE

Enabling SSL for data transmission between TaskManagers may lead to a drop of system performance.

2. In the `bin` directory of the Flink client, run `sh generate_keystore.sh <password>`. The configuration items in [Table 1-31](#) are set by default, you can set these parameters as needed.

**Table 1-31 Description**

Parameter	Example Value	Description
security.ssl.keystore	<code> \${path}/flink.keystore</code>	Path for storing the <b>keystore</b> . <b>flink.keystore</b> indicates the name of the <b>keystore</b> file generated by the <b>generate_keystore.sh*</b> tool.
security.ssl.keystore-password	-	A user-defined password of <b>keystore</b> .
security.ssl.key-password	-	A user-defined password of the SSL key.
security.ssl.truststore	<code> \${path}/flink.truststore</code>	Path for storing the <b>truststore</b> . <b>flink.truststore</b> indicates the name of the <b>truststore</b> file generated by the <b>generate_keystore.sh*</b> tool.

Parameter	Example Value	Description
security.ssl.truststore-password	-	A user-defined password of <b>truststore</b> .

 NOTE

The **path** directory is a user-defined directory for storing configuration files of the SSL keystore and truststore. The commands vary according to the relative path and absolute path. For details, see [3](#) and [4](#).

3. If the **keystore** or **truststore** file path is a relative path, the Flink client directory where the command is executed needs to access this relative path directly. Either of the following method can be used to transmit the keystore and truststore file:

- Add **-t** option to the CLI **yarn-session.sh** command of Flink to transmit the keystore and truststore files to each execution node. The following is an example:  

```
cd /opt/hadoopclient/Flink/flink  
.bin/yarn-session.sh -t ssl/
```
- Add **-yt** to the **flink run** command to transfer the **keystore** and **truststore** file to execution nodes. The following is an example:  

```
.bin/flink run -yt ssl/ -ys 3 -m yarn-cluster -c  
com.huawei.SocketWindowWordCount ..lib/flink-eg-1.0.jar --  
hostname r3-d3 --port 9000
```

 NOTE

- In the example, **ssl/** is a user-defined subdirectory in the Flink client directory, which is used to store configuration files of the SSL keystore and truststore.
- The relative path of **ssl/** must be accessible from the current path where the Flink client command is run.

4. If the **keystore** or **truststore** file path is an absolute path, the **keystore** or **truststore** file must exist in the absolute path on Flink Client and all Yarn nodes.

Either of the following methods can be used to execute applications. The **-t** or **-yt** option does not need to be added to transmit the **keystore** and **truststore** files.

- Run the CLI **yarn-session.sh** command of Flink to execute applications. The following is an example:  

```
.bin/yarn-session.sh
```
- Run the **flink run** command to execute applications. The following is an example:  

```
.bin/flink run -ys 3 -m yarn-cluster -c  
com.huawei.SocketWindowWordCount ..lib/flink-eg-1.0.jar --  
hostname r3-d3 --port 9000
```

### 1.8.2.5 Configuring a Spring Boot Sample Project

#### Scenarios

Run the sample code of the Spring Boot interface in FusionInsight MRS Hive. Currently, GaussDB(DWS) and Elasticsearch SpringBoot sample projects are supported.

In the following example, an application is developed in Linux to connect GaussDB(DWS) to Flink using Spring Boot.

##### NOTE

Before running the GaussDB(DWS) sample, log in to the node where GaussDB(DWS) is deployed to create an empty table **test\_lzh1** for receiving data. The creation command is as follows:

```
create table test_lzh1 (id integer not null);
```

#### Procedure

##### Step 1 Install a client that contains only the Flink service.

The following example shows how to install the Flink client on a node in the cluster:

1. Log in to FusionInsight Manager. Choose **Cluster > Services > Flink > More > Download Client**.
2. In the dialog box that is displayed, select **Complete Client** for **Select Client Type**, select the platform type that matches the architecture of the node where the client is to install, Select **Server** from the **Select Download Location**, and click **OK**.

The generated file is stored in the **/tmp/FusionInsight-Client** directory on the active management node by default.

The name of the client software package is in the following format: **FusionInsight\_Cluster\_<Cluster ID>\_Flink\_Client.tar**. The following content uses the cluster ID **1** as an example.

3. Log in to the server where the client is to be installed as the client installation user.
4. Go to the directory where the installation package is stored and run the following commands to decompress the package:  
**cd /tmp/FusionInsight-Client**  
**tar -xvf FusionInsight\_Cluster\_1\_Flink\_Client.tar**
5. Run the following command to verify the decompressed file and check whether the command output is consistent with the information in the **sha256** file:  
**sha256sum -c FusionInsight\_Cluster\_1\_Flink\_ClientConfig.tar.sha256**
6. Decompress the obtained installation file.  
**tar -xvf FusionInsight\_Cluster\_1\_Flink\_ClientConfig.tar**
7. Go to the directory where the installation package is stored, and run the following command to install the client to a specified directory (absolute path), for example, **/opt/hadoopclient**:

```
cd /tmp/FusionInsight-Client/FusionInsight_Cluster_1_Flink_ClientConfig  
.install.sh /opt/hadoopclient
```

8. Configure security authentication and encryption. For details, see "Using Flink from Scratch" in the *Component Operation Guide*.

**Step 2** Obtain the sample project folder **flink-dws-sink-example** from **src\springboot\flink-examples** in the directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

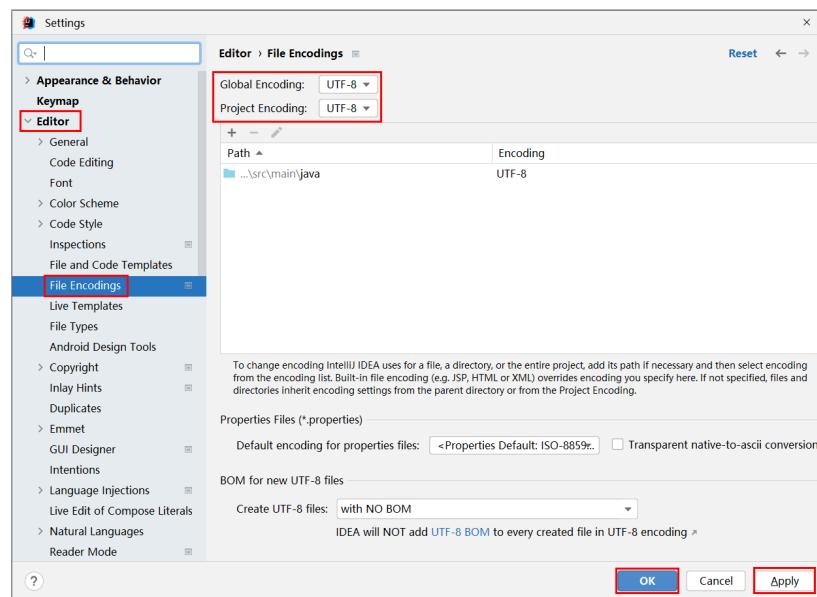
**Step 3** Import the sample project to the IntelliJ IDEA development environment.

1. On the menu bar of IntelliJ IDEA, choose **File > Open....**
2. In the **Open File or Project** dialog box that is displayed, select the **flink-dws-sink-example** folder and click **OK**.

**Step 4** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

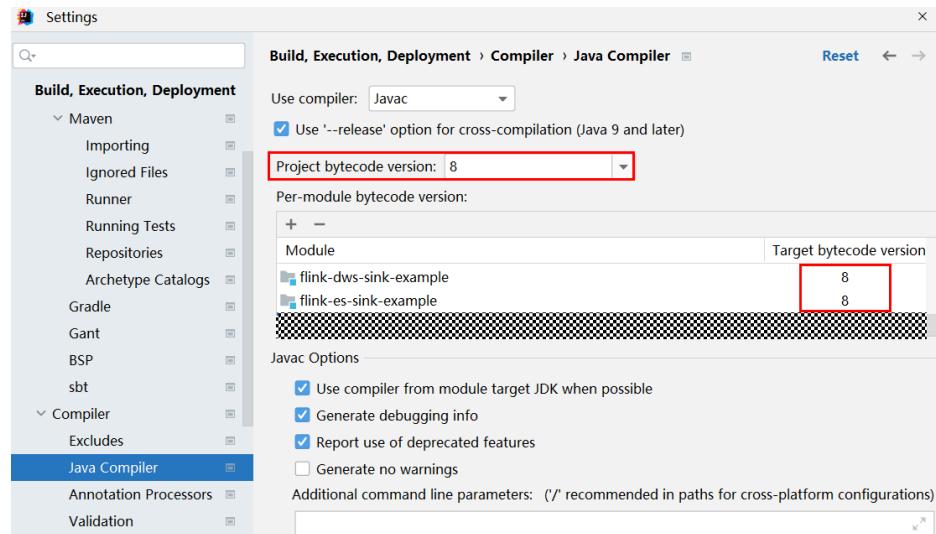
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. In the left navigation pane, choose **Editor > File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**.

**Figure 1-69** Setting the IntelliJ IDEA coding format

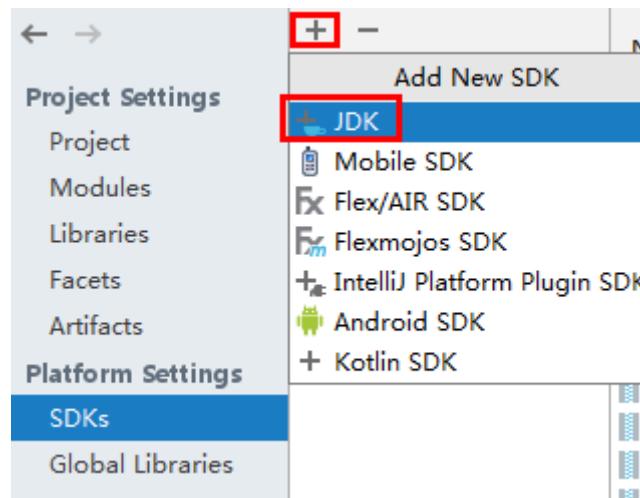


**Step 5** Set the JDK of the project.

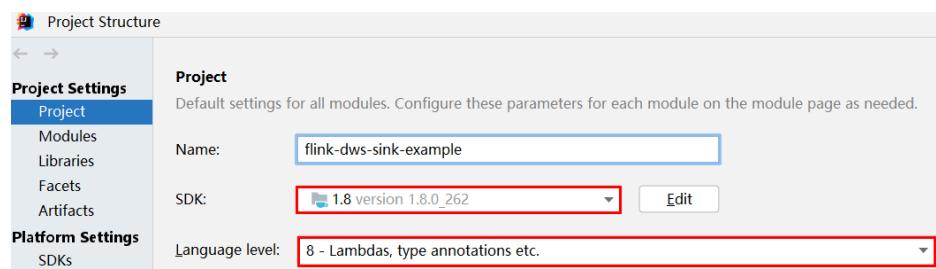
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **flink-dws-sink-example** to **8**.



3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.



6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Language level** drop-down list box.

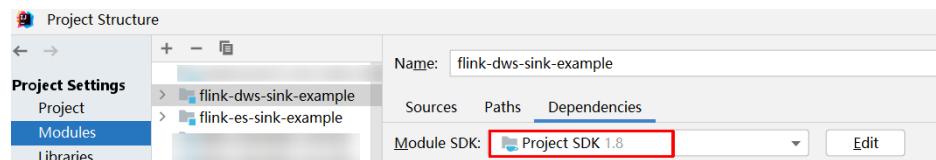


9. Click **Apply**.

- Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc..**



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.

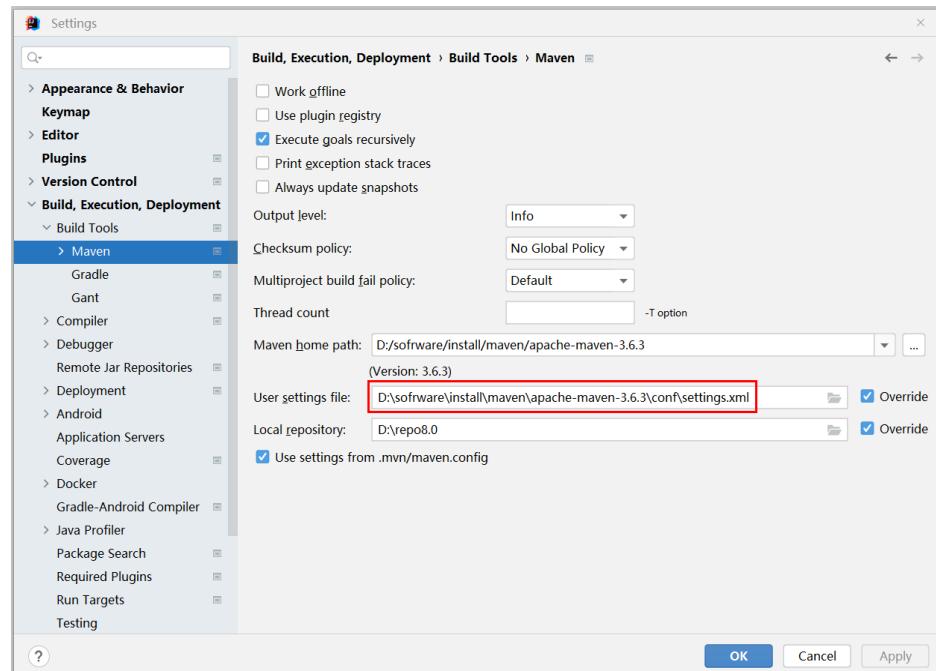


- Click **Apply** and then **OK**.

#### Step 6 Configure Maven.

- Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
- On the IntelliJ IDEA page, choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

**Figure 1-70** Directory for storing the **settings.xml** file



- Click the drop-down list next to **Maven home directory** and select the Maven installation directory.

- Click **Apply** and then **OK**.

**Step 7** Find the **application.properties** file in **src\main\resources** and add the following content to the file:

- Run the GaussDB(DWS) sample  
spring.datasource.dws.url=jdbc:postgresql://IP address of the GaussDB(DWS) node:8000/postgres  
spring.datasource.dws.username=dbadmin  
spring.datasource.dws.password=Password of dbadmin  
spring.datasource.dws.driver=org.postgresql.Driver
- Run the Elasticsearch sample  
spring.datasource.es.host=Service IP address of Elasticsearch EsNode  
spring.datasource.es.port=Elasticsearch EsNode Port  
spring.datasource.es.scheme=http

 **NOTE**

- log in to FusionInsight Manager and choose **Cluster > Services > Elasticsearch**. On the page that is displayed, click **Instance** and view the service IP addresses of all EsNode instances in the Elasticsearch cluster.
- Log in to FusionInsight Manager, choose **Cluster > Services > Elasticsearch > Configuration > All Configurations**, search for **SERVER\_PORT**, and view the EsNode port.

----End

## 1.8.3 Developing an Application

### 1.8.3.1 DataStream Application

#### 1.8.3.1.1 Scenarios

##### Scenarios

Develop a DataStream application of Flink to perform the following operations on logs about dwell durations of netizens for shopping online.

 **NOTE**

The DataStream application can run in both the Windows environment and the Linux environment.

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours in total in a real time manner.
- The first column in the log file records names, the second column records genders, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
```

```
CaiXuyu,female,50  
FangBo,female,60
```

log2.txt: logs collected on Sunday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20  
YuanJing,male,10  
CaiXuyu,female,50  
FangBo,female,50  
GuoYijun,male,5  
CaiXuyu,female,50  
Liyuan,male,20  
CaiXuyu,female,50  
FangBo,female,50  
LiuYang,female,20  
YuanJing,male,10  
FangBo,female,50  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

## Data Planning

Data of DataStream sample project is stored in a **.txt** file.

Place the **log1.txt** and **log2.txt** in two directories, for example, **/opt/log1.txt** and **/opt/log2.txt**.

### NOTE

- If the data file is stored in the local file system, the data file must be stored in the specified directory on all nodes where Yarn NodeManager is deployed, and the running user access permission must be set.
- Alternatively, store the data file on HDFS and set the file read path in the program to the HDFS path, for example, **hdfs://hacluster/path/to/file**.

## Development Approach

Collect the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

1. Read text data, generate DataStreams, and parse data to generate UserRecord information.
2. Filter the data about the time that female netizens spend online.
3. Perform keyby operation based on the name and gender, and collect the time that female netizens spend online within a time window.
4. Filter data about users whose consecutive online duration exceeds the threshold, and obtain the result.

### 1.8.3.1.2 Java Sample Code

## Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

## DataStream FlinkStreamJavaExample Sample Code

The following code segment is an example. For details, see com.huawei.bigdata.flink.examples.FlinkStreamJavaExample.

```
//Parameter description:  
//<filePath> indicates paths where text is read. Paths are separated by commas (,).  
//<windowTime> indicates the period covered by statistics window. The unit is minute.  
public class FlinkStreamJavaExample {  
    public static void main(String[] args) throws Exception {  
        //Print reference command for executing flink run.  
        System.out.println("use command as: ");  
        System.out.println("./bin/flink run --class  
com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/test.jar --filePath /opt/log1.txt,/opt/  
log2.txt --windowTime 2");  
        System.out.println("*****");  
        System.out.println("<filePath> is for text file to read data, use comma to separate");  
        System.out.println("<windowTime> is the width of the window, time as minutes");  
        System.out.println("*****");  
  
        //Paths where text is read. Paths are separated by commas (,)  
        final String[] filePaths = ParameterTool.fromArgs(args).get("filePath", "/opt/log1.txt,/opt/  
log2.txt").split(",");  
        assert filePaths.length > 0;  
  
        //windowTime specifies the size of the time window. By default, a window with 2 minutes as the size  
can read all data in a text file.  
        final int windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2);  
  
        //Build the execution environment and run eventTime to process window data.  
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);  
        env.setParallelism(1);  
  
        //Read DataStream of the text.  
        DataStream<String> unionStream = env.readTextFile(filePaths[0]);  
        if (filePaths.length > 1) {  
            for (int i = 1; i < filePaths.length; i++) {  
                unionStream = unionStream.union(env.readTextFile(filePaths[i]));  
            }  
        }  
  
        //Convert data, build the logic for the entire data process, calculate, and print results.  
        unionStream.map(new MapFunction<String, UserRecord>() {  
            @Override  
            public UserRecord map(String value) throws Exception {  
                return getRecord(value);  
            }  
        }).assignTimestampsAndWatermarks(  
            new Record2TimestampExtractor()  
        ).filter(new FilterFunction<UserRecord>() {  
            @Override  
            public boolean filter(UserRecord value) throws Exception {  
                return value.sex.equals("female");  
            }  
        }).keyBy(  
            new UserRecordSelector()  
        ).window(  
            TumblingEventTimeWindows.of(Time.minutes(windowTime))  
        ).reduce(new ReduceFunction<UserRecord>() {  
            @Override  
            public UserRecord reduce(UserRecord value1, UserRecord value2)  
                throws Exception {  
                value1.shoppingTime += value2.shoppingTime;  
                return value1;  
            }  
        }).filter(new FilterFunction<UserRecord>() {  
            @Override  
            public boolean filter(UserRecord value) throws Exception {  
                return value.shoppingTime > 0;  
            }  
        }).print();  
    }  
}
```

```
        return value.shoppingTime > 120;
    }
}).print();

//Call execute to trigger the execution.
env.execute("FemaleInfoCollectionPrint java");
}

//Build the keyword of keyBy as the grouping basis.
private static class UserRecordSelector implements KeySelector<UserRecord, Tuple2<String, String>> {
    @Override
    public Tuple2<String, String> getKey(UserRecord value) throws Exception {
        return Tuple2.of(value.name, value.sex);
    }
}

//Parse the row data of the text and build UserRecord data structure.
private static UserRecord getRecord(String line) {
    String[] elems = line.split(",");
    assert elems.length == 3;
    return new UserRecord(elems[0], elems[1], Integer.parseInt(elems[2]));
}

//Defines UserRecord data structure and rewrite toString printing method.
public static class UserRecord {
    private String name;
    private String sexy;
    private int shoppingTime;

    public UserRecord(String n, String s, int t) {
        name = n;
        sexy = s;
        shoppingTime = t;
    }

    public String toString() {
        return "name: " + name + " sexy: " + sexy + " shoppingTime: " + shoppingTime;
    }
}

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and watermark.
private static class Record2TimestampExtractor implements
AssignerWithPunctuatedWatermarks<UserRecord> {

    //add tag in the data of datastream elements
    @Override
    public long extractTimestamp(UserRecord element, long previousTimestamp) {
        return System.currentTimeMillis();
    }

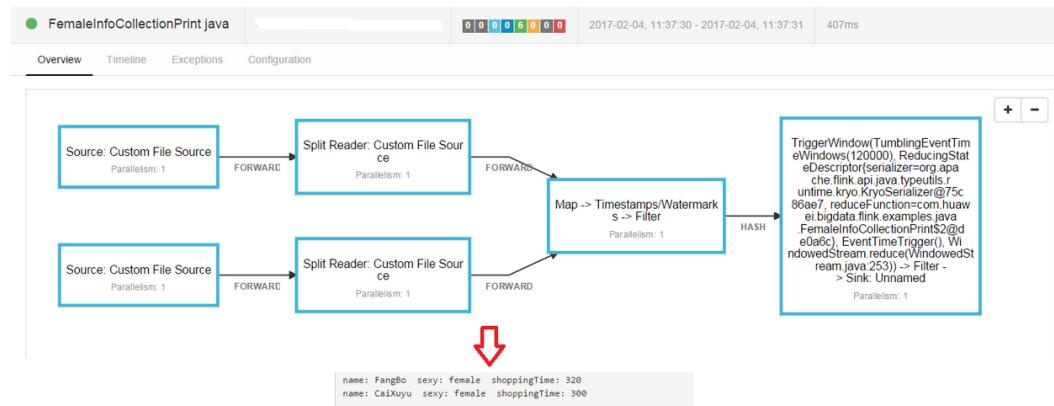
    //give the watermark to trigger the window to execute, and use the value to check if the window
    //elements are ready
    @Override
    public Watermark checkAndGetNextWatermark(UserRecord element, long extractedTimestamp) {
        return new Watermark(extractedTimestamp - 1);
    }
}
```

The following is the command output:

```
name: FangBo sexy: female shoppingTime: 320
name: CaiXuyu sexy: female shoppingTime: 300
```

[Figure 1-71](#) shows the process of execution.

**Figure 1-71 Process of execution**



### 1.8.3.1.3 Scala Sample Code

#### Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

#### DataStream FlinkStreamScalaExample Sample Code

The following code is an example. For details, see com.huawei.bigdata.flink.examples.FlinkStreamScalaExample.

```
//Parameter description
//filePath indicates paths where text is read. Paths are separated by commas (,).
//windowTime indicates the period covered by statistics window. The unit is minute.
object FlinkStreamScalaExample {
def main(args: Array[String]) {
    //Print the reference command used to execute flink run.
    System.out.println("use command as:")
    System.out.println("./bin/flink run --class"
com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/test.jar --filePath /opt/log1.txt,/opt/
log2.txt --windowTime 2")
    System.out.println("*****")
    System.out.println("<filePath> is for text file to read data, use comma to separate")
    System.out.println("<windowTime> is the width of the window, time as minutes")
    System.out.println("*****")

    //Paths where text is read. Paths are separated by commas (,)
    val filePaths = ParameterTool.fromArgs(args).get("filePath",
        "/opt/log1.txt,/opt/log2.txt").split(",").map(_.trim)
    assert(filePaths.length > 0)

    //windowTime specifies the size of the time window. By default, a window with 2 minutes as the size can
    //read all data in a text file.
    val windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2)

    //Build the execution environment and run eventTime to process window data.
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
    env.setParallelism(1)
    //Read DataStream of the text.
    val unionStream = if (filePaths.length > 1) {
        val firstStream = env.readTextFile(filePaths.apply(0))
        firstStream.union(filePaths.drop(1).map(it => env.readTextFile(it)): _*)
    } else {
        env.readTextFile(filePaths.apply(0))
    }
}
```

```
}

//Convert data, build the logic for the entire data process, calculate, and print results.
unionStream.map(getRecord(_))
    .assignTimestampsAndWatermarks(new Record2TimestampExtractor)
    .filter(_._sexy == "female")
    .keyBy("name", "sexy")
    .window(TumblingEventTimeWindows.of(Time.minutes(windowTime)))
    .reduce((e1, e2) => UserRecord(e1.name, e1.sex, e1.shoppingTime + e2.shoppingTime))
    .filter(_._shoppingTime > 120).print()

//Call execute to trigger the execution
env.execute("FemaleInfoCollectionPrint scala")
}

//Parse the row data of the text and build UserRecord data structure.
def getRecord(line: String): UserRecord = {
    val elems = line.split(",")
    assert(elems.length == 3)
    val name = elems(0)
    val sexy = elems(1)
    val time = elems(2).toInt
    UserRecord(name, sexy, time)
}

//Defines UserRecord data structure.
case class UserRecord(name: String, sexy: String, shoppingTime: Int)

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and waterMark.
private class Record2TimestampExtractor extends AssignerWithPunctuatedWatermarks[UserRecord] {

    //add tag in the data of datastream elements
    override def extractTimestamp(element: UserRecord, previousTimestamp: Long): Long = {
        System.currentTimeMillis()
    }

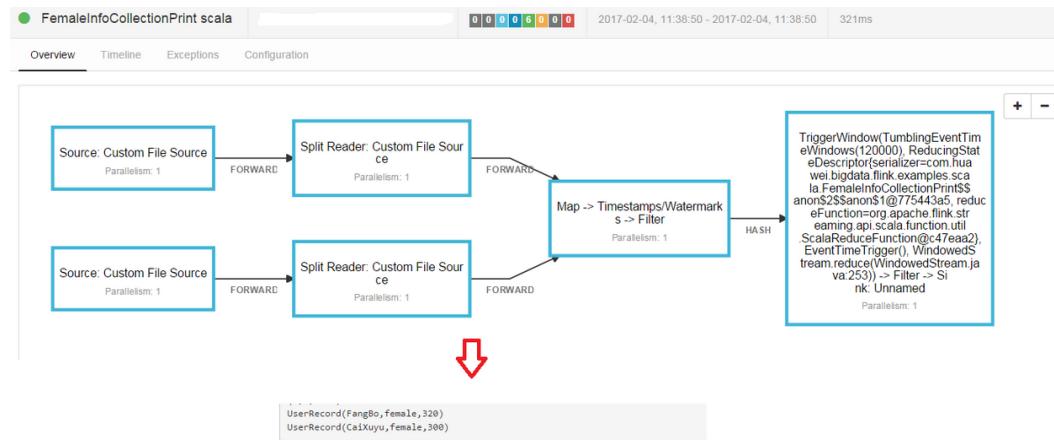
    //give the watermark to trigger the window to execute, and use the value to check if the window
    //elements are ready
    def checkAndGetNextWatermark(lastElement: UserRecord,
                                 extractedTimestamp: Long): Watermark = {
        new Watermark(extractedTimestamp - 1)
    }
}
```

The following is the command output:

```
UserRecord(FangBo,female,320)
UserRecord(CaiXuyu,female,300)
```

**Figure 1-72** shows the process of execution.

**Figure 1-72 Process of execution**



### 1.8.3.2 Interconnecting with Kafka

#### 1.8.3.2.1 Scenario Description

##### Scenario

Assume that a Flink service receives one message record every second.

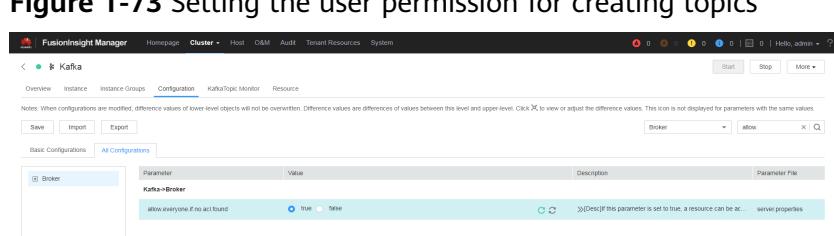
A Flink application is developed to output prefixed message content in real time based on service requirements.

##### Data Preparation

Flink's sample project data is stored in Kafka. A user with Kafka permission can send data to Kafka and receive data from it.

1. Ensure that clusters have been installed, including HDFS, Yarn, Flink, and Kafka.
2. Create a topic.
  - a. Configure the user permission for creating topics on the server. Change the value of the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**. See [Figure 1-73](#). After the configuration is complete, restart the Kafka service.

**Figure 1-73 Setting the user permission for creating topics**



- b. Run a Linux command line to create a topic. Before running a command, run the **kinit** command, for example, **kinit flinkuser**, to authenticate the human-machine account.

 NOTE

To create a Flink user, you need to have the permission to create Kafka topics. For details, see [Preparing a Developer Account](#).

The command for creating a Kafka topic is as follows:

```
bin/kafka-topics.sh --create --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --partitions {partitionNum} --replication-factor {replicationNum} --topic {Topic}
```

**Table 1-32** Parameters

Parameter	Description
{Kafka cluster IP address}	Service IP address of the Broker node in the Kafka cluster.
{partitionNum}	Number of topic partitions
{replicationNum}	Number of data replicas of each partition in a topic
{Topic}	Topic name

For example, run the following command on the Kafka client, and the topic name is **topic1**.

```
bin/kafka-topics.sh --create --bootstrap-server  
10.96.101.32:21007,10.96.101.251:21007,10.96.101.177:21007 --partitions 5 --replication-factor 1  
--topic topic1
```

3. Perform security authentication.

You can use Kerberos authentication, SSL encryption authentication, or Kerberos + SSL authentication.

- **Kerberos authentication**

i. Configure the client.

In Flink configuration file **flink-conf.yaml**, add configurations about Kerberos authentication. For example, add **KafkaClient** in **contexts** as follows:

```
security.kerberos.login.keytab: /home/demo/flink/release/flink-x.x.x/keytab/user.keytab  
security.kerberos.login.principal: flinkuser  
security.kerberos.login.contexts: Client,KafkaClient  
security.kerberos.login.use-ticket-cache: false
```

ii. Run parameters.

Running parameters about the **SASL\_PLAINTEXT** protocol are as follows:

```
--topic topic1 --bootstrap.servers 10.96.101.32:21007 --security.protocol SASL_PLAINTEXT  
--sasl.kerberos.service.name kafka --kerberos.domain.name hadoop.System domain name.com
```

 NOTE

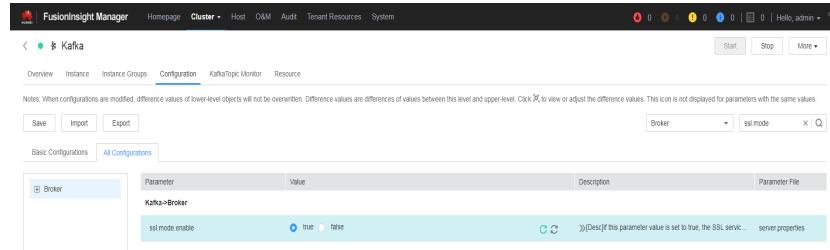
- *10.96.101.32:21007*: IP address and port number of the Kafka server.
- *System domain name*: You can log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**.

### - SSL encryption

- Configure the server.

Set `ssl.mode.enable` to `true`. See [Figure 1-74](#).

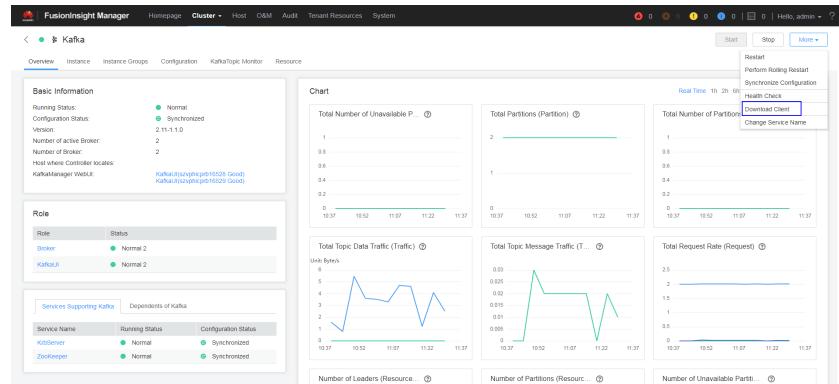
**Figure 1-74** Configuring the server



- Configure the client.

- Log in to FusionInsight Manager and choose **Cluster > Name of the desired cluster > Services > Kafka**. On the page that is displayed, choose **More > Download Client** to download the Kafka client. See [Figure 1-75](#).

**Figure 1-75** Configuring the client



- Use the `ca.crt` certificate file in the client root directory to generate the `truststore` file for the client.

Run the following command:

```
keytool -noprompt -import -alias myservercert -file ca.crt -keystore truststore.jks
```

The command output is as follows:

```
drwx----- 5 zgd users 4096 Feb 4 16:22 .
drwxr-xr-x 10 zgd users 4096 Jan 22 17:38 ..
-rwx----- 1 zgd users 135 Jan 22 17:31 application.properties
-rwx----- 1 zgd users 790 Jan 22 17:31 bigdata_env.sample
-rw----- 1 zgd users 1322 Jan 22 17:31 ca.crt
-rwx----- 1 zgd users 4508 Jan 22 17:31 conf.py
-rw----- 1 zgd users 120 Jan 22 17:31 hosts
-rwx----- 1 zgd users 745 Jan 22 17:31 install.bat
-rwx----- 1 zgd users 15082 Jan 22 17:31 install.sh
drwxr-xr-x 2 zgd users 4096 Jan 22 17:38 JDK
-rwx----- 1 zgd users 37021723 Jan 22 17:31 jython-standalone-2.7.0.jar
drwxr-xr-x 5 zgd users 4096 Jan 22 17:38 Kafka
drwxr-xr-x 3 zgd users 4096 Jan 22 17:38 KrbClient
-rwx----- 1 zgd users 473 Jan 22 17:31 log4j.properties
-rwx----- 1 zgd users 2107 Jan 22 17:31 README
-rwx----- 1 zgd users 6949 Jan 22 17:31 refreshConfig.sh
-rwx----- 1 zgd users 1736 Jan 22 17:31 switchuser.py
-rw-r--r-- 1 root root 1004 Feb 4 16:22 truststore.jks
```

3) Run parameters.

The value of **ssl.truststore.password** must be the same as the password you entered when creating **truststore**. Run the following command to run parameters:

```
--topic topic1 --bootstrap.servers 10.96.101.32:21008 --security.protocol SSL --  
ssl.truststore.location /home/zgd/software/FusionInsight_XXX_Kafka_ClientConfig/  
truststore.jks --ssl.truststore.password xxx //10.96.101.32:21008 indicates the IP:Port  
of the Kafka server. XXX indicates the FusionInsight version, xxx indicates the  
password.
```

- **Kerberos+SSL encryption**

After completing preceding configurations of the client and server of Kerberos and SSL, modify the port number and protocol type in running parameters to enable the Kerberos+SSL encryption mode.

```
--topic topic1 --bootstrap.servers 10.96.101.32:21009 --security.protocol SASL_SSL --  
sasl.kerberos.service.name kafka --ssl.truststore.location --kerberos.domain.name  
hadoop.System domain name.com /home/zgd/software/FusionInsight_XXX_Kafka_ClientConfig/  
truststore.jks --ssl.truststore.password xxx //10.96.101.32:21009 indicates the IP:Port of the  
Kafka server. XXX indicates the FusionInsight version, xxx indicates the password.
```

## Development Guideline

1. Start Flink Kafka Producer to send data to Kafka.
2. Start Flink Kafka Consumer to receive data from Kafka. Ensure that Flink Kafka Consumer and Flink Kafka Producer have the same topics.
3. Add prefixes to data content and print the results.

### 1.8.3.2.2 Java Sample Code

#### Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

#### Sample Code

If you want to use FusionInsight in security mode, ensure that the **kafka-clients-\*jar** is obtained from the FusionInsight client directory.

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see `com.huawei.bigdata.flink.examples.WriteIntoKafka` and `com.huawei.bigdata.flink.examples.ReadFromKafka`.

```
//producer code  
public class WriteIntoKafka {  
  
    public static void main(String[] args) throws Exception {  
        //Print the reference command of flink run.  
  
        System.out.println("use command as: ");  
  
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +  
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");  
  
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +  
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
```

```
SASL_PLAINTEXT --sasl.kerberos.service.name kafka");

        System.out.println
("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println
("*****");
//Build the execution environment.
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
//Configure the parallelism.
env.setParallelism(1);
//Parse the execution parameter.
ParameterTool paraTool = ParameterTool.fromArgs(args);
//Build the StreamGraph and write data generated by customized source into Kafka.
DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());

messageStream.addSink(new FlinkKafkaProducer<>(paraTool.get("topic"),
        new SimpleStringSchema(),
        paraTool.getProperties()));
//Call execute to trigger the execution.
env.execute();
}

//Customize source to continuously generate messages every one second.
public static class SimpleStringGenerator implements SourceFunction<String> {

    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
    long i = 0;

    @Override
    public void run(SourceContext<String> ctx) throws Exception {
        while (running) {
            ctx.collect("element-" + (i++));
            Thread.sleep(1000);
        }
    }

    @Override
    public void cancel() {
        running = false;
    }
}
```

```
}

//consumer code
public class ReadFromKafka {

    public static void main(String[] args) throws Exception {
        //Print the reference command of flink run.
        System.out.println("use command as:");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka");

        System.out.println
("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println
("*****");
        //Build the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Configure the parallelism.
        env.setParallelism(1);
        //Parse the execution parameter.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Build the StreamGraph, read data from Kafka and print the result in another row.
        DataStream<String> messageStream = env.addSource(new
FlinkKafkaConsumer<>(paraTool.get("topic"),
            new SimpleStringSchema(),
            paraTool.getProperties()));

        messageStream.rebalance().map(new MapFunction<String, String>() {
            @Override
            public String map(String s) throws Exception {
                return "Flink says " + s + System.getProperty("line.separator");
            }
        }).print();
        //Call execute to trigger the execution.
        env.execute();
    }
}
```

### 1.8.3.2.3 Scala Sample Code

## Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

## Sample Code

If you want to use FusionInsight in security mode, ensure that the **kafka-clients.jar** is obtained from the FusionInsight client directory.

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see com.huawei.bigdata.flink.examples.WriteIntoKafka and com.huawei.bigdata.flink.examples.ReadFromKafka.

```
//Code of producer
object WriteIntoKafka {

    def main(args: Array[String]) {
        //Print the reference command of flink run.
        System.out.println("use command as: ")

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005")

        System.out.println
        ("*****")
        System.out.println("<topic> is the kafka topic name")
        System.out.println("<bootstrap.servers> is the ip:port list of brokers")
        System.out.println
        ("*****")
        //Build the execution environment.
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        //Configure the parallelism.
        env.setParallelism(1)
        //Parse the execution parameter.
        val paraTool = ParameterTool.fromArgs(args)
        //Build the StreamGraph and write data generated by customized source into Kafka
        val messageStream: DataStream[String] = env.addSource(new SimpleStringGenerator)

        messageStream.addSink(new FlinkKafkaProducer(
            paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))
        //Call execute to trigger the execution.
        env.execute
    }
}

//Customize source to continuously generate messages every one second.
class SimpleStringGenerator extends SourceFunction[String] {

    var running = true
    var i = 0

    override def run(ctx: SourceContext[String]) {
        while (running) {
            ctx.collect("element-" + i)
            i += 1
            Thread.sleep(1000)
        }
    }
}
```

```
    }

}

override def cancel() {
    running = false
}

//consumer code
object ReadFromKafka {

    def main(args: Array[String]) {
        //Print the reference command of flink run.
        System.out.println("use command as: ")

        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005")

        System.out.println("*****")
        System.out.println("<topic> is the kafka topic name")
        System.out.println("<bootstrap.servers> is the ip:port list of brokers")
        System.out.println("*****")

        //Build the execution environment
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        //Configure the parallelism.
        env.setParallelism(1)
        //Parse the execution parameter.
        val paraTool = ParameterTool.fromArgs(args)
        //Build the StreamGraph, read data from Kafka and print the result in another row.
        val messageStream = env.addSource(new FlinkKafkaConsumer(
            paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))

        messageStream
            .map(s => "Flink says " + s + System.getProperty("line.separator")).print()
        //Call execute to trigger the execution
        env.execute()
    }
}
```

### 1.8.3.3 Asynchronous Checkpoint Mechanism

#### 1.8.3.3.1 Scenarios

##### Scenarios

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status. In

this case, when the application is recovered from failure, all operator statuses are the same.

## Data Planning

1. Customized operators generate about 10000 pieces of data per second.
2. Generated data is of four tuples (Long, String, String, Integer).
3. Statistic results are printed on the devices.
4. Printed data is of the long type.

## Development Approach

1. The source operator sends 10000 pieces of data and injects the data to the window operator every second.
2. The window operator calculates the data volume of preceding 4 seconds at the interval of one second.
3. The statistics is printed to the device at the interval of one second. Please refer to the specific[Viewing the Debugging Result](#)
4. The checkpoint is triggered at the interval of 6 seconds and the checkpoint result is stored in HDFS.

### 1.8.3.3.2 Java Sample Code

#### Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

#### Sample Code

1. Snapshot data

The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

```
import java.io.Serializable;

//The class is a part of the snapshot and is used to store user-defined statuses.
public class UDFState implements Serializable {
    private long count;

    //Initialize user-defined statuses.
    public UDFState() {
        count = 0L;
    }

    //Configure self-defined statuses.
    public void setState(long count) {
        this.count = count;
    }

    //Obtain user-defined statuses.
    public long getState() {
        return this.count;
    }
}
```

2. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the count variable.

```
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.source.RichSourceFunction;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

//The class is the source operator with checkpoint.
public class SEventSourceWithChk extends RichSourceFunction<Tuple4<Long, String, String, Integer>>
implements ListCheckpointed<UDFState> {
    private Long count = 0L;
    private boolean isRunning = true;
    private String alphabet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnoprstuvwxyzABCDEFGHJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnoprstuvwxyzABCDEFGHIJKLMNPQRSTUVWXYZ0987654321";

    //The main logic of the operator is to inject 10000 tuples to the StreamGraph.
    public void run(SourceContext<Tuple4<Long, String, String, Integer>> ctx) throws Exception {
        Random random = new Random();
        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple4.of(random.nextLong(), "hello-" + count, alphabet, 1))
                count++;
            }
            Thread.sleep(1000);
        }
    }

    //Call this when the task is canceled.
    public void cancel() {
        isRunning = false;
    }

    //Customize a snapshot.
    public List<UDFState> snapshotState(long l, long ll) throws Exception {
        UDFState udfState = new UDFState();
        List<UDFState> listState = new ArrayList<UDFState>();
        udfState.setState(count);
        listState.add(udfState);
        return listState;
    }

    //Restore data from customized snapshots.
    public void restoreState(List<UDFState> list) throws Exception {
        UDFState udfState = list.get(0);
        count = udfState.getState();
    }
}
```

### 3. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```
import org.apache.flink.api.java.tuple.Tuple;
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.windowing.WindowFunction;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;

import java.util.ArrayList;
import java.util.List;
```

```
//The class is the window operator with checkpoint.
public class WindowStatisticWithChk implements WindowFunction<Tuple4<Long, String, String, Integer>, Long, Tuple, TimeWindow>, ListCheckpointed<UDFState> {
    private Long total = 0L;

    //The implementation logic of the window operator, which is used to calculate the number of tuples in a window.
    void apply(Tuple key, TimeWindow window, Iterable<Tuple4<Long, String, String, Integer>> input,
               Collector<Long> out) throws Exception {
        long count = 0L;
        for (Tuple4<Long, String, String, Integer> event : input) {
            count++;
        }
        total += count;
        out.collect(count);
    }

    //Customize snapshot.
    public List<UDFState> snapshotState(Long l, Long ll) {
        List<UDFState> listState = new ArrayList<UDFState>();
        UDFState udfState = new UDFState();
        udfState.setState(total);
        listState.add(udfState);
        return listState;
    }

    //Restore data from customized snapshots.
    public void restoreState(List<UDFState> list) throws Exception {
        UDFState udfState = list.get(0);
        total = udfState.getState();
    }
}
```

#### 4. Application code

The code is about the definition of StreamGraph and is used to implement services. The processing time is used as the timestamp for triggering the window.

```
import org.apache.flink.runtime.state.filesystem.FsStateBackend;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;

public class FlinkProcessingTimeAPIChkMain {
    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        //Set configurations and enable checkpoint.
        env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"));
        env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
        env.getCheckpointConfig.setCheckpointInterval(6000);

        //Application logic.
        env.addSource(new SEventSourceWithChk())
            .keyBy(0)
            .window(SlidingProcessingTimeWindows.of(Time.seconds(4), Time.seconds(1)))
            .apply(new WindowStatisticWithChk())
            .print()

        env.execute();
    }
}
```

### 1.8.3.3.3 Scala Sample Code

## Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

# Sample Code

- ## 1. Formats of sent data.

```
case class SEvent(id: Long, name: String, info: String, count: Int)
```

- ## 2. Snapshot data

The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

//User-defined statuses.

```
class UDFState extends Serializable{  
    private var count = 0L  
  
    //Configure user-defined statuses.  
    def setState(s: Long) = count = s  
  
    //Obtain user-defined statuses.  
    def getState = count  
}
```

- ### 3. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the *count* variable.

```
//Create a snapshot.
override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {
    val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]
    val udfState = new UDFState
    udfState.setState(count)
    udfList.add(udfState)
    udfList
}

//Obtain status from the snapshot.
override def restoreState(list: util.List[UDFState]): Unit = {
    val udfState = list.get(0)
    count = udfState.getState
}
}
```

#### 4. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```
import java.util
import org.apache.flink.api.java.tuple.Tuple
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed
import org.apache.flink.streaming.api.scala.function.WindowFunction
import org.apache.flink.streaming.api.windowing.windows.TimeWindow
import org.apache.flink.util.Collector

//The class is the window operator with checkpoint.
class WindowStatisticWithChk extends WindowFunction[SEvent, Long, Tuple, TimeWindow] with
ListCheckpointed[UDFState]{
    private var total = 0L

    //The implementation logic of the window operator, which is used to calculate the number of
    tuples in a window.
    override def apply(key: Tuple, window: TimeWindow, input: Iterable[SEvent], out: Collector[Long]): Unit = {
        var count = 0L
        for (event <- input) {
            count += 1L
        }
        total += count
        out.collect(count)
    }

    //Customize a snapshot.
    override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {
        val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]
        val udfState = new UDFState
        udfState.setState(total)
        udfList.add(udfState)
        udfList
    }

    //Restore data from customized snapshots.
    override def restoreState(list: util.List[UDFState]): Unit = {
        val udfState = list.get(0)
        total = udfState.getState
    }
}
```

#### 5. Application code

The code is about the definition of StreamGraph and is used to implement services. The **event time** is used as the timestamp for triggering the window.

```
import com.huawei.rt.flink.core.{SEvent, SEventSourceWithChk, WindowStatisticWithChk}
import org.apache.flink.contrib.streaming.state.RocksDBStateBackend
import org.apache.flink.streaming.api.functions.AssignerWithPeriodicWatermarks
import org.apache.flink.streaming.api.{CheckpointingMode, TimeCharacteristic}
import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
```

```
import org.apache.flink.streaming.api.watermark.Watermark
import org.apache.flink.streaming.api.windowing.assigners.SlidingEventTimeWindows
import org.apache.flink.streaming.api.windowing.time.Time
import org.apache.flink.api.scala._
import org.apache.flink.runtime.state.filesystem.FsStateBackend
import org.apache.flink.streaming.api.environment.CHECKPOINT_CONFIG.ExternalizedCheckpointCleanup

object FlinkEventTimeAPIChkMain {
    def main(args: Array[String]): Unit ={
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
        env.getConfig.setAutoWatermarkInterval(2000)
        env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
        env.getCheckpointConfig.setCheckpointInterval(6000)

        //Application logic.
        env.addSource(new SEventSourceWithChk)
            .assignTimestampsAndWatermarks(new AssignerWithPeriodicWatermarks[SEvent] {
                //Configure watermark.
                override def getCurrentWatermark: Watermark = {
                    new Watermark(System.currentTimeMillis())
                }
                //Add timestamp for each tuple.
                override def extractTimestamp(t: SEvent, l: Long): Long = {
                    System.currentTimeMillis()
                }
            })
            .keyBy(0)
            .window(SlidingEventTimeWindows.of(Time.seconds(4), Time.seconds(1)))
            .apply(new WindowStatisticWithChk)
            .print()
        env.execute()
    }
}
```

### 1.8.3.4 Job Pipeline Program

#### 1.8.3.4.1 Scenario

##### Scenario

Assume that there are three jobs, a publisher and two subscribers. The publisher generates 10000 pieces of data each second. Data is sent by the NettySink operator from the publisher job to downstream subscribers which subscribe a same piece of data.

##### Data Planning

1. The publisher uses customized operators to generate about 10000 pieces of data per second.
2. The data contains two attributes, which are of integer and string types.
3. Configuration file
  - nettyconnector.registerserver.topic.storage: (Mandatory) Configures the path (on a third-party server) to information about IP address, port numbers, and concurrency of NettySink. For example:  
nettyconnector.registerserver.topic.storage: /flink/nettyconnector
  - nettyconnector.sinkserver.port.range: (Mandatory) Configures the range of port numbers of NettySink. For example:  
nettyconnector.sinkserver.port.range: 28444-28943

- nettyconnector.ssl.enabled: Configures whether to enable SSL encryption between NettySink and NettySource. The default value is false. For example:  
nettyconnector.ssl.enabled: true
- nettyconnector.sinkserver.subnet: Configure the network domain. For example:  
nettyconnector.sinkserver.subnet: 10.162.0.0/16

#### 4. Security authentication configuration:

- SASL authentication of ZooKeeper depends on HA-related configurations in **flink-conf.yaml**.
- SSL configurations such as keystore, truststore, keystore password, truststore password, and password inherit from **flink-conf.yaml**. For details, see [Encrypted Transmission](#).

#### 5. Description of APIs

- RegisterServer API

RegisterServerHandler stores information such as IP address, port number, and concurrency of NettySink for the connection with NettySource. Following APIs are provided for users:

```
public interface RegisterServerHandler {  
  
    /**  
     * Start the RegisterServer  
  
     * @param The configuration indicates the configuration type of Flink.  
  
     */  
    void start(Configuration configuration) throws Exception;  
    /**  
     *Create a topic node (directory) on the RegisterServer  
     *  
     * @param The topic indicates the name of the topic node  
  
     */  
    void createTopicNode(String topic) throw Exception;  
    /**  
     *Register the information to a topic node (directory)  
  
     * @param topic @param The topic indicates the directory to be registered with  
  
     * @param The registerRecord indicates the information to be registered  
  
     */  
    void register(String topic, RegisterRecord registerRecord) throws Exception;  
    /**  
     *Delete the topic node.  
     * @param The topic indicates the topic to be deleted  
     */  
    void deleteTopicNode(String topic) throws Exception;  
    /**  
     *Deregister the registration inDeregister the registration information formation  
     * @param The topic indicates the topic where the registration information locates.  
     * @param The recordId indicates the ID of the registration information to be deregistered  
  
     */  
    void unregister(String topic, int recordId) throws Exception;  
    /**  
     * Query information  
     * @param Topic where the query information locates  
     * @param The recordId indicates the ID of the query information  
  
     */  
    RegisterRecord query(String topic, int recordId) throws Exception;
```

```
/**  
 * Query whether a topic exist.  
 * @param topic  
 */  
Boolean isExist(String topic) throws Exception;  
/**  
 *Disable RegisterServerHandler.  
 */  
void shutdown() throws Exception;
```

In addition to the preceding APIs, Flink provides ZookeeperRegisterHandler.

- NettySink operator

```
Class NettySink(String name,  
String topic,  
RegisterServerHandler registerServerHandler,  
int numberOfSubscribedJobs)
```

- name: Name of a current NettySink.
- topic: The topic that generates data for the current NettySink. Different NettySinks must use different topics. Otherwise, the subscription may be disordered and data transmission may be abnormal.
- registerServerHandler: Handler of the registration server.
- numberOfSubscribedJobs: Specific number of jobs that subscribe the current NettySink. NettySink sends data only when all subscribers are connected to NettySink.

- NettySource operator

```
Class NettySource(String name,  
String topic,  
RegisterServerHandler registerServerHandler)
```

- name: name of the NettySource. The NettySource must be unique (concurrency excluded). Otherwise, connection with NettySink may be conflicted.
- topic: topic of subscribed NettySink.
- registerServerHandler: Handler of the registration server.

 NOTE

The concurrency of NettySource and the concurrency NettySink must be the same. Otherwise, the connection cannot be created.

## Development Approach

1. There are three jobs, one publisher and two subscribers.
2. The publisher transforms generated data into byte[] and sends them to the subscribers.
3. After receiving the byte[], subscribers transform data into the string type and print sampled data.

### 1.8.3.4.2 Java Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- com.huawei.bigdata.flink.examples.UserSource.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySink.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySource1.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySource2.

1. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;

import java.io.Serializable;

public class UserSource extends RichParallelSourceFunction<Tuple2<Integer, String>> implements Serializable {

    private boolean isRunning = true;

    public void open(Configuration configuration) throws Exception {
        super.open(configuration);

    }

    /**
     * Data generation function, which is used to generate 10000 pieces of data each second.
     */
    public void run(SourceContext<Tuple2<Integer, String>> ctx) throws Exception {

        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple2.of(i, "hello-" + i));
            }
            Thread.sleep(1000);
        }
    }

    public void close() {
        isRunning = false;
    }

    public void cancel() {
        isRunning = false;
    }
}
```

2. Code for the publisher:

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.sink.NettySink;
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;

public class TestPipeline_NettySink {

    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency of job to 2.
        env.setBufferTimeout(2);

        //Create a ZookeeperRegisterServerHandler.
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
```

```
// Add a customized source operator.  
    env.addSource(new UserSource()  
        .keyBy(0)  
        .map(new MapFunction<Tuple2<Integer, String>, byte[]>() {  
            //Transform the to-be-sent data into a byte array.  
  
        @Override  
            public byte[] map(Tuple2<Integer, String> integerStringTuple2) throws Exception {  
                return integerStringTuple2.f1.getBytes();  
            }  
        }).addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2));//NettySink  
transmits the data.  
  
        env.execute();  
    }  
}
```

### 3. Code for the first subscriber.

```
package com.huawei.bigdata.flink.examples;  
  
import org.apache.flink.api.common.functions.MapFunction;  
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
import org.apache.flink.streaming.connectors.netty.source.NettySource;  
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;  
  
public class TestPipeline_NettySource1 {  
  
    public static void main(String[] args) throws Exception{  
  
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
        // Set the concurrency of job to 2.  
  
        env.setParallelism(2);  
  
        // Create a ZookeeperRegisterServerHandler.  
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();  
        //Add a NettySource operator to receive messages from the publisher.  
        env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler))  
            .map(new MapFunction<byte[], String>() {  
                // Transform the received byte stream into character strings  
            @Override  
                public String map(byte[] b) {  
                    return new String(b);  
                }  
            }).print();  
  
        env.execute();  
    }  
}
```

### 4. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples;  
  
import org.apache.flink.api.common.functions.MapFunction;  
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
import org.apache.flink.streaming.connectors.netty.source.NettySource;  
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;  
  
public class TestPipeline_NettySource2 {  
  
    public static void main(String[] args) throws Exception {  
  
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
        // Set the concurrency of job to 2.  
        env.setParallelism(2);  
  
        //Create a ZookeeperRegisterServerHandler.  
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
```

```
//Add a NettySource operator to receive messages from the publisher.  
    env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler))  
        .map(new MapFunction<byte[], String>() {  
            //Transform the received byte array into character strings.  
            @Override  
            public String map(byte[] b) {  
                return new String(b);  
            }  
        }).print();  
  
    env.execute();  
}
```

#### 1.8.3.4.3 Scala Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- com.huawei.bigdata.flink.examples.UserSource.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySink.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySource1.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySource2.

##### 1. Code for sending messages:

```
package com.huawei.bigdata.flink.examples  
  
case class Inforamtion(index: Int, content: String) {  
  
    def this() = this(0, "")  
}
```

##### 2. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples  
  
import org.apache.flink.configuration.Configuration  
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction  
import org.apache.flink.streaming.api.functions.source.SourceFunction.SourceContext  
  
class UserSource extends RichParallelSourceFunction[Inforamtion] with Serializable{  
  
    var isRunning = true  
  
    override def open(parameters: Configuration): Unit = {  
        super.open(parameters)  
    }  
  
    // Generate 10000 pieces of data each second.  
    override def run(sourceContext: SourceContext[Inforamtion]) = {  
  
        while (isRunning) {  
            for (i <- 0 until 10000) {  
                sourceContext.collect(Inforamtion(i, "hello-" + i));  
            }  
            Thread.sleep(1000)  
        }  
    }  
  
    override def close(): Unit = super.close()  
  
    override def cancel() = {  
        isRunning = false  
    }  
}
```

```
}
```

### 3. Code for the publisher:

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.sink.NettySink
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

object TestPipeline_NettySink {

    def main(args: Array[String]): Unit = {

        val env = StreamExecutionEnvironment.getExecutionEnvironment
        // Set the concurrency of job to 2.
        env.setParallelism(2)
        //Set Zookeeper as the registration server
        val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
        //Add a user-defined operator to generate data.
        env.addSource(new UserSource).keyBy(0).map(x=>x.content.getBytes)//Transform the to-be-sent data
        into a byte array.
        .addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2))//Add NettySink
        operator to send data.
        env.execute()
    }
}
```

### 4. Code for the first subscriber

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.source.NettySource
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

import scala.util.Random

object TestPipeline_NettySource1 {

    def main(args: Array[String]): Unit = {

        val env = StreamExecutionEnvironment.getExecutionEnvironment
        // Set the concurrency of job to 2.
        env.setParallelism(2)
        //Set ZooKeeper as the registration server
        val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
        //Add a NettySource operator to receive messages from the publisher.
        env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler))
        .map(x => (1, new String(x)))//Add a NettySource operator to receive messages from the publisher.
        .filter(x => {
            Random.nextInt(50000) == 10
        })
        .print

        env.execute()
    }
}
```

### 5. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.source.NettySource
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._
```

```
import scala.util.Random

object TestPipeline_NettySource2 {
    def main(args: Array[String]): Unit = {
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        //Set the concurrency of job to 2.
        env.setParallelism(2)
        //Set the concurrency of job to 2.
        val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
        //Add NettySource operator and receive data.

        env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler))
            .map(x=>(2, new String(x)))//Transform the received byte array into character strings.

            .filter(x=>{
                Random.nextInt(50000) == 10
            })
            .print()

        env.execute()
    }
}
```

### 1.8.3.5 JOIN Operation between Configuration Tables and Streams

#### 1.8.3.5.1 Scenario Description

##### Scenario

Assume that there is a log text file about dwell durations of netizens for shopping online and a CSV table about netizen information. Develop a Flink application that achieves following functions:

- Collects statistics on female netizens who spend more than 2 hours in total for online shopping during the weekend in real time.  
The name fields in the log file and the CSV table can be used as keywords, based on which the two files are joined.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration (in minutes). Three columns are separated by commas (,).

**data.txt:** logs of netizen dwell duration on weekends

```
LA,female,20
YA,male,10
GA,male,5
CA,female,50
LAA,male,20
FA,female,50
LA,female,20
YA,male,10
GA,male,50
CA,female,50
FA,female,60
LA,female,20
YA,male,10
CA,female,50
FA,female,50
GA,male,5
```

```
CA,female,50  
LB,male,20  
CA,female,50  
FA,female,50  
LA,female,20  
YA,male,10  
FA,female,50  
GA,male,50  
CA,female,50  
FA,female,60  
NotExist,female,200
```

- **configtable.csv**: contains netizens' personal information. Fields starting from the first column to the ninth column are the name, age, company name, work place, academic degree, years of working, mobile number, domicile place, and school of graduation. These columns are separated by commas (,).

```
username,age,company,workLocation,educational,workYear,phone,nativeLocation,school  
LA,25,C1,W1,college,5,1312345678,N1,S1 university  
YA,26,C2,W2,master,6,1312345679,N2,S2 university  
GA,27,C3,W3,college,7,1312345680,N3,S3 university  
CA,28,C4,W4,master,8,1312345681,N4,S4 university  
LB,29,C5,W5,doctor,9,1312345682,N5,S5 university  
FA,30,C6,W6,doctor,10,1312345683,N6,S6 university
```

## Data Preparation

The stream data of the example project is saved in a TXT file, and the configuration table is in CSV format.

1. Ensure that clusters including HDFS, Yarn, secure Redis, and Flink are successfully installed.

### NOTE

- Redis in normal mode can be installed in clusters in security mode. To install Redis in security mode, set **REDIS\_SECURITY\_ENABLED** to **true** during installation. To install Redis in normal mode, set **REDIS\_SECURITY\_ENABLED** to **false**. When Redis is in security mode, users are authenticated using Kerberos, which affects performance. When Redis is in normal mode, users can directly access intranet clients without the need to be authenticated.
- The sample code is used to install secure Redis in a security cluster.
- For details about the sample code for installing normal Redis in a security cluster, see [JOIN Operation between Configuration Tables and Streams](#).

2. Create a Redis cluster, add the Redis user and grant permissions, and download the **user.keytab** and **krb5.conf** files.
3. Modify the **import.properties** and **read.properties** files, which are in the **config** directory specified in the sample code.

### NOTE

In the Scala sample code, the IP addresses and port numbers of all the instances in the logical cluster must be configured for the **Redis\_IP\_Port** parameter.

The formula for calculating the port number of the Redis instance is:  $22400 + Instance ID - 1$ .

To obtain the instance ID, choose **Cluster > Name of the desired cluster > Service > Redis > Redis Manager** on FusionInsight Manager and click the target Redis cluster name. For example, the port numbers of the Redis instances corresponding to the role R1 and role R2 are 22400 ( $22400 + 1 - 1 = 22400$ ) and 22401 ( $22400 + 2 - 1 = 22401$ ), respectively.

- The following is an example of configurations in the **import.properties** file:

```
#path to read csv files, it can be file or directory  
CsvPath=config/configtable.csv  
  
#csv file headers exist in file first line or not  
CsvHeaderExist=true  
#csv file headers, also the redis field names  
#Notice: if CsvHeaderExist false, you must set it, if CsvHeaderExist true, it read from csv file  
ColumnNames=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school  
  
#redis security mode open or not  
Redis_Security=true  
#redis hostname/ip and port when you need to connect to redis  
Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400  
#redis user principal  
Redis_Principal=test11@HADOOP.COM  
#redis keytab file path  
Redis_KeytabFile=config/user.keytab  
#redis krb5 file path  
Redis_Krb5File=config/krb5.conf  
# redis ssl on  
Redis_ssl_on=false
```

- The following is an example of configurations in the **read.properties** file:  
#the redis field names, configure which fields you need to read, Notice you need English field names  
ReadFields=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school  
#redis security mode open or not  
Redis\_Security=true  
#redis hostname/ip and port when you need to connect to redis  
Redis\_IP\_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400  
#redis user principal  
Redis\_Principal=test11@<System domain name>  
#redis keytab file path  
Redis\_KeytabFile=config/user.keytab  
#redis krb5 file path  
Redis\_Krb5File=config/krb5.conf  
# redis ssl on  
Redis\_ssl\_on=false

4. On the Flink client, create a **config** directory, and copy the **user.keytab**, **krb5.conf**, **data.txt**, **configtable.csv**, **import.properties**, and **read.properties** files to the **config** directory, for example, `/opt/hadoopclient/Flink/flink/config/configtable.csv`.

 NOTE

The **data.txt** and **configtable.csv** files are in the **data** directory specified in the sample code.

## Development Guideline

Collect the detailed information about female netizens who continuously spend more than 2 hours on online shopping on the current weekend.

To achieve the objective, the process is as follows:

- Modify the configurations in **import.properties** and **read.properties**, configure fields in the CSV file, fields read by Redis, and Redis security.
- Import the **configtable.csv** table to Redis for storage.
- Read text data, generate DataStreams, and parse data to generate OriginalRecord information.

- Call the function of asynchronous I/O, use the OriginalRecord username field as the keyword to query personal information in Redis, and transform the information into UserRecord.
- Filter the data about the time that female netizens spend online.
- Perform keyby operation based on names, and collect the time that female netizens spend online within a time window.
- Filter data about netizens whose consecutive online duration exceeds the threshold, and obtain the results.

### 1.8.3.5.2 Java Sample Code

#### Description

Collect the information from the log file about female netizens who continuously spend more than 2 hours in online shopping, collect personal information from the CSV table, and use the names of netizens as the keywords for joining the log file and the CSV table.

#### Sample Code

The following code snippet is used as an example for importing the **configtable.csv** file to Redis clusters in security mode. For the complete code, see [com.huawei.bigdata.flink.examples.RedisDataImport](#).

```
package com.huawei.bigdata.flink.examples;

import com.huawei.bigdata.security.LoginUtil;
import org.apache.flink.api.java.utils.ParameterTool;

import org.supercsv.cellprocessor.constraint.NotNull;
import org.supercsv.cellprocessor.ift.CellProcessor;
import org.supercsv.io.CsvBeanReader;
import org.supercsv.io.ICsvBeanReader;
import org.supercsv.prefs.CsvPreference;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

/**
 * Read data from csv file and import to redis.
 */
public class RedisDataImport {

    private static final int MAX_ATTEMPTS = 2;

    private static final int TIMEOUT = 60000;

    public static void main(String[] args) throws Exception {
        // print comment for command to use run flink
        System.out.println("use command as: \n" +
            "java -cp /opt/FI-Client/Flink/flink/lib/*:/opt/FlinkConfigtableJavaExample.jar" +
            " com.huawei.bigdata.flink.examples.RedisDataImport --configPath <config filePath>" +
            "*****\n" +
            "<config filePath> is for configure file to load\n" +
            "you may write following content into config filePath: \n" +
            "CsvPath=config/configtable.csv\n" +
            "CsvHeaderExist=true\n" +
```

```
"ColumnNames=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school\n"
+
    "Redis_Security=true\n" +
    "Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
    "Redis_Principal=test11@<System domain name>\n" +
    "Redis_KeytabFile=config/user.keytab\n" +
    "Redis_Krb5File=config/krb5.conf\n" +
"*****");
}

// read all configures
final String configureFilePath = ParameterTool.fromArgs(args).get("configPath", "config/
import.properties");
final String csvFilePath = ParameterTool.fromPropertiesFile(configureFilePath).get("CsvPath", "config/
configtable.csv");
final boolean isHasHeaders =
ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("CsvHeaderExist", true);
final String csvScheme = ParameterTool.fromPropertiesFile(configureFilePath).get("ColumnNames");
final boolean isSecurity =
ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("Redis_Security", true);
final String redisIPPort = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_IP_Port");
final String principal = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_Principal");
final String keytab = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_KeytabFile");
final String krb5 = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_Krb5File");
final boolean ssl = ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("Redis_ssl_on",
false);

// init redis client
initRedis(isSecurity, principal, keytab, krb5);
Set<HostAndPort> hosts = new HashSet<HostAndPort>();
for (String hostAndPort : redisIPPort.split(",")) {
    hosts.add(new HostAndPort(hostAndPort.split(":")[0], Integer.parseInt(hostAndPort.split(":")[1])));
}

JedisPoolConfig poolConfig = new JedisPoolConfig();
final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();
final JedisCluster client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS, "", "", "",
poolConfig, ssl, socketFactory, null, null, null);

// get all files under csv file path
ArrayList<File> files = getListFiles(csvFilePath);
System.out.println("Read file or directory under " + csvFilePath
+ ", total file num: " + files.size() + ", columns: " + csvScheme);

// run read csv file and analyze it
for (int index = 0; index < files.size(); index++) {
    readWithCsvBeanReader(files.get(index).getAbsolutePath(), csvScheme, isHasHeaders, client);
}
client.close();
System.out.println("Data import finish!!!");

}

public static void initRedis(boolean isRedisSecurity, String userPrincipal, String keytabPath, String
krb5Path) throws IOException {
    // redis security
    System.setProperty("redis.authentication.jaas", isRedisSecurity ? "true" : "false");

    // check and set
    if (System.getProperty("redis.authentication.jaas", "false").equals("true")) {
        LoginUtil.setJaasFile(userPrincipal, keytabPath);
        LoginUtil.setKrb5Config(krb5Path);
    }
}

public static ArrayList<File> getListFiles(Object obj) {
    File directory = null;
    if (obj instanceof File) {
```

```
        directory = (File) obj;
    } else {
        directory = new File(obj.toString());
    }
    ArrayList<File> files = new ArrayList<File>();
    if (directory.isFile()) {
        files.add(directory);
        return files;
    } else if (directory.isDirectory()) {
        File[] fileArr = directory.listFiles();
        for (int i = 0; i < fileArr.length; i++) {
            File fileOne = fileArr[i];
            files.addAll(getListFiles(fileOne));
        }
    }
    return files;
}

/**
 * Sets up the processors used for read csv. There are 9 CSV columns. Empty
 * columns are read as null (hence the NotNull() for mandatory columns).
 */
* @return the cell processors
*/
private static CellProcessor[] getProcessors() {
    final CellProcessor[] processors = new CellProcessor[] {
        new NotNull(), // username
        new NotNull(), // age
        new NotNull(), // company
        new NotNull(), // workLocation
        new NotNull(), // educational
        new NotNull(), // workYear
        new NotNull(), // phone
        new NotNull(), // nativeLocation
        new NotNull(), // school
    };
    return processors;
}

private static void readWithCsvBeanReader(String path, String csvScheme, boolean isSkipHeader,
JedisCluster client) throws Exception {
    ICsvBeanReader beanReader = null;
    try {
        beanReader = new CsvBeanReader(new FileReader(path), CsvPreference.STANDARD_PREFERENCE);

        // the header elements are used to map the values to the bean (names must match)
        final String[] header = isSkipHeader ? beanReader.getHeader(true) : csvScheme.split(",");
        final CellProcessor[] processors = getProcessors();

        UserInfo userinfo;
        while( (userinfo = beanReader.read(UserInfo.class, header, processors)) != null ) {
            System.out.println(String.format("lineNo=%s, rowNo=%s, userinfo=%s",
beanReader.getLineNumber(),
                beanReader.getRowNumber(), userinfo));

            // set redis key and value
            client.hmset(userinfo.getKeyValue(), userinfo.getMapInfo());
        }
    } finally {
        if( beanReader != null ) {
            beanReader.close();
        }
    }
}
```

```
// define the UserInfo structure
public static class UserInfo {
    private String username;
    private String age;
    private String company;
    private String workLocation;
    private String educational;
    private String workYear;
    private String phone;
    private String nativeLocation;
    private String school;

    public UserInfo() {
    }

    public UserInfo(String nm, String a, String c, String w, String e, String wy, String p, String nl, String sc) {
        username = nm;
        age = a;
        company = c;
        workLocation = w;
        educational = e;
        workYear = wy;
        phone = p;
        nativeLocation = nl;
        school = sc;
    }

    public String toString() {
        return "UserInfo----[username: " + username + " age: " + age + " company: " + company
               + " workLocation: " + workLocation + " educational: " + educational
               + " workYear: " + workYear + " phone: " + phone + " nativeLocation: " + nativeLocation + "
school: " + school + "]";
    }

    // get key
    public String getKeyValue() {
        return username;
    }

    public Map<String, String> getMapInfo() {
        Map<String, String> info = new HashMap<String, String>();
        info.put("username", username);
        info.put("age", age);
        info.put("company", company);
        info.put("workLocation", workLocation);
        info.put("educational", educational);
        info.put("workYear", workYear);
        info.put("phone", phone);
        info.put("nativeLocation", nativeLocation);
        info.put("school", school);
        return info;
    }

    /**
     * @return the username
     */
    public String getUsername() {
        return username;
    }

    /**
     * @param username
     *          the username to set
     */
    public void setUsername(String username) {
        this.username = username;
    }
}
```

```
/*
 * @return the age
 */
public String getAge() {
    return age;
}

/**
 * @param age
 *      the age to set
 */
public void setAge(String age) {
    this.age = age;
}

/**
 * @return the company
 */
public String getCompany() {
    return company;
}

/**
 * @param company
 *      the company to set
 */
public void setCompany(String company) {
    this.company = company;
}

/**
 * @return the workLocation
 */
public String getWorkLocation() {
    return workLocation;
}

/**
 * @param workLocation
 *      the workLocation to set
 */
public void setWorkLocation(String workLocation) {
    this.workLocation = workLocation;
}

/**
 * @return the educational
 */
public String getEducational() {
    return educational;
}

/**
 * @param educational
 *      the educational to set
 */
public void setEducational(String educational) {
    this.educational = educational;
}

/**
 * @return the workYear
 */
public String getWorkYear() {
    return workYear;
}

/**
```

```
* @param workYear
*      the workYear to set
*/
public void setWorkYear(String workYear) {
    this.workYear = workYear;
}

/**
* @return the phone
*/
public String getPhone() {
    return phone;
}

/**
* @param phone
*      the phone to set
*/
public void setPhone(String phone) {
    this.phone = phone;
}

/**
* @return the nativeLocation
*/
public String getNativeLocation() {
    return nativeLocation;
}

/**
* @param nativeLocation
*      the nativeLocation to set
*/
public void setNativeLocation(String nativeLocation) {
    this.nativeLocation = nativeLocation;
}

/**
* @return the school
*/
public String getSchool() {
    return school;
}

/**
* @param school
*      the school to set
*/
public void setSchool(String school) {
    this.school = school;
}
}
```

The following code snippet is used as an example. The function of the following code is to read stream data from the **log.txt** file, use netizen names as keywords to query from Redis clusters in security mode, perform the JOIN operation, and print the results. For the complete code, see [com.huawei.bigdata.flink.examples.FlinkConfigurableJavaExample](#).

```
package com.huawei.bigdata.flink.examples;

import com.huawei.bigdata.security.LoginUtil;
import org.apache.flink.api.common.functions.FilterFunction;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.common.functions.ReduceFunction;
import org.apache.flink.api.java.functions.KeySelector;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.configuration.Configuration;
```

```
import org.apache.flink.shaded.com.google.common.cache.CacheBuilder;
import org.apache.flink.shaded.com.google.common.cache.CacheLoader;
import org.apache.flink.shaded.com.google.common.cache.LoadingCache;
import org.apache.flink.streaming.api.datastream.AsyncDataStream;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.functions.AssignerWithPunctuatedWatermarks;
import org.apache.flink.streaming.api.functions.async.AsyncFunction;
import org.apache.flink.streaming.api.functions.async.RichAsyncFunction;
import org.apache.flink.streaming.api.functions.async.collector.AsyncCollector;
import org.apache.flink.streaming.api.watermark.Watermark;
import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;

import java.util.*;
import java.util.concurrent.TimeUnit;
/**
 * Read stream data and join from configure table from redis.
 */
public class FlinkConfigtableJavaExample {

    private static final int MAX_ATTEMPTS = 2;

    private static final int TIMEOUT = 60000;

    public static void main(String[] args) throws Exception {
        // print comment for command to use run flink
        System.out.println("use command as: \n" +
            "./bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableJavaExample" +
            " -m yarn-cluster -yt /opt/config -yn 3 -yjm 1024 -ytm 1024 " +
            "/opt/FlinkConfigtableJavaExample.jar --dataPath config/data.txt" +
            "*****\n" +
            "Especially you may write following content into config filePath, as in config/read.properties: \n" +
            "+

"ReadFields=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school\n" +
"Redis_Security=true\n" +
"Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
"Redis_Principal=test11@<System domain name>\n" +
"Redis_KeytabFile=config/user.keytab\n" +
"Redis_Krb5File=config/krb5.conf\n" +
"*****");

        // set up the execution environment
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.setParallelism(1);

        // get configure and read data and transform to OriginalRecord
        final String dataPath = ParameterTool.fromArgs(args).get("dataPath", "config/data.txt");
        DataStream<OriginalRecord> originalStream = env.readTextFile(
            dataPath
        ).map(new MapFunction<String, OriginalRecord>() {
            @Override
            public OriginalRecord map(String value) throws Exception {
                return getRecord(value);
            }
        }).assignTimestampsAndWatermarks(
            new Record2TimestampExtractor()
        ).disableChaining();

        // read from redis and join to the whole user information
        AsyncFunction<OriginalRecord, UserRecord> function = new AsyncRedisRequest();
        // timeout set to 2 minutes, max parallel request num set to 5, you can modify this to optimize
        DataStream<UserRecord> result = AsyncDataStream.unorderedWait(
            originalStream,
```

```
        function,
        2,
        TimeUnit.MINUTES,
        5);

    // data transform
    result.filter(new FilterFunction<UserRecord>() {
        @Override
        public boolean filter(UserRecord value) throws Exception {
            return value.sex.equals("female");
        }
    }).keyBy(
        new UserRecordSelector()
    ).window(
        TumblingEventTimeWindows.of(Time.seconds(30))
    ).reduce(new ReduceFunction<UserRecord>() {
        @Override
        public UserRecord reduce(UserRecord value1, UserRecord value2)
            throws Exception {
            value1.shoppingTime += value2.shoppingTime;
            return value1;
        }
    }).filter(new FilterFunction<UserRecord>() {
        @Override
        public boolean filter(UserRecord value) throws Exception {
            return value.shoppingTime > 120;
        }
    }).print();

    // execute program
    env.execute("FlinkConfigtable java");
}

private static class UserRecordSelector implements KeySelector<UserRecord, String> {
    @Override
    public String getKey(UserRecord value) throws Exception {
        return value.name;
    }
}

// class to set watermark and timestamp
private static class Record2TimestampExtractor implements
AssignerWithPunctuatedWatermarks<OriginalRecord> {

    // add tag in the data of datastream elements
    @Override
    public long extractTimestamp(OriginalRecord element, long previousTimestamp) {
        return System.currentTimeMillis();
    }

    // Give the watermark to trigger the window to start execution, and use the value to check if the
    // window elements are ready.
    @Override
    public Watermark checkAndGetNextWatermark(OriginalRecord element, long extractedTimestamp) {
        return new Watermark(extractedTimestamp - 1);
    }
}

private static OriginalRecord getRecord(String line) {
    String[] elems = line.split(",");
    assert elems.length == 3;
    return new OriginalRecord(elems[0], elems[1], Integer.parseInt(elems[2]));
}

public static class OriginalRecord {
    private String name;
    private String sex;
    private int shoppingTime;
```

```
public OriginalRecord(String n, String s, int t) {
    name = n;
    sexy = s;
    shoppingTime = t;
}
}

public static class UserRecord {
    private String name;
    private int age;
    private String company;
    private String workLocation;
    private String educational;
    private int workYear;
    private String phone;
    private String nativeLocation;
    private String school;
    private String sexy;
    private int shoppingTime;

    public UserRecord(String nm, int a, String c, String w, String e, int wy, String p, String nl, String sc,
String sx, int st) {
        name = nm;
        age = a;
        company = c;
        workLocation = w;
        educational = e;
        workYear = wy;
        phone = p;
        nativeLocation = nl;
        school = sc;
        sexy = sx;
        shoppingTime = st;
    }

    public void setInput(String input_nm, String input_sx, int input_st) {
        name = input_nm;
        sexy = input_sx;
        shoppingTime = input_st;
    }

    public String toString() {
        return "UserRecord----name: " + name + " age: " + age + " company: " + company
               + " workLocation: " + workLocation + " educational: " + educational
               + " workYear: " + workYear + " phone: " + phone + " nativeLocation: " + nativeLocation + "
school: " + school
               + " sexy: " + sexy + " shoppingTime: " + shoppingTime;
    }
}

public static class AsyncRedisRequest extends RichAsyncFunction<OriginalRecord, UserRecord>{
    private String fields = "";
    private transient JedisCluster client;
    private LoadingCache<String, UserRecord> cacheRecords;

    @Override
    public void open(Configuration parameters) throws Exception {
        super.open(parameters);

        // init cache builder
        cacheRecords = CacheBuilder.newBuilder()
            .maximumSize(10000)
            .expireAfterAccess(7, TimeUnit.DAYS)
            .build(new CacheLoader<String, UserRecord>() {
                public UserRecord load(String key) throws Exception {
                    //load from redis
                    return loadFromRedis(key);
                }
            });
    }
}
```

```
// get configure from config/read.properties, you must put this with commands:  
// ./bin/yarn-session.sh -t config -jm 1024 -tm 1024 or  
// ./bin/flink run -m yarn-cluster -yt config -yn 3 -yjm 1024 -ytm 1024 /opt/test.jar  
String configPath = "config/read.properties";  
fields = ParameterTool.fromPropertiesFile(configPath).get("ReadFields");  
final boolean isSecurity =  
ParameterTool.fromPropertiesFile(configPath).getBoolean("Redis_Security", true);  
final String hostPort = ParameterTool.fromPropertiesFile(configPath).get("Redis_IP_Port");  
final String principal = ParameterTool.fromPropertiesFile(configPath).get("Redis_Principal");  
final String keytab = ParameterTool.fromPropertiesFile(configPath).get("Redis_KeytabFile");  
final String krb5 = ParameterTool.fromPropertiesFile(configPath).get("Redis_Krb5File");  
final boolean ssl = ParameterTool.fromPropertiesFile(configPath).getBoolean("Redis_ssl_on", false);  
  
// init redis security mode  
System.setProperty("redis.authentication.jaas", isSecurity ? "true" : "false");  
if (System.getProperty("redis.authentication.jaas", "false").equals("true")) {  
    LoginUtil.setJaasFile(principal, keytab);  
    LoginUtil.setKrb5Config(krb5);  
}  
  
// create jedisCluster client  
Set<HostAndPort> hosts = new HashSet<HostAndPort>();  
for (String node : hostPort.split(",")) {  
    hosts.add(new HostAndPort(node.split(":")[0], Integer.parseInt(node.split(":")[1])));  
}  
  
JedisPoolConfig poolConfig = new JedisPoolConfig();  
final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustALLSslSocketFactory();  
client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS, "", "", poolConfig, ssl,  
socketFactory, null, null, null);  
System.out.println("JedisCluster init, getClusterNodes: " + client.getClusterNodes().size());  
}  
  
@Override  
public void close() throws Exception {  
    super.close();  
  
    if (client != null) {  
        System.out.println("JedisCluster close!!!!");  
        client.close();  
    }  
}  
  
public UserRecord loadFromRedis(final String key) throws Exception {  
    if (client.getClusterNodes().size() <= 0) {  
        System.out.println("JedisCluster init failed, getClusterNodes: " + client.getClusterNodes().size());  
    }  
    if (!client.exists(key)) {  
        System.out.println("test-----cannot find data to key: " + key);  
        return new UserRecord(  
            "null",  
            0,  
            "null",  
            "null",  
            "null",  
            0,  
            "null",  
            "null",  
            "null",  
            "null",  
            0);  
    } else {  
        // get some fields  
        List<String> values = client.hmget(key, fields.split(", "));  
        System.out.println("test-----key: " + key + " get some fields: " + values.toString());  
        return new UserRecord(  
    }
```

```
        values.get(0),
        Integer.parseInt(values.get(1)),
        values.get(2),
        values.get(3),
        values.get(4),
        Integer.parseInt(values.get(5)),
        values.get(6),
        values.get(7),
        values.get(8),
        "null",
        0);
    }
}

public void asyncInvoke(final OriginalRecord input, final AsyncCollector<UserRecord> collector) throws
Exception {
    // Set key string, if your key is more than one column, build your key string with columns.
    String key = input.name;
    UserRecord info = cacheRecords.get(key);
    info.setInput(input.name, input.sex, input.shoppingTime);
    collector.collect(Collections.singletonList(info));
}
}
```

### 1.8.3.5.3 Scala Sample Code

#### Description

Collect the information from the log file about female netizens who continuously spend more than 2 hours in online shopping, collect personal information from the CSV table, and use the names of netizens as the keywords for joining the log file and the CSV table.

#### Sample Code

The following code snippet is used as an example for importing the **configurable.csv** file to Redis clusters in security mode. For the complete code, see [com.huawei.bigdata.flink.examples.RedisDataImport](#).

```
package com.huawei.bigdata.flink.examples;

import com.huawei.bigdata.security.LoginUtil;
import org.apache.flink.api.java.utils.ParameterTool;

import org.supercsv.cellprocessor.constraint.NotNull;
import org.supercsv.cellprocessor.ift.CellProcessor;
import org.supercsv.io.CsvBeanReader;
import org.supercsv.io.ICsvBeanReader;
import org.supercsv.prefs.CsvPreference;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

/**
 * Read data from csv file and import to redis.
 */
public class RedisDataImport {
    public static void main(String[] args) throws Exception {
        // print comment for command to use run flink
        System.out.println("use command as: \n" +
```

```

"java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/FlinkConfigtableJavaExample.jar" +
" com.huawei.bigdata.flink.examples.RedisDataImport --configPath <config filePath>" +
"*****\n" +
"<config filePath> is for configure file to load\n" +
"you may write following content into config filePath: \n" +
"CsvPath=config/configtable.csv\n" +
"CsvHeaderExist=true\n" +

"ColumnNames=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school\n" +
+
"Redis_Security=true\n" +
"Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
"Redis_Principal=test11@<System domain name>\n" +
"Redis_KeytabFile=config/user.keytab\n" +
"Redis_Krb5File=config/krb5.conf\n" +
"*****");

// read all configures
final String configureFilePath = ParameterTool.fromArgs(args).get("configPath", "config/
import.properties");
final String csvFilePath = ParameterTool.fromPropertiesFile(configureFilePath).get("CsvPath", "config/
configtable.csv");
final boolean isHasHeaders =
ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("CsvHeaderExist", true);
final String csvScheme = ParameterTool.fromPropertiesFile(configureFilePath).get("ColumnNames");
final boolean isSecurity =
ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("Redis_Security", true);
final String redisIPPort = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_IP_Port");
final String principal = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_Principal");
final String keytab = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_KeytabFile");
final String krb5 = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_Krb5File");
final boolean ssl = ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("Redis_ssl_on",
false);

// init redis client
initRedis(isSecurity, principal, keytab, krb5);
Set<HostAndPort> hosts = new HashSet<HostAndPort>();
for (String hostAndPort : redisIPPort.split(":")) {
    hosts.add(new HostAndPort(hostAndPort.split(":")[0], Integer.parseInt(hostAndPort.split(":")[1])));
}

JedisPoolConfig poolConfig = new JedisPoolConfig();
final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();
final JedisCluster client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS,"","");
poolConfig, ssl, socketFactory, null,null, null);

// get all files under csv file path
ArrayList<File> files = getListFiles(csvFilePath);
System.out.println("Read file or directory under " + csvFilePath
+ ", total file num: " + files.size() + ", columns: " + csvScheme);

// run read csv file and analyze it
for (int index = 0; index < files.size(); index++) {
    readWithCsvBeanReader(files.get(index).getAbsolutePath(), csvScheme, isHasHeaders, client);
}
client.close();
System.out.println("Data import finish!!!!");
}

public static void initRedis(boolean isRedisSecurity, String userPrincipal, String keytabPath, String
krb5Path) throws IOException {
    // redis security
    System.setProperty("redis.authentication.jaas", isRedisSecurity ? "true" : "false");

    // check and set
    if (System.getProperty("redis.authentication.jaas", "false").equals("true")) {
        LoginUtil.setJaasFile(userPrincipal, keytabPath);
    }
}

```

```
        LoginUtil.setKrb5Config(krb5Path);
    }

    public static ArrayList<File> getListFiles(Object obj) {
        File directory = null;
        if (obj instanceof File) {
            directory = (File) obj;
        } else {
            directory = new File(obj.toString());
        }
        ArrayList<File> files = new ArrayList<File>();
        if (directory.isFile()) {
            files.add(directory);
            return files;
        } else if (directory.isDirectory()) {
            File[] fileArr = directory.listFiles();
            for (int i = 0; i < fileArr.length; i++) {
                File fileOne = fileArr[i];
                files.addAll(getListFiles(fileOne));
            }
        }
        return files;
    }

    /**
     * Sets up the processors used for read csv. There are 9 CSV columns. Empty
     * columns are read as null (hence the NotNull() for mandatory columns).
     *
     * @return the cell processors
     */
    private static CellProcessor[] getProcessors() {
        final CellProcessor[] processors = new CellProcessor[] {
            new NotNull(), // username
            new NotNull(), // age
            new NotNull(), // company
            new NotNull(), // workLocation
            new NotNull(), // educational
            new NotNull(), // workYear
            new NotNull(), // phone
            new NotNull(), // nativeLocation
            new NotNull(), // school
        };
        return processors;
    }

    private static void readWithCsvBeanReader(String path, String csvScheme, boolean isSkipHeader,
JedisCluster client) throws Exception {
    ICsvBeanReader beanReader = null;
    try {
        beanReader = new CsvBeanReader(new FileReader(path), CsvPreference.STANDARD_PREFERENCE);

        // the header elements are used to map the values to the bean (names must match)
        final String[] header = isSkipHeader ? beanReader.getHeader(true) : csvScheme.split(",");
        final CellProcessor[] processors = getProcessors();

        UserInfo userinfo;
        while( (userinfo = beanReader.read(UserInfo.class, header, processors)) != null ) {
            System.out.println(String.format("lineNo=%s, rowNo=%s, userinfo=%s",
beanReader.getLineNumber(),
            beanReader.getRowNumber(), userinfo));

            // set redis key and value
            client.hmset(userinfo.getKeyValue(), userinfo.getMapInfo());
        }
    }
    finally {
        if( beanReader != null ) {
```

```
        beanReader.close();
    }
}

// define the UserInfo structure
public static class UserInfo {
    private String username;
    private String age;
    private String company;
    private String workLocation;
    private String educational;
    private String workYear;
    private String phone;
    private String nativeLocation;
    private String school;

    public UserInfo() {
    }

    public UserInfo(String nm, String a, String c, String w, String e, String wy, String p, String nl, String sc) {
        username = nm;
        age = a;
        company = c;
        workLocation = w;
        educational = e;
        workYear = wy;
        phone = p;
        nativeLocation = nl;
        school = sc;
    }

    public String toString() {
        return "UserInfo----[username: " + username + " age: " + age + " company: " + company
               + " workLocation: " + workLocation + " educational: " + educational
               + " workYear: " + workYear + " phone: " + phone + " nativeLocation: " + nativeLocation + "
school: " + school + "]";
    }

    // get key
    public String getKeyValue() {
        return username;
    }

    public Map<String, String> getMapInfo() {
        Map<String, String> info = new HashMap<String, String>();
        info.put("username", username);
        info.put("age", age);
        info.put("company", company);
        info.put("workLocation", workLocation);
        info.put("educational", educational);
        info.put("workYear", workYear);
        info.put("phone", phone);
        info.put("nativeLocation", nativeLocation);
        info.put("school", school);
        return info;
    }

    /**
     * @return the username
     */
    public String getUsername() {
        return username;
    }

    /**
     * @param username
     */
}
```

```
*      the username to set
*/
public void setUsername(String username) {
    this.username = username;
}

/**
 * @return the age
 */
public String getAge() {
    return age;
}

/**
 * @param age
 *      the age to set
 */
public void setAge(String age) {
    this.age = age;
}

/**
 * @return the company
 */
public String getCompany() {
    return company;
}

/**
 * @param company
 *      the company to set
 */
public void setCompany(String company) {
    this.company = company;
}

/**
 * @return the workLocation
 */
public String getWorkLocation() {
    return workLocation;
}

/**
 * @param workLocation
 *      the workLocation to set
 */
public void setWorkLocation(String workLocation) {
    this.workLocation = workLocation;
}

/**
 * @return the educational
 */
public String getEducational() {
    return educational;
}

/**
 * @param educational
 *      the educational to set
 */
public void setEducational(String educational) {
    this.educational = educational;
}

/**
 * @return the workYear
 */
```

```
public String getWorkYear() {
    return workYear;
}

/**
 * @param workYear
 *      the workYear to set
 */
public void setWorkYear(String workYear) {
    this.workYear = workYear;
}

/**
 * @return the phone
 */
public String getPhone() {
    return phone;
}

/**
 * @param phone
 *      the phone to set
 */
public void setPhone(String phone) {
    this.phone = phone;
}

/**
 * @return the nativeLocation
 */
public String getNativeLocation() {
    return nativeLocation;
}

/**
 * @param nativeLocation
 *      the nativeLocation to set
 */
public void setNativeLocation(String nativeLocation) {
    this.nativeLocation = nativeLocation;
}

/**
 * @return the school
 */
public String getSchool() {
    return school;
}

/**
 * @param school
 *      the school to set
 */
public void setSchool(String school) {
    this.school = school;
}
}
```

The following code snippet is used as an example. The function of the following code is to read stream data from the **data.txt** file, use netizen names as keywords to query from Redis clusters in security mode, perform the JOIN operation, and print the results. For the complete code, see [com.huawei.bigdata.flink.examples.FlinkConfigurableScalaExample](#).

```
package com.huawei.bigdata.flink.examples

import java.util.HashSet
import java.util.Set
```

```

import java.util.concurrent.TimeUnit

import com.huawei.bigdata.security.LoginUtil
import org.apache.flink.api.java.utils.ParameterTool
import org.apache.flink.streaming.api.TimeCharacteristic
import org.apache.flink.streaming.api.functions.AssignerWithPunctuatedWatermarks
import org.apache.flink.streaming.api.scala._
import org.apache.flink.streaming.api.scala.async.AsyncCollector
import org.apache.flink.streaming.api.watermark.Watermark
import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows
import org.apache.flink.streaming.api.windowing.time.Time
import redis.clients.jedis.HostAndPort
import redis.clients.jedis.JedisCluster

import scala.concurrent.{ExecutionContext, Future}

/**
 * Read stream data and join from configure table from redis.
 */
object FlinkConfigurableScalaExample {

    private val MAX_ATTEMPTS = 2

    private val TIMEOUT = 60000
    def main(args: Array[String]) {
        // print comment for command to use run flink
        System.out.println("use command as: \n" +
            "./bin/flink run -m yarn-cluster -yt /opt/config -yn 3 -yjm 1024 -ymt 1024 " +
            "/opt/FlinkConfigurableScalaExample.jar --dataPath config/data.txt" +
            "\n*****\n" +
            "Especially you may write following content into config filePath, as in config/read.properties: \n" +
            "ReadFields=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school\n" +
+
            "Redis_Security=true\n" +
            "Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
            "Redis_Principal=test11@<System domain name>\n" +
            "Redis_KeytabFile=config/user.keytab\n" +
            "Redis_Krb5File=config/krb5.conf\n" +
            "\n*****")}

        // set up the execution environment
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
        env.setParallelism(1)

        // get configure and read data and transform to OriginalRecord
        val dataPath = ParameterTool.fromArgs(args).get("dataPath", "config/data.txt")
        val originalStream = env.readTextFile(dataPath)
            .map(it => getRecord(it)).assignTimestampsAndWatermarks(new
        Record2TimestampExtractor).disableChaining()

        // read from redis and join to the whole user information
        val resultStream = AsyncDataStream.unorderedWait(
            originalStream,
            2,
            TimeUnit.MINUTES,
            5) {
            (input, collector: AsyncCollector[UserRecord]) =>
            Future {
                // get configure from config/read.properties, you must put this with commands:
                // ./bin/yarn-session.sh -t /opt/config -jm 1024 -tm 1024 or
                // ./bin/flink run -m yarn-cluster -yt /opt/config -yn 3 -yjm 1024 -ymt 1024 /opt/test.jar
                val configPath = "config/read.properties"
                val fields = ParameterTool.fromPropertiesFile(configPath).get("ReadFields")
                val isSecurity = ParameterTool.fromPropertiesFile(configPath).getBoolean("Redis_Security", true)
                val hostPort = ParameterTool.fromPropertiesFile(configPath).get("Redis_IP_Port")
                val principal = ParameterTool.fromPropertiesFile(configPath).get("Redis_Principal")
                val keytab = ParameterTool.fromPropertiesFile(configPath).get("Redis_KeytabFile")
            }
        }
    }
}

```

```
val krb5 = ParameterTool.fromPropertiesFile(configPath).get("Redis_Krb5File")
val ssl = ParameterTool.fromPropertiesFile(configPath).getBoolean("Redis_ssl_on", false)

// init redis security mode
System.setProperty("redis.authentication.jaas", if (isSecurity) "true" else "false")
if (System.getProperty("redis.authentication.jaas", "false").equals("true")) {
    LoginUtil.setJaasFile(principal, keytab)
    LoginUtil.setKrb5Config(krb5)
}

// create jedisCluster client
val hosts: Set[HostAndPort] = new HashSet[HostAndPort]()
hostPort.split(",").foreach(it => hosts.add(new HostAndPort(it.split(":").apply(0),
Integer.parseInt(it.split(":").apply(1)))))

val poolConfig = new JedisPoolConfig
val socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory
val client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS, "", "", poolConfig, ssl,
socketFactory, null, null, null)

System.out.println("JedisCluster init, getClusterNodes: " + client.getClusterNodes.size())

if (client.getClusterNodes.size() <= 0) {
    System.out.println("JedisCluster init failed, getClusterNodes: " + client.getClusterNodes.size())
}
// Set key string, if your key is more than one column, build your key string with columns
val key = input.name
if (!client.exists(key)) {
    System.out.println("test-----cannot find data to key: " + key)
    collector.collect(Seq(new UserRecord(
        input.name,
        0,
        "null",
        "null",
        "null",
        0,
        "null",
        "null",
        "null",
        input.sex,
        input.shoppingTime)))
} else {
    val values = client.hmget(key, fields.split(":"):_*)
    System.out.println("test-----key: " + key + " get some fields: " + values.toString)
    collector.collect(Seq(new UserRecord(
        values.get(0),
        Integer.parseInt(values.get(1)),
        values.get(2),
        values.get(3),
        values.get(4),
        Integer.parseInt(values.get(5)),
        values.get(6),
        values.get(7),
        values.get(8),
        input.sex,
        input.shoppingTime)))
}
client.close()
} (ExecutionContext.global)
}

// data transform
resultStream.filter(_.sex == "female")
.keyBy("name")
.window(TumblingEventTimeWindows.of(Time.seconds(30)))
.reduce((e1, e2) => UserRecord(e1.name, e2.age, e2.company, e2.workLocation, e2.educational,
e2.workYear,
e2.phone, e2.nativeLocation, e2.school, e2.sex, e1.shoppingTime + e2.shoppingTime))
```

```
.filter(_.shoppingTime > 120).print()

// execute program
env.execute("FlinkConfigtable scala")
}

// get enums of record
def getRecord(line: String): OriginalRecord = {
    val elems = line.split(",")
    assert(elems.length == 3)
    val name = elems(0)
    val sexy = elems(1)
    val time = elems(2).toInt
    OriginalRecord(name, sexy, time)
}

// the scheme of record read from txt
case class OriginalRecord(name: String, sexy: String, shoppingTime: Int)

case class UserRecord(name: String, age: Int, company: String, workLocation: String, educational: String,
workYear: Int,
    phone: String, nativeLocation: String, school: String, sexy: String, shoppingTime: Int)

// class to set watermark and timestamp
private class Record2TimestampExtractor extends AssignerWithPunctuatedWatermarks[OriginalRecord] {

    // add tag in the data of datastream elements
    override def extractTimestamp(element: OriginalRecord, previousTimestamp: Long): Long = {
        System.currentTimeMillis()
    }

    // Give the watermark to trigger the window to start execution, and use the value to check if the window
    // elements are ready.
    def checkAndGetNextWatermark(lastElement: OriginalRecord,
                                extractedTimestamp: Long): Watermark = {
        new Watermark(extractedTimestamp - 1)
    }
}
```

### 1.8.3.6 Stream SQL Join Program

#### 1.8.3.6.1 Scenario

##### Scenario

Assume that a Flink service (service 1) receives a message every second. The message records the basic information about a user, including the name, gender, and age. Another Flink service (service 2) receives a message irregularly, and the message records the name and career information about the user.

To meet the requirements of some services, the Flink application is developed to achieve the following function: uses the username recorded in the message received by service 2 as the keyword to jointly query two pieces of service data.

##### Data Planning

The data of service 1 is stored in the Kafka component. Service 1 sends data (requiring Kafka user rights) to and receives data from the Kafka component. For details about how to configure Kafka, see the data planning section of [Data Preparation](#)

Service 2 receives messages using the socket. You can run the **netcat** command to input the analog data source.

- Run the **netcat -l -p <port>** command to start a simple text server.
- After starting the application to connect to the port monitored by **netcat**, enter the data information to the netcat terminal.

## Development Approach

1. Start the Flink Kafka Producer application to send data to Kafka.
2. Start the Flink Kafka Consumer application to receive data from Kafka, construct **Table1**, and ensure that the Topic is the same as that of Producer.
3. Read data from the socket and construct **Table2**.
4. Use Flink SQL to query and print **Table1** and **Table2**.

### 1.8.3.6.2 Java Sample Code

#### Function

In the Flink application, this code invokes the flink-connector-kafka module's API to generate and consume data.

#### Sample Code

If the user needs to use FusionInsight Kafka interconnected with the security mode before the development, obtain the **kafka-clients-\*jar** JAR file from the Kafka client directory.

The following lists Producer, Consumer, and the main logic code used by Flink Stream SQL Join.

For the complete code, see  
[com.huawei.bigdata.flink.examples.WriteIntoKafka](#) and  
[com.huawei.bigdata.flink.examples.SqlJoinWithSocket](#).

1. A piece of user information is generated in Kafka every second. The user information includes the name, age, and gender.

```
//Producer code
public class WriteIntoKafka {
    public static void main(String[] args) throws Exception {
        //Print the command reference for flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");

        //Construct the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency.
        env.setParallelism(1);
        //Parse the running parameters.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Construct a flow diagram and write the data generated from self-defined sources to Kafka.
```

```

DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
FlinkKafkaProducer<String> producer = new FlinkKafkaProducer<>(paraTool.get("topic"),
    new SimpleStringSchema(),
    paraTool.getProperties());
producer.setWriteTimestampToKafka(true);
messageStream.addSink(producer);
//Invoke execute to trigger the execution.
env.execute();
}
//Customize the sources and generate a message every second.
public static class SimpleStringGenerator implements SourceFunction<String> {
    static final String[] NAME = {"Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James"};
    static final String[] SEX = {"MALE", "FEMALE"};
    static final int COUNT = NAME.length;
    boolean running = true;
    Random rand = new Random(47);
    @Override
    //Use rand to randomly generate a combination of the name, gender, and age.
    public void run(SourceContext<String> ctx) throws Exception {
        while (running) {
            int i = rand.nextInt(COUNT);
            int age = rand.nextInt(70);
            String sexy = SEX[rand.nextInt(2)];
            ctx.collect(NAME[i] + "," + age + "," + sexy);
            thread.sleep(1000);
        }
    }
    @Override
    public void cancel() {
        running = false;
    }
}
}

```

2. Generate **Table1** and **Table2**, use Join to jointly query **Table1** and **Table2**, and print the output result.

```

public class SqJoinWithSocket {
    public static void main(String[] args) throws Exception{
        final String hostname;
        final int port;
        System.out.println("use command as: ");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxxx.xxx.xxx:21005 --hostname
xxxx.xxxx.xxxx --port xxx");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket" +
            " /opt/test.jar --topic topic-test -bootstrap.servers xxxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka"
            + "--hostname xxx.xxx.xxxx --port xxx");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");
        try {
            final ParameterTool params = ParameterTool.fromArgs(args);
            hostname = params.has("hostname") ? params.get("hostname") : "localhost";
            port = params.getInt("port");
        } catch (Exception e) {
            System.err.println("No port specified. Please run 'FlinkStreamSqlJoinExample " +
                "--hostname <hostname> --port <port>', where hostname (localhost by default) " +
                "and port is the address of the text server");
            System.err.println("To start a simple text server, run 'netcat -l -p <port>' and " +
                "type the input text into the command line");
            return;
        }
        EnvironmentSettings fsSettings =
EnvironmentSettings.newInstance().useOldPlanner().inStreamingMode().build();
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env, fsSettings);
        //Perform processing based on EventTime.
    }
}

```

```
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
env.setParallelism(1);
ParameterTool paraTool = ParameterTool.fromArgs(args);
//Use Stream1 to read data from Kafka.
DataStream<Tuple3<String, String, String>> kafkaStream = env.addSource(new
FlinkKafkaConsumer<>(paraTool.get("topic"),
    new SimpleStringSchema(),
    paraTool.getProperties()).map(new MapFunction<String, Tuple3<String, String, String>>()
{
    @Override
    public Tuple3<String, String, String> map(String s) throws Exception {
        String[] word = s.split(",");
        return new Tuple3<>(word[0], word[1], word[2]);
    }
});
//Register Stream1 as Table1.
tableEnv.registerDataStream("Table1", kafkaStream, "name, age, sexy, proctime.proctime");
//Use Stream2 to read data from the socket.
DataStream<Tuple2<String, String>> socketStream = env.socketTextStream(hostname, port,
"\n").
map(new MapFunction<String, Tuple2<String, String>>() {
    @Override
    public Tuple2<String, String> map(String s) throws Exception {
        String[] words = s.split("\s");
        if (words.length < 2) {
            return new Tuple2<>();
        }
        return new Tuple2<>(words[0], words[1]);
    }
});
//Register Stream2 as Table2.
tableEnv.registerDataStream("Table2", socketStream, "name, job, proctime.proctime");
//Run SQL Join to perform a combined query.
Table result = tableEnv.sqlQuery("SELECT t1.name, t1.age, t1.sex, t2.job, t2.proctime as shiptime
\n" +
    "FROM Table1 AS t1\n" +
    "JOIN Table2 AS t2\n" +
    "ON t1.name = t2.name\n" +
    "AND t1.proctime BETWEEN t2.proctime - INTERVAL '1' SECOND AND t2.proctime +
INTERVAL '1' SECOND");
//Convert the query result into the stream and print the output.
tableEnv.toAppendStream(result, Row.class).print();
env.execute();
}
})
```

### 1.8.3.6.3 Scala Sample Code

#### Function

In a Flink application, call the API of the flink-connector-kafka module to produce and consume data.

#### Sample Code

If you need to interconnect with Kafka in security mode before application development, **kafka-clients-\*jar** of FusionInsight is required. You can obtain the JAR file from the Kafka client directory.

The following example provides the main code logic of Producer, Consumer, and Flink Stream SQL Join.

Complete code is provided in  
**com.huawei.bigdata.flink.examples.WriteIntoKafka** and  
**com.huawei.bigdata.flink.examples.SqJoinWithSocket**.

1. Produce a piece of user information in Kafka every second. The user information includes the name, age, and gender.

```
//Producer

object WriteIntoKafka {
    def main(args: Array[String]): Unit = {
        System.out.println("use command as: ")
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
        " /opt/test.jar --topic topic-test -bootstrap.servers xxx.xxx.xxx.xxx:21005")
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka")

        System.out.println("*****")
        System.out.println("<topic> is the kafka topic name")
        System.out.println("<bootstrap.servers> is the ip:port list of brokers")
        System.out.println("*****")
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        env.setParallelism(1)

        val paraTool = ParameterTool.fromArgs(args)

        val messageStream = env.addSource(new WriteIntoKafka.SimpleStringGenerator)
        val producer = new FlinkKafkaProducer[String](paraTool.get("topic"), new SimpleStringSchema,
        paraTool.getProperties)

        producer.setWriteTimestampToKafka(true)

        messageStream.addSink(producer)
        env.execute
    }

    /**
     * String source class
     *
     */
    object SimpleStringGenerator {
        private[examples] val NAME = Array("Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James")
        private[examples] val SEX = Array("MALE", "FEMALE")
        private[examples] val COUNT = NAME.length
    }

    class SimpleStringGenerator extends SourceFunction[String] {
        private[examples] var running = true
        private[examples] val rand = new Random(47)

        @throws[Exception]
        override def run(ctx: SourceFunction.SourceContext[String]): Unit = {
            while (running) {
                val i = rand.nextInt(SimpleStringGenerator.COUNT)
                val age = rand.nextInt(70)
                val sexy = SimpleStringGenerator(SEX(rand.nextInt(2)))
                ctx.collect(SimpleStringGenerator.NAME(i) + "," + age + "," + sexy)
                Thread.sleep(1000)
            }
        }

        override def cancel(): Unit = {
            running = false
        }
    }
}
```

2. Generate Table1 and Table2, use **Join** to query both Table1 and Table2, and print the output.

```

object SqJoinWithSocket {
    def main(args: Array[String]): Unit = {
        var hostname: String = null
        var port = 0
        System.out.println("use command as: ")
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxxx.xxx.xxx.xxx:21005 --hostname xxxx.xxx.xxx.xxx --
port xxx")
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka--hostname xxxx.xxx.xxx.xxx --port xxx")

        System.out.println("*****")
        System.out.println("<topic> is the kafka topic name")
        System.out.println("<bootstrap.servers> is the ip:port list of brokers")
        System.out.println("*****")
        try {
            val params = ParameterTool.fromArgs(args)
            hostname = if (params.has("hostname")) params.get("hostname")
            else "localhost"
            port = params.getInt("port")
        } catch {
            case e: Exception =>
                System.err.println("No port specified. Please run 'FlinkStreamSqlJoinExample " + "--hostname
<hostname> --port <port>', where hostname (localhost by default) " + "and port is the address of the
text server")
                System.err.println("To start a simple text server, run 'netcat -l -p <port>' and " + "type the input
text into the command line")
                return
        }

        val fsSettings = EnvironmentSettings.newInstance.inStreamingMode.build
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        val tableEnv = StreamTableEnvironment.create(env, fsSettings)

        env.getConfig.setAutoWatermarkInterval(200)
        env.setParallelism(1)
        val paraTool = ParameterTool.fromArgs(args)

        val kafkaStream = env.addSource(new FlinkKafkaConsumer[String](paraTool.get("topic"), new
SimpleStringSchema, paraTool.getProperties)).map(new MapFunction[String, Tuple3[String, String,
String]]()) {
            @throws[Exception]
            override def map(str: String): Tuple3[String, String, String] = {
                val word = str.split(",")
                new Tuple3[String, String, String](word(0), word(1), word(2))
            }
        })

        tableEnv.createTemporaryView("Table1", kafkaStream, $("name"), $("age"), $("sexy"), $(
"proctime").proctime)

        val socketStream = env.socketTextStream(hostname, port, "\n").map(new MapFunction[String,
Tuple2[String, String]]()) {
            @throws[Exception]
            override def map(str: String): Tuple2[String, String] = {
                val words = str.split("\\s")
                if (words.length < 2) return new Tuple2[String, String]
                new Tuple2[String, String](words(0), words(1))
            }
        })

        tableEnv.createTemporaryView("Table2", socketStream, $("name"), $("job"), $(
"proctime").proctime)

        val result = tableEnv.sqlQuery("SELECT t1.name, t1.age, t1.sex, t2.job, t2.proctime as shiptime\n"
+ "FROM Table1 AS t1\n" + "JOIN Table2 AS t2\n" + "ON t1.name = t2.name\n" + "AND t1.proctime

```

```
BETWEEN t2.proctime - INTERVAL '1' SECOND AND t2.proctime + INTERVAL" + " '1' SECOND")  
  
    tableEnv.toAppendStream(result, classOf[Row]).print  
    env.execute  
}  
  
}
```

### 1.8.3.7 Submitting a SQL Job Using Flink Jar

#### 1.8.3.7.1 Scenario Description

##### Description

If SQL statements of a job are frequently modified, submit Flink SQL statements in Flink Jar mode to reduce your workload.

##### Development Guideline

Use the current sample to submit and execute specified SQL statements. Use semicolons (;) to separate multiple statements.

#### 1.8.3.7.2 Java Sample Code

The core logic for submitting SQL statements is as follows. Currently, only **CREATE** and **INSERT** statements can be submitted. For details about the complete code, see com.huawei.bigdata.flink.examples.FlinkSQLExecutor.

```
public class FlinkSQLExecutor {  
    public static void main(String[] args) throws IOException {  
        System.out.println("----- begin init -----");  
        final String sqlPath = ParameterTool.fromArgs(args).get("sql", "config/redisSink.sql");  
        final StreamExecutionEnvironment streamEnv =  
        StreamExecutionEnvironment.getExecutionEnvironment();  
        EnvironmentSettings bsSettings = EnvironmentSettings.newInstance().inStreamingMode().build();  
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(streamEnv, bsSettings);  
        StatementSet statementSet = tableEnv.createStatementSet();  
        String sqlStr = FileUtils.readFileToString(FileUtils.getFile(sqlPath), "utf-8");  
        String[] sqlArr = sqlStr.split(";");  
        for (String sql : sqlArr) {  
            sql = sql.trim();  
            if (sql.toLowerCase(Locale.ROOT).startsWith("create")) {  
                System.out.println("-----\nexecuteSql=\n" + sql);  
                tableEnv.executeSql(sql);  
            } else if (sql.toLowerCase(Locale.ROOT).startsWith("insert")) {  
                System.out.println("-----\ninser=\n" + sql);  
                statementSet.addInsertSql(sql);  
            }  
        }  
        System.out.println("----- begin exec sql -----");  
        statementSet.execute();  
    }  
}
```



##### NOTE

Copy the dependency package required by the current sample, that is, the JAR package in the **lib** file after compilation, to the **lib** folder on the client.

The following uses Kafka in a normal cluster as an example to describe how to submit SQL statements:

```
create table kafka_sink
(
    uuid varchar(20),
    name varchar(10),
    age int,
    ts timestamp(3),
    p varchar(20)
) with (
    'connector' = 'kafka',
    'topic' = '/input2',
    'properties.bootstrap.servers' = 'IP address of the Kafka broker instance',
    'properties.group.id' = 'testGroup2',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'json'
);
create TABLE datagen_source
(
    uuid varchar(20),
    name varchar(10),
    age int,
    ts timestamp(3),
    p varchar(20)
) WITH (
    'connector' = 'datagen',
    'rows-per-second' = '1'
);
INSERT INTO kafka_sink
SELECT *
FROM datagen_source;
```

### 1.8.3.8 FlinkServer REST API JavaExample

#### 1.8.3.8.1 Scenario Description

##### Scenario

Call the FlinkServer RESTful API to create tenants.

##### Data Preparation

- Prepare the files required for user authentication. Specifically, log in to FusionInsight Manager and download the **user.keytab** and **krb5.conf** files.
- Prepare information required for creating a tenant, for example, **tenantId** is **92**, **tenantName** is **test92**, and **remark** is **test tenant remark1**.
- If you run this sample program in Windows, add the host names and IP addresses of all nodes where FlinkServer resides to the **C:\Windows\System32\drivers\etc\hosts** file.

##### Development Guideline

1. Configure user authentication information.
2. Log in as a user.
3. Send requests.

#### 1.8.3.8.2 Java Sample Code

##### Description

Call the FlinkServer RESTful API to create tenants.

## Sample Code

```
public class TestCreateTenants {
    public static void main(String[] args) {
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        final String hostName = paraTool.get("hostName"); // Replace hostName in the hosts file with the
actual host name.
        final String keytab = paraTool.get("keytab file path"); // user.keytab file path
        final String krb5 = paraTool.get("krb5 file path"); // krb5.conf file path
        final String principal = paraTool.get("Authentication username"); // Authentication user

        System.setProperty("java.security.krb5.conf", krb5);
        String url = "https://" + hostName + ":28943/flink/v1/tenants";
        String jsonstr = "{" +
            "\n\t\"tenantId\":\"92\"," +
            "\n\t\"tenantName\":\"test92\"," +
            "\n\t\"remark\":\"test tenant remark\"," +
            "\n\t\"updateUser\":\"test_updateUser\"," +
            "\n\t\"createUser\":\"test_createUser\\"" +
            "\n}";

        try {
            LoginClient.getInstance().setConfigure(url, principal, keytab, "");
            LoginClient.getInstance().login();
            System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", true));
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

### 1.8.3.8.3 Accessing Flinkserver RESTful API as a Proxy User

## Function

Call the FlinkServer RESTful API as a proxy user. Use a proxy to access the API as a FlinkServer administrator to obtain common user permissions.

## Sample Code

Assume that the tenant user is **test92**, the tenant ID is **92**, and the user name is **flinkserveradmin** with FlinkServer administrator permissions. The following code is a complete example.

```
public class TestCreateTenants {
    public static void main(String[] args) {
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        final String hostName = paraTool.get("hostName"); // Replace hostName in the hosts file with the
actual host name.
        final String keytab = paraTool.get("keytab"); // Path for storing user.keytab
        final String krb5 = paraTool.get("krb5"); // Path for storing krb5.conf
        final String principal = paraTool.get("principal"); // Authentication user

        System.setProperty("java.security.krb5.conf", krb5);
        String url = "https://" + hostName + ":28943/flink/v1/tenants";
        String jsonstr = "{" +
            "\n\t\"tenantId\":\"92\"," +
            "\n\t\"tenantName\":\"test92\"," +
            "\n\t\"remark\":\"test tenant remark\"," +
            "\n\t\"updateUser\":\"test_updateUser\"," +
            "\n\t\"createUser\":\"test_createUser\\"" +
            "\n}";

        try {
            LoginClient.getInstance().setConfigure(url, principal, keytab, "");
            LoginClient.getInstance().login(); // Log in as the FlinkServer administrator.
        }
```

```
String proxyUrl = "https://" + hostName + ":28943/flink/v1/proxyUserLogin"; // Call the proxy user API  
to obtain the common user token.  
String result = HttpClientUtil.doPost(proxyUrl, "{\n" +  
"\t\"realUser\": \"flinkserveradmin\"\n" +  
"}", "utf-8", true);  
Gson gson = new Gson();  
JsonObject jsonObject = gson.fromJson(result, JsonObject.class);  
String token = jsonObject.get("result").toString();  
token = "hadoop_auth=" + token;  
  
System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", true , token));  
} catch (Exception e) {  
    System.out.println(e);  
}  
}  
}
```

### 1.8.3.9 Flink Reading Data from and Writing Data to HBase

#### 1.8.3.9.1 Scenario Description

##### Typical Scenario Description

Use Flink API jobs to read data from and write data to HBase.

##### Data Preparation

Prepare the HBase configuration file and download the cluster configuration on FusionInsight Manager to obtain the **hbase-site.xml** file.

##### Development Guideline

1. Writes data to HBase:
  - a. Specify the parent directory of the **hbase-site.xml** file. Flink Sink can obtain the HBase connection.
  - b. Use the connection to determine whether a table exists. If it does not, create one.
  - c. Convert received data into Put objects and writes the Put objects to HBase.
2. Reads data from HBase:
  - a. Specify the parent directory of the **hbase-site.xml** file. Flink Source can obtain the HBase connection.
  - b. Use the connection to determine whether a table exists. If it does not, the job fails. In this case, you need to create a table in HBase shell or an upstream job.
  - c. Read data from HBase, converts result data into Row objects, and sends the Row objects to downstream operators.

#### 1.8.3.9.2 Java Sample Code

##### Description

Call Flink APIs to read data from and write data to HBase.

## Sample Code

The following example shows the main logic code of WriteHBase and ReadHBase.

For details about the complete code, see  
[com.huawei.bigdata.flink.examples.WriteHBase](#) and  
[com.huawei.bigdata.flink.examples.ReadHBase](#).

- Main logic code of WriteHBase

```
public static void main(String[] args) throws Exception {
    System.out.println("use command as: ");
    System.out.println(
        ".bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase"
        + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");
    System.out.println(
        "*****");
    System.out.println("<tableName> hbase tableName");
    System.out.println("<confDir> hbase conf dir");
    System.out.println(
        "*****");
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.setParallelism(1);
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    DataStream<Row> messageStream = env.addSource(new SimpleStringGenerator());
    messageStream.addSink(
        new HBaseWriteSink(paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
    env.execute("WriteHBase");
}

private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
    LOG.info("Create HBase configuration.");
    org.apache.hadoop.conf.Configuration hbaseConf =
    HBaseConfigurationUtil.getHBaseConfiguration();
    if (confDir != null) {
        File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
        if (hbaseSite.exists()) {
            LOG.info("Add hbase-site.xml");
            hbaseConf.addResource(new Path(hbaseSite.getPath()));
        }
        File coreSite = new File(confDir + File.separator + "core-site.xml");
        if (coreSite.exists()) {
            LOG.info("Add core-site.xml");
            hbaseConf.addResource(new Path(coreSite.getPath()));
        }
        File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
        if (hdfsSite.exists()) {
            LOG.info("Add hdfs-site.xml");
            hbaseConf.addResource(new Path(hdfsSite.getPath()));
        }
    }
    LOG.info("HBase configuration created successfully.");
    return hbaseConf;
}

private static class HBaseWriteSink extends RichSinkFunction<Row> {
    private Connection conn;
    private BufferedMutator bufferedMutator;
    private String tableName;
    private final byte[] serializedConfig;
    private Admin admin;
    private org.apache.hadoop.conf.Configuration hbaseConf;
    private long flushTimeIntervalMillis = 5000; //5s
    private long preFlushTime;

    public HBaseWriteSink(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
        this.tableName = sourceTable;
        this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
    }
}
```

```
private void deserializeConfiguration() {
    LOG.info("Deserialize HBase configuration.");
    hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
        serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
    LOG.info("Deserialization successfully.");
}

private void createTable() throws IOException {
    LOG.info("Create HBase Table.");
    if (admin.tableExists(TableName.valueOf(tableName))) {
        LOG.info("Table already exists.");
        return;
    }
    // Specify the table descriptor.
    TableDescriptorBuilder htd =
TableDescriptorBuilder.newBuilder(TableName.valueOf(tableName));
    // Set the column family name to f1.
    ColumnFamilyDescriptorBuilder hcd =
ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("f1"));
    // Set data encoding methods. HBase provides DIFF,FAST_DIFF,PREFIX
    hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
    // Set compression methods, HBase provides two default compression
    // methods:GZ and SNAPPY
    hcd.setCompressionType(Compression.Algorithm.SNAPPY);
    htd.setColumnFamily(hcd.build());
    try {
        admin.createTable(htd.build());
    } catch (IOException e) {
        if (!(e instanceof TableExistsException)
        || !admin.tableExists(TableName.valueOf(tableName))) {
            throw e;
        }
        LOG.info("Table already exists, ignore.");
    }
    LOG.info("Table created successfully.");
}

@Override
public void open(Configuration parameters) throws Exception {
    LOG.info("Write sink open");
    super.open(parameters);
    deserializeConfiguration();
    conn = ConnectionFactory.createConnection(hbaseConf);
    admin = conn.getAdmin();
    createTable();
    bufferedMutator = conn.getBufferedMutator(TableName.valueOf(tableName));
    preFlushTime = System.currentTimeMillis();
}

@Override
public void close() throws Exception {
    LOG.info("Close HBase Connection.");
    try {
        if (admin != null) {
            admin.close();
            admin = null;
        }
        if (bufferedMutator != null) {
            bufferedMutator.close();
            bufferedMutator = null;
        }
        if (conn != null) {
            conn.close();
            conn = null;
        }
    } catch (IOException e) {
        LOG.error("Close HBase Exception:", e);
        throw new RuntimeException(e);
    }
}
```

```

        }
        LOG.info("Close successfully.");
    }

@Override
public void invoke(Row value, Context context) throws Exception {
    LOG.info("Write data to HBase.");
    Put put = new Put(Bytes.toBytes(value.getField(0).toString()));
    put.addColumn(Bytes.toBytes("f1"), Bytes.toBytes("q1"),
    (Bytes.toBytes(value.getField(1).toString())));
    bufferedMutator.mutate(put);

    if (preFlushTime + flushTimeIntervalMillis >= System.currentTimeMillis()) {
        LOG.info("Flush data to HBase.");
        bufferedMutator.flush();
        preFlushTime = System.currentTimeMillis();
        LOG.info("Flush successfully.");
    } else {
        LOG.info("Skip Flush.");
    }

    LOG.info("Write successfully.");
}
}

public static class SimpleStringGenerator implements SourceFunction<Row> {
    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
    long i = 0;
    Random random = new Random();

    @Override
    public void run(SourceContext<Row> ctx) throws Exception {
        while (running) {
            Row row = new Row(2);
            row.setField(0, "rk" + random.nextLong());
            row.setField(1, "v" + random.nextLong());
            ctx.collect(row);
            Thread.sleep(1000);
        }
    }

    @Override
    public void cancel() {
        running = false;
    }
}

```

- Main logic code of ReadHBase

```

public static void main(String[] args) throws Exception {
    System.out.println("use command as:");
    System.out.println(
        "./bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase"
        + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");

    System.out.println(
        "*****");
    System.out.println("<tableName> hbase tableName");
    System.out.println("<confDir> hbase conf dir");
    System.out.println(
        "*****");
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.setParallelism(1);
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    DataStream<Row> messageStream =
        env.addSource(
            new HBaseReaderSource(
                paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
    messageStream
        .rebalance()

```

```
.map(
    new MapFunction<Row, String>() {
        @Override
        public String map(Row s) throws Exception {
            return "Flink says " + s + System.getProperty("line.separator");
        }
    })
.print();
env.execute("ReadHBase");
}

private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
    LOG.info("Create HBase configuration.");
    org.apache.hadoop.conf.Configuration hbaseConf =
    HBaseConfigurationUtil.getHBaseConfiguration();
    if (confDir != null) {
        File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
        if (hbaseSite.exists()) {
            LOG.info("Add hbase-site.xml");
            hbaseConf.addResource(new Path(hbaseSite.getPath()));
        }
        File coreSite = new File(confDir + File.separator + "core-site.xml");
        if (coreSite.exists()) {
            LOG.info("Add core-site.xml");
            hbaseConf.addResource(new Path(coreSite.getPath()));
        }
        File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
        if (hdfsSite.exists()) {
            LOG.info("Add hdfs-site.xml");
            hbaseConf.addResource(new Path(hdfsSite.getPath()));
        }
    }
    LOG.info("HBase configuration created successfully.");
    return hbaseConf;
}

private static class HBaseReaderSource extends RichSourceFunction<Row> {

    private Connection conn;
    private Table table;
    private Scan scan;
    private String tableName;
    private final byte[] serializedConfig;
    private Admin admin;
    private org.apache.hadoop.conf.Configuration hbaseConf;

    public HBaseReaderSource(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
        this.tableName = sourceTable;
        this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
    }

    @Override
    public void open(Configuration parameters) throws Exception {
        LOG.info("Read source open");
        super.open(parameters);
        deserializeConfiguration();
        conn = ConnectionFactory.createConnection(hbaseConf);
        admin = conn.getAdmin();
        if (!admin.tableExists(TableName.valueOf(tableName))) {
            throw new IOException("table does not exist.");
        }
        table = conn.getTable(TableName.valueOf(tableName));
        scan = new Scan();
    }

    private void deserializeConfiguration() {
        LOG.info("Deserialize HBase configuration.");
        hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
            serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
    }
}
```

```
        LOG.info("Deserialization successfully.");
    }

    @Override
    public void run(SourceContext<Row> sourceContext) throws Exception {
        LOG.info("Read source run");
        try (ResultScanner scanner = table.getScanner(scan)) {
            Iterator<Result> iterator = scanner.iterator();
            while (iterator.hasNext()) {
                Result result = iterator.next();
                String rowKey = Bytes.toString(result.getRow());
                byte[] value = result.getValue(Bytes.toBytes("f1"), Bytes.toBytes("q1"));
                Row row = new Row(2);
                row.setField(0, rowKey);
                row.setField(1, Bytes.toString(value));
                sourceContext.collect(row);
                LOG.info("Send data successfully.");
            }
        }
        LOG.info("Read successfully.");
    }

    @Override
    public void close() throws Exception {
        closeHBase();
    }

    private void closeHBase() {
        LOG.info("Close HBase Connection.");
        try {
            if (admin != null) {
                admin.close();
                admin = null;
            }
            if (table != null) {
                table.close();
                table = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (IOException e) {
            LOG.error("Close HBase Exception:", e);
            throw new RuntimeException(e);
        }
        LOG.info("Close successfully.");
    }

    @Override
    public void cancel() {
        closeHBase();
    }
}
```

### 1.8.3.10 Flink Reading Data from and Writing Data to Hudi

#### 1.8.3.10.1 Scenario Description

#### Typical Scenario Description

In this example, the job generates one data record per second, writes the data to the Hudi table, and reads and prints the data in the Hudi table.

## Development Guideline

1. Write data to Hudi:
  - a. Generate data through a random data generation class.
  - b. Convert the generated data into **DataStream<RowData>**.
  - c. Write data to the Hudi table.
2. Read data from Hudi:
  - a. Read data from the Hudi table.
  - b. Combine the read data into the JSON format and print the data.

### 1.8.3.10.2 Java Sample Code

#### Description

Call Flink APIs to read data from and write data to Hudi.

#### Sample Code

The following example shows the main logic code of **WriteIntoHudi** and **ReadFromHudi**.

For details about the complete code, see  
**com.huawei.bigdata.flink.examples.WriteIntoHudi** and  
**com.huawei.bigdata.flink.examples.ReadFromHudi**.

- Main logic code of **WriteIntoHudi**

```
public class WriteIntoHudi {  
    public static void main(String[] args) throws Exception {  
        System.out.println("use command as: ");  
        System.out.println(  
            "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteIntoHudi"  
            + " /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/  
            flinkHudi/hudiTable");  
        System.out.println(  
            "*****");  
        System.out.println("<hudiTableName> is the hudi table name. (Default value is hudiSinkTable)");  
        System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://  
            hacluster/tmp/flinkHudi/hudiTable)");  
        System.out.println(  
            "*****");  
  
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
        env.setParallelism(1);  
        env.getCheckpointConfig().setCheckpointInterval(10000);  
        ParameterTool paraTool = ParameterTool.fromArgs(args);  
        DataStream<RowData> stringDataStreamSource = env.addSource(new SimpleStringGenerator()  
            .map(new MapFunction<Tuple5<String, String, Integer, String, String>, RowData>() {  
                @Override  
                public RowData map(Tuple5<String, String, Integer, String, String> tuple5) throws  
                    Exception {  
                    GenericRowData rowData = new GenericRowData(5);  
                    rowData.setField(0, StringData.fromString(tuple5.f0));  
                    rowData.setField(1, StringData.fromString(tuple5.f1));  
                    rowData.setField(2, tuple5.f2);  
                    rowData.setField(3, TimestampData.fromTimestamp(Timestamp.valueOf(tuple5.f3)));  
                    rowData.setField(4, StringData.fromString(tuple5.f4));  
                    return rowData;  
                }  
            }));  
        String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
```

```

String targetTable = paraTool.get("hudiTableName", "hudiSinkTable");
Map<String, String> options = new HashMap<>();
options.put(FlinkOptions.PATH.key(), basePath);
options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
options.put(FlinkOptions.PRECOMBINE_FIELD.key(), "ts");
options.put(FlinkOptions.INDEX_BOOTSTRAP_ENABLED.key(), "true");
HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
    .column("uuid VARCHAR(20)")
    .column("name VARCHAR(10)")
    .column("age INT")
    .column("ts TIMESTAMP(3)")
    .column("p VARCHAR(20)")
    .pk("uuid")
    .partition("p")
    .options(options);
builder.sink(stringDataStreamSource, false); // The second parameter indicating whether the
input data stream is bounded
env.execute("Hudi_Sink");
}
public static class SimpleStringGenerator implements SourceFunction<Tuple5<String, String,
Integer, String, String>> {
    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
    Integer i = 0;

    @Override
    public void run(SourceContext<Tuple5<String, String, Integer, String, String>> ctx) throws
Exception {
        while (running) {
            i++;
            String uuid = "uuid" + i;
            String name = "name" + i;
            Integer age = new Integer(i);
            String ts = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"));
            String p = "par" + i % 5;
            Tuple5<String, String, Integer, String, String> tuple5 = Tuple5.of(uuid, name, age, ts, p);
            ctx.collect(tuple5);
            Thread.sleep(1000);
        }
    }
    @Override
    public void cancel() {
        running = false;
    }
}
}

```

- Main logic code of ReadFromHudi

```

public class ReadFromHudi {
    public static void main(String[] args) throws Exception {
        System.out.println("use command as:");
        System.out.println(
            "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadFromHudi"
            + " /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/"
            + " --read.start-commit 20221206111532"
        );
        System.out.println(
            "*****"
        );
        System.out.println("<hudiTableName> is the hoodie table name. (Default value is"
            + " hudiSourceTable)");
        System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://"
            + " hacluster/tmp/flinkHudi/hudiTable)");
        System.out.println("<read.start-commit> Start commit instant for reading, the commit time"
            + " format should be 'yyyyMMddHHmmss'. (Default value is earliest)");
        System.out.println(
            "*****"
        );
    }
}
ParameterTool paraTool = ParameterTool.fromArgs(args);

```

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
String targetTable = paraTool.get("hudiTableName", "hudiSourceTable");
String startCommit = paraTool.get(FlinkOptions.READ_START_COMMIT.key(),
    FlinkOptions.START_COMMIT_EARLIEST);
Map<String, String> options = new HashMap();
options.put(FlinkOptions.PATH.key(), basePath);
options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
options.put(FlinkOptions.READ_AS_STREAMING.key(), "true"); // This option enables streaming
read.
options.put(FlinkOptions.READ_START_COMMIT.key(), startCommit); // specifies the start
commit instant time
HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
    .column("uuid VARCHAR(20)")
    .column("name VARCHAR(10)")
    .column("age INT")
    .column("ts TIMESTAMP(3)")
    .column("p VARCHAR(20)")
    .pk("uuid")
    .partition("p")
    .options(options);

DataStream<RowData> rowDataDataStream = builder.source(env);
rowDataDataStream.map(new MapFunction<RowData, String>() {
    @Override
    public String map(RowData rowData) throws Exception {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        sb.append("\"uuid\":\"").append(rowData.getString(0)).append("\",");
        sb.append("\"name\":\"").append(rowData.getString(1)).append("\",");
        sb.append("\"age\":").append(rowData.getInt(2)).append(",");
        sb.append("\"ts\":\"").append(rowData.getTimestamp(3, 0)).append("\",");
        sb.append("\"p\":\"").append(rowData.getString(4)).append("\"");
        sb.append("}");
        return sb.toString();
    }
}).print();
env.execute("Hudi_Source");
}
}
```

### 1.8.3.11 Python Development Examples

#### 1.8.3.11.1 Submitting a Regular Job Using Python

##### Description

Assume that you need to submit a Flink task to an MRS cluster. The main language used by the service platform is Python. The following content uses an example Python program to read and write Kafka jobs.

##### Python Sample Code

##### Function

Submit Flink Kafka read and write jobs to Yarn through Python APIs.

##### Sample Code

The main logic code in **pyflink-kafka.py** is provided. Before submitting the code, ensure that **file\_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see **pyflink-kafka.py** in **flink-examples/pyflink-example/pyflink-kafka**.

```
import os
import logging
import sys
from pyflink.common import JsonRowDeserializationSchema, JsonRowSerializationSchema
from pyflink.common.typeinfo import Types
from pyflink.datastream.connectors import FlinkKafkaProducer, FlinkKafkaConsumer
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import TableEnvironment, EnvironmentSettings
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
    # Change the SQL path before job submission.
    # file_path = "/opt/client/Flink/flink/insertData2kafka.sql"
    # file_path = os.getcwd() + "/../../../../../yarnship/insertData2kafka.sql"
    # file_path = "/opt/client/Flink/flink/conf/ssl/insertData2kafka.sql"
    file_path = "insertData2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("-----create-----")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
            print("-----insert-----")
            print(sqlStr)
            statement_set.add_insert_sql(sqlStr)
    statement_set.execute()
def read_write_kafka():
    # find kafka connector jars
    env = StreamExecutionEnvironment.get_execution_environment()
    env.set_parallelism(1)
    specific_jars = "file:///opt/client/Flink/flink/lib/flink-connector-kafka-xxx.jar"
    # specific_jars = "file://" + os.getcwd() + "/../../../../../yarnship/flink-connector-kafka-xxx.jar"
    # specific_jars = "file:///opt/client/Flink/flink/conf/ssl/flink-connector-kafka-xxx.jar"
    # the sql connector for kafka is used here as it's a fat jar and could avoid dependency issues
    env.add_jars(specific_jars)
    kafka_properties = {'bootstrap.servers': '192.168.20.162:21005', 'group.id': 'test_group'}
    deserialization_schema = JsonRowDeserializationSchema.builder() \
        .type_info(type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_consumer = FlinkKafkaConsumer(
        topics='test_source_topic',
        deserialization_schema=deserialization_schema,
        properties=kafka_properties)
    print("-----read -----")
    ds = env.add_source(kafka_consumer)
    serialization_schema = JsonRowSerializationSchema.builder().with_type_info(
        type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_producer = FlinkKafkaProducer(
        topic='test_sink_topic',
        serialization_schema=serialization_schema,
        producer_config=kafka_properties)
    print("-----write-----")
    ds.add_sink(kafka_producer)
    env.execute("pyflink kafka test")
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format"%(message)s")
    print("-----insert data to kafka-----")
    exec_sql()
    print("-----read_write_kafka-----")
    read_write_kafka()
```

**Table 1-33** Parameters for submitting a regular job using Python

Parameter	Description	Example
bootstrap.servers	Service IP address and port number of the Broker instance of Kafka	192.168.12.25:21005
specific_jars	<p>Package path: <i>Client installation directory/Flink/flink/lib/flink-connector-kafka-*.jar</i>. You are advised to use a full path.</p> <p><b>NOTE</b> If a job needs to be submitted as yarn-application, replace the following path. Replace the JAR package version with the actual one.</p> <pre>specific_jars="file://" + os.getcwd() + "/../../../../../yarnship/flink-connector-kafka-xxx-h0.cbu.mrs.xxx.jar"</pre>	specific_jars = file:///Client installation directory/Flink/flink/lib/flink-connector-kafka-xxx-h0.cbu.mrs.xxx.jar
file_path	<p>Path of the <b>insertData2kafka.sql</b> file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client.</p> <p><b>NOTE</b> If a job needs to be submitted as yarn-application, replace the following path:</p> <pre>file_path = os.getcwd() + "/../../../../../yarnship/insertData2kafka.sql"</pre>	file_path = /Client installation directory/Flink/flink/insertData2kafka.sql

The following is an example SQL statement:

```
create table kafka_sink_table (
    age int,
    name varchar(10)
) with (
    'connector' = 'kafka',
    'topic' = 'test_source_topic', --Name of the topic written to Kafka. Ensure that the topic name is the same as that in the Python file.
    'properties.bootstrap.servers' = 'IP address of the Kafka broker instance.Kafka port number',
    'properties.group.id' = 'test_group',
    'format' = 'json'
);
create TABLE datagen_source_table (
    age int,
    name varchar(10)
) WITH (
    'connector' = 'datagen',
    'rows-per-second' = '1'
);
INSERT INTO
    kafka_sink_table
SELECT
    *
FROM
    datagen_source_table;
```

# Running the Program

- Step 1** Obtain `pyflink-kafka.py` and `insertData2kafka.sql` from the sample project `flink-examples/pyflink-example/pyflink-kafka`.

**Step 2** Package the prepared Python virtual environment by referring to [Preparing Development and Operating Environment](#) and obtain the `venv.zip` file.

```
zip -q -r venv.zip venv/
```

- Step 3** Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-kafka.py**, and **insertData2kafka.sql** files obtained in **Step 1** and **Step 2** to the client environment.

  - Per-job: Upload the preceding files to *Client installation directory/Flink/flink*.
  - yarn-application: Upload the preceding files and the **flink-connector-kafka-Actual version number.jar** package to *Client installation directory/Flink/flink/yarnship*.

- Step 4** Change the `specific_jars` path in `pyflink-kafka.py`.

- per-job: Change the path to the actual path of the SQL file, for example,  
`file:///Client installation directory/Flink/flink/lib/flink-connector-kafka-Actual version number.jar`.
  - yarn-application: Change to `file://" + os.getcwd() + "/../../../../yarnship/flink-connector-kafka-Actual version number.jar`.

- ## **Step 5 Change file\_path in pyflink-kafka.py.**

- per-job: Change the path to the actual path of the SQL file. For example:  
*Client installation directory/Flink/flink/insertData2kafka.sql*
  - yarn-application: Change the path to **os.getcwd () + "../../../../../yarnship/insertData2kafka.sql"**

- Step 6** Run the following command to specify the running environment:

```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

- Step 7** Run the following command to run the program:

- Per-job:  
./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py\_kafka -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-kafka.py

## Execution result:

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Class name contains multiple static handles.

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Actual class found in file /etc/apollo/apollo.yaml

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Actual binding found in file /etc/apollo/apollo.yaml

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Found Yaml parse error at line 103: java.util.concurrent.ConcurrentHashMap<java.lang.String,java.util.List>=new java.util.concurrent.ConcurrentHashMap<java.lang.String,java.util.List>();

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Thread-0 Found the file for the first time passed through the location of the org.apache.ignite.yarn.YarnNodeDescriptorFactory to locate the org.apache.ignite.yarn.YarnNodeDescriptorFactory interface.

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Thread-1 Adding resource type to the org.apache.ignite.yarn.YarnNodeDescriptorFactory interface.

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

The thread-1[1] stuck ready. Thread can not be used because libignite can't be loaded. It's org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Thread-2 Adding dependency to the org.apache.ignite.yarn.YarnNodeDescriptorFactory interface.

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Thread-3 Creating plan for absent org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>(); update:000000100105, requestedValue:>006, masterKey:d59 on ip-10-10-10-10 [1]. org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Attempting to attach YarnNodes security realm.

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

YarnNodes security realm has not been configured to use Kerberos.

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Submitted application application\_100000000000001 on ip-10-10-10-10 [1]. org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Detecting cluster status. Account 100000000000001 is not yet available. org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

2 tasks "v1" are running on cluster ip-10-10-10-10 [1]. org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

A new application with id application\_100000000000001 has been registered. org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

Application application\_100000000000001 has been registered. org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

```
[root@ip-10-10-10-10 ~]# ./checkfile.sh /etc/apollo/apollo.yaml & ./apollo start v=v1 --skip-ssl-validation >/dev/null
```

2 tasks "v1" are running on cluster ip-10-10-10-10 [1]. org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>=new org.apache.ignite.yarn.YarnNodeDescriptorFactory<java.util.List>();

- **yarn-application**  
./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py\_kafka -Dyarn.ship-files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-kafka

## Execution result:

-----End

### 1.8.3.11.2 Submitting a SQL Job Using Python

## Description

Assume that you need to submit Flink tasks to the MRS cluster, the main language used by the service platform is Python, and core service processing requires SQL. The following content provides an example to describe how to submit a SQL job using Python.

# Python Sample Code

## Function

Submit a Flink SQL job to Yarn through Python APIs.

## Sample Code

The main logic code in **pyflink-sql.py** is provided. Before submitting the code, ensure that **file\_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see `pyflink-sql.py` in `flink-examples/pyflink-example/pyflink-sql`.

```
import logging
import sys
import os
from pyflink.table import (EnvironmentSettings, TableEnvironment)
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
    # Change the SQL path before job submission.
    file_path = "datagen2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("-----create-----")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
            print("-----insert-----")
            print(sqlStr)
            statement_set.add_insert_sql(sqlStr)
    statement_set.execute()
```

```
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format"%(message)s")
    exec_sql()
```

**Table 1-34** Parameters for submitting a SQL job using Python

Parameter	Description	Example
file_path	<p>Path of the <b>datagen2kafka.sql</b> file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client.</p> <p><b>NOTE</b> If a job needs to be submitted as yarn-application, replace the following path: <code>file_path = os.getcwd() + "/../../../../../yarnship/datagen2kafka.sql"</code></p>	<code>file_path = /Client installation directory/ Flink/flink/ datagen2kafka.sql</code>

The following is an example SQL statement:

```
create table kafka_sink (
    uuid varchar(20),
    name varchar(10),
    age int,
    ts timestamp(3),
    p varchar(20)
) with (
    'connector' = 'kafka',
    'topic' = 'input2',
    'properties.bootstrap.servers' = 'IP address of the Kafka broker instance.Kafka port number',
    'properties.group.id' = 'testGroup2',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'json'
);
create TABLE datagen_source (
    uuid varchar(20),
    name varchar(10),
    age int,
    ts timestamp(3),
    p varchar(20)
) WITH (
    'connector' = 'datagen',
    'rows-per-second' = '1'
);
INSERT INTO
    kafka_sink
SELECT
    *
FROM
    datagen_source;
```

## Running the Program

**Step 1** Obtain **pyflink-sql.py** and **datagen2kafka.sql** from the sample project **flink-examples/pyflink-example/pyflink-sql**.

**Step 2** Package the prepared Python virtual environment by referring to [Preparing Development and Operating Environment](#) and obtain the **venv.zip** file.

```
zip -q -r venv.zip venv/
```

**Step 3** Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-sql.py**, and **datagen2kafka.sql** files obtained in **Step 1** and **Step 2** to the client environment.

- Per-job: Upload the preceding files to *Client installation directory/Flink/flink*.
- yarn-application: Upload the preceding files to *Client installation directory/Flink/flink/yarnship*.
- yarn-session: Upload the preceding files to *Client installation directory/Flink/flink/conf/ssl*.

**Step 4** Change **file\_path** in **pyflink-sql.py**.

- per-job: Change the path to the actual path of the SQL file. For example: *Client installation directory/Flink/flink/datagen2kafka.sql*
- yarn-application: Change the path to **os.getcwd() + "/../../..../yarnship/datagen2kafka.sql"**
- yarn-session: Change the path to the actual path of the SQL file. For example: *Client installation directory/Flink/flink/conf/ssl//datagen2kafka.sql*

**Step 5** Run the following command to specify the running environment:

```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

**Step 6** Run the following command to run the program:

- Per-job:

```
./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py_sql -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-sql.py
```

Execution result:

```
[2023-07-19 19:41:52.665 [INFO] [main] Login succeeded for user centos using keytab file user.keytab. Keytab file removed in 0.0001 [org.apache.hadoop.security.UserGroupInformation.loginFromKeytabForDelegationToken, java:1120]
[2023-07-19 20:41:58.972 [WARN] [main] No HDFS path for YARN session. Using local file system instead. [org.apache.flink.yarn.YarnClusterDescriptor.deployJobCluster(YarnClusterDescriptor.java:402)
[2023-07-19 20:41:58.972 [INFO] [main] Job Clusters are deprecated since Flink 1.15. Please use of YARN session instead. [org.apache.flink.yarn.YarnClusterDescriptor.deployJobCluster(YarnClusterDescriptor.java:402)
[2023-07-19 19:41:59.146 [INFO] [main] [Thread-7] Adding resource type - name = yarn.io.gpu.units - , type = COMPUTE [org.apache.hadoop.yarn.util.ResourceUtils.getAvailableResourceNameFromConfig(ResourceUtils.java:281)
[2023-07-19 19:41:59.206 [INFO] [main] [Thread-7] Cluster configuration passed. Using local features if available. [org.apache.hadoop.yarn.util.ResourceUtils.getAvailableResourceNameFromConfig(ResourceUtils.java:281)
[2023-07-19 19:42:17.234 [INFO] [main] [Thread-7] Address keytab optMasterKeytab to locate managed HDFS shortcircuit domain. [org.apache.hadoop.yarn.util.ResourceUtils.getAvailableResourceNameFromConfig(ResourceUtils.java:110)
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] Obtaining delegation tokens for HDFS and YARN. [org.apache.flink.yarn.Utils.setDelegationFor(Utils.java:207)
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] User has been authenticated. [org.apache.hadoop.yarn.util.Renewer.setRenewer(Utils.java:101)
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] The flink YARN session cluster has been started in detached mode. In order to stop Flink gracefully, use the following command:
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] kill -9 $(ps aux | grep flink | grep -v grep | awk '{print $2}')
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] If this should not be possible, then you can also kill Flink via YARN's web interface or via:
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] Note that killing Flink might not clean up all job artifacts and temporary files. [org.apache.flink.yarn.YarnClusterDescriptor.logGetAttachedClusterInformation(YarnClusterDescriptor.java:1877)
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] Cluster started. Yarn cluster with application_id application_16000000000000000000000000000000 has been submitted with jobID 220ef088bc09decc870dc7a014.
[2023-07-19 19:42:17.283 [INFO] [main] [Thread-7] Job has been submitted with jobID 220ef088bc09decc870dc7a014]
```

- yarn-application

```
./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py_sql -Dyarn.ship-files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-sql
```

Execution result:

```
[2023-07-20 20:51:31.183 [INFO] [main] Found binding in [jarfile:/opt/client/Flink/flink/lib/logo] sf4; amf: 2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class
[2023-07-20 20:51:31.183 [INFO] [main] Found binding in [jarfile:/opt/client/Flink/flink/lib/protobuf] sf4; amf: 2.17.1.jar!/com/google/protobuf/jspb/StaticLoggerBinder.class
[2023-07-20 20:51:31.183 [INFO] [main] Found binding in [jarfile:/opt/client/Flink/flink/lib/protos] sf4; amf: 2.17.1.jar!/com/google/protobuf/jspb/StaticLoggerBinder.class
[2023-07-20 20:51:31.183 [INFO] [main] Found Yarn properties file under /opt/client/Flink/tmp/yarn-properties-nest. [org.apache.flink.yarn.cli.FlinkYarnSessionCli.<init>:flinkYarnSessionCli.java:200]
[2023-07-20 20:51:31.183 [INFO] [main] [Thread-1] Found Yarn properties file under /opt/client/Flink/tmp/yarn-properties-nest. [org.apache.flink.yarn.cli.FlinkYarnSessionCli.<init>:flinkYarnSessionCli.java:200]
[2023-07-20 20:51:31.183 [INFO] [main] [Thread-1] Path for the file has been passed. Using the location of class org.apache.flink.yarn.YarnClusterDescriptor.startAppMaster(YarnClusterDescriptor.java:1250)
[2023-07-20 20:51:31.183 [INFO] [main] [Thread-1] Submitter application master application_16000000000000000000000000000000 has been submitted. [org.apache.hadoop.yarn.applicationscheduler.SubmitterImpl$submitApplication, java:360]
[2023-07-20 20:51:31.183 [INFO] [main] [Thread-1] Waiting for the cluster to be allocated. [org.apache.flink.yarn.YarnClusterDescriptor.startAppMaster(YarnClusterDescriptor.java:1250)
[2023-07-20 20:51:31.183 [INFO] [main] [Thread-1] The short-circuit local read feature cannot be used because libhadoop cannot be loaded. [org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory.<init>:DomainSocketFactory.java:116]
[2023-07-20 20:51:31.183 [INFO] [main] [Thread-1] Adding delegation tokens to the AM container. [org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory.<init>:DomainSocketFactory.java:111]
[2023-07-20 20:51:33.049 [INFO] [main] [Thread-1] The flink YARN session cluster has been started in detached mode. In order to stop Flink gracefully, use the following command:
[2023-07-20 20:51:33.049 [INFO] [main] [Thread-1] kill -9 $(ps aux | grep flink | grep -v grep | awk '{print $2}')
[2023-07-20 20:51:33.049 [INFO] [main] [Thread-1] If this should not be possible, then you can also kill Flink via YARN's web interface or via:
[2023-07-20 20:51:33.049 [INFO] [main] [Thread-1] Note that killing Flink might not clean up all job artifacts and temporary files. [org.apache.flink.yarn.YarnClusterDescriptor.logGetAttachedClusterInformation(YarnClusterDescriptor.java:1877)
[2023-07-20 20:51:33.049 [INFO] [main] [Thread-1] Cluster started. Yarn cluster with application_id application_16000000000000000000000000000000 has been submitted with jobID 220ef088bc09decc870dc7a014.
[2023-07-20 20:51:33.049 [INFO] [main] [Thread-1] Job has been submitted with jobID 220ef088bc09decc870dc7a014]
```

- yarn-session

Before starting the Yarn session, prepare the running environment by referring to **Preparing Development and Operating Environment**. Run the following command to start **yarn-session**:

```
bin/yarn-session.sh -jm 1024 -tm 4096 -t conf/ssl/ -d
```

Run the following command to submit the job:

```
./bin/flink run --detached -t yarn-session -Dyarn.application.name=py_sql -  
Dyarn.application.id=application_1685505909197_0285 -pyarch conf/ssl/venv.zip -pyexec conf/ssl/  
venv.zip/venv/bin/python3 -py conf/ssl/pyflink-sql.py
```

#### Execution result:

```
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [/opt/client/Flink/lib/taglib-slfa4-impl-2.17.3.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [/opt/client/Flink/lib/taglib-slfa4-impl-2.17.3.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
2023-09-20 21:59:19,212 | INFO | [main] | Found Yarn properties file under /opt/client/Flink/tmp/yarn.properties.rest. | org.apache.flink.yarn.cli.FlinkYarnSessionCli.<init>(FlinkYarnSessionCli.java:299)  
2023-09-20 21:59:19,212 | INFO | [main] | [main] | Found Yarn properties file under /opt/client/Flink/tmp/yarn.properties.rest. | org.apache.flink.yarn.cli.FlinkYarnSessionCli.<init>(FlinkYarnSessionCli.java:299)  
2023-09-20 21:59:33,721 | INFO | [Thread-0] | [main] | No path for the flink jar passed. Using the location of class org.apache.flink.yarn.FrameworkKeyTab[UserGroupInformation.java:111]. | org.apache.flink.yarn.YarnClusterDescriptor.setLocalFlinkJarPath(YarnClusterDescriptor.java:105)  
2023-09-20 21:59:33,721 | INFO | [Thread-0] | [main] | Application application_1685505909197_0285 has been submitted with JobID 63497977da55663593400c59f4d5  
create table test_sink (  
    uid varchar(20),  
    name varchar(10),  
    age int,  
    ts timestamp(3),  
    ts_offset double  
) WITH (  
    format = 'avro'  
) topic = 'input2';  
properties.bootstrap.servers = '192.168.20.162:21005',  
properties.group.id = 'testGroup2',  
scan.startup.mode = 'latest-offset',  
format = 'json';  
-----  
create TABLE datagen_source (  
    uid varchar(20),  
    name varchar(10),  
    age int,  
    ts timestamp(3),  
    ts_offset double  
) WITH (  
    format = 'avro',  
    rows_per_second = '1'  
);  
-----  
Insert  
-----  
INSERT INTO  
    test_sink  
SELECT  
    *  
FROM  
    datagen_source
```

----End

## 1.8.4 Debugging the Application

### 1.8.4.1 Compiling and Running the Application

#### Scenarios

After developing the application code, you can compile the code into a JAR package and upload it to the Linux client for running. The procedures for running applications developed using Scala or Java are the same on the Flink client.

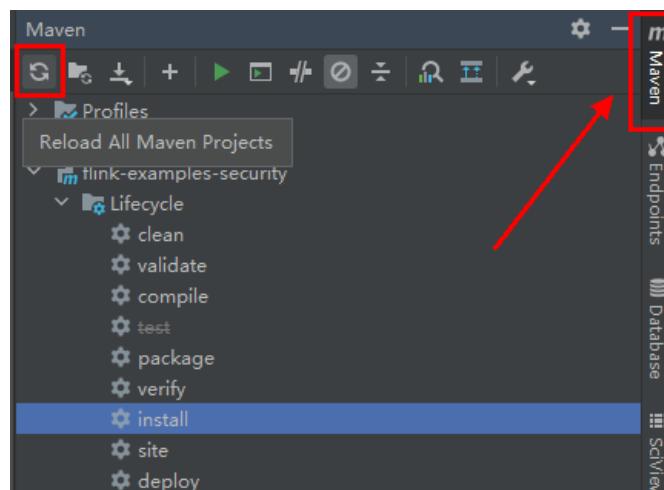


Flink applications of a Yarn cluster can run only on Linux, but not on Windows.

#### Procedure

- Step 1** In IntelliJ IDEA, click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependencies.

Figure 1-76 Reload projects

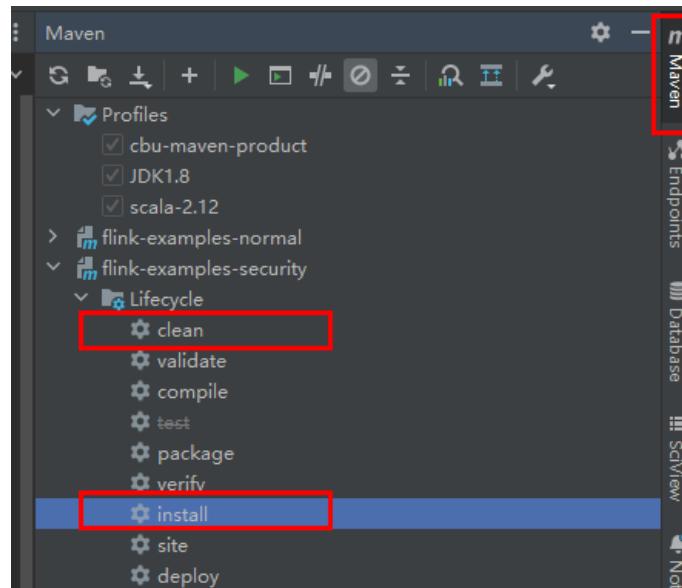


**Step 2** Compile and run the application.

Use either of the following two methods:

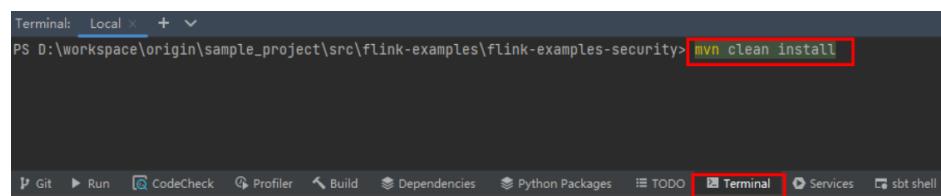
- Method 1:
  - a. Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.
  - b. Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

**Figure 1-77** Maven clean and install



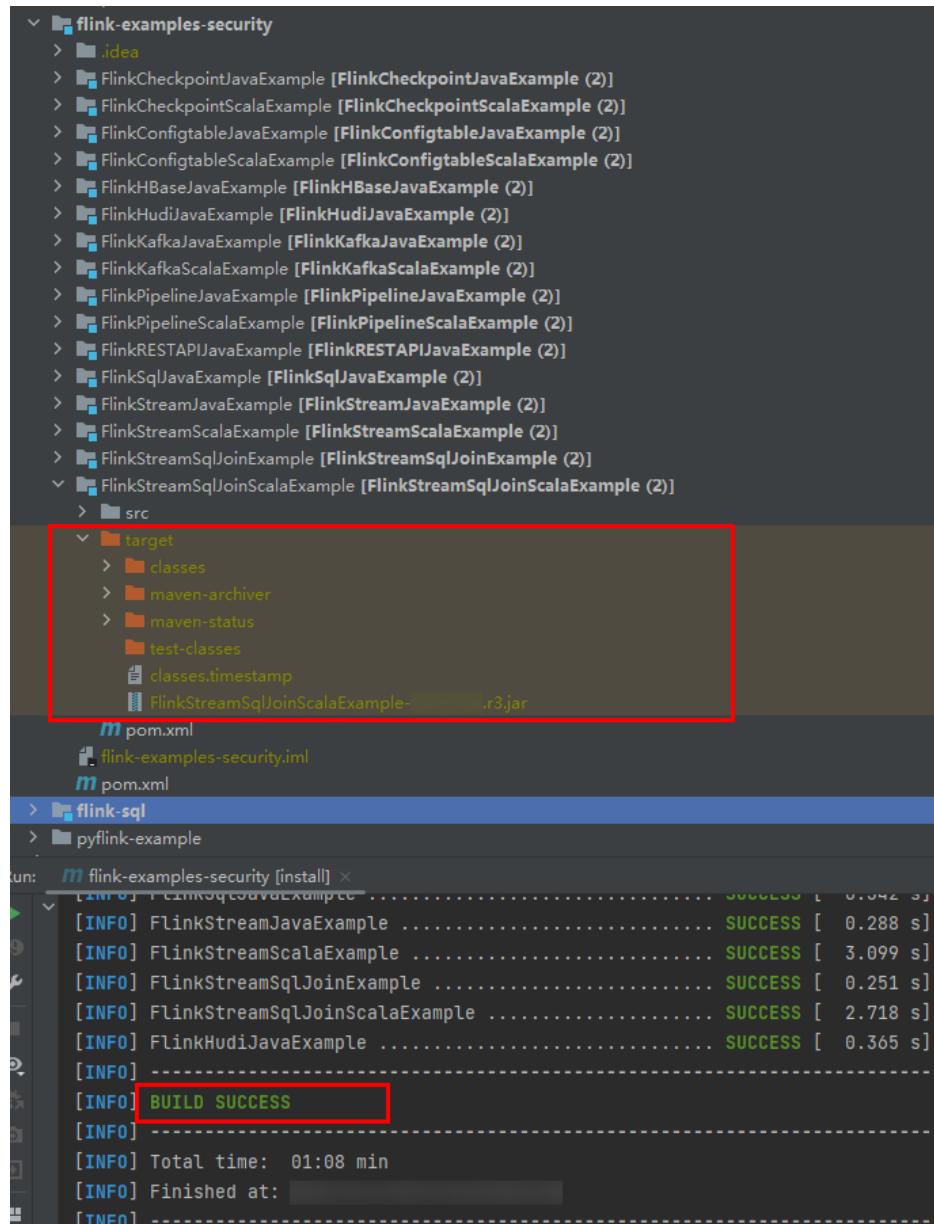
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

**Figure 1-78** Entering **mvn clean install** in the IDEA Terminal text box



**Step 3** After the compilation is complete, the message "BUILD SUCCESS" is printed and the **target** directory is generated. Obtain the JAR package in the directory.

Figure 1-79 Compilation completed



**Step 4** Copy the JAR package (for example, **FlinkStreamJavaExample.jar**) generated in **Step 3** to the related directory of the Flink client node, for example, **/opt/hadoopclient**. Create the **conf** directory in the directory and copy the required configuration file to the **conf** directory. For details, see [Preparing an Operating Environment](#). Run the Flink application.

Start the Flink cluster before running the Flink applications on Linux. On the Flink client, run the **yarn session** command to start the Flink cluster.

The following is an example:

```
cd /opt/hadoopclient/Flink/flink
bin/yarn-session.sh -jm 1024 -tm 4096 -t conf/ssl/
```

 NOTE

- Before running the **yarn-session.sh** command, copy the dependency package of the Flink application to the client directory `#{Client installation directory}/Flink/flink/lib`. For details about the dependency packages of the application, see [Reference information about the dependency package for running the sample project](#).
  - Do not restart the HDFS service or all DataNode instances during Flink job running. Otherwise, the job may fail and some temporary application data cannot be cleared.
  - In the example, **ssl/** is a user-defined subdirectory in the Flink client directory, which is used to store configuration files of the SSL keystore and truststore.
  - The memory size of TaskManagers specified by using the **-tm** command must be at least 4,096 MB.
- Running the DataStream sample application (in Scala or Java)
- Open another window on the terminal. Go to the Flink client directory and call the **bin/flink run** script to run code.
- Java  
**bin/flink run --class com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/hadoopclient/FlinkStreamJavaExample.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2**
  - Scala  
**bin/flink run --class com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/hadoopclient/FlinkStreamScalaExample.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2**

 NOTE

If **log1.txt** and **log2.txt** need to be stored locally, they must be stored on each node where the Yarn NodeManager instance is deployed. The permission is 755.

**Table 1-35** Parameters

Parameter	Description
<code>&lt;filePath&gt;</code>	File path in the local file system. The <code>/opt/log1.txt</code> and <code>/opt/log2.txt</code> files must be stored on every node. The default value can be retained or changed.
<code>&lt;windowTime&gt;</code>	Duration of a time window. The unit is minute. The default value can be retained or changed.

- Running the sample application for producing and consuming data in Kafka (in Java or Scala)

Command for starting the application to produce data

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/hadoopclient/  
FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sasl.kerberos.service.name]  
[ssl.truststore.location] [ssl.truststore.password] [kerberos.domain.name]
```

Command for starting the application to consume data

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/  
FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sasl.kerberos.service.name]  
[ssl.truststore.location] [ssl.truststore.password]
```

**Table 1-36** Parameters

Parameter	Description	Mandatory or Not
topic	Kafka topic name	Yes
bootstrap.server	List of IP addresses or ports of broker clusters	Yes

Parameter	Description	Mandatory or Not
security.protocol	<p>The parameter can be set to protocols PLAINTEXT (optional), SASL_PLAINTEXT, SSL, and SASL_SSL, corresponding to ports 21005, 21007, 21008, and 21009 of the FusionInsight Kafka cluster, respectively.</p> <ul style="list-style-type: none"> <li>- If the SASL is configured, the value of <b>sasl.kerberos.service.name</b> must be set to <b>kafka</b> and that of <b>kerberos.domain.name</b> must be set to <b>hadoop.SystemDomain name</b>, and the configuration items related to <b>security.kerberos.login</b> in <b>conf/flink-conf.yaml</b> must be set.</li> </ul> <p><b>NOTE</b>  Log in to FusionInsight Manager and choose <b>System &gt; Permission &gt; Domain and Mutual Trust</b>. On the displayed page, the value of <b>Local Domain</b> is the system domain name. (All letters in the system domain name must be converted into lowercase letters.)</p> <ul style="list-style-type: none"> <li>- If the SSL is configured, <b>ssl.truststore.location</b> (path of <b>truststore</b>) and <b>ssl.truststore.password</b> (password of <b>truststore</b>) must be set.</li> </ul>	<p>No</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>- If this parameter is not configured, Kafka is in non-security mode.</li> <li>- If SSL needs to be configured, find more information about how to generate the <b>truststore.jks</b> file in <b>More Information &gt; External Interfaces &gt; SSL Encryption Function Used by a Client of Kafka Development Guide</b>.</li> </ul>

 NOTE

- To execute the sample application, set **allow.everyone.if.no.acl.found** to **true**.
- The following **.jar** packages need to be added to Kafka applications:
  - **flink-dist\_\*.jar** in the **lib** directory under the installation directory of Flink server.
  - **flink-connector-kafka\_\*.jar** in the **opt** directory under the installation directory of Flink server.
  - **kafka-clients-\*.jar** in the **lib** directory under the installation directory of Kafka client or server.
  - If **truststore.jks** is set to an absolute path, place the **truststore.jks** file in the specified directory of each Yarn nodemanager. If this parameter is set to a relative path, add the upload directory when executing the **yarn-session.sh** script. For example, before executing **Command 4**, run the **yarn-session.sh -t config/ \*\*\*** command.

The following commands use ReadFromKafka as an example and the cluster domain name is **HADOOP.COM**:

- Command 1:  

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21005
```
- Command 2:  

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21007 --
security.protocol SASL_PLAINTEXT --sasl.kerberos.service.name kafka --kerberos.domain.name
hadoop.hadoop.com
```
- Command 3:  

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/conf/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21008 --
security.protocol SSL --ssl.truststore.location /home/truststore.jks --ssl.truststore.password xxx
```
- Command 4:  

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21009 --
security.protocol SASL_SSL --sasl.kerberos.service.name kafka --ssl.truststore.location config/
truststore.jks --ssl.truststore.password xxx --kerberos.domain.name hadoop.hadoop.com
```
- Running the sample application of the asynchronous checkpoint mechanism (in Scala or Java)

To diversify sample code, the processing time is used as a timestamp for data stream in Java, and the event time is used as a timestamp for data stream in Scala. The command reference is as follows:

- Saving checkpoint snapshot information to HDFS
  - Java  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkProcessingTimeAPIMain /opt/
hadoopclient/FlinkCheckpointJavaExample.jar --chkPath hdfs://hacluster/flink/checkpoint/
```
  - Scala  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkEventTimeAPIChkMain /opt/
hadoopclient/conf/FlinkCheckpointScalaExample.jar --chkPath hdfs://hacluster/flink/
checkpoint/
```
- Saving checkpoint snapshot information to a local file
  - Java  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkProcessingTimeAPIMain /opt/
hadoopclient/FlinkCheckpointJavaExample.jar --chkPath file:///home/zzz/flink-checkpoint/
```

■ Scala

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkEventTimeAPIChkMain  
/opt/hadoopclient/FlinkCheckpointScalaExample.jar --ckkPath file:///home/zzz/flink-  
checkpoint/
```

 NOTE

- Checkpoint source file path: **flink/checkpoint/fd5f5b3d08628d83038a30302b611/chk-X/4f854bf4-ea54-4595-a9d9-9b9080779ffe**

**flink/checkpoint:** indicates the specified root directory.

**fd5f5b3d08628d83038a30302b611:** indicates the level-2 directory named after jobID.

**chk-X:** "X" indicates the checkpoint number, which is the level-3 directory.

**4f854bf4-ea54-4595-a9d9-9b9080779ffe:** indicates a checkpoint source file.

- If Flink is in cluster mode, the checkpoint stores the file in HDFS. A local path can only be used when Flink is in local mode, facilitating commissioning.

● Running the Pipeline sample application

- Java

- i. Start the publisher job.

```
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipeline_NettySink /opt/  
hadoopclient/FlinkPipelineJavaExample.jar
```

- ii. Start the subscriber Job1.

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource1 /opt/  
hadoopclient/FlinkPipelineJavaExample.jar
```

- iii. Start the subscriber Job2.

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource2 /opt/  
hadoopclient/FlinkPipelineJavaExample.jar
```

- Scala

- i. Start the publisher job.

```
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipeline_NettySink /opt/  
hadoopclient/FlinkPipelineScalaExample.jar
```

- ii. Start the subscriber Job1.

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource1 /opt/  
hadoopclient/FlinkPipelineScalaExample.jar
```

- iii. Start the subscriber Job2.

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource2 /opt/  
hadoopclient/FlinkPipelineScalaExample.jar
```

● Sample application of the JOIN between configuration tables and streams.  
FlinkConfigtableJavaExample and FlinkConfigtableScalaExample are used as examples.

The execution modes of Java are as follows:

- yarn-session mode

- i. Import **configtable.csv** to the Redis cluster.

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/  
FlinkConfigtableJavaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --  
configPath config/import.properties
```

- ii. Start the Flink cluster.

```
bin/yarn-session.sh -t config -jm 1024 -tm 1024
```

- iii. Stream data reads personal information from Redis using netizen names as keywords, joins the information, and generates the output.

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableJavaExample /opt/  
hadoopclient/FlinkConfigtableJavaExample.jar --dataPath config/data.txt
```

- yarn-cluster mode
  - i. Import **configtable.csv** to the Redis cluster.  

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/  
FlinkConfigtableJavaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --  
configPath config/import.properties
```
  - ii. Start the Flink cluster. Stream data reads personal information from  
Redis using netizen names as keywords, joins the information, and  
generates the output.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableJavaExample -m  
yarn-cluster -yt config -yjm 1024 -ytm 4096 /opt//opt/hadoopclient/  
FlinkConfigtableJavaExample.jar --dataPath config/data.txt
```

The execution modes of Scala are as follows:

- yarn-session mode
  - i. Import **configtable.csv** to the Redis cluster.  

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/  
FlinkConfigtableScalaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --  
configPath config/import.properties
```
  - ii. Start the Flink cluster.  

```
bin/yarn-session.sh -t config -jm 1024 -tm 4096
```
  - iii. Stream data reads personal information from Redis using netizen  
names as keywords, joins the information, and generates the output.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableScalaExample /opt/  
hadoopclient/FlinkConfigtableScalaExample.jar --dataPath config/data.txt
```
- yarn-cluster mode
  - i. Import **configtable.csv** to the Redis cluster.  

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/  
FlinkConfigtableScalaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --  
configPath config/import.properties
```
  - ii. Start the Flink cluster. Stream data reads personal information from  
Redis using netizen names as keywords, joins the information, and  
generates the output.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableScalaExample -m  
yarn-cluster -yt config -yjm 1024 -ytm 4096 /opt/hadoopclient/  
FlinkConfigtableScalaExample.jar --dataPath config/data.txt
```

- Running the Stream SQL Join sample application

- Java
  - i. Start the application to produce data in Kafka. For details about how  
to configure Kafka, see [Running the sample application for  
producing and consuming data in Kafka \(in Java or Scala\)](#).  

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/hadoopclient/  
FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.21005
```
  - ii. Run the **netcat** command on any node in the cluster to wait for an  
application connection.  

```
netcat -l -p 9000
```
  - iii. Start the application to receive socket data and perform a joint  
query.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/  
hadoopclient/FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers  
xxx.xxx.xxx.21005 --hostname xxx.xxx.xxx.21005 --port 9000
```
- Scala
  - i. Start the application to produce data in Kafka. For details about how  
to configure Kafka, see [Running the sample application for  
producing and consuming data in Kafka \(in Java or Scala\)](#).

- ```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/hadoopclient/
FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers
xxx.xxx.xxx.21005
```
  - ii. Run the **netcat** command on any node in the cluster to wait for an application connection.  

```
netcat -l -p 9000
```
  - iii. Start the application to receive socket data and perform a joint query.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/
hadoopclient/FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers
xxx.xxx.xxx.21005 --hostname xxx.xxx.xxx.9000 --port 9000
```
- Running the Flink HBase sample application .
    - yarn-session mode
      - i. Start the Flink cluster.  

```
./bin/yarn-session.sh -t config -jm 1024 -tm 4096
```
      - ii. Run the Flink application and enter parameters.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/Flink/flink/
FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/Flink/flink/
FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```
    - yarn-cluster mode
      - ```
bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/
Flink/flink/FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```
      - ```
bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/
Flink/flink/FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```
  - Running the Flink Hudi sample application .
    - yarn-session mode
      - i. Start the Flink cluster.  

```
./bin/yarn-session.sh -t config -jm 1024 -tm 4096
```
      - ii. Run the Flink application and enter parameters.  

```
./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/test.jar --
hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable
```

```
./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromHudi /opt/test.jar --
hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable --
read.start-commit xxx
```
    - yarn-cluster mode
      - ```
./bin/flink run -m yarn-cluster -yt config --class
com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/test.jar --hudiTableName hudiSinkTable --
hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable
```
      - ```
./bin/flink run -m yarn-cluster -yt config --class
com.huawei.bigdata.flink.examples.ReadFromHudi /opt/test.jar --hudiTableName
hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable --read.start-commit xxx
```
  - Running the REST APIs for tenant sample program creation (The TestCreateTenants program is used as an example.)
    - yarn-session mode
      - i. Start the Flink cluster.  

```
./bin/yarn-session.sh -t config -jm 1024 -tm 4096
```
      - ii. Run the Flink program and enter parameters.  

```
./bin/flink run --class com.huawei.bigdata.flink.examples.TestCreateTenants /opt/
hadoopclient/FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx --keytab xx/xx/
user.keytab --krb5 xx/xx/krb5.conf --principal username
```
    - yarn-cluster mode

```
./bin/flink run -m yarn-cluster -yt config -yjm 1024 -ytm 4096 --class com.huawei.bigdata.flink.examples.TestCreateTenants /opt//opt/hadoopclient/FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx --keytab xx/xx/user.keytab --krb5 xx/xx/krb5.conf --principal username
```

- Run a SQL task to submit Flink JAR jobs.
  - yarn-session mode
    - i. Start the Flink cluster.  
./bin/yarn-session.sh -t config -jm 1024 -tm 4096
    - ii. Run the Flink application and enter parameters.  
bin/flink run -d --class com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/datanen2kafka.sql
  - yarn-cluster mode  
bin/flink run -m yarn-cluster -yt config -yjm 1024 -ytm 4096 -d --class com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/datanen2kafka.sql

----End

#### NOTE

For details about sample projects of dependency packages provided by Flink, see [Reference information about the dependency package for running the sample project](#).

If HA is enabled for Flink, the value of **--hostName** in the REST API example for creating a tenant is the floating IP address of FlinkServer.

### 1.8.4.2 Viewing the Debugging Result

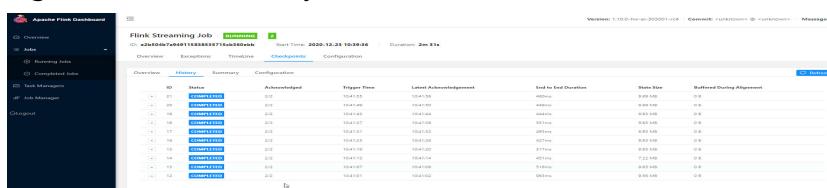
#### Scenarios

After a Flink application completes running, you can view the running result, or use Apache Flink Dashboard to view application running status.

#### Procedure

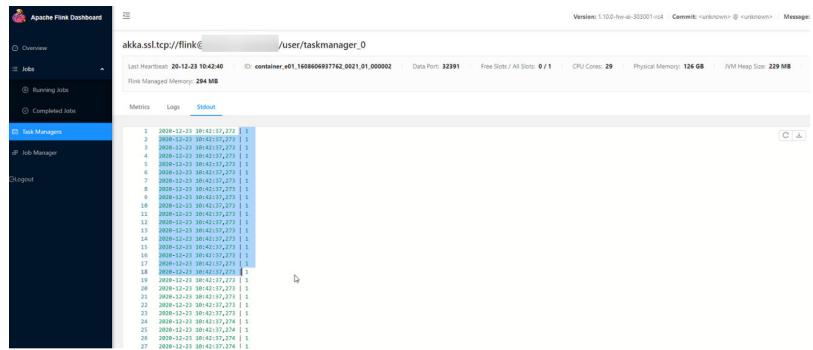
- **View the running result of the Flink application.**
  - If you want to check the execution result, view the Stdout log of Task Manager on the Apache Flink Dashboard.
  - If the execution result is exported to a file or a location specified by Flink, view the result from the exported file or the location. The checkpoint, pipeline, and join between configuration tables and streams are used as examples.
  - **View checkpoint results and files**
    - The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Checkpoints** to view the submitted jobs as shown in [Figure 1-80](#). Choose **Task Managers > Stdout** to view the running result, as shown in [Figure 1-81](#).

Figure 1-80 submitted jobs



| ID | Name | Acknowledged        | Start Time          | Latest Acknowledgement | End-to-End Duration | State Size | Buffered During Acknowledgment |
|----|------|---------------------|---------------------|------------------------|---------------------|------------|--------------------------------|
| 1  | 1    | 2020-12-29 10:39:32 | 2020-12-29 10:41:59 | 2020-12-29 10:41:59    | 460ms               | 0.0B       | 0.0                            |
| 2  | 2    | 2020-12-29 10:41:59 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 1ms                 | 0.0B       | 0.0                            |
| 3  | 3    | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 4  | 4    | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 5  | 5    | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 6  | 6    | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 7  | 7    | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 8  | 8    | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 9  | 9    | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 10 | 10   | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 11 | 11   | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 12 | 12   | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |
| 13 | 13   | 2020-12-29 10:42:00 | 2020-12-29 10:42:00 | 2020-12-29 10:42:00    | 0ms                 | 0.0B       | 0.0                            |

Figure 1-81 execution result



- Either the following methods can be used to view the checkpoint file:
  - If the checkpoint snapshot information is saved in the HDFS, run the **hdfs dfs -ls hdfs://hacluster/flink/checkpoint/** command to view checkpoint files.
  - If the checkpoint snapshot information is saved to a local file, log in to each node to view checkpoint files.

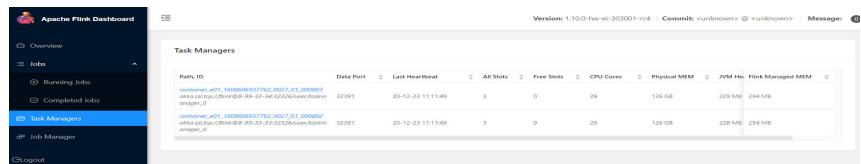
#### - View pipeline results

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in [Figure 1-82](#). Choose **Task Managers**. You can see two tasks, as shown in [Figure 1-83](#). Click any task, choose **Stdout** to view the output of the task, as shown in [Figure 1-84](#) and [Figure 1-85](#).

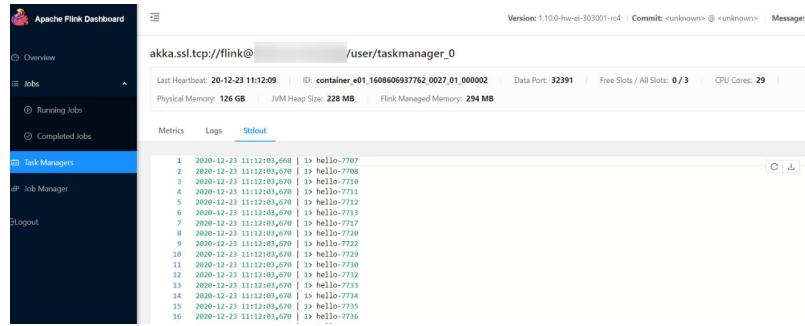
Figure 1-82 running jobs



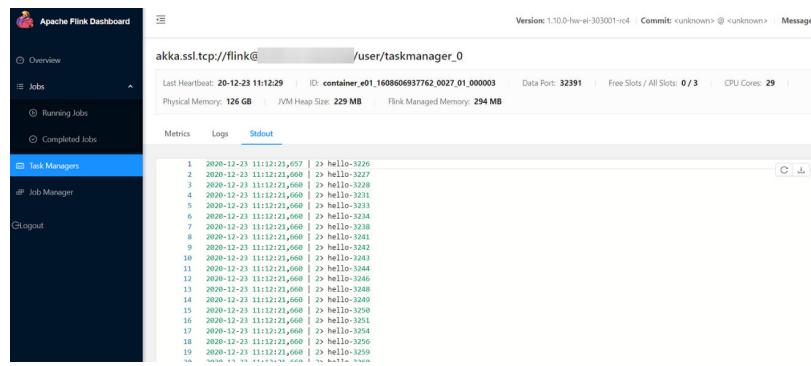
Figure 1-83 submitted Tasks



**Figure 1-84** output of task1



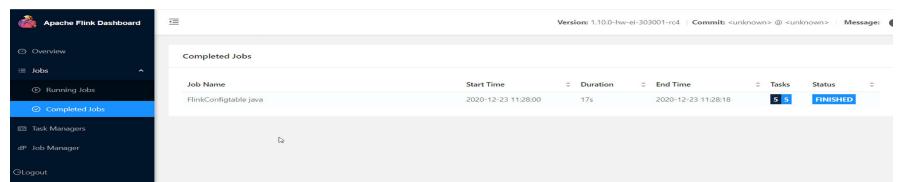
**Figure 1-85** output of task2



- View the JOIN result of configuration table and streams

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Completed Jobs** to view the completed job as shown in **Figure 1-86**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 1-87**. Choose **Stdout** to view the running result, as shown in **Figure 1-88**.

## **Figure 1-86 completed job**



**Figure 1-87** submitted task

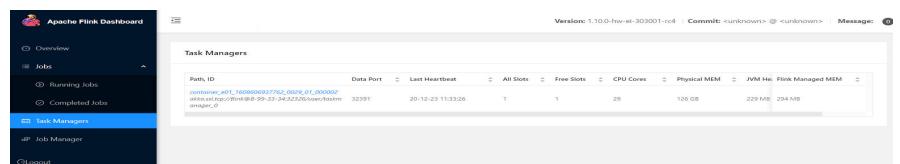
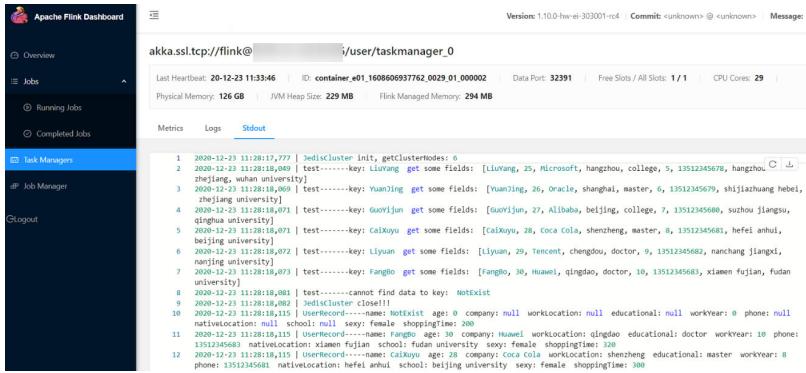


Figure 1-88 execution result



### - View the result of DataStream

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Completed Jobs** to view the completed job as shown in **Figure 1-89**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 1-90**. Choose **Stdout** to view the running result, as shown in **Figure 1-91**.

Figure 1-89 completed job

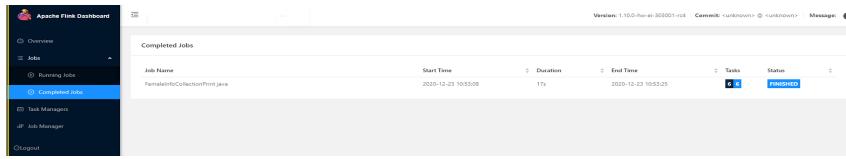


Figure 1-90 submitted task

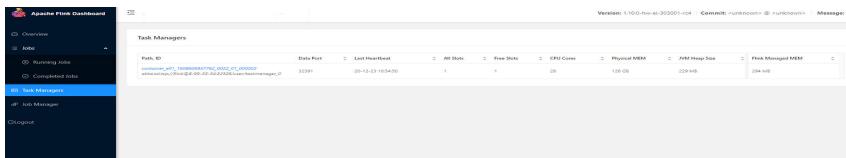
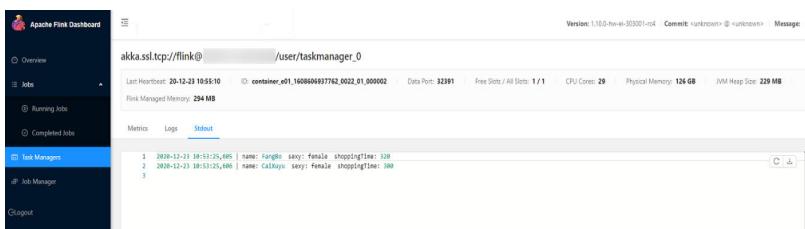


Figure 1-91 execution result



### - View the result of stream sql join

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in **Figure 1-92**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 1-93**. Choose **Stdout** to view the running result, as shown in **Figure 1-94**.

Figure 1-92 running jobs

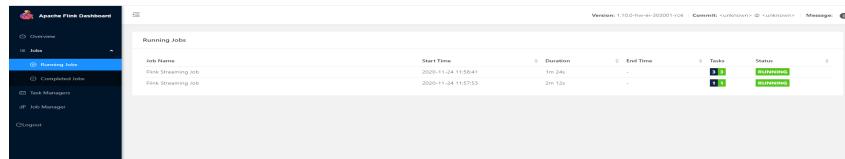


Figure 1-93 submitted task

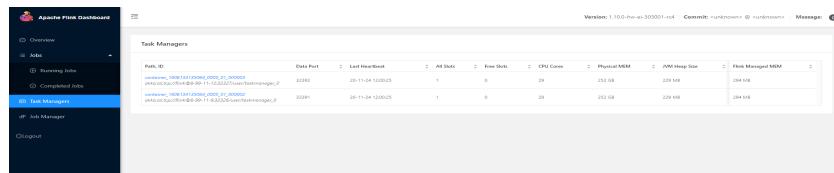
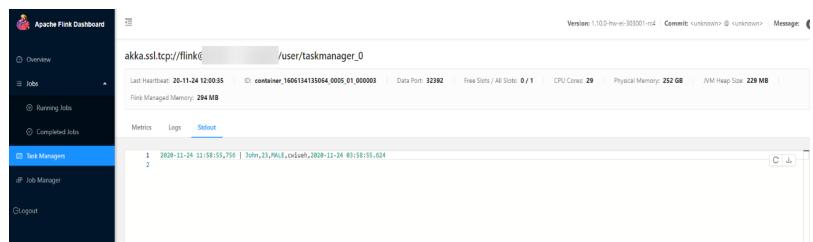


Figure 1-94 execution result



- View the result of produce and consume data in Kafka
- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in [Figure 1-95](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 1-96](#). Choose **Stdout** to view the running result, as shown in [Figure 1-97](#).

Figure 1-95 running jobs

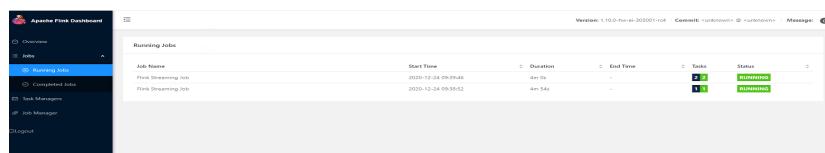


Figure 1-96 submitted task



Figure 1-97 execution result



- Use Apache Flink Dashboard to view the running status of the Flink application.

The Apache Flink Dashboard mainly includes Overview, Running Jobs, Completed Jobs, Task Managers, Job Manager and Logout and so on.

On In the YARN web UI, find the desired Flink application. Click the **ApplicationMaster** at the last column of the application to switch to the Apache Flink Dashboard.

View the print results of the program execution: find the corresponding **Task Manager** to see the corresponding **Stdout** tag log information.

- **View Flink logs.**

Three methods can be used to obtain Flink logs:

- Log in to the Apache Flink Dashboard and view logs of TaskManagers and JobManager.
  - Log in to the YARN web UI to view logs about JobManager and GC.

On the YARN web UI wind, find the desired Flink application. Click the **ID** of the application. On the switched page, click **Logs** in the Logs column.

- On the Yarn client, obtain or view logs of Task Managers and Job Manager.

i. Download and install the Yarn client, for example, in the /opt/hadoopclient directory.

ii. Log in to the node where the client is installed as the client installation user.

iii. Run the following command to switch to the client installation directory:

**cd /opt/hadoopclient**

iv. Run the following command to configure environment variables:

**source bigdata\_env**

v. If the cluster employs the security mode, run the following command to authenticate the user. If the normal mode is used, skip this step.

**kinit component service user**

vi. Run the following commands to obtain container information of the Flink cluster:

**yarn logs -applicationId application\_\* -show\_application\_log\_info**

```
[redacted]:/opt/flinkclient/Flink/flink # yarn logs -applicationId application_1547547065745_0001 -show_application_log_info
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
Application State: Running.
Container: container_1547547065745_0001_01_000001 on [redacted]:26009
Container: container_1547547065745_0001_01_000002 on [redacted]:26009
Container: container_1547547065745_0001_01_000003 on [redacted]:26009
Container: container_1547547065745_0001_01_000004 on [redacted]:26009
```

vii. Run the following command to obtain run logs of the specified container. Generally, container\_\*\_000001 is the container where the Job Manager is running.

**yarn logs -applicationId application\_\* --containerId container\_1547547065745\_0001\_01\_000004 -out logdir/**

```
[redacted]:/opt/flinkclient/Flink/flink/logdir/[redacted]_0001_01_000004_26009 # 11
total 172
-rw-r--r-- 1 root root 170605 Jan 17 10:24 container_1547547065745_0001_01_000004
[redacted]:/opt/flinkclient/Flink/flink/logdir/[redacted]_0001_01_000004_26009 #
```

After the command is executed, container run logs, including run logs of the Task Manager and Job Manager and GC logs, are downloaded to the local host.

- viii. You can also run a command to obtain the log with the specified name:

The following command is used to obtain the container log list.

```
yarn logs -applicationId application_* -show_container_log_info --containerId container_1547547065745_0001_01_000004
```

| LogFile                    | LogLength | LastModificationTime           | LogAggregationType |
|----------------------------|-----------|--------------------------------|--------------------|
| container-localizer-syslog | 184       | Wed Jan 16 17:49:27 +0800 2019 | LOCAL              |
| taskmanager.log            | 131430    | Wed Jan 16 17:49:35 +0800 2019 | LOCAL              |
| tasklog.0.com              | 17950     | Wed Jan 16 17:49:35 +0800 2019 | LOCAL              |
| taskmanager.out            | 0         | Wed Jan 16 17:49:31 +0800 2019 | LOCAL              |
| launch_container.sh        | 12232     | Wed Jan 16 17:49:31 +0800 2019 | LOCAL              |
| directory.info             | 3661      | Wed Jan 16 17:49:31 +0800 2019 | LOCAL              |
| taskmanager.out            | 1060      | Wed Jan 16 17:49:31 +0800 2019 | LOCAL              |
| prelauch.out               | 160       | Wed Jan 16 17:49:31 +0800 2019 | LOCAL              |
| prelauch.err               | 0         | Wed Jan 16 17:49:31 +0800 2019 | LOCAL              |

Download **taskmanager.log** to the local host.

```
yarn logs -applicationId application_* --containerId
container_1547547065745_0001_01_000004 -log_files
taskmanager.log -out localpath
```

### 1.8.4.3 Running a Spring Boot Sample Project and Viewing Results

#### Running the Spring Boot Sample in CLI

**Step 1** Use Maven to run **install** on the IDEA.

If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR file containing the **flink-dws-sink-example-1.0.0-SNAPSHOT** field is generated in the **target** directory of the sample project.

```
[INFO] Dependency-reduced POM written at: D:\code\springsample_project\src\main\flink-examples\flink-dws-sink-example\dependency-reduced-pom.xml
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ flink-dws-sink-example ---
[INFO] Installing D:\code\springsample_project\src\main\flink-examples\flink-dws-sink-example\target\flink-dws-sink-example.jar to D:\soft\maven\local\org\springframework\boot\flink-dws-sink-example\flink-dws-sink-example-1.0.0-SNAPSHOT.jar
[INFO] Installing D:\code\springsample_project\src\main\flink-examples\flink-dws-sink-example\dependency-reduced-pom.xml to D:\soft\maven\local\org\springframework\boot\flink-dws-sink-example\flink-dws-sink-example-1.0.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 15.581 s
[INFO] Finished at: 2023-08-18T11:39:01+08:00
[INFO] -----
```

Process finished with exit code 0

**Step 2** On Linux, go to the client installation directory, for example, **/opt/client/Flink/flink/conf**, and save the JAR packages whose names contain **flink-dws-sink-example-1.0.0-SNAPSHOT** in the **target** directory generated in to this directory.



The Flink client running the Spring Boot sample must contain only the Flink service.

**Step 3** Run the following command to create a Yarn session:

```
yarn-session.sh -t ssl/ -nm "session-spring11" -d
```

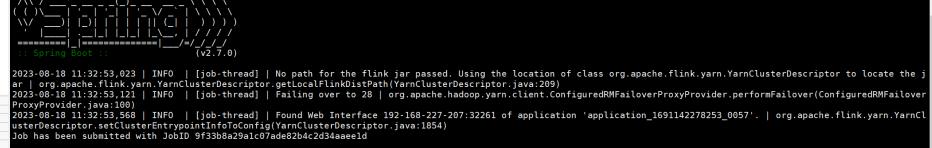
```
[root@host1 conf]#
[root@host1 conf]#
[root@host1 conf]# yarn-session.sh -t ssl/ -nm "session-spring11" -d
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client123/Flink/flink/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client123/HDFS/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar]
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
2023-08-18 09:54:32,938 | INFO | [main] | Loading configuration property: akka.ask.timeout, 300 s | org.apache.logging.slf4j.impl.StaticLoggerBinder
2023-08-18 09:54:32,945 | INFO | [main] | Loading configuration property: akka.client-socket-worker-pool.poolSize | org.apache.logging.slf4j.impl.StaticLoggerBinder
2023-08-18 09:54:32,945 | INFO | [main] | Loading configuration property: akka.client-socket-worker-pool.configuration.loadYAMLResource(GlobalConfiguration.java:224)
2023-08-18 09:54:32,945 | INFO | [main] | Loading configuration property: akka.client-socket-worker-pool.poolSize | org.apache.logging.slf4j.impl.StaticLoggerBinder
2023-08-18 09:54:32,945 | INFO | [main] | Loading configuration property: akka.client-socket-worker-pool.configuration.loadYAMLResource(GlobalConfiguration.java:224)
```

**Step 4** Run the following command to start the SpringBoot service:

- Run the GaussDB(DWS) sample

**flink run flink-dws-sink-example.jar**

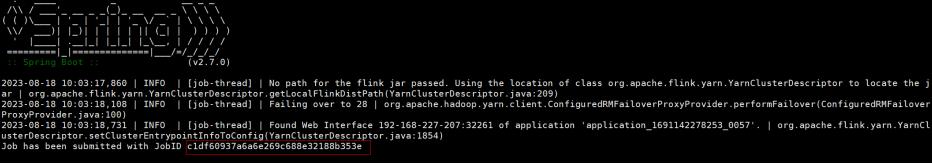
```
Password for test@host001:root:
[root@host01 conf]# flink run flink-dws-sink-example.jar
OpenJDK 64-Bit Server VM warning: Cannot open file <LOG_DIR>/gc.log due to No such file or directory
SLF4J: Class path contains multiple SLF4J bindings
SLF4J: Found binding in [jar:file:/opt/client123/Flink/lib/libflink-log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client123/HDFS/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory].
2023-08-18 11:32:48,153 | INFO | [main] | Found Yarn properties file under /opt/client123/Flink/tmp/.yarn-properties-root. | org.apache.flink.yarn.cli.FlinkYarnSessionCl
i.<init>(FlinkYarnSessionCli.java:293)
2023-08-18 11:32:48,153 | INFO | [main] | Found Yarn properties file under /opt/client123/Flink/tmp/.yarn-properties-root. | org.apache.flink.yarn.cli.FlinkYarnSessionCl
i.<init>(FlinkYarnSessionCli.java:293)
2023-08-18 11:32:48,474 | INFO | [main] | Login successful for user test using keytab file user.keytab. Keytab auto renewal enabled : false | org.apache.hadoop.security.
UserGroupInformation.loginUserFromKeytab(UserGroupInformation.java:1129)


```

- Run the Elasticsearch sample

**flink run flink-es-sink-example.jar**

```
root@host01 conf# flink run flink-es-sink-example.jar
OpenJDK 64-Bit Server VM warning: Cannot open file <LOG_DIR>/gc.log due to No such file or directory
SLF4J: Class path contains multiple SLF4J bindings
SLF4J: Found binding in [jar:file:/opt/client123/Flink/lib/libflink-log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client123/HDFS/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory].
2023-08-18 10:03:13.035 | INFO | [main] | Found Yarn properties file under /opt/client123/Flink/tmp/.yarn-properties-root. | org.apache.flink.yarn.cli.FlinkYarnSessionCl
i.<init>(FlinkYarnSessionCli.java:293)
2023-08-18 10:03:13.035 | INFO | [main] | Found Yarn properties file under /opt/client123/Flink/tmp/.yarn-properties-root. | org.apache.flink.yarn.cli.FlinkYarnSessionCl
i.<init>(FlinkYarnSessionCli.java:293)
2023-08-18 10:03:13.508 | INFO | [main] | Login successful for user test using keytab file user.keytab. Keytab auto renewal enabled : false | org.apache.hadoop.security.
UserGroupInformation.loginUserFromKeytab(UserGroupInformation.java:1129)


```

----End

## 1.8.5 More Information

### 1.8.5.1 Introduction to Common APIs

#### 1.8.5.1.1 Java

To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

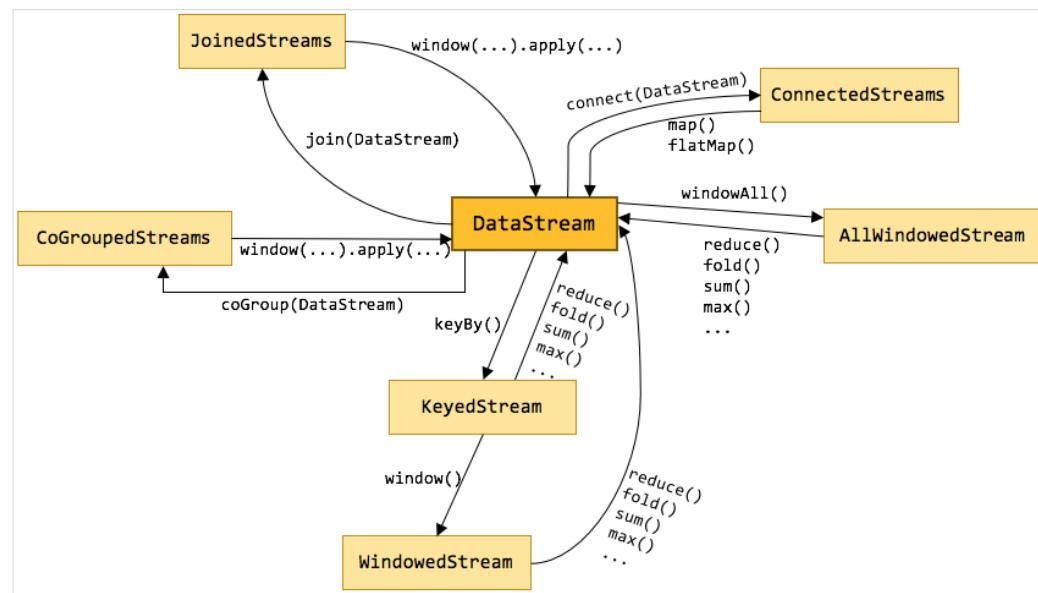
### Common APIs of Flink

Flink mainly uses the following APIs:

- StreamExecutionEnvironment: provides the execution environment, which is the basis of Flink stream processing.
- DataStream: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
- KeyedStream: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
- WindowedStream: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.

- AllWindowedStream: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.
- ConnectedStreams: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- JoinedStreams: performs equijoin (which is performed when two values are equal, for example, a.id = b.id) operation to data in the window. The join operation is a special scenario of coGroup operation.
- CoGroupedStreams: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

**Figure 1-98** Conversion of Flink stream types



## Data Stream Source

**Table 1-37** APIs about data stream source

| API                                                                                                              | Description                                                                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>public final &lt;OUT&gt; DataStreamSource&lt;OUT&gt; fromElements(OUT... data)</pre>                        | Obtain user-defined data of multiple elements as the data stream source. <ul style="list-style-type: none"> <li>• <b>type</b> indicates the data type of an element.</li> <li>• <b>data</b> indicates the data of multiple elements.</li> </ul> |
| <pre>public final &lt;OUT&gt; DataStreamSource&lt;OUT&gt; fromElements(Class&lt;OUT&gt; type, OUT... data)</pre> |                                                                                                                                                                                                                                                 |

| API                                                                                                                                                          | Description                                                                                                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public <OUT><br>DataStreamSource<OUT><br>fromCollection(Collection<OUT> data)                                                                                | Obtain the user-defined data collection as the data stream source. <ul style="list-style-type: none"><li>• <b>type</b> indicates the data type of elements in the collection.</li></ul>                                          |
| public <OUT><br>DataStreamSource<OUT><br>fromCollection(Collection<OUT> data,<br>TypeInformation<OUT> typeInfo)                                              | <ul style="list-style-type: none"><li>• <b>typeInfo</b> indicates the type information obtained based on the element data type in the collection.</li></ul>                                                                      |
| public <OUT><br>DataStreamSource<OUT><br>fromCollection(Iterator<OUT> data,<br>Class<OUT> type)                                                              | <ul style="list-style-type: none"><li>• <b>data</b> indicates the iterator.</li></ul>                                                                                                                                            |
| public <OUT><br>DataStreamSource<OUT><br>fromCollection(Iterator<OUT> data,<br>TypeInformation<OUT> typeInfo)                                                |                                                                                                                                                                                                                                  |
| public <OUT><br>DataStreamSource<OUT><br>fromParallelCollection(SplittableIterator<OUT> iterator, Class<OUT> type)                                           | Obtain the user-defined data collection as parallel data stream source. <ul style="list-style-type: none"><li>• <b>type</b> indicates the data type of elements in the collection.</li></ul>                                     |
| public <OUT><br>DataStreamSource<OUT><br>fromParallelCollection(SplittableIterator<OUT> iterator,<br>TypeInformation<OUT> typeInfo)                          | <ul style="list-style-type: none"><li>• <b>typeInfo</b> indicates the type information obtained based on the element data type in the collection.</li></ul>                                                                      |
| private <OUT><br>DataStreamSource<OUT><br>fromParallelCollection(SplittableIterator<OUT> iterator,<br>TypeInformation<OUT> typeInfo,<br>String operatorName) | <ul style="list-style-type: none"><li>• <b>iterator</b> indicates the iterator that can be divided into multiple partitions.</li></ul>                                                                                           |
| public DataStreamSource<Long><br>generate Sequence(long from, long to)                                                                                       | Obtain a sequence of user-defined data as the data stream source. <ul style="list-style-type: none"><li>• <b>from</b> indicates the starting point of numbers.</li><li>• <b>to</b> indicates the end point of numbers.</li></ul> |
| public DataStreamSource<String><br>readTextFile(String filePath)                                                                                             | Obtain the user-defined text file from a specific path as the data stream source. <ul style="list-style-type: none"><li>• <b>filePath</b> indicates the path of the text file.</li></ul>                                         |
| public DataStreamSource<String><br>readTextFile(String filePath, String<br>charsetName)                                                                      | <ul style="list-style-type: none"><li>• <b>charsetName</b> indicates the encoding format.</li></ul>                                                                                                                              |

| API                                                                                                                                                                                                 | Description                                                                                                                                                                                                                                                                                                                  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public <OUT><br>DataStreamSource<OUT><br>readFile(FileInputformat<OUT><br>inputformat, String filePath)                                                                                             | Obtain the user-defined file from a specific path as the data stream source. <ul style="list-style-type: none"><li>• <b>filePath</b> indicates the file path.</li></ul>                                                                                                                                                      |
| public <OUT><br>DataStreamSource<OUT><br>readFile(FileInputformat<OUT><br>inputformat, String filePath,<br>FileProcessingMode watchType, long<br>interval)                                          | <ul style="list-style-type: none"><li>• <b>inputformat</b> indicates the format of the file.</li><li>• <b>watchType</b> indicates the file processing mode, which can be <b>PROCESS_ONCE</b> or <b>PROCESS_CONTINUOUSLY</b>.</li><li>• <b>interval</b> indicates the interval for processing directories or files.</li></ul> |
| public <OUT><br>DataStreamSource<OUT><br>readFile(FileInputformat<OUT><br>inputformat, String filePath,<br>FileProcessingMode watchType, long<br>interval, TypeInformation<OUT><br>typeInformation) |                                                                                                                                                                                                                                                                                                                              |
| public DataStreamSource<String><br>socketTextStream(String hostname, int<br>port, String delimiter, long maxRetry)                                                                                  | Obtain user-defined socket data as the data stream source. <ul style="list-style-type: none"><li>• <b>hostname</b> indicates the host name of the socket server.</li></ul>                                                                                                                                                   |
| public DataStreamSource<String><br>socketTextStream(String hostname, int<br>port, String delimiter)                                                                                                 | <ul style="list-style-type: none"><li>• <b>port</b> indicates the listening port of the server.</li><li>• <b>delimiter</b> indicates the separator of messages.</li></ul>                                                                                                                                                    |
| public DataStreamSource<String><br>socketTextStream(String hostname, int<br>port)                                                                                                                   | <ul style="list-style-type: none"><li>• <b>maxRetry</b> refers to the maximum retry times that can be triggered by abnormal connections.</li></ul>                                                                                                                                                                           |
| public <OUT><br>DataStreamSource<OUT><br>addSource(SourceFunction<OUT><br>function)                                                                                                                 | Customize the SourceFunction and addSource methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction. <ul style="list-style-type: none"><li>• <b>function</b> indicates the user-defined <b>SourceFunction</b> function.</li></ul>                                            |
| public <OUT><br>DataStreamSource<OUT><br>addSource(SourceFunction<OUT><br>function, String sourceName)                                                                                              | <ul style="list-style-type: none"><li>• <b>sourceName</b> indicates the name of data source.</li></ul>                                                                                                                                                                                                                       |
| public <OUT><br>DataStreamSource<OUT><br>addSource(SourceFunction<OUT><br>function, TypeInformation<OUT><br>typeInfo)                                                                               | <ul style="list-style-type: none"><li>• <b>typeInfo</b> indicates the type information obtained based on the element data type.</li></ul>                                                                                                                                                                                    |

| API                                                                                                                                      | Description |
|------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| public <OUT><br>DataStreamSource<OUT><br>addSource(SourceFunction<OUT><br>function, String sourceName,<br>TypeInformation<OUT> typeInfo) |             |

## Data Output

**Table 1-38** APIs about data output

| API                                                                                                                                          | Description                                                                                                                    |
|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| public DataStreamSink<T> print()                                                                                                             | Print data as the standard output stream.                                                                                      |
| public DataStreamSink<T> printToErr()                                                                                                        | Print data as the standard error output stream.                                                                                |
| public DataStreamSink<T><br>writeAsText(String path)                                                                                         | Write data to a specific text file.<br>• <b>path</b> indicates the path of the text file.                                      |
| public DataStreamSink<T><br>writeAsText(String path, WriteMode<br>writeMode)                                                                 | • <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b> .                          |
| public DataStreamSink<T><br>writeAsCsv(String path)                                                                                          | Writhe data to a specific .csv file.<br>• <b>path</b> indicates the path of the text file.                                     |
| public DataStreamSink<T><br>writeAsCsv(String path, WriteMode<br>writeMode)                                                                  | • <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b> .                          |
| public <X extends Tuple><br>DataStreamSink<T> writeAsCsv(String<br>path, WriteMode writeMode, String<br>rowDelimiter, String fieldDelimiter) | • <b>rowDelimiter</b> indicates the row separator.<br>• <b>fieldDelimiter</b> indicates the column separator.                  |
| public DataStreamSink<T><br>writeToSocket(String hostName, int<br>port, SerializationSchema<T> schema)                                       | Write data to the socket connection.<br>• <b>hostName</b> indicates the host name.<br>• <b>port</b> indicates the port number. |
| public DataStreamSink<T><br>writeUsingOutputfor-<br>mat(Outputformat<T> format)                                                              | Write data to a file, for example, a binary file.                                                                              |

| API                                                                  | Description                                                                                                                                                                              |
|----------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public DataStreamSink<T><br>addSink(SinkFunction<T><br>sinkFunction) | Export data in a user-defined manner. The flink-connectors allows the addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction. |

## Filtering and Mapping

**Table 1-39** APIs about filtering and mapping

| API                                                                                      | Description                                                                          |
|------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| public <R> SingleOutputStreamOperator<R> map(MapFunction<T, R><br>mapper)                | Transform an element into another element of the same type.                          |
| public <R> SingleOutputStreamOperator<R> flatMap(FlatMapFunction<T, R><br>flatMapMapper) | Transform an element into zero, one, or multiple elements.                           |
| public SingleOutputStreamOperator<T> filter(FilterFunction<T> filter)                    | Run a Boolean function on each element and retain elements that return <b>true</b> . |

## Aggregation

**Table 1-40** APIs about aggregation

| API                                                          | Description                                                                                                                                                                                               |
|--------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public KeyedStream<T, Tuple><br>keyBy(int... fields)         | Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is internally implemented using hash partition. A KeyedStream is returned.               |
| public KeyedStream<T, Tuple><br>keyBy(String... fields)      | Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby.                                                        |
| public <K> KeyedStream<T, K><br>keyBy(KeySelector<T, K> key) | <ul style="list-style-type: none"> <li>• <b>fields</b> indicates the numbers of columns or names of member variables.</li> <li>• <b>key</b> indicates the user-defined basis for partitioning.</li> </ul> |

| API                                                                                      | Description                                                                                                                                                                                                                  |
|------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> reducer)                   | Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.                                                    |
| public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> folder) | Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.       |
| public SingleOutputStreamOperator<T> sum(int positionToSum)                              | Calculate the sum in a KeyedStream in a rolling manner.                                                                                                                                                                      |
| public SingleOutputStreamOperator<T> sum(String field)                                   | <b>positionToSum</b> and <b>field</b> indicate calculating the sum of a specific column.                                                                                                                                     |
| public SingleOutputStreamOperator<T> min(int positionToMin)                              | Calculate the minimum value in a KeyedStream in a rolling manner. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values.                                               |
| public SingleOutputStreamOperator<T> min(String field)                                   | <b>positionToMin</b> and <b>field</b> indicate calculating the minimum value of a specific column.                                                                                                                           |
| public SingleOutputStreamOperator<T> max(int positionToMax)                              | Calculate the maximum value in KeyedStream in a rolling manner. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values.                                                 |
| public SingleOutputStreamOperator<T> max(String field)                                   | <b>positionToMax</b> and <b>field</b> indicate calculating the maximum value of a specific column.                                                                                                                           |
| public SingleOutputStreamOperator<T> minBy(int positionToMinBy)                          | Obtain the row where the minimum value of a column locates in a KeyedStream. <b>minBy</b> returns all elements of that row.                                                                                                  |
| public SingleOutputStreamOperator<T> minBy(String positionToMinBy)                       |                                                                                                                                                                                                                              |
| public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)           | <ul style="list-style-type: none"><li>• <b>positionToMinBy</b> indicates the column on which the minBy operation is performed.</li><li>• <b>first</b> indicates whether to return the first or last minimum value.</li></ul> |
| public SingleOutputStreamOperator<T> minBy(String field, boolean first)                  |                                                                                                                                                                                                                              |

| API                                                                            | Description                                                                                                                                                                                                                  |
|--------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)                | Obtain the row where the maximum value of a column locates in a KeyedStream. maxBy returns all elements of that row.                                                                                                         |
| public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)             |                                                                                                                                                                                                                              |
| public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first) |                                                                                                                                                                                                                              |
| public SingleOutputStreamOperator<T> maxBy(String field, boolean first)        | <ul style="list-style-type: none"><li>• <b>positionToMaxBy</b> indicates the column on which the maxBy operation is performed.</li><li>• <b>first</b> indicates whether to return the first or last maximum value.</li></ul> |

## DataStream Distribution

**Table 1-41** APIs about DataStream distribution

| API                                                                                                             | Description                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public <K><br>DataStream<T><br>partitionCustom(Partitioner<K> partitioner,<br>int field)                        | Use a user-defined partitioner to select target task for each element. <ul style="list-style-type: none"><li>• <b>partitioner</b> indicates the user-defined method for repartitioning.</li><li>• <b>field</b> indicates the input parameters of partitioner.</li><li>• <b>keySelector</b> indicates the user-defined input parameters of partitioner.</li></ul> |
| public <K><br>DataStream<T><br>partitionCustom(Partitioner<K> partitioner,<br>String field)                     |                                                                                                                                                                                                                                                                                                                                                                  |
| public <K><br>DataStream<T><br>partitionCustom(Partitioner<K> partitioner,<br>KeySelector<T, K><br>keySelector) |                                                                                                                                                                                                                                                                                                                                                                  |
| public<br>DataStream<T><br>shuffle()                                                                            | Randomly and evenly partition elements.                                                                                                                                                                                                                                                                                                                          |
| public<br>DataStream<T><br>rebalance()                                                                          | Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.                                                                                                                                                                                                                           |
| public<br>DataStream<T><br>rescale()                                                                            | Distribute elements into downstream subset in round-robin manner.                                                                                                                                                                                                                                                                                                |

| API                                    | Description                               |
|----------------------------------------|-------------------------------------------|
| public<br>DataStream<T><br>broadcast() | Broadcast each element to all partitions. |

## Project Capabilities

**Table 1-42** APIs about projecting

| API                                                                                          | Description                                                                                                                                                                    |
|----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public <R extends Tuple><br>SingleOutputStreamOperator<R><br>project(int...<br>fieldIndexes) | Select some field subset from the tuple.<br><b>fieldIndexes</b> indicates some sequences of the tuple.<br><b>NOTE</b><br>Only tuple data type is supported by the project API. |

## Configuring the eventtime Attribute

**Table 1-43** APIs about configuring the eventtime attribute

| API                                                                                                                                            | Description                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| public<br>SingleOutputStreamOperator<T><br>assignTimestampsAndWatermarks(AssignerWithPeriodicWatermarks<T><br>timestampAndWatermarkAssigner)   | Extract timestamp from records, enabling event time window to trigger computing. |
| public<br>SingleOutputStreamOperator<T><br>assignTimestampsAndWatermarks(AssignerWithPunctuatedWatermarks<T><br>timestampAndWatermarkAssigner) |                                                                                  |

**Table 1-44** lists differences of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs.

**Table 1-44** Difference of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs

| Parameter                        | Description                                                                                                               |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| AssignerWithPeriodicWatermarks   | Generate Watermark based on the getConfig().setAutoWatermarkInterval(200L) timestamp of StreamExecutionEnvironment class. |
| AssignerWithPunctuatedWatermarks | Generate a Watermark each time an element is received. Watermarks can be different based on received elements.            |

## Iteration

**Table 1-45** APIs about iteration

| API                                                       | Description                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public IterativeStream<T> iterate()                       | In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator.<br><b>NOTE</b> <ul style="list-style-type: none"><li>This API is helpful to algorithms that require constant update of models.</li><li>long maxWaitTimeMillis: The timeout period of each round of iteration.</li></ul> |
| public IterativeStream<T> iterate(long maxWaitTimeMillis) |                                                                                                                                                                                                                                                                                                                               |

## Stream Splitting

**Table 1-46** APIs about stream splitting

| API                                                           | Description                                                                                                                                                                                                                               |
|---------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public SplitStream<T> split(OutputSelector<T> outputSelector) | Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created. |
| public DataStream<OUT> select(String... outputNames)          | Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.                                                                                     |

## Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

**Table 1-47** lists APIs for generating windows.

**Table 1-47** APIs for generating windows

| API                                                                                                            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public <W extends Window><br>WindowedStream<T,<br>KEY, W><br>window(WindowAssigner<? super T, W><br>assigner)  | Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| public <W extends Window><br>AllWindowedStream<T,<br>W><br>windowAll(WindowAssigner<? super T, W><br>assigner) | Define windows in DataStreams.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| public<br>WindowedStream<T,<br>KEY, TimeWindow><br>timeWindow(Time size)                                       | Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"><li>• <b>size</b> indicates the duration of the window.</li><li>• <b>slide</b> indicates the sliding time of window.</li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• WindowedStream and AllWindowedStream indicates two types of streams.</li><li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li></ul> |
| public<br>WindowedStream<T,<br>KEY, TimeWindow><br>timeWindow(Time size,<br>Time slide)                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| API                                                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public<br>AllWindowedStream<T,<br>TimeWindow><br>timeWindowAll(Time<br>size)                | Define time windows in partitioned DataStream,<br>select ProcessingTime or EventTime based on the<br>environment.getStreamTimeCharacteristic()<br>parameter, and determine whether the window is<br>TumblingWindow or SlidingWindow depends on the<br>number of parameters.                                                                                                                                                                                               |
| public<br>AllWindowedStream<T,<br>TimeWindow><br>timeWindowAll(Time<br>size, Time slide)    | <ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>                                                                                                                                                                                                                                                                                                       |
| public<br>WindowedStream<T,<br>KEY, GlobalWindow><br>countWindow(long size)                 | Divide windows according to the number of<br>elements and define windows in partitioned<br>KeyedStreams.                                                                                                                                                                                                                                                                                                                                                                  |
| public<br>WindowedStream<T,<br>KEY, GlobalWindow><br>countWindow(long size,<br>long slide)  | <ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• WindowedStream and AllWindowedStream indicates two types of streams.</li> <li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li> </ul> |
| public<br>AllWindowedStream<T,<br>GlobalWindow><br>countWindowAll(Time<br>size)             | Divide windows according to the number of<br>elements and define windows in DataStreams.                                                                                                                                                                                                                                                                                                                                                                                  |
| public<br>AllWindowedStream<T,<br>GlobalWindow><br>countWindowAll(Time<br>size, Time slide) | <ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>                                                                                                                                                                                                                                                                                                       |

**Table 1-48** lists APIs for operating windows.**Table 1-48** APIs for operating windows

| Method | API                                                                                            | Description                                                                                                                                                                                                                                                                                       |
|--------|------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Window | public <R> SingleOutputStrea-<br>mOperator<R><br>apply(WindowFunction<T, R, K,<br>W> function) | <p>Apply a general function to a<br/>window. The data in the window is<br/>calculated as a whole.</p> <ul style="list-style-type: none"> <li>• <b>function</b> indicates the window<br/>function to be executed.</li> <li>• <b>resultType</b> indicates the type of<br/>returned data.</li> </ul> |

| Method | API                                                                                                                                                                                             | Description                                                                                                                                                                                                            |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | <pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; apply(WindowFunction&lt;T, R, K, W&gt; function, TypeInformation&lt;R&gt; resultType)</pre>                                           |                                                                                                                                                                                                                        |
|        | <pre>public SingleOutputStreamOperator&lt;T&gt; reduce(ReduceFunction&lt;T&gt; function)</pre>                                                                                                  | Apply a reduce function to the window and return the result. <ul style="list-style-type: none"><li>• <b>reduceFunction</b> indicates the reduce function to be executed.</li></ul>                                     |
|        | <pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; reduce(ReduceFunction&lt;T&gt; reduceFunction, WindowFunction&lt;T, R, K, W&gt; function)</pre>                                       | <ul style="list-style-type: none"><li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a reduce operation.</li></ul>                                                          |
|        | <pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; reduce( ReduceFunction&lt;T&gt; reduceFunction, WindowFunction&lt;T, R, K, W&gt; function, TypeInformation&lt;R&gt; resultType)</pre> | <ul style="list-style-type: none"><li>• <b>resultType</b> indicates the type of returned data.</li></ul>                                                                                                               |
|        | <pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; fold(R initialValue, FoldFunction&lt;T, R&gt; function)</pre>                                                                         | Apply a fold function to the window and return the result. <ul style="list-style-type: none"><li>• <b>initialValue</b> indicates the initial value.</li></ul>                                                          |
|        | <pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; fold(R initialValue, FoldFunction&lt;T, R&gt; function, TypeInformation&lt;R&gt; resultType)</pre>                                    | <ul style="list-style-type: none"><li>• <b>foldFunction</b> indicates the fold function.</li><li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a fold operation.</li></ul> |
|        | <pre>public &lt;ACC, R&gt; SingleOutputStreamOperator&lt;R&gt; fold(ACC initialValue, FoldFunction&lt;T, ACC&gt; foldFunction, WindowFunction&lt;ACC, R, K, W&gt; function)</pre>               | <ul style="list-style-type: none"><li>• <b>resultType</b> indicates the type of returned data.</li></ul>                                                                                                               |

| Method     | API                                                                                                                                                                                                                                                | Description                                                                                                                                                                             |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | public <ACC, R><br>SingleOutputStreamOperator<R> fold(ACC initialValue,<br>FoldFunction<T, ACC><br>foldFunction,<br>WindowFunction<ACC, R, K,<br>W> function,<br>TypeInformation<ACC><br>foldAccumulatorType,<br>TypeInformation<R><br>resultType) |                                                                                                                                                                                         |
| Window All | public <R> SingleOutputStreamOperator<R><br>apply(AllWindowFunction<T, R,<br>W> function)                                                                                                                                                          | Apply a general function to a window. The data in the window is calculated as a whole.                                                                                                  |
|            | public <R> SingleOutputStreamOperator<R><br>apply(AllWindowFunction<T, R,<br>W> function,<br>TypeInformation<R><br>resultType)                                                                                                                     |                                                                                                                                                                                         |
|            | public SingleOutputStreamOperator<T><br>reduce(ReduceFunction<T><br>function)                                                                                                                                                                      | Apply a reduce function to the window and return the result.<br><ul style="list-style-type: none"> <li>• <b>reduceFunction</b> indicates the reduce function to be executed.</li> </ul> |
|            | public <R> SingleOutputStreamOperator<R><br>reduce(ReduceFunction<T><br>reduceFunction,<br>AllWindowFunction<T, R, W><br>function)                                                                                                                 | <ul style="list-style-type: none"> <li>• <b>AllWindowFunction</b> indicates triggering an operation to the window after a reduce operation.</li> </ul>                                  |
|            | public <R> SingleOutputStreamOperator<R><br>reduce(ReduceFunction<T><br>reduceFunction,<br>AllWindowFunction<T, R, W><br>function, TypeInformation<R><br>resultType)                                                                               | <ul style="list-style-type: none"> <li>• <b>resultType</b> indicates the type of returned data.</li> </ul>                                                                              |

| Method                | API                                                                                                                                                                                                                                                                    | Description                                                                                                                                                                                                               |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                       | <pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; fold(R initialValue, FoldFunction&lt;T, R&gt; function)</pre>                                                                                                                                                | Apply a fold function to the window and return the result.<br><ul style="list-style-type: none"> <li>• <b>initialValue</b> indicates the initial value.</li> </ul>                                                        |
|                       | <pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; fold(R initialValue, FoldFunction&lt;T, R&gt; function, TypeInformation&lt;R&gt; resultType)</pre>                                                                                                           | <ul style="list-style-type: none"> <li>• <b>foldFunction</b> indicates the fold function.</li> <li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a fold operation.</li> </ul> |
|                       | <pre>public &lt;ACC, R&gt; SingleOutputStreamOperator&lt;R&gt; fold(ACC initialValue, FoldFunction&lt;T, ACC&gt; foldFunction, AllWindowFunction&lt;ACC, R, W&gt; function)</pre>                                                                                      | <ul style="list-style-type: none"> <li>• <b>resultType</b> indicates the type of returned data.</li> </ul>                                                                                                                |
|                       | <pre>public &lt;ACC, R&gt; SingleOutputStreamOperator&lt;R&gt; fold(ACC initialValue, FoldFunction&lt;T, ACC&gt; foldFunction, AllWindowFunction&lt;ACC, R, W&gt; function, TypeInformation&lt;ACC&gt; foldAccumulatorType, TypeInformation&lt;R&gt; resultType)</pre> |                                                                                                                                                                                                                           |
| Window and Window All | <pre>public SingleOutputStreamOperator&lt;T&gt; sum(int positionToSum)</pre>                                                                                                                                                                                           | Sum a specified column of the window data.<br><b>field</b> and <b>positionToSum</b> indicate a specific column of the data.                                                                                               |
|                       | <pre>public SingleOutputStreamOperator&lt;T&gt; sum(String field)</pre>                                                                                                                                                                                                |                                                                                                                                                                                                                           |
|                       | <pre>public SingleOutputStreamOperator&lt;T&gt; min(int positionToMin)</pre>                                                                                                                                                                                           | Calculate the minimum value of a specified column of the window data. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values.                                        |
|                       | <pre>public SingleOutputStreamOperator&lt;T&gt; min(String field)</pre>                                                                                                                                                                                                | <b>positionToMin</b> and <b>field</b> indicate calculating the minimum value of a specific column.                                                                                                                        |

| Method | API                                                                                  | Description                                                                                                                                                                        |
|--------|--------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | public SingleOutputStreamOperator<T> minBy(int positionToMinBy)                      | Obtain the row where the minimum value of a column locates in the window data. <b>minBy</b> returns all elements of that row.                                                      |
|        | public SingleOutputStreamOperator<T> minBy(String positionToMinBy)                   | <ul style="list-style-type: none"><li>• <b>positionToMinBy</b> indicates the column on which the <b>minBy</b> operation is performed.</li></ul>                                    |
|        | public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)       | <ul style="list-style-type: none"><li>• <b>first</b> indicates whether to return the first or last minimum value.</li></ul>                                                        |
|        | public SingleOutputStreamOperator<T> minBy(String field, boolean first)              |                                                                                                                                                                                    |
|        | public SingleOutputStreamOperator<T> max(int positionToMax)                          | Calculate the maximum value of a specified column of the window data. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values. |
|        | public SingleOutputStreamOperator<T> max(String field)                               | <b>positionToMax</b> and <b>field</b> indicate calculating the maximum value of a specific column.                                                                                 |
|        | The default public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)//true    | Obtain the row where the maximum value of a column locates in the window data. <b>maxBy</b> returns all elements of that row.                                                      |
|        | The default public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)//true | <ul style="list-style-type: none"><li>• <b>positionToMaxBy</b> indicates the column on which the <b>maxBy</b> operation is performed.</li></ul>                                    |
|        | public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first)       | <ul style="list-style-type: none"><li>• <b>first</b> indicates whether to return the first or last maximum value.</li></ul>                                                        |
|        | public SingleOutputStreamOperator<T> maxBy(String field, boolean first)              |                                                                                                                                                                                    |

## Combining Multiple DataStreams

**Table 1-49** APIs about combining multiple DataStreams

| API                                                                                                                                         | Description                                                                                                                                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>public final<br/>DataStream&lt;T&gt;<br/>union(DataStream&lt;T&gt;..<br/>. streams)</pre>                                              | <p>Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams.</p> <p><b>NOTE</b><br/>If you perform union operation on a piece of data with itself, there are two copies of the same data.</p> |
| <pre>public &lt;R&gt;<br/>ConnectedStreams&lt;T,<br/>R&gt;<br/>connect(DataStream&lt;R<br/>&gt; dataStream)</pre>                           | Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.                                                              |
| <pre>public &lt;R&gt;<br/>SingleOutputStreamOperator&lt;R&gt;<br/>map(CoMapFunction&lt;I<br/>N1, IN2, R&gt; coMapper)</pre>                 | Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.                                                                                     |
| <pre>public &lt;R&gt;<br/>SingleOutputStreamOperator&lt;R&gt;<br/>flatMap(CoFlatMapFun<br/>ction&lt;IN1, IN2, R&gt;<br/>coFlatMapper)</pre> | Perform mapping operation, which is similar to flatMap operation in DataStream, on elements.                                                                                                                                                                     |

## Join Operation

**Table 1-50** APIs about join operation

| API                                                                                                                   | Description                                                       |
|-----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| <pre>public &lt;T2&gt;<br/>JoinedStreams&lt;T, T2&gt;<br/>join(DataStream&lt;T2&gt;<br/>otherStream)</pre>            | Join two DataStreams using a given key in a specified window.     |
| <pre>public &lt;T2&gt;<br/>CoGroupedStreams&lt;T,<br/>T2&gt;<br/>coGroup(DataStream&lt;<br/>T2&gt; otherStream)</pre> | Co-group two DataStreams using a given key in a specified window. |

### 1.8.5.1.2 Scala

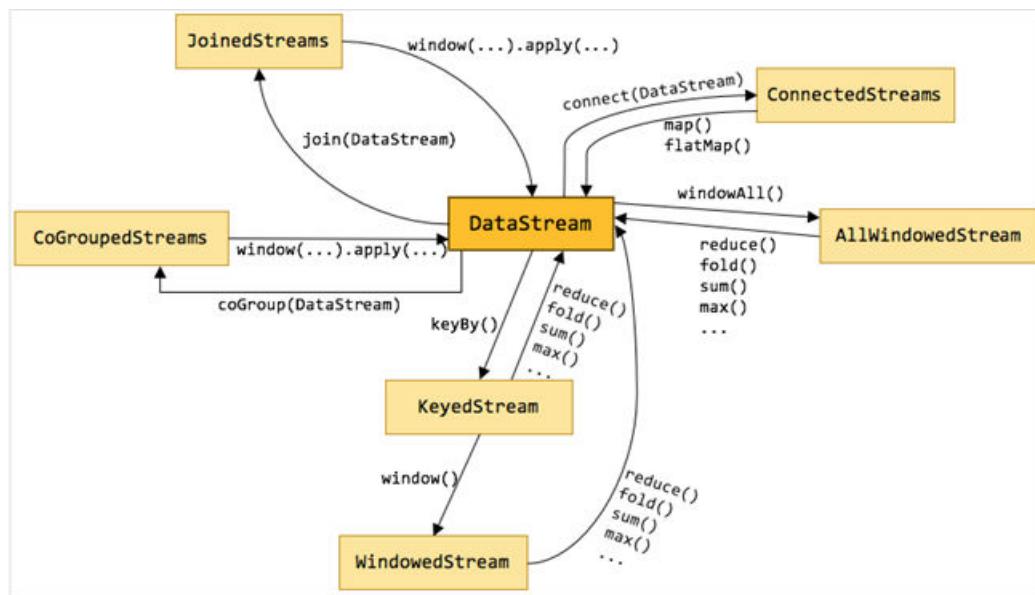
To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended.

## Common APIs of Flink

Flink mainly uses the following APIs:

- StreamExecutionEnvironment: provides the execution environment, which is the basis of Flink stream processing.
- DataStream: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
- KeyedStream: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
- WindowedStream: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.
- AllWindowedStream: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.
- ConnectedStreams: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- JoinedStreams: performs equijoin operation to data in the window. The join operation is a special scenario of coGroup operation.
- CoGroupedStreams: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

Figure 1-99 Conversion of Flink stream types



## Data Stream Source

**Table 1-51** APIs about data stream source

| API                                                                                                                                                            | Description                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| def fromElements[T: TypeInformation]<br>(data: T*): DataStream[T]                                                                                              | Obtain user-defined data of multiple elements as the data stream source.<br><b>data</b> is the specific data of multiple elements.                                                                                                                                                                                                                                              |
| def fromCollection[T: TypeInformation]<br>(data: Seq[T]): DataStream[T]                                                                                        | Obtain the user-defined data collection as input data stream.                                                                                                                                                                                                                                                                                                                   |
| def fromCollection[T: TypeInformation]<br>(data: Iterator[T]): DataStream[T]                                                                                   | <b>data</b> can be a data collection or a data body that can be iterated.                                                                                                                                                                                                                                                                                                       |
| def fromParallelCollection[T:<br>TypeInformation] (data:<br>SplittableIterator[T]): DataStream[T]                                                              | Obtain the user-defined data collection as parallel data stream source.<br><b>data</b> indicates the iterator that can be divided into multiple partitions.                                                                                                                                                                                                                     |
| def generateSequence(from: Long, to:<br>Long): DataStream[Long]                                                                                                | Obtain a sequence of user-defined data as the data stream source. <ul style="list-style-type: none"><li>• <b>from</b> indicates the starting point of numbers.</li><li>• <b>to</b> indicates the end point of numbers.</li></ul>                                                                                                                                                |
| def readTextFile(filePath: String):<br>DataStream[String]                                                                                                      | Obtain the user-defined text file from a specific path as the data stream source.                                                                                                                                                                                                                                                                                               |
| def readTextFile(filePath: String,<br>charsetName: String):<br>DataStream[String]                                                                              | <ul style="list-style-type: none"><li>• <b>filePath</b> indicates the path of the text file.</li><li>• <b>charsetName</b> indicates the encoding format.</li></ul>                                                                                                                                                                                                              |
| def readFile[T: TypeInformation]<br>(inputFormat: FileInputFormat[T],<br>filePath: String)                                                                     | Obtain the user-defined file from a specific path as the data stream source.                                                                                                                                                                                                                                                                                                    |
| def readFile[T: TypeInformation]<br>(inputFormat: FileInputFormat[T],<br>filePath: String, watchType:<br>FileProcessingMode, interval: Long):<br>DataStream[T] | <ul style="list-style-type: none"><li>• <b>filePath</b> indicates the file path.</li><li>• <b>inputFormat</b> indicates the format of the file.</li><li>• <b>watchType</b> indicates the file processing mode, which can be <b>PROCESS_ONCE</b> or <b>PROCESS_CONTINUOUSLY</b>.</li><li>• <b>interval</b> indicates the interval for processing directories or files.</li></ul> |

| API                                                                                                                          | Description                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>def socketTextStream(hostname: String, port: Int, delimiter: Char = '\n', maxRetry: Long = 0): DataStream[String]</pre> | <p>Obtain user-defined socket data as the data stream source.</p> <ul style="list-style-type: none"><li>• <b>hostname</b> indicates the host name of the socket server.</li><li>• <b>port</b> indicates the listening port of the server.</li><li>• <b>delimiter</b> and <b>maxRetry</b> are not supported by Scala APIs.</li></ul>                       |
| <pre>def addSource[T: TypeInformation](function: SourceFunction[T]): DataStream[T]</pre>                                     | <p>Customize the <b>SourceFunction</b> and <b>addSource</b> methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction.</p> <ul style="list-style-type: none"><li>• <b>function</b> indicates the user-defined <b>SourceFunction</b> function.</li><li>• Simplified format is supported by Scala.</li></ul> |
| <pre>def addSource[T: TypeInformation](function: SourceContext[T] =&gt; Unit): DataStream[T]</pre>                           |                                                                                                                                                                                                                                                                                                                                                           |

## Data Output

**Table 1-52** APIs about data output

| API                                                                                          | Description                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>def print(): DataStreamSink[T]</pre>                                                    | Print data as the standard output stream.                                                                                                                                                                       |
| <pre>def printToErr()</pre>                                                                  | Print data as the standard error output stream.                                                                                                                                                                 |
| <pre>def writeAsText(path: String): DataStreamSink[T]</pre>                                  | Write data to a specific text file.                                                                                                                                                                             |
| <pre>def writeAsText(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]</pre> | <ul style="list-style-type: none"><li>• <b>path</b> indicates the path of the text file.</li><li>• <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b>.</li></ul> |

| API                                                                                                                            | Description                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| def writeAsCsv(path: String): DataStreamSink[T]                                                                                | Writhe data to a specific .csv file.<br>• <b>path</b> indicates the path of the text file.                                                                                           |
| def writeAsCsv(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]                                               | • <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b> .                                                                                |
| def writeAsCsv(path: String, writeMode: FileSystem.WriteMode, rowDelimiter: String, fieldDelimiter: String): DataStreamSink[T] | • <b>rowDelimiter</b> indicates the row separator.<br>• <b>fieldDelimiter</b> indicates the column separator.                                                                        |
| def writeUsingOutputFormat(format: OutputFormat[T]): DataStreamSink[T]                                                         | Write data to a file, for example, a binary file.                                                                                                                                    |
| def writeToSocket(hostname: String, port: Integer, schema: SerializationSchema[T]): DataStreamSink[T]                          | Write data to the socket connection.<br>• <b>hostName</b> indicates the host name.<br>• <b>port</b> indicates the port number.                                                       |
| def addSink(sinkFunction: SinkFunction[T]): DataStreamSink[T]                                                                  | Export data in a user-defined manner. The flink-connectors allows addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction. |
| def addSink(fun: T => Unit): DataStreamSink[T]                                                                                 |                                                                                                                                                                                      |

## Filtering and Mapping

Table 1-53 APIs about filtering and mapping

| API                                                                                  | Description                                                 |
|--------------------------------------------------------------------------------------|-------------------------------------------------------------|
| def map[R: TypeInformation](fun: T => R): DataStream[R]                              | Transform an element into another element of the same type. |
| def map[R: TypeInformation](mapper: MapFunction[T, R]): DataStream[R]                |                                                             |
| def flatMap[R: TypeInformation](flatMapMapper: FlatMapFunction[T, R]): DataStream[R] | Transform an element into zero, one, or multiple elements.  |
| def flatMap[R: TypeInformation](fun: (T, Collector[R]) => Unit): DataStream[R]       |                                                             |
| def flatMap[R: TypeInformation](fun: T => TraversableOnce[R]): DataStream[R]         |                                                             |

| API                                                  | Description                                                                          |
|------------------------------------------------------|--------------------------------------------------------------------------------------|
| def filter(filter: FilterFunction[T]): DataStream[T] | Run a Boolean function on each element and retain elements that return <b>true</b> . |
| def filter(fun: T => Boolean): DataStream[T]         |                                                                                      |

## Aggregation

**Table 1-54** APIs about aggregation

| API                                                                                      | Description                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| def keyBy(fields: Int*): KeyedStream[T, JavaTuple]                                       | Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is internally implemented using hash partition. A KeyedStream is returned.                                                                                                                                                                                                                       |
| def keyBy(firstField: String, otherFields: String*): KeyedStream[T, JavaTuple]           |                                                                                                                                                                                                                                                                                                                                                                                                                   |
| def keyBy[K: TypeInformation](fun: T => K): KeyedStream[T, K]                            | Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. <ul style="list-style-type: none"><li>• <b>fields</b> indicates the IDs of certain columns</li><li>• <b>firstField</b> and <b>otherFields</b> are names of member variables.</li><li>• <b>key</b> indicates the user-defined basis for partitioning.</li></ul> |
| def reduce(fun: (T, T) => T): DataStream[T]                                              | Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.                                                                                                                                                                                                                                         |
| def reduce(reducer: ReduceFunction[T]): DataStream[T]                                    |                                                                                                                                                                                                                                                                                                                                                                                                                   |
| def fold[R: TypeInformation] (initialValue: R)(fun: (R,T) => R): DataStream[R]           | Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.                                                                                                                                                                                            |
| def fold[R: TypeInformation] (initialValue: R, folder: FoldFunction[T,R]): DataStream[R] |                                                                                                                                                                                                                                                                                                                                                                                                                   |
| def sum(position: Int): DataStream[T]                                                    | Calculate the sum in a KeyedStream in a rolling manner.                                                                                                                                                                                                                                                                                                                                                           |
| def sum(field: String): DataStream[T]                                                    | <b>position</b> and <b>field</b> indicate calculating the sum of a specific column.                                                                                                                                                                                                                                                                                                                               |

| API                                     | Description                                                                                                                                                                    |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| def min(position: Int): DataStream[T]   | Calculate the minimum value in a KeyedStream in a rolling manner. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values. |
| def min(field: String): DataStream[T]   | <b>position and field</b> indicate calculating the minimum value of a specific column.                                                                                         |
| def max(position: Int): DataStream[T]   | Calculate the maximum value in KeyedStream in a rolling manner. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values.   |
| def max(field: String): DataStream[T]   | <b>position and field</b> indicate calculating the maximum value of a specific column.                                                                                         |
| def minBy(position: Int): DataStream[T] | Obtain the row where the minimum value of a column locates in a KeyedStream. <b>minBy</b> returns all elements of that row.                                                    |
| def minBy(field: String): DataStream[T] | <b>position and field</b> indicate the column on which the <b>minBy</b> operation is performed.                                                                                |
| def maxBy(position: Int): DataStream[T] | Obtain the row where the maximum value of a column locates in a KeyedStream. <b>maxBy</b> returns all elements of that row.                                                    |
| def maxBy(field: String): DataStream[T] | <b>position and field</b> indicate the column on which the <b>maxBy</b> operation is performed.                                                                                |

## DataStream Distribution

**Table 1-55** APIs about DataStream distribution

| API                                                                                                              | Description                                                                                                                                                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], field: Int) : DataStream[T]</code>    | Use a user-defined partitioner to select target task for each element. <ul style="list-style-type: none"><li>• <b>partitioner</b> indicates the user-defined method for repartitioning.</li><li>• <b>field</b> indicates the input parameters of partitioner.</li><li>• <b>keySelector</b> indicates the user-defined input parameters of partitioner.</li></ul> |
| <code>def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], field: String):DataStream[T]</code>   |                                                                                                                                                                                                                                                                                                                                                                  |
| <code>def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], fun: T =&gt; K): DataStream[T]</code> |                                                                                                                                                                                                                                                                                                                                                                  |
| <code>def shuffle: DataStream[T]</code>                                                                          | Randomly and evenly partition elements.                                                                                                                                                                                                                                                                                                                          |
| <code>def rebalance: DataStream[T]</code>                                                                        | Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.                                                                                                                                                                                                                           |
| <code>def rescale: DataStream[T]</code>                                                                          | Distribute elements into downstream subset in round-robin manner.<br><b>NOTE</b><br>The method for checking the code is similar to the rebalance method.                                                                                                                                                                                                         |
| <code>def broadcast: DataStream[T]</code>                                                                        | Broadcast each element to all partitions.                                                                                                                                                                                                                                                                                                                        |

## Configuring the eventtime Attribute

**Table 1-56** APIs about configuring the eventtime attribute

| API                                                                                                        | Description                                                                      |
|------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| <code>def assignTimestampsAndWatermarks(assigner: AssignerWithPeriodicWatermarks[T]): DataStream[T]</code> | Extract timestamp from records, enabling event time window to trigger computing. |

| API                                                                                                          | Description |
|--------------------------------------------------------------------------------------------------------------|-------------|
| <code>def assignTimestampsAndWatermarks(assigner: AssignerWithPunctuatedWatermarks[T]): DataStream[T]</code> |             |

## Iteration

**Table 1-57** APIs about iteration

| API                                                                                                                                                                           | Description                                                                                                                                                                                                                                                                                                                                     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>def iterate[R](stepFunction: DataStream[T] =&gt; (DataStream[T], DataStream[R]), maxWaitTimeMillis: Long = 0, keepPartitioning: Boolean = false) : DataStream[R]</code> | <p>In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• This API is helpful to algorithms that require constant update of models.</li> <li>• long maxWaitTimeMillis: The timeout period of each round of iteration.</li> </ul> |
| <code>def iterate[R, F: TypeInformation](stepFunction: ConnectedStreams[T, F] =&gt; (DataStream[F], DataStream[R]), maxWaitTimeMillis: Long): DataStream[R]</code>            |                                                                                                                                                                                                                                                                                                                                                 |

## Stream Splitting

**Table 1-58** APIs about stream splitting

| API                                                                 | Description                                                                                                                                                                                                                               |
|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>def split(selector: OutputSelector[T]): SplitStream[T]</code> | Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created. |
| <code>def select(outputNames: String*): DataStream[T]</code>        | Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.                                                                                     |

## Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

[Table9 APIs for generating windows](#) lists APIs for generating windows.

**Table 1-59** APIs for generating windows

| API                                                                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>def window[W &lt;: Window](assigner: WindowAssigner[_ &gt;: T, W]): WindowedStream[T, K, W]</code>    | Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>def windowAll[W &lt;: Window](assigner: WindowAssigner[_ &gt;: T, W]): AllWindowedStream[T, W]</code> | Define windows in DataStreams.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <code>def timeWindow(size: time<br/>WindowedStream[T, K, TimeWindow]</code>                                 | Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"><li>• <b>size</b> indicates the duration of the window.</li><li>• <b>slide</b> indicates the sliding time of window.</li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• WindowedStream and AllWindowedStream indicates two types of streams.</li><li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li></ul> |
| <code>def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]</code>                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| API                                                                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| def timeWindowAll(size: time<br>AllWindowedStream[T, TimeWindow]                | Define time windows in partitioned DataStream, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters.<br><ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul> |
| def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]    | Divide windows according to the number of elements and define windows in partitioned KeyedStreams.<br><ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>                                                                                                                                                           |
| def countWindow(size: Long): WindowedStream[T, K, GlobalWindow]                 | <b>NOTE</b> <ul style="list-style-type: none"> <li>• WindowedStream and AllWindowedStream indicates two types of streams.</li> <li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li> </ul>                                                                                                                                      |
| def countWindowAll(size: Long, slide: Long): AllWindowedStream[T, GlobalWindow] | Divide windows according to the number of elements and define windows in DataStreams.<br><ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>                                                                                                                                                                        |
| def countWindowAll(size: Long): AllWindowedStream[T, GlobalWindow]              |                                                                                                                                                                                                                                                                                                                                                                                                                                     |

**Table 1-60** lists APIs for operating windows.

**Table 1-60** APIs for operating windows

| Method | API                                                                                                   | Description                                                                                                                                             |
|--------|-------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Window | def apply[R: TypeInformation]<br>(function: WindowFunction[T, R, K, W]): DataStream[R]                | Apply a general function to a window. The data in the window is calculated as a whole.<br><b>function</b> indicates the window function to be executed. |
|        | def apply[R: TypeInformation]<br>(function: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R] |                                                                                                                                                         |

| Method     | API                                                                                                                                                                                               | Description                                                                                                                                                      |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | <pre>def reduce(function: ReduceFunction[T]): DataStream[T]</pre>                                                                                                                                 | Apply a reduce function to the window and return the result.                                                                                                     |
|            | <pre>def reduce(function: (T, T) =&gt; T): DataStream[T]</pre>                                                                                                                                    | <ul style="list-style-type: none"> <li>• <b>reduceFunction</b> indicates the reduce function to be executed.</li> </ul>                                          |
|            | <pre>def reduce[R: TypeInformation] (preAggregator: ReduceFunction[T], function: WindowFunction[T, R, K, W]): DataStream[R]</pre>                                                                 | <ul style="list-style-type: none"> <li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a reduce operation.</li> </ul>  |
|            | <pre>def reduce[R: TypeInformation] (preAggregator: (T, T) =&gt; T, windowFunction: (K, W, Iterable[T], Collector[R]) =&gt; Unit): DataStream[R]</pre>                                            |                                                                                                                                                                  |
|            | <pre>def fold[R: TypeInformation] (initialValue: R, function: FoldFunction[T,R]): DataStream[R]</pre>                                                                                             | Apply a fold function to the window and return the result.                                                                                                       |
|            | <pre>def fold[R: TypeInformation] (initialValue: R)(function: (R, T) =&gt; R): DataStream[R]</pre>                                                                                                | <ul style="list-style-type: none"> <li>• <b>initialValue</b> indicates the initial value.</li> <li>• <b>foldFunction</b> indicates the fold function.</li> </ul> |
|            | <pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: FoldFunction[T, ACC], function: WindowFunction[ACC, R, K, W]): DataStream[R]</pre>                       | <ul style="list-style-type: none"> <li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a fold operation.</li> </ul>    |
|            | <pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: (ACC, T) =&gt; ACC, windowFunction: (K, W, Iterable[ACC], Collector[R]) =&gt; Unit): DataStream[R]</pre> |                                                                                                                                                                  |
| Window All | <pre>def apply[R: TypeInformation] (function: AllWindowFunction[T, R, W]): DataStream[R]</pre>                                                                                                    | Apply a general function to a window. The data in the window is calculated as a whole.                                                                           |

| Method | API                                                                                                                                                                                              | Description                                                                                                                                                                                    |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | def apply[R: TypeInformation]<br>(function: (W, Iterable[T],<br>Collector[R]) => Unit):<br>DataStream[R]                                                                                         |                                                                                                                                                                                                |
|        | def reduce(function:<br>ReduceFunction[T]):<br>DataStream[T]                                                                                                                                     | Apply a reduce function to the<br>window and return the result.<br><ul style="list-style-type: none"> <li>• <b>reduceFunction</b> indicates the<br/>reduce function to be executed.</li> </ul> |
|        | def reduce(function: (T, T) =><br>T): DataStream[T]                                                                                                                                              |                                                                                                                                                                                                |
|        | def reduce[R:<br>TypeInformation]<br>(preAggregator:<br>ReduceFunction[T],<br>windowFunction:<br>AllWindowFunction[T, R, W]):<br>DataStream[R]                                                   |                                                                                                                                                                                                |
|        | def reduce[R:<br>TypeInformation]<br>(preAggregator: (T, T) => T,<br>windowFunction: (W,<br>Iterable[T], Collector[R]) =><br>Unit): DataStream[R]                                                |                                                                                                                                                                                                |
|        | def fold[R: TypeInformation]<br>(initialValue: R, function:<br>FoldFunction[T,R]):<br>DataStream[R]                                                                                              | Apply a fold function to the window<br>and return the result.<br><ul style="list-style-type: none"> <li>• <b>initialValue</b> indicates the initial<br/>value.</li> </ul>                      |
|        | def fold[R: TypeInformation]<br>(initialValue: R)(function: (R,<br>T) => R): DataStream[R]                                                                                                       | <ul style="list-style-type: none"> <li>• <b>foldFunction</b> indicates the fold<br/>function.</li> </ul>                                                                                       |
|        | def fold[ACC:<br>TypeInformation, R:<br>TypeInformation](initialValue:<br>ACC, preAggregator:<br>FoldFunction[T, ACC],<br>windowFunction:<br>AllWindowFunction[ACC, R,<br>W]): DataStream[R]     | <ul style="list-style-type: none"> <li>• <b>function of WindowFunction</b><br/>indicates triggering an operation<br/>to the window after a fold<br/>operation.</li> </ul>                      |
|        | def fold[ACC:<br>TypeInformation, R:<br>TypeInformation](initialValue:<br>ACC, preAggregator: (ACC, T)<br>=> ACC, windowFunction: (W,<br>Iterable[ACC], Collector[R])<br>=> Unit): DataStream[R] |                                                                                                                                                                                                |

| Method                         | API                                        | Description                                                                                                                                                                        |
|--------------------------------|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Window<br>and<br>Window<br>All | def sum(position: Int):<br>DataStream[T]   | Sum a specified column of the window data.<br><b>field</b> and <b>position</b> indicate a specific column of the data.                                                             |
|                                | def sum(field: String):<br>DataStream[T]   |                                                                                                                                                                                    |
|                                | def min(position: Int):<br>DataStream[T]   | Calculate the minimum value of a specified column of the window data. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values. |
|                                | def min(field: String):<br>DataStream[T]   | <b>position</b> and <b>field</b> indicate calculating the minimum value of a specific column.                                                                                      |
|                                | def max(position: Int):<br>DataStream[T]   | Calculate the maximum value of a specified column of the window data. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values. |
|                                | def max(field: String):<br>DataStream[T]   | <b>position</b> and <b>field</b> indicate calculating the maximum value of a specific column.                                                                                      |
|                                | def minBy(position: Int):<br>DataStream[T] | Obtain the row where the minimum value of a column locates in the window data. <b>minBy</b> returns all elements of that row.                                                      |
|                                | def minBy(field: String):<br>DataStream[T] | <b>position</b> and <b>field</b> indicate the column on which the <b>minBy</b> operation is performed.                                                                             |
|                                | def maxBy(position: Int):<br>DataStream[T] | Obtain the row where the maximum value of a column locates in the window data. <b>maxBy</b> returns all elements of that row.                                                      |
|                                | def maxBy(field: String):<br>DataStream[T] | <b>position</b> and <b>field</b> indicate the column on which the <b>maxBy</b> operation is performed.                                                                             |

## Combining Multiple DataStreams

**Table 1-61** APIs about combining multiple DataStreams

| API                                                                                                                                     | Description                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>def union(dataStreams: DataStream[T]*): DataStream[T]</code>                                                                      | Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams.<br><b>NOTE</b><br>If you perform union operation on a piece of data with itself, there are two copies of the same data. |
| <code>def connect[T2] (dataStream: DataStream[T2]): ConnectedStreams[T, T2]</code>                                                      | Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.                                                  |
| <code>def map[R: TypeInformation] (coMapper: CoMapFunction[IN1, IN2, R]): DataStream[R]</code>                                          | Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.                                                                         |
| <code>def map[R: TypeInformation](fun1: IN1 =&gt; R, fun2: IN2 =&gt; R): DataStream[R]</code>                                           |                                                                                                                                                                                                                                                      |
| <code>def flatMap[R: TypeInformation] (coFlatMapper: CoFlatMapFunction[IN1, IN2, R]): DataStream[R]</code>                              | Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams.<br><b>NOTE</b><br>If you perform union operation on a piece of data with itself, there are two copies of the same data. |
| <code>def flatMap[R: TypeInformation](fun1: (IN1, Collector[R]) =&gt; Unit, fun2: (IN2, Collector[R]) =&gt; Unit): DataStream[R]</code> |                                                                                                                                                                                                                                                      |
| <code>def flatMap[R: TypeInformation] (fun1: IN1 =&gt; TraversableOnce[R], fun2: IN2 =&gt; TraversableOnce[R]): DataStream[R]</code>    |                                                                                                                                                                                                                                                      |

## Join Operation

**Table 1-62** APIs about join operation

| API                                                                                                 | Description                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>def join[T2]<br/>(otherStream:<br/>DataStream[T2]):<br/>JoinedStreams[T, T2]</code>           | Join two DataStreams using a given key in a specified window.<br>The key value of the join operation is specified by the <b>where</b> and <b>equalTo</b> method, indicating filtering data with equivalent conditions from two DataStreams. |
| <code>def coGroup[T2]<br/>(otherStream:<br/>DataStream[T2]):<br/>CoGroupedStreams[T,<br/>T2]</code> | Co-group two DataStreams using a given key in a specified window.<br>The key value of the coGroup operation is specified by the where and equalTo method, indicating partitioning two DataStreams using equivalent conditions.              |

### 1.8.5.2 Overview of RESTful APIs

Flink has a monitoring API that can be used to query status and statistics of running jobs, as well as recent completed jobs. This monitoring API is used by Apache Flink Dashboard.

The monitoring API is a RESTful API that accepts HTTP GET requests and responds with JSON data. RESTful API is a set of APIs used to log in to the web server. In Flink, web server is a module of JobManager and shares the same process with JobManager. By default, the listening port of web server is 8081. If you want to change the listening port, modify **jobmanager.web.port** in the **flink-conf.yaml** file.

The *Netty* and the *Netty Router* library are used to handle REST requests and analysis URLs.

RESTful APIs are executed through HTTP requests.

The format of the HTTP requests is `http://<JobManager_IP>:<JobManager_Port><Path>`

The *JobManager\_IP* indicates the IP address of JobManager, *JobManager\_Port* indicates the listening port of JobManager, and *Path* indicates the path. For details, see [Table 1-63](#). For example, `http://10.162.181.57:32261/config`.

#### NOTE

If you want to modify the configuration file **flink-conf.yaml** of the Flink Client, add to-be-visited IP addresses (separated with commas) in **jobmanager.web.allow-access-address** and **jobmanager.web.access-control-allow-origin** parameters.

[Table 1-63](#) lists all RESTful API paths supported by Flink.

**Table 1-63** Paths supported by Flink

| Path                              | Description                                                                                                                                                                                           |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /config                           | Some information about the monitoring API and the server setup.                                                                                                                                       |
| /logout                           | Some information about the logout.                                                                                                                                                                    |
| /overview                         | Simple summary of the Flink cluster status.                                                                                                                                                           |
| /jobs                             | IDs of the jobs, grouped by status <i>running</i> , <i>finished</i> , <i>failed</i> , <i>canceled</i> .                                                                                               |
| /jobmanager/config                | the configuration of the jobmanager.                                                                                                                                                                  |
| /joboverview                      | Jobs, grouped by status, each with a small summary of its status.                                                                                                                                     |
| /joboverview/running              | Jobs, grouped by status, each with a small summary of its status. The same as <a href="#">/joboverview</a> , but containing only currently running jobs.                                              |
| /joboverview/completed            | Jobs, grouped by status, each with a small summary of its status. The same as <a href="#">/joboverview</a> , but containing only completed (finished, canceled, or failed) jobs.                      |
| /jobs/<jobid>                     | Summary of one job, listing dataflow plan, status, timestamps of state transitions, aggregate information for each vertex (operator).                                                                 |
| /jobs/<jobid>/vertices            | Currently the same as <a href="#">/jobs/&lt;jobid&gt;</a> .                                                                                                                                           |
| /jobs/<jobid>/config              | The user-defined execution config used by the job.                                                                                                                                                    |
| /jobs/<jobid>/exceptions          | The non-recoverable exceptions that have been observed by the job. The truncated flag defines whether more exceptions occurred, but are not listed, because the response would otherwise get too big. |
| /jobs/<jobid>/accumulators        | The aggregated user accumulators plus job accumulators.                                                                                                                                               |
| /jobs/<jobid>/checkpoints         | checkpoint stats for a job.                                                                                                                                                                           |
| /jobs/<jobid>/metrics             | a job a list of all available metrics.                                                                                                                                                                |
| /jobs/<jobid>/vertices/<vertexid> | Information about one specific vertex, with a summary for each of its subtasks.                                                                                                                       |

| Path                                                                                    | Description                                                                                                                                                                                                      |
|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /jobs/<jobid>/vertices/<vertexid>/subtasktimes                                          | This request returns the timestamps for the state transitions of all subtasks of a given vertex. These can be used, for example, to create time-line comparisons between subtasks.                               |
| /jobs/<jobid>/vertices/<vertexid>/taskmanagers                                          | TaskManager statistics for one specific vertex. This is an aggregation of subtask statistics returned by /jobs/<jobid>/vertices/<vertexid>.                                                                      |
| /jobs/<jobid>/vertices/<vertexid>/accumulators                                          | The aggregated user-defined accumulators, for a specific vertex.                                                                                                                                                 |
| /jobs/<jobid>/vertices/<vertexid>/checkpoints                                           | checkpoint stats for a single job vertex.                                                                                                                                                                        |
| /jobs/<jobid>/vertices/<vertexid>/backpressure                                          | back pressure stats for a single job vertex and all its sub tasks.                                                                                                                                               |
| /jobs/<jobid>/vertices/<vertexid>/metrics                                               | a given task of the values for a set of metrics.                                                                                                                                                                 |
| /jobs/<jobid>/vertices/<vertexid>/subtasks/accumulators                                 | Gets all user-defined accumulators for all subtasks of a given vertex. These are the individual accumulators that are returned in aggregated form by the request /jobs/<jobid>/vertices/<vertexid>/accumulators. |
| /jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>                                 | Summary of the current or latest execution attempt of a specific subtask. See below for a sample.                                                                                                                |
| /jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>              | Summary of a specific execution attempt of a specific subtask. Multiple execution attempts happen in case of failure/recovery.                                                                                   |
| /jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>/accumulators | The accumulators collected for one specific subtask during one specific execution attempt (multiple attempts happen in case of failure/recovery).                                                                |
| /jobs/<jobid>/plan                                                                      | The dataflow plan of a job. The plan is also included in the job summary (/jobs/<jobid>).                                                                                                                        |
| /taskmanagers                                                                           | Some information of the TaskManagers.                                                                                                                                                                            |
| /taskmanagers/<taskmanagerid>/metrics                                                   | the metrics information of a TaskManager.                                                                                                                                                                        |

| Path                                 | Description                                                                       |
|--------------------------------------|-----------------------------------------------------------------------------------|
| /taskmanagers/<taskmanagerid>/log    | the log information of a TaskManager.                                             |
| /taskmanagers/<taskmanagerid>/stdout | the stdout of a TaskManager.                                                      |
| /jobmanager/log                      | the log of Jobmanager.                                                            |
| /jobmanager/stdout                   | the stdout of Jobmanager.                                                         |
| /jobmanager/metrics                  | the metrics of Jobmanager.                                                        |
| /*                                   | services requests to web frontend's static files, such as HTML, CSS, or JS files. |

**Table 1-64** describes variables listed in **Table 1-63**.

**Table 1-64** Description of variables

| Variable      | Description                      |
|---------------|----------------------------------|
| jobid         | ID of jobs                       |
| vertexid      | Vertices ID of the flow diagram. |
| subtasknum    | Sum of subtasks.                 |
| attempt       | Times of attempts                |
| taskmanagerid | ID of TaskManager                |

### 1.8.5.3 Overview of Savepoints CLI

#### Overview

Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. They use Flink's checkpoint mechanism to create a snapshot of the state of your streaming program and write the checkpoint meta data out to an external file system.

It is highly recommended that you adjust your programs as described in this section in order to be able to upgrade your programs in the future. The main required change is to manually specify operator IDs via the **uid(String)** method. These IDs are used to scope the state of each operator.

```
DataStream<String> stream = env
//Statefulesource(e.g.Kafka)withID
.addSource(new StatefulSource())
.uid("source-id") //IDforthesourceoperator
.shuffle()
//StatefulemapperwithID
.map(new StatefulMapper())
```

```
.uid("mapper-id") //IDforthemapper  
//Statelessprintingsink  
.print(); //Auto-generatedID
```

## Savepoint Recovery

If you do not specify the IDs manually, they will be generated automatically. You can automatically restore from the savepoint if these IDs do not change. The generated IDs depend on the structure of your program and are sensitive to program changes. Therefore, it is highly recommended to assign these IDs manually. When a savepoint is triggered, a single savepoint file will be created containing the checkpoint metadata. The actual checkpoint state will be kept around in the configured checkpoint directory, for example, with a FsStateBackend or RocksDBStateBackend:

1. Trigger a savepoint.

```
$ bin/flink savepoint jobId [targetDirectory]
```

This command will trigger a savepoint for the job with ID:*jobid*. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager. The targetDirectory is optional. If targetDirectory is not configured, the directory specified by **state.savepoints.dir** in the configuration file is used to store savepoint.

You can configure a default savepoint target directory via the **state.savepoints.dir** key in the **flink-conf.yaml** file.

```
# Default savepoint target directory
```

### NOTE

You are advised to configure targetDirectory to an HDFS path.

For example:

```
bin/flink savepoint 405af8c02cf6dc069a0f9b7a1f7be088 hdfs://savepoint.
```

2. Cancel a job with a savepoint.

```
$ bin/flink cancel -s [targetDirectory] jobId
```

This will atomically trigger a savepoint for the job with ID:*jobid* and cancel the job. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager.

3. Resume jobs.

- Resume from a savepoint.

```
$ bin/flink run -s savepointPath [runArgs]
```

This command submits a job and specifies the savepoint path. The execution will resume from the respective savepoint state.

### NOTE

**runArgs** is a user-defined parameter with parameter format and name varying depending on users.

- Allow non-restored state.

```
$ bin/flink run -s savepointPath -n [runArgs]
```

By default the resume operation will try to map all state of the savepoint back to the program you are restoring with. If you dropped an operator, you can skip the state that cannot be mapped to the new program via -allowNonRestoredState (short: -n).

4. Dispose savepoints.

```
$ bin/flink savepoint -d savepointPath
```

This command disposes the savepoint stored in: *savepointPath*.

## Precautions

- Chained operators are identified by the ID of the first task. It is not possible to manually assign an ID to an intermediate chained task, for example, in the chain [ a -> b -> c ] only **a** can have its ID assigned manually, but not **b** or **c**. To work around this, you can manually define the task chains. To manually define chains, use the **disableChaining()** interface. See the following example:

```
env.addSource(new GetDataSource())
.keyBy(0)
.timeWindow(Time.seconds(2)).uid("window-id")
.reduce(_+_).uid("reduce-id")
.map(f->(f1)).disableChaining().uid("map-id")
.print().disableChaining().uid("print-id")
```
- During job upgrade, the data type of operators cannot be changed.

### 1.8.5.4 Introduction to Flink Client CLI

#### Common CLIs

Common Flink CLIs are as follows:

- yarn-session.sh**
  - You can run **yarn-session.sh** to start a standing Flink cluster to receive tasks submitted by clients. Run the following command to start a Flink cluster with three TaskManager instances:

```
bin/yarn-session.sh
```
  - Run the following command to obtain other parameters of **yarn-session.sh**:

```
bin/yarn-session.sh -help
```
- flink**
  - You can run Flink commands to submit a Flink job to a standing Flink cluster or to execute the job in single-server mode.
    - Run the following command to submit a Flink job to a standing Flink cluster:

```
bin/flink run ..//examples/streaming/WindowJoin.jar
```

#### NOTE

Before using the command to submit a task, you need to run **yarn-session.sh** to start the Flink cluster.

- Run the following command to execute a per-job YARN Cluster mode:

```
bin/flink run -m yarn-cluster ..//examples/streaming/WindowJoin.jar
```

#### NOTE

The **-m** **yarn-cluster** parameter is used to specify a job to independently start a Flink cluster.

- List scheduled and running jobs (including their JobIDs):

```
bin/flink list
```

- Cancel a job:  
`bin/flink cancel <jobID>`
- Stop a job (streaming jobs only):  
`bin/flink stop <jobID>`

 NOTE

The difference between cancelling and stopping a (streaming) job is the following:

- Cancel a job: On a cancel call, the operators in a job immediately receive a cancel() method call to cancel them as soon as possible. If operators are not stopping after the cancel call, Flink will start interrupting the thread periodically until it stops.
  - Stop a job: Stop is only available for jobs which use sources that implement the `StoppableFunction` interface. The job will keep running until all sources properly shut down. Stop a job is more graceful than cancel, but it may cause the job to stop failing.
- Run the following command to obtain other parameters of Flink commands:  
`bin/flink --help`

## Precautions

- If `yarn-session.sh` uses `-z` to configure the specified ZooKeeper NameSpace, you need to use `-yid` to specify the application ID and use `-yz` to specify the ZooKeeper NameSpace when using `flink run`. The NameSpaces must the same.

Example:

```
bin/yarn-session.sh -z YARN101  
bin/flink run -yid application_****_*** -yz YARN101 examples/streaming/WindowJoin.jar
```

- If `yarn-session.sh` does not use `-z` to configure the specified ZooKeeper NameSpace, do not use `-yz` to specify the ZooKeeper NameSpace when using `flink run`.

Example:

```
bin/yarn-session.sh  
bin/flink run examples/streaming/WindowJoin.jar
```

- You can use `-yz` to specify a ZooKeeper NameSpace when using `flink run -m yarn-cluster` to start a cluster.
- A NameSpace cannot be shared by multiple clusters.
- If you use `-z` to specify a ZooKeeper NameSpace when starting a cluster or submitting a job, you need to use `-z` again to specify the NameSpace when deleting, stopping, or querying the job or triggering the savepoint.

## 1.8.5.5 FAQ

### 1.8.5.5.1 Savepoints-related Problems

1. Should I assign IDs to all operators of the job?

Strictly speaking, you can assign IDs to only operators with statuses because savepoints save only statuses of operators that have statuses and not operators without statuses.

However, in actual situations, you are advised to allocate IDs to all operators because some internal operators, such as window operators of Flink have statuses. Whether an operator has status or not is not obvious. If you are specific that an operator does not have status, you do not need to call `uid()` to allocate an ID to the operator.

2. What would be the impact if I add an operator with status while upgrading the job?

If you add an operator with status to the job, the status of the operator is not saved in the savepoint and thus the status cannot be recovered. The operator is processed as an operator without status and is executed from the start.

3. What would be the impact if I delete an operator with status while upgrading the job?

By default, savepoints attempt to recover all saved statuses. If the savepoint saves the status of the deleted operator, recovery fails.

You can run the following command and use the `-allowNonRestoredState (-n` in the following command) parameter to skip recovering the status of the deleted operator:

```
$ bin/flink run -s savepointPath -n [runArgs]
```

4. What would be the impact if I rearrange the sequence of operators with statuses?

- If you have allocated IDs to the operators, the statuses would be recovered normally.
- If you do not allocate IDs to the operators, IDs would be automatically allocated to the operators in the new sequence. Then, the status recovery would fail.

5. What would be the impact if I delete or add an operator without status or rearrange the sequence of operators without statuses?

- If you have allocated IDs to operators with statuses, operators without statuses do not affect status recovery from savepoints.
- If you do not allocate IDs to operators, operators with statuses may be allocated with new IDs due to the sequence change. This would cause status recovery failure.

6. What would be the impact if I change the operator concurrency during the status recovery?

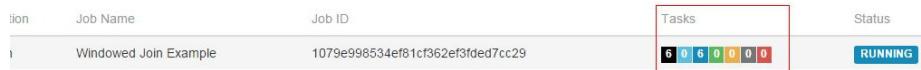
If the Flink version is higher than 1.2.0 and discarded status APIs, such as `checkpointed`, are not used, you can recover statuses from savepoints. Otherwise, statuses cannot be recovered.

### 1.8.5.5.2 What If the Chrome Browser Cannot Display the Title

#### Question

What to do if the title is not displayed when I use Chrome browser to access the Apache Flink Dashboard? This section takes the Tasks field as an example. When the pointer is placed on a colorful box in Tasks, the title of the box is not displayed, as shown in [Figure 1-100](#). [Figure 1-101](#) shows the normal situation when the title is displayed.

**Figure 1-100** Title not displayed on the page



**Figure 1-101** Title displayed normally



## Answer

If the Chrome browser does not display the title, perform the following procedure:

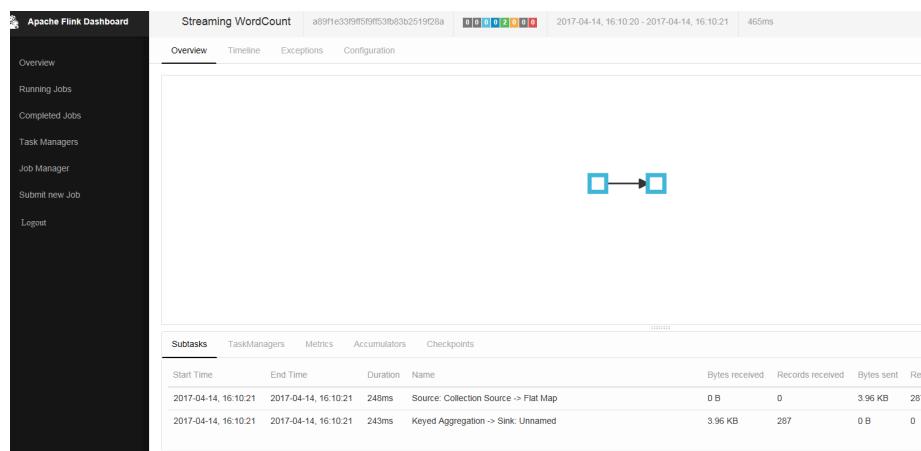
Check whether a tool that affects the tooltip display by the Chrome browser is running. If so, shut down the tool.

### 1.8.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11

## Question

What to do if Internet Explorer 10/11 does not display operator texts, as shown in [Figure 1-102](#)?

**Figure 1-102** Page displayed abnormally on Internet Explorer 10/11



## Answer

Flink uses the `foreignObject` element to draw scalable vector graphics (SVGs) but Internet Explorer 10/11 does not support `foreignObject` and thus operators cannot be displayed normally. Google Chrome browser is recommended.

#### 1.8.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large

##### Question

What to do if checkpoint is executed slowly in RocksDBStateBackend mode when the data amount is large?

##### Cause Analysis

Customized windows are used and the window state is ListState. There are many values under the same key. In the case of a new value, the merge operation of RocksDB is used. When calculation is triggered, all values under the key are read.

- The RocksDB mode is `merge() > merge() ... > merge() > read()`. Data reading in this mode consumes much time, as shown in [Figure 1-103](#).
- The source operator sends a large amount of data in a short period of time and the data keys are the same. The window operator fails to process data fast enough and barriers accumulate in the cache. The time consumed for snapshot preparation is too long and the window operator cannot report snapshot completion to CheckpointCoordinator in the specified time. Therefore, CheckpointCoordinator determines snapshot preparation failure, as shown in [Figure 1-104](#).

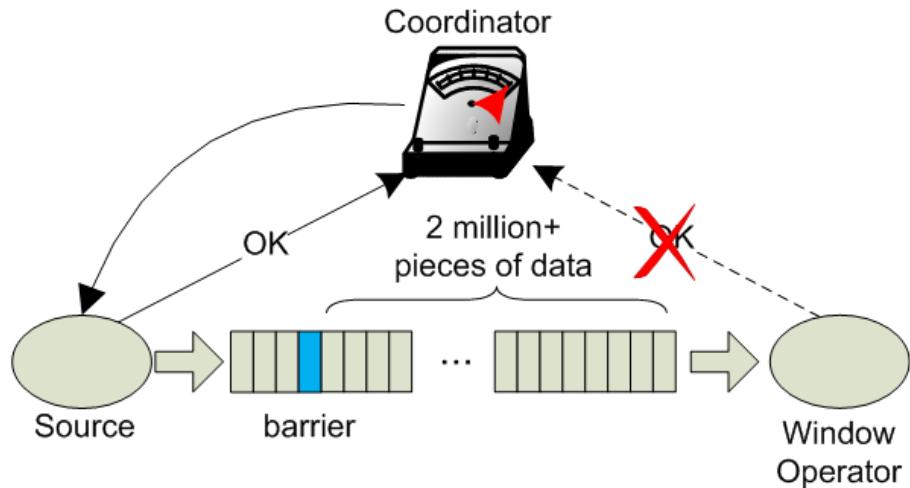
Figure 1-103 Time monitoring

```
send count: 200000 at:1489402491794
send count: 400000 at:1489402491874
send count: 600000 at:1489402491923
send count: 800000 at:1489402491963
send count: 1000000 at:1489402492006
send count: 1200000 at:1489402492045
send count: 1400000 at:1489402492094
send count: 1600000 at:1489402492134
send count: 1800000 at:1489402492173
send count: 2000000 at:1489402492260
=====
Begin get at:1489402493099
=====
End get at: 1489402534548
=====
End iterator
((200000,400000),1489402534606)
snap in source
send count: 2200000 at:1489402535363
=====
Begin get at:1489402535775
=====
End get at: 1489402577386
=====
End iterator
((400000,800000),1489402577414)
send count: 2400000 at:1489402577795
=====
Begin get at:1489402578462
=====
End get at: 1489402619442
=====
End iterator
((600000,1200000),1489402619463)
send count: 2600000 at:1489402619930
=====
Begin get at:1489402620571
=====
End get at: 1489402660263
=====
End iterator
((800000,1600000),1489402660282)
send count: 2800000 at:1489402660838
=====
Begin get at:1489402661316
=====
End get at: 1489402702431
=====
End iterator
((1000000,2000000),1489402702450)
```

2 million pieces of data are sent within 466 ms

40S+  
39.7s is spent on reading data from RocksDB

Figure 1-104 Relationship



## Answer

Flink introduces the third-party software package RocksDB, whose defect causes the problem. You are advised to set checkpoint to FsStateBackend mode.

Set checkpoint to FsStateBackend mode in the application code as follows:

```
env.setCheckpointInterval(1000);
```

### 1.8.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home

## Question

When **blob.storage.directory** is set to **/home**, the **blobStore-UUID** file cannot be created in **/home**. This causes yarn-session start failure

## Answer

**Step 1** It is recommended that **blob.storage.directory** be set to **/tmp** or **/opt/huawei/Bigdata/tmp**.

**Step 2** When you set **blob.storage.directory** to a customized directory, manually grant permissions to the directory. This section takes the **admin** user of FusionInsight as an example.

1. Modify **conf/flink-conf.yaml** on the Flink client and run the **blob.storage.directory: /home/testdir/testdirdir/xxx** command.
2. Create the **/home/testdir** directory (level 1 is enough) and set the directory to be managed by the **admin** user.

```
SZV1000064084:/home # id admin  
uid=20000(admin) gid=9998(ficcommon) groups=9998(ficcommon),8003(SystemAdministrator_186)  
SZV1000064084:/home # chown admin:ficcommon testdir/ -R
```

### NOTE

The **testdirdir/xxx** directory under the **/home/testdir/** directory is automatically created on each node when Flink cluster starts.

3. Run `./bin/yarn-session.sh -jm 2048 -tm 3072` on the client to check that yarn-session is normally started and the directory is successfully created.

```
SV1000064084:/home # ll testdir/
total 4
drwxr-x--- 3 admin ficommon 4096 Mar 13 11:55 testdirdir
SV1000064084:/home # ll testdir/testdirdir/
total 4
drwxr-x--- 4 admin ficommon 4096 Mar 13 11:55 xxx
SV1000064084:/home # ll testdir/testdirdir/xxx/
total 8
drwxr-x--- 2 admin ficommon 4096 Mar 13 11:55 blobStore-6fb3f049-ecf3-49ac-9fc9-95ad0aeeffd3
drwxr-x--- 2 admin ficommon 4096 Mar 13 11:55 blobStore-ad89b118-8545-4ece-8cae-1334b01de857
```

----End

### 1.8.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?

#### Question

After Flink kernel is upgraded to 1.3.0 or later versions, if Kafka calls the FlinkKafkaProducer010 that contains the non-static KafkaPartitioner class object as parameter to construct functions, an error is reported.

The error message is as follows:

```
org.apache.flink.api.common.InvalidProgramException: The implementation of the FlinkKafkaPartitioner is not serializable. The object probably contains or references non serializable fields.
```

#### Answer

In the 1.3.0 version of Flink, the FlinkKafkaDelegatePartitioner class is added, so that Flink allows APIs that use KafkaPartitioner, for example, FlinkKafkaProducer010 that contains KafkaPartitioner object, to construct functions.

The FlinkKafkaDelegatePartitioner class defines the member variable kafkaPartitioner.

```
private final KafkaPartitioner<T> kafkaPartitioner;
```

When Flink input parameter KafkaPartitioner constructs FlinkKafkaProducer010, the call stack is as follows:

```
FlinkKafkaProducer010(String topicId, KeyedSerializationSchema<T> serializationSchema, Properties
producerConfig, KafkaPartitioner<T> customPartitioner)
-> FlinkKafkaProducer09(String topicId, KeyedSerializationSchema<IN> serializationSchema, Properties
producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
---> FlinkKafkaProducerBase(String defaultTopicId, KeyedSerializationSchema<IN> serializationSchema,
Properties producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
-----> ClosureCleaner::clean(Object func, boolean checkSerializable)
```

Run the KafkaPartitioner object to construct a FlinkKafkaDelegatePartitioner object, and then check whether the object can be serializable. The ClosureCleaner::clean function is a static function. If the KafkaPartitioner object in a case is non-static, the ClosureCleaner::clean function cannot access the non-static member variable kafkaPartitioner in the KafkaDelegatePartitioner class and an exception is reported.

Either of the following methods can be used to solve the problem:

- Change the KafkaPartitioner class into static class.
- Use the FlinkKafkaProducer010 that contains FlinkKafkaPartitioner as the parameter to construct functions. In this case, FlinkKafkaDelegatePartitioner is not constructed and the exception about member variable is avoided.

### 1.8.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?

#### Question

When I use a newly-created Flink user to submit tasks, the task submission fails because of insufficient permission on the ZooKeeper directory. The error message in the log is as follows:

```
NoAuth for /flink_base/flink/application_1499222480199_0013
```

#### Answer

1. Check whether the permission on the /flink\_base directory in ZooKeeper is 'world,'anyone: cdrwa; If no, change the permission on the /flink\_base directory to 'world,'anyone: cdrwa and go to **step 2**. If yes, go to **step 2**.
2. In the configuration file of Flink, the default value of `high-availability.zookeeper.client.acl` is `creator`, indicating that only the creator of the directory has permission on it. The user created later has no access to the /flink\_base/flink directory in ZooKeeper because only the user created earlier has permission on it.

To solve the problem, perform the following operation as the newly-created user:

- a. Check the configuration file `conf/flink-conf.yaml` on the client.
- b. Modify the parameter `high-availability.zookeeper.path.root` to the corresponding ZooKeeper directory, for example, `/flink2`.
- c. Submit tasks again.

### 1.8.5.5.8 Why Cannot I Access the Apache Flink Dashboard?

#### Question

Why cannot I access the Apache Flink Dashboard through the URL: `http://IP address of JobManager:port of JobManager`.

#### Answer

The IP address of the computer you used has not been added to the whitelist of Apache Flink Dashboard. To solve this problem, modify the `conf/flink-conf.yaml` configuration file as follows:

1. Check whether the value of `jobmanager.web.ssl.enabled` is `false`. If not, set it to `false`.
2. Check whether the IP address of the computer you used has been added to the values of `jobmanager.web.access-control-allow-origin` and

**jobmanager.web.allow-access-address.** If the IP address has not been added, add it to the two parameters:

jobmanager.web.access-control-allow-origin:*IP address of the computer where the browser is installed*  
jobmanager.web.allow-access-address:*IP address of the computer where the browser is installed*

### 1.8.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?

#### Question

System.out.println is added to the Flink service codes for printing debugging information. How do I view the debugging log? How do I export service logs to a specified file to differentiate service logs from run logs?

#### Answer

All run logs of Flink are printed to the local directory of Yarn. By default, all logs are exported to **taskmanager.log** in the local directory of Yarn container. All logs generated by invoking System.out will be exported to the **taskmanager.out** file. You can perform as follows to view the logs:

1. Log in to the native Flink web page.
2. Choose **Task Managers > Logs** or **Task Managers > Stdout** on the left to view log information.

The screenshot shows the Apache Flink Dashboard. On the left, there's a sidebar with links for Overview, Running Jobs, Completed Jobs, Task Managers (which is highlighted with a red box), and Job Manager. The main area is titled 'Task Manager' and shows a 'Last Heartbeat: 2019-01-17, 11:35:47' and ' akka.tcp'. Below this, there are tabs for Metrics, Logs (which is highlighted with a red box), and Stdout. The 'Overview' section contains a table with three columns: Data Port, All Slots, and Free Slots. The values are 32391, 1, and 0 respectively.

Configure the function of printing service logs and Task Manager run logs separately:

#### NOTE

If service logs and Task Manager run logs are separately printed, service logs are not exported to the **taskmanager.log** file and cannot be viewed on the web page.

1. Modify the configuration file **logback.xml** in the **conf** directory of the client. Add the following log configuration information to the file. Modify the information in bold according to the actual situation.

```
<appender name="TEST" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>/path/test.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
        <fileNamePattern>/path/test.log.%i</fileNamePattern>
        <minIndex>1</minIndex>
        <maxIndex>20</maxIndex>
    </rollingPolicy>
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
        <maxFileSize>20MB</maxFileSize>
    </triggeringPolicy>
    <encoder>
```

```
<pattern>%d{"yyyy-MM-dd HH:mm:ss,SSS"} | %m %n</pattern>
</encoder>
</appender>

<logger name="com.huawei.bigdata.flink.examples" additivity="false">
    <level value="INFO"/>
    <appender-ref ref="TEST"/>
</logger>
```

2. Run **yarn-session.sh** to submit the task.



If the configuration file **logback.xml** contains `<file>/path/test.log</file>`, ensure that the user (configured **flink-conf.yaml**) used for running the task has the write and read permissions on the directory.

### 1.8.5.5.10 Incorrect GLIBC Version

#### Question

When **State Backend** is set to **RocksDB** for a Flink task, the following error message is displayed:

```
Caused by: java.lang.UnsatisfiedLinkError: /srv/BigData/hadoop/data1/nm/usercache/**/appcache/application_****/rocksdb-lib-****/librocksdbjni-linux64.so: /lib64/libpthread.so.0: version `GLIBC_2.12` not found (required by /srv/BigData/hadoop/**/librocksdbjni-linux64.so)
at java.lang.ClassLoader$NativeLibrary.load(Native Method)
at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1965)
at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1890)
at java.lang.Runtime.load0(Runtime.java:795)
at java.lang.System.load(System.java:1062)
at org.rocksdb.NativeLibraryLoader.loadLibraryFromJar(NativeLibraryLoader.java:78)
at org.rocksdb.NativeLibraryLoader.loadLibrary(NativeLibraryLoader.java:56)
at
org.apache.flink.contrib.streaming.state.RocksDBStateBackend.ensureRocksDBIsLoaded(RocksDBStateBackend.java:734)
... 11 more
```

#### Possible Causes

The version of the system where the task runs and the version of the system where the compilation environment locates are different, resulting in the incompatibility of GLIBC versions.

#### Troubleshooting Method

Run the **strings /lib64/libpthread.so.0 | grep GLIBC** command to check whether the GLIBC version is earlier than 2.12.

#### Procedure

If the version of the GLIBC is too early, use the file of later version (2.12) to replace **libpthread-\* .so**. (This is a link file. You need to replace only the file that is linked to this link file.)

#### References

None

# 1.9 GraphBase Development Guide

## 1.9.1 Overview

### 1.9.1.1 Application Development Overview

#### GraphBase Introduction

FusionInsight GraphBase is a distributed graph database based on HBase and Elasticsearch. It builds a property graph model for storage and provides powerful graph query, analysis, and traversal capabilities. It has the following features:

- The HBase-based distributed storage mechanism is provided to process massive data.
- The Spark distributed memory computing technology is provided to support quick data import.
- The Elasticsearch-based index mechanism is provided to quickly query data based on indexes.

#### Interface Type Introduction

- REST API Interface

REST APIs are developed using the Java language that is simple and easy. Therefore, you are advised to use the Java language to develop upper-layer applications. For details, see **GraphBase** in *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

The following table describes the functions provided by GraphBase after REST API is invoked.

Function	Description
User login and logout	This function can be used for access authentication.
Metadata read/write	This function can be used to add, delete, modify, and query metadata.
Metadata-based operations on vertices and edges	This function can be used to add, delete, modify, and query vertices and edges.
Filtering and querying relationships for a large amount of data	This function can be used for full graph query, conditional query, line expansion query, and path query for vertices and edges.
Index management	This function can be used to create and query full graph indexes and edge indexes, and query the status of asynchronous index tasks.

- Gremlin API Interface

Gremlin is the graph traversal language of Apache TinkerPop. Gremlin is a functional, data-flow language that enables users to succinctly express complex traversals on (or queries of) their application's property graph. Every Gremlin traversal is composed of a sequence of (potentially nested) steps. A step performs an atomic operation on the data stream. For details about Gremlin, visit <http://tinkerpop.apache.org/docs/3.3.2/reference/#traversal>.

Gremlin has the following three types of basic operations:

- map-step: converts objects in a data flow.
- filter-step: filters objects in a data flow.
- sideEffect-step: calculates data flows.

### 1.9.1.2 Basic Concepts

Like most graph databases, GraphBase uses property graphs for modeling. Based on the property graph models, GraphBase has the following basic concepts:

- **Vertex**: A vertex is also called a node, which is used to specify an entity object in the real world, such as a person.
- **Vertex label**: A vertex label indicates a node type that specifies the type of an entity object in the real world. Example: person. In GraphBase, a node has only one vertex label. The default value is **vertex**.
- **Edge**: An edge, also known as a relationship that is used to specify the connections between two entity objects (vertices in a graph) in the real world. For example, the relationship between two persons is friend. Edges in GraphBase are unidirectional, which starts from a vertex and ends at another. Therefore, a bidirectional edge has an incoming and an outgoing edge.
- **Edge label**: An edge label specifies the type of a relationship in the real world. For example, the relationship is friend.
- **Property**: A property describes some attribute of either a vertex or an edge in the format of key-value pair. Property key is used to describe the key in the key-value pair, property value describes a specific value. For example, a property key is **name**, the property value is **Zhang San**.

### 1.9.1.3 Development Process

This section describes how to develop a GraphBase application based on GraphBase REST API and Gremlin API.

**Figure 1-105** shows the development process, and **Table 1-65** describes each phase.

Figure 1-105 GraphBase application development process

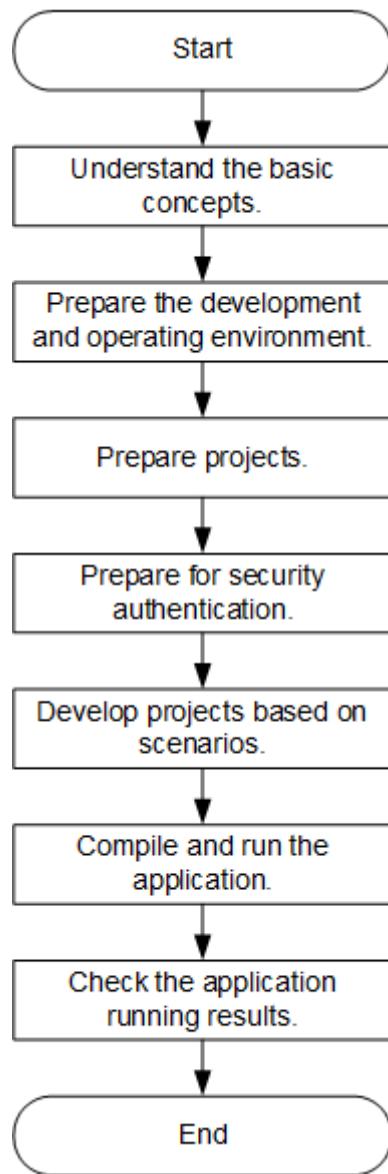


Table 1-65 GraphBase application development process

Phase	Description	Reference Document
Understand basic concepts about GraphBase.	Before application development, learn basic concepts of GraphBase, understand the scenario requirements, and design tables.	<a href="#">Basic Concepts</a>

Phase	Description	Reference Document
Prepare the development and running environment.	REST APIs and the Gremlin language are recommended for GraphBase application development. You can use the IntelliJ IDEA tool. The GraphBase running environment is the GraphBase client. Install and configure the client according to the guide.	<ul style="list-style-type: none"><li>For details about how to prepare the development environment for the REST API, see <a href="#">Preparing Development and Operating Environment</a>.</li><li>For details about how to prepare the development environment for the Gremlin API, see <a href="#">Preparing Development and Operating Environment</a>.</li></ul>
Prepare a project.	GraphBase provides sample projects for different scenarios. You can import a sample project to learn the application.	<ul style="list-style-type: none"><li>For details about how to prepare a project for the REST API, see <a href="#">Configuring and Importing Sample Projects</a>.</li><li>For details about how to prepare a project for the Gremlin API, see <a href="#">Configuring and Importing a Piece of Sample Code</a>.</li></ul>
Preparing for security authentication.	The security authentication is mandatory if security clusters are used.	<ul style="list-style-type: none"><li>For details about how to perform security authentication for the REST API, see <a href="#">Preparing for Security Authentication</a>.</li><li>For details about how to perform security authentication for the Gremlin API, see <a href="#">Preparing for Security Authentication</a>.</li></ul>

Phase	Description	Reference Document
Develop a project based on the scenario.	<ul style="list-style-type: none"><li>Sample projects for developing REST APIs are provided, including projects for creating and deleting graphs and performing full graph query, path query, and line expansion query.</li><li>The Gremlin syntax scenario is provided, including syntax scenarios for querying a specified graph object, querying one or more properties, traversing graph objects and paths.</li></ul>	<a href="#">Developing an Application</a>
Compile and run applications.	You can compile the developed application and submit it for running.	<ul style="list-style-type: none"><li><a href="#">Compiling and Running an Application</a></li><li><a href="#">Compiling and Running an Application</a></li></ul>
Check the application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	<ul style="list-style-type: none"><li><a href="#">Viewing Windows Commissioning Results</a></li><li><a href="#">Viewing Linux Commissioning Results</a></li></ul>

## 1.9.2 Environment Preparation

### 1.9.2.1 REST API Development Environment Preparation

#### 1.9.2.1.1 Preparing Development and Operating Environment

[Table 1-66](#) describes the development and operating environment required for secondary development.

**Table 1-66** Environment

Preparation Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows 7 or later</li><li>• Operating environment: Windows OS or Linux OS. If the program needs local debugging, the operating environment must be able to communicate with the cluster service plane.</li></ul>
JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA	<p>Tool used for developing GraphBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you are using an IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If you are using an Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If you are using an OpenJDK, ensure that the JDK configured in IntelliJ IDEA is the OpenJDK.</li></ul>
Maven	Basic configuration of the development environment. This tool is used for project management throughout the lifecycle of software development.

Preparation Item	Description
JUnit plug-ins	Basic configuration of the development environment.
User development	See <a href="#">Preparing a Developer Account</a> for configuration.
7-Zip	Tool used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

## Preparing the Operating Environment

During application development, prepare the environment for running and commissioning code to verify that the application can run properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required for commissioning; configure the network connection, and commission the application in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) and click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar**. Then, decompress this file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\GraphBase\config** and import the configuration file to the configuration file directory (usually the **conf** folder) of the GraphBase sample project.

The keytab file obtained during [Preparing a Developer Account](#) is also stored in this directory. **Table 1-67** describes the main configuration files.

**Table 1-67** Configuration files

File	Description
graphbase.properties	Parameters required for the GraphBase sample code
Person.xml	Files required by Schema
user.keytab	User information for Kerberos security authentication

File	Description
krb5.conf	Kerberos Server configuration

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, install the client in the **/opt/hadoopclient** directory.

The difference between the client time and the cluster time must be less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. [Accessing FusionInsight Manager of an MRS Cluster](#). Download the cluster client software package to the active management node and decompress it. Then log in to the active management node as user **root**. Go to the decompression directory of the cluster client, and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/GraphBase/config** directory to the client node to the **conf** directory where the compiled JAR package is stored, for example, **/opt/hadoopclient/conf**, for subsequent commissioning.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active OMS node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp GraphBase/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-68](#) describes the main configuration files.

**Table 1-68** Configuration files

File	Description
graphbase.properties	Parameters required for the GraphBase sample code
Person.xml	Files required by Schema
user.keytab	User information for Kerberos security authentication
krb5.conf	Kerberos Server configuration

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

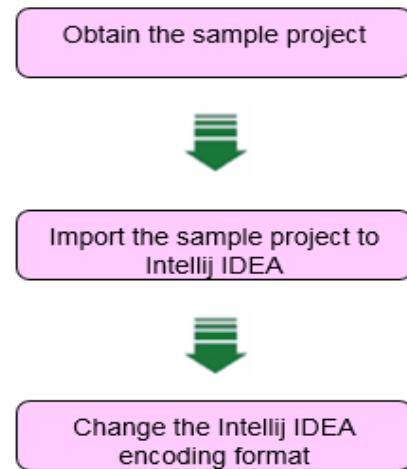
### 1.9.2.1.2 Configuring and Importing Sample Projects

#### Prerequisites

The GraphBase client has been installed.

#### Scenario

GraphBase provides sample projects for multiple scenarios to help you quickly master RESTful APIs. The following describes how to import the sample code to IntelliJ IDEA. [Figure 1-106](#) shows the operation process.

**Figure 1-106** Procedure of importing sample projects

## Procedure

- Step 1** Obtain the sample project folder **graphbase-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

Download the sample project **graphbase-core-example** to the local PC.

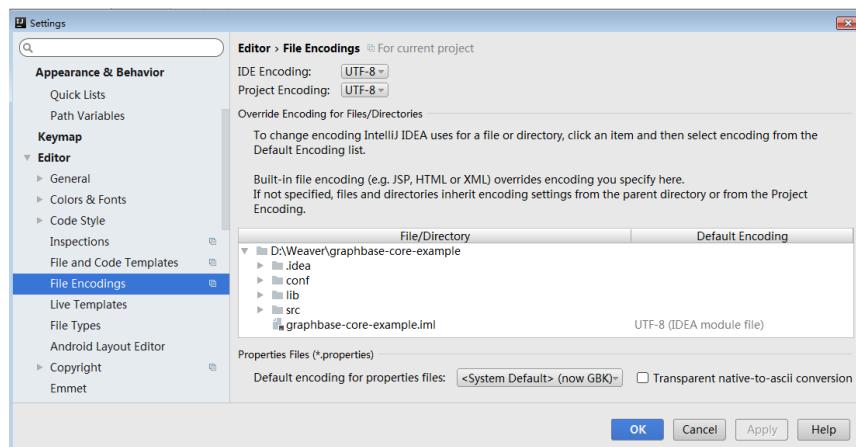
- Step 2** Import the sample project to the IntelliJ IDEA development environment.

1. Open IntelliJ IDEA and choose **File > New > Project from Existing Sources....**
2. Select the sample project folder **graphbase-core-example** and click **OK**.
3. In the **Import Project** window, select **Create project from existing sources**.
4. Click **Next** until JDK is set to jdk1.8.0\_402. Then click **Next** and **Finish**.

- Step 3** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

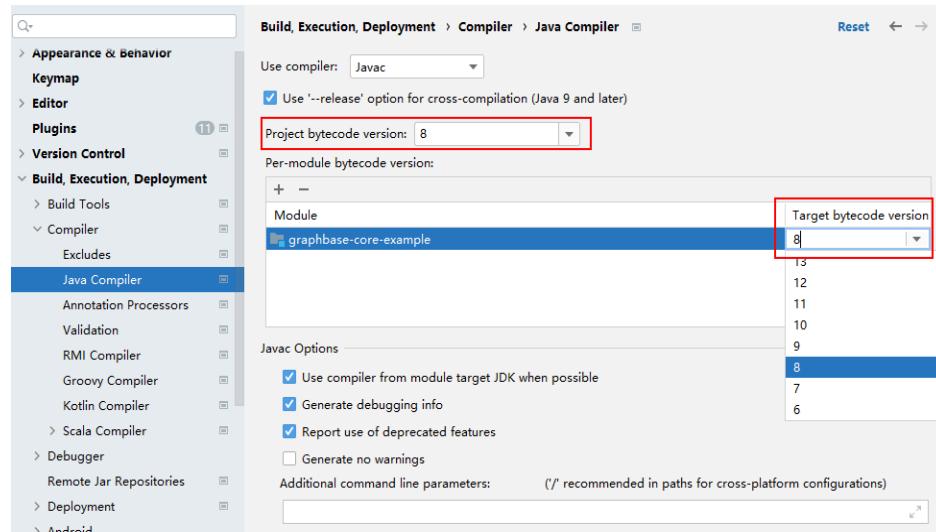
1. On the IntelliJ IDEA menu bar, choose **File > Settings....**
2. In the displayed **Settings** window, choose **Editor > File Encoding** in the navigation pane on the left, set the parameter to **UTF-8**, click **Apply**, and click **OK**.

**Figure 1-107** Setting the IntelliJ IDEA encoding format

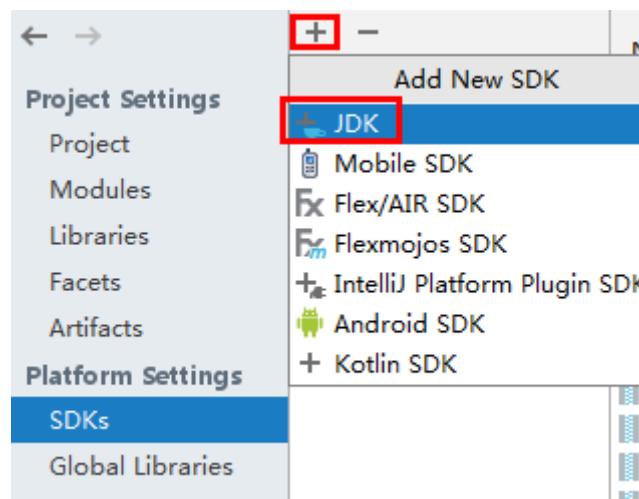


- Step 4** Set the JDK of the project.

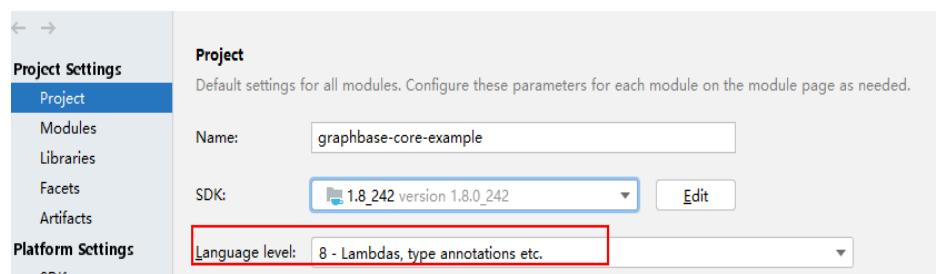
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **graphbase-core-example** to **8**.



3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.

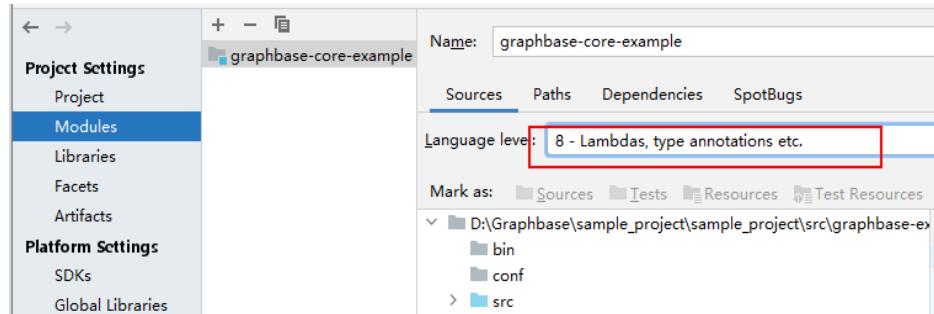


6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.

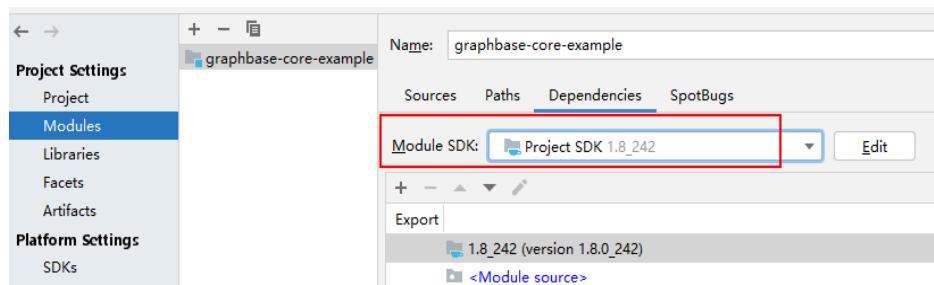


9. Click **Apply**.

- Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.



- Click **Apply** and then **OK**.

----End

### 1.9.2.1.3 Preparing for Security Authentication

GraphBase uses keytab authentication which never expires. You are advised to use keytab files for authentication on the client.

- Modify the configuration data.

- In the **conf** directory of the root directory of the sample project, set the **graphbase.properties** configuration file, IP address, port number, and username as you need.

#### NOTE

The IP address and port number of the GraphBase service is the floating IP address and port number specified when the GraphBase component is installed. Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > GraphBase > Configurations > All Configurations**, and search for **loadbalancer.floatip**. The value of **loadbalancer.floatip** is the floating IP address, the value of **loadbalancer.https.port** is the port number.

Parameter	Value
* loadbalancer.floatip	10 . 5 . 89 . 50
loadbalancer.https.port	22380

**Note:** In an IPv6 network, the floating IP address must be enclosed by square brackets, for example, [*/IPv6 address*].

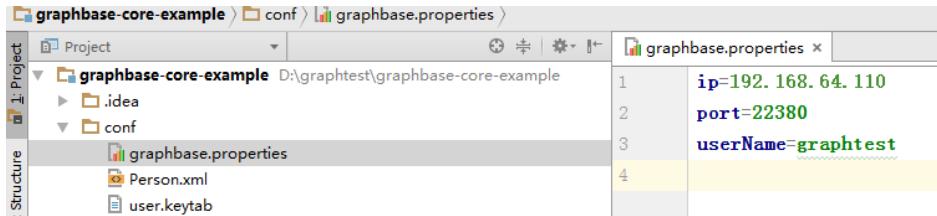
- Download the keytab file of the human-machine user and copy it to the **conf** directory in the root directory of the sample project.

Log in to FusionInsight Manager, click **System**, and choose **Permission > User**. On the displayed page, click **More > Download Authentication Credential** to obtain the **user.keytab** authentication file.

#### NOTE

The authentication credential contains the **krb5.conf** file of Kerberos. If there are multiple clusters, select the **krb5.conf** file of the corresponding cluster.

- c. Modify the configuration file as follows.



2. Obtain Keytab authentication code.

Construct an HTTP client, log in to it with Kerberos authentication, obtain CSRF TOKEN information, and call RESTful APIs. The following code snippets are in the **GraphHttpClient** class of the **com.huawei.security** package.

```
/* step 1: create http client */
final X509TrustManager trustManager = new X509TrustManager() {
    @Override
    public final X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    @Override
    public final void checkClientTrusted(X509Certificate ax509certificate[], String s) throws CertificateException {
        // to do nothing.
    }

    @Override
    public final void checkServerTrusted(X509Certificate ax509certificate[], String s) throws CertificateException {
        // to do nothing.
    }
};

final javax.net.ssl.SSLContext context = javax.net.ssl.SSLContext.getInstance("TLS");
context.init(null, new TrustManager[]{trustManager}, null);
final ThreadSafeClientConnManager connectionManager = new ThreadSafeClientConnManager();
connectionManager.setMaxTotal(100);
connectionManager.getSchemeRegistry().register(new Scheme("https", 443, new SSLSocketFactory(
    context, SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER)));

CloseableHttpClient httpclient = new DefaultHttpClient(connectionManager);

/*
 * step 2:
 * usage: kerberos login based on username and keytab,
 * note: Password never expired
 */
HttpPost loginPost = new HttpPost(httpAuthInfo.getBaseUrl() + "/login");
MultipartEntityBuilder entityBuilder = MultipartEntityBuilder.create();
entityBuilder.addPart("username", new StringBody(httpAuthInfo.getUsername(),
ContentType.create("text/plain", Consts.UTF_8)));
entityBuilder.addBinaryBody("keytabfile", new File(httpAuthInfo.getKeytabFilePath()));
loginPost.setEntity(entityBuilder.build());
loginRsp = httpclient.execute(loginPost);
RestHelper.checkHttpRsp(loginRsp);

/* step 3: get CSRF-Token */
```

```
HttpGet httpGet = new HttpGet(httpAuthInfo.getBaseUrl() + "/graph/user/me");
CloseableHttpResponse csrfTokenRsp = httpClient.execute(httpGet);
}
```

## 1.9.2.2 Gremlin API Development Environment Preparation

### 1.9.2.2.1 Preparing Development and Operating Environment

**Table 1-69** describes the development and operating environment required for secondary development.

**Table 1-69** Environment

Preparation Item	Description
OS	<ul style="list-style-type: none"><li>Development environment: Windows 7 or later</li><li>Operating environment: Windows OS or Linux OS. If the program needs local debugging, the operating environment must be able to communicate with the cluster service plane.</li></ul>
JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>

Preparation Item	Description
IntelliJ IDEA	<p>Tool used for developing GraphBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you are using an IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If you are using an Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If you are using an OpenJDK, ensure that the JDK configured in IntelliJ IDEA is the OpenJDK.</li></ul>
Maven	Basic configuration of the development environment. This tool is used for project management throughout the lifecycle of software development.
JUnit plug-ins	Basic configuration of the development environment.
User development	See <a href="#">Preparing a Developer Account</a> for configuration.
7-Zip	Tool used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

## Preparing the Operating Environment

During application development, prepare the environment for running and commissioning code to verify that the application can run properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required for commissioning; configure the network connection, and commission the application in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) and click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar**. Then, decompress this file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

    - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\GraphBase\config** and import the configuration file to the configuration file directory (usually the **conf** folder) of the GraphBase sample project.

The keytab file obtained during [Preparing a Developer Account](#) is also stored in this directory. [Table 1-70](#) describes the main configuration files.

**Table 1-70** Configuration files

File	Description
graphbase.properties	Parameters required for the GraphBase sample code
Person.xml	Files required by Schema
user.keytab	User information for Kerberos security authentication
krb5.conf	Kerberos Server configuration

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
  - The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, install the client in the **/opt/hadoopclient** directory.

The difference between the client time and the cluster time must be less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then log in to the active management node as user **root**. Go to the decompression directory of the cluster client, and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/GraphBase/config** directory to the client node to the **conf** directory where the compiled JAR package is stored, for example, **/opt/hadoopclient/conf**, for subsequent commissioning.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active OMS node, run the following commands:

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
cd FusionInsight_Cluster_1_Services_ClientConfig
```

```
scp GraphBase/config/* root@IP address of the client node:/opt/hadoopclient/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-71](#) describes the main configuration files.

**Table 1-71** Configuration files

File	Description
log4j.properties	Gremlin log related parameters
remote-objects.yaml	Detailed parameters for Gremlin query
user.keytab	User information for Kerberos security authentication
krb5.conf	Kerberos Server configuration

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

### 1.9.2.2 Configuring and Importing a Piece of Sample Code

1. Obtain the sample project folder **gremlin-demo4j** in the **graphbase-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Import the sample code to IntelliJ IDEA. For details, see [Configuring and Importing Sample Projects](#).
3. Configure the **krb5.conf** and **user.keytab** files and copy the **krb5.conf** and keytab files that you applied for to the **gremlin-demo4j\conf** directory in the sample program.

### 1.9.2.3 Preparing for Security Authentication

In a secure cluster environment, the Gremlin client and server need to authenticate each other to ensure communication security. The LoginUtil-related interfaces are provided to complete such authentication configuration. In the following sample code, you only need to configure the account name (applied by the user) and keytab file name. For details about the sample code, see the LoginUtil class in the **com.huawei.graphbase.gremlin.util** package of the sample project.

Authentication sample code:

```
/*
 * kerberos authentication
 * @throws IOException
 */
```

```
private static void krbLogin() throws IOException{
    /**
     * Account name applied by the user.
     */
    String user_principal = "Account name applied by the user";
    String krb5Path = System.getProperty("user.dir") + File.separator + "conf" + File.separator +
"krb5.conf";
    File userKrb5File = FileUtils.getFile(krb5Path);
    if (!userKrb5File.exists())
    {
        throw new RuntimeException("userKrb5File(" + userKrb5File.getAbsolutePath() + ") does not
exist.");
    }
    //Set the krb5 file.
    System.setProperty("java.security.krb5.conf", userKrb5File.getAbsolutePath());

    //Set keytab, jaasconf
    String keytab = System.getProperty("user.dir") + File.separator + "conf" + File.separator + "user.keytab";
    LoginUtil.setJaasConf("gremlinclient", user_principal, keytab);
}
```

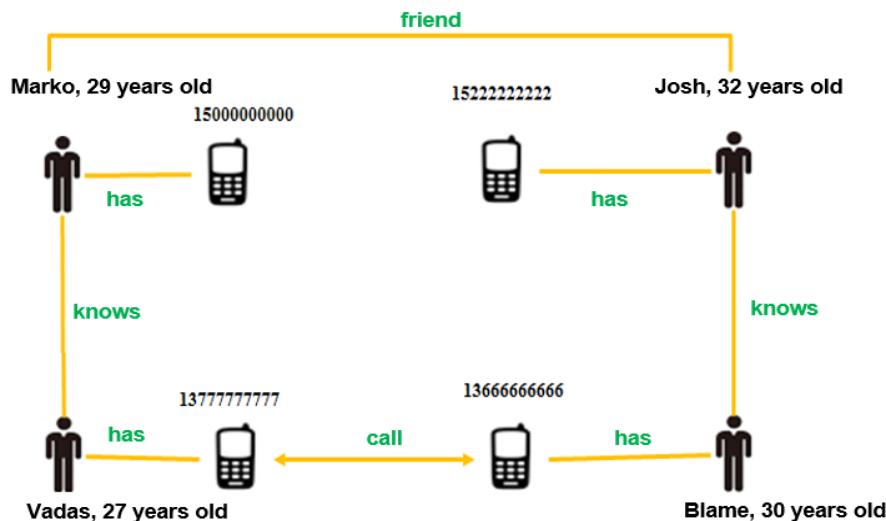
## 1.9.3 Developing an Application

### 1.9.3.1 Typical Application Scenario

You can quickly learn and master the GraphBase development process and know key interface functions in a typical application scenario.

#### Actual Scenario

Typical social relationships:



In the scenario, there are eight real world entities, four persons and four mobile phones.

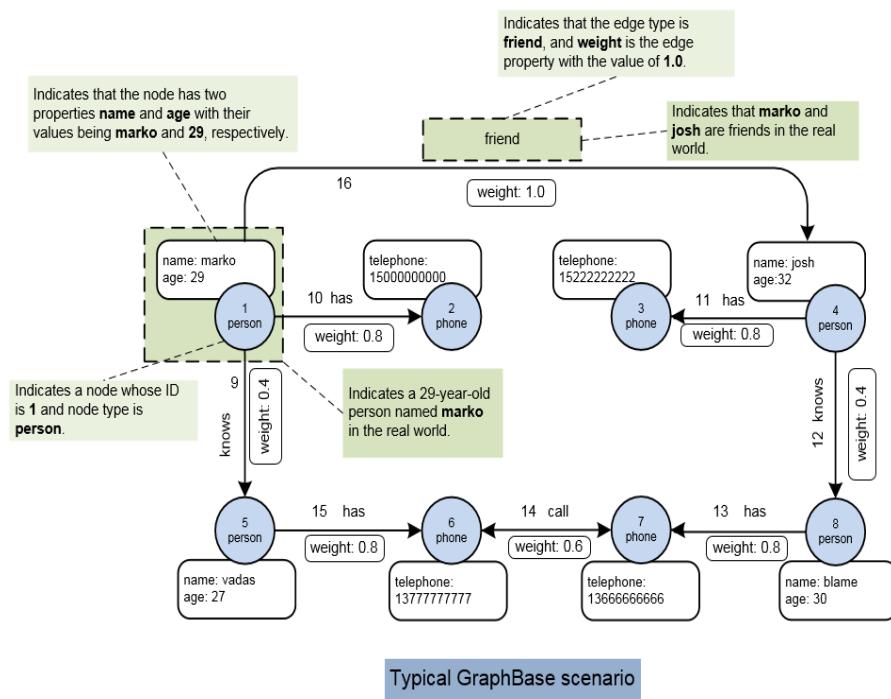
There are eight relationships between the entity objects, including **friend**, **knows**, **call**, and **has**.

## Graph Modeling Scenario

The purpose of graph modeling is to map the real world relationships to GraphBase using a graph:

- **Vertex:** maps persons and mobile phones in the real world to GraphBase using vertices.
- **Vertex label:** defines the types of persons and mobile phone in the real world in GraphBase. That is, people are labeled as **person**, and mobile phones are defined as **phone** in the graph.
- **Edge:** maps the relationships between people and people, people and mobile phones, and mobile phones and mobile phones in the real world to GraphBase using edges.
- **Edge label:** defines the relationships between entity objects in the real world in GraphBase. The definitions of these relationships in this scenario are as follows:
  - The friend relationship between people is defined as the edge label **friend**.
  - The acquaintance relationship is defined as the edge label **knows**.
  - The ownership between mobile phones and people is defined as **has**.
  - The communication relationship between mobile phones is defined as the edge label **call**.
- **Property:** defines the feature information of each entity object in the real world in GraphBase. The name and age of a person is defined as the property **name** and **age**, respectively, and the mobile phone number is defined as the property **telephone**.
- **New property:** Considering that GraphBase supports the value properties obtained by the evaluation system, the relationship weight property definition is added to indicate the importance of relationships. That is, the **weight** property is defined. It is considered that the value property has been obtained.

The graph model is as follows:



## Development Guidelines

Based on the preceding scenario description, the basic operation process of creating a GraphBase service and querying data using the created service are as follows:

- Take **node 1 > node 2** in the scenario as an example, there are three elements included: node 1, node 2, and edge 10.
  - Create a schema using the two nodes, see instructions provided in [Creating a Schema](#).  
You need to create two vertex labels, **person** and **phone**, and an edge label, **has**.  
You need to create the properties of all nodes and edges: **name**, **age**, **telephone**, and **weight**.
  - (Optional) Create indexes for properties to improve query efficiency. For details, see [Creating an Index](#).
  - Create vertices and edges.  
Create vertex 1, vertex 2, and the edge 10. For details about how to create vertices and edges, see [Creating and Querying a Vertex or an Edge](#).
  - Query data.  
Method 1: Invoke a REST API for query.  
Invoke the full graph query interface to query vertices and edges. For details, see [Performing a Full Graph Query](#).  
Method 2: Perform data query using Gremlin statements. For details, see [Gremlin Console](#), [Gremlin Java](#) and [Gremlin Application Scenarios](#).  
You can create other nodes and edges in the same way.

- If the data volume is large, you can import the data in batches. The procedure is as follows:
  - a. Create a schema.  
Compile an XML file. For details, see [Compiling a Schema File \(XML\)](#).  
Create a schema by uploading the XML file. For details, see [Creating a Schema by Uploading an XML File](#).
  - b. Import data.  
Compile a data file (CSV), data description file (DESC), and a graph mapping rule file (.mapper). For details, see [Preparing Data Files](#).  
Import data in real time or in batches. For details, see [Importing Data](#).
  - c. Query data.  
Method 1: Invoke a REST API for query.  
Invoke the full graph query interface to query vertices and edges. For details, see [Performing a Full Graph Query](#).  
Method 2: Perform data query using Gremlin statements. For details, see [Gremlin Console](#), [Gremlin Java](#) and [Gremlin Application Scenarios](#).
- During data query, except vertex and edge query, you can query paths between vertices and perform line expansion queries. For details, see [Performing a Path Query](#) and [Performing a Line Expansion Query](#).

### 1.9.3.2 Preparing Data Files

#### 1.9.3.2.1 Compiling a Schema File (XML)

##### Defining the Schema File (XML)

In GraphBase, vertices and edges are entities. You need to create the metadata information for the entity labels, such as vertex labels, edge labels, property names, property value types. Such metadata information is the schema information of a graph.

1. Define a schema file **person.xml** based on the scenario described in [Typical Application Scenario](#) to create a schema file (containing vertexLabel, edgeLabel, and propertyKey).

The following code snippets are used as an example. For complete codes, see [GraphBase.examples.person-demo](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<schema>
    <vertexLabelList>
        <vertexLabel name="person"/>
        <vertexLabel name="phone"/>
    </vertexLabelList>
    <edgeLabelList>
        <edgeLabel name="friend"/></edgeLabel>
        <edgeLabel name="knows"/></edgeLabel>
        <edgeLabel name="call"/></edgeLabel>
        <edgeLabel name="has"/></edgeLabel>
    </edgeLabelList>
    <propertyKeyList>
        <propertyKey name="name" dataType="STRING"/>
        <propertyKey name="age" dataType="INTEGER"/>
        <propertyKey name="telephone" dataType="STRING"/>
        <propertyKey name="weight" dataType="FLOAT"/>
    </propertyKeyList>
</schema>
```

```
</propertyKeyList>
<graphIndexList>
    <graphIndex name="name_index" elementCategory="VERTEX" type="COMPOSITE">
        <keyTextTypeList name="name" textType="" />
    </graphIndex>
    <graphIndex name="age_index" elementCategory="VERTEX" type="MIXED">
        <keyTextTypeList name="age" textType="DEFAULT" />
    </graphIndex>
    <graphIndex name="telephone_index" elementCategory="VERTEX" type="COMPOSITE">
        <keyTextTypeList name="telephone" textType="" />
    </graphIndex>
    <graphIndex name="weight_index" elementCategory="EDGE" type="MIXED">
        <keyTextTypeList name="weight" textType="DEFAULT" />
    </graphIndex>
</graphIndexList>
</schema>
```

- **vertexLabelList**: indicates the vertex label set. These labels are optional in the preceding XML file, and the label content can be empty.

**vertexLabel**: indicates the vertex label. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name**: indicates the vertex label name, which is mandatory.

- **edgeLabelList**: indicates the edge label list. This parameter can be missing from the preceding XML file, and the parameter value can also be left blank.

**edgeLabel**: indicates an edge label. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name**: indicates the edge label name, which is mandatory.

**primaryKeys**: indicates the primary key to which EdgeLabel are bound with. Multiple primary keys can be specified. This parameter is optional. If there are multiple edges between two vertices and they have the same primaryKeys value, the edges can be considered as the same. For example:

```
<edgeLabel name="friend">
    <primaryKeys>p1</primaryKeys>
    <primaryKeys>p2</primaryKeys>
</edgeLabel>
```

- **propertyKeyList**: indicates the property key set. This parameter can be missing from the preceding XML file, and the parameter value can also be left blank.

**propertyKey**: indicates a property key. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name**: indicates the property name, which is mandatory.

**dataType**: indicates the property value data type, which is optional. The default value is **STRING**.

- **graphIndexList**: indicates the index set. This parameter can be missing from the preceding XML file, and the parameter value can also be left blank.

**graphIndex**: indicates the index. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name**: indicates the index name, which is mandatory.

**elementCategory:** indicates the index entity type, which is mandatory. The options are **VERTEX** and **EDGE**, which is case insensitive.

**type:** indicates the index type, which is mandatory. The options are as follows: **COMPOSITE** (Composite index: internal exact match index with index data stored in HBase) or **MIXED** (mixed index: external mixed match index with index data stored in Elasticsearch) which is case insensitive.

The analyzer cannot be configured by using COMPOSITE indexes.

- If the property key of this index does not contain any Chinese, you do not need to configure the analyzer.
- You can configure the analyzer only for data types, such as TEXT and TEXTSTRING.
- The analyzer can be set for MIXED indexes:
  - ik\_max\_word (fine-grained)
  - ik\_smart (coarse-grained, recommended)

**keyTextTypeList:** indicates the property key list in text format. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name:** indicates the property name, which is mandatory.

**textType:** indicates the index data type. The options are as follows:

- **TEXT:** indicates that the index data is of the text type.
- **TEXTSTRING:** indicates that the index data is of the text string type.
- **PREFIX\_TREE:** indicates that the index is of the geographical location type.
- **DEFAULT:** indicates the default index type. The value is case insensitive and can be empty. If this parameter is left empty, the default type is used.

## NOTICE

- GraphBase index categories:
  - Composite index: is the internal exact match index with the index data stored in HBase and does not support full-text search or splitter.
  - Mixed Index: is the external mixed match index with the index data stored in Elasticsearch and supports full-text search and splitter.
- Suggestions for using GraphBase indexes:
  - When a property is the vertex primary key, an internal index (Composite Index) must be created.
  - You are advised to create mixed indexes in common cases because the index search performance is better and the functions are and richer than the composite index.
- The field definitions in the XML file labels are the same as those in the preceding labels. For details, see "Adding a Vertex Label", "Adding an Edge Label", "Adding a Property Key", "Adding a Graph Index", and "Vertex-centric Index Management" in "GraphBase" of the *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

## 2. Create metadata using the schema file.

REST APIs can be used to upload the schema file. For details, see [Creating a Schema by Uploading an XML File](#) in the sample project.

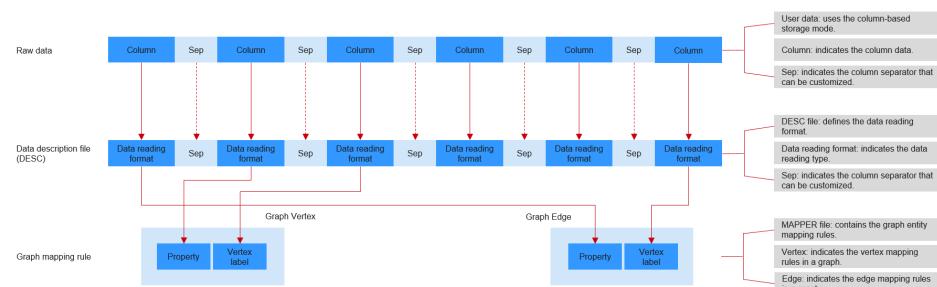
### 1.9.3.2.2 Compiling a Data File (CSV)

#### Data-Related Files

Data-related files are classified into the following types:

1. User data files, also called raw data files that are in CSV format.
2. Data description file is used to define the schema mapping relationship for data, and the file format is DESC.
3. Graph mapping rule file, which defines the mapping between raw data and entities in the graph database. The file format is MAPPER.

The following figure shows the relationship between the three types of files.



## Compiling Raw Data Files

Based on the instructions provided in [Typical Application Scenario](#), define the raw data file as follows (The following code snippets are used as an example. For complete codes, see [GraphBase.examples.person-demo](#)):

Note: The data in each column corresponds to the **index** value of properties in the graph mapping file (.mapper).

```
marko,29,,vadas,27,,knows,0.4,  
marko,29,150000000000,,,0.8,,  
josh,32,1522222222,,0.8,,  
josh,32,,blame,30,,knows,0.4,  
blame,30,136666666666,,,0.8,,  
,,136666666666,,137777777777,,0.6  
,,137777777777,,136666666666,,,0.6  
vadas,27,137777777777,,0.8,,  
marko,29,,josh,32,,friend,1.0,
```

### 1.9.3.2.3 Compiling a Data Description File (DESC)

#### Scenarios

Data description files (in DESC or HVDE format) corresponding to CSV data files are required during data reading.

Based on the information provided in [Typical Application Scenario](#), design data description files as follows (The following code snippets are used as an example. For complete codes, see [GraphBase.examples.person-demo](#)):

```
{"delim":",",  
"includeRowHead":false,  
"attributes": [  
 {"name":"attr_0","type":"STRING","nominalList":[]},  
 {"name":"attr_1","type":"INTEGER","nominalList":[]},  
 {"name":"attr_2","type":"STRING","nominalList":[]},  
 {"name":"attr_3","type":"STRING","nominalList":[]},  
 {"name":"attr_4","type":"INTEGER","nominalList":[]},  
 {"name":"attr_5","type":"STRING","nominalList":[]},  
 {"name":"attr_6","type":"REAL","nominalList":[]},  
 {"name":"attr_7","type":"STRING","nominalList":[]},  
 {"name":"attr_8","type":"REAL","nominalList":[]},  
 {"name":"attr_9","type":"REAL","nominalList":[]} ]}
```

#### File Format Description

For details about the data description files (in DESC format) corresponding to CSV data files, see [Table 1-72](#).

**Table 1-72** Data description file examples

File Format	Description
DESC	<p>The file contains <b>delim</b> and <b>attributes</b>.</p> <ul style="list-style-type: none"><li>• <b>delim</b>: defines the delimiter for data in the CSV data file. Ensure that you use the same delimiter in the CSV data file, otherwise, data fails to be imported, including commas (,), spaces, tabs, and user-defined delimiters. For example, a slash (/) or a colon (:).</li><li>• <b>includeRowHead</b>: specifies whether a header is included. If it is set to <b>false</b>, no header is included. If it is set to <b>true</b>, a header is included.</li><li>• <b>attributes</b>: includes <b>name</b>, <b>type</b>, <b>nominalList</b>, and <b>valueFormat</b>.</li><li>• <b>type</b>: specifies the feature data type, which can be String, Integer, Real, Date, Time, or Timestamp.</li><li>• <b>name</b>: specifies the feature name.</li><li>• <b>nominalList</b>: specifies the enumerated value list.</li><li>• <b>valueFormat</b>: specifies the date format when <b>type</b> is set to <b>DateTime</b>, for example, yyyy-MM-dd HH:mm:ss.</li></ul> <p>Metadata is defined as follows:</p> <pre>{   "delim": ",",   "includeRowHead": false,   "attributes": [     {"name": "name", "type": "STRING", "nominalList": []},     {"name": "date", "type": "DATETIME", "nominalList": [], "valueFormat": "yyyy-MM-dd HH:mm:ss"}   ] }</pre>

 NOTE

- A column name cannot contain special characters, for example, #@\*.
- The yyyyMMdd time format corresponds to year, month, and day. A complete time format is yyyyMMdd HH:mm:ss, corresponding to year, month, day, hour, minute, and second. The common time formats are yyyy/MM/dd and yyyy-MM-dd.

#### 1.9.3.2.4 Compiling a Graph Mapping Rule File (.mapper)

A graph mapping rule describes the mapping between the columns of a data set and the graph elements.

### Scenarios

Based on [Typical Application Scenario](#), design the graph mapping rule file as follows (The following code snippets are used as an example. For complete codes, see [GraphBase.examples.person-demo](#)):

```
{
  "entities": {
    "entity1": {
      "conceptIRI": "person",
```

```
"primaryKey": "name",
"properties": {
    "name": {
        "index": 0
    },
    "age": {
        "index": 1
    }
},
"entity2": {
    "conceptIRI": "phone",
    "primaryKey": "telephone",
    "properties": {
        "telephone": {
            "index": 2
        }
    }
},
"entity3": {
    "conceptIRI": "person",
    "primaryKey": "name",
    "properties": {
        "name": {
            "index": 3
        },
        "age": {
            "index": 4
        }
    }
},
"entity4": {
    "conceptIRI": "phone",
    "primaryKey": "telephone",
    "properties": {
        "telephone": {
            "index": 5
        }
    }
},
"relationships": [
    {
        "source": "entity1",
        "target": "entity2",
        "unique": true,
        "label": "has",
        "properties": {
            "weight": {
                "index": 6
            }
        }
    },
    {
        "source": "entity1",
        "target": "entity3",
        "unique": true,
        "labelType": "columnLookup",
        "labelColumn": {
            "index": 7
        },
        "properties": {
            "weight": {
                "index": 8
            }
        }
    },
    {
        "source": "entity2",
        "target": "entity4",
        "unique": true,
        "label": "isUsedFor"
    }
]
```

```
        "unique": true,
        "label": "call",
        "properties": [
            "weight": {
                "index": 9
            }
        ]
    }
}
```

## File Structure

The basic structure is as follows:

(The texts in bold are the defined property names, which are set by users.) The italic words can be entered by users based on scenarios and requirements. The underscored italic words indicate that the corresponding schemas must be created in the graph in advance.)

```
{
    "entities": {
        "entityName1": {
            "conceptIRI": "conceptName1",
            "primaryKey": "primaryKeyName1",
            "properties": {
                "propertyName1": {
                    "type": "propertyValueType",
                    "": ":{ }"
                },
                "propertyName2": {
                    "type": "propertyValueType",
                    "": ":{ }"
                }
            }
        },
        "entityName2": {
            "conceptIRIType": "constant",
            "conceptIRI": "conceptName2",
            "primaryKey": "primaryKeyName2",
            "properties": {
                "propertyName1": {
                    ...
                }
            }
        }
    },
    "relationships": [
        {
            "source": "entityName1",
            "target": "entityName2",
            "label": "labelName1"
        },
        {
            "source": "entityName3",
            "target": "entityName4",
            "label": "labelName2",
            "properties": {
                "propertyName1": {
                    "type": "propertyValueType",
                    "": ":{ }"
                },
                "propertyName2": {
                    "type": "propertyValueType",
                    "": ":{ }"
                }
            }
        }
    ]
}
```

- **entities** (mandatory): specifies entity sets. Entities indicate vertices in a graph. An entity set contains multiple entities.
- **relationships** (mandatory): specifies relationship sets. Relationships indicate edges in a graph.
- **entityName** (mandatory): specifies the entity name, which identifies an entity object and is referenced when a relationship is defined. In a mapping rule, entities cannot use the same name.

---

**NOTICE**

The entity that is defined after a used name in the graph overwrites the previously defined entity with the same name.

- **conceptIRIType** (optional): specifies the conceptIRI (entity label) definition mode. The available values are **constant** and **columnLookup**. If this parameter is not defined, the default value **constant** is used.
  - **constant**  
Specifies that a constant is defined as conceptIRI.  
Parameter:  
**conceptIRI** (mandatory): specifies an entity label, that is, an entity category. For example, conceptIRI can be a person or an item. **conceptIRI** is a string.
  - **columnLookup**  
Defines a value as the **conceptIRI**.  
Parameter:  
**concept** (mandatory): specifies a label value. The definition method is the same as that used for defining a property value.
- **primaryKey** (mandatory): specifies a primary key, which is the unique identifier of an entity (vertex). For example, the ID card number of a person. The value of **primaryKey** must be a property name in properties, and the property must be of the single type or a combination of required and single. That is, the property value must be a column of values in the input data rather than a multi-condition combination value. In addition, Built-in indexes must be created for the property that is used as a primary key. Currently, only one value can be specified for **primaryKey**, and the value cannot contain multiple primary keys.
- **createIfAbsent** (optional): specifies whether to create an entity when the entity does not exist. That is, whether the entity needs to be imported again. If this parameter is not set, the default value **true** is used.
  - **true**  
If this parameter is set to **true**, the common process is used. Check whether each entity has been imported. If yes, update the properties. If no, create required entities and add corresponding properties for the entities.
  - **false**  
If this parameter is set to **false**, the entity does not need to be imported. You can skip the entity query and property update. If some entities in the

data are not imported before, the entities will not be imported, and the relationships associated with the entities are not imported. If an entity has been imported before, the relationships associated with the entity can still be imported.

Setting this parameter to **false** improves the data import speed and reduces unnecessary query verification. However, you need to set this parameter based on the actual service scenario. In the following two typical scenarios, you are advised to set this parameter to **false**.

- If an exception occurs when data is imported. After the exception is handled, you are advised to import the data again. For example, if an exception occurs when the second entity is imported after the first entity is successfully imported. In this case, you can set this parameter to **false** in the mapping rule of the first entity for the second import. After this configuration, the first entity will be skipped and the data import will start from the second entity where the exception occurred. If the last exception occurs during the relationship import, you can set this parameter to **false** in the mapping rules of all entities for this import. After this configuration, the task will skip the phase of importing entity data, and only the relationship data is imported. In this way, data import speed is greatly improved and the data integrity is not affected.
- Some entities of the data have been imported for other data before. In this case, you must ensure that the entity type IDs of the data exist in the graph and that you do not need to update the property values this time. For example, to import the data relationship table of a transaction, the card numbers in the table have been imported when the card number table is imported. In this case, the card numbers in the transaction exist in the graph, and you do not need to update other property values of the card number. You can set **createIfAbsent** to **false** in the mapping rule of the card number to speed up the import.
- **properties** (mandatory): specifies properties of entities, such as the height and gender of a person. When defining an entity, you must define at least one property as the primary key. Property names, values, and rules must be provided. **propertyName** is defined by users. Property values have multiple types which can be specified by **propertyValueType**. Currently, the following nine types are supported: **single**, **required**, **constant**, **formattedMultiColumn**, **geopoint**, and **geocircle** (mandatory), **geobox**, **mappedMultiColumn**, and **fallback**. If the type is not specified, the default value **single** is used. Multiple property types can be nested. The following uses **ColumnValue** as an example to describe how to nest a property method.
  - **single**  
A column of values in the data set is used as the property values. This is the most common scenario.  
Parameter:  
**index** (mandatory): specifies a column of subscripts corresponding to the property value.
  - **required**

If a property value is defined as **required**, the value cannot be empty. If the value is not defined as **required**, the value can be empty. This definition method needs to be nested with other definition methods.

Parameter:

**Column** (mandatory). The value is **ColumnValue**. This parameter is used to define a value that cannot be empty.

- **constant**

Specifies that the property value is a constant.

Parameter:

**value** (mandatory): The value must be a constant. The value data type can only be a basic type, for example, numeric, string, or Boolean.

- **formattedMultiColumn**

Multiple property values are combined in sequence based on the defined format.

Parameter:

**columns** (mandatory): specifies the definition of multiple values in List[*ColumnValue*] format. Note that the definition sequence is related to the combination sequence.

**format** (mandatory): specifies the combination format. Placeholders are used, for example, %s-%s, indicates that two strings are connected using a hyphen. %d indicates an integer, and %f indicates a floating point number.

- **geopoint**

Specifies the geographical location that indicates the location of a point.

Parameter:

**latitudeColumn** (mandatory): *ColumnValue* Specifies the latitude. The value must be a numeral ranging from -90 to 90.

**longitudeColumn** (mandatory): *ColumnValue* Specifies the longitude. The value must be a numeral ranging from -180 to 180.

- **geocircle**

Specifies a geographical circle, which is determined by the location of a point and its radius.

Parameter:

**latitudeColumn** (mandatory): *ColumnValue* Specifies the latitude. The value must be a numeral ranging from -90 to 90.

**longitudeColumn** (mandatory): *ColumnValue* Specifies the longitude. The value must be a numeral ranging from -180 to 180.

**radiusColumn** (mandatory): *ColumnValue* Specifies the radius. The value must be a numeral. The unit is km.

- **geobox**

Specifies a geographical frame (a trapezoid on a sphere). You can determine the scope of the trapezoid by using the latitude and longitude values in the lower left corner of the rectangle and those in the upper right corner of the trapezoid.

Parameter:

**southWestLatitudeColumn** (mandatory): *ColumnName* Specifies the latitude of the southwest point. The value must be a numeral ranging from -90 to 90.

**southWestLongitudeColumn** (mandatory): *ColumnName* Specifies the longitude of the southwest point. The value must be a numeral ranging from -180 to 180.

**northEastLatitudeColumn** (mandatory): *ColumnName* Specifies the latitude of the northeast point. The value must be a numeral ranging from -90 to 90.

**northEastLongitudeColumn** (mandatory): *ColumnName* Specifies the longitude of the northeast point. The value must be a numeral ranging from -180 to 180.

- **mappedMultiColumn**

Combines multiple values in sequence to obtain a property value. Based on the defined mapping relationship, if the property value meets the mapping relationship, obtain the mapping value based on rules described in the mapping and use the mapping value as the final property value. If the mapping relationship cannot be found, delete the last property value and then search for the mapping relationship. Repeat this operation until only the first property value is left. If no mapping relationship is found still, the default value **null** is returned.

Parameter:

**keyColumns** (mandatory): specifies the definition of multiple values in List[*ColumnName*] format. Note that the definition sequence is related to the combination sequence.

**valueMap** (mandatory): The mapping relationship is in the following format: Map<*String*, *String*>. If a combined value does not meet the mapping requirement, the default value **null** is used. If you want to set the property value to a default value when the mapping rules are not met, you can set a null string as the key in the mapping relationship and set value to the default value.

**separator** (optional): specifies the separator. Separators are connectors between properties. **If this parameter is not specified, a colon (:) is used as the separator by default.**

If the property value is empty, the separator is also used. For example, if three values are defined for **keyColumns**, which are A, B, and C, the processing sequence is as follows:

- i. Combine the three values to obtain **A:B:C**.
- ii. Check whether there is a property whose key is **A:B:C** exists in the mapping. If the property exists, the mapping value is returned. If the property does not exist, delete the last value in the combination to obtain **A:B**.
- iii. Check whether there is a property whose key is **A:B** exists in the mapping. If the property exists, the mapping value is returned. If the property does not exist, delete the last value in the combination to obtain **A**.
- iv. Check whether the property whose key is **A** exists in the mapping. If the property does not exist, delete the last value **A**. In this case, an empty string is left.

- v. Check whether there is a null string mapping in the mapping (that is, the default value). If the mapping exists, the default value is returned. If no mapping is found, the **null** value is returned.
- **fallback**

In this definition method, two *ColumnValue* can be defined. One is active, and the other is the standby. If the active value is null or the value matches the corresponding **fallbackIf** condition, the standby value is used.

Parameter:

**Primarycolumn** (mandatory): *Columnvalue*. Specifies the first value. The property value takes precedence over the value of this parameter. Then, the system determines whether to obtain the standby value based on the condition.

**fallbackColumn** (required): *ColumnValue*. Specifies the standby value. When the primary value is empty or the value meets the **fallbackIf** condition, the value is used as the property value.

**fallbackIf** (mandatory): specifies the predicate, that is, the matching criteria. The following code uses Predicate as an example. Example:

```
"fallbackIf": {  
    "op": "stringEq",  
    "value": "unknown"  
}
```

Currently, the following nine basic conditional expression predicates are supported: **and**, **or**, **isNull**, **not**, **eq**, **stringEq**, **emptyStr**, **moreThan**, and **lessThan**.

- **and**

If multiple conditions are matched at the same time, the value is **true**.

Parameter:

**predicates** (mandatory): List[*Predicate*]: Specifies multiple conditions.

- **or**

If any condition is matched, the value is **true**.

Parameter:

**predicates** (mandatory): List[*Predicate*]: Specifies multiple conditions.

- **isNull**

If the value is empty, the value is **true**.

Specifies that no parameter is set.

- **not**

If criteria is not matched, the value is **true**.

Parameter:

**predicate** (mandatory): specifies the predicate, that is, a single condition.

- **eq**

If the values are the same, the value is **true**.

Parameter:

**value** (mandatory): specifies the value for judgment, which can be a basic type value.

- **stringEq**

If the strings are the same, the value is **true**. You can define whether the strings are case sensitive.

Parameter:

**value** (mandatory): specifies the value for judgment, which is a string.

**caseSensitive** (optional): specifies whether the value is case sensitive. The default value is **false** that is case insensitive.

- **emptyStr**

Checks whether the entered string is empty or all characters are spaces.  
Note: Empty strings are not null.

Specifies that no parameter is set.

- **moreThan**

Checks whether the entered value is greater than the specified value.

Parameter:

Value (mandatory): Specifies the value for judgment. The value must be a number, and the entered value must be a number too.

- **lessThan**

Checks whether the entered value is less than the specified value.

Parameter:

Value (mandatory): Specifies the value for judgment. The value must be a number, and the entered value must be a number too.

- **action**

A property rule is used to perform operations on the property value. The parameter is defined as an action. Currently, the following actions are supported:

**assign**: specifies that the property value is updated.

**assignIfAbsent**: specifies if the property does not exist, the property is updated. If the property does not exist, the property is not updated.

**assignIfBigger**: compares the existing value and a new value of the property to obtain the larger value and then updates the value. Only the numeric type and date type are supported.

**assignIfSmaller**: compares the existing value and a new value of the property to obtain the smaller value and then updates the value. Only the numeric type and date type are supported.

**assignSum**: sums up the existing value and a new value of the property, and then updates the value. Only the numeric type is supported.

If the property field is not set to **action**, the default value **assign** is used.

- The rules for filling in **relationships** are as follows:
- *source* (mandatory): specifies the start entity. The value must be an entity name defined above.
- *target* (mandatory): specifies the target entity. The value must be an entity name defined above.
- *labelType* (optional): specifies the label (relationship) definition mode. The available values are **constant** and **columnLookup**. If this parameter is not defined, the default value **constant** is used.

- **constant**  
Specifies that a constant is defined as conceptIRI.  
Parameter:  
**label** (mandatory): specifies the edge label. The value must be an edge label defined in the graph.
  - **columnLookup**  
Defines a value as the label.  
Parameter:  
**labelColumn** (mandatory): specifies the edge label value. The definition method is the same as that for property values described in the previous content. For details, see the related information provided above.
  - **unique** (optional): specifies whether modification is required based on the original relationship. If this parameter is not set, the default value **false** is used.
    - **true**  
If this parameter is set to **true**, the system checks whether the label relationship exists between the source entity and target entity before you import the relationship. If no, add an edge. If yes, update the attribute based on the original relationship. If there are multiple label relationships between two entities, you can set the **uniqueFilter** field to update a relationship. If the field is not set, the attribute of a relationship corresponding to the label is used for update. If all relationships do not meet the condition, you need to add an edge and ignore the update. The **uniqueFilter** field contains **propertyKey** and **propertyValue**, which respectively indicate the key and value of a special attribute of the edge. Multiple attribute keys can be set in the **uniqueFilter** field. Use commas (,) to separate multiple attribute keys. You can set only **propertyKey** but not **propertyValue**.
- The query rule description is as follows:
- i. If **propertyKey** is configured but **propertyValue** is not configured, the corresponding column value in the data is used for query. If the column value is null, keys are used for query.
  - ii. If **propertyKey** and **propertyValue** are both configured but the value is not null, use the value. If the value is null, replace it with the data in the CSV file.
  - iii. If **propertyKey** and **propertyValue** are both configured but the value is null, only keys are used for query.
- The following is a configuration example of the uniqueFilter in common scenarios:
- The configuration of **uniqueFilter** in common scenarios is as follows:
- ```
"uniqueFilter" : {  
    "propertyKey" : "p1,p2"  
}
```
- **false**  
No query operation is performed before the relationship is imported. The relationship of the corresponding label is directly added, and the original relationship is not changed.

- **optimizeUniqueCheck** (optional): specifies whether to optimize the check of **unique**. This parameter is valid only when **unique** is set to **true**. If this parameter is not set, the default value **false** is used.
  - **true**  
If this parameter is set to **true**, the label relationships to be added between the entities are queried in batches, reducing the number of HBase requests and accelerating the query of **unique**.
  - **false**  
If this parameter is set to **false**, the query is performed only when a relationship is added for two entities.

### NOTICE

Note: When an entity has a large number of identical label relationships (for example, the call between a person and another), setting this parameter to **true** may lead to slow query speed because the amount of returned data is too large or even may cause query failure for insufficient memory. In practical use, estimate the number of entity edges based on the service and data. Use this parameter with caution.

- **Properties** (optional): specifies the properties corresponding to the label. The parameter setting rule is the same as that for the entity property.

## Example

Assume that the node input mapping rules are as follows. For the meanings of the mapping rules, see the comments on the right of these rules.

```
{  
    "entities": {  
        "person": { // Defines an entity and is named as person. This name is used only when relationships is defined and it does not need to be defined in the schema.  
            "conceptIRI": "person", // The entity label is person, and conceptIRIType is omitted here. The default value is constant, that is, the value of conceptIRI is directly defined. The other islabel is columnLookup. For details, see the employer entity described below.  
            "primaryKey": "UID", // The entity ID is the property value whose name is UID in properties below. You can use this ID to uniquely identify an entity in the graph.  
            "properties": { // Defines the properties of person. UID and Name are property names, and their values are property values.  
                "UID": {  
                    "type": "required", // If this parameter is set to required, the value cannot be empty. If the value is empty, an error is reported.  
                    "column": {  
                        "index": 0,  
                    }  
                },  
                "Name": {  
                    "type": "formattedMultiColumn", // formattedMultiColumn indicates that Name is composed of the values whose subscript are 1 and 2, respectively, in the format of %s-%s. For example, the values whose subscripts are 1 and 2 are zhang and san, respectively. Name is zhang-san.  
                    "columns": [  
                        {  
                            "index": 1  
                        },  
                        {  
                            "index": 2  
                        }  
                    ]  
                }  
            }  
        }  
    }  
}
```

```

        ],
        "format": "%s-%s"
    },
    "shortName": {
        "type": "mappedMultiColumn",      // mappedMultiColumn indicates that the value of shortName
is composed of the value whose subscript is 1 and the value whose subscript is 2, and the two values are
separated by a colon (:). It is used to check whether the combination result has a mapping in valueMap.
        "keyColumns": [      // If the mapping exists, use the mapping value. For example, if the value
whose subscript 1 is zhang and the value whose subscript is 2 is san, the combination is zhang:san. In
valueMap, the value of zhang:san is zs. Therefore, the value of shortName is zs.
9           {      // If the mapping does not exist, remove the last value of the combination result and then
search for the mapping. For example, if the value whose subscript is 1 is zhao, and the value whose
subscript is 2 is wu, the combination is zhao:wu.
        "index": 1      // If valueMap does not contain zhao:wu, remove wu, and only zhao is left.
valueMap still does not contain the keyword zhao. Delete zhao to obtain an empty string.
        },      // The value corresponding to an empty string in valueMap is no short name. Therefore,
the value of shortName is no short name.
        {      // If the mapping corresponding to an empty string is not defined in valueMap, the value of
shortName is null.
            "index": 2
        }
    ],
    "valueMap": {
        "": "no short name",
        "zhang:san": "zs",
        "li:si": "ls"
    },
    "separator": ":"      // This parameter is optional. If separator is not defined, the default value is a
colon(:).
},
"BIRTH DATE": {
    "type": "single",      // single indicates that the value of BIRTH DATE is the value whose subscript is
3. The property value type is single by default, which can be omitted. For example, the following property
Married, its property type is ommited.
    "index": 3
},
"Married": {
    "index": 4      // The type single is omitted.
},
"Deceased": {
    "type": "constant",      // constant indicates that all Deceased properties are set to false.
    "value": "false"
},
"Home Address": {
    "type": "geopoint",      // geopoint indicates that the value of Home Address is the location
corresponding to the combination of the value whose latitude is the value of subscript 5 and the value
whose longitude is the value of subscript 6.
    "latitudeColumn": {
        "index": 5
    },
    "longitudeColumn": {
        "index": 6
    }
},
"Home Area": {
    "type": "geocircle",      // geocircle indicates that the value of Home Area is the scope that the
latitude is subscript 7, the longitude is subscript 8, and the radius is subscript 9.
    "latitudeColumn": {
        "index": 7
    },
    "longitudeColumn": {
        "index": 8
    },
    "radiusColumn": {
        "index": 9
    }
},
"Country Area": {
    "type": "geobox",      // geobox indicates that the value of Country Area is a geographical area
}

```

with the latitude ranges from the value of subscript 10 to that of subscript 12, the longitude ranges from the value of subscript 11 to that of subscript 13.

```

    "southWestLatitudeColumn": {
        "index": 10
    },
    "southWestLongitudeColumn": {
        "index": 11
    },
    "northEastLatitudeColumn": {
        "index": 12
    },
    "northEastLongitudeColumn": {
        "index": 13
    }
},
"employer": { // Defines an entity named employer
    "conceptIRIType": "columnLookup", // If conceptIRIType is set to columnLookup, a value is defined to indicate the conceptIRI. For example, the value of the subscript 14 is used as the conceptIRI.
        "concept": {"index": 14},
        "primaryKey": "companyName",
        "properties": {
            "companyName": {
                "index": 15
            },
            "locationAddress": {
                "type": "geopoint",
                "latitudeColumn": {
                    "index": 16
                },
                "longitudeColumn": {
                    "index": 17
                }
            },
            "locationPlace": {
                "type": "fallback", // fallback indicates that the value of primaryColumn is used preferentially as the property value. If the value of primaryColumn is empty or meets the fallbackIf condition, the value of fallbackColumn is used.
                    "primaryColumn":{ // If the value of subscript 18 is empty and the value of subscript 19 is guangdong, the value of locationPlace is guangdong.
                        "index": 18 // If the value of subscript 18 is unknown and the value of subscript 19 is guangdong, the fallbackIf condition is met. In this case, the value of locationPlace is guangdong.
                    },
                    // The value of subscript 18 is shenzhen, the value of subscript 19 is guangdong, the value shenzhen is neither empty nor meets the fallbackIf condition. In this case, the value of locationPlace is shenzhen.
                    "fallbackColumn":{
                        "index": 19
                    },
                    "fallbackIf": {
                        "op": "stringEq",
                        "value": "unknown"
                    }
                },
                "Earnings": {
                    "type": "required",
                    "column": {
                        "index": 20
                    }
                }
            },
            "home": {
                "conceptIRI": "location",
                "primaryKey": "house_number",
                "properties": {
                    "Population": {
                        "index": 22
                    }
                }
            }
}

```

```
        },
        "house_number": {
            "type": "required",
            "column": {
                "index": 21
            }
        }
    },
    "social utility": {
        "conceptIRI": "utility",
        "primaryKey": "userID",
        "properties": {
            "userID": {
                "index": 23
            }
        }
    },
    "relationships": [
        {
            "source": "person", // indicates the relationship that the person lives at home.
            "target": "home",
            "label": "livesAt"
        },
        {
            "source": "employer",
            "target": "person",
            "label": "employs",
            "unique": true, // This parameter is optional. If unique is not defined, the default value false is used, indicating that an employs relationship is added. If this parameter is set to true, the full-time property is updated in the original relationship when the employs relationship already exists between employer and person.
            "uniqueFilter":{} //This parameter is optional when unique is set to true. Based on this filter condition, update an edge between the two vertices in the same label. This parameter is left blank when unique is set to false.
            "propertyKey": "place",
            "propertyValue": "xian"
        },
        "properties": {} // Properties are not mandatory. For example, the preceding relationship does not have properties.
        "full-time": {
            "index": 24
        }
    ],
    {
        "source": "person",
        "target": "social utility",
        "labelType": "columnLookup", // If labelType is set to columnLookup, a value is defined to indicate the label. For example, the value of the subscript 27 is defined and used as the label.
        "labelColumn": {
            "index": 27
        },
        "properties": {
            "twitter": {
                "index": 25
            },
            "facebook": {
                "index": 26
            }
        }
    }
]
```

### 1.9.3.3 Importing Data

### 1.9.3.3.1 Importing Data in Batches Using Tools

#### ⚠ CAUTION

Data inconsistency occurs when GraphBase imports and deletes data at the same time. If you are using a data ingestion tool on the client, ensure that no batch deletion is executed at the same time.

The system automatically checks whether there are other asynchronous tasks when tasks are submitted through a REST API. You are advised to use the REST API **graphwriter** to submit batch import tasks.

## Prerequisites

1. Install clients for all components in the cluster.
2. A graph has been created. For details about how to create a graph, see [Creating or Deleting a Graph](#).
3. A schema file (for example, an XML file) has been prepared and uploaded. For details about how to upload a schema file, see [Creating a Schema by Uploading an XML File](#).
4. You have prepared the data file (for example, a CSV file). For details about how to prepare a data file, see the example on the GraphBase client in the `/opt/hadoopclient/GraphBase/examples/person-demo/` directory.  
For details about the data description file (DESC file), see [Compiling a Data Description File \(DESC\)](#).  
The graph mapping rule file (.mapper file) has been uploaded. For details, see [Compiling a Graph Mapping Rule File \(.mapper\)](#).
5. Required users have been created and the authentication files have been downloaded. For details about how to create a user, see [Preparing a Developer Account](#).

#### 📖 NOTE

The created user must have the rights of the **graphbaseadmin** user group.

## (Optional) Change the default username

Spark and the username are configured in the configuration **resource.properties** in the `/opt/hadoopclient/GraphBase/graphwriter/conf/` directory, as shown in the following:

```
USERNAME = graphtest  
BINDING_SPARK_SERVICE = Spark  
RESOURCE = --executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2 --conf  
spark.yarn.security.credentials.hbase.enabled=true --conf spark.inputFormat.cache.enabled=false
```

1. If the created username is **graphtest**, you do not need to change the username. Otherwise, change the username to the created username.
2. Configure the **RESOURCE** parameter as required. You can also add Spark advanced parameters, for example **--conf spark.default.parallelism** that is set to **105**.

When data is imported in batches, the Yarn resource queue in Spark parameters is the default queue. To change the default resource queue, add

the **--conf spark.yarn.queue** parameter and set it to the specified queue name in the **RESOURCE** parameter. For details about how to configure Yarn resource queues, see section "Tenant Resources" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

## Upload related files

1. Upload user authentication files.

Use WinSCP to upload the downloaded user authentication files **krb5.conf** and **user.keytab** to the **/opt/hadoopclient/GraphBase/graphwriter/conf** directory on the GraphBase client, as shown in the following figure.

```
[root@hd-184 conf]#ll
total 16
-rwxr-xr-x 1 root root 1178 Nov 25 15:58 janusgraph-loading.properties
-rw-r--r-- 1 root root 1255 Nov 26 19:02 krb5.conf
-rwxr-xr-x 1 root root 108 Nov 25 15:58 resource.properties
-rw-r--r-- 1 root root 142 Nov 26 19:02 user.keytab
```

2. Upload data files to the HDFS.

Switch to the **/graphwriter** directory in the installation directory of the GraphBase client.

```
cd /opt/hadoopclient/GraphBase/graphwriter
```

Run the following command to execute environment variables:

```
source /opt/hadoopclient/bigdata_env
```

Log in to the system as a GraphBase user and enter the password (specified by the user).

```
kinit graphtest
```

Create a directory storing HDFS files. (If the directory exists, skip this operation.)

```
hdfs dfs -mkdir /user/graphtest
```

Upload the data files to the **/user/graphtest** directory (for example, **person-demo**) of the HDFS.

```
hdfs dfs -put ..//examples/person-demo/Person.csv ..//examples/person-demo/Person.csv.mapper ..//examples/person-demo/Person.desc /user/graphtest
```

The details are shown in the following figure.

```
hadoopNode2:/opt/graphbaseClient/GraphBase/graphwriter # hdfs dfs -ls /user/graphtest
Found 7 items
-rw-r--r-- 3 root hadoop 100 2018-08-04 14:47 /user/graphtest/Person.csv
-rw-r--r-- 3 root hadoop 740 2018-08-04 14:47 /user/graphtest/Person.csv.mapper
-rw-r--r-- 3 root hadoop 368 2018-08-04 14:47 /user/graphtest/Person.desc
```

## Importing Data

Execute the following script and transfer the *graphName* parameter (*graphName* is user-defined that must exist. If it does not exist, create the graph first.)

```
./bin/graphWriter.sh graphName /user/graphtest/Person.csv /user/graphtest/Person.csv.mapper /user/graphtest/Person.desc
```

Note: The data file path must be an HDFS path that is in the following format:  
*Data file/Data mapping file.mapper/Metadata file.desc*

If no metadata file exists, the file path is: *Data file/Data mapping file.mapper*

After the script is executed, the following information is displayed:

```
hadoopNode2:/opt/graphbaseclient/graphBase/graphWriter # ./bin/graphWriter.sh graphName /user/graphtest/Person.csv /user/graphtest/Person.csv.mapper /user/graphtest/Person.desc
RESOURCE= -executor-memory 5g -driver-memory 5g --num-executors 5 --executor-cores 5
USERNAME= graphtest
spark.yarn.am.cores is set but does not apply in cluster mode.
spark.yarn.am.cores is set but does not apply in cluster mode.
2018-08-06 11:31:09.510 | WARN | main | Unable to load native-hadoop library for your platform... using builtin-java classes where applicable | org.apache.hadoop.util.N
der.<clinit>(NativeCodeLoader.java:62)
2018-08-06 11:31:09.482 | WARN | main | spark.varn.am.extraJavaOptions will not take effect in cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
```

#### NOTE

During data import, if the feature data type (type attribute in attributes) configured in the data description file (.desc file) is different from the data type in the data file (for example, .csv file) to be imported, the batch import tool ignores data of different data types by default and imports only the data of the same type in the data description file and data file. For example, the type of data in a column defined in the data description file is "REAL", as indicated by the following: {"name": "Attr\_0", "type": "REAL", "nominalList": []}. In the data file to be imported, the data of the corresponding column is the character string "mary". The data type of "mary" is STRING, which is inconsistent with the "REAL" type in the description file. In this case, when the batch import tool is used to import data, the data of this error type is ignored and will not be imported.

### 1.9.3.3.2 Importing Data in Real Time Using Tool

Data import in real time is implemented by using SparkStreaming. After SparkStreaming is started, data is read from the corresponding topic based on the configuration and then imported to GraphBase.

#### CAUTION

Data inconsistency occurs when GraphBase imports and deletes data at the same time. If you are using a data ingestion tool on the client, ensure that no batch deletion is executed at the same time.

## Prerequisites

1. Install clients for all components in the cluster.
2. The schema file (that is, the XML file) has been uploaded. For details, see [Creating a Schema by Uploading an XML File](#).
3. The graph mapping rule file (.mapper file) has been uploaded. For details, see [Compiling a Graph Mapping Rule File \(.mapper\)](#).
4. Required users have been created and the authentication files have been downloaded. For details about how to create a user, see [Preparing a Developer Account](#).

#### NOTE

The created user must have the rights of the **graphbaseadmin** user group.

5. The Kafka parameter **allow.everyone.if.no.acl.found** has been set to **true**, and the Kafka service has been restarted.

| Parameter                      | Value   |
|--------------------------------|---|
| <b>Kafka-&gt;Broker</b>        |   |
| allow.everyone.if.no.acl.found | <input checked="" type="radio"/> true <input type="radio"/> false |

## Modify the configuration file

The configuration file directory is **/opt/hadoopclient/GraphBase/graphstream/conf/**.

1. Modify the **graph-streaming.properties** file.

The parameter configuration is listed in the following table.

| Parameter                            | Description  | Example  |
|--------------------------------------|--|--|
| graph.user.name                      | Specifies the name of the user to be created.  | graphtest  |
| spark.streaming.checkpoint.directory | Specifies the HDFS directory of the SparkStreaming checkpoint. If the directory does not exist, create it in advance. (It is recommended that this directory be created separately because a large number of checkpoint files are generated during the program running.) | /user/graphtest/graphstreaming/checkpoint          |
| spark.streaming.batch.time           | Specifies the time spent by SparkStreaming on data processing in batches. The unit is s.   | 1  |
| spark.streaming.partition.number     | Specifies the number of partitions of the node where SparkStreaming resides.   | 50   |
| kafka.topic.name                     | Specifies the name of the topic where data needs to be consumed.   | Person.csv   |
| kafka.metadata.broker.list           | Specifies the IP address of the node where Kafka metadata resides (manual configuration is not required).  | 10.3.70.133:21005,10.3.1.67:21005,10.3.71.24:21005 |

| Parameter                     | Description  | Example  |
|-------------------------------|--|--|
| csv.mapper.hdfs.file          | Specifies the path of the Mapper file in the HDFS.   | /user/graphtest/graphstreaming/Person.csv.mapper |
| kafka.save.failed.message     | Specifies whether to save failure messages. The available values are <b>false</b> and <b>true</b> . <b>false</b> indicates that the failure message is not saved, and <b>true</b> indicates the reverse. | false  |
| kafka.save.failed.to.pic.name | Specifies that the failure messages are saved as Kafka topic names.  | default  |
| kafka.topic.message.separator | Specifies a separator of Kafka messages.   | ,  |

- Set the **RESOURCE** parameter.

In **resource.properties**, allocate resources based on YARN resource usage. Ensure that the resources to be allocated do not exceed the total resources of YARN.

```
RESOURCE='--executor-memory 4g --driver-memory 2g --num-executors 5 --executor-cores 5'.
```

When data is imported in batches, the Yarn resource queue in Spark parameters is the default queue. To change the default resource queue, add the **--conf spark.yarn.queue** parameter and set it to the specified queue name in the **RESOURCE** parameter.

## Upload related files

- Upload user authentication files.

Use WinSCP to upload the downloaded user authentication files **krb5.conf** and **user.keytab** to the **/opt/hadoopclient/GraphBase/graphstream/conf** directory on the GraphBase client.

- Upload the customized Mapper file to the HDFS.

Execute environment variables as user **root** (**/opt/hadoopclient/** is the installation directory of the cluster client).

```
source /opt/hadoopclient/bigdata_env
```

Log in to the system as user **weaver** using the user account and password (specified by the user).

```
kinit graphtest
```

Create a directory storing HDFS files. (If the directory exists, skip this operation.)

```
hdfs dfs -mkdir /user/graphtest/graphstreaming
```

Upload the MAPPER data file to the **/user/graphtest/graphstreaming** directory of HDFS (Take the **person-demo** directory as an example).

```
hdfs dfs -put /opt/hadoopclient/GraphBase/examples/person-demo/  
Person.csv.mapper /user/graphtest/graphstreaming
```

## Create a topic

The APIs or scripts of Kafka can be used to create topics. For details, see "Managing Kafka Themes" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

## Importing Data

1. Go to the **graphstream** directory.

```
cd /opt/hadoopclient/GraphBase/graphstream
```

2. Execute the following script and transfer the *graphbase* parameter (*graphbase* is user-defined that must exist. If it does not exist, create the graph first. Before you create the graph, ensure that all prerequisites have been completed):

```
./bin/graphStreaming.sh graphbase
```

If you need to customize the path of the configuration file, you can specify a local absolute path for the configuration file and copy the information in **graph-streaming.properties** to the configuration file. If the path is not specified, the **graph-streaming.properties** file in the **conf/** directory is used as the configuration file by default. Example:

```
./bin/graphStreaming.sh graphbase /opt/hadoopclient/GraphBase/  
graphstream/conf/graph-streaming.properties
```

3. (Optional) Send a message to the **Person.csv** topic.

Send the message by following the following procedure:

- a. Upload data files. If the HDFS directory does not exist, create it in advance.

```
hdfs dfs -put ..//examples/person-demo/Person.csv /user/graphtest/  
graphstreaming/data
```

Run the **vi conf/graph-producer.properties** command to configure the **graph-producer.properties** file in **conf**. The parameter configuration in this file is listed as follows.

| Parameter        | Description  | Example          |
|------------------|--|------------------|
| graph.user.name  | Specifies the name of the user to be created.                    | <b>graphtest</b> |
| kafka.topic.name | Specifies the name of the topic where data needs to be consumed. | Person.csv       |

| Parameter                  | Description   | Example   |
|----------------------------|---|---|
| kafka.metadata.broker.list | Specifies the IP address of the node where Kafka metadata resides (manual configuration is not required). | 10.3.70.133:21005,10.3.71.67:21005,10.3.71.24:21005 |
| csv.data.hdfs.directory    | Specifies the directory for storing data in the HDFS.   | /user/graphtest/graphstreaming/data                 |

- b. Run the following command to send a message to a specified Kafka topic:

**./bin/graphProducer.sh**



This mode is used to test the production data. You can use other methods too.

## Monitor tasks

1. Click **ApplicationMaster** on the YARN task page to go to the Spark task page.



2. Click **Streaming**.



3. Check the processing progress.

| Batch Time          | Input Size     | Scheduling Delay <small>(?)</small> | Processing Time <small>(?)</small> | Total Delay <small>(?)</small> |
|---------------------|----------------|-------------------------------------|------------------------------------|--------------------------------|
| 2018/07/19 11:43:40 | 3579000 events | 42 min                              | 3.6 min                            | 45 min                         |
| 2018/07/19 11:43:30 | 3690000 events | 38 min                              | 3.9 min                            | 42 min                         |
| 2018/07/19 11:43:20 | 3608500 events | 34 min                              | 3.7 min                            | 38 min                         |
| 2018/07/19 11:43:10 | 3294500 events | 31 min                              | 3.4 min                            | 35 min                         |
| 2018/07/19 11:43:00 | 3675000 events | 27 min                              | 3.9 min                            | 31 min                         |
| 2018/07/19 11:42:50 | 9673500 events | 24 min                              | 3.8 min                            | 28 min                         |

## Tune parameters

- The product of the number of executors and the number of cores of each executor determines the task concurrency capability. The value of **spark.streaming.partition.number** determines the total number of tasks that can be started.  
If the resources are sufficient, you can set the executor and partition quantities to the same.
- spark.streaming.batch.time** of SparkStreaming and the data traffic of Kafka messages determine the amount of data that can be processed by SparkStreaming at a certain time. The data is distributed on each partition.  
The data traffic value is negatively correlated with the batchtime value. If the data traffic is small, set **batchtime** to a larger value. Otherwise, set **batchtime** to a small value. In this way, the amount of data to be processed can be ensured moderate.

## Stop a job

After a SparkStreaming task is submitted, it keeps running and reads messages from Kafka. To stop the task, perform the following operations: Obtain the submitted task ID on the YARN web client. Run the **yarn application -kill \${appid}** command to kill the task. Example: **yarn application -kill \$Application\_1530539359099\_0053** If no data is queried after the data import, kill the task and collect logs. Search for **ERROR** in the logs, and identify the cause. Run the following command to collect logs:

```
yarn logs -applicationId ${appid} > log
```

### 1.9.3.3.3 Using the Hive Data Import Tool

 CAUTION

Data inconsistency occurs when GraphBase imports and deletes data at the same time. If you are using a data ingestion tool on the client, ensure that no batch deletion is executed at the same time.

## Prerequisites

1. Install clients for all components in the cluster.
2. Create a graph by referring to [Creating or Deleting a Graph](#).
3. Prepare and upload a schema file (for example, an XML file) by referring to [Creating a Schema by Uploading an XML File](#).
4. The Hive component has been installed in the cluster.
5. Prepare a Hive table. The field names in the figure can be different from those in the Hive table. Compile a graph mapping rule file (**.mapper** file) based on the data columns of vertices and edges in Hive. For details, see [Compiling a Graph Mapping Rule File \(.mapper\)](#).
6. Create a user (with the Hive permission) and download the authentication file to **/opt/hadoopclient/GraphBase/graphwriter/conf**. For details, see section [Preparing a Developer Account](#).
7. For details about how to create Hive data, see section [Developing an Application](#).

## Modify the configuration file

The configuration file directory is **/opt/hadoopclient/GraphBase/graphwriter/conf/**.

1. Modify the **hiveTable.properties** file.

The parameter configuration is listed in the following table.

| Parameter        | Description   | Example       |
|------------------|---|---------------|
| table.namespace  | Indicates the namespace where the Hive table is located. Retain the <b>default</b> value.   | default       |
| table.tablename  | Indicates the Hive table name.  | table1        |
| table.columns    | Indicates the columns to be selected in the table. By default, the asterisk (*) indicates a global column. Multiple column names are separated by commas (,). | * or name,age |
| table.conditions | Indicates that you can write a condition expression to select the column that meets the condition. The format is the same as that of the WHERE clause.        | age=20        |

## 2. Set the **RESOURCE** parameter.

In **resource.properties**, allocate resources based on YARN resource usage. Ensure that the resources to be allocated do not exceed the total resources of YARN.

`RESOURCE="--executor-memory 4g --driver-memory 2g --num-executors 5 --executor-cores 5"`

When data is imported in real time, the YARN resource queue in Spark parameters is the default queue. To change the default resource queue, add the **--conf spark.yarn.queue=default** parameter and set it to the specified queue name in the **RESOURCE** parameter.

## Import data

Run the following script that contains the graph name (*graphName* is a user-defined graph name and must exist. If the graph name does not exist, create it first.) and the mapper file path (the mapper file path must be an HDFS path).

`./bin/hiveWriter.sh graphName /user/graphtest/xxx.mapper`

Example description:

```
CREATE EXTERNAL TABLE IF NOT EXISTS table1  
(
```

```
sourcelD STRING,  
targetId STRING,  
attr1 STRING,  
attr2 STRING  
)  
ROW FORMAT delimited fields terminated by ','  
STORED AS TEXTFILE;
```

Create the preceding Hive table. **sourcelD** corresponds to the primary key attribute of the start vertex, and **targetId** corresponds to the primary key attribute of the end vertex. If the start vertex and end vertex are not of the same type (for example, the start vertex is an ID card number and the end vertex is a mobile number), **entities** in the mapper file are defined as follows:

```
"entities": {  
  "entity1": {  
    "conceptIRI": "person",  
    "primaryKey": "idCard",  
    "properties": {  
      "name": {  
        "index": 0  
      }  
    }  
  },  
  "entity2": {  
    "conceptIRI": "phone",  
    "primaryKey": "phoneNum",  
    "properties": {  
      "name": {  
        "index": 1  
      }  
    }  
  }  
}
```

If the start vertex and end vertex are of the same type (for example, the start vertex is a mobile number and the end vertex is a mobile number), entities in the mapper file are defined as follows:

```
"entities": {  
  "entity1": {  
    "conceptIRI": "phone",  
    "primaryKey": "phoneNum",  
    "properties": {  
      "name": {  
        "index": 0  
      }  
    }  
  },  
  "entity2": {  
    "conceptIRI": "phone",  
    "primaryKey": "phoneNum",  
    "properties": {  
      "name": {  
        "index": 1  
      }  
    }  
  }  
}
```

#### 1.9.3.4 Data Export



##### NOTE

When exporting data, check whether the specified HDFS path exists. If it does not, check whether the default HDFS path exists. If it does, delete the path. Otherwise, an error is reported and the task fails.

## Prerequisites

1. Install clients for all components in the cluster.
2. Create a user and download the authentication file. For details, see section [Preparing a Developer Account](#).

## Uploading User Authentication Files

Use WinSCP to upload the downloaded user authentication files **krb5.conf** and **user.keytab** to the **/opt/graphbaseClient/GraphBase/graphwriter/conf** directory on the GraphBase client, as shown in the following figure.

```
[root@8-5-199-1 conf]# ll
total 24
-rwxr-xr-x 1 root root 171 Mar  2 17:39 edge.properties
-rwxr-xr-x 1 root root 1347 Mar  2 17:39 janusgraph-loading.properties
-rwxr-xr-x 1 root root 1239 Mar  2 17:36 krb5.conf
-rwxr-xr-x 1 root root 223 Mar  2 17:36 resource.properties
-rwxr-xr-x 1 root root 122 Mar  2 17:36 user.keytab
-rwxr-xr-x 1 root root 99 Mar  2 17:38 vertex.properties
```

## Configuring Data Collection

The mapper file is used to collect the parameter information required during the export. The following figure shows the relationships between the vertex with label set to Person and Tag and the edge with label set to hasInterest:



The mapper file of the Person vertex is as follows:

```
{
  "entities": {
    "entity1": {
      "conceptIRI": "Person",           ----- This item specifies label (type) of the vertex.
      "primaryKey": "personId",        ----- This item specifies primaryKey of the vertex.
      "properties": {
        "personId": {                 ----- This item specifies the attribute key of the vertex.
          "index": 0
        },
        "firstName": {                ----- This item specifies the attribute key of the vertex.
          "index": 1
        },
        "lastName": {                  ----- This item specifies the attribute key of the vertex.
          "index": 2
        },
        "gender": {                   ----- This item specifies the attribute key of the vertex.
          "index": 3
        },
        "birthDay": {                  ----- This item specifies the attribute key of the vertex.
          "index": 4
        },
        "creationDate": {             ----- This item specifies the attribute key of the vertex.
          "index": 5
        },
        "locationIP": {               ----- This item specifies the attribute key of the vertex.
          "index": 6
        },
        "browserUsed": {              ----- This item specifies the attribute key of the vertex.
          "index": 7
        }
      }
    }
}
```

```
    }
}
}
```

The mapper file of the Tag vertex is as follows:

```
{
  "entities": {
    "entity1": {
      "conceptIRI": "Tag",           ----- This item specifies label (type) of the vertex.
      "primaryKey": "tagId",        ----- This item specifies primaryKey of the vertex.
      "properties": {
        "tagId": {                  ----- This item specifies the attribute key of the vertex.
          "index": 0
        },
        "name": {                   ----- This item specifies the attribute key of the vertex.
          "index": 1
        },
        "url": {                    ----- This item specifies the attribute key of the vertex.
          "index": 2
        }
      }
    }
}
```

The mapper file of the hasInterest edge is as follows:

```
{
  "entities": {
    "sourceEntity": {             ----- This item specifies the attribute of the outvertex.
      "conceptIRI": "Person",     ----- This item specifies the label (type) of the outvertex.
      "primaryKey": "personId",  ----- This item specifies the primaryKey of the outvertex.
      "createIfAbsent": false,
      "properties": {
        "personId": {            ----- This item specifies the attribute key of the outvertex.
          "index": 0
        }
      }
    },
    "targetEntity": {             ----- This item specifies the attribute of the invertex.
      "conceptIRI": "Tag",       ----- This item specifies the label (type) of the invertex.
      "primaryKey": "tagId",    ----- This item specifies the primaryKey of the invertex.
      "createIfAbsent": false,
      "properties": {
        "tagId": {              ----- This item specifies the attribute key of the invertex.
          "index": 1
        }
      }
    },
    "relationships": [
      {
        "source": "sourceEntity",  ----- This item specifies the entity of the outvertex.
        "target": "targetEntity",  ----- This item specifies the entity of the invertex.
        "labelType": "constant",  ----- This item specifies the labelType of the edge
        "label": "hasInterest"
      }
    ]
}
```

## Modifying Configuration Files

The configuration file directory is **/opt/hadoopclient/GraphBase/graphreader/conf/**.

1. Modify the **vertex.properties** file.

The parameter configurations are listed in the following table.

| Parameter         | Definition   | Example                    |
|-------------------|--|----------------------------|
| graphname         | Name of the graph instance to be operated. This parameter is mandatory.  | graphbase                  |
| label             | Type of the vertex to be exported. This parameter is mandatory.  | Forum                      |
| propertykeylist   | Property key list of the vertex to be exported. This parameter is mandatory. The primary key must be specified. If multiple properties need to be exported, separate the property keys by commas (,). The primary key must be placed in the first place.   | forumId,title,creationDate |
| dest.storage.path | HDFS path for storing the data to be exported. This parameter is optional and not recommended. Assume that the current domain name is HADOOP.COM, by default, the data is stored in <b>/user/graphtest@HADOO P.COM/graphbase/vertex/Person</b> . In the preceding directory, <b>graphtest</b> indicates the user name, <b>graphbase</b> indicates the graph name, and <b>Person</b> indicates the vertex type. | /user/vertex               |

| Parameter | Definition  | Example   |
|-----------|---|---|
| delimiter | Separator between properties of the data to be exported. The separator can be a common character, a special character, or a unicode. The separators specified for exporting the incoming and outgoing vertices must be the same. If you do not configure this parameter, commas (,) are used as the separators by default. This parameter is optional.    | \u002a  |
| filters   | Filtering conditions of properties in the data to be exported. The data format is JSONArray. For details, see the description for <i>PropertyFilter</i> in section "GraphBase" > "REST" > "Performing Full Graph Query on Vertices" in <i>MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)</i> . This parameter is optional | [<br>{"propertyName":"age","predicate":">>","values":[27]},<br>{"propertyName":"name","predicate":}"string_contains _prefix","values":["m"]}] |

## 2. Modify the **edge.properties** file.

The parameter configuration is listed in the following table.

| Parameter | Definition  | Example   |
|-----------|---|-----------|
| graphname | Name of the diagram instance to be operated. This parameter is mandatory. | graphbase |

| Parameter            | Definition  | Example   |
|----------------------|---|-----------|
| label                | Type of the edge to be exported. This parameter is mandatory.   | hasMember |
| outvertex.label      | Type of the outvertex at the two vertices corresponding to the edge to be exported. This parameter is mandatory.  | Forum     |
| invertex.label       | Type of the invertex at the two vertices corresponding to the edge to be exported. This parameter is mandatory.   | Person    |
| nvertex.primarykey   | The primary key of the incoming vertex of the edge to be exported. This parameter is mandatory.   | forumId   |
| outvertex.primarykey | The primary key of the outgoing vertex of the edge to be exported. This parameter is mandatory.   | personId  |
| propertykeylist      | Attribute key list of the edge to be exported. This parameter is optional. If multiple attributes need to be exported, use commas (,) to separate all attribute keys. | joinDate  |

| Parameter         | Definition   | Example    |
|-------------------|--|------------|
| dest.storage.path | HDFS path for storing the data to be exported. This parameter is optional and not recommended. Assume that the current domain name is HADOOP.COM, by default, the data is stored in <b>/user/graphtest@HADOO P.COM/graphbase/edge/friend</b> . In the preceding path, <b>graphtest</b> is the created username, <b>graphbase</b> is the graph name, and <b>friend</b> is the type of the edge. | /user/edge |
| delimiter         | Separator between attributes of the data to be exported. The separator can be a common character, a special character, or a unicode. The separators specified for exporting the incoming and outgoing vertices must be the same. If you do not configure this parameter, commas (,) are used as the separators by default. This parameter is optional.   | \u002a     |

| Parameter | Definition   | Example   |
|-----------|--|---|
| filters   | Filtering conditions of properties in the data to be exported. The data format is JSONArray. For details, see the description for <i>PropertyFilter</i> in section "GraphBase" > "REST" > "Performing Full Graph Query on Vertices" in <i>MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)</i> . This parameter is optional. | [<br>{"propertyName":"age","predicate":">>","values":[27]},<br>{"propertyName":"name","predicate":}"string_contains _prefix","values":["m"]}<br>] |

### 3. Modify the **resource.properties** file.

The parameter configurations are listed in the following table.

| Parameter             | Definition   | Example   |
|-----------------------|--|---|
| USERNAME              | Name of a user to be created.                                      | graphtest                                       |
| BINDING_SPARK_SERVICE | Bound Spark service, which is used to adapt to multiple instances. | Spark   |
| RESOURCE              | --executor-memory  | Memory size allocated to each executor in Spark |
|                       | --driver-memory  | Memory size allocated to driver in Spark        |
|                       | --num-executors  | Number of executors started by Spark            |
|                       | --executor-cores   | Number of cores started in each executor        |

| Parameter | Definition  | Example |
|-----------|---|---------|
|           | --conf spark.hbase.obtainToken.enabled<br>Whether to dynamically obtain the token information of the HBase. | true    |
|           | --conf spark.inputFormat.cache.enabled<br>Whether to enable the HBase cache                                 | false   |

### NOTE

The following special characters can be used as separators.

\u0020 Space  
\u0021 ! Exclamation mark  
\u0022 " Double quotation marks  
\u0023 # Pond key  
\u0024 \$ Currency symbol  
\u0025 % Percentage symbol  
\u0026 & And sign  
\u0027 ' Single quotation mark  
\u0028 ( Open parenthesis  
\u0029 ) Close parenthesis  
\u002a \* Asterisk  
\u002b + Plus sign  
\u002c , Comma  
\u002d - Hyphen

## Parameter Optimization

### 1. Parameter Description

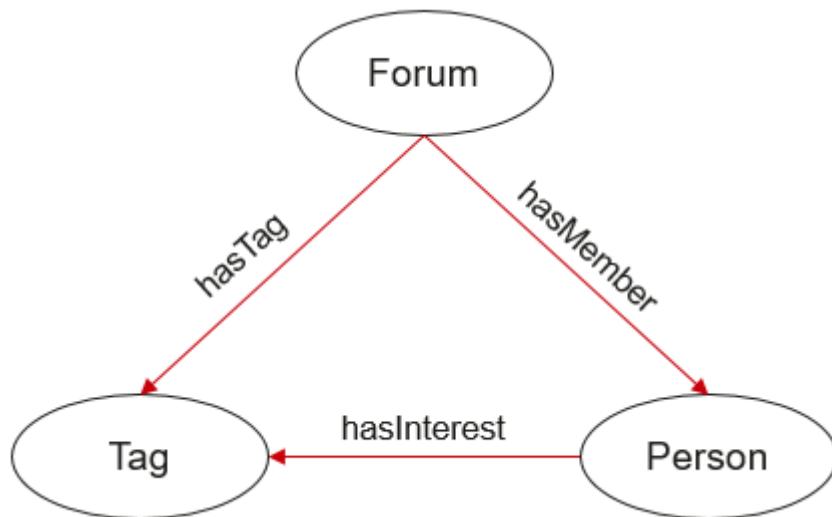
- In the **RESOURCE** configuration, the product of **num-executors** and **executor-cores** is the total number of **executor-core**, which determines the task concurrency capability.
- The product of the number of partitions and the number of nodes in the configuration item **graph.regions-per-server** in GraphBase is the total number of partitions, which determines the total number of started tasks.
- In the **RESOURCE** configuration, the product of **num-executors** and **executor-memory** is the total memory required by the task. The **yarn.nodemanager.resource.memory-mb** configuration item in Yarn determines the total memory that can be used.
- In the RESOURCE configuration, the value of **executor-memory** divided by **executor-cores** is the memory that can be used by each executor-core.

### 2. Parameter Optimization

- If the resources are sufficient, you can set the total number of executor-cores to be the same as the total number of partitions. In practice, you are advised to set **executor-cores** to a smaller value and increase the value of **num-executors**.
- The total memory configured for a task cannot exceed the total memory configured for Yarn, preventing memory overflow.
- If the memory is sufficient, you can increase the value of **executor-memory** to increase the memory size of each executor-core. You can also increase the value of **spark.yarn.executor.memoryOverhead** to dynamically increase executor memory to prevent performance deterioration caused by long GC time.

## Exporting Data

For example, the graph name is **graphbase** and the username is **graphtest**. The following figure shows the relationships between the vertex with label set to Forum, Person and Tag and the edge with label set to hasMember, hasInterest and hasTag.



1. Export the vertex.

Access the **graphreader** directory:

```
cd /opt/hadoopclient/GraphBase/graphreader
```

Run the following command to execute the environment variable:

```
source /opt/hadoopclient/bigdata_env
```

Authenticate the user and enter the user password.

```
kinit graphtest
```

- Export a vertex whose type is Forum.

Copy the configuration file:

```
cp conf/vertex.properties conf/Forum.properties
```

Modify the **Forum.properties** configuration file.

```
vim conf/vertex.properties
```

For example:

```
propertykeylist = forumId,title,creationDate  
dest.storage.path =  
graphname = graphbase  
label = Forum  
delimiter = \u002a  
filters =
```

Run the script to export data:

**./bin/graphReader.sh vertex Forum.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh vertex Forum.properties  
-----  
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2  
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false  
username=graphtest  
graphName=graphbase  
type=vertex  
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/Forum.properties  
delimiter=\u002a  
-----  
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your  
platform... using builtin-java classes where applicable |  
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)  
spark.yarn.am.memory is set but does not apply in cluster mode.  
spark.yarn.am.cores is set but does not apply in cluster mode.  
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in  
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)  
[root@10-5-199-1 graphreader]#
```

- Export the vertex whose type is Person.

Copy the configuration file:

**cp conf/vertex.properties conf/Person.properties**

Modify the **Person.properties** configuration file:

**vim conf/Person.properties**

For example:

```
propertykeylist = personId,firstName,lastName,gender,birthDay  
dest.storage.path =  
graphname = graphbase  
label = Person  
delimiter = \u002a  
filters =
```

Run the script to export data:

**./bin/graphReader.sh vertex Person.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh vertex Person.properties  
-----  
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2  
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false  
username=graphtest  
graphName=graphbase  
type=vertex  
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/Person.properties  
-----  
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your  
platform... using builtin-java classes where applicable |  
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)  
spark.yarn.am.memory is set but does not apply in cluster mode.  
spark.yarn.am.cores is set but does not apply in cluster mode.  
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
```

```
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
[root@10-5-199-1 graphreader]#
```

- Export the vertex whose type is Tag.

Copy the configuration file:

**cp conf/vertex.properties conf/Tag.properties**

Modify the **Tag.properties** configuration file:

**vim conf/Tag.properties**

For example:

```
propertykeylist = tagId,name,url
dest.storage.path =
graphname = graphbase
label = Tag
delimiter = \u002a
filters =
```

Run the script to export data:

**./bin/graphReader.sh vertex Tag.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh vertex Tag.properties
```

```
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false
username=graphtest
graphName=graphbase
type=vertex
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/Tag.properties
```

```
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable |
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)
```

spark.yarn.am.memory is set but does not apply in cluster mode.

spark.yarn.am.cores is set but does not apply in cluster mode.

```
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
```

```
[root@10-5-199-1 graphreader]#
```

## 2. Export the edge.

Access **graphreader** directory:

**cd /opt/hadoopclient/GraphBase/graphreader**

Run the following command to execute the environment variable:

**source /opt/hadoopclient/bigdata\_env**

Authenticate the user and enter the user password.

**kinit graphtest**

- Export the edge whose type is hasMember.

Copy the configuration file:

**cp conf/edge.properties conf/hasMember.properties**

Modify the **hasMember.properties** configuration file.

**vim conf/hasMember.properties**

For example:

```
invertex.primarykey = personId
outvertex.primarykey = forumId
propertykeylist =
invertex.label = Person
```

```
dest.storage.path =
graphname = graphbase
outvertex.label = Forum
label = hasMember
delimiter = \u002a
filters =
```

Run the script to export data:

**./bin/graphReader.sh edge hasMember.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh edge hasMember.properties
-----
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false
username=graphtest
graphName=graphbase
type=edge
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/hasMember.properties
-----
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable |
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)
spark.yarn.am.memory is set but does not apply in cluster mode.
spark.yarn.am.cores is set but does not apply in cluster mode.
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
[root@10-5-199-1 graphreader]#
```

- Export the edge whose type is hasInterest.

Copy the configuration file:

**cp conf/edge.properties conf/hasInterest.properties**

Modify the **hasInterest.properties** configuration file:

**vim conf/hasInterest.properties**

For example:

```
invertex.primarykey = tagId
outvertex.primarykey = personId
propertykeylist =
invertex.label = Tag
dest.storage.path =
graphname = graphbase
outvertex.label = Person
label = hasInterest
delimiter = \u002a
filters =
```

Run the script to export data:

**./bin/graphReader.sh edge hasInterest.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh edge hasInterest.properties
-----
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false
username=graphtest
graphName=graphbase
type=edge
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/hasInterest.properties
-----
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable |
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)
```

```
spark.yarn.am.memory is set but does not apply in cluster mode.
spark.yarn.am.cores is set but does not apply in cluster mode.
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
[root@10-5-199-1 graphreader]#
```

- Export the edge whose type is hasTag.

Copy the configuration file:

**cp conf/edge.properties conf/hasTag.properties**

Modify the **hasTag.properties** configuration file:

**vim conf/hasTag.properties**

For example:

```
invertex.primarykey = tagId
outvertex.primarykey = forumId
propertykeylist =
invertex.label = Tag
dest.storage.path =
graphname = graphbase
outvertex.label = Forum
label = hasTag
delimiter = \u002a
filters =
```

Run the script to export data:

**./bin/graphReader.sh edge hasTag.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh edge hasTag.properties
-----
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false
username=graphtest
graphName=graphbase
type=edge
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/hasTag.properties
-----
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable |
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)
spark.yarn.am.memory is set but does not apply in cluster mode.
spark.yarn.am.cores is set but does not apply in cluster mode.
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
[root@10-5-199-1 graphreader]#
```

## Viewing Execution Results

1. Check the execution process on Yarn.
  - a. Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Yarn**, and select **ResourceManager (Active)** in **Basic Information** on the **Overview** page. The Yarn WebUI is displayed.



- b. Click the task ID to view the task overview.

The screenshot shows the Hadoop Application Overview page for application ID application\_1552331280773\_0007. It displays various metrics such as Application Owner (root), Queue (Default), Application Name (graphbase\_1552331280773), Application Type (Non-MapReduce), and Application Status (RUNNING). The Application Metrics section includes counters for Total Resource Requested (10.00 GB), Total Number of Map Tasks Executed (0), Total Number of All Executors Preempted (0), Resource Requested from Current Attempt (0.00 GB), and Number of Non-All-Committed Task Attempts (0). The Application Resource Allocation section shows 1.000 MB seconds, 100 msec seconds, and 100% Resource Usage.

- c. Click **ApplicationMaster** to view the task execution progress.

The screenshot shows the Spark Jobs page with a single job listed. The job has a status of "Submitted" and is currently at step 0 of 1. The progress bar indicates 0% completion. The table below shows the stages of the job, with the first stage having 100% completed and the second stage having 0% completed.

2. Check the exported data on HDFS.

Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > HDFS**, and select **NameNode (Active)** in **Basic Information** on the **Overview** page. On the HDFS WebUI that is displayed, choose **Utilities > Browse the file system** to go to the root directory.

- Go to the Person vertex export directory and check the Person data.

The screenshot shows the HDFS Browse Directory page for the path /user/hive/graphbase\_157/vertices/Person. A table lists 11 files, all of which are part of the SUCCESS file. The files have sizes ranging from 0.8 KB to 22.1 KB and were last modified on Tue Mar 12 07:29:37 2019.

| Permission | Owner | Group  | Size     | Last Modified            | Replication | Block Size | Name       |
|------------|-------|--------|----------|--------------------------|-------------|------------|------------|
| -rw-r--r-- | dbc   | hadoop | 0.8      | Tue Mar 12 07:29:37 2019 | 3           | 128 MB     | _SUCCESS   |
| -rw-r--r-- | dbc   | hadoop | 16.23 KB | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00000 |
| -rw-r--r-- | dbc   | hadoop | 17.5 KB  | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00001 |
| -rw-r--r-- | dbc   | hadoop | 18.19 KB | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00002 |
| -rw-r--r-- | dbc   | hadoop | 19.41 KB | Tue Mar 12 07:29:35 2019 | 3           | 128 MB     | part-00003 |
| -rw-r--r-- | dbc   | hadoop | 14.3 KB  | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00004 |
| -rw-r--r-- | dbc   | hadoop | 14.37 KB | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00005 |
| -rw-r--r-- | dbc   | hadoop | 15.05 KB | Tue Mar 12 07:29:35 2019 | 3           | 128 MB     | part-00006 |
| -rw-r--r-- | dbc   | hadoop | 15.29 KB | Tue Mar 12 07:29:35 2019 | 3           | 128 MB     | part-00007 |
| -rw-r--r-- | dbc   | hadoop | 16.77 KB | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00008 |
| -rw-r--r-- | dbc   | hadoop | 15.49 KB | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00009 |
| -rw-r--r-- | dbc   | hadoop | 22.1 KB  | Tue Mar 12 07:29:21 2019 | 3           | 128 MB     | part-00010 |

- Go to the Tag vertex export directory and check Tag data.

| Browse Directory                    |       |        |          |                          |             |            |            |     |
|-------------------------------------|-------|--------|----------|--------------------------|-------------|------------|------------|-----|
| /user/dbc/graphbase_17/vertices/Tag |       |        |          |                          |             |            |            |     |
| Permission                          | Owner | Group  | Size     | Last Modified            | Replication | Block Size | Name       | Get |
| -r--r--r--                          | dbc   | hadoop | 0 B      | Tue Mar 12 07:53:05 2019 | 3           | 128 MB     | _SUCCESS   |     |
| -r--r--r--                          | dbc   | hadoop | 20.81 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00000 |     |
| -r--r--r--                          | dbc   | hadoop | 23.52 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00001 |     |
| -r--r--r--                          | dbc   | hadoop | 29.84 KB | Tue Mar 12 07:53:17 2019 | 3           | 128 MB     | part-00002 |     |
| -r--r--r--                          | dbc   | hadoop | 26.68 KB | Tue Mar 12 07:53:31 2019 | 3           | 128 MB     | part-00003 |     |
| -r--r--r--                          | dbc   | hadoop | 22.79 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00004 |     |
| -r--r--r--                          | dbc   | hadoop | 21.36 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00005 |     |
| -r--r--r--                          | dbc   | hadoop | 21.99 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00006 |     |
| -r--r--r--                          | dbc   | hadoop | 22.54 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00007 |     |
| -r--r--r--                          | dbc   | hadoop | 21.95 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00008 |     |
| -r--r--r--                          | dbc   | hadoop | 25.27 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00009 |     |
| -r--r--r--                          | dbc   | hadoop | 28.73 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00010 |     |

- Go to the hasInterest edge export directory and check the hasInterest data.

| Browse Directory                         |       |        |           |                          |             |            |            |     |
|--|-------|--------|-----------|--------------------------|-------------|------------|------------|-----|
| /user/dbc/graphbase_17/edges/hasInterest |       |        |           |                          |             |            |            |     |
| Permission                               | Owner | Group  | Size      | Last Modified            | Replication | Block Size | Name       | Get |
| -r--r--r--                               | dbc   | hadoop | 0 B       | Tue Mar 12 08:22:55 2019 | 3           | 128 MB     | _SUCCESS   |     |
| -r--r--r--                               | dbc   | hadoop | 65.42 KB  | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00000 |     |
| -r--r--r--                               | dbc   | hadoop | 72.06 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00001 |     |
| -r--r--r--                               | dbc   | hadoop | 47.31 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00002 |     |
| -r--r--r--                               | dbc   | hadoop | 70.57 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00003 |     |
| -r--r--r--                               | dbc   | hadoop | 57.21 KB  | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00004 |     |
| -r--r--r--                               | dbc   | hadoop | 80.04 KB  | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00005 |     |
| -r--r--r--                               | dbc   | hadoop | 72.99 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00006 |     |
| -r--r--r--                               | dbc   | hadoop | 100.39 KB | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00007 |     |
| -r--r--r--                               | dbc   | hadoop | 67.7 KB   | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00008 |     |
| -r--r--r--                               | dbc   | hadoop | 60.81 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00009 |     |
| -r--r--r--                               | dbc   | hadoop | 76.42 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00010 |     |

### NOTE

Data export formats of the Date and Geoshape are as follows:

- Date

During the import, the data may be the time of the long type: 1552894952164 or time of the date type: Mon Mar 18 15:43:34 GMT+08:00 2019.

During the export, the format is 2016-03-18 15:45:18.

- Geoshape

Data to be imported may be longitude and latitude: 23,52 (latitude, longitude) or a circle area: 23,52,18 (latitude,longitude,radius).

After the data export, if a comma (,) is used as the separator, the data format remains unchanged: 23,52 (latitude, longitude) or 23,52,18 (latitude,longitude,radius).

### 1.9.3.5 REST API Development

REST APIs are provided by GraphBase 8.3.1 and used by developers for secondary development based on the GraphBase platform.

The GraphBase API provides unified access for upper-layer applications and encapsulates the unified operations performed on the GraphBase platform through highly flexible REST APIs.

When an HTTP requester sets up a connection to the server and sends an HTTP request, no other requests can be sent from the connection until responses to the sent request are received.

## 1.9.3.6 REST API Invocation Examples and Running Results

### 1.9.3.6.1 Example

This section describes an example of creating a PropertyKey API. For details about how to create other APIs, see **GraphBase** in *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*. The code snippet is in the `GraphBaseRestExample` class in the `com.huawei.util` package of the sample project. After code is run, the result is returned on the console.

#### NOTE

1. Ensure that the input parameters and the invoked URL are correct.
2. Ensure that the entered IP address, port number, username, and password are correct.
1. Initialize the configuration of `graphbase.properties`. For details, see [Preparing for Security Authentication](#).
2. Obtain the connection information and perform security authentication.

```
GraphHttpClient client = null;
InputStream inputStream = null;
inputStream = new FileInputStream(System.getProperty("user.dir") + File.separator +
DEFAULT_CONFIG_FILE);
Properties p = new Properties();
p.load(inputStream);
HttpAuthInfo httpAuthInfo = HttpAuthInfo.newBuilder()
.setIp(p.getProperty("ip"))
.setPort(Integer.valueOf(p.getProperty("port")))
.setService(RestApi.SERVICE)
.setUsername(p.getProperty("userName"))
/*.setPassword(p.getProperty("password"))*/
.setKeytabFile(System.getProperty("user.dir") + File.separator + KEY_TAB)
.build();
client = GraphHttpClient.newKeytabClient(httpAuthInfo);
RestApi api = new RestApi(client);
```

3. Prepare the metadata-related classes, including `PropertyKey`, `EdgeLabel`, and `VertexLabel` in `com.huawei.entity` of the sample code.
4. Create the related REST APIs, for example, a property key API. (For details, see the sample project `com.huawei.graphbase.RestApi`.)

```
//Create a propertyKey interface.
public boolean addPropertyKey(PropertyKey propertyKey, String graphName) {
    JSONObject rspJson = null;
    try {
        //Invoke the REST API URL and transfer the request parameter propertyKey.
        rspJson = sendHttpPostReq("/graph/" + graphName + "/schema/property-key",
        RestHelper.toJsonString(propertyKey));
    } catch (JsonProcessingException e) {
        LOG.info(ERROR_MSG);
    }
    //Parse the returned rspJson parameter.
    final boolean isSuccessfully = (null == rspJson ? false : rspJson.getString("code").equals("0"));
    //Print logs to the console.
    String log = String.format("[%s]: create property key[%s] %s",
        (isSuccessfully ? "SUCCESS" : "FAIL"),
        propertyKey.getName(),
        (isSuccessfully ? "successfully." : "fail."));
    System.out.println(log);
    return isSuccessfully;
}
```

```
}

//Send a request containing CSRF_TOKEN.
private JSONObject sendHttpPutReq(String uri, JSONObject reqJson) {
    return sendHttpPutReq(uri, reqJson.toString());
}

private JSONObject sendHttpPutReq(String uri) {
    return sendHttpPutReq(uri, "");
}

private JSONObject sendHttpPutReq(String uri, String reqJsonStr) {
    String fullUrl = buildUrl(uri);
    HttpPut httpPut = new HttpPut(fullUrl);
    httpPut.addHeader(CSRF_TOKEN, this.client.csrfToken);

    CloseableHttpResponse response = null;
    JSONObject rspJson = null;

    try {
        if (reqJsonStr != null && !reqJsonStr.isEmpty()) {
            httpPut.setEntity(new StringEntity(reqJsonStr, ContentType.create("text/plain",
Consts.UTF_8)));
        }
        printHttpReqHeader(httpPut);
        printHttpReqBody(reqJsonStr);

        response = client.httpClient.execute(httpPut);
        RestHelper.checkHttpRsp(response);

        rspJson = new JSONObject(EntityUtils.toString(response.getEntity(), Charset.forName("UTF-8")));
        printHttpRspBody(rspJson);
    } catch (Exception e) {
        LOG.info(e.getMessage());
        return null;
    } finally {
        if (null != response) {
            try {
                response.close();
            } catch (IOException e) {
                // nothing to do
            }
        }
    }
}

return rspJson;
}
```

### 1.9.3.6.2 Creating or Deleting a Graph

To invoke Rest APIs to perform operations on graphs, refer to the following examples provided in com.huawei.graphbase.GraphBaseRestExample:

Create a graph.

```
String graphName = "graphbase";
api.createGraph(graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph HTTP/1.1
REQ BODY: {"graphName":"graphbase"}
RSP BODY: {
    "msg": "create graph success.",
    "code": "0"
}
[SUCCESS]: create graph[graphbase] successfully.
```

Delete a graph.

```
String graphName = "graphbase";
api.deleteGraph(graphName);
```

The command output is as follows:

```
REQ HEADER: DELETE https://10.7.66.150:22380/graphbase/graph?graphName=graphbase HTTP/1.1
REQ BODY: null
RSP BODY: {
    "msg": "delete graph success.",
    "code": "0"
}
[SUCCESS]: delete graph[graphbase] successfully.
```

### 1.9.3.6.3 Creating a Schema by Uploading an XML File



**graphName** specifies the name of a graph. This parameter is required when the specified REST API is invoked.

In the example, the com.huawei.graphbase.GraphBaseRestExample class has defined global variables. You can modify the configuration as required.

```
String graphName = "graphbase";
```

1. Save the XML file defined in [Compiling a Schema File \(XML\)](#) to a local directory.
2. Import the sample project. For details, see [Configuring and Importing Sample Projects](#).
3. Invoke the REST API of the addSchema class in **com.huawei.graphbase.GraphBaseRestExample**.

```
File file = new File(System.getProperty("user.dir") + File.separator + SCHEMA_FILE);
api.addSchema(file, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema HTTP/1.1
RSP BODY: {
    "msg": "Make schema file successful",
    "code": "0"
}
[SUCCESS]: create schema successfully.
```

### 1.9.3.6.4 Creating a Schema

The following code snippets are used as an example. For complete codes, see **com.huawei.graphbase.GraphBaseRestExample**.

- Create a property key.

```
PropertyKey propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.String);
propertyKey.setName("name");
api.addPropertyKey(propertyKey, graphName);
propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.Integer);
propertyKey.setName("age");
api.addPropertyKey(propertyKey, graphName);
propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.String);
propertyKey.setName("telephone");
api.addPropertyKey(propertyKey, graphName);
propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.Float);
propertyKey.setName("weight");
api.addPropertyKey(propertyKey, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/property-key  
HTTP/1.1  
REQ BODY: {"name":"name","dataType":"String"}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {  
        "dataType": "String",  
        "name": "name",  
        "cardinality": "SINGLE"  
    }  
}  
[SUCCESS]: create property key[name] successfully.
```

- Query a property key based on the name.  
api.queryPropertyKey("name",graphName);

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/property-key/  
p_name HTTP/1.1  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {  
        "dataType": "String",  
        "name": "name",  
        "cardinality": "SINGLE",  
        "ttl":0  
    }  
}  
[SUCCESS]: query property key[name] successfully.
```

- Query all property keys.  
api.queryAllPropertyKey(graphName);

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/property-key  
HTTP/1.1  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"propertyKeyList": [  
        {  
            "dataType": "String",  
            "name": "name",  
            "cardinality": "SINGLE",  
            "ttl":0  
        },  
        {  
            "dataType": "Integer",  
            "name": "age",  
            "cardinality": "SINGLE",  
            "ttl":0  
        },  
        {  
            "dataType": "String",  
            "name": "telephone",  
            "cardinality": "SINGLE",  
            "ttl":0  
        },  
        {  
            "dataType": "Float",  
            "name": "weight",  
            "cardinality": "SINGLE",  
            "ttl":0  
        }  
    ]}  
}  
[SUCCESS]: query all property key successfully.  
===== Description of the JSON data result
```

```
=====
|---msg          message
|---code         code
|---data         Result data storage location
|---propertyKeyList   Property key list.
|---dataType      Data type of the property value.
|---name          Property name.
|---cardinality   Property value types. The values are as follows: SINGLE, SET, and LIST.
SINGLE specifies that there is only one property value. SET specifies that there are multiple different values for a property. LIST specifies that there are multiple repeated values for a property.
=====
```

- Create a vertex label.

```
api.addVertexLabel("person",graphName);
api.addVertexLabel("phone",graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/vertex-label
HTTP/1.1
REQ BODY: {"name":"person"}
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {
    "name": "person"
  }
}
[SUCCESS]: create vertex label[person] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/vertex-label
HTTP/1.1
REQ BODY: {"name":"person"}
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {
    "name": "phone"
  }
}
[SUCCESS]: create vertex label[phone] successfully.
```

- Query a vertex label based on the name.

```
api.queryVertexLabel("person",graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/vertex-label/person
HTTP/1.1
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {
    "name": "person"
  }
}
[SUCCESS]: query vertex label[person] successfully.
```

- Query all vertex labels.

```
api.queryAllVertexLabel(graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.5.199.50:22380/graphbase/graph/graphbase/schema/vertex-label
HTTP/1.1
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {"vertexLabelList": [
    {
      "name": "person"
    },
    {
      "name": "phone"
    }
  ]
}
```

```
        }
    ]
}
[SUCCESS]: query all vertex label successfully.
```

- Create an edge label.

```
EdgeLabel edgeLabel = new EdgeLabel();
edgeLabel.setName("friend");
api.addEdgeLabel(edgeLabel,graphName);
edgeLabel = new EdgeLabel();
edgeLabel.setName("knows");
api.addEdgeLabel(edgeLabel,graphName);
edgeLabel = new EdgeLabel();
edgeLabel.setName("call");
api.addEdgeLabel(edgeLabel,graphName);
edgeLabel = new EdgeLabel();
edgeLabel.setName("has");
api.addEdgeLabel(edgeLabel,graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/edge-label
HTTP/1.1
REQ BODY: {"name":"friend"}
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {
    "multiplicity": "MULTI",
    "name": "friend",
    "ttl": 0
  }
}
[SUCCESS]: create edge label[friend] successfully.
```

- Query an edge label based on the name.

```
api.queryEdgeLabel("friend",graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/edge-label/
friend
HTTP/1.1
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {
    "multiplicity": "MULTI",
    "name": "friend",
    "ttl": 0
  }
}
[SUCCESS]: query edge label[friend] successfully.
```

## Periodic Data Deletion

GraphBase provides the following periodic data deletion functions:

1. When creating a property key, you can set the **ttl** parameter for the property. The value must be an integer, in seconds. For example, if you set **ttl** for the **name** property to **120**, the **name** property of all vertices will be deleted 120s later after being created.

This parameter takes effect only for vertex properties, and does not take effect for edge properties.

Note: If a vertex has only one property and the **ttl** parameter is set for the property, the vertex cannot be queried using this property after the property is deleted. However, this vertex can be queried by performing full-graph traversal, and the vertex ID is also recorded on the corresponding edge of the

vertex. You are not advised to configure **ttl** for the unique property of a vertex.

2. When creating an edge label, you can set the **ttl** parameter for the edge label. The value must be an integer, in seconds. For example, if you set **ttl** for the **know** label to **120**, edges labeled by the **know** label will be deleted 120s later after being created.

Note: The **ttl** parameter is not allowed for data stored in Elasticsearch. For example, if a MIXED index is created for the property of an edge labeled by **know**, the index data of the edge cannot be deleted with the edge 120s later, and the deleted edge cannot be found using the index. To delete the index data, you are advised to create a MIXED index for the corresponding edge property and recreate the index. After the index is successfully recreated, delete the original MIXED index.

#### NOTICE

If **ttl** is set to **0** or a negative number, the corresponding property or edges are not deleted. The default value is **0**.

### 1.9.3.6.5 Creating an Index

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

- Create an index.

#### NOTE

MIXED: specifies the mixed index that is a kind of external hybrid match index. This type of indexes searches vertices and edges using any combination of indexes created in the databases. The property key corresponding to such indexes is in the `<key, value>` format. The options of `value` are as follows: **TEXT**, **TEXTSTRING**, **PREFIX\_TREE**, and **DEFAULT** (these values are case insensitive.) External indexes are used for non-exact match queries.

COMPOSITE: specifies the composite index that is a kind of internal exact match index. This type of indexes retrieve vertices or edges using a fixed combination of one or more properties. Such indexes are accurate, simple, and efficient, and can greatly accelerate the retrieval speed, but have high requirements on the property sequence in the combination. The property key corresponding to such indexes is in the `<key, "">` format. Built-in indexes are used for exact match queries.

You can create an index only after a schema is created and before a vertex or edge is created. Otherwise, data cannot be queried. You need to recreate indexes to query data.

```
//Create indexes for all vertices and edges.  
addIndex(api, graphName);  
private static void addIndex(RestApi api, String graphName) {  
//Create a built-in index for the vertex property name.  
    GraphIndexReqObj graphIndexReqObj = new GraphIndexReqObj();  
    graphIndexReqObj.setElementCategory(ElementCategory.VERTEX);  
    graphIndexReqObj.setName("name_index");  
    graphIndexReqObj.setType(IndexType.COMPOSITE);  
    List<KeyTextType> keyTextTypeList1 = new ArrayList<>();  
    KeyTextType keyTextType = new KeyTextType();  
    keyTextType.setName("name");  
    KeyTextType.setTextType(""); //If IndexType is set to COMPOSITE, TextType is empty.  
    keyTextTypeList1.add(keyTextType);  
    graphIndexReqObj.setKeyTextTypeList(keyTextTypeList1);
```

```
api.addGraphIndex(graphIndexReqObj, graphName);
//Create a built-in index for the vertex property age.
graphIndexReqObj = new GraphIndexReqObj();
graphIndexReqObj.setElementCategory(ElementCategory.VERTEX);
graphIndexReqObj.setName("age_index");
graphIndexReqObj.setType(IndexType.COMPOSITE);
List<KeyTextType> keyTextTypeList2 = new ArrayList<>();
keyTextType = new KeyTextType();
keyTextType.setName("age");
keyTextType.setTextType(""); //If IndexType is set to COMPOSITE, TextType is empty.
keyTextTypeList2.add(keyTextType);
graphIndexReqObj.setKeyTextTypeList(keyTextTypeList2);
api.addGraphIndex(graphIndexReqObj, graphName);
//Create a mixed index for the vertex property telephone.
graphIndexReqObj = new GraphIndexReqObj();
graphIndexReqObj.setElementCategory(ElementCategory.VERTEX);
graphIndexReqObj.setName("telephone_index");
graphIndexReqObj.setType(IndexType.COMPOSITE);
List<KeyTextType> keyTextTypeList3 = new ArrayList<>();
keyTextType = new KeyTextType();
keyTextType.setName("telephone");
keyTextType.setTextType(""); //If IndexType is set to COMPOSITE, TextType is empty.
keyTextTypeList3.add(keyTextType);
graphIndexReqObj.setKeyTextTypeList(keyTextTypeList3);
api.addGraphIndex(graphIndexReqObj, graphName);
...//Creates a mixed index for the edge property weight.
graphIndexReqObj = new GraphIndexReqObj();
graphIndexReqObj.setElementCategory(ElementCategory.EDGE);
graphIndexReqObj.setName("weight_index");
graphIndexReqObj.setType(IndexType.MIXED);
List<KeyTextType> keyTextTypeList4 = new ArrayList<>();
keyTextType = new KeyTextType();
keyTextType.setName("weight");
keyTextType.setTextType("DEFAULT"); //If IndexType is set to MIXED, TextType is transferred based
on the information provided in the interface document.
keyTextTypeList4.add(keyTextType);
graphIndexReqObj.setKeyTextTypeList(keyTextTypeList4);
api.addGraphIndex(graphIndexReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph
HTTP/1.1
REQ BODY: {"name":"name_index","elementCategory":"VERTEX","unique":false,"keyTextTypeList": [
  {"name":"name","textType":""}], "type":"COMPOSITE"}
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {"indexName": "name_index"}
}
[SUCCESS]: create COMPOSITE graph index[name_index] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph
HTTP/1.1
REQ BODY: {"name":"age_index","elementCategory":"VERTEX","unique":false,"keyTextTypeList": [
  {"name":"age","textType":""}], "type":"COMPOSITE"}
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {"indexName": "age_index"}
}
[SUCCESS]: create COMPOSITE graph index[age_index] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph
HTTP/1.1
REQ BODY: {"name":"telephone_index","elementCategory":"VERTEX","unique":false,"keyTextTypeList": [
  {"name":"telephone","textType":"STRING"}], "type":"MIXED"}
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {"indexName": "telephone_index"}
```

```
[SUCCESS]: create MIXED graph index[telephone_index] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph  
HTTP/1.1  
REQ BODY: {"name":"weight_index","elementCategory":"EDGE","unique":false,"keyTextTypeList":  
[{"name":"weight","textType":"DEFAULT"}],"type":"MIXED"}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"indexName": "weight_index"}  
}  
[SUCCESS]: create MIXED graph index[weight_index] successfully.
```

- Recreate an index.

```
api.reCreateGraphIndex("name_index", graphName);  
api.reCreateGraphIndex("age_index", graphName);  
api.reCreateGraphIndex("telephone_index", graphName);  
api.reCreateGraphIndex("weight_index", graphName);
```

The command output is as follows:

```
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/  
name_index HTTP/1.1  
REQ BODY:  
RSP BODY: {  
    "msg": "Asynchronous task submitted successfully.",  
    "code": "0",  
    "data": {"id": "1433552"}  
}  
[SUCCESS]: reCreate graph index[name_index] successfully.  
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/  
age_index HTTP/1.1  
REQ BODY:  
RSP BODY: {  
    "msg": "Asynchronous task submitted successfully.",  
    "code": "0",  
    "data": {"id": "2482128"}  
}  
[SUCCESS]: reCreate graph index[age_index] successfully.  
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/  
telephone_index HTTP/1.1  
REQ BODY:  
RSP BODY: {  
    "msg": "Asynchronous task submitted successfully.",  
    "code": "0",  
    "data": {"id": "1135248"}  
}  
[SUCCESS]: reCreate graph index[telephone_index] successfully.  
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/  
weight_index HTTP/1.1  
REQ BODY:  
RSP BODY: {  
    "msg": "Asynchronous task submitted successfully.",  
    "code": "0",  
    "data": {"id": "2183824"}  
}  
[SUCCESS]: reCreate graph index[weight_index] successfully.
```

- Query all indexes.

```
api.queryAllIndex(graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index HTTP/1.1  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"indexList": [  
        {  
            "elementCategory": "VERTEX",  
            "name": "name_index",  
            "propertyKeyStatusList": [{  
                "name": "name",  
                "status": "OK"  
            }]  
        }  
    ]  
}
```

```
        "status": "ENABLED"
    }],
    "type": "COMPOSITE"
},
{
    "elementCategory": "VERTEX",
    "name": "age_index",
    "propertyKeyStatusList": [
        {
            "name": "age",
            "status": "ENABLED"
        }
    ],
    "type": "COMPOSITE"
},
{
    "elementCategory": "VERTEX",
    "name": "telephone_index",
    "propertyKeyStatusList": [
        {
            "name": "telephone",
            "status": "ENABLED"
        }
    ],
    "type": "MIXED"
},
{
    "elementCategory": "EDGE",
    "name": "weight_index",
    "propertyKeyStatusList": [
        {
            "name": "weight",
            "status": "ENABLED"
        }
    ],
    "type": "MIXED"
}
]
}
]
}

[SUCCESS]: query all index successfully.
```

### 1.9.3.6.6 Creating and Querying a Vertex or an Edge

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

- Create a vertex.

```
//Create four nodes labeled as person 1, person 4, person 5, and person 8.
addVertexPerson(api, graphName);
private static void addVertexPerson(RestApi api, String graphName) {
...//Create the node person 1.
    AddVertexReqObj addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("person");
    List<PropertyReqObj> propertyReqObjList1 = new ArrayList<>();
    PropertyReqObj propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("name");
    propertyReqObj.setValue("marko");
    propertyReqObjList1.add(propertyReqObj);
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("age");
    propertyReqObj.setValue(29);
    propertyReqObjList1.add(propertyReqObj);
    addVertexReqObj.setPropertyList(propertyReqObjList1);
    api.addVertex(addVertexReqObj, graphName);
//Create the node person 4.
    addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("person");
    List<PropertyReqObj> propertyReqObjList2 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("name");
    propertyReqObj.setValue("josh");
    propertyReqObjList2.add(propertyReqObj);
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("age");
    propertyReqObj.setValue(32);
```

```
propertyReqObjList2.add(propertyReqObj);
addVertexReqObj.setPropertyList(propertyReqObjList2);
api.addVertex(addVertexReqObj, graphName);
...//Create the node person 5.
addVertexReqObj = new AddVertexReqObj();
addVertexReqObj.setVertexLabel("person");
List<PropertyReqObj> propertyReqObjList3 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("name");
propertyReqObj.setValue("vadas");
propertyReqObjList3.add(propertyReqObj);
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("age");
propertyReqObj.setValue(27);
propertyReqObjList3.add(propertyReqObj);
addVertexReqObj.setPropertyList(propertyReqObjList3);
api.addVertex(addVertexReqObj, graphName);
...//Create the node person 8.
addVertexReqObj = new AddVertexReqObj();
addVertexReqObj.setVertexLabel("person");
List<PropertyReqObj> propertyReqObjList4 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("name");
propertyReqObj.setValue("blame");
propertyReqObjList4.add(propertyReqObj);
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("age");
propertyReqObj.setValue(30);
propertyReqObjList4.add(propertyReqObj);
addVertexReqObj.setPropertyList(propertyReqObjList4);
api.addVertex(addVertexReqObj, graphName);
}
//Create four nodes labeled as phone 2, phone 3, phone 6, and phone 7, respectively.
addVertexPhone(api, graphName);
private static void addVertexPhone(RestApi api, String graphName) {
    //Create the node phone 2.
    AddVertexReqObj addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("phone");
    List<PropertyReqObj> propertyReqObjList1 = new ArrayList<>();
    PropertyReqObj propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("telephone");
    propertyReqObj.setValue("15000000000");
    propertyReqObjList1.add(propertyReqObj);
    addVertexReqObj.setPropertyList(propertyReqObjList1);
    api.addVertex(addVertexReqObj, graphName);
    //Create the node phone 3.
    addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("phone");
    List<PropertyReqObj> propertyReqObjList2 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("telephone");
    propertyReqObj.setValue("1522222222");
    propertyReqObjList2.add(propertyReqObj);
    addVertexReqObj.setPropertyList(propertyReqObjList2);
    api.addVertex(addVertexReqObj, graphName);
    //Create the node phone 6.
    addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("phone");
    List<PropertyReqObj> propertyReqObjList3 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("telephone");
    propertyReqObj.setValue("13777777777");
    propertyReqObjList3.add(propertyReqObj);
    addVertexReqObj.setPropertyList(propertyReqObjList3);
    api.addVertex(addVertexReqObj, graphName);
    //Create the node phone 7.
    addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("phone");
    List<PropertyReqObj> propertyReqObjList4 = new ArrayList<>();
```

```
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("telephone");
propertyReqObj.setValue("13666666666");
propertyReqObjList4.add(propertyReqObj);
addVertexReqObj.setPropertyList(propertyReqObjList4);
api.addVertex(addVertexReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"marko"}, {"name":"age","value":29}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1261176"}
}
[SUCCESS]: add vertex[person] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"josh"}, {"name":"age","value":32}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1170064"}
}
[SUCCESS]: add vertex[person] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"vadas"}, {"name":"age","value":27}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1747680"}
}
[SUCCESS]: add vertex[person] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"blame"}, {"name":"age","value":30}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1478712"}
}
[SUCCESS]: add vertex[person] successfully.

REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"15000000000"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "2218640"}
}
[SUCCESS]: add vertex[phone] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"15222222222"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1297704"}
}
[SUCCESS]: add vertex[phone] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"13777777777"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1508048"}
}
[SUCCESS]: add vertex[phone] successfully.
```

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"13666666666"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "2556624"}
}
[SUCCESS]: add vertex[phone] successfully.
```

- Query a vertex based on the vertex ID.

```
String vertexId = getVertexIdByProperty(api,graphName,"person","name","marko");
api.queryVertex(vertexId, graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/vertex/1261176 HTTP/1.1
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {
        "propertyList": [
            {
                "name": "name",
                "id": "99vz-r14o-4if9",
                "value": "marko"
            },
            {
                "name": "age",
                "id": "c30v-r14o-5wzp",
                "value": "29"
            }
        ],
        "id": "1261176",
        "vertexLabel": "person"
    }
}
[SUCCESS]: query vertex[1261176] successfully.
```

- Create an edge.

```
//Create the following six edges: 9, 10, 11, 12, 13, 14, 15, and 16.
addEdges(api, graphName);
private static void addEdges(RestApi api, String graphName) {
...//Create the edge 9 in the scenario.
    AddEdgeReqObj addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("knows");
    addEdgeReqObj.setOutVertexId("1261176");
    addEdgeReqObj.setInVertexId("1747680");
    List<PropertyReqObj> propertyReqObjList1 = new ArrayList<>();
    PropertyReqObj propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("weight");
    propertyReqObj.setValue(0.4);
    propertyReqObjList1.add(propertyReqObj);
    addEdgeReqObj.setPropertyList(propertyReqObjList1);
    api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 10 in the scenario.
    addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("has");
    addEdgeReqObj.setOutVertexId("1261176");
    addEdgeReqObj.setInVertexId("2218640");
    List<PropertyReqObj> propertyReqObjList2 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("weight");
    propertyReqObj.setValue(0.8);
    propertyReqObjList2.add(propertyReqObj);
    addEdgeReqObj.setPropertyList(propertyReqObjList2);
    api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 11 in the scenario.
    addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("has");
    addEdgeReqObj.setOutVertexId("1170064");
    addEdgeReqObj.setInVertexId("1297704");
```

```
List<PropertyReqObj> propertyReqObjList3 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("weight");
propertyReqObj.setValue(0.8);
propertyReqObjList3.add(propertyReqObj);
addEdgeReqObj.setPropertyList(propertyReqObjList3);
api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 12 in the scenario.
addEdgeReqObj = new AddEdgeReqObj();
addEdgeReqObj.setEdgeLabel("knows");
addEdgeReqObj.setOutVertexId("1170064");
addEdgeReqObj.setInVertexId("1478712");
List<PropertyReqObj> propertyReqObjList4 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("weight");
propertyReqObj.setValue(0.4);
propertyReqObjList4.add(propertyReqObj);
addEdgeReqObj.setPropertyList(propertyReqObjList4);
api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 13 in the scenario.
addEdgeReqObj = new AddEdgeReqObj();
addEdgeReqObj.setEdgeLabel("has");
addEdgeReqObj.setOutVertexId("1478712");
addEdgeReqObj.setInVertexId("2556624");
List<PropertyReqObj> propertyReqObjList5 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("weight");
propertyReqObj.setValue(0.8);
propertyReqObjList5.add(propertyReqObj);
addEdgeReqObj.setPropertyList(propertyReqObjList5);
api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 14 in the scenario. Edge 14 is a bidirectional edge, and you need to specify an incoming edge, and an outgoing edge.
addEdgeReqObj = new AddEdgeReqObj();
addEdgeReqObj.setEdgeLabel("call");
addEdgeReqObj.setOutVertexId("2556624");
addEdgeReqObj.setInVertexId("1508048");
List<PropertyReqObj> propertyReqObjList6 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("weight");
propertyReqObj.setValue(0.6);
propertyReqObjList6.add(propertyReqObj);
addEdgeReqObj.setPropertyList(propertyReqObjList6);
api.addEdge(addEdgeReqObj, graphName);
addEdgeReqObj = new AddEdgeReqObj();
addEdgeReqObj.setEdgeLabel("call");
addEdgeReqObj.setOutVertexId("1508048");
addEdgeReqObj.setInVertexId("2556624");
List<PropertyReqObj> propertyReqObjList7 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("weight");
propertyReqObj.setValue(0.6);
propertyReqObjList7.add(propertyReqObj);
addEdgeReqObj.setPropertyList(propertyReqObjList7);
api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 15 in the scenario.
addEdgeReqObj = new AddEdgeReqObj();
addEdgeReqObj.setEdgeLabel("has");
addEdgeReqObj.setOutVertexId("1508048");
addEdgeReqObj.setInVertexId("1747680");
List<PropertyReqObj> propertyReqObjList8 = new ArrayList<>();
propertyReqObj = new PropertyReqObj();
propertyReqObj.setName("weight");
propertyReqObj.setValue(0.8);
propertyReqObjList8.add(propertyReqObj);
addEdgeReqObj.setPropertyList(propertyReqObjList8);
api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 16 in the scenario.
addEdgeReqObj = new AddEdgeReqObj();
```

```
        addEdgeReqObj.setEdgeLabel("friend");
        addEdgeReqObj.setOutVertexId("1261176");
        addEdgeReqObj.setInVertexId("2016584");
        List<PropertyReqObj> propertyReqObjList9 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("weight");
        propertyReqObj.setValue(1.0);
        propertyReqObjList9.add(propertyReqObj);
        addEdgeReqObj.setPropertyList(propertyReqObjList9);
        api.addEdge(addEdgeReqObj, graphName);
    }
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1261176","inVertexId":"1747680","edgeLabel":"knows","propertyList": [{"name":"weight","value":0.4}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "ew5r-r14o-bj9x-11gio"}
}
[SUCCESS]: add edge[knows] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1261176","inVertexId":"2218640","edgeLabel":"has","propertyList": [{"name":"weight","value":0.8}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "hpan-r14o-ecet-1bjww"}
}
[SUCCESS]: add edge[has] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1170064","inVertexId":"1297704","edgeLabel":"has","propertyList": [{"name":"weight","value":0.8}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "nqky-p2ts-ecet-rtbc"}
}
[SUCCESS]: add edge[has] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1170064","inVertexId":"1478712","edgeLabel":"knows","propertyList": [{"name":"weight","value":0.4}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "qjpu-p2ts-bj9x-vozc"}
}
[SUCCESS]: add edge[knows] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1478712","inVertexId":"2556624","edgeLabel":"has","propertyList": [{"name":"weight","value":0.8}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "gflz-vozc-ecet-1ispc"}
}
[SUCCESS]: add edge[has] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"2556624","inVertexId":"1508048","edgeLabel":"call","propertyList": [{"name":"weight","value":0.6}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "kzq2-1ispc-cxud-wbm8"}
}
[SUCCESS]: add edge[call] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1508048","inVertexId":"2556624","edgeLabel":"call","propertyList": [{"name":"weight","value":0.6}]}
```

```
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "nsuy-wbm8-cxud-1ispC"}
}
[SUCCESS]: add edge[call] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1508048","inVertexId":"1747680","edgeLabel":"has","propertyList":[{"name":"weight","value":0.8}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "qlzu-wbm8-ecet-11gio"}
}
[SUCCESS]: add edge[has] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1261176","inVertexId":"1170064","edgeLabel":"friend","propertyList":[{"name":"weight","value":1.0}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "kifj-r14o-a4ph-p2ts"}
}
[SUCCESS]: add edge[friend] successfully.
```

- Query an edge based on the edge ID.

```
String edgeld = getEdgeldByProperty(api,graphName,"call","weight","0.6");
api.queryEdge(edgeld, graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.5.199.173:22380/graphbase/graph/graphbasejpps/edge/b8lu-mhlc-5x-mhig HTTP/1.1
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "propertyList": [
            {
                "name": "weight",
                "value": "0.6"
            }
        ],
        "inVertexId": "1049128",
        "edgeLabel": "call",
        "id": "b8lu-mhlc-5x-mhig",
        "outVertexId": "1049232"
    }
}
[SUCCESS]: query edge[b8lu-mhlc-5x-mhig] successfully.
```

### 1.9.3.6.7 Performing a Full Graph Query

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

- Perform full graph query for vertices.

```
//Perform full graph query for vertices in the following scenario: To query the users who are older than 29 years old and list the users by age in ascending order. The upper limit of the number of users is 3.
```

```
VertexSearchReqObj vertexSearchReqObj = new VertexSearchReqObj();
vertexSearchReqObj.setVertexLabel("person");
List<PropertyFilter> propertyFilterList = new ArrayList<>();
PropertyFilter propertyFilter = new PropertyFilter();
propertyFilter.setPropertyName("age");
propertyFilter.setPredicate(PropertyPredicate.GREATER_THAN);
List<Integer> values = new ArrayList();
values.add(29);
propertyFilter.setValues(values);
propertyFilterList.add(propertyFilter);
vertexSearchReqObj.setFilterList(propertyFilterList);
List<PropertyKeySort> sortList = new ArrayList<>();
```

```
PropertyKeySort propertyKeySort = new PropertyKeySort();
propertyKeySort.setPropertyKeyName("age");
propertyKeySort.setSortType("ASC");
sortList.add(propertyKeySort);
vertexSearchReqObj.setPropertyKeySortList(sortList);
vertexSearchReqObj.setLimit(3);
//Perform full graph query for vertices.
api.searchVertex(vertexSearchReqObj, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex/search HTTP/1.1
REQ BODY: {"filterList":[{"propertyName":"age","predicate":">>","values":
[29]}],"vertexLabel":"person","propertyKeySortList":
[{"propertyKeyName":"age","sortType":"ASC"}],"limit":3}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {
        "totalHits": 2,
        "vertexList": [
            {
                "propertyList": [
                    {
                        "name": "name",
                        "id": "atc7-vozc-4if9",
                        "value": "blame"
                    },
                    {
                        "name": "age",
                        "id": "dmh3-vozc-5wzp",
                        "value": "30"
                    }
                ],
                "id": "1478712",
                "vertexLabel": "person"
            },
            {
                "propertyList": [
                    {
                        "name": "name",
                        "id": "9owi-p2ts-4if9",
                        "value": "josh"
                    },
                    {
                        "name": "age",
                        "id": "c1e-p2ts-5wzp",
                        "value": "32"
                    }
                ],
                "id": "1170064",
                "vertexLabel": "person"
            }
        ]
    }
}
[SUCCESS]: search vertex [person] successfully.
=====
Description of the JSON data result
format=====
|---msg          message
|---code         code
|---data         Result data storage location
|---totalHits   Number of vertices
|---vertexList  Vertex set
|---vertexLabel Vertex label
|---id           Vertex ID
|---propertyList Property set
|---name         Property name
|---value        Property value
|---id           Property ID
```

- Perform full graph query for edges.

```
//Perform full graph query for edges in the following scenario: To query the edge that is labeled as
knows, and the value of the edge property weight is equal to or less than 0.6.
EdgeSearchReqObj edgeSearchReqObj = new EdgeSearchReqObj();
edgeSearchReqObj.setEdgeLabel("knows");
edgeSearchReqObj.setLimit(2);
List<PropertyFilter> propertyFilterList = new ArrayList<>();
PropertyFilter propertyFilter = new PropertyFilter();
propertyFilter.setPropertyName("weight");
propertyFilter.setPredicate(PropertyPredicate.LESS_THAN_EQUAL);
List<Float> values = new ArrayList();
values.add(new Float(0.6));
propertyFilter.setValues(values);
propertyFilterList.add(propertyFilter);
edgeSearchReqObj.setFilterList(propertyFilterList);
api.searchEdge(edgeSearchReqObj, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge/search HTTP/1.1
REQ BODY: {"filterList": [{"propertyName": "weight", "predicate": "<=", "values": [0.6]}, {"edgeLabel": "knows", "limit": 2}]
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {
        "totalHits": 2,
        "edgeList": [
            {
                "propertyList": [
                    {
                        "name": "weight",
                        "value": "0.4"
                    }
                ],
                "inVertexId": "1747680",
                "edgeLabel": "knows",
                "outVertexId": "1261176"
            },
            {
                "propertyList": [
                    {
                        "name": "weight",
                        "value": "0.4"
                    }
                ],
                "inVertexId": "1478712",
                "edgeLabel": "knows",
                "outVertexId": "1170064"
            }
        ]
    }
}
[SUCCESS]: search edge [knows] successfully.
```

===== Description of the JSON data result  
format=====

|                 |                              |
|-----------------|------------------------------|
| ---msg          | message                      |
| ---code         | code                         |
| ---data         | Result data storage location |
| ---totalHits    | Number of edges              |
| ---edgeList     | Edge set                     |
| ---edgeLabel    | Edge label                   |
| ---outVertexId  | Start vertex ID              |
| ---inVertexId   | End vertex ID                |
| ---propertyList | Property set                 |
| ---name         | Property name                |
| ---value        | Property value               |

=====

### 1.9.3.6.8 Performing a Path Query

- Graph traversal: Access the vertices of the graph along the side from any vertex of the graph. Graph traversal is a basic operation in graph databases, and it is similar to tree traversal. In a narrow sense, a vertex can be accessed only once in a graph traversal. The graph traversal described in this document is that in a broad sense.
- Graph path: indicates the set of vertices and edges that passed by during graph traversal. The number of edges in a path is called the path depth.

Use marko in [Typical Application Scenario](#) as an example, the paths formed by the vertices and edges found based on the first layer expansion query are as follows:

- marko -[friend]-> josh
- marko -[has]-> 15000000000
- marko -[knows]-> vadas

In this example, the person name or mobile number indicates the entity object. -[relationship]-> is used to indicate the edge, and the relationship in the square brackets([]) indicates the edge label. The depth of these paths is 1.

Continue traversing based on the preceding path result, and obtain a second layer edge and a vertex of marko to form a new path. The original path is named the parent path of the new path, and the new path is named the sub-path of the original path.

Example:

- Original path (path A): marko -[friend]-> josh
- New path (path B): marko -[friend]-> josh -[has]-> 1522222222
- Path A is the parent path of path B, and path B is the sub-path of path A.

- Full path: indicates all graph paths between two vertices that meet specified conditions.

Scenario 1: Query the full path between node 1 and node 8.

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

```
allPathSearch(api, graphName);
private static void allPathSearch(RestApi api, String graphName) {
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "marko"));
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "blame"));
    pathSearchReqObj.setVertexIdList(vertexIdList);
    pathSearchReqObj.setLayer(7);
    api.searchPath(pathSearchReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/default/paths HTTP/1.1
```

```
REQ BODY: {"vertexIdList":["1793880","2842456"],"layer":7}
```

```
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {"pathSet": [
    {
      "edgeList": [
        {
          "propertyList": [
            {
              "name": "weight",
              "value": "0.8"
            }
          ]
        }
      ]
    }
  ]}
}
```

```
"start": "2842456",
"end": "1683696",
"id": "a4x7-1ox94-700l-1035c",
"label": "has",
"type": "edge"
},
{
"propertyList": [
{
"name": "weight",
"value": "0.6"
}],
"start": "1561040",
"end": "1683696",
"id": "3hkq-xgi8-5lg5-1035c",
"label": "call",
"type": "edge"
},
{
"propertyList": [
{
"name": "weight",
"value": "0.4"
}],
"start": "2016584",
"end": "2842456",
"id": "6kdl-17808-46vp-1ox94",
"label": "knows",
"type": "edge"
},
{
"propertyList": [
{
"name": "weight",
"value": "0.4"
}],
"start": "1793880",
"end": "3065160",
"id": "4inf-12g60-46vp-1tp3c",
"label": "knows",
"type": "edge"
},
{
"propertyList": [
{
"name": "weight",
"value": "0.6"
}],
"start": "1683696",
"end": "1561040",
"id": "3pni-1035c-5lg5-xgi8",
"label": "call",
"type": "edge"
},
{
"propertyList": [
{
"name": "weight",
"value": "0.8"
}],
"start": "3065160",
"end": "1561040",
"id": "9dih-1tp3c-700l-xgi8",
"label": "has",
"type": "edge"
},
{
"propertyList": [
{
"name": "weight",
"value": "1.0"
}],
"start": "1793880",
"end": "2016584",
"id": "cy23-12g60-2sb9-17808",
```

```
        "label": "friend",
        "type": "edge"
    },
],
"start": "1793880",
"vertexList": [
{
    "labelList": ["phone"],
    "propertyList": [
        {
            "name": "telephone",
            "value": "13666666666"
        }
    ],
    "id": "1683696",
    "type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "josh"
        },
        {
            "name": "age",
            "value": "32"
        }
    ],
    "id": "2016584",
    "type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "vadas"
        },
        {
            "name": "age",
            "value": "27"
        }
    ],
    "id": "3065160",
    "type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "marko"
        },
        {
            "name": "age",
            "value": "29"
        }
    ],
    "id": "1793880",
    "type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "blame"
        },
        {
            "name": "age",
            "value": "34"
        }
    ]
}]
```

```

        "value": "30"
    },
    "id": "2842456",
    "type": "vertex"
},
{
    "[labelList": ["phone"],
    "propertyList": [
        {
            "name": "telephone",
            "value": "13777777777"
        }
    ],
    "id": "1561040",
    "type": "vertex"
}
],
"end": "2842456",
"pathList": [
    [
        [
            "cy23-12g60-2sb9-17808",
            "6kdl-17808-46vp-1ox94"
        ],
        [
            [
                "4inf-12g60-46vp-1tp3c",
                "9dih-1tp3c-700l-xgi8",
                "3pni-1035c-5lg5-xgi8",
                "a4x7-1ox94-700l-1035c"
            ],
            [
                "4inf-12g60-46vp-1tp3c",
                "9dih-1tp3c-700l-xgi8",
                "3hkq-xgi8-5lg5-1035c",
                "a4x7-1ox94-700l-1035c"
            ]
        ]
    ]
}
]
}
]

[SUCCESS]: search path successfully.
=====
Description of the JSON data result
format=====
|---msg           message
|---code          code
|---data          Result data storage location
|---pathSet       All path data. Multiple vertex pairs correspond to multiple data structures.
|---start         Start node ID
|---end           End node ID
|---edgeList      Detailed data of all relationships in the data result
|---vertexList    Detailed data of all vertices included in the current data result
|---pathList      Path set. Each path stores only the relationship access sequence table. Only
relationship IDs are recorded.
=====
```

Build a path based on the path result (applicable to all paths):

- Find a path from the path list. For example:

```
[
    "cy23-12g60-2sb9-17808",
    "6kdl-17808-46vp-1ox94"
]
```

- Find an edge from the edge list based on the edge ID. For example:

Find the edge with the ID of cy23-12g60-2sb9-17808:

```
{
    "propertyList": [
        {
            "name": "weight",
            "value": "1.0"
        }
    ],
    "start": "1793880",
}
```

```

    "end": "2016584",
    "id": "cy23-12g60-2sb9-17808",
    "label": "friend",
    "type": "edge"
}

```

- c. Identify the start vertex, obtain the end node ID based on the edge data in **b**, and obtain the vertex data from the vertex list. For example:

Start from the edge with the ID of cy23-12g60-2sb9-17808 to find the end node with the vertex ID of 2016584

```

{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "josh"
        },
        {
            "name": "age",
            "value": "32"
        }
    ],
    "id": "2016584",
    "type": "vertex"
}

```

- d. Repeat **b** and **c** until the data in **a** is exhausted.

The path in the preceding example is simplified as follows:

```
(1793880,marko) -[cy23-12g60-2sb9-17808:friend]-> (2016584,josh)
-[6kdl-17808-46vp-1ox94:knows]-> (2842456:blame)
```

Vertices are marked in parentheses and in the format of (Vertex ID, Person name). Edges are marked in square brackets and in the format of -[Edge ID:Edge type]->

- Shortest path

It indicates the shortest path between two vertices, specifically the path with the smallest sum of relationship weights. This document considers that the path with the smallest sum of path relationship weights is not the shortest path. Instead, the shortest path must comply with the specified weight rule. If the weight calculation rule is not specified, the shortest path is the path with the minimum path depth by default. The shortest path complying with the relationship weighting rule is the weight-based shortest path.

Scenario 2: Query the shortest path from node 1 to node 8.

The following code snippets are used as an example. For complete codes, see **com.huawei.graphbase.GraphBaseRestExample**.

```

shortestPathSearch(api, graphName);
private static void shortestPathSearch(RestApi api, String graphName) {
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "marko"));
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "blame"));
    pathSearchReqObj.setVertexIdList(vertexIdList);
    pathSearchReqObj.setLayer(7);
    ...PathSearchReqObj.setOption("shortest");//If option is not transferred, the default value is all, that
    is, the full path.
    api.searchPath(pathSearchReqObj, graphName);
}

```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/default/paths HTTP/1.1
```

```
REQ BODY: {"vertexIdList":["1793880","2842456"],"layer":7,"option":"shortest"}
```

```
RSP BODY: {
```

```
    "msg": "success",
```

```
"code": "0",
"data": {"pathSet": [
  "edgeList": [
    {
      "propertyList": [
        {"name": "weight", "value": "0.4"}
      ],
      "start": "2016584",
      "end": "2842456",
      "id": "6kdl-17808-46vp-1ox94",
      "label": "knows",
      "type": "edge"
    },
    {
      "propertyList": [
        {"name": "weight", "value": "1.0"}
      ],
      "start": "1793880",
      "end": "2016584",
      "id": "cy23-12g60-2sb9-17808",
      "label": "friend",
      "type": "edge"
    }
  ],
  "start": "1793880",
  "vertexList": [
    {
      "labelList": ["person"],
      "propertyList": [
        {
          "name": "name",
          "value": "josh"
        },
        {
          "name": "age",
          "value": "32"
        }
      ],
      "id": "2016584",
      "type": "vertex"
    },
    {
      "labelList": ["person"],
      "propertyList": [
        {
          "name": "name",
          "value": "marko"
        },
        {
          "name": "age",
          "value": "29"
        }
      ],
      "id": "1793880",
      "type": "vertex"
    },
    {
      "labelList": ["person"],
      "propertyList": [
        {
          "name": "name",
          "value": "blame"
        },
        {
          "name": "age",
          "value": "30"
        }
      ]
    }
  ]
]}
```

```

        ],
        "id": "2842456",
        "type": "vertex"
    }
],
"end": "2842456",
"pathList": [
    "cy23-12g60-2sb9-17808",
    "6kdl-17808-46vp-1ox94"
]
}]}
}
[SUCCESS]: search path successfully.
=====
Description of the JSON data result
format=====
|---msg           message
|---code          code
|---data          Result data storage location
|---pathSet       All path data. Multiple vertex pairs correspond to multiple data structures.
|---start         Start node ID
|---end           End node ID
|---edgeList      Detailed data of all relationships in the data result
|---vertexList    Detailed data of all vertices included in the current data result
|---pathList      Path set. Each path stores only the relationship access sequence table. Only
relationship IDs are recorded.
=====
```

- Scenario 3: Query the paths where the **age** values of all nodes are greater than or equal to 29 between node 1 and node 8.

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

```
vertexFilterPathSearch(api, graphName);
private static void vertexFilterPathSearch(RestApi api, String graphName) {
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "marko"));
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "blame"));
    pathSearchReqObj.setVertexIdList(vertexIdList);
    VertexFilter vertexFilter = new VertexFilter();
    List<PropertyFilter> propertyFilterList3 = new ArrayList<>();
    PropertyFilter propertyFilter = new PropertyFilter();
    propertyFilter.setProperty("age");
    propertyFilter.setPredicate(PropertyPredicate.GREATER_THAN_EQUAL);
    List<Integer> values2 = new ArrayList();
    values2.add(29);
    propertyFilter.setValues(values2);
    propertyFilterList3.add(propertyFilter);
    vertexFilter.setFilterList(propertyFilterList3);
    pathSearchReqObj.setVertexFilter(vertexFilter);
    pathSearchReqObj.setLayer(7);
    api.searchPath(pathSearchReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/default/paths HTTP/1.1
REQ BODY: {"vertexIdList":["1793880","2842456"],"layer":7,"vertexFilter":{"filterList":
[{"propertyName":"age","predicate":">=","values":[29]}],"limit":0}}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"pathSet": [
        {
            "edgeList": [
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "0.4"
                        }
                    ],
                    "start": "2016584",
                    "end": "2842456"
                }
            ]
        }
    ]}}
```

```
"end": "2842456",
"id": "6kdl-17808-46vp-1ox94",
"label": "knows",
"type": "edge"
},
{
"propertyList": [
{
"name": "weight",
"value": "1.0"
}],
"start": "1793880",
"end": "2016584",
"id": "cy23-12g60-2sb9-17808",
"label": "friend",
"type": "edge"
}
],
"start": "1793880",
"vertexList": [
{
"labelList": ["person"],
"propertyList": [
{
"name": "name",
"value": "josh"
},
{
"name": "age",
"value": "32"
}
],
"id": "2016584",
"type": "vertex"
},
{
"labelList": ["person"],
"propertyList": [
{
"name": "name",
"value": "marko"
},
{
"name": "age",
"value": "29"
}
],
"id": "1793880",
"type": "vertex"
},
{
"labelList": ["person"],
"propertyList": [
{
"name": "name",
"value": "blame"
},
{
"name": "age",
"value": "30"
}
],
"id": "2842456",
"type": "vertex"
}
],
"end": "2842456",
"pathList": [
"cy23-12g60-2sb9-17808",
"6kdl-17808-46vp-1ox94"
]
```

```

        ]}
    }
[SUCCESS]: search path successfully.
=====
format=====
|---msg           message
|---code          code
|---data          Result data storage location
|---pathSet       All path data. Multiple vertex pairs correspond to multiple data structures.
|---start         Start node ID
|---end           End node ID
|---edgeList      Detailed data of all relationships in the data result
|---vertexList    Detailed data of all vertices included in the current data result
|---pathList      Path set. Each path stores only the relationship access sequence table. Only
relationship IDs are recorded.
=====
=====
```

### 1.9.3.6.9 Performing a Line Expansion Query

- Line Expansion Query
  - Similar to graph traversal, a line expansion query starts from a specified vertex and traverses the incident edges to find the target vertex. In a graph traversal, the process of finding the vertices that have direct connection with the specified vertex is the first layer expansion for the vertex.
  - The graph traversal proceeds with the query from these vertices directly connected with the start vertex to find the vertices that are indirectly connected with the start vertex. This process is called second layer expansion. In this way the graph traversal will expand n layers to find all vertices that are directly and indirectly connected with the start vertex.

```

//Scenario: Satrt from node 1 and expand two layers.
//Layer 1 query condition: To query the vertices whose age is equal to or less than 29.
//Layer 2 query condition: To query the verticeswhose vertexlabel is phone.
lineSearch(api, graphName);
private static void lineSearch(RestApi api, String graphName) {
    LineSearchReqObj lineSearchReqObj = new LineSearchReqObj();
    List<Integer> vertexIdList = new ArrayList<>();
    String vertexId = getVertexIdByProperty(api, graphName, "person", "name", "marko");
    vertexIdList.add(Integer.valueOf(vertexId));
    lineSearchReqObj.setVertexIdList(vertexIdList);
    List<VertexFilter> vertexFilterList = new ArrayList<>();
    VertexFilter vertexFilter = new VertexFilter();
    List<PropertyFilter> propertyFilterList1 = new ArrayList<>();
    PropertyFilter propertyFilter = new PropertyFilter();
    propertyFilter.setProperty("age");
    propertyFilter.setPredicate(PropertyPredicate.LESS_THAN_EQUAL);
    List<Integer> values2 = new ArrayList();
    values2.add(29);
    propertyFilter.setValues(values2);
    propertyFilterList1.add(propertyFilter);
    vertexFilter.setFilterList(propertyFilterList1);
    vertexFilterList.add(vertexFilter);
    vertexFilter = new VertexFilter();
    List<String> vertexLabelList = new ArrayList<>();
    vertexLabelList.add("phone");
    vertexFilter.setVertexLabelList(vertexLabelList);
    vertexFilterList.add(vertexFilter);
    lineSearchReqObj.setVertexFilterList(vertexFilterList);
    lineSearchReqObj.setLayer(2);
    lineSearchReqObj.setLimit(5);
    api.searchLines(lineSearchReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graphbase/lines HTTP/1.1
REQ BODY: {"vertexIdList":[1261176],"layer":2,"vertexFilterList":[{"filterList":[{"propertyName":"age","predicate":"<=","values":[29]}],"limit":0}],"vertexLabelList":["phone"],"limit":0}],"limit":5}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {
        "edgeList": [
            {
                "propertyList": [
                    {
                        "name": "weight",
                        "value": "0.4"
                    }
                ],
                "start": "1261176",
                "end": "1747680",
                "id": "ew5r-r14o-bj9x-11gio",
                "label": "knows",
                "type": "edge"
            },
            {
                "propertyList": [
                    {
                        "name": "weight",
                        "value": "0.8"
                    }
                ],
                "start": "1508048",
                "end": "1747680",
                "id": "qlzu-wbm8-ecet-11gio",
                "label": "has",
                "type": "edge"
            }
        ],
        "vertexList": [
            {
                "labelList": ["person"],
                "propertyList": [
                    {
                        "name": "name",
                        "value": "marko"
                    },
                    {
                        "name": "age",
                        "value": "29"
                    }
                ],
                "id": "1261176",
                "type": "vertex"
            },
            {
                "labelList": ["person"],
                "propertyList": [
                    {
                        "name": "name",
                        "value": "vadas"
                    },
                    {
                        "name": "age",
                        "value": "27"
                    }
                ],
                "id": "1747680",
                "type": "vertex"
            },
            {
                "labelList": ["phone"],
                "propertyList": [
                    {
                        "name": "telephone",
                        "value": "13777777777"
                    }
                ]
            }
        ]
    }
}
```

```
        },
        "id": "1508048",
        "type": "vertex"
    }
}
}

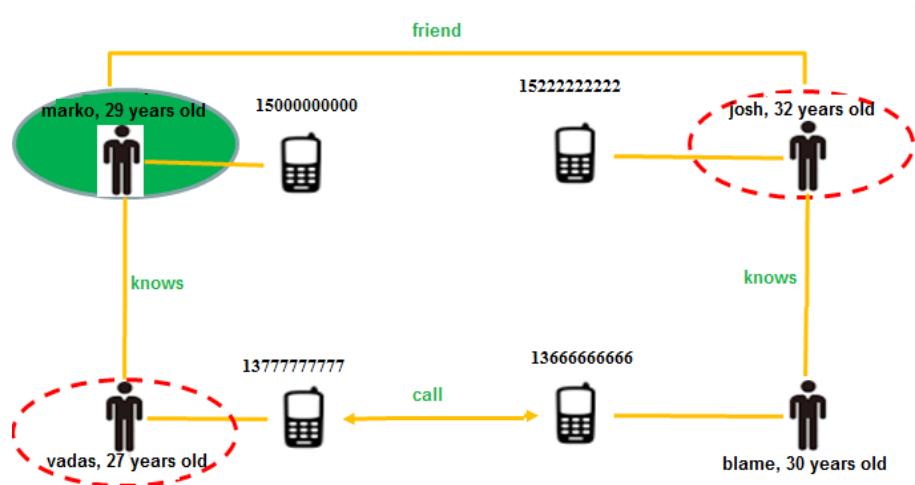
[SUCCESS]: search path successfully.
=====
Description of the JSON data result
=====
|---msg          message
|---code         code
|---data         Result data storage location
|---edgeList     All edges contained in the query result.
|---vertexList   All vertices contained in the query result.
=====
```

### 1.9.3.6.10 Analyzing the Typical Application Scenario

#### Direct Contact Search

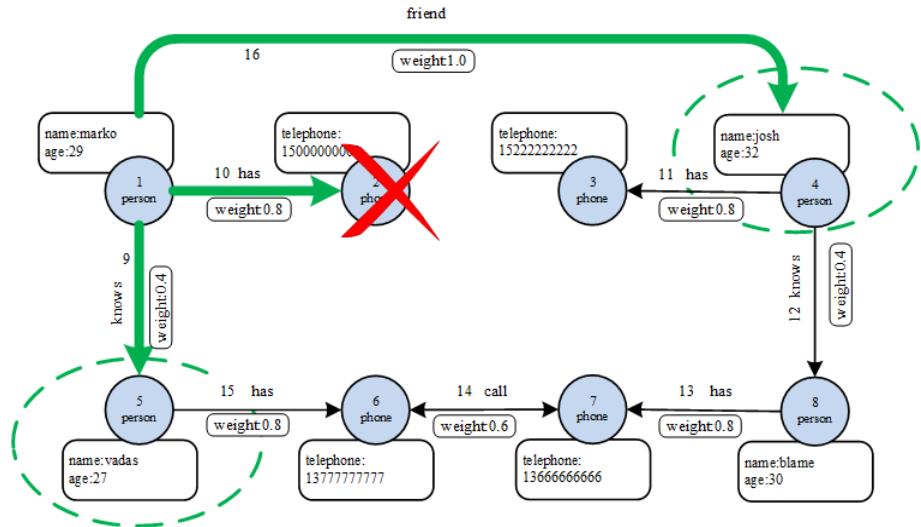
Direct contact: A person who has direct contact with a specified person.

The following graph shows the relationship between persons and their direct contacts in the typical scenario. (marko is used as an example. The direct contacts of marko are marked using the red dotted ovals.)



In GraphBase, the line expansion query function can be used to implement this scenario. [Figure 1-108](#) shows how to find the direct contacts for marko. (The green arrows indicate the expansion direction, the red cross mark indicates that the vertex does not meet the search condition, and the green dotted ovals indicate the search result.)

**Figure 1-108 Direct contact search in GraphBase**



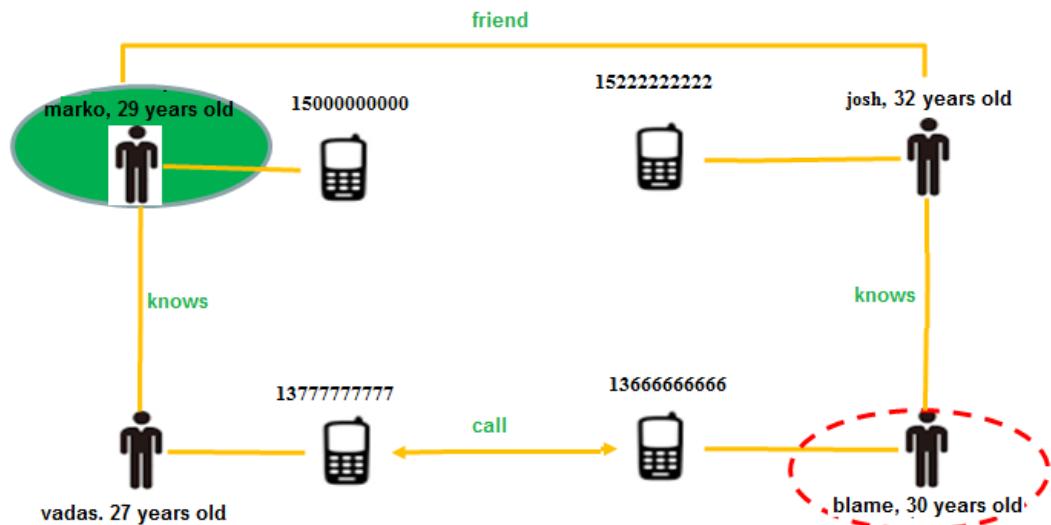
As shown in the preceding graph, josh and vadas are directly connected with marko. The related code is as follows:

```
/*
 * <p> Search for the direct contact</p>
 * @param api REST API encapsulation
 * @param graphName Graph name
 * @param startNodeId Start node ID
 * NOTE: The search result is the response data of the end node.
 */
static String findDirectRelations(RestApi api, String graphName, String startNodeId)
{
    // Node preparation
    LineSearchReqObj lineSearchReqObj = new LineSearchReqObj();
    List<Integer> vertexIdList = new ArrayList<>();
    vertexIdList.add(Integer.valueOf(startNodeId));
    lineSearchReqObj.setVertexIdList(vertexIdList);
    // Query depth
    lineSearchReqObj.setLayer(1);
    // End node condition (the label is person)
    List<VertexFilter> vertexFilterList = new ArrayList<>();
    VertexFilter vertexFilter = new VertexFilter();
    List<String> vertexLabelList = new ArrayList<>();
    vertexLabelList.add("person");
    vertexFilter.setVertexLabelList(vertexLabelList);
    vertexFilterList.add(vertexFilter);
    lineSearchReqObj.setVertexFilterList(vertexFilterList);
    // Perform line expansion query. (The query result is displayed as strings in JSON format.)
    String result = api.searchLines(lineSearchReqObj, graphName);
    return result;
}
```

## Indirect Contact Search

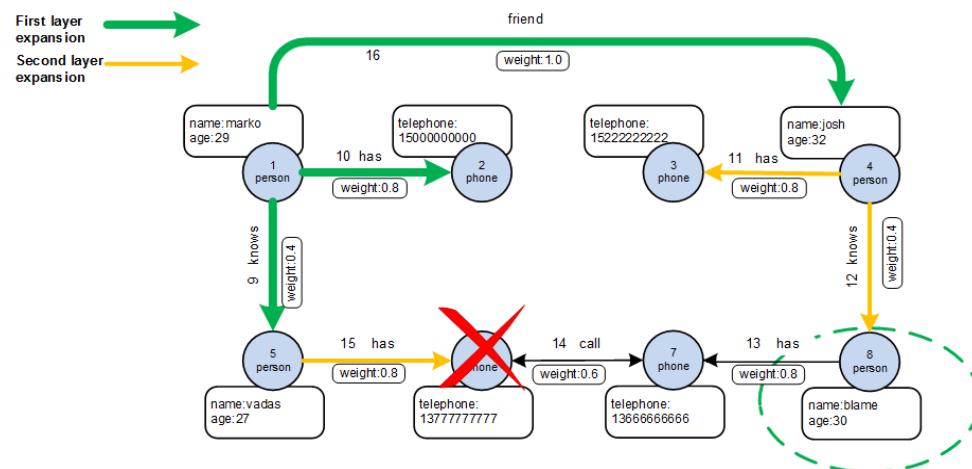
Indirect contact: A person who has indirect contact with a specified person.

The following graph shows the relationship between persons and their indirect contacts in the typical scenario. (marko is used as an example. The indirect contacts of marko are marked using the red dotted ovals.)



In GraphBase, the line expansion query can be used to find the indirect contact of marko in this scenario. [Figure 1-109](#) shows how the indirect contacts are searched in GraphBase. In the scenario, two layers are expanded: the green arrows indicate the query at the first layer, the yellow arrows indicate the query at the second layer. The red cross indicates that the vertex does not meet the search criteria, and the green dotted oval marks out the search result.

**Figure 1-109** Graph example of indirect contact search



As shown in the preceding graph, only one link between marko and blame meets the search criteria. The related code is as follows:

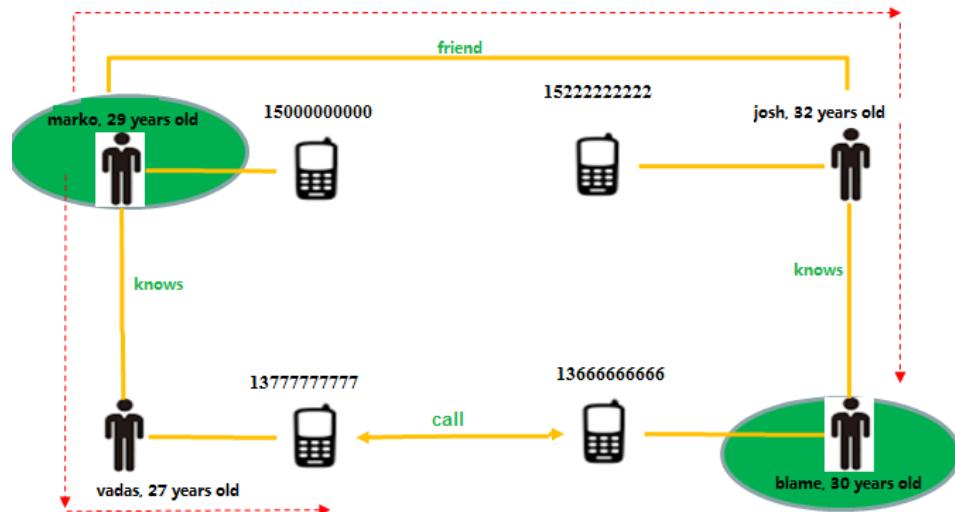
```
/*
 * <p> Search for the indirect contact</p>
 * @param api REST API encapsulation
 * @param graphName Graph name
 * @param startNodeID Start node ID
 * NOTE: The search result is the response data of the end node.
 */
static String findIndirectRelations(RestApi api, String graphName, String startNodeID)
{
    // Node preparation
    LineSearchReqObj lineSearchReqObj = new LineSearchReqObj();
    List<Integer> vertexIdList = new ArrayList<>();
    vertexIdList.add(Integer.valueOf(startNodeID));
```

```
lineSearchReqObj.setVertexIdList(vertexIdList);
// Query depth
lineSearchReqObj.setLayer(2);
// End node condition (the vertex label is person)
List<VertexFilter> vertexFilterList = new ArrayList<>();
VertexFilter vertexFilter = new VertexFilter();
List<String> vertexLabelList = new ArrayList<>();
vertexLabelList.add("person");
vertexFilter.setVertexLabelList(vertexLabelList);
vertexFilterList.add(vertexFilter);
lineSearchReqObj.setVertexFilterList(vertexFilterList);
// Perform line expansion query. (The query result is displayed as strings in JSON format.)
String result = api.searchLines(lineSearchReqObj, graphName);
return result;
}
```

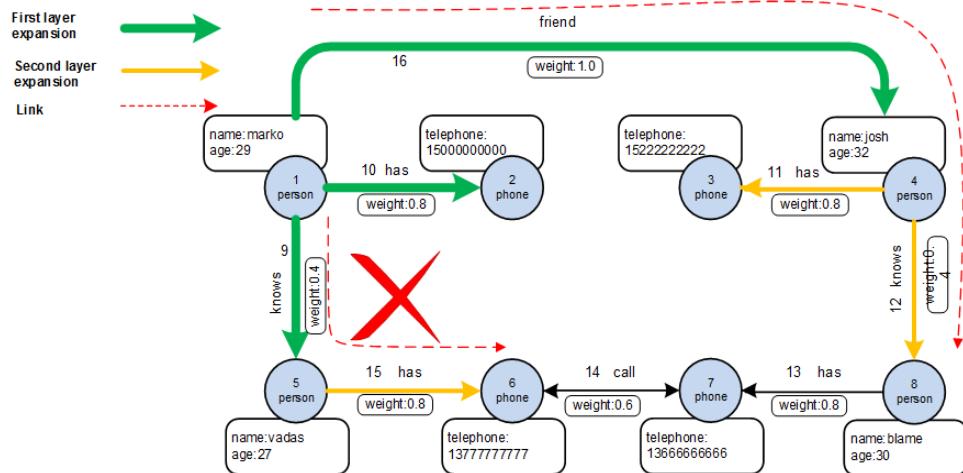
## Search for Link

Link refers to the link between two real world entities.

In the real world, the link between two persons is shown in the following graph. (The link between marko and blame is used as an example. The red dashed arrows indicate the link directions.)



In the link search scenario, the full path query function needs to be used. The following graph shows how to search for links in GraphBase. In this graph, the link between marko and blame is used as an example, and the path depth must be within two layers. The green arrows indicate the first layer expansion, the yellow arrow indicate the line expansion at the second layer, the red dotted arrows indicate the link direction, and the red cross indicates that the link does not meet the requirements.



As shown in the preceding graph, only one link between marko and blame meets the search conditions. The related code is as follows:

```

/*
 * <p> Search for links </p>
 * @param api      REST API encapsulation
 * @param graphName Graph name
 * @param startNodeID Start node ID
 * @param endNodeID End node ID
 * NOTE: The result data is the path set of the response data.
 */
static String findLinkedPaths(RestApi api, String graphName, String startNodeID, String endNodeID) {
    // Node preparation
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(startNodeID);
    vertexIdList.add(endNodeID);
    pathSearchReqObj.setVertexIdList(vertexIdList);
    // Query depth.
    pathSearchReqObj.setLayer(2);
    // Perform full path query. (The query result is displayed as strings in JSON format).
    String result = api.searchPath(pathSearchReqObj, graphName);
    return result;
}


```

### 1.9.3.7 Gremlin Console

#### Prerequisites

You have installed all component clients required in the cluster.

#### Performing Query Using Gremlin Statements

1. Use PuTTY to log in to the node where the client is installed as user **root**.
2. Go to the **/GraphBase/gremlinserver/gremlin-console** directory where the GraphBase client is installed.

`cd /opt/hadoopclient/GraphBase/gremlinserver/gremlin-console`



In this example, the GraphBase client is installed in the **/opt/hadoopclient** directory.

3. Run the required environment variables as user **root**.

- ```
source /opt/hadoopclient/bigdata_env
```
4. Perform the kinit security authentication.  
**kinit Component service user**
  5. Enter the Gremlin operation block.  
**sh bin/gremlin.sh**
  6. Connect to the remote GraphBase.  
**:remote connect tinkerpop.server conf/remote-graphbase-gremlinserver.yaml**
  7. Omit the remote command ">".  
**:remote console** (This operation is optional. If you perform this operation, you do not need to add ">" before the Gremlin language.)
  8. Traverse a specified graph.  
**:remote config alias g Graph name**
  9. Query a vertex, as shown in the following example. (For other application scenarios, see [Gremlin Application Scenarios](#).)  
gremlin> :>  
=>[name:[zhangsan],age:[30]]  
gremlin> :remote console  
=>All scripts will now be sent to Gremlin Server - [szvphicprb00285/10.4.64.123:22375] - type  
'remote console' to return to local mode  
gremlin> g.V(13464736).valueMap()  
=>[name:[zhangsan],age:[30]]

```
[root@hd-123 gremlin-console]#source /opt/graphbaseClient/bigdata_env
[root@hd-123 gremlin-console]#kinit zky
Password for zky@HADOOP.COM:
[root@hd-123 gremlin-console]#sh bin/gremlin.sh
..../
(o o)
.....o0Oo-(3)-o0Oo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
gremlin> :remote connect tinkerpop.server conf/remote-graphbase-gremlinserver.yaml
=>Configured hd-123/187.4.64.123:22381, hd-184/187.4.64.184:22381, HD-10/187.4.65.10:22381
gremlin> :remote config alias g gremlintest_1109
=>g=gremlintest_1109
gremlin> :remote console
=>All scripts will now be sent to Gremlin Server - [hd-123/187.4.64.123:22381, hd-184/187.4.64.184:22381, HD-10/187.4.65.10:22381]
gremlin> g.V().valueMap().limit(10)
=>[name:[blame], age:[30]]
=>[telephone:[13612493615]]
=>[telephone:[137235842111]]
=>[telephone:[13612493615]]
=>[telephone:[15023614521]]
=>[telephone:[15291463258]]
=>[name:[blame], age:[30]]
=>[name:[blame], age:[30]]
=>[telephone:[13612493615]]
=>[telephone:[15023614521]]
gremlin> 
```

### 1.9.3.8 Gremlin Java

#### 1.9.3.8.1 Connecting Gremlin Servers and Clients

- Modify the Gremlin client configuration file **remote-objects.yaml**.

Change the host name in hosts to the actual IP address.

```
hosts:[10.7.66.111,10.7.66.151,10.7.66.220]
```

```
port: 22381
```

```
connectionPool: {
    maxContentLength: 65536000,
    maxInProcessPerConnection: 60,
    maxSize: 16,
    resultIterationBatchSize: 960
}
protocol: gremlinserver
jaasEntry: GremlinConsole
serializer:
```

```
{ className: org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0, config:  
{ ioRegistries: [org.janusgraph.graphdb.tinkerpop.JanusGraphIoRegistry] } }  
serializer:  
{ className: org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0, config:  
{ serializeResultToString: true } }
```

- Configure the mapping between the service IP Address of GraphServer and the host name in the local HOSTS file.

Directory of the HOSTS file on the Windows OS: **C:\Windows\|System32\drivers\etc\hosts**

Directory of the HOSTS file on the Linux OS: **/etc/hosts**

```
#Mapping between the IP address and host name.  
10.7.66.111 hd-111  
10.7.66.151 hd-151  
10.7.66.220 hd-220
```

- Connect the server and client.

Method 1:

```
//Connect the client and one or more Gremlin server instances.  
Cluster cluster = GremlinClient.getInstance().getCluster();  
//Create a client based on a cluster object.  
Client client = cluster.connect().alias ("Graph name");
```

Method 2:

```
//Connect the client and one or more Gremlin server instances.  
Cluster cluster = GremlinClient.getInstance().getCluster();  
//Configure a remote graph traversal object on the client.  
Graph graph = GremlinClient.getInstance().getGraph();  
GraphTraversalSource g = graph.traversal().withRemote(DriverRemoteConnection.using (cluster,  
"Graph name"));
```

### 1.9.3.8.2 Running Gremlin Java API Example Code

- Submit Gremlin DSL statements as a client object.

```
//Submit Gremlin DSL statements synchronously.  
ResultSet resultSet = Client.submit("g.V().has('propertyKey', 'huawei').valueMap().limit(10)");  
//Submit Gremlin DSL statements asynchronously.  
Future<ResultSet> resultSet =  
Client.submitAsync("g.V().hasLabel('person').valueMap('name','age').limit(10)");  
//Generate the result.  
==>{name=[vadas], age=[27]}  
==>{name=[josh], age=[32]}  
==>{name=[blame], age=[30]}  
==>{name=[marko], age=[29]}
```

- Perform a graph traversal.

```
//Perform vertex traversal.  
Iterator<Map<String,Object>> iterator = g.V(ids).valueMap(properties).limit(10)  
//Perform edge traversal.  
Iterator<Map<String,Object>> iterator = g.E(ids).valueMap(properties).limit(10)  
//Generate the result.  
==>{name=[vadas], age=[27]}  
==>{name=[josh], age=[32]}  
==>{name=[blame], age=[30]}  
==>{name=[marko], age=[29]}
```

### 1.9.3.9 Gremlin Application Scenarios

#### 1.9.3.9.1 Querying a Specified Graph Object

- Query a specified vertex.

```
//Taverse all properties of the vertex based on the vertex ID.  
gremlin> g.V(1151560,1195696).valueMap()
```

```
==>{name=[vadas], age=[27]}\n==>{name=[josh], age=[32]}
```

- **Query a specified edge.**  
gremlin> g.E('fryh-oojs-93th-108hc', '102fq-pmls-6aol-vegw').valueMap()  
==>{weight=0.8}\n==>{weight=0.4}

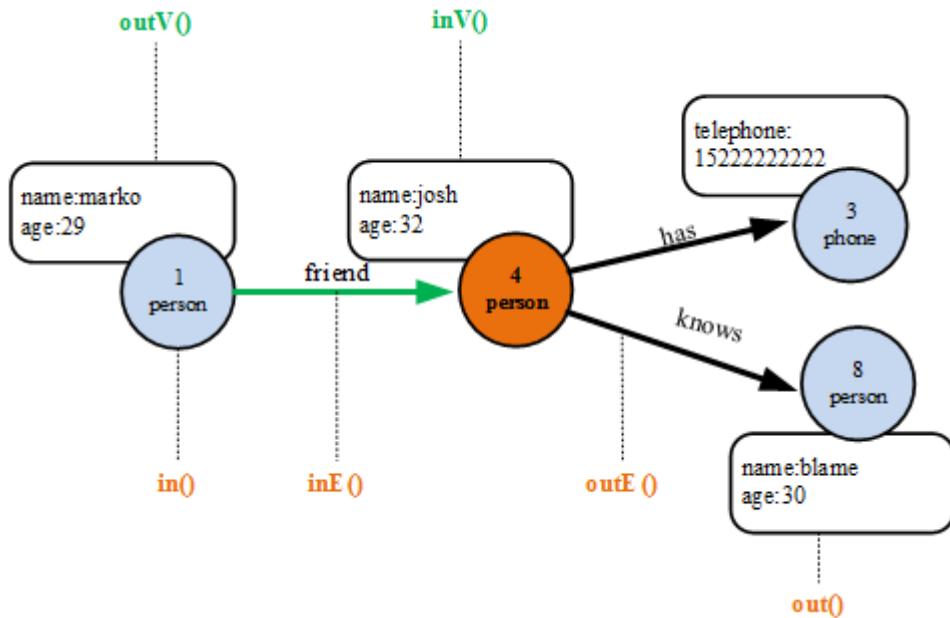
### 1.9.3.9.2 Queries One or More Properties

Querying one or more properties of a graph object.

```
gremlin> g.V().hasLabel('person').valueMap('name','age')\n==>{name=[vadas], age=[27]}\n==>{name=[josh], age=[32]}\n==>{name=[blame], age=[30]}\n==>{name=[marko], age=[29]}
```

### 1.9.3.9.3 Traversing a Graph Object

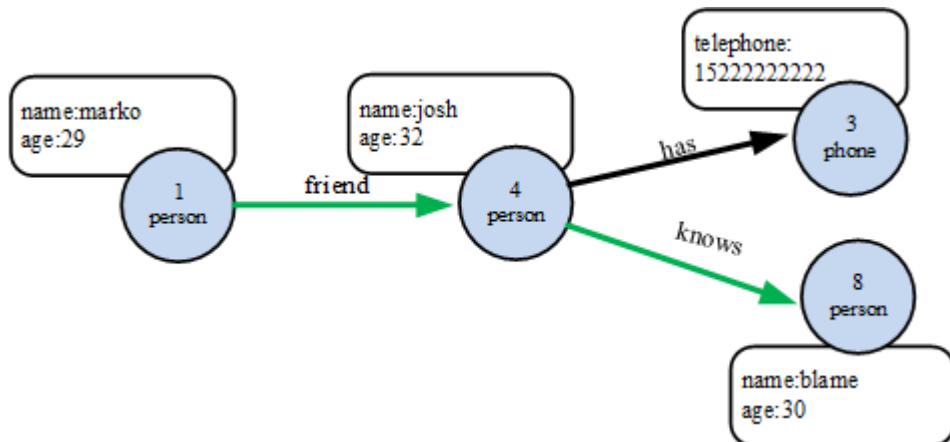
Graph Object Traversal	Description
Out(string )	Move to the outgoing adjacent vertices given the edge labels.
In(string )	Move to the incoming adjacent vertices given the edge labels.
Both(string )	Move to both the incoming and outgoing adjacent vertices given the edge labels.
OutE(string )	Move to the outgoing incident edges given the edge labels.
InE(string )	Move to the incoming incident edges given the edge labels.
BothE(string )	Move to both the incoming and outgoing incident edges given the edge labels.
outV()	Move to the outgoing vertex.
inV()	Move to the incoming vertex.
bothV()	Move to both vertices.



```

//Start from vertex 4 and traverse the edge to the out direction to obtain the incoming vertex of the edge.
gremlin> g.V(4).outE().inV()
==>v[3]
==>v[8]
//Move from vertex 4 to the out direction to obtain the peer vertex.
gremlin> g.V(4).out()
==>v[3]
==>v[8]
//Start from vertex 4 and traverse all edges to the out direction.
gremlin> g.V(4).outE()
==>e[10][4-has->3]
==>e[11][4-knows->8]
//Start from vertex 4 and traverse the edge to the out or in direction. The edge labels must be of a
specified edge type, such as knows, has, or friend.
gremlin> g.V(4).bothE('knows','has','friend')
==>e[10][4-has->3]
==>e[11][4-knows->8]
==>e[8][1-friend->4]
  
```

#### 1.9.3.9.4 Traversing a Path



Function: path()

```
//Start from vertex 1 and traverse two layers to the out direction to obtain the peer vertex of the second layer.  
gremlin> g.V(1).out().out().values('name')  
==>blame  
//Start from vertex 1 and traverse two layers to the out direction to return the track path of the traversal.  
gremlin> g.V(1).out().out().values('name').path()  
==>[v[1],v[4],v[8],blame]
```

### 1.9.3.9.5 Filtering by Condition

- Filtering function: has()

has	Description
has(key, value)	Remove the traverser if its element does not have the provided key/value property.
has(label, key, value)	Remove the traverser if its element does not have the specified label and provided key/value property.
has(key, predicate)	Remove the traverser if its element does not have a key value that satisfies the predicate.
HasLabel(labels )	Remove the traverser if its element does not have any of the labels.
HasId(ids )	Remove the traverser if its element does not have any of the IDs.
HasKey(keys )	Remove the traverser if the property does not have all of the provided keys.
HasValue(values )	Remove the traverser if its property does not have all of the provided values.
has(key)	Remove the traverser if its element does not have a value for the key.
hasNot(key)	Remove the traverser if its element has a value for the key.

- Expression: predicate

Predicate	Description
eq(object)	Is the incoming object equal to the provided object?
neq(object)	Is the incoming object not equal to the provided object?
lt(number)	Is the incoming number less than the provided number?
lte(number)	Is the incoming number less than or equal to the provided number?
gt(number)	Is the incoming number greater than the provided number?

Predicate	Description
gte(number)	Is the incoming number less than or equal to the provided number?
inside(number, number)	Is the incoming number greater than the first provided number and less than the second?
outside(number, number)	Is the incoming number less than the first provided number or greater than the second?
between(number, number)	Is the incoming number greater than or equal to the first provided number and less than the second?
Within(objects )	Is the incoming object in the array of provided objects?
Without(objects )	Is the incoming object not in the array of the provided objects?
TextP.startsWith(string)	Does the incoming String start with the provided String?
TextP.endsWith(string)	Does the incoming String end with the provided String?
TextP.contains(string)	Does the incoming String contain the provided String?
TextP.notStartingWith(string)	Does the incoming String not start with the provided String?
TextP.notEndingWith(string)	Does the incoming String not end with the provided String?
TextP.notContaining(string)	Does the incoming String not contain the provided String?

```
//Traverse the vertices that are of the person type.
gremlin> g.V().hasLabel('person')
==>v[1151560]
==>v[1195696]
==>v[1465088]
==>v[1421128]
//Traverse the vertices with the age property.
gremlin> g.V().properties().hasKey('age').value()
==>27
==>32
==>30
==>29
//Traverse the vertices where the values of the age properties are within the range from 20 to 30.
gremlin> g.V().has('age',inside(20,30)).values('age')
==>29
==>27
//Traverse the veretxes where the value of the name property is josh or marko.
gremlin> g.V().has('name',within('josh','marko')).valueMap()
==>{name=[josh], age=[32]}
==>{name=[marko], age=[29]}
```

### 1.9.3.9.6 Sorting the Result

```
//Sort the result based on the value of the age property in ascending order.
gremlin> g.V().hasLabel('person').order().by('age', incr).valueMap('name','age')
==>{name=[vadas], age=[27]}
==>{name=[marko], age=[29]}
==>{name=[blame], age=[30]}
==>{name=[josh], age=[32]}
//Sort the result based on the value of the age property in descending order.
gremlin> g.V().hasLabel('person').order().by('age', decr).valueMap('name','age')
==>{name=[josh], age=[32]}
==>{name=[blame], age=[30]}
==>{name=[marko], age=[29]}
==>{name=[vadas], age=[27]}
```

### 1.9.3.9.7 Displaying the Result in Multiple Pages

```
//Sort the result based on the value of the age property in ascending order and only obtain the first two data records.
gremlin> g.V().hasLabel('person').order().by('age', incr).limit(2).valueMap('name','age')
==>{name=[vadas], age=[27]}
==>{name=[marko], age=[29]}
//Sort the result based on the value of the age property in ascending order and obtain data from the third to the fourth records.
gremlin> g.V().hasLabel('person').order().by('age', incr).range(2,4).valueMap('name','age')
==>{name=[blame], age=[30]}
==>{name=[josh], age=[32]}
```

### 1.9.3.9.8 Calculating the Intersection Between Two Data Sets

```
//During the traversal, the concept of the intersection between two data sets must be differentiated from the join operation in the SQL syntax.
gremlin> g.V().hasLabel('person').match(
    __.as('c').values('name').as('name') //Data set 1. The column alias is name.
    __.as('c').out('friend','knows').values('name').as('friend_or_knows') //Data set 2. The column alias is friend_or_knows.
    ).select('name', 'friend_or_knows') //Calculate the intersection between the two data sets.
==>{name=josh, friend_or_knows=blame}
==>{name=marko, friend_or_knows=josh}
==>{name=marko, friend_or_knows=vadas}
```

### 1.9.3.9.9 Calculating the Union of Two Data Sets

```
//During the traversal, calculate the union of the following two data sets.
gremlin> g.V().has('name','josh').union(
    __.in('friend').values('name'), //Data set 1
    __.out('knows').values('name')) //Data set 2
==>marko
==>blame
```

### 1.9.3.9.10 Using Statistical Functions

Function	Description
count()	Collects data.
max()	Calculates the maximum value.
min()	Calculates the minimum value.
mean()	Calculates the average value.
sum()	Sums up data.

```
//Calculate the number of vertices whose type is person
gremlin> g.V().hasLabel('person').count()
==>4
//Traverse all vertices to obtain the maximum value of the age property.
gremlin> g.V().values('age').max()
==>32
```

## 1.9.4 Application Commissioning

### 1.9.4.1 Commissioning an Application in Windows

#### 1.9.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.



If IBM JDK is used in the Windows environment, the application cannot be run in the Windows environment.

##### Procedure

In the development environment (for example, running the REST API on the IntelliJ IDEA environment), right-click **GraphBaseRestExample.java** and choose **Run > GraphBaseRestExample.main()**.

#### 1.9.4.1.2 Viewing Windows Commissioning Results

##### Scenario

After the running of GraphBase application completes, you can view the running results in the IntelliJ IDEA running result.

##### Procedure

The following information is displayed in the running results:

```
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph HTTP/1.1
REQ BODY: {"graphName":"graphbase"}
RSP BODY: {
    "msg": "create graph success.",
    "code": "0"
}
[SUCCESS]: create graph[graphbase] successfully.
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
REQ BODY: {"name":"person"}
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "person"
    }
}
```

```
[SUCCESS]: create vertex label[person] successfully.  
REQ HEADER: POST https://10.154.4.90:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1  
REQ BODY: {"name":"phone"}  
RSP BODY: {  
    "msg": "Success.",  
    "code": "0",  
    "data": {  
        "name": "phone"  
    }  
}  
[SUCCESS]: create vertex label[phone] successfully.  
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label/person  
HTTP/1.1  
RSP BODY: {  
    "msg": "Success.",  
    "code": "0",  
    "data": {  
        "name": "person"  
    }  
}  
[SUCCESS]: query vertex label[person] successfully.  
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1  
RSP BODY: {  
    "msg": "Success.",  
    "code": "0",  
    "data": {"vertexLabelList": [  
        {  
            "name": "person"  
        },  
        {  
            "name": "phone"  
        }  
    ]}  
}  
[SUCCESS]: query all vertex label successfully.  
...
```

## 1.9.4.2 Commissioning an Application in Linux

### 1.9.4.2.1 Compiling and Running an Application

#### Scenario

GraphBase applications can run in a Linux OS. After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

#### Prerequisites

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by the JAR package exported by IntelliJIDEA.
- If the Linux host is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node. The host name and IP address must be in one-to-one mapping.

#### Procedure

##### Step 1 Export a JAR package.

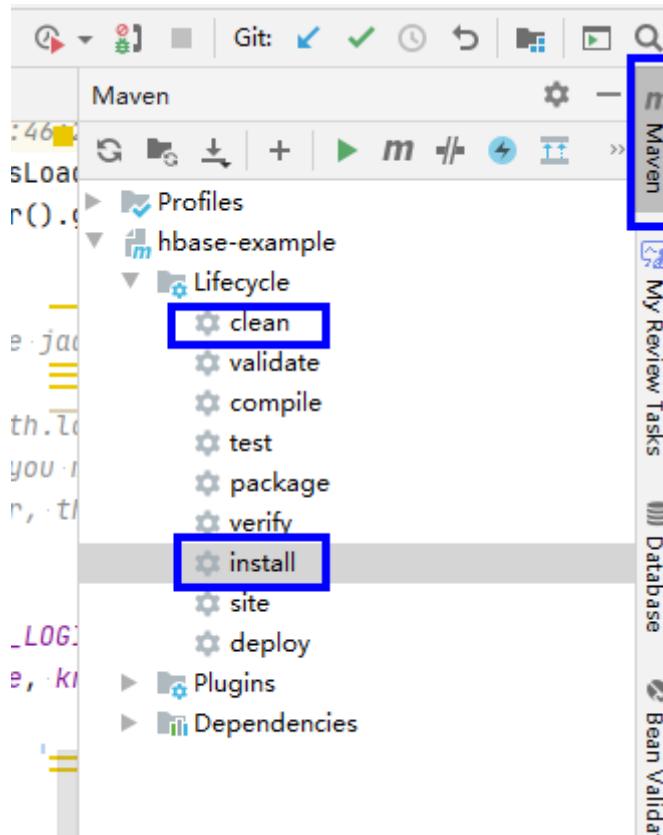
You can build a JAR package in either of the following ways:

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean**. Double-click **clean** to run the clean command of Maven.

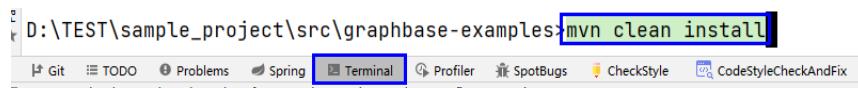
Choose **Maven** > *Sample project name* > **Lifecycle** > **install**. Double-click **install** to run the install command of Maven.

**Figure 1-110** clean and install



- Method 2: Access the directory where the **pom.xml** file is located in the **Terminal** window at the bottom of the IDEA. Run the **mvn clean install** command to compile the file.

**Figure 1-111** mvn clean install



After the compilation is complete, "Build Success" is printed, and the target directory is generated. The generated JAR package is stored in the target directory.

## Step 2 Export the JAR package on which the sample project depends.

Access the directory where the pom.xml file is located in the Terminal window or other command line tools at the bottom of the IDEA, and then run the following command:

```
mvn dependency:copy-dependencies -DoutputDirectory=lib
```

The lib folder is generated in the pom.xml directory, which contains the JAR package on which the sample project depends.

**Step 3** Prepare for the required JAR packages and configuration files.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories**conf** and **lib**. Upload the JAR packages in **Step 1** and **Step 2** to the **lib** directory in Linux. Upload the conf configuration files in the sample project to the **conf** directory in Linux.
2. Import the user authentication credential files.

Import the authentication credential files (**user.keytab** and **krb5.conf**) of the current user to the **conf** directory. On the FusionInsight Manager page, choose **System > Permission > User** to obtain the authentication credential files. Locate the row that contains the user whose authentication credentials need to be exported, and choose **More > Download Authentication Credential**.

3. In **/opt/test**, create the**GraphBase.sh** script, modify the following content, and save the file:

```
function Startup()
{
    SCRIPT_PATH=$(cd $(dirname $(readlink -f "${BASH_SOURCE[0]}")) && pwd )
    for f in ${SCRIPT_PATH}/lib/*.jar;
    do
        CLASSPATH=${CLASSPATH}:$f
    done
    JAVA_EXE=java
    JAVA_OPTS="$JAVA_OPTS -Xms2G -Xmx4G"
    ${JAVA_EXE} ${JAVA_OPTS} -classpath ${CLASSPATH}
    com.huawei.graphbase.example.GraphBaseRestExample
}
Startup
```

In the preceding content, "com.huawei.graphbase.example.GraphBaseRestExample" is used as an example.

**Step 4** Go to **/opt/test** and run the following commands to run the JAR packages:

**./graphbase.sh**

**----End**

#### 1.9.4.2.2 Viewing Linux Commissioning Results

##### Scenario

After an GraphBase application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Viewing GraphBase logs.

##### Procedure

The following information is displayed in the running results:

```
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph HTTP/1.1
REQ BODY: {"graphName":"graphbase"}
RSP BODY: {
```

```
"msg": "create graph success.",
"code": "0"
}
[SUCCESS]: create graph[graphbase] successfully.
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
REQ BODY: {"name":"person"}
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "person",
        "isPartitioned": "false"
    }
}
[SUCCESS]: create vertex label[person] successfully.
REQ HEADER: POST https://10.154.4.90:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
REQ BODY: {"name":"phone"}
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "phone",
        "isPartitioned": "false"
    }
}
[SUCCESS]: create vertex label[phone] successfully.
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label/person
HTTP/1.1
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "person",
        "isPartitioned": false
    }
}
[SUCCESS]: query vertex label[person] successfully.
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {"vertexLabelList": [
        {
            "name": "person",
            "isPartitioned": false
        },
        {
            "name": "phone",
            "isPartitioned": false
        }
    ]}
}
[SUCCESS]: query all vertex label successfully.
```

## 1.9.5 More Information

### 1.9.5.1 Common APIs

#### 1.9.5.1.1 REST API

For details about the complete and detailed description of GraphBase REST APIs, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

### 1.9.5.1.2 Gremlin API

For details about the GraphBase Gremlin API, see [Gremlin Console](#), [Gremlin Java](#) and [Gremlin Application Scenarios](#).

## 1.10 HBase Development Guide

### 1.10.1 Overview

#### 1.10.1.1 Application Development Overview

##### HBase Introduction

HBase is a column-oriented scalable distributed storage system featuring high reliability and high performance. HBase is designed to break through the limitation when relational databases are used to process massive data.

HBase applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require efficient random read of massive data.
- Scenarios that require good scalability.
- Structured and unstructured data is concurrently processed.
- The Atomicity, Consistency, Isolation, Durability (ACID) feature supported by traditional relational databases is not required.
- HBase tables provide the following features:
  - Large: One table contains a hundred million rows and one million columns.
  - Column-oriented: Storage and rights control is implemented based on columns (families), and columns (families) are independently retrieved.
  - Sparse: Null columns do not occupy storage space, so a table is sparse.

##### Interface Type Introduction

The Java language is recommended for HBase application development because HBase is developed based on Java and Java is a concise, universal, and easy-to-understand language.

HBase adopts the same interfaces as those of Apache HBase.

**Table 1-73** describes the functions that HBase can provide by invoking interfaces.

**Table 1-73** Functions provided by HBase interfaces

Function	Description
Data CRUD function	Data creating, retrieving, updating, and deleting
Advanced feature	Filter, secondary index, and coprocessor
Management function	Table management and cluster management

### 1.10.1.2 Common Concepts

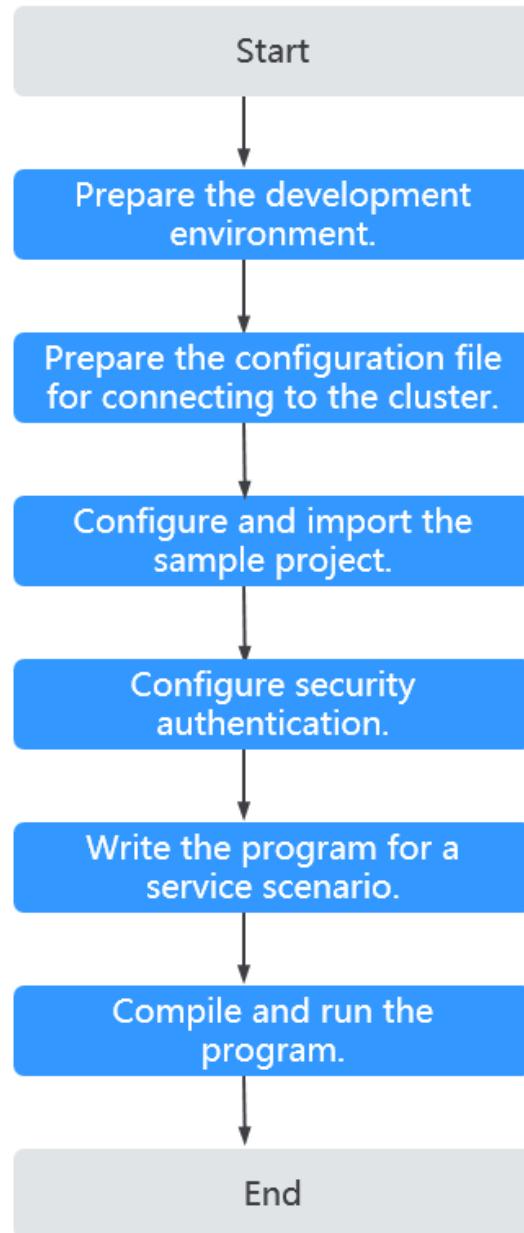
- **Filter**  
Filters provide powerful features to help users improve the table data processing efficiency of HBase. Users can use the filters predefined in HBase and customized filters.
- **Coprocessor**  
Coprocessors enable users to perform region-level operations and provide functions similar to those of triggers in relational database management systems (RDBMSs).
- **keytab file**  
The keytab file is a key file that stores user information. In security mode, applications use the key file for API authentication on HBase.
- **Client**  
Users can access the server from the client through the Java API, HBase Shell or WebUI to read and write HBase tables. The HBase client in this document indicates the HBase client installation package, see [External Interfaces](#).

### 1.10.1.3 Development Process

This document describes HBase application development based on the Java API.

For information about each phase in the development process, see [Figure 1-112](#) and [Table 1-74](#).

Figure 1-112 HBase application development process



**Table 1-74** HBase application development process description

Phase	Description	Reference Document
Prepare the development environment.	Before developing an application, you need to prepare the development environment. You are advised to use the Java language and IntelliJ IDEA tool for development. In addition, you need to complete the initial configuration of the JDK and Maven.	<a href="#">Preparing for Development Environment</a>
Prepare the Configuration File for Connecting to the Cluster.	During application development or running, you need to connect to the MRS cluster through cluster configuration files. Configuration files include cluster component information files and user files used for security authentication. You can obtain related content from the created MRS cluster. Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured.	<a href="#">Preparing the Connection Cluster Configuration File</a>
Configure and import the sample project.	HBase provides multiple sample programs in different scenarios. You can obtain sample projects and import them to the local development environment for program learning.	<a href="#">Configuring and Importing Sample Projects</a>
Configure security authentication.	If you are using an MRS cluster with Kerberos authentication enabled, you need to perform security authentication.	<a href="#">Preparing for Security Authentication</a>

Phase	Description	Reference Document
Develop programs based on business scenarios.	Develop programs based on actual service scenarios and invoke component interfaces to implement corresponding functions.	<a href="#">Developing an Application</a>
Compile and run the application.	You can commission and run the program in the local Windows development environment, or compile the program into a JAR package and submit it to the Linux node for running.	<a href="#">Application Commissioning</a>

#### 1.10.1.4 HBase Sample Project Introduction

To obtain the MRS sample project, visit <https://github.com/huaweicloud/huaweicloud-mrs-example>. Switch to the version branch that matches the MRS cluster, download the package to the local PC, and decompress the package to obtain the sample code project of each component.

Currently, MRS provides the following HBase-related sample projects. You can select an example based on the actual service scenario:

**Table 1-75** HBase-related sample projects

Sample Project Location	Function Description
hbase-examples/hbase-example	<p>Application development example of HBase data read/write operations and global secondary indexes. The following functions can be implemented by calling HBase APIs:</p> <ul style="list-style-type: none"><li>• You can call HBase APIs to create user tables, import user data, add user information, query user information, and create secondary indexes for user tables. For details, see <a href="#">Service Scenario Description</a>.</li><li>• Creates and deletes global secondary indexes, modifies the status of global secondary indexes, and queries data based on global secondary indexes. For details about related service scenarios, see <a href="#">Service Scenario Description</a>.</li><li>• When multiple threads concurrently read data from or write data into an HBase table, region information can be preloaded. For details about related service scenarios, see <a href="#">Sample Program for Preloading Meta Tables When Request Concurrency Is High</a>.</li></ul>
hbase-examples/hbase-rest-example	<p>HBase REST API development sample. Using REST APIs to query clusters, obtain tables, operate Namespaces, and operate tables. For details, see <a href="#">HBase Rest API Invoking Sample Program</a>.</p>
hbase-examples/hbase-thrift-example	<p>Access the HBase ThriftServer development sample. Accessing ThriftServer to operate tables, writing data to tables, and reading data from tables. For details, see <a href="#">Accessing the HBase ThriftServer Sample Program</a>.</p>

Sample Project Location	Function Description
hbase-examples/hbase-zk-example	Application development example for HBase to access ZooKeeper. The same client process accesses both the FusionInsight ZooKeeper and third-party ZooKeeper. The HBase client accesses the FusionInsight ZooKeeper, and the customer application accesses the third-party ZooKeeper. For details, see <a href="#">Sample Program for HBase to Access Multiple ZooKeepers</a> .
springboot/hbase-examples	Provide an example of connecting Phoenix to SpringBoot. This example describes how to interconnect HBase with Phoenix with Spring Boot. For details, see <a href="#">Security Authentication for Interconnecting HBase/Phoenix with Spring Boot</a> .

## 1.10.2 Environment Preparation

### 1.10.2.1 Preparing for Development Environment

[Table 1-76](#) describes the environment required for secondary development.

**Table 1-76** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>

Item	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
JUnit plug-in installation	Basic configuration for the development environment.

Item	Description
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.  Huawei provides Huawei Mirrors for you to download all dependency JAR files of sample projects. However, you need to download the rest dependency open-source JAR files from the Maven central repository or other custom repository address. For details, see <a href="#">Configuring Huawei Open-Source Mirrors</a> .
7-zip	Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

### 1.10.2.2 Preparing the Connection Cluster Configuration File

#### Preparing a Developer Account

For an MRS cluster with Kerberos authentication enabled, you need to prepare a user who has the permission to perform operations on related components for program authentication.

The following HBase permission configuration example is for reference only. You can adjust the permission configuration based on service requirements in actual service scenarios.

**Step 1** Log in to FusionInsight Manager.

**Step 2** Choose **Cluster > Services > HBase > More > Enable Ranger**, and check whether this parameter is dimmed.

- If yes, create a user and grant related operation rights to the user in Ranger.
  - a. Choose **System > Permission > User > Create**. On the Create user page, create a Machine-Machine user, for example, **developuser**. The user group must be added to the **hadoop** user group.
  - b. Log in to the Ranger management page as the Ranger administrator **rangeradmin**.
  - c. On the home page, click a component plug-in name in the **HBASE** area, for example, HBase.
  - d. Click  in the Operation column of **all - table, column-family, column** for Policy Name.
  - e. In the **Allow Conditions** area, add a policy condition, select the new user name in the **Select User** column, and select **Select All** in the **Permissions** column.
  - f. Click **Save**.
- If no, create a user and grant related operation permissions to the user on FusionInsight Manager.
  - a. Choose **System > Permission > Role > Create Role**.

- i. Enter the role name, for example, **developrole**.
- ii. In **Configure Resource Permission**, choose *Name of the desired cluster* > **HBase** > **HBase Scope** > **global**, select the **admin**, **create**, **read**, **write**, and **execute** permissions, and click **OK**.
- iii. Choose **User** > **Create**. On the Create user page, create a Machine-Machine user, for example, **developuser**.
  - o The user group must be added to the **hadoop** user group.
  - o Add the new role to **Role**.

**Step 3** Log in to FusionInsight Manager as the **admin** user and choose **System** > **Permission** > **User**. In the Operation column of the user named **developuser**, choose **More** > **Download Authentication Credential** to download the authentication credential file. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files for security authentication in the sample project.

----End

## Preparing an Operating Environment

During application development or running, you need to connect to the MRS cluster through cluster configuration files. Configuration files include cluster component information files and user files for security authentication. You can obtain related content from the created MRS cluster.

Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured.

- Scenario 1: Prepare the configuration files required by the commissioning program in the local Windows development environment.
  - a. Log in to the FusionInsight Manager, click **Download Client** in the upper right corner of the Homepage, set **Select Client Type** to **Configuration Files Only** and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client configuration file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file.
  - b. Go to the client configuration file decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\HBase\config**, obtain related configuration files from the [Table 1-77](#).

**Table 1-77** Configuration files

file	Function
core-site.xml	Configures Hadoop Core parameters.
hbase-site.xml	Configures HBase parameters.

file	Function
hdfs-site.xml	Configures HDFS parameters.

 NOTE

To obtain the configuration file required by the HBase ThriftServer sample project, see [Preparing the ThriftServer Instance Configuration File](#).

- c. Copy the content in the **hosts** file in the decompression directory to the local **hosts** file.

 NOTE

- During application development, if you need to commission applications in the local Windows system, ensure that the local node can communicate with the hosts listed in the hosts file.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- When configuring IPv6 hosts on the local PC running Windows, add % and InterfaceIndex of the network interface card (NIC) to the end of the IPv6 address. The InterfaceIndex can be obtained by running the **ipconfig** command.

Example:

fec1:0:0:e505:8:99:5:1%10

- Scenario 2: Prepare the configuration file required for running programs in the Linux environment.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.
  - b. To obtain the configuration file.
    - i. Log in to the FusionInsight Manager, click **Download Client** in the upper right corner of the Homepage, set **Select Client Type** to **Configuration Files Only** and click **OK**. Download the client configuration file to the active OMS node in the cluster.
    - ii. Log in to the active OMS node as user **root**, go to the directory where the client configuration file is stored (the default directory is **/tmp/FusionInsight-Client/**), and decompress the software package to obtain the configuration files in the [Table 1-77](#) in the *FusionInsight\_Cluster\_1\_Services\_ClientConfig/HBase/config* directory.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, the download path is **/tmp/FusionInsight-Client** on the active management node.

**cd /tmp/FusionInsight-Client**

**tar -xvf FusionInsight\_Cluster\_1\_Services\_Client.tar**

**tar -xvf**

**FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**

**cd FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles**

 NOTE

HBase ThriftServer sample project, see [Preparing the ThriftServer Instance Configuration File](#).

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## Preparing the ThriftServer Instance Configuration File

To access HBase ThriftServer and perform table-related operations, perform the following steps:

- Step 1** Log in to FusionInsight Manager, choose **Cluster > Service > HBase > Configuration** and click **All Configurations**, search for and modify the parameter **hbase.thrift.security.qop** of the ThriftServer instance. The value of this parameter must be the same as that of **hbase.rpc.protection**. Save the configuration and restart the node service for the configuration to take effect.

 NOTE

The mapping between **hbase.rpc.protection** and **hbase.thrift.security.qop** is as follows:

- "privacy" - "auth-conf"
- "authentication" - "auth"
- "integrity" - "auth-int"

- Step 2** Obtain the configuration file of the ThriftServer instance in the cluster.

- Method 1: Choose **Cluster > Service > HBase > Instance**, click the ThriftServer instance to go to the details page. In the **Configuration File** area of the **Dashboard** page, click the names of the configuration files **hdfs-site.xml**, **core-site.xml**, and **hbase-site.xml** to obtain the configuration files.
- Method 2: Obtain the configuration files by decompressing the client file in [Preparing an Operating Environment](#). Manually add the following configuration to **hbase-site.xml**. The value of **hbase.thrift.security.qop** must be the same as that in [Step 1](#).

```
<property>
<name>hbase.thrift.security.qop</name>
<value>auth</value>
</property>
<property>
<name>hbase.thrift.kerberos.principal</name>
<value>thrift/hadoop.hadoop.com@HADOOP.COM</value>
</property>
<property>
<name>hbase.thrift.keytab.file</name><value>/opt/huawei/Bigdata/FusionInsight_HD_8.3.1/install/FusionInsight-HBase-2.4.14/keytabs/HBase/thrift.keytab</value>
</property>
```

----End

### 1.10.2.3 Configuring and Importing Sample Projects

#### Background

Obtain the HBase development example project. Import the project to IntelliJ IDEA for learning.

#### Prerequisites

- Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the MRS cluster time in the bottom-right corner on FusionInsight Manager.
- The development environment and MRS cluster configuration files have been prepared. For details, see [Preparing for Development Environment](#).

#### Procedure

- Step 1** Obtain the sample project from the `src` directory in the directory where the sample code is decompressed by referring to [Obtaining Sample Projects from Huawei Mirrors](#). You can select an example based on the actual service scenario. For details about the example, see [HBase Sample Project Introduction](#).
- Step 2** If you need to commission the HBase sample code on the local Windows operating system, you need to place the configuration files and authentication files required by each sample project by referring to the following table:

**Table 1-78** Place the configuration files/authentication files required for each sample project

Sample Project Location	Configuration/authentication files to be placed
hbase-examples/hbase-example (single cluster)	Place the following files in the <code>../src/main/resources/conf</code> directory of the sample project: <ul style="list-style-type: none"><li>• The configuration files are <code>core-site.xml</code>, <code>hbase-site.xml</code>, and <code>hdfs-site.xml</code> obtained in <a href="#">Preparing an Operating Environment</a>.</li><li>• The authentication files are the keytab authentication files <code>user.keytab</code> and <code>krb5.conf</code> obtained by <a href="#">Preparing a Developer Account</a>.</li></ul>

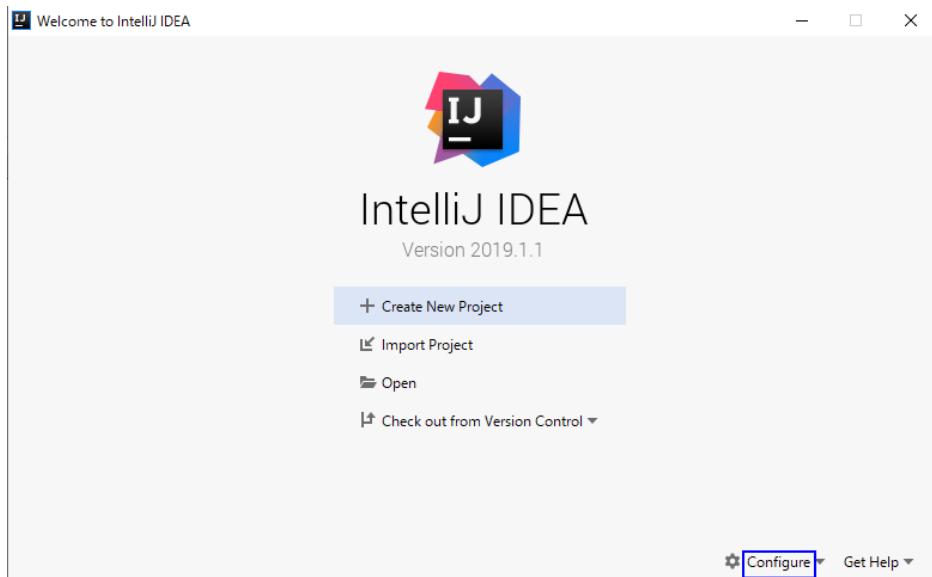
Sample Project Location	Configuration/authentication files to be placed
hbase-examples/hbase-example (multi-cluster)	<p>Place the authentication credential and configuration file of a cluster with the same user name in the mutual trust scenario to the <b>..src/main/resources/hadoopDomain</b> directory, and place the configuration file of the other cluster to the <b>..src/main/resources/hadoop1Domain</b> directory.</p> <ul style="list-style-type: none"><li>The configuration files are <b>core-site.xml</b>, <b>hbase-site.xml</b>, and <b>hdfs-site.xml</b> obtained in section <a href="#">Preparing an Operating Environment</a>.</li><li>The authentication files are the keytab authentication files <b>user.keytab</b> and <b>krb5.conf</b> obtained by <a href="#">Preparing a Developer Account</a>.</li></ul>
hbase-examples/hbase-rest-example	<p>Place the keytab authentication files <b>user.keytab</b> and <b>krb5.conf</b> obtained in <a href="#">Preparing a Developer Account</a> to the <b>..src/main/resources/conf</b> directory. If the conf directory does not exist, create it.</p>
hbase-examples/hbase-thrift-example	<p>Place the following files in the <b>..src/main/resources/conf</b> directory of the sample project:</p> <ul style="list-style-type: none"><li>The configuration files are <b>core-site.xml</b>, <b>hbase-site.xml</b>, and <b>hdfs-site.xml</b> obtained in <a href="#">Preparing the ThriftServer Instance Configuration File</a>.</li><li>The authentication files are the keytab authentication files <b>user.keytab</b> and <b>krb5.conf</b> obtained by <a href="#">Preparing a Developer Account</a>.</li></ul>

Sample Project Location	Configuration/authentication files to be placed
hbase-examples/hbase-zk-example	<p>Place the following files in the <code>./src/main/resources/conf</code> directory of the sample project:</p> <ul style="list-style-type: none"> <li>The configuration files are <code>core-site.xml</code>, <code>hbase-site.xml</code>, and <code>hdfs-site.xml</code> obtained in <a href="#">Preparing an Operating Environment</a>.</li> <li>The authentication files are the keytab authentication files <code>user.keytab</code> and <code>krb5.conf</code> obtained by <a href="#">Preparing a Developer Account</a>.</li> <li>Ensure that the <code>zoo.cfg</code> and <code>jaas.conf</code> files required for authentication in the <a href="#">Security Authentication for Accessing Multiple ZooKeepers</a> exist in the directory.</li> </ul>
springboot/hbase-examples	<p>The following files need to be placed on your local Windows:</p> <ul style="list-style-type: none"> <li>The configuration files are <code>core-site.xml</code>, <code>hbase-site.xml</code>, and <code>hdfs-site.xml</code> obtained in <a href="#">Preparing an Operating Environment</a>.</li> <li>The authentication files are the keytab authentication files <code>user.keytab</code> and <code>krb5.conf</code> obtained by <a href="#">Preparing a Developer Account</a>.</li> </ul>

**Step 3** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.

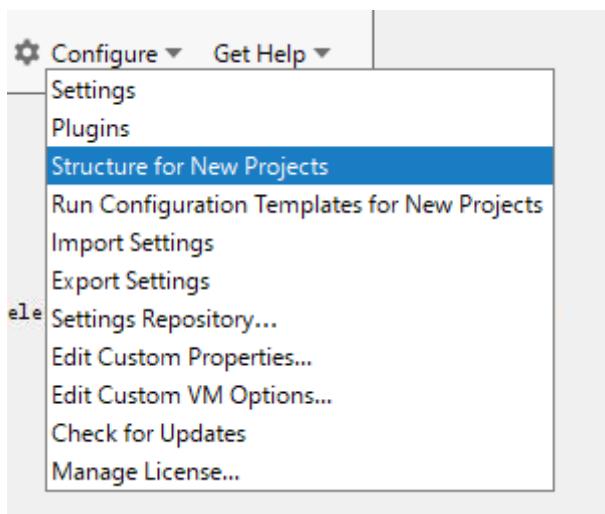
1. Start the IntelliJ IDEA and click **Configure**.

Figure 1-113 Quick Start



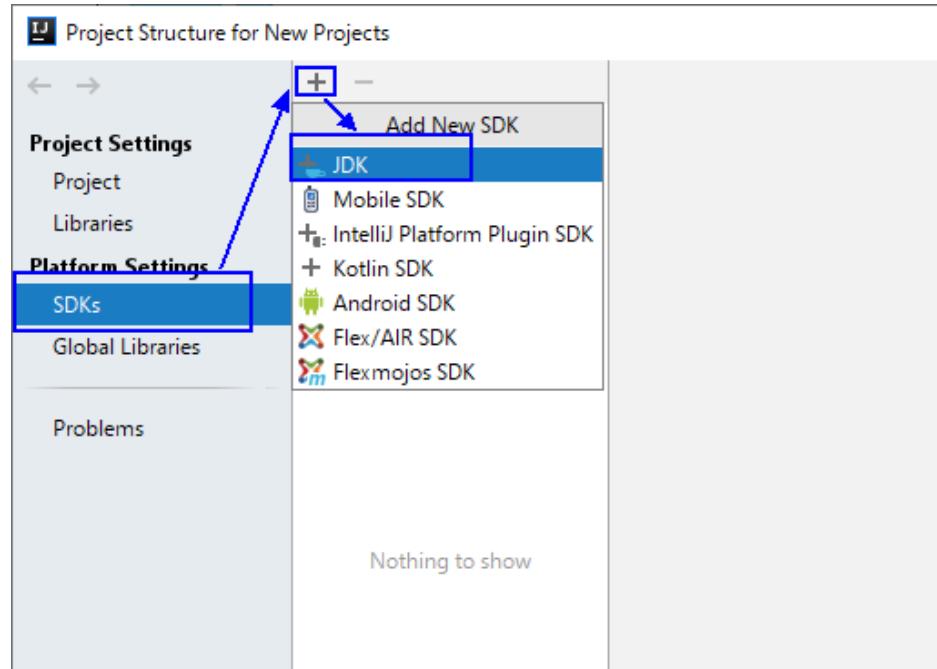
2. Click **Structure for New Projects** from the drop-down list.

Figure 1-114 Configure



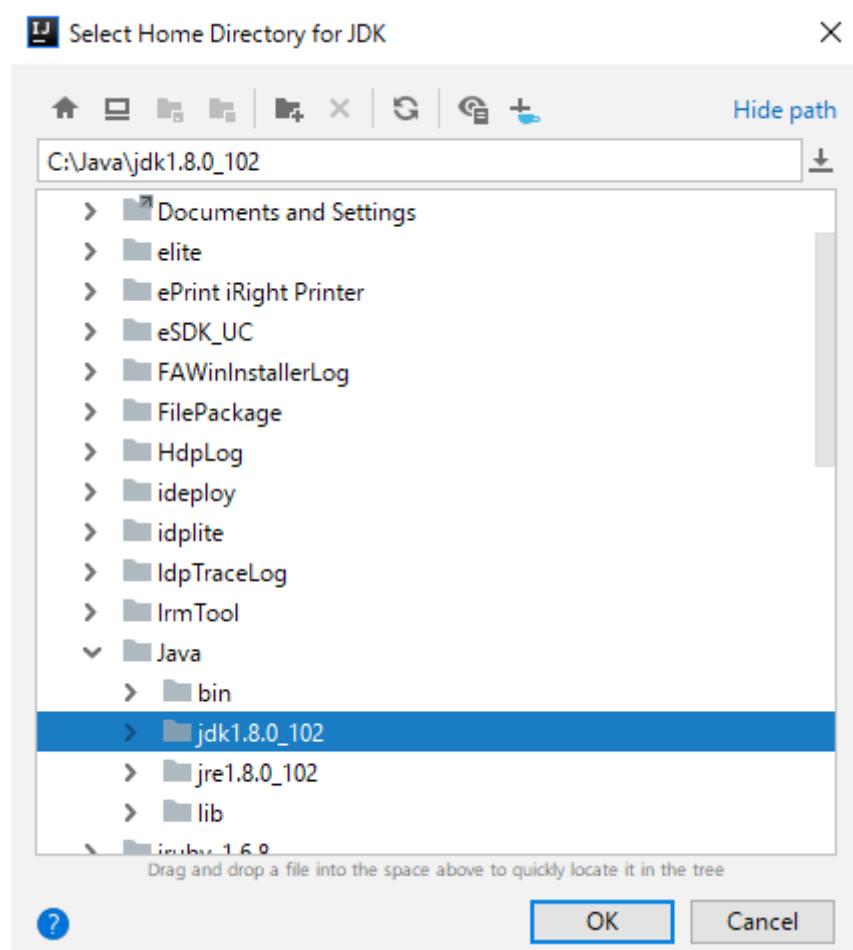
3. On the displayed **Project Structure for New Projects** page, select **SDKs** and click the plus sign (+) to add the **JDK**.

Figure 1-115 Project Structure for New Projects



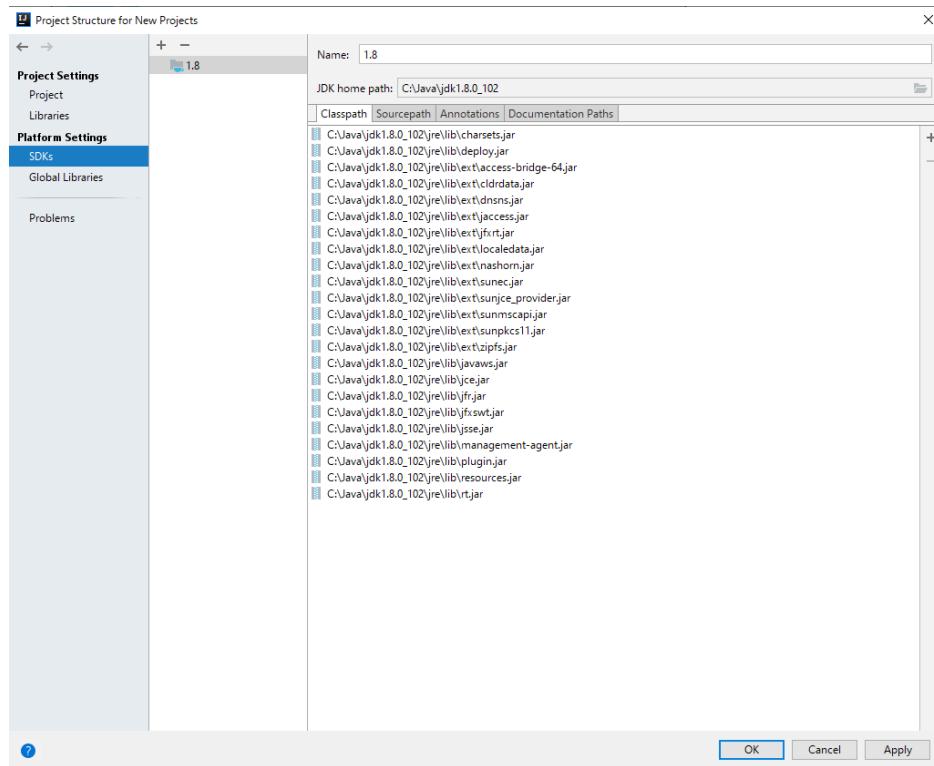
4. In the **Select Home Directory for JDK** window that is displayed, select the JDK directory, and click **OK**.

Figure 1-116 Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.

Figure 1-117 Complete JDK configuration



NOTE

The operations vary by the IDEA version. The operation procedure varies according to the actual version.

**Step 4** Import the example project to the IntelliJ IDEA development environment.

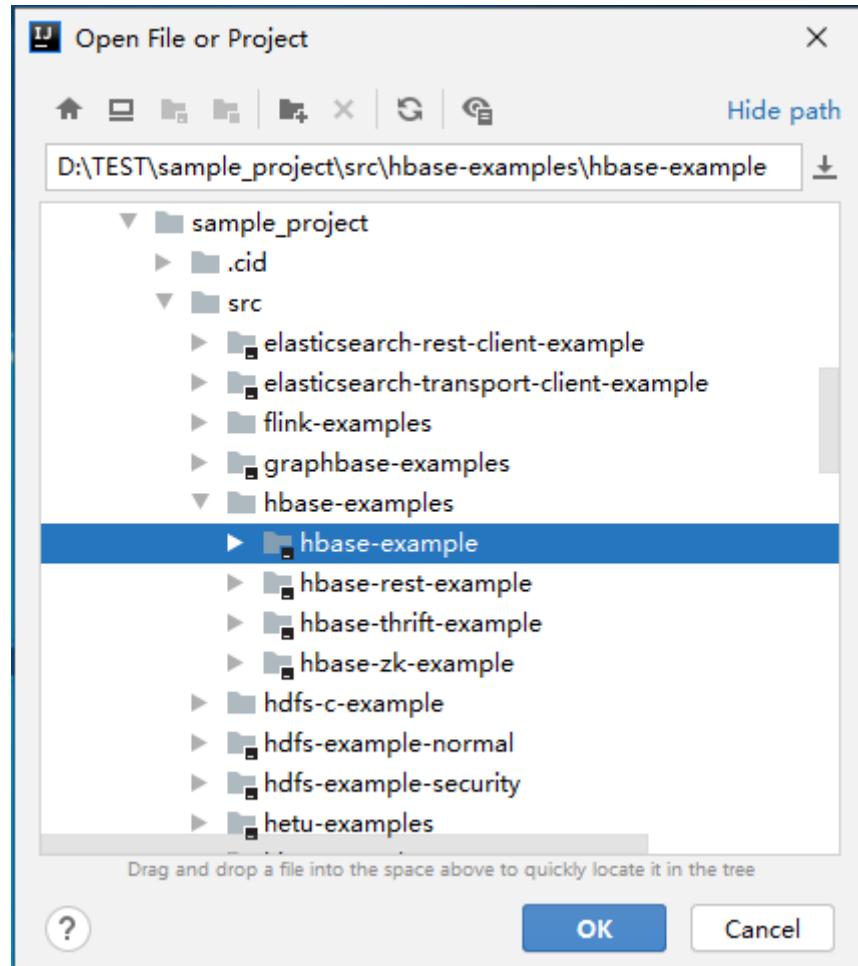
1. Start the IntelliJ IDEA, and click **Import Project** on the Open or Import.
- For the IDEA tool that has been used, you can choose **File > Import project...** on the main interface to import the sample project.

Figure 1-118 Open or Import (Quick Start page)



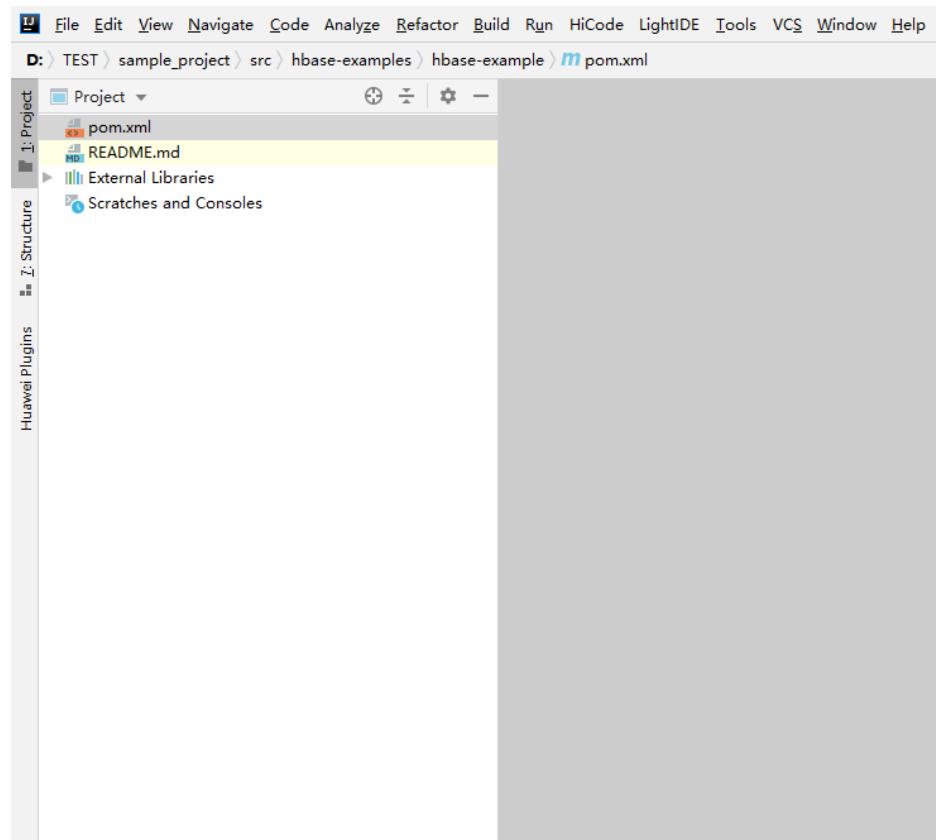
2. Select the example project folder **hbase-example**, and click **OK**.

**Figure 1-119** Select File or Directory to Import



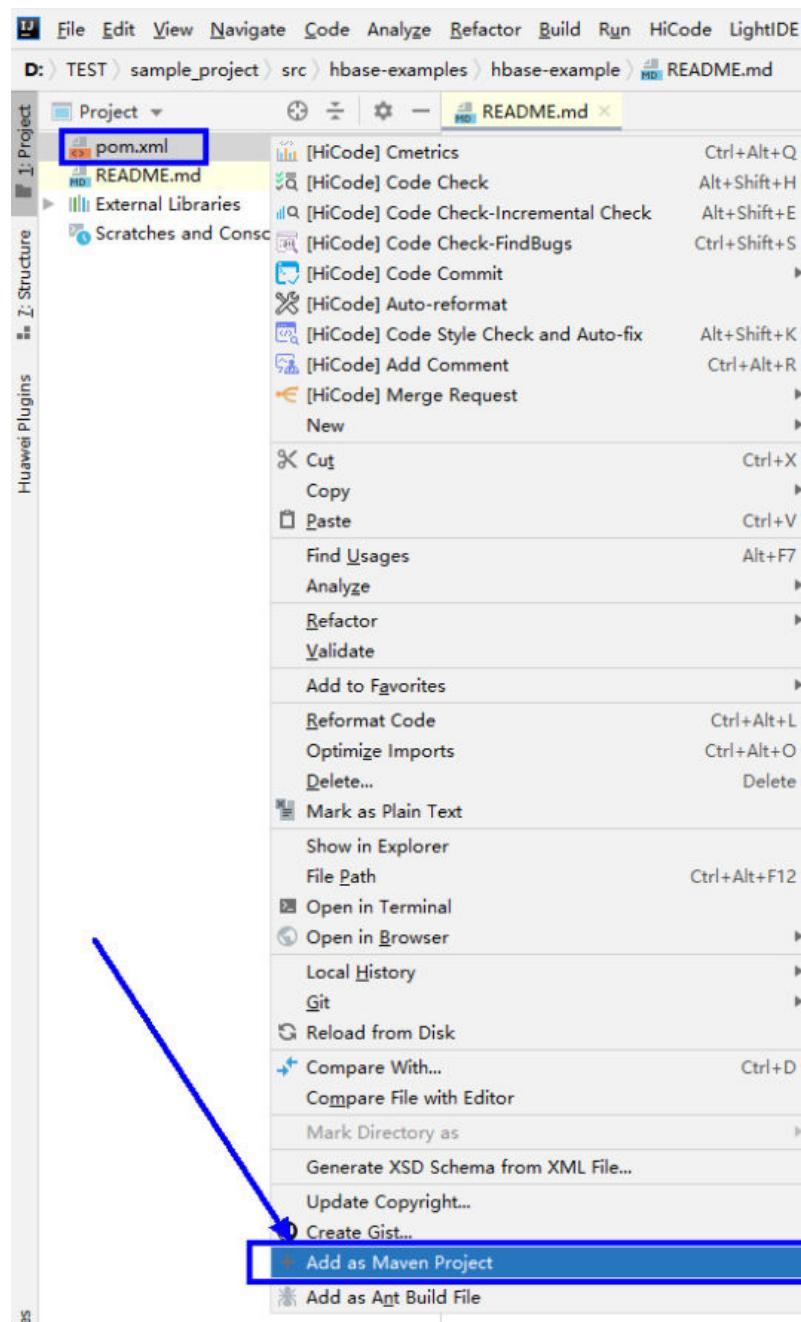
3. After the import, the imported sample project is displayed on the IDEA home page.

Figure 1-120 Successfully importing the sample project



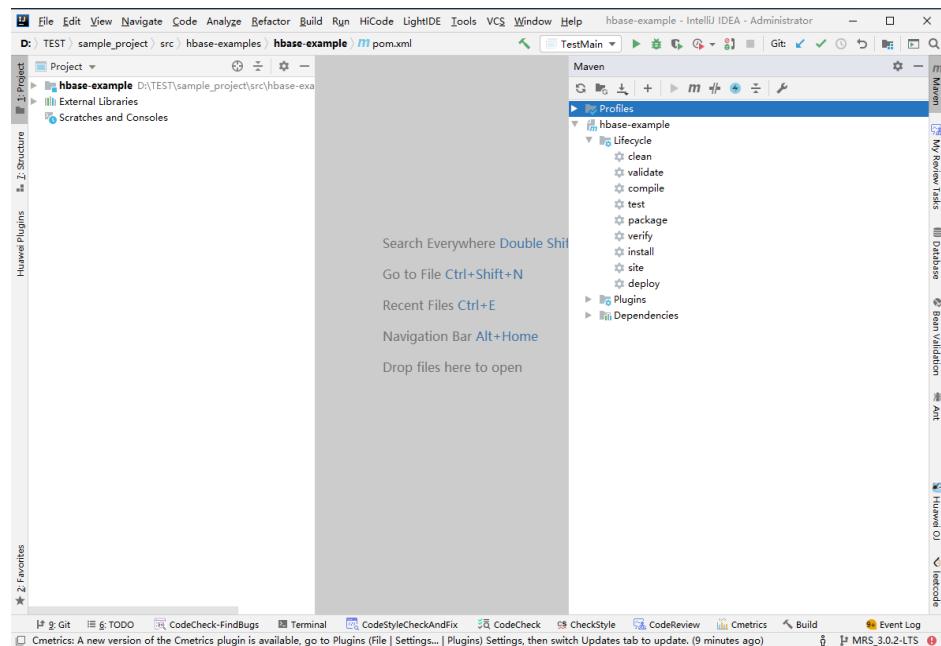
4. Right-click **pom.xml** and choose **Add as Maven Project** from the shortcut menu to add the project as a Maven Project. If the **pom.xml** icon is as shown in , go to the next step.

Figure 1-121 Add as Maven Project



In this case, the IDEA can identify the project as a Maven project.

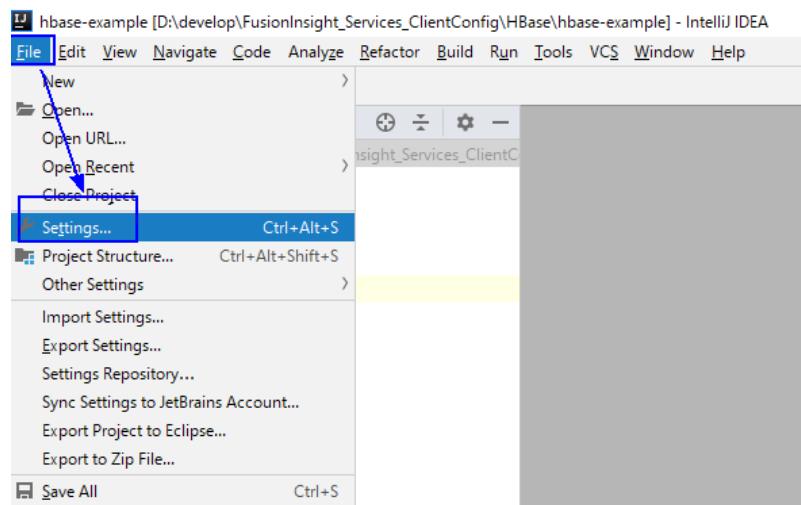
**Figure 1-122** Sample project displayed in IDEA as a Maven project



### Step 5 Set the Maven version used by the project.

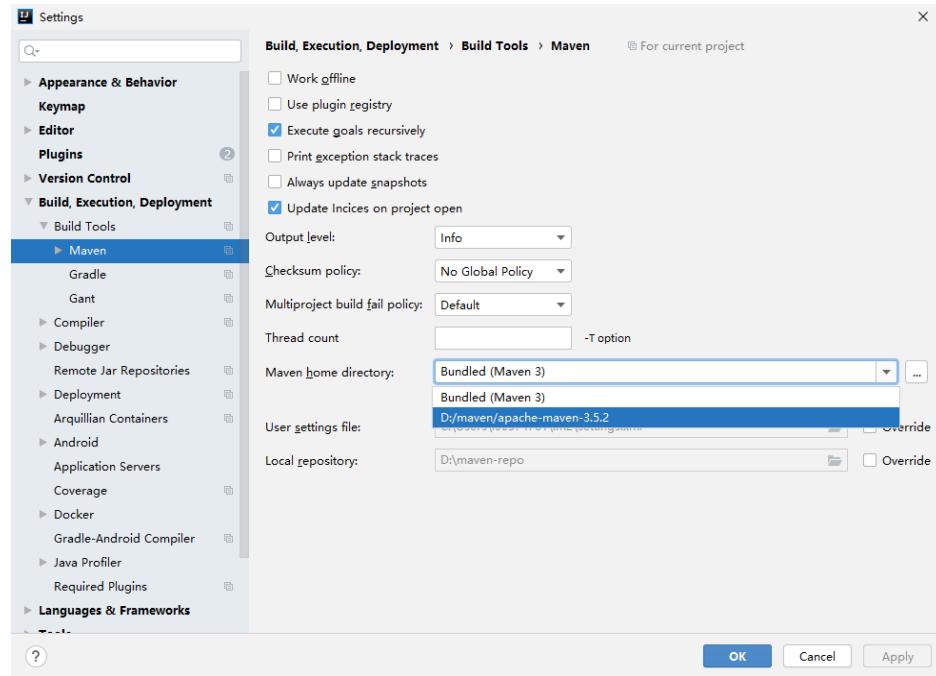
- Choose **File > Settings...** from the main menu of IntelliJ IDEA.

**Figure 1-123** Settings



- Choose **Build,Execution,Deployment > Maven** and set **Maven home directory** to the Maven version installed on the local host.  
Set **User settings file** and **Local repository** parameters based on the site requirements, and choose **Apply > OK**.

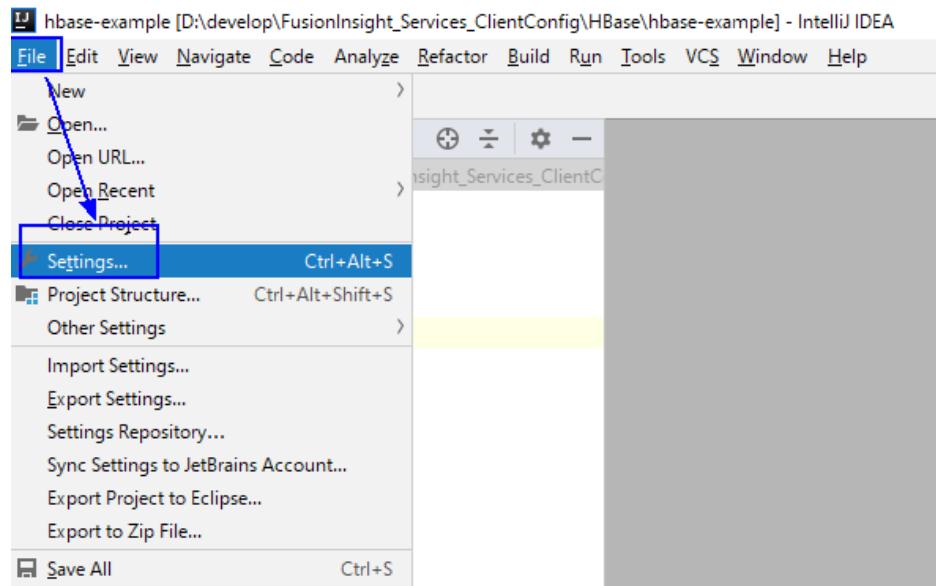
Figure 1-124 Selecting the local Maven installation directory



**Step 6** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

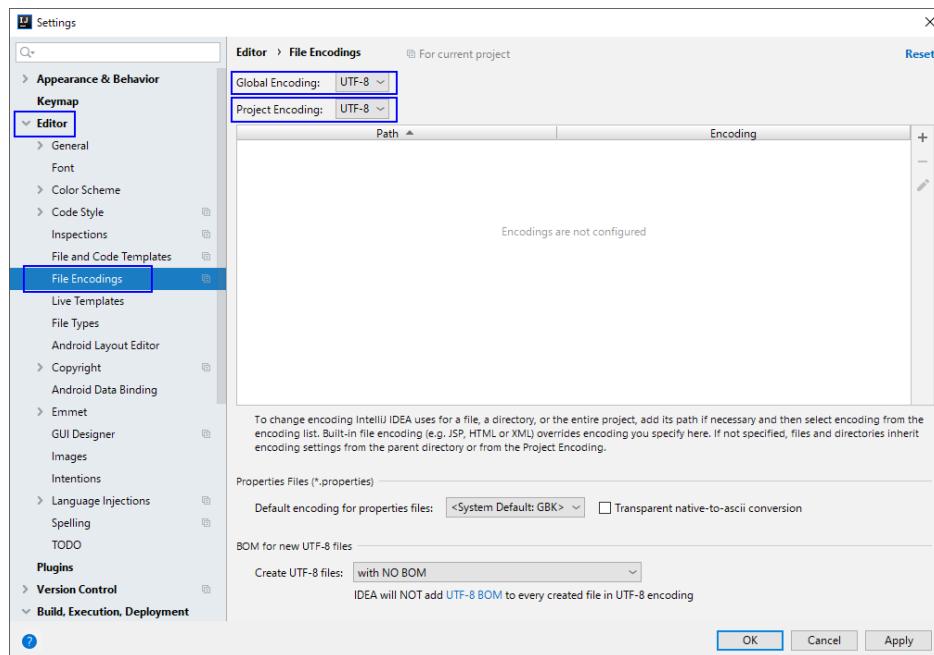
1. On the IntelliJ IDEA menu bar, choose **File > Settings....**

Figure 1-125 Settings



2. In the navigation tree of the **Settings** page, choose **Editor > File Encodings**, and select **UTF-8** for **Global Encoding** and **Project Encoding**.

Figure 1-126 File Encodings



3. Click **Apply** and **OK** to complete the encoding configuration.

----End

#### 1.10.2.4 Preparing for Security Authentication

##### 1.10.2.4.1 Security Authentication for HBase Data Read and Write and Global Secondary Index Example (Single-Cluster Scenario)

###### Scenario

In a security cluster environment, the components must be mutually authenticated before communicating with each other to ensure communication security. ZooKeeper and Kerberos security authentications are required for HBase application development. The **jaas.conf** file is used for ZooKeeper authentication, and the **keytab** and **krb5.conf** files are used for Kerberos security authentication.

The code authentication mode is used for security authentication. Oracle Java and IBM Java are supported.

 NOTE

- Before using an IBM JDK on a client, check whether the MRS cluster runs one of the following versions. If not, upgrade the cluster version first.
  - MRS 8.2.0 or later
  - MRS 8.1.2.5 or later 8.1.x
  - MRS 8.0.2.6 or later 8.0.2.x
- SASL of IBM JDKs is incompatible with JDKs such as OpenJDK. As a result, the authentication fails. In this case, you need to check the client parameters and then perform the authentication. To perform the authentication, perform the following steps:
  1. Open the HBase configuration file **hbase-site.xml** to be loaded on the client or to an application.
  2. Check the **hbase.rpc.client.impl** value.
  3. If the value is **org.apache.hadoop.hbase.ipc.NettyRpcClient**, go to [Changing the Configuration Value](#) to modify the configuration. Otherwise, no further action is required.
  4. Change the **hbase.rpc.client.impl** value to **org.apache.hadoop.hbase.ipc.BlockingRpcClient** or **org.apache.hadoop.hbase.ipc.RpcClientImpl**.

The following code snippet belongs to the **TestMain** class of the **com.huawei.bigdata.hbase.examples** packet.

- Code authentication

```
try {  
    init();  
    login();  
}  
catch (IOException e) {  
    LOG.error("Failed to login because ", e);  
    return;  
}
```

- Initializing configuration

```
private static void init() throws IOException {  
    // Default load from conf directory  
    conf = HBaseConfiguration.create();  
    //In Windows environment  
    String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;  
    [1] //In Linux environment  
    //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;  
    conf.addResource(new Path(userdir + "core-site.xml"), false);  
    conf.addResource(new Path(userdir + "hdfs-site.xml"), false);  
    conf.addResource(new Path(userdir + "hbase-site.xml"), false);  
}
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation.

## Prerequisites

You have obtained the configuration file and authentication file required for running the sample project. For details, see [Preparing the Connection Cluster Configuration File](#).

## Configuring Secure Login

Modify the following parameters in the **TestMain** class of the **com.huawei.bigdata.hbase.examples** package based on the site requirements:

- Change the value of **userName** to the username you actually use, for example, **developuser**.
- Change the value of **ZOOKEEPER\_DEFAULT\_SERVER\_PRINCIPAL** to **zookeeper/hadoop**.*Cluster domain name*. To obtain the value of cluster domain name, log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and view the value of **Local Domain**.

```
private static void login() throws IOException {
    if (User.isHBaseSecurityEnabled(conf)) {
        userName = "developuser";

        //In Windows environment
        String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
        //In Linux environment
        //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

        /*
         * if need to connect zk, please provide jaas info about zk. of course,
         * you can do it as below:
         * System.setProperty("java.security.auth.login.config", confDirPath +
         * "jaas.conf"); but the demo can help you more : Note: if this process
         * will connect more than one zk cluster, the demo may be not proper. you
         * can contact us for more help
         */
        LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userName, userKeytabFile);
        LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
        LoginUtil.login(userName, userKeytabFile, krb5File, conf);
    }
}
```

#### 1.10.2.4.2 Security Authentication for HBase Data Read and Write and Global Secondary Index Example (Mutual Trust Scenarios)

### Description

When multiple clusters in different security modes need to access each other's resources, the administrator can set up a mutual trust system so that users of external systems can use the system. The usage range of users in each system is called a **domain**. Each Manager system must have a unique domain name. Cross-Manager access means users to be used across domains. For details about how to configure mutual trust between clusters, see "Managing Mutual Trust Relationships Between Managers" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

As a user that meets the cross-domain access requirements, you can use the keytab and principal files for Kerberos security authentication obtained from one Manager system and the client configuration files of multiple Manager systems to access and invoke the HBase service of multiple clusters after one authentication login in the multi-cluster mutual trust scenario.

The following code snippets belong to the **TestMultipleLogin** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-example** sample project.

- Code authentication

```
List<String> confDirectories = new ArrayList<>();
List<Configuration> confs = new LinkedList<>();
try {
    // conf directory
```

```
confDirectories.add("hadoopDomain");[1]
confDirectories.add("hadoop1Domain");[2]

for (String confDir : confDirectories) {
    confs.add(init(confDir));[3]
}

login(confs.get(0), confDirectories.get(0));[4]
} catch (IOException e) {
    LOG.error("Failed to login because ", e);
    return;
}
```

[1] **hadoopDomain** indicates the name of the directory for storing user credentials and the configuration file of a cluster. The relative path of the directory is **hbase-example/src/main/resources/hadoopDomain**, which can be changed as required.

[2] **hadoop1Domain** is the name of the directory for storing the configuration file of the other cluster. The relative path of the directory is **hbase-example/src/main/resources/hadoop1Domain**, which can be changed as required.

[3] Initialize the conf objects in sequence.

[4] Perform login authentication.

- Initialization configuration

```
private static Configuration init(String confDirectoryName) throws IOException {
    // Default load from conf directory
    Configuration conf = HBaseConfiguration.create();
    //In Windows environment
    String userdir = TestMain.class.getClassLoader().getResource(confDirectoryName).getPath() +
File.separator;
    //In Linux environment
    //String userdir = System.getProperty("user.dir") + File.separator + confDirectoryName +
File.separator;
    conf.addResource(new Path(userdir + "core-site.xml"), false);
    conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
    conf.addResource(new Path(userdir + "hbase-site.xml"), false);
    return conf;
}
```

## Prerequisites

You have obtained the configuration file and authentication file required for running the sample project. For details, see [Preparing the Connection Cluster Configuration File](#).

## Configuring Secure Login

Change **userName** in the **TestMultipleLogin** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-example** sample project to the actual user name, for example, **developuser**.

```
private static void login(Configuration conf, String confDir) throws IOException {

    if (User.isHBaseSecurityEnabled(conf)) {
        userName = " developuser ";

        //In Windows environment
        String userdir = TestMain.class.getClassLoader().getResource(confDir).getPath() + File.separator;
        //In Linux environment
        //String userdir = System.getProperty("user.dir") + File.separator + confDir + File.separator;

        userKeytabFile = userdir + "user.keytab";
```

```
    krb5File = userdir + "krb5.conf";

    /*
     * if need to connect zk, please provide jaas info about zk. of course,
     * you can do it as below:
     * System.setProperty("java.security.auth.login.config",confDirPath +
     * "jaas.conf"); but the demo can help you more : Note: if this process
     * will connect more than one zk cluster, the demo may be not proper. you
     * can contact us for more help
     */

    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userName,userKeytabFile);
    LoginUtil.login(userName, userKeytabFile, krb5File, conf);
}
```

### 1.10.2.4.3 Accessing HBase REST Service Security Authentication

#### Description

When installing the HBase service, you can optionally deploy the RESTServer instance. You can access the HBase REST service to invoke HBase operations, including operations on namespaces and tables. Kerberos authentication is also required for accessing the HBase REST service.

#### Prerequisites

You have obtained the configuration file and authentication file required for running the sample project. For details, see [Preparing the Connection Cluster Configuration File](#).

#### Configuring Secure Login

In this scenario, initial configuration is not required. Only the **keytab** and **krb5.conf** files used for Kerberos security authentication are required.

The following code snippets belong to the **HBaseRestTest** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-rest-example** sample project.

- Code authentication

Change **principal** to the actual user name, for example, **developuser**.

```
//In Windows environment
String userdir = HBaseRestTest.class.getClassLoader().getResource("conf").getPath() +
File.separator+[1]
//In Linux environment
//String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
String principal = "developuser";
login(principal, userKeytabFile, krb5File);
// RESTServer's hostname.
String restHostName = "10.120.16.170";[2]
String securityModeUrl = new
StringBuilder("https://").append(restHostName).append(":21309").toString();
String nonSecurityModeUrl = new
StringBuilder("http://").append(restHostName).append(":21309").toString();
HBaseRestTest test = new HBaseRestTest();

//If cluster is non-security mode,use nonSecurityModeUrl as parameter.
test.test(securityModeUrl);[3]
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation.

[2] Change the value of **restHostName** to the IP address of the node where the RestServer instance to be accessed is located, and configure the node IP address in the hosts file on the local host where the sample code is run. To obtain the IP address of the RestServer instance, log in to FusionInsight Manager and choose **Cluster > Services > HBase > Instance**.

[3] In security mode, access the HBase REST service in HTTPS mode and use **nonSecurityModeUrl** as the **test.test()** parameter.

- Security login

```
private static void login(String principal, String userKeytabFile, String krb5File) throws
LoginException {
    Map<String, String> options = new HashMap<>();
    options.put("useTicketCache", "false");
    options.put("useKeyTab", "true");
    options.put("keyTab", userKeytabFile);

    /**
     * Krb5 in GSS API needs to be refreshed so it does not throw the error
     * Specified version of key is not available
     */

    options.put("refreshKrb5Config", "true");
    options.put("principal", principal);
    options.put("storeKey", "true");
    options.put("doNotPrompt", "true");
    options.put("isInitiator", "true");
    options.put("debug", "true");
    System.setProperty("java.security.krb5.conf", krb5File);
    Configuration config = new Configuration() {
        @Override
        public AppConfigurationEntry[] getAppConfigurationEntry(String name) {
            return new AppConfigurationEntry[] {
                new AppConfigurationEntry("com.sun.security.auth.module.Krb5LoginModule",
                    AppConfigurationEntry.LoginModuleControlFlag.REQUIRED, options)
            };
        }
    };
    subject = new Subject(false, Collections.singleton(new KerberosPrincipal(principal)),
    Collections.EMPTY_SET,
    Collections.EMPTY_SET);
    LoginContext loginContext = new LoginContext("Krb5Login", subject, null, config);
    loginContext.login();
}
```

#### 1.10.2.4.4 Security Authentication for Accessing the ThriftServer Service

##### Scenario

HBase combines Thrift to provide HBase services for external applications. The ThriftServer instance is optional during HBase service installation. The ThriftServer system can access HBase users and has the read, write, execute, creation, and management permissions on all HBase namespaces and tables. Kerberos authentication is also required for accessing the ThriftServer service. HBase implements two sets of Thrift Server services. **hbase-thrift-example** is used to call the ThriftServer instance service.

##### Prerequisites

You have obtained the configuration file and authentication file required for running the sample project. For details, see [Preparing the Connection Cluster Configuration File](#).

## Sample Configuration

- Code authentication

The following code snippets belong to the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-thrift-example** sample project.

```
private static void init() throws IOException {
    // Default load from conf directory
    conf = HBaseConfiguration.create();

    String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
[1]   //In Linux environment
    //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
    conf.addResource(new Path(userdir + "core-site.xml"), false);
    conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
    conf.addResource(new Path(userdir + "hbase-site.xml"), false);
}
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation.

- Security login

Set **userName** to the actual username based on the actual situation, for example, **developuser**.

```
private static void login() throws IOException {
    if (User.isHBaseSecurityEnabled(conf)) {
        userName = " developuser ";

        //In Windows environment
        String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() +
File.separator;
        //In Linux environment
        //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

        userKeytabFile = userdir + "user.keytab";
        krb5File = userdir + "krb5.conf";

        /*
         * if need to connect zk, please provide jaas info about zk. of course,
         * you can do it as below:
         * System.setProperty("java.security.auth.login.config", confDirPath +
         * "jaas.conf"); but the demo can help you more : Note: if this process
         * will connect more than one zk cluster, the demo may be not proper. you
         * can contact us for more help
         */
        LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userName,
userKeytabFile);
        LoginUtil.login(userName, userKeytabFile, krb5File, conf);
    }
}
```

- Connecting to a ThriftServer instance

```
try {
    test = new ThriftSample();
    test.test("10.120.16.170", THRIFT_PORT, conf);[2]
} catch (TException | IOException e) {
    LOG.error("Test thrift error", e);
}
```

[2] The value of the input parameter **test.test()** is the IP address of the node where the target ThriftServer instance is deployed. Change the IP address to the actual one. The IP address of the node must be configured in the hosts file of the local host where the sample code is run.

**THRIFT\_PORT** is the value of **hbase.regionserver.thrift.port** configured for the ThriftServer instance. To obtain the IP address of the node where the

ThriftServer instance is deployed, log in to FusionInsight Manager, choose **Cluster > Services > HBase**, and click the **Instance** tab.

### 1.10.2.4.5 Security Authentication for Accessing Multiple ZooKeepers

#### Scenario

To avoid ZooKeeper authentication conflicts when a client process accesses a FusionInsight ZooKeeper and a third-party ZooKeeper at the same time, sample code is provided for the HBase client to access the FusionInsight ZooKeeper and for customer applications to access the third-party ZooKeeper.

#### Prerequisites

You have obtained the configuration file and authentication file required for running the sample project. For details, see [Preparing for Development Environment](#).

#### Sample Configuration

The following lists the authentication configuration files in the **src/main/resources** directory.

- **zoo.cfg**

```
# The configuration in jaas.conf used to connect fi
zookeeper.zookeeper.sasl.clientconfig=Client_new[1]
# Principal of fi zookeeper server side.
zookeeper.server.principal=zookeeper/hadoop.hadoop.com[2]
# Set true if the fi cluster is security mode.
# The other two parameters do not take effect if the value is false.
zookeeper.sasl.client=true[3]
```

[1] **zookeeper.sasl.clientconfig**: specifies the configuration in the **jaas.conf** file used for accessing FusionInsight ZooKeeper.

[2] **zookeeper.server.principal**: specifies the principal used by the ZooKeeper server. The format is **zookeeper/hadoop.System domain name**, for example, **zookeeper/hadoop.HADOOP.COM**. To obtain the system domain name, log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and view the value of **Local Domain**.

[3] **zookeeper.sasl.client**: If the MRS cluster works in the security mode, set this parameter to **true**. Otherwise, set this parameter to **false**. If this parameter is set to **false**, the **zookeeper.sasl.clientconfig** and **zookeeper.server.principal** parameters do not take effect.
- **jaas.conf**

```
Client_new { [4]
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    keyTab="D:\\work\\sample_project\\src\\hbase-examples\\hbase-zk-example\\target\\classes\\conf\\user.keytab" [5]
    principal="hbaseuser1"
    useTicketCache=false
    storeKey=true
    debug=true;
}
Client { [6]
    org.apache.zookeeper.server.auth.DigestLoginModule required
    username="bob"
    password="xxxxxx"; [7]
}
```

- [4] **Client\_new**: reads configuration specified in the **zoo.cfg** file. When the name is changed, the corresponding configuration in the **zoo.cfg** file must be modified accordingly.
- [5] **keyTab**: specifies the path for storing the **user.keytab** file used by the project on the host where the sample is run. Use an absolute path to better locate the file. Use \\ in the Windows and \ in Linux.
- [6] **Client**: A third-party ZooKeeper uses this configuration for access. The connection authentication configuration depends on the third-party ZooKeeper version.
- [7] **password**: Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.

### 1.10.2.4.6 Security Authentication for Interconnecting HBase/Phoenix with Spring Boot

#### Scenario

Run Spring Boot interface sample code of MRS HBase/Phoenix.

#### Prerequisites

You have obtained the configuration file and authentication file required for running the sample project. For details, see [Preparing the Connection Cluster Configuration File](#).

#### Sample Configuration

**Step 1** In the IntelliJ IDEA development environment, click the **springclient.properties** file in the **src/springboot/hbase-examples/src/main/resources** directory and modify the parameters listed in [Table 1-79](#) as needed:

**Table 1-79** Configuration parameters

Parameter	Definition
principal	Username created when you prepare a developer account.
user.keytab.path	Path of the user authentication file <b>user.keytab</b> created during developer account preparation.
krb5.conf.path	Path of the user authentication file <b>krb5.conf</b> created during developer account preparation.
conf.path	Path of the HBase configuration file.

Parameter	Definition
zookeeper.server.principal	Principal of the ZooKeeper server, in <b>zookeeper/hadoop.System domain name</b> format. To obtain the system domain name, log in to FusionInsight Manager, choose <b>System &gt; Permission &gt; Domain and Mutual Trust</b> , and view the value of <b>Local Domain</b> .

----End

## 1.10.3 Developing an Application

### 1.10.3.1 HBase Data Read/Write Sample Program

#### 1.10.3.1.1 Service Scenario Description

Based on typical scenarios, users can quickly learn and master the HBase development process and know important interface functions.

#### Scenario

Develop an application to manage information about users who use service A in an enterprise. **Table 1-80** provides the user information. The operation process of service A is described as follows:

- Create a user information table.
- Add diplomas and titles to the user information table.
- Query user names and addresses by user ID.
- Query information by user name.
- Query information about users whose ages range from 20 to 29.
- Collect statistics on the number and the maximum, minimum, and average ages of users.
- Deregister users, and delete user data.
- Delete the user information table after service A ends.

**Table 1-80** User information

ID	Name	Gender	Age	Address
1200500020 1	Zhang San	Male	19	Shenzhen, Guangdong
1200500020 2	Li Wanting	Female	23	Shijiazhuang, Hebei
1200500020 3	Wang Ming	Male	26	Ningbo, Zhejiang

ID	Name	Gender	Age	Address
1200500020 4	Li Gang	Male	18	Xiangyang, Hubei
1200500020 5	Zhao Enru	Female	21	Shangrao, Jiangxi
1200500020 6	Chen Long	Male	32	Zhuzhou, Hunan
1200500020 7	Zhou Wei	Female	29	Nanyang, Henan
1200500020 8	Yang Yiwen	Female	30	Kaixian, Chongqing
1200500020 9	Xu Bing	Male	26	Weinan, Shaanxi
1200500021 0	Xiao Kai	Male	25	Dalian, Liaoning

## Data Planning

Proper design of the table structure, RowKeys, and column names can give full play to the advantages of HBase. In this example, the unique ID is used as the RowKey, and columns are stored in the **info** column family.

### CAUTION

HBase tables are stored in *Namespace:Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table namespace.

### 1.10.3.1.2 Application Development Approach

#### Function Decomposition

Functions are decomposed based on the preceding service scenario. [Table 1-81](#) provides the functions to be developed.

**Table 1-81** Functions to be developed in HBase

No.	Procedure	Code Implementation
1	Create a table based on <a href="#">Table 1-80</a> .	For details, see <a href="#">Creating a Table</a> .

No.	Procedure	Code Implementation
2	Import user data.	For details, see <a href="#">Inserting Data</a> .
3	Add the <b>Education</b> column family, and add diplomas and titles to the user information table.	For details, see <a href="#">Modifying a Table</a> .
4	Query user names and addresses by user ID.	For details, see <a href="#">Reading Data Using Get</a> .
5	Query information by user name.	For details, see <a href="#">Filtering Data</a> .
6	To improve query performance, create or delete secondary indexes.	For details, see <a href="#">Creating a Secondary Index</a> and <a href="#">Secondary Index-based Query</a> .
7	Deregister users, and delete user data.	For details, see <a href="#">Deleting Data</a> .
8	Delete the user information table after service A ends.	For details, see <a href="#">Deleting a Table</a> .

## Key Design Principle

HBase is a distributed database system based on the lexicographic order of RowKeys. The RowKey design has great impact on performance, so the RowKeys must be designed based on specific services.

### 1.10.3.1.3 Creating Configuration

#### Function

HBase obtain configuration items by using the login method. The configuration items include user login information and security authentication information.

#### Example Codes

The following code snippet belongs to the `createClientConf` method in the `createClientConf` class of the `com.huawei.bigdata.hadoop.security.examples` package.

```
public static Configuration createClientConf() {  
    // In Windows environment  
    String userDir = Utils.class.getClassLoader().getResource(CONF_DIRECTORY).getPath() + File.separator;  
    // In Linux environment  
    // String userDir = System.getProperty("user.dir") + File.separator + CONF_DIRECTORY + File.separator;  
    return createConfByUserDir(userDir);  
}  
public static Configuration createConfByUserDir(String userDir) {  
    // Default load from conf directory  
    Configuration conf = HBaseConfiguration.create();  
    if (userDir == null || userDir.isEmpty()) {
```

```
        return conf;
    }
    conf.addResource(new Path(userDir + CLIENT_CORE_FILE), false);
    conf.addResource(new Path(userDir + CLIENT_HDFS_FILE), false);
    conf.addResource(new Path(userDir + CLIENT_HBASE_FILE), false);
    return conf;
}
```

#### 1.10.3.1.4 Creating Connection

### Function

HBase creates a Connection object using the `ConnectionFactory.createConnection(configuration)` method. The transferred parameter is the Configuration created in the last step.

Connection encapsulates the connections between underlying applications and servers and ZooKeeper. Connection is instantiated using the `ConnectionFactory` class. Creating Connection is a heavyweight operation. Connection is thread-safe. Therefore, multiple client threads can share one Connection.

In a typical scenario, a client program uses a Connection, and each thread obtains its own Admin or Table instance and invokes the operation interface provided by the Admin or Table object. You are not advised to cache or pool Table and Admin. The lifecycle of Connection is maintained by invokers that frees up resources by invoking `close()`.

### Example Code

The following code snippet exemplifies login, creating Connection, and creating a table. It belongs to the `HbaseSample` method in the `HbaseSample` class of the `com.huawei.bigdata.hbase.examples` package.

```
private TableName tableName = null;
private Connection conn = null;

public HbaseSample(Configuration conf) throws IOException {
    this.tableName = TableName.valueOf("hbase_sample_table");
    this.conn = ConnectionFactory.createConnection(conf);
}
```



Avoid invoking login code repeatedly.

#### 1.10.3.1.5 Creating a Table

### Function

Create a table using the `createTable` method of the `org.apache.hadoop.hbase.client.Admin` object and specify the table name and column family name. Tables can be created in two modes. (The mode of creating a table using preassigned regions is strongly recommended.)

- Quickly create a table. A newly created table contains only one region which will be split into multiple new regions as data increases.

- Create a table using preassigned regions. Preassign multiple regions before creating a table. This mode accelerates data write at the beginning of massive data write.

 NOTE

The column name and column family name of an HBase table consists of letters, digits, and underscores and cannot contain any special characters.

## Example Code

The following code snippet belongs to the **testCreateTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testCreateTable() {  
    LOG.info("Entering testCreateTable.");  
    // Specify the table descriptor.  
    TableDescriptorBuilder htd = TableDescriptorBuilder.newBuilder(tableName);(1)  
  
    // Set the column family name to info.  
    ColumnFamilyDescriptorBuilder hcd =  
    ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info"));(2)  
  
    // Set data encoding methods, HBase provides DIFF,FAST_DIFF,PREFIX  
  
    hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);  
    // Set compression methods, HBase provides two default compression  
    // methods:GZ and SNAPPY  
    // GZ has the highest compression rate, but low compression and  
    // decompression efficiency, fit for cold data  
    // SNAPPY has low compression rate, but high compression and  
    // decompression efficiency, fit for hot data.  
    // It is advised to use SNAPPY  
    hcd.setCompressionType(Compression.Algorithm.SNAPPY); //Note [1]  
    htd.setColumnFamily(hcd.build()); (3)  
    Admin admin = null;  
    try {  
        // Instantiate an Admin object.  
        admin = conn.getAdmin(); (4)  
        if (!admin.tableExists(tableName)) {  
            LOG.info("Creating table...");  
            admin.createTable(htd.build()); //Note [2] (5)  
            LOG.info(admin.getClusterMetrics().toString());  
            LOG.info(admin.listNamespaceDescriptors().toString());  
            LOG.info("Table created successfully.");  
        } else {  
            LOG.warn("table already exists");  
        }  
    } catch (IOException e) {  
        LOG.error("Create table failed " ,e);  
    } finally {  
        if (admin != null) {  
            try {  
                // Close the Admin object.  
                admin.close();  
            } catch (IOException e) {  
                LOG.error("Failed to close admin " ,e);  
            }  
        }  
    }  
    LOG.info("Exiting testCreateTable.");  
}
```

## Explanation

1. Create a table descriptor.

2. Create a column family descriptor.
3. Add the column family descriptor to the table descriptor.
4. Obtain an Admin object. The Admin object provides functions for creating a table, creating a column family, checking whether a table exists, modifying the table structure, modifying the column family structure, and deleting a table.
5. Invoke the table creation method of Admin.

## Precautions

- 1. The compression mode of a column family can be set. The code snippets are as follows:

```
// Set the encoding algorithm. HBase supports the DIFF, FAST_DIFF, PREFIX encoding algorithms.  
hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);  
  
// Set the file compression mode. HBase provides the GZ and SNAPPY compression algorithms by default.  
// GZ provides a high compression rate but low compression and decompression performance. GZ is suitable for cold data.  
// SNAPPY provides a low compression rate but high compression and decompression performance. SNAPPY is suitable for hot data.  
// It is recommended that SNAPPY be enabled by default.  
hcd.setCompressionType(Compression.Algorithm.SNAPPY);
```
- 2. A table can be created by specifying the start and end RowKeys or preassigning regions using RowKey arrays. The code snippets are as follows:

```
// Create a table whose regions are preassigned.  
byte[][] splits = new byte[4][];  
splits[0] = Bytes.toBytes("A");  
splits[1] = Bytes.toBytes("H");  
splits[2] = Bytes.toBytes("O");  
splits[3] = Bytes.toBytes("U");  
admin.createTable(htd, splits);
```

### 1.10.3.1.6 Deleting a Table

#### Function

Delete a table using the **deleteTable** method of **org.apache.hadoop.hbase.client.Admin**.

#### Example Code

The following code snippet belongs to the **dropTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropTable() {  
    LOG.info("Entering dropTable.");  
    Admin admin = null;  
    try {  
        admin = conn.getAdmin();  
        if (admin.tableExists(tableName)) {  
            // Disable the table before deleting it.  
            admin.disableTable(tableName);  
            // Delete table.  
            admin.deleteTable(tableName); // Note[1]  
        }  
        LOG.info("Drop table successfully.");  
    } catch (IOException e) {  
        LOG.error("Drop table failed " ,e);  
    } finally {
```

```
if (admin != null) {  
    try {  
        // Close the Admin object.  
        admin.close();  
    } catch (IOException e) {  
        LOG.error("Close admin failed " ,e);  
    }  
}  
LOG.info("Exiting dropTable.");
```

## Precautions

A table can be deleted only when the table is disabled. Therefore, **deleteTable** is used together with **disableTable**, **enableTable**, **tableExists**, **isTableEnabled**, and **isTableDisabled**.

### 1.10.3.1.7 Modifying a Table

#### Function

Modify table information using the **modifyTable** method of **org.apache.hadoop.hbase.client.Admin**.

#### Example Code

The following code snippet belongs to the **testModifyTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testModifyTable() {  
    LOG.info("Entering testModifyTable.");  
  
    // Specify the column family name.  
    byte[] familyName = Bytes.toBytes("education");  
    Admin admin = null;  
    try {  
        // Instantiate an Admin object.  
        admin = conn.getAdmin();  
        // Obtain the table descriptor.  
        TableDescriptor htd = admin.getTableDescriptor(tableName);  
  
        // Check whether the column family is specified before modification.  
        if (!htd.hasColumnFamily(familyName)) {  
            // Create the column descriptor.  
            TableDescriptor tableBuilder = TableDescriptor.newBuilder(htd)  
                .setColumnFamily(ColumnFamilyDescriptor.newBuilder(familyName).build()).build();  
  
            // Disable the table to get the table offline before modifying  
            // the table.  
            admin.disableTable(tableName); // Note[1]  
            // Submit a modifyTable request.  
            admin.modifyTable(tableBuilder);  
            // Enable the table to get the table online after modifying the  
            // table.  
            admin.enableTable(tableName);  
        }  
        LOG.info("Modify table successfully.");  
    } catch (IOException e) {  
        LOG.error("Modify table failed " ,e);  
    } finally {  
        if (admin != null) {  
            try {
```

```
// Close the Admin object.  
admin.close();  
} catch (IOException e) {  
    LOG.error("Close admin failed " ,e);  
}  
}  
}  
LOG.info("Exiting testModifyTable.");  
}
```

## Precautions

**modifyTable** takes effect only when a table is disabled.

### 1.10.3.1.8 Inserting Data

#### Function

HBase is a column-oriented database. A row of data may have multiple column families, and a column family may contain multiple columns. When writing data, you must specify the columns (including the column family names and column names) to which data is written. In HBase, data (a row of data or data sets) is inserted using the **put** method of HTable.

#### Example Code

The following code snippet belongs to the **testPut** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testPut() {  
    LOG.info("Entering testPut.");  
  
    // Specify the column family name.  
    byte[] familyName = Bytes.toBytes("info");  
    // Specify the column name.  
    byte[][] qualifiers = {  
        Bytes.toBytes("name"), Bytes.toBytes("gender"), Bytes.toBytes("age"), Bytes.toBytes("address")  
    };  
  
    Table table = null;  
    try {  
        // Instantiate an HTable object.  
        table = conn.getTable(tableName);  
        List<Put> puts = new ArrayList<Put>();  
  
        // Instantiate a Put object.  
        Put put = putData(familyName, qualifiers,  
            Arrays.asList("012005000201", "Zhang San", "Male", "19", "Shenzhen, Guangdong"));  
        puts.add(put);  
  
        put = putData(familyName, qualifiers,  
            Arrays.asList("012005000202", "Li Wanting", "Female", "23", "Shijiazhuang, Hebei"));  
        puts.add(put);  
  
        put = putData(familyName, qualifiers,  
            Arrays.asList("012005000203", "Wang Ming", "Male", "26", "Ningbo, Zhejiang"));  
        puts.add(put);  
  
        put = putData(familyName, qualifiers,  
            Arrays.asList("012005000204", "Li Gang", "Male", "18", "Xiangyang, Hubei"));  
        puts.add(put);  
  
        put = putData(familyName, qualifiers,  
            Arrays.asList("012005000205", "Zhao Enru", "Female", "21", "Shangrao, Jiangxi"));  
    } catch (IOException e) {  
        LOG.error("Error occurred while performing testPut.", e);  
    } finally {  
        if (table != null) {  
            try {  
                table.close();  
            } catch (IOException e) {  
                LOG.error("Error occurred while closing table.", e);  
            }  
        }  
    }  
}
```

```
puts.add(put);

put = putData(familyName, qualifiers,
    Arrays.asList("012005000206", "Chen Long", "Male", "32", "Zhuzhou, Hunan"));
puts.add(put);

put = putData(familyName, qualifiers,
    Arrays.asList("012005000207", "Zhou Wei", "Female", "29", "Nanyang, Henan"));
puts.add(put);

put = putData(familyName, qualifiers,
    Arrays.asList("012005000208", "Yang Yiwen", "Female", "30", "Kaixian, Chongqing"));
puts.add(put);

put = putData(familyName, qualifiers,
    Arrays.asList("012005000209", "Xu Bing", "Male", "26", "Weinan, Shaanxi"));
puts.add(put);

put = putData(familyName, qualifiers,
    Arrays.asList("012005000210", "Xiao Kai", "Male", "25", "Dalian, Liaoning"));
puts.add(put);

// Submit a put request.
table.put(puts);

LOG.info("Put successfully.");
} catch (IOException e) {
    LOG.error("Put failed ", e);
} finally {
    if (table != null) {
        try {
            // Close the HTable object.
            table.close();
        } catch (IOException e) {
            LOG.error("Close table failed ", e);
        }
    }
}
LOG.info("Exiting testPut.");
}
```

## Precautions

Multiple threads are not allowed to use the same HTable instance at the same time. HTable is a non-thread safe class. If an HTable instance is used by multiple threads at the same time, concurrency problems will occur.

### 1.10.3.1.9 Deleting Data

#### Function

Delete data (a row of data or data sets) using the **delete** method of a Table instance.

#### Example Code

The following code snippet belongs to the **testDelete** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testDelete() {
    LOG.info("Entering testDelete.");

    byte[] rowKey = Bytes.toBytes("012005000201");
```

```
Table table = null;
try {
    // Instantiate an HTable object.
    table = conn.getTable(tableName);

    // Instantiate a Delete object.
    Delete delete = new Delete(rowKey);

    // Submit a delete request.
    table.delete(delete);

    LOG.info("Delete table successfully.");
} catch (IOException e) {
    LOG.error("Delete table failed " ,e);
} finally {
    if (table != null) {
        try {
            // Close the HTable object.
            table.close();
        } catch (IOException e) {
            LOG.error("Close table failed " ,e);
        }
    }
}
LOG.info("Exiting testDelete.");
```

 **NOTE**

If secondary index is created in the family of the column where the deleted cell is, the index data is synchronously deleted.

### 1.10.3.1.10 Reading Data Using Get

#### Function

Before reading data from a table, instantiate the Table instance of the table, and then create a Get object. You can also set parameters for the Get object, such as the column family name and column name. Query results are stored in the Result object that stores multiple Cells.

#### Example Code

The following code snippet belongs to the **testGet** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testGet() {
    LOG.info("Entering testGet.");
    // Specify the column family name.
    byte[] familyName = Bytes.toBytes("info");
    // Specify the column name.
    byte[][] qualifier = { Bytes.toBytes("name"), Bytes.toBytes("address") };
    // Specify RowKey.
    byte[] rowKey = Bytes.toBytes("012005000201");
    Table table = null;
    try {
        // Create the Table instance.
        table = conn.getTable(tableName);
        // Instantiate a Get object.
        Get get = new Get(rowKey);
        // Set the column family name and column name.
        get.addColumn(familyName, qualifier[0]);
        get.addColumn(familyName, qualifier[1]);
        // Submit a get request.
        Result result = table.get(get);
```

```
// Print query results.
for (Cell cell : result.rawCells()) {
    LOG.info("{}:{}:{}",
        Bytes.toString(CellUtil.cloneRow(cell)),
        Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
        Bytes.toString(CellUtil.cloneValue(cell)));
}
LOG.info("Get data successfully.");
} catch (IOException e) {
    LOG.error("Get data failed " ,e);
} finally {
    if (table != null) {
        try {
            // Close the HTable object.
            table.close();
        } catch (IOException e) {
            LOG.error("Close table failed " ,e);
        }
    }
}
LOG.info("Exiting testGet.");
}
```

### 1.10.3.1.11 Reading Data Using Scan

#### Function

Before reading data from a table, instantiate the Table instance of the table, create a Scan object, and set parameters for the Scan object based on search criteria. To improve query efficiency, you are advised to specify **StartRow** and **StopRow**. Query results are stored in the ResultScanner object where each row of data is stored as a Result object that stores multiple Cells.

#### Example Code

The following code snippet belongs to the **testScanData** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testScanData() {
    LOG.info("Entering testScanData.");
    Table table = null;
    // Instantiate a ResultScanner object.
    ResultScanner rScanner = null;
    try {
        // Create the Configuration instance.
        table = conn.getTable(tableName);
        // Instantiate a Get object.
        Scan scan = new Scan();
        scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
        // Set the cache size.
        scan.setCaching(1000);

        // Submit a scan request.
        rScanner = table.getScanner(scan);
        // Print query results.
        for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
            for (Cell cell : r.rawCells()) {
                LOG.info("{}:{}:{}",
                    Bytes.toString(CellUtil.cloneRow(cell)),
                    Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
                    Bytes.toString(CellUtil.cloneValue(cell)));
            }
        }
        LOG.info("Scan data successfully.");
    } catch (IOException e) {
        LOG.error("Scan data failed " ,e);
    } finally {
```

```
if (rScanner != null) {  
    // Close the scanner object.  
    rScanner.close();  
}  
if (table != null) {  
    try {  
        // Close the HTable object.  
        table.close();  
    } catch (IOException e) {  
        LOG.error("Close table failed ",e);  
    }  
}  
LOG.info("Exiting testScanData.");  
}
```

## Precuations

1. You are advised to specify **StartRow** and **StopRow** to ensure good performance with a specified Scan scope.
2. You can set **Batch** and **Caching**.
  - **Batch**  
Indicates the maximum number of records returned each time when the **next** interface is invoked using Scan. This parameter is related to the number of columns read each time.
  - **Caching**  
Indicates the maximum number of **next** records returned for a remote procedure call (RPC) request. This parameter is related to the number of rows read by an RPC.

### 1.10.3.1.12 Filtering Data

#### Function

HBase Filter is used to filter data during Scan and Get. You can specify the filter criteria, such as filtering by RowKey, column name, or column value.

#### Example Code

The following code snippet belongs to the **testSingleColumnValueFilter** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testSingleColumnValueFilter() {  
    LOG.info("Entering testSingleColumnValueFilter.");  
    Table table = null;  
  
    ResultScanner rScanner = null;  
  
    try {  
        table = conn.getTable(tableName);  
  
        Scan scan = new Scan();  
        scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));  
        // Set the filter criteria.  
        SingleColumnValueFilter filter = new SingleColumnValueFilter(  
            Bytes.toBytes("info"), Bytes.toBytes("name"), CompareOperator.EQUAL,  
            Bytes.toBytes("Xu Bing"));  
        scan.setFilter(filter);  
        // Submit a scan request.  
    }
```

```
rScanner = table.getScanner(scan);
// Print query results.
for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
    for (Cell cell : r.rawCells()) {
        LOG.info("{{}:{}:{}:{}}", Bytes.toString(CellUtil.cloneRow(cell)),
            Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
            Bytes.toString(CellUtil.cloneValue(cell)));
    }
}
LOG.info("Single column value filter successfully.");
} catch (IOException e) {
    LOG.error("Single column value filter failed " ,e);
} finally {
    if (rScanner != null) {
        // Close the scanner object.
        rScanner.close();
    }
    if (table != null) {
        try {
            // Close the HTable object.
            table.close();
        } catch (IOException e) {
            LOG.error("Close table failed " ,e);
        }
    }
}
LOG.info("Exiting testSingleColumnValueFilter.");
```

## Precautions

Currently, secondary indexes do not support the comparators that use objects of the SubstringComparator class as filters.

For example, the following sample code is not supported:

```
Scan scan = new Scan();
filterList = new FilterList(FilterList.Operator.MUST_PASS_ALL);
filterList.addFilter(new SingleColumnValueFilter(Bytes
.toBytes(columnFamily), Bytes.toBytes(qualifier),
CompareOperator.EQUAL, new SubstringComparator(substring)));
scan.setFilter(filterList);
```

### 1.10.3.1.13 Creating a Secondary Index

#### Function

You can manage HBase secondary indexes using methods provided in **org.apache.hadoop.hbase.hindex.client.HIndexAdmin**. This class provides methods of creating an index.



#### NOTE

Secondary indexes cannot be modified. If you need to modify them, delete old indexes and create new ones.

#### Example Code

The following code snippet belongs to the **createIndex** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void createIndex() {
    LOG.info("Entering createIndex.");
```

```

String indexName = "index_name";
// Create hindex instance
TableIndices tableIndices = new TableIndices();
IndexSpecification iSpec = new IndexSpecification(indexName);
iSpec.addIndexColumn(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info")).build(),
    "name", ValueType.STRING); // Note[1]
tableIndices.addIndex(iSpec);

HIndexAdmin iAdmin = null;
Admin admin = null;
try {

    admin = conn.getAdmin();
    iAdmin = HIndexClient.newHIndexAdmin(admin);

    // add index to the table
    iAdmin.addIndices(tableName, tableIndices);

    LOG.info("Create index successfully.");
} catch (IOException e) {
    LOG.error("Create index failed ", e);
} finally {
    if (admin != null) {
        try {
            admin.close();
        } catch (IOException e) {
            LOG.error("Close admin failed ", e);
        }
    }
    if (iAdmin != null) {
        try {
            // Close IndexAdmin Object
            iAdmin.close();
        } catch (IOException e) {
            LOG.error("Close admin failed ", e);
        }
    }
}
LOG.info("Exiting createIndex.");
}

```

By default, newly created level-2 indexes are disabled. To enable a specified level-2 index, see the following code snippet. The following code snippet belongs to the `enableIndex` method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```

public void enableIndex() {
    LOG.info("Entering createIndex.");

    // Name of the index to be enabled
    String indexName = "index_name";

    List<String> indexNameList = new ArrayList<String>();
    indexNameList.add(indexName);

    HIndexAdmin iAdmin = null;
    Admin admin = null;
    try {
        admin = conn.getAdmin();
        iAdmin = HIndexClient.newHIndexAdmin(admin);

        // Alternately, enable the specified indices
        iAdmin.enableIndices(tableName, indexNameList);
        LOG.info("Successfully enable indices {} of the table {}", indexNameList, tableName);
    } catch (IOException e) {
        LOG.error("Failed to enable indices {} of the table {} . {}", indexNameList, tableName, e);
    } finally {
        if (admin != null) {
            try {

```

```
        admin.close();
    } catch (IOException e) {
        LOG.error("Close admin failed ", e);
    }
}
if (iAdmin != null) {
    try {
        iAdmin.close();
    } catch (IOException e) {
        LOG.error("Close admin failed ", e);
    }
}
}
```

## Precautions

Create a combination index.

HBase supports creation of secondary indexes on multiple fields, for example, the name and age columns.

```
HIndexSpecification iSpecUnite = new HIndexSpecification(indexName);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "name", ValueType.String, 10);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "age", ValueType.String, 3);
```

## Related Operations

### Create an index table by running a command.

You can also use the TableIndexer tool to create an index in an existing user table.



The <table\_name> user table must exist.

```
hbase org.apache.hadoop.hbase.hindex.mapreduce.TableIndexer -Dtablename.to.index=<table_name> -
Dindexspecs.to.add='IDX1=>cf1:[q1->datatype];cf2:[q2->datatype],[q3-
>datatype]#IDX2=>cf1:[q5->datatype]' -Dindexnames.to.build='IDX1'
```

A number sign "#" is used to separate indexes. A semicolon ";" is used to separate column families. A comma "," is used to separate columns.

**tablename.to.index:** indicates the name of the table where the index is created.

**indexspecs.to.add:** indicates the user table columns corresponding to the index.

The parameters in the command are described as follows:

- **IDX1:** indicates the index name.
- **cf1:** indicates the column family name.
- **q1:** indicates the column name.
- **datatype:** indicates the data type. Only the Integer, String, Double, Float, Long, Short, Byte and Char formats are supported.

### 1.10.3.1.14 Deleting an Index

#### Function

You can manage HBase secondary indexes using methods provided in **org.apache.hadoop.hbase.hindex.client.HIndexAdmin**. This class provides methods of querying and deleting an index.

#### Example Code

The following code snippet belongs to the **dropIndex** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropIndex() {
    LOG.info("Entering dropIndex.");
    String indexName = "index_name";
    List<String> indexNameList = new ArrayList<String>();
    indexNameList.add(indexName);

    IndexAdmin iAdmin = null;
    try {
        // Instantiate HIndexAdmin Object
        iAdmin = HIndexClient.newHIndexAdmin(conn.getAdmin());
        // Delete Secondary Index
        iAdmin.dropIndex(tableName, indexNameList);

        LOG.info("Drop index successfully.");
    } catch (IOException e) {
        LOG.error("Drop index failed.");
    } finally {
        if (iAdmin != null) {
            try {
                // Close Secondary Index
                iAdmin.close();
            } catch (IOException e) {
                LOG.error("Close admin failed.");
            }
        }
    }
    LOG.info("Exiting dropIndex.");
}
```

### 1.10.3.1.15 Secondary Index-based Query

#### Function

In user tables with secondary indexes, you can use Filter to query data. The data query performance is higher than that in user tables without secondary indexes.

#### NOTE

- For details about the secondary index principles, see "HIndex" in the "Product Introduction > Components > HBase > HBase Enhanced Open Source Features".
- HIndex supports three Filter types: SingleColumnValueFilter, SingleColumnValueExcludeFilter, and SingleColumnValuePartitionFilter.
- HIndex supports the following Comparator types: binary comparator, bit comparator, long comparator, decimal comparator, double comparator, float comparator, int comparator, and null comparator.

The secondary index usage rules are as follows:

- For scenarios in which a single index is created for one or multiple columns:
  - When you use this column for AND or OR query filtering, the index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND/OR  
Filter\_Condition(IndexCol2).
  - When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND  
Filter\_Condition(IndexCol2) AND Filter\_Condition(NonIndexCol1).
  - When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.  
For example, Filter\_Condition(IndexCol1) AND/OR  
Filter\_Condition(IndexCol2) OR Filter\_Condition(NonIndexCol1).
- For scenarios in which a combination index is created for multiple columns:
  - When the columns used for query are all or part of the columns of the combination index and are in the same sequence with the combination index, the index is used to improve the query performance.  
For example, a combination index is created for C1, C2, and C3. The index takes effect in the following scenarios:  
Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND  
Filter\_Condition(IndexCol3)  
Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2)  
Filter\_Condition(IndexCol1)  
The index does not take effect in the following scenarios:  
Filter\_Condition(IndexCol2) AND Filter\_Condition(IndexCol3)  
Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol3)  
Filter\_Condition(IndexCol2)  
Filter\_Condition(IndexCol3)
  - When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.  
For example:  
Filter\_Condition(IndexCol1) AND Filter\_Condition(NonIndexCol1)  
Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND  
Filter\_Condition(NonIndexCol1)
  - When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.  
For example:  
Filter\_Condition(IndexCol1) OR Filter\_Condition(NonIndexCol1)  
(Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2))OR  
( Filter\_Condition(NonIndexCol1))
  - When multiple columns are used for query, a value range can be specified only for the last column in the combination index and the other columns can only be set to a specified value.

For example, a combination index is created for C1, C2, and C3. In range query, a value range can be set only for C3 and the filter criterion is "C1 = XXX, C2 = XXX, and C3 = value range".

- For scenarios in which secondary index is created in a user table, you can use Filter to query data. The query results of the single and combination index with filter are the same as those in the table without secondary index. The data query performance is higher than that in user tables without secondary indexes.

## Example Code

The following code snippet belongs to the **testScanDataByIndex** method in the **HbaseSample** class of the **com.huawei.hadoop.hbase.example** package.

### Example: Query data using secondary indexes.

```
public void testScanDataByIndex() {
    LOG.info("Entering testScanDataByIndex.");
    Table table = null;
    ResultScanner scanner = null;
    try {
        table = conn.getTable(tableName);

        // Create a filter for indexed column.
        Filter filter = new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("name"),
            CompareOperator.EQUAL, "Li Gang".getBytes());
        Scan scan = new Scan();
        scan.setFilter(filter);
        scanner = table.getScanner(scan);
        LOG.info("Scan indexed data.");

        for (Result result : scanner) {
            for (Cell cell : result.rawCells()) {
                LOG.info("{}:{}:{}",
                    Bytes.toString(CellUtil.cloneRow(cell)),
                    Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
                    Bytes.toString(CellUtil.cloneValue(cell)));
            }
        }
        LOG.info("Scan data by index successfully.");
    } catch (IOException e) {
        LOG.error("Scan data by index failed.");
    } finally {
        if (scanner != null) {
            // Close the scanner object.
            scanner.close();
        }
        try {
            if (table != null) {
                table.close();
            }
        } catch (IOException e) {
            LOG.error("Close table failed.");
        }
    }
    LOG.info("Exiting testScanDataByIndex.");
}
```

## Precaution

Create secondary indexes for the **name** field first.

## Related Operations

Query a table using a secondary index.

The following provides an example:

Add an index to the **name** column of the **info** column family in **hbase\_sample\_table**. Run the following command on the client:

```
hbase org.apache.hadoop.hbase.hindex.mapreduce.TableIndexer -Dtablename.to.index=hbase_sample_table -Dindexspecs.to.add='IDX1=>info:[name->String]' -Dindexnames.to.build='IDX1'
```

Query **info:name**. Run the following command on the HBase shell client:

```
>scan 'hbase_sample_table',  
{FILTER=>"SingleColumnValueFilter(family,qualifier,compareOp,comparator,filterIfMissing,latestVersionOnly)"}
```



Use APIs to perform complex query on the HBase shell client.

The parameters are described as follows:

- **family**: indicates the column family where the column to be queried locates, such as **info**.
- **qualifier**: indicates the column to be queried, such as **name**.
- **compareOp**: indicates the comparison operator, such as = and >.
- **comparator**: indicates the target value to be queried, such as **binary:Zhang San**.
- **filterIfMissing**: indicates whether a row is filtered if the column does not exist in this row. The default value is **false**.
- **latestVersionOnly**: indicates whether only values of the latest version are to be queried. The default value is **false**.

For example:

```
>scan 'hbase_sample_table',{FILTER=>"SingleColumnValueFilter('info','name',=,'binary:Zhang San',true,true)"}
```

### 1.10.3.1.16 Multi-Point Region Division

## Function

You can perform multi-point division by using **org.apache.hadoop.hbase.client.HBaseAdmin**. Note that the division operations take effect on empty regions only.

In this example, the multi-point division is performed on an HBase table by using **multiSplit**. The table will be split into 5 parts: "-∞~A", "A~D", "D~F", "F~H", and "H~+∞".

## Example Code

The following code snippet belongs to the **testMultiSplit** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testMultiSplit() {  
    LOG.info("Entering testMultiSplit.");  
  
    Table table = null;  
    Admin admin = null;
```

```

try {
    admin = conn.getAdmin();

    // initialize a HTable object
    table = conn.getTable(tableName);
    Set<HRegionInfo> regionSet = new HashSet<HRegionInfo>();
    List<HRegionLocation> regionList = conn.getRegionLocator(tableName).getAllRegionLocations();
    for(HRegionLocation hrl : regionList){
        regionSet.add(hrl.getRegionInfo());
    }
    byte[][] sk = new byte[4][];
    sk[0] = "A".getBytes();
    sk[1] = "D".getBytes();
    sk[2] = "F".getBytes();
    sk[3] = "H".getBytes();
    for (RegionInfo regionInfo : regionSet) {
        admin.multiSplitSync(regionInfo.getRegionName(), sk);
    }
    LOG.info("MultiSplit successfully.");
} catch (Exception e) {
    LOG.error("MultiSplit failed.");
} finally {
    if (table != null) {
        try {
            // Close table object
            table.close();
        } catch (IOException e) {
            LOG.error("Close table failed " ,e);
        }
    }
    if (admin != null) {
        try {
            // Close the Admin object.
            admin.close();
        } catch (IOException e) {
            LOG.error("Close admin failed " ,e);
        }
    }
}
LOG.info("Exiting testMultiSplit.");
}

```

Note that the division operations take effect on empty regions only.

### 1.10.3.1.17 Creating a Phoenix Table

#### Function

Phoenix can be installed on HBase to enable it to support SQL and JDBC APIs. This way, SQL users can access the HBase cluster.

#### Example Code

The following code snippet belongs to the **testCreateTable** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examplespackage**.

```

/**
 * Create Table
 */
public void testCreateTable() {
    LOG.info("Entering testCreateTable.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Create table
    String createTableSQL =
        "CREATE TABLE IF NOT EXISTS TEST (id integer not null primary key, name varchar, "
        + "account char(6), birth date)";

```

```
try (Connection conn = DriverManager.getConnection(url, props);
    Statement stat = conn.createStatement()) {
    // Execute Create SQL
    stat.executeUpdate(createTableSQL);
    LOG.info("Create table successfully.");
} catch (Exception e) {
    LOG.error("Create table failed.", e);
}
LOG.info("Exiting testCreateTable.");
}
/**
 * Drop Table
 */
public void testDrop() {
    LOG.info("Entering testDrop.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Delete table
    String dropTableSQL = "DROP TABLE TEST";

    try (Connection conn = DriverManager.getConnection(url, props);
        Statement stat = conn.createStatement()) {
        stat.executeUpdate(dropTableSQL);
        LOG.info("Drop successfully.");
    } catch (Exception e) {
        LOG.error("Drop failed.", e);
    }
    LOG.info("Exiting testDrop.");
}
```

### 1.10.3.1.18 Writing Data to the PhoenixTable

#### Function

The Phoenix table enables data writing in HBase.

#### Example Code

The following code snippet belongs to the **testPut** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
/*
 * Put data
 */
public void testPut() {
    LOG.info("Entering testPut.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Insert
    String upsertSQL =
        "UPSERT INTO TEST VALUES(1,'John','100000', TO_DATE('1980-01-01','yyyy-MM-dd'))";
    try (Connection conn = DriverManager.getConnection(url, props);
        Statement stat = conn.createStatement()){
        stat.executeUpdate(upsertSQL);
        conn.commit();
        LOG.info("Put successfully.");
    } catch (Exception e) {
        LOG.error("Put failed.", e);
    }
    LOG.info("Exiting testPut.");
}
```

### 1.10.3.1.19 Reading the PhoenixTable

#### Function

The Phoenix table enables data reading.

#### Example Code

The following code snippet belongs to the `testSelect` method in the `PhoenixSample` class of the `com.huawei.bigdata.hbase.examples` package.

```
/*
 * Select Data
 */
public void testSelect() {
    LOG.info("Entering testSelect.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Query
    String querySQL = "SELECT * FROM TEST WHERE id = ?";
    Connection conn = null;
    PreparedStatement preStat = null;
    Statement stat = null;
    ResultSet result = null;
    try {
        // Create Connection
        conn = DriverManager.getConnection(url, props);
        // Create Statement
        stat = conn.createStatement();
        // Create PrepareStatement
        preStat = conn.prepareStatement(querySQL);
        // Execute query
        preStat.setInt(1, 1);
        result = preStat.executeQuery();
        // Get result
        while (result.next()) {
            int id = result.getInt("id");
            String name = result.getString(1);
            System.out.println("id: " + id);
            System.out.println("name: " + name);
        }
        LOG.info("Select successfully.");
    } catch (Exception e) {
        LOG.error("Select failed.", e);
    } finally {
        if (null != result) {
            try {
                result.close();
            } catch (Exception e2) {
                LOG.error("Result close failed.", e2);
            }
        }
        if (null != stat) {
            try {
                stat.close();
            } catch (Exception e2) {
                LOG.error("Stat close failed.", e2);
            }
        }
        if (null != conn) {
            try {
                conn.close();
            } catch (Exception e2) {
                LOG.error("Connection close failed.", e2);
            }
        }
    }
}
```

```
    LOG.info("Exiting testSelect.");
}
```

### 1.10.3.1.20 Using HBase Dual-Read Capability

#### Scenario

The HBase client application loads the configuration items of the active and standby clusters by customization to implement the dual-read capability. HBase dual-read is a key feature that improves the high availability of the HBase cluster system. It applies to four query scenarios: reading data using **Get**, reading data in batches using **Get**, reading data using **Scan**, and querying data using a secondary index. HBase can read data from the active and standby clusters at the same time, reducing the query glitch time. The advantages are as follows:

- High success rate: The concurrent dual-read mechanism ensures a high success rate of read requests.
- High availability: When a single cluster is faulty, the query service is not interrupted. A short network jitter does not prolong the query time.
- High generality: The dual-read feature does not support dual-write, but does not affect the original real-time write scenario.
- Ease-of-use: Client encapsulation is performed, which is not sensed by services.

#### NOTE

Restrictions on HBase dual-read:

- The HBase dual-read feature is implemented based on replication. Data read from the standby cluster may be different from that from the active cluster. Therefore, only eventual consistency can be achieved.
- Currently, the HBase dual-read feature is used only for query. When the active cluster breaks down, the latest data cannot be synchronized. As a result, the latest data cannot be queried in the standby cluster.
- A **Scan** operation of HBase may be split into multiple RPC operations. Data may not be completely the same because related session information is not synchronized between different clusters. Therefore, the dual-read feature takes effect only when an RPC operation is performed for the first time. Requests before ResultScanner close access the cluster used for the first RPC operation.
- The HBase Admin API and real-time write API access only the active cluster. Therefore, after the active cluster breaks down, the Admin API and real-time write API are unavailable, and only the **Get** and **Scan** query services are available.

HBase dual-read supports the following two methods to configure the active/standby cluster:

- Add active/standby cluster configurations to the **hbase-dual.xml** file.
- Add the active/standby cluster configuration to **HBaseMultiClusterConnection**.

#### Add the Active/Standby Cluster Configuration to the hbase-dual.xml File

**Step 1** Save the keytab authentication files **user.keytab** and **krb5.conf** of the active cluster obtained when [Preparing a Developer Account](#) to the **src/main/resources/conf** secondary sample directory.

- Step 2** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the HBase active cluster and save them to the **src/main/resources/conf/active** directory. This directory needs to be created by yourself. For details, see [Preparing the Connection Cluster Configuration File](#).
- Step 3** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the standby cluster and save them to the **src/main/resources/conf/standby** directory. For details, see [Preparing the Connection Cluster Configuration File](#).
- Step 4** Create the **hbase-dual.xml** configuration file and save it to the **src/main/resources/conf/** directory. This directory needs to be created by yourself. For details about the configuration items in the configuration file, see [Table 1-83](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<!--Configuration file directory of the active cluster-->
<property>
    <name>hbase.dualclient.active.cluster.configuration.path</name>
    <value>{Sample code directory}\src\main\resources\active</value>
</property>
<!--Configuration file directory of the standby cluster-->
<property>
    <name>hbase.dualclient.standby.cluster.configuration.path</name>
    <value>{Sample code directory}\src\main\resources\standby</value>
</property>
<!--Connection implementation of the dual-read mode-->
<property>
    <name>hbase.client.connection.impl</name>
    <value>org.apache.hadoop.hbase.client.HBaseMultiClusterConnectionImpl</value>
</property>
<!--Security mode-->
<property>
    <name>hbase.security.authentication</name>
    <value>kerberos</value>
</property>
<!--Security mode-->
<property>
    <name>hadoop.security.authentication</name>
    <value>kerberos</value>
</property>
</configuration>
```

- Step 5** Create a dual-read Configuration and delete the comment of **testHBaseDualReadSample** from the **main** method of the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package. Ensure that the value of **IS\_CREATE\_CONNECTION\_BY\_XML** in the **HBaseDualReadSample** class in the **com.huawei.bigdata.hbase.examples** package is **true**.

**Step 6** Determining the data source cluster

- GET request. The following code snippet belongs to the **testGet** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
Result result = table.get(get);
if (result instanceof DualResult) {
    LOG.info(((DualResult)result).getClusterId());
}
```

- Scan request. The following code snippet belongs to the **testScanData** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
ResultScanner rScanner = table.getScanner(scan);
if (rScanner instanceof HBaseMultiScanner) {
    LOG.info(((HBaseMultiScanner)rScanner).getClusterId());
}
```

**Step 7** The client can print metric information.

Add the following content to the **log4j.properties** file so that the client can export metric information to the specified file: For details about the metrics, see [Printing Metric Information](#).

```
log4j.logger.DUAL=debug,DUAL
log4j.appendender.DUAL=org.apache.log4j.RollingFileAppender
log4j.appendender.DUAL.File=/var/log/dual.log //Local dual-read log path on the client. Change the value to
the actual directory, but ensure that the directory has the write permission.
log4j.additivity.DUAL=false
log4j.appendender.DUAL.MaxFileSize=${hbbase.log.maxfilesize}
log4j.appendender.DUAL.MaxBackupIndex=${hbbase.log.maxbackupindex}
log4j.appendender.DUAL.layout=org.apache.log4j.PatternLayout
log4j.appendender.DUAL.layout.ConversionPattern=%d{ISO8601} %--5p [%t] %c{2}: %m%n
```

----End

## Configure the Active/Standby Cluster Configuration in HBaseMultiClusterConnection

- Step 1** Save the keytab authentication files **user.keytab** and **krb5.conf** of the active cluster obtained when [Preparing a Developer Account](#) to the **src/main/resources/conf** secondary sample directory.
- Step 2** Create a dual-read Configuration and delete the comment of **testHBaseDualReadSample** from the **main** method of the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package. Ensure that the value of **IS\_CREATE\_CONNECTION\_BY\_XML** in the **HBaseDualReadSample** class in the **com.huawei.bigdata.hbase.examples** package is **false**.
- Step 3** Add related configurations to the **addHbaseDualXmlParam** method of the **HBaseDualReadSample** class. For details about related configuration items, see [HBase Dual-Read Configuration Items](#).

```
private void addHbaseDualXmlParam(Configuration conf) {
    // We need to set the optional parameters contained in hbase-dual.xml to conf
    // when we use configuration transfer solution
    conf.set(CONNECTION_IMPL_KEY, DUAL_READ_CONNECTION);
    // conf.set("", "");
}
```

- Step 4** Add configurations related to the Active cluster client to the **initActiveConf** method of the **HBaseDualReadSample** class.

```
private void initActiveConf() {
    // The hbase-dual.xml configuration scheme is used to generate the client configuration of the active
    // cluster.
    // In actual application development, you need to generate the client configuration of the active cluster.
    String activeDir =
        HBaseDualReadSample.class.getClassLoader().getResource(Utils.CONF_DIRECTORY).getPath()
            + File.separator + ACTIVE_DIRECTORY + File.separator;
    Configuration activeConf = Utils.createConfByUserDir(activeDir);
    HBaseMultiClusterConnection.setActiveConf(activeConf);
}
```

- Step 5** Add configurations related to the Standby cluster client to the **initStandbyConf** method of the **HBaseDualReadSample** class.

```
private void initStandbyConf() {
    // The hbase-dual.xml configuration scheme is used to generate the client configuration of the standby
    // cluster.
    // In actual application development, you need to generate the client configuration of the standby cluster.
    String standbyDir =
        HBaseDualReadSample.class.getClassLoader().getResource(Utils.CONF_DIRECTORY).getPath()
```

```
+ File.separator + STANDBY_DIRECTORY + File.separator;
Configuration standbyConf = Utils.createConfByUserDir(standbyDir);
HBaseMultiClusterConnection.setStandbyConf(standbyConf);
}
```

**Step 6** Determining the data source cluster.

- GET request. The following code snippet belongs to the **testGet** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
Result result = table.get(get);
if (result instanceof DualResult) {
    LOG.info(((DualResult)result).getClusterId());
}
```
- Scan request. The following code snippet belongs to the **testScanData** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
ResultScanner rScanner = table.getScanner(scan);
if (rScanner instanceof HBaseMultiScanner) {
    LOG.info(((HBaseMultiScanner)rScanner).getClusterId());
}
```

**Step 7** The client can print metric information.

Add the following content to the **log4j.properties** file so that the client can export metric information to the specified file: For details about the metrics, see [Printing Metric Information](#)

```
log4j.logger.DUAL=debug,DUAL
log4j.appendер.DUAL=org.apache.log4j.RollingFileAppender
log4j.appendер.DUAL.File=/var/log/dual.log //Local dual-read log path on the client. Change the value to
the actual directory, but ensure that the directory has the write permission.
log4j.additivity.DUAL=false
log4j.appendер.DUAL.MaxFileSize=${hbse.log.maxfilesize}
log4j.appendер.DUAL.MaxBackupIndex=${hbse.log.maxbackupindex}
log4j.appendер.DUAL.layout=org.apache.log4j.PatternLayout
log4j.appendер.DUAL.layout.ConversionPattern=%d{ISO8601} %-5p [%t] %c{2}: %m%n
```

----End

### 1.10.3.1.21 Configuring Log4j Log Output

#### Function Description

This section describes how to output HBase client logs to a specified log file separately from service logs to facilitate HBase problem analyzing and locating.

If the **log4j** configuration exists in the process, copy the RFA and RFAS configurations in **hbse-example\src\main\resources\log4j.properties** to the existing **log4j** configuration.

#### Sample Code

```
hbse.root.logger=INFO,console,RFA          //HBase client log output configuration. console: outputs to
the console; RFA: outputs to the log files.
hbse.security.logger=DEBUG,console,RFAS     //HBase client security logs output configuration. console:
outputs to the console; RFAS: outputs the log files.
hbse.log.dir=/var/log/Bigdata/hbase/client/ //Log directory. Modify based on the actual directory. Ensure
that the directory has the write permission.
hbse.log.file=hbse-client.log               //Log file name
hbse.log.level=INFO                         //Log level. If detailed logs are required for fault locating, change it
to DEBUG. The modification takes effect after restart.
hbse.log.maxbackupindex=20                   //Maximum number of log files that can be saved.
# Security audit appender
hbse.security.log.file=hbse-client-audit.log //Command of the audit log file
```

## 1.10.3.2 HBase Global Secondary Index Sample Program

### 1.10.3.2.1 Service Scenario Description

HBase allows you to use global secondary indexes to accelerate conditional queries. This sample shows you how to manage and use global secondary indexes.

#### Scenario

Assume that you are developing an application that records user information and addresses. The following table lists the data to be recorded.

**Table 1-82** User information

<b>id</b>	<b>name</b>	<b>age</b>	<b>address</b>
1	Zhang	20	CityA
2	Li	30	CityB
3	Wang	35	CityC

#### Data Preparation

Proper design of a table structure, RowKeys, and column names enable you to make full use of HBase advantages. If you use a global secondary index in a scan condition query, the query is performed on an index table. You do not need to pay attention to the rowkeys of the user table. In this example, the rowkeys are in "r1", "r2", "r3"... format. All columns are stored in the **info** column family.

#### Sample Description

The examples in following content describe how to create and delete global secondary indexes, modify statuses, , and use them to accelerate queries.

### 1.10.3.2.2 Creating HBase Global Secondary Indexes

#### Function

Manage HBase global secondary indexes by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin**. **addIndices** is used for creating global secondary indexes.

To create a global secondary index, you need to configure the index column, covering column (optional), and pre-created index table regions (recommended).

### NOTE

To create global secondary indexes on a table with existing data, you need to pre-create regions for the index table to prevent hot-spotting. The rowkeys of the index table data consists of index columns and separators. The format is `\x01/index value\x00`. Specify the format when you pre-create a region, for example, when the **id** and **age** columns are used as the index columns, both columns need to be integers. Use the **id** column to pre-create regions. You can specify the pre-created index table region as follows:

`\x010,\x011,\x012....`

## Code Sample

The following code snippets are in the **addIndices** method in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, an index named **index\_id\_age** is created for the **user\_table** data table. The **id** and **age** columns in the data are used as index columns and the **name** column is overwritten. (The query condition is not used, but the query result needs to return to this column).

```
/**  
 * createIndex  
 */  
public void testCreateIndex() {  
    LOG.info("Entering createIndex.");  
    // Create index instance  
    TableIndices tableIndices = new TableIndices();  
    // Create index spec  
    // idx_id_age covered info:name  
    HIndexSpecification indexSpec = new HIndexSpecification("idx_id_age");  
  
    // Set index column  
    indexSpec.addIndexColumn(Bytes.toBytes("info"), Bytes.toBytes("id"), ValueType.STRING);  
    indexSpec.addIndexColumn(Bytes.toBytes("info"), Bytes.toBytes("age"), ValueType.STRING);  
  
    // Set covered column  
    // If you want cover one column, use addCoveredColumn  
    // If you want cover all column in one column family, use addCoveredFamilies  
    // If you want cover all column of all column family, use setCoveredAllColumns  
    indexSpec.addCoveredColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));  
  
    // Need specify index table split keys, it should specify by index column.  
    // Note: index data's row key has a same prefix "\x01"  
    // For example:  
    // Our index column include "id" and "age", "id" is a number, we  
    // could specify split key like \x010 \x011 \x012...  
    byte[][] splitKeys = new byte[10][];  
    for (int i = 0; i < 10; i++) {  
        splitKeys[i] = Bytes.toBytesBinary("\x01" + i);  
    }  
    indexSpec.setSplitKeys(splitKeys);  
    tableIndices.addIndex(indexSpec);  
  
    // iAdmin will close the inner admin instance  
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient newIndexAdmin(conn.getAdmin())) {  
        // add index to the table  
        iAdmin.addIndices(tableName, tableIndices);  
        LOG.info("Create index successfully.");  
    } catch (IOException e) {  
        LOG.error("Create index failed.", e);  
    }  
    LOG.info("Exiting createIndex.");  
}
```

### 1.10.3.2.3 Querying Global Secondary Indexes

#### Function

Manage HBase global secondary index by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin.listIndices** is used for querying indexes, including the definitions and status of all indexes related to the current user table.

#### Code Sample

The following code snippets are in the **listIndices** method in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, all indexes corresponding to the user table **user\_table** are queried.

```
/*
 * List indexes
 */
public void testListIndexes() {
    LOG.info("Entering testListIndexes.");
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient.newIndexAdmin(conn.getAdmin())) {
        for (Pair<HIndexSpecification, IndexState> indexPair : iAdmin.listIndices(tableName)) {
            LOG.info("index spec:{} , index state:{}" , indexPair.getFirst(), indexPair.getSecond());
        }
        LOG.info("List indexes successfully.");
    } catch (IOException e) {
        LOG.error("List indexes failed." , e);
    }
    LOG.info("Exiting testListIndexes.");
}
```

### 1.10.3.2.4 Querying Based on Global Secondary Indexes

#### Function

When a user table with a global secondary index is used for query, the query can be converted to a range query of the index table. The query performance is higher than that of a user table without a secondary index. For details about global secondary indexes, see

"Introduction to Global Secondary Indexes" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).

#### Code Sample

The following code snippets are in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, the values of **id**, **age**, and **name** of the specified **id** are queried, and the **idx\_id\_age** index is selected. The query results are completely overwritten. As a result, you do not need to query in the original table to achieve optimal query performance.

```
/*
 * Scan data by secondary index.
```

```

/*
public void testScanDataByIndex() {
    LOG.info("Entering testScanDataByIndex.");

    Scan scan = new Scan();
    // Create a filter for indexed column.
    SingleColumnValueFilter filter = new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("id"),
        CompareOperator.EQUAL, Bytes.toBytes("3"));
    filter.setFilterIfMissing(true);
    scan.setFilter(filter);

    // Specify returned columns
    // If returned columns not included in index table, will query back user table,
    // it's not the fast way to get data, suggest to cover all needed columns.
    // If you want to confirm whether using index for scanning, please set hbase client log level to DEBUG.
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("id"));
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("age"));
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));

    LOG.info("Scan indexed data.");
    try (Table table = conn.getTable(tableName); ResultScanner scanner = table.getScanner(scan)) {
        for (Result result : scanner) {
            for (Cell cell : result.rawCells()) {
                LOG.info("{}:{}:{},{}:{}:{}",
                    Bytes.toString(CellUtil.cloneRow(cell)),
                    Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
                    Bytes.toString(CellUtil.cloneValue(cell)));
            }
        }
        LOG.info("Scan data by index successfully.");
    } catch (IOException e) {
        LOG.error("Scan data by index failed ", e);
    }
    LOG.info("Exiting testScanDataByIndex.");
}

```

### 1.10.3.2.5 Disabling Global Secondary Indexes

#### Function

The status of a global secondary index determines whether the index is valid. By modifying the index status, you can disable, enable, or discard an index (no index data will be generated). You can modify the index status by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin**. For details, see "Introduction to Global Secondary Index APIs" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).

#### Code Sample

The following code snippets are in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, the **idx\_id\_age** index is disabled. That is, the index is not used during query, but index data is generated.

```

/**
 * alter index to UNUSABLE state.
 */
public void testAlterIndex() {
    LOG.info("Entering testAlterIndex.");
    List<String> indexNameList = Lists.newArrayList("idx_id_age");
    // Instantiate HIndexAdmin Object
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient newIndexAdmin(conn.getAdmin())) {
        iAdmin.alterGlobalIndicesUnusable(tableName, indexNameList);
    }
}

```

```
        LOG.info("Alter indices to UNUSABLE successfully.");
    } catch (IOException e) {
        LOG.error("Alter indices to UNUSABLE failed.", e);
    }
    LOG.info("Exiting testAlterIndex.");
}
```

### 1.10.3.2.6 Deleting Global Secondary Indexes

#### Function

Manage HBase global secondary indexes by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin**.

**dropIndices** is used to delete indexes.

#### Code Sample

The following code snippets are in the **dropIndices** method in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, the **idx\_id\_age** index is deleted from the **user\_table** table.

```
/*
 * dropIndex
 */
public void testDropIndex() {
    LOG.info("Entering testDropIndex.");
    List<String> indexNameList = Lists.newArrayList("idx_id_age");
    // Instantiate HIndexAdmin Object
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient newIndexAdmin(conn.getAdmin())) {
        // Delete Secondary Index
        iAdmin.dropIndices(tableName, indexNameList);
        LOG.info("Drop index successfully.");
    } catch (IOException e) {
        LOG.error("Drop index failed ", e);
    }
    LOG.info("Exiting testDropIndex.");
}
```

### 1.10.3.3 Sample Program for Preloading Meta Tables When Request Concurrency Is High

#### 1.10.3.3.1 Service Scenario Description

When there is a large number of concurrent access requests to a data table, the client caches the required region location information again after an application is restarted. Each thread waits until the cache is complete before executing the requests. If the client timeout is too short and the target table contains a large number of regions, a large number of requests will be retried due to timeout. To prevent this issue, you need to cache the region information in the table before the application threads run so that subsequent requests will not time out.

This sample contains HBase multi-thread read/write tasks. The meta table will be preloaded before the thread tasks are started.

 NOTE

In this sample, multiple threads concurrently access HBase, where:

- The number of regions of the HBase table is greater than or equal to 100.
- The data table is concurrently accessed by a single application.
- The HBase request timeout on the application side is less than 2s.

### 1.10.3.3.2 Preloading Meta Tables When Request Concurrency Is High

#### Function

Preload region location information in the scenario where multiple threads concurrently read and write HBase tables.

#### Code Sample

The following code snippets are in the **MultiThreadSample** class of the **com.huawei.bigdata.hbase.examples** package.

- Create a user table, pre-split 300 regions, and compress the table using SNAPPY.

```
private void createUserTable() {
    // Use number id as the first part of rowKey and pre-split 300 regions by the decimal string split
    // algorithm.
    ColumnFamilyDescriptor cf = ColumnFamilyDescriptorBuilder
        .newBuilder(Bytes.toBytes("f"))
        .setCompressionType(Compression.Algorithm.SNAPPY)
        .build();
    TableDescriptor td = TableDescriptorBuilder
        .newBuilder(tableName)
        .setColumnFamily(cf)
        .build();
    try (Admin admin = conn.getAdmin()) {
        if (!admin.tableExists(tableName)) {
            RegionSplitter.SplitAlgorithm splitAlgo = RegionSplitter
                .newSplitAlgoInstance(this.conn.getConfiguration(),
                    RegionSplitter.DecimalStringSplit.class.getName());
            admin.createTable(td, splitAlgo.split(REGION_NUMS));
            LOG.info("Create table success.");
        } else {
            LOG.warn("Table already exists.");
        }
    } catch (IOException e) {
        LOG.error("Create table failed!", e);
    }
}
```

- Preload the meta table and obtain the location information of all regions in the user table.

```
private void preLoadTableRegionLocations() {
    try (RegionLocator rl = conn.getRegionLocator(tableName)) {
        long startTime = System.currentTimeMillis();
        rl.getAllRegionLocations();
        LOG.info("Cache table region location success, cost {} ms.", System.currentTimeMillis() -
            startTime);
    } catch (IOException e) {
        LOG.error("Cache table region location failed!", e);
    }
}
```

- Perform multi-thread read and write operations.

```
private void doOperations() {
    int cores = Math.max(32, Runtime.getRuntime().availableProcessors());
```

```
// Pool for user service including HBase data read/write.
ThreadPoolExecutor pool = new ThreadPoolExecutor(cores, cores,
    60,
    TimeUnit.SECONDS,
    new LinkedBlockingQueue<>(cores * 100),
    new ThreadFactoryBuilder()
        .setDaemon(true)
        .setUncaughtExceptionHandler((t, e) ->
            LOG.warn("Thread:{} exited with Exception:{}", t, StringUtils.stringifyException(e)))
        .setNameFormat("Application-Pool-%d").build());

for (int i = 0; i < OPERATIONS_COUNT; i++) {
    pool.execute(() -> {
        try (Table table = conn.getTable(tableName)) {
            Put put = new Put(Bytes.toBytes(
                ThreadLocalRandom.current().nextLong(0, 100000L) + "#"
                    + RandomStringUtils.randomAlphabetic(5)));
            put.addColumn(Bytes.toBytes("f"), Bytes.toBytes("name"),
                Bytes.toBytes(RandomStringUtils.randomAlphabetic(10)));
            put.addColumn(Bytes.toBytes("f"), Bytes.toBytes("ts"),
                Bytes.toBytes(Long.toString(System.currentTimeMillis())));
            table.put(put);
        } catch (IOException e) {
            LOG.error("Put data failed ", e);
        }
    });
    pool.execute(() -> {
        Scan scan = new Scan()
            .withStartRow(Bytes.toBytes(
                ThreadLocalRandom.current().nextLong(0, 100000L) + "#"))
            .setLimit(1);
        try (Table table = conn.getTable(tableName);
            ResultScanner scanner = table.getScanner(scan)) {
            Result result;
            while ((result = scanner.next()) != null) {
                for (Cell cell : result.rawCells()) {
                    LOG.info("ROW:{};CF:{};CQ:{};VALUE:{}",
                        Bytes.toString(CellUtil.cloneRow(cell)),
                        Bytes.toString(CellUtil.cloneFamily(cell)),
                        Bytes.toString(CellUtil.cloneQualifier(cell)),
                        Bytes.toString(CellUtil.cloneValue(cell)));
                }
            }
        } catch (IOException e) {
            LOG.error("Scan data failed ", e);
        }
    });
}
shutDownPool(pool);
LOG.info("Finish do operations.");
}

private void shutDownPool(ThreadPoolExecutor pool) {
    pool.shutdown();
    try {
        if (!pool.awaitTermination(SHUT_DOWN_POOL_WAIT_TIMEOUT_SEC, TimeUnit.SECONDS)) {
            pool.shutdownNow();
        }
    } catch (InterruptedException e) {
        pool.shutdownNow();
    }
}
```

- Clear the user table.

```
private void dropUserTable() {
    try (Admin admin = conn.getAdmin()) {
        admin.disableTable(tableName);
        admin.deleteTable(tableName);
        LOG.info("Drop table success.");
    } catch (IOException e) {
        LOG.error("Drop table failed ", e);
    }
}
```

```
    }
```

### 1.10.3.4 HBase Rest API Invoking Sample Program

#### 1.10.3.4.1 Querying Cluster Information Using REST

##### Function

Use the REST service to transfer the URL consisting of the host and port to obtain the cluster version and status information through HTTPS.

##### Example Code

- Obtaining the cluster version information

The following code snippets are in the **getClusterVersion** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterVersion(String url) {
    String endpoint = "/version/cluster";
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
    handleNormalResult((Optional<ResultModel>) result);
}
```

- Obtaining the cluster status information

The following code snippets are in the **getClusterStatus** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterStatus(String url) {
    String endpoint = "/status/cluster";
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
    handleNormalResult(result);
}
```

#### 1.10.3.4.2 Obtaining All Tables Using REST

##### Function

Use the REST service and transfer the URL consisting of the host and port to obtain all tables using HTTPS.

##### Example Code

The following code snippets are in the **getAllUserTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllUserTables(String url) {
    String endpoint = "/";
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
    handleNormalResult(result);
}
```

### 1.10.3.4.3 Operate Namespaces Using REST

#### Function

Use the REST service to import the URL consisting of the host and port and the specified namespace, Use HTTPS to create, query, and delete namespaces, and obtain tables in the specified namespace.

#### ⚠ CAUTION

HBase tables are stored in *Namespace.Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table namespace.

#### Example Code

- Invoking methods

```
// Namespace operations.  
createNamespace(url, "testNs");  
getAllNamespace(url);  
deleteNamespace(url, "testNs");  
getAllNamespaceTables(url, "default");
```

- Creating a namespace

The following code snippets are in the **createNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createNamespace(String url, String namespace) {  
    String endpoint = "/namespaces/" + namespace;  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.POST, null);  
    if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_CREATED) {  
        LOG.info("Create namespace '{}' success.", namespace);  
    } else {  
        LOG.error("Create namespace '{}' failed.", namespace);  
    }  
}
```

- Querying all namespaces

The following code snippets are in the **getAllNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllNamespace(String url) {  
    String endpoint = "/namespaces";  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);  
    handleNormalResult(result);  
}
```

- Deleting a specified namespace

The following code snippets are in the **deleteNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteNamespace(String url, String namespace) {  
    String endpoint = "/namespaces/" + namespace;  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.DELETE, null);  
    if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_OK) {  
        LOG.info("Delete namespace '{}' success.", namespace);  
    } else {  
        LOG.error("Delete namespace '{}' failed.", namespace);  
    }  
}
```

```
        LOG.error("Delete namespace '{}' failed.", namespace);
    }
}
```

- Obtain tables in a specified namespace.

The following code snippets are in the **getAllNamespaceTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllNamespaceTables(String url, String namespace) {
    String endpoint = "/namespaces/" + namespace + "/tables";
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
    handleNormalResult(result);
}
```

#### 1.10.3.4.4 Operate Tables Using REST

##### Function

Use the REST service to transfer the URL consisting of the host and port as well as the specified **tableName** and **jsonHTD** to query, modify, create, and delete table information through HTTPS.

##### Example Code

- Invoking methods

```
// Add a table with specified info.
createTable(url, "testRest",
    "{\"name\":\"default:testRest\",\"ColumnSchema\":[{\"name\":\"cf1\"},\"cf2\"]}");

// Add column family 'testCF1' if not exist, else update the 'VERSIONS' to 3.
// Notes: The unspecified property of this column family will be updated to default value.
modifyTable(url, "testRest",
    "{\"name\":\"testRest\",\"ColumnSchema\":[{\"name\":\"testCF1\",\"VERSIONS\":3}]}");

// Describe specific Table.
descTable(url, "default:testRest");

// delete a table with specified info.
deleteTable(url, "default:testRest",
    "{\"name\":\"default:testRest\",\"ColumnSchema\":[{\"name\":\"testCF1\",\"VERSIONS\":3}]");
```

- Querying table information

The following code snippets are in the **descTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void descTable(String url, String tableName) {
    String endpoint = "/" + tableName + "/schema";
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
    handleNormalResult((Optional<ResultModel>) result);
}
```

- Modifying table information

The following code snippets are in the **modifyTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void modifyTable(String url, String tableName, String jsonHTD) {
    LOG.info("Start modify table.");
    String endpoint = "/" + tableName + "/schema";
    JsonElement tableDesc = new JsonParser().parse(jsonHTD);
```

```
// Add a new column family or modify it.  
handleNormalResult(sendAction(url + endpoint, MethodType.POST, tableDesc));  
}
```

- Creating a table

The following code snippets are in the **createTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(String url, String tableName, String jsonHTD) {  
    LOG.info("Start create table.");  
    String endpoint = "/" + tableName + "/schema";  
    JsonElement tableDesc = new JsonParser().parse(jsonHTD);  
  
    // Add a table.  
    handleCreateTableResult(sendAction(url + endpoint, MethodType.PUT, tableDesc));  
}
```

- Deleting a table

The following code snippets are in the **deleteTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteTable(String url, String tableName, String jsonHTD) {  
    LOG.info("Start delete table.");  
    String endpoint = "/" + tableName + "/schema";  
    JsonElement tableDesc = new JsonParser().parse(jsonHTD);  
  
    // delete a table.  
    handleNormalResult(sendAction(url + endpoint, MethodType.DELETE, tableDesc));  
}
```

### 1.10.3.5 Accessing the HBase ThriftServer Sample Program

#### 1.10.3.5.1 Accessing the ThriftServer Operation Table

##### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access ThriftServer, and obtain table names, create a table, and delete a table based on the specified namespace.

##### Example Code

- Invoking methods

```
// Get table of specified namespace. getTableNamesByNamespace(client, "default");  
// Create table. createTable(client, TABLE_NAME);  
// Delete specified table.  
deleteTable(client, TABLE_NAME);
```

- Obtains table names based on the specified namespace.

The following code snippets are in the **getTableNamesByNamespace** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getTableNamesByNamespace(THBaseService.Iface client, String namespace) throws  
TException {  
    client.getTableNamesByNamespace(namespace)  
        .forEach(  
            tTableName -> LOGGER.info("{}: {}, TableName.valueOf(tTableName.getNs(),  
tTableName.getQualifier())));  
}
```

- Creating a table

The following code snippets are in the **createTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(THBaseService.Iface client, String tableName) throws TException, IOException {
    TTableName table = getTableName(tableName);
    TTableDescriptor descriptor = new TTableDescriptor(table);
    descriptor.setColumns(
        Collections.singletonList(new
        TColumnFamilyDescriptor(). setName(COLUMN_FAMILY.getBytes())));
    if (client.tableExists(table)) {
        LOGGER.warn("Table {} is exists, delete it.", tableName);
        client.disableTable(table);
        client.deleteTable(table);
    }
    client.createTable(descriptor, null);
    if (client.tableExists(table)) {
        LOGGER.info("Created {}.", tableName);
    } else {
        LOGGER.error("Create {} failed.", tableName);
    }
}
```

- Deleting a table

The following code snippets are in the **deleteTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteTable(THBaseService.Iface client, String tableName) throws TException, IOException {
    TTableName table = getTableName(tableName);
    if (client.tableExists(table)) {
        client.disableTable(table);
        client.deleteTable(table);
        LOGGER.info("Deleted {}.", tableName);
    } else {
        LOGGER.warn("{} not exist.", tableName);
    }
}
```

### 1.10.3.5.2 Accessing ThriftServer to Write Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use Put and putMultiple to write data.

#### Example Code

- Invoking methods

```
// Write data with put.
putData(client, TABLE_NAME);

// Write data with putlist.
putDataList(client, TABLE_NAME);
```

- Using Put to write data.

The following code snippets are in the **putData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void putData(THBaseService.Iface client, String tableName) throws TException {
    LOGGER.info("Test putData.");
    TPut put = new TPut();
    put.setRow("row1".getBytes());

    TColumnValue columnValue = new TColumnValue();
    columnValue.setFamily(COLUMN_FAMILY.getBytes());
    columnValue.setQualifier("q1".getBytes());
    columnValue.setValue("test value".getBytes());
    List<TColumnValue> columnValues = new ArrayList<>(1);
    columnValues.add(columnValue);
    put.setColumnValues(columnValues);

    ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
    client.put(table, put);
    LOGGER.info("Test putData done.");
}
```

- Using `putMultiple` to write data.

The following code snippets are in the `putDataList` method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void putDataList(THBaseService.Iface client, String tableName) throws TException {
    LOGGER.info("Test putDataList.");
    TPut put1 = new TPut();
    put1.setRow("row2".getBytes());
    List<TPut> putList = new ArrayList<>();

    TColumnValue q1Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
        ByteBuffer.wrap("q1".getBytes()), ByteBuffer.wrap("test value".getBytes()));
    TColumnValue q2Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
        ByteBuffer.wrap("q2".getBytes()), ByteBuffer.wrap("test q2 value".getBytes()));
    List<TColumnValue> columnValues = new ArrayList<>(2);
    columnValues.add(q1Value);
    columnValues.add(q2Value);
    put1.setColumnValues(columnValues);
    putList.add(put1);

    TPut put2 = new TPut();
    put2.setRow("row3".getBytes());

    TColumnValue columnValue = new TColumnValue();
    columnValue.setFamily(COLUMN_FAMILY.getBytes());
    columnValue.setQualifier("q1".getBytes());
    columnValue.setValue("test q1 value".getBytes());
    put2.setColumnValues(Collections.singletonList(columnValue));
    putList.add(put2);

    ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
    client.putMultiple(table, putList);
    LOGGER.info("Test putDataList done.");
}
```

### 1.10.3.5.3 Accessing ThriftServer to Read Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use **get** and **scan** methods to read data.

## Example Code

- Invoking methods

```
// Get data with single get.  
getData(client, TABLE_NAME);  
  
// Get data with getlist.  
getDataList(client, TABLE_NAME);  
  
// Scan data.  
scanData(client, TABLE_NAME);
```

- Using the **get** method to write data.

The following code snippets are in the **getData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getData(THBaseService.Iface client, String tableName) throws TException {  
    LOGGER.info("Test getData.");  
    TGet get = new TGet();  
    get.setRow("row1".getBytes());  
  
    ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());  
    TResult result = client.get(table, get);  
    printResult(result);  
    LOGGER.info("Test getData done.");  
}
```

- Using the **getlist** method to write data.

The following code snippets are in the **getDataList** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getDataList(THBaseService.Iface client, String tableName) throws TException {  
    LOGGER.info("Test getDataList.");  
    List<TGet> getList = new ArrayList<>();  
    TGet get1 = new TGet();  
    get1.setRow("row1".getBytes());  
    getList.add(get1);  
  
    TGet get2 = new TGet();  
    get2.setRow("row2".getBytes());  
    getList.add(get2);  
  
    ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());  
    List<TResult> resultList = client.getMultiple(table, getList);  
    for (TResult tResult : resultList) {  
        printResult(tResult);  
    }  
    LOGGER.info("Test getDataList done.");  
}
```

- Using the **scan** method to write data.

The following code snippets are in the **scanData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void scanData(THBaseService.Iface client, String tableName) throws TException {  
    LOGGER.info("Test scanData.");  
    int scannerId = -1;  
    try {  
        ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());  
        TScan scan = new TScan();  
        scan.setLimit(500);  
        scannerId = client.openScanner(table, scan);  
        List<TResult> resultList = client.getScannerRows(scannerId, 100);  
        while (resultList != null && !resultList.isEmpty()) {  
            for (TResult tResult : resultList) {
```

```
        printResult(tResult);
    }
    resultList = client.getScannerRows(scannerId, 100);
}
} finally {
    if (scannerId != -1) {
        client.closeScanner(scannerId);
        LOGGER.info("Closed scanner {}.", scannerId);
    }
}
LOGGER.info("Test scanData done.");
}
```

## 1.10.3.6 Sample Program for HBase to Access Multiple ZooKeepers

### 1.10.3.6.1 Accessing Multiple ZooKeepers

#### Function

This function allows simultaneous access to FusionInsight ZooKeeper from the HBase client and third-party ZooKeeper from the customer application in the same client process.

#### Example code

The following code snippet is in the **TestZKSample** class of the **hbase-zk-example** **\src\main\java\com\huawei\hadoop\hbase\example**. You need to pay attention to the **login** and **connectApacheZK** methods.

```
private static void login(String keytabFile, String principal) throws IOException {
    conf = HBaseConfiguration.create();
    //In Windows environment
    String confDirPath = TestZKSample.class.getClassLoader().getResource("").getPath() + File.separator;
[1]    //In Linux environment
    //String confDirPath = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

    // Set zoo.cfg for hbase to connect to fi zookeeper.
    conf.set("hbase.client.zookeeper.config.path", confDirPath + "zoo.cfg");
    if (User.isHBaseSecurityEnabled(conf)) {
        // jaas.conf file, it is included in the client package
        System.setProperty("java.security.auth.login.config", confDirPath + "jaas.conf");
        // set the kerberos server info, point to the kerberosclient
        System.setProperty("java.security.krb5.conf", confDirPath + "krb5.conf");
        // set the keytab file name
        conf.set("username.client.keytab.file", confDirPath + keytabFile);
        // set the user's principal
        try {
            conf.set("username.client.kerberos.principal", principal);
            User.login(conf, "username.client.keytab.file", "username.client.kerberos.principal",
                InetAddress.getLocalHost().getCanonicalHostName());
        } catch (IOException e) {
            throw new IOException("Login failed.", e);
        }
    }
}
private void connectApacheZK() throws IOException, org.apache.zookeeper.KeeperException {
    try {
        // Create apache zookeeper connection.
        ZooKeeper digestZk = new ZooKeeper("127.0.0.1:24002", 60000, null);
        LOG.info("digest directory: {}", digestZk.getChildren("/", null));
        LOG.info("Successfully connect to apache zookeeper.");
    } catch (InterruptedException e) {
        LOG.error("Found error when connect apache zookeeper ", e);
    }
}
```

```
}
```

- [1] **userdir** obtains the **conf** directory in the resource path after compilation. Save the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** configuration files required for initialization and the user credential file used for security authentication to the **src/main/resources** directory.
- The **jaas.conf** file specified by the **java.security.auth.login.config** parameter in the **login** method is used to set the authentication information for accessing ZooKeeper. The example code contains the Client\_new and Client configurations. The Client\_new configuration is used to access FusionInsight ZooKeeper and the Client configuration is used to access Apache ZooKeeper.
- The **hbase.client.zookeeper.config.path** parameter in the **login** method controls the access to the FusionInsight ZooKeeper client. The following parameters are involved:
  - **zookeeper.sasl.clientconfig**: Specifies the configuration in the **jaas.conf** file used for accessing FusionInsight ZooKeeper.
  - **zookeeper.server.principal**: Specifies the principle of the ZooKeeper server. The format is **zookeeper/hadoop.System domain name**, for example, **zookeeper/hadoop.HADOOP.COM**. To obtain the system domain name, log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and view the value of Local Domain.
  - **zookeeper.sasl.client**: If the MRS cluster works in the security mode, set this parameter to **true**. Otherwise, set this parameter to **false**. If this parameter is set to **false**, the **zookeeper.sasl.clientconfig** and **zookeeper.server.principal** parameters do not take effect.

## 1.10.4 Application Commissioning

### 1.10.4.1 Commissioning an Application in Windows

#### 1.10.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

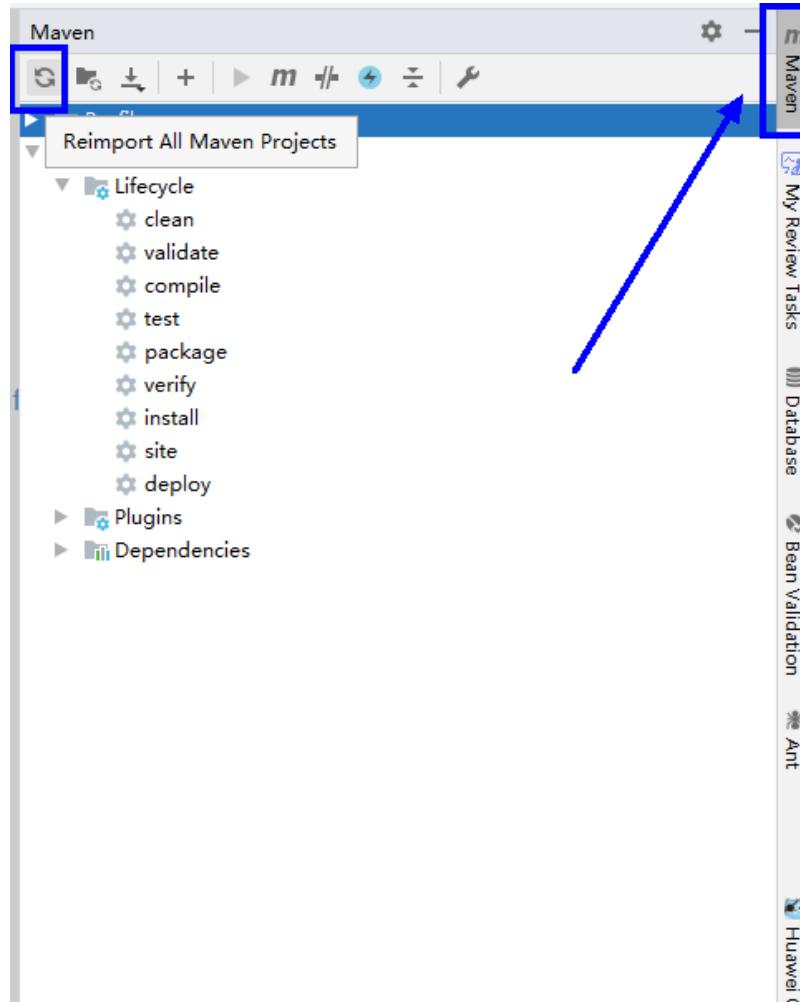


- NOTE**
- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
  - You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

## Procedure

**Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

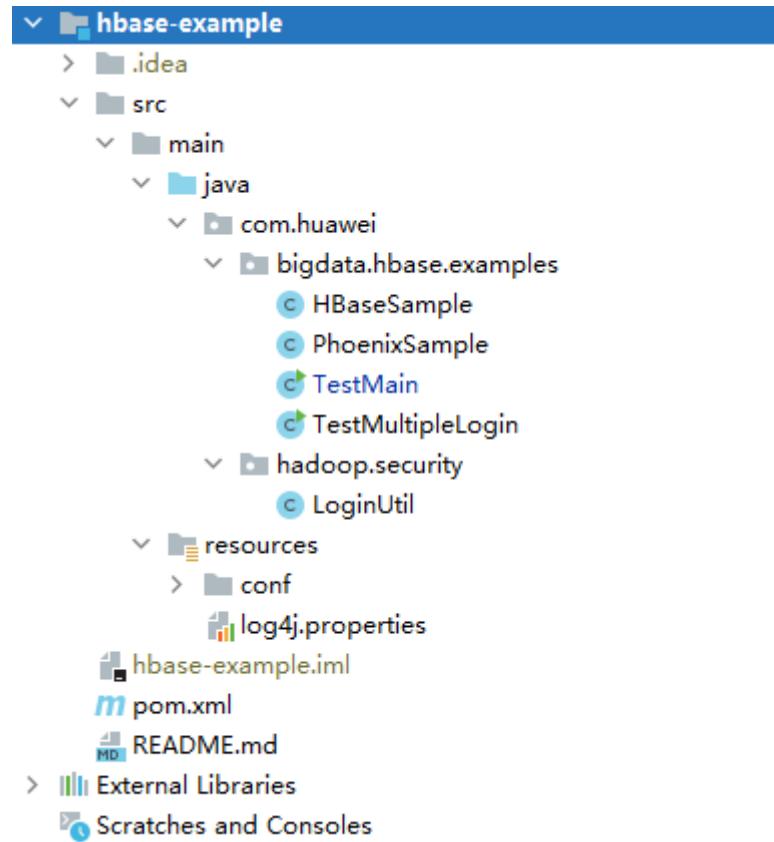
**Figure 1-127** reimport projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user.  
See [Figure 1-128](#).

Figure 1-128 Directory list of **hbase-example** to be compiled



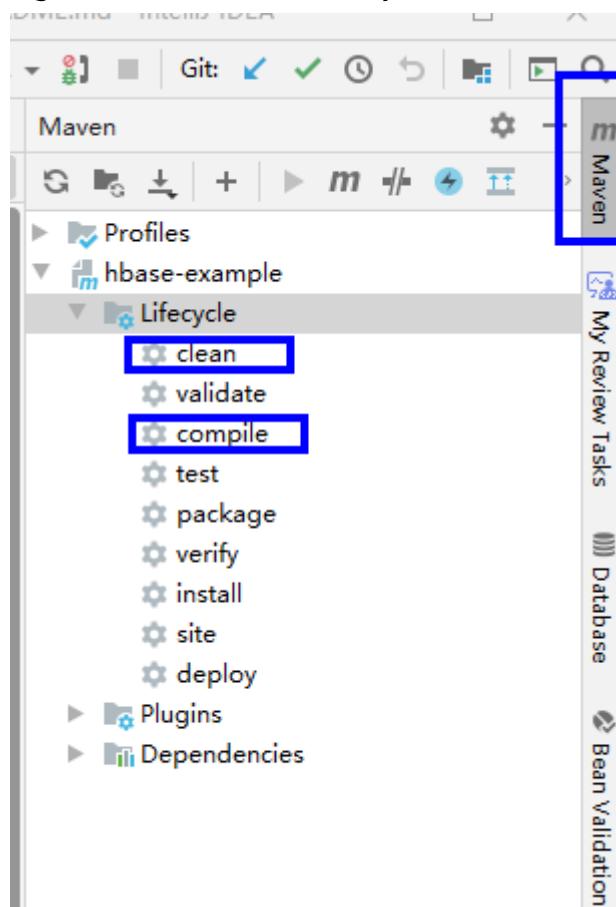
1. Compile an application.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

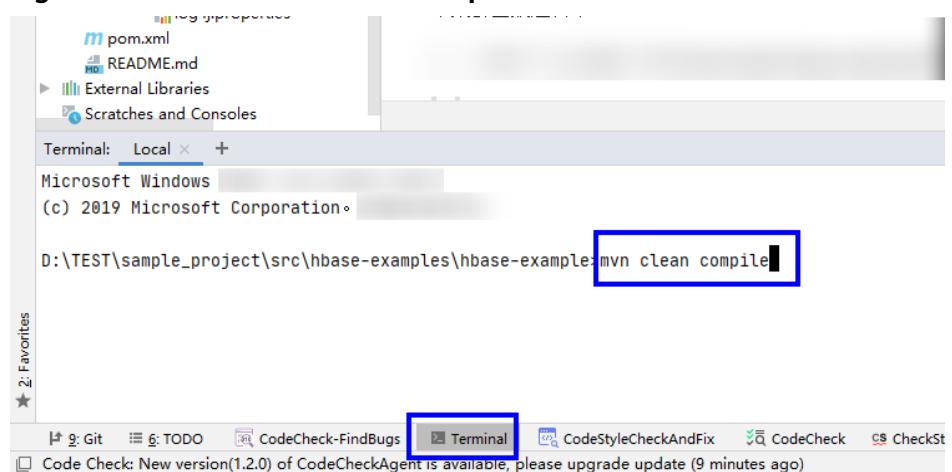
Figure 1-129 clean and compile tools of Maven



- Method 2:

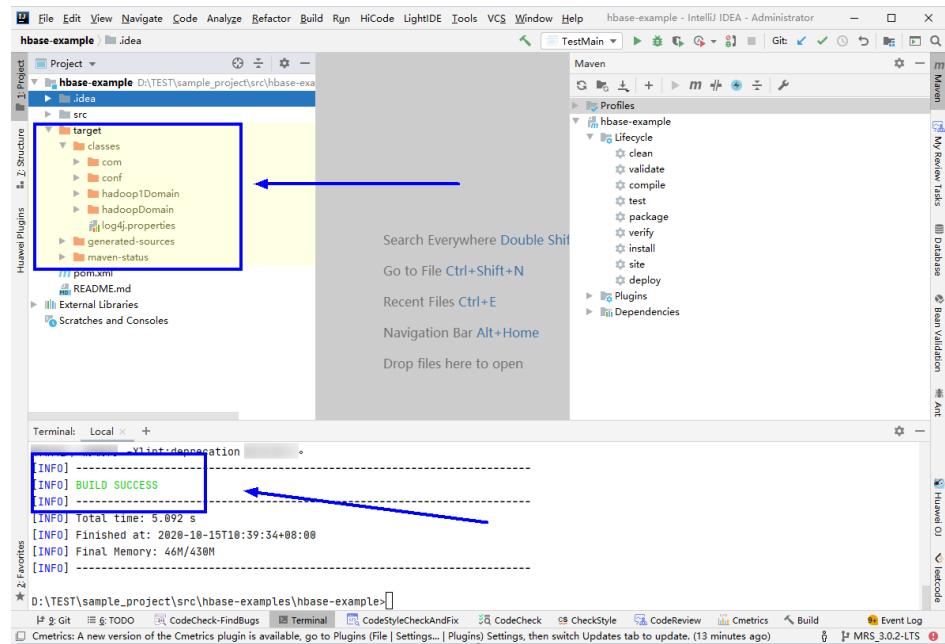
Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 1-130 Enter mvn clean compile in the IDEA Terminal text box.



2. After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

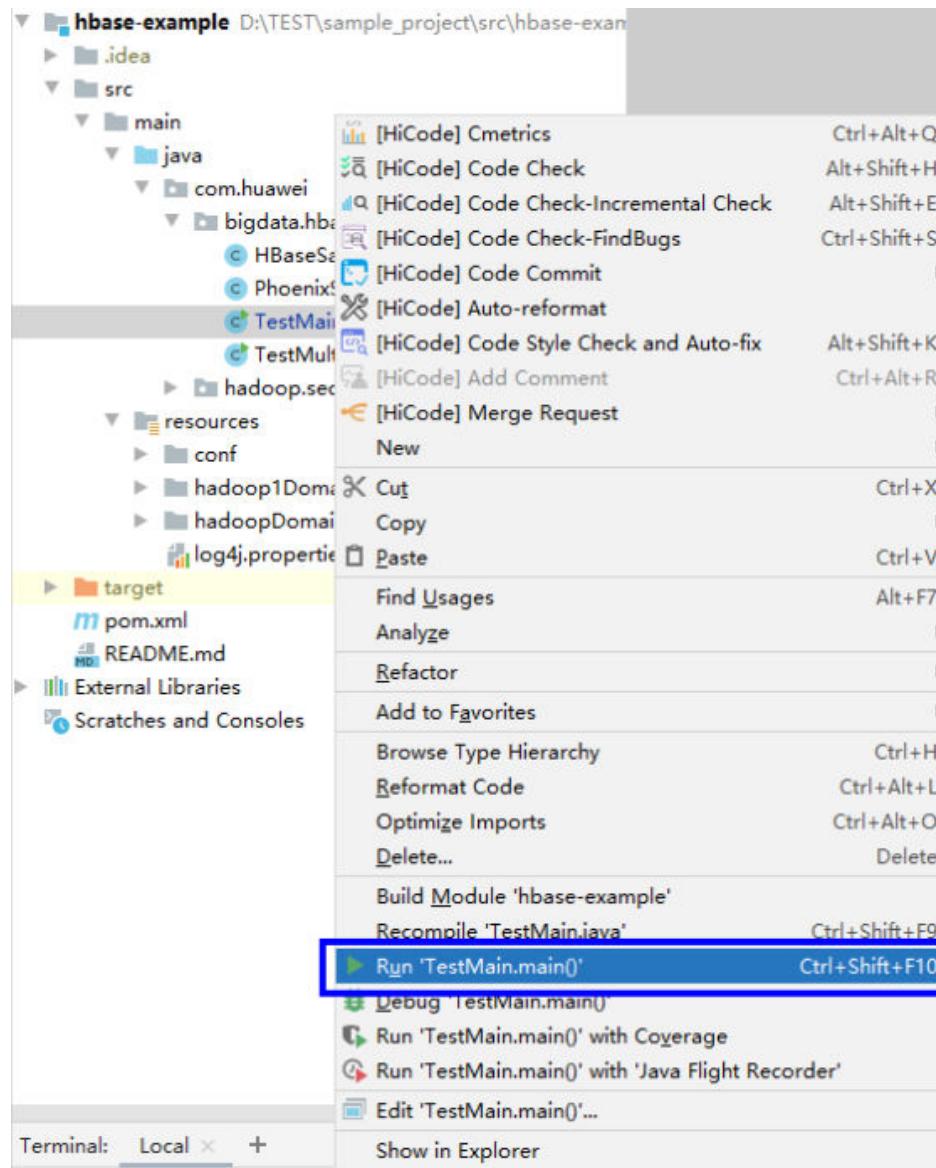
Figure 1-131 Compilation completed



3. Run the program.

Right-click the **TestMain.java** file and choose **Run'TestMain.main()'** from the shortcut menu.

Figure 1-132 Run the application.



----End

#### 1.10.4.1.2 Viewing Windows Commissioning Results

##### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the IntelliJ IDEA running result.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see **More Information > External Interfaces > WebUI**.
- Using HBase Shell command, see **More Information > External Interfaces > Shell**.

## Procedure

- After the **hbase-sample** is successfully executed, the following information is displayed:

```
...
2020-09-09 22:11:48,496 INFO [main] example.TestMain: Entering testCreateTable.
2020-09-09 22:11:48,894 INFO [main] example.TestMain: Creating table...
2020-09-09 22:11:50,545 INFO [main] example.TestMain: Master:
10-1-131-140,21300,1441784082485
Number of backup masters: 1
10-1-131-130,21300,1441784098969
Number of live region servers: 3
10-1-131-150,21302,1441784158435
10-1-131-130,21302,1441784126506
10-1-131-140,21302,1441784118303
Number of dead region servers: 0
Average load: 1.0
Number of requests: 0
Number of regions: 3
Number of regions in transition: 0
2020-09-09 22:11:50,562 INFO [main] example.TestMain:
Lorg.apache.hadoop.hbase.NamespaceDescriptor;@11c6af6
2020-09-09 22:11:50,562 INFO [main] example.TestMain: Table created successfully.
2020-09-09 22:11:50,563 INFO [main] example.TestMain: Exiting testCreateTable.
2020-09-09 22:11:50,563 INFO [main] example.TestMain: Entering testMultiSplit.
2020-09-09 22:11:50,630 INFO [main] example.TestMain: MultiSplit successfully.
2020-09-09 22:11:50,630 INFO [main] example.TestMain: Exiting testMultiSplit.
2020-09-09 22:11:50,630 INFO [main] example.TestMain: Entering testPut.
2020-09-09 22:11:51,148 INFO [main] example.TestMain: Put successfully.
2020-09-09 22:11:51,148 INFO [main] example.TestMain: Exiting testPut.
2020-09-09 22:11:51,148 INFO [main] example.TestMain: Entering createIndex.
...
...
```

### NOTE

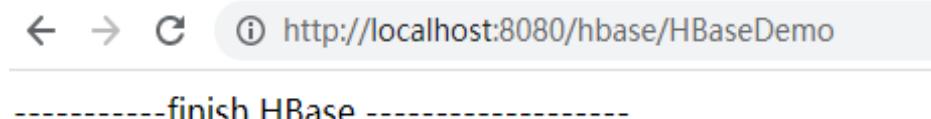
In the Windows environment, the following exception occurs but does not affect services.

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

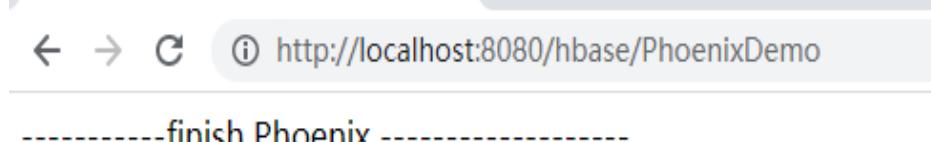
- Result of interconnecting HBase/Phoenix with SpringBoot:

Open a browser and access **http://IP address of the sample running node:8080/hbase/HBaseDemo** and **http://IP address of the sample running node:8080/hbase/PhoenixDemo**. IDEA prints logs normally, and "finish HBase" and "finish Phoenix" are returned.

**Figure 1-133 "finish HBase" displayed**



**Figure 1-134 "finish Phoenix" returned**



- Log description

The log level is **INFO** by default and more detailed information can be viewed by changing the log level (**DEBUG**, **INFO**, **WARN**, **ERROR**, and **FATAL**). You can modify the **log4j.properties** file to change log levels, for example:

```
hbase.root.logger=INFO,console  
...  
log4j.logger.org.apache.zookeeper=INFO  
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG  
log4j.logger.org.apache.hadoop.hbase=INFO  
# Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.  
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO  
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO  
...
```

## 1.10.4.2 Commissioning an Application in Linux

### 1.10.4.2.1 Compiling and Running an Application When a Client Is Installed

#### Scenario

In a Linux environment where an HBase client is installed, you can upload the JAR package to the prepared Linux running environment and then run an application after the code development is complete.

#### Prerequisites

- You have installed the HBase client.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node where the client locates. The host name and IP address must be in one-to-one mapping.

#### Procedure

##### Step 1 Export a JAR package.

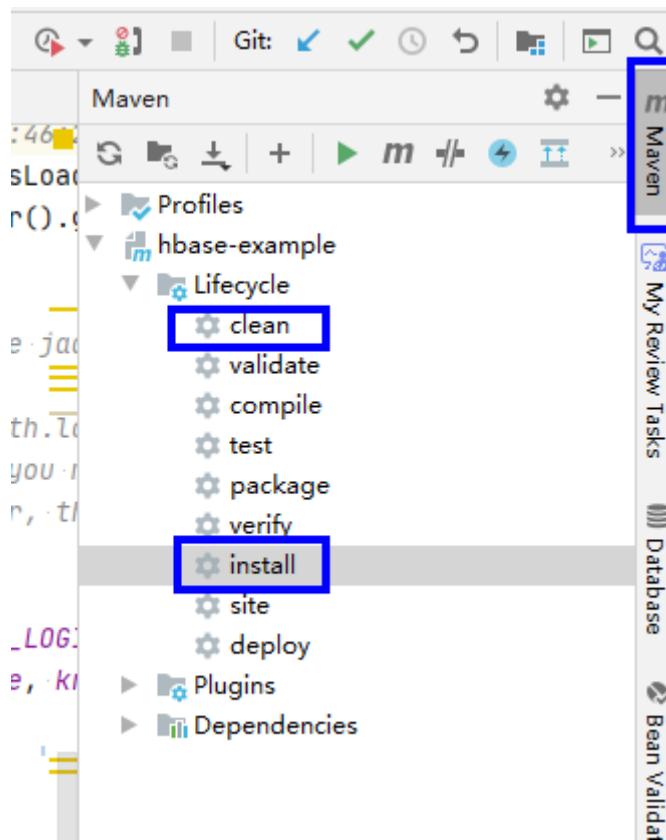
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

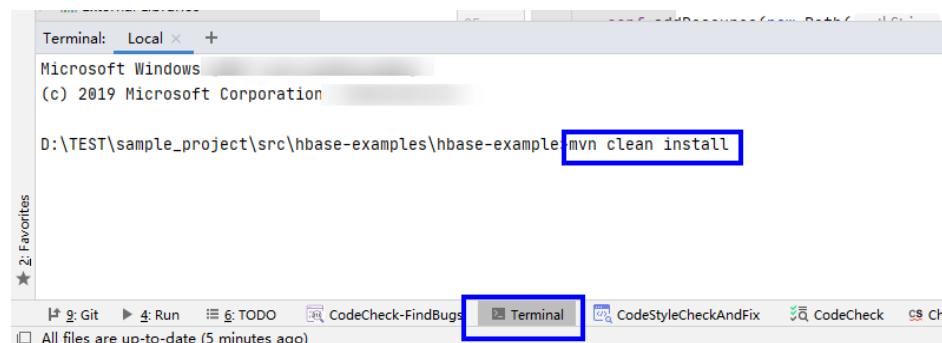
Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 1-135 Maven tools: clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 1-136 Enter mvn clean compile in the IDEA Terminal text box.



After the compilation is completed, the message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

#### Step 2 Example of whether to run HBase/Phoenix to interconnect with Spring Boot:

- If yes, perform the following steps to run the sample:
  - Create a running directory in the Linux environment and place **hbase-springboot-\*jar** in the target directory to the directory. Upload the

configuration file and user authentication file to the corresponding path specified in [Step 1](#).

- b. Switch to the running directory and run the following command to run the JAR package:

**java -jar hbase-springboot-\*.jar**

- If no, go to [Step 3](#).

**Step 3** Export the JAR file on which the sample project depends.

Access the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of IDEA or by using other command line tools.

Run the command **mvn dependency:copy-dependencies -DoutputDirectory=lib**.

The **lib** folder is generated in the directory where the **pom.xml** file is located. The **lib** folder contains the JAR files on which the sample project depends.

**Step 4** Run the JAR package.

1. Before running the JAR package on the Linux client, run the following command to go to the client directory:

**cd \$BIGDATA\_CLIENT\_HOME**



**\$BIGDATA\_CLIENT\_HOME** is the installation directory of the HBase client.

2. Then run the following command:

**source bigdata\_env**



After the multi-instance function is enabled, run the following command to switch to the client of the specified service instance before performing application development for the HBase service instance.

For example, for HBase2, run the **source /opt/hadoopclient/HBase2/component\_env** command.

3. Upload the JAR file (non-dependent JAR file) of the sample project generated in the application development environment to the *Client installation directory/HBase/hbase/lib* directory in the client running environment. If the HBase dual-read sample needs to be executed, Upload the hbase-dualclient JAR package exported in [Step 3](#) to the directory. In addition, you need to copy the configuration file and authentication file obtained in section [Preparing the Connection Cluster Configuration File](#) to the *Client installation directory/HBase/hbase/conf* directory.
4. Go to the **\$BIGDATA\_CLIENT\_HOME/HBase/hbase** directory and run the following command to run the JAR package. The task is complete.

**hbase com.huawei.bigdata.hbase.examples.TestMain**

In the preceding command, **hbase com.huawei.bigdata.hbase.examples.TestMain** is used as an example.

----End

### 1.10.4.2.2 Compiling and Running an Application When No Client Is Installed

#### Scenario

In a Linux environment where no HBase client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

#### Prerequisites

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by the JAR package exported by IntelliJ IDEA.
- If the Linux host is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node. The host name and IP address must be in one-to-one mapping.

#### Procedure

**Step 1** Export a JAR package. For details, see [Step 1](#) in section [Compiling and Running an Application When a Client Is Installed](#).

**Step 2** Example of whether to run HBase/Phoenix to interconnect with Spring Boot:

- If yes, perform the following steps to run the sample:
  - a. Create a running directory in the Linux environment and place **hbase-springboot-\*jar** in the target directory to the directory. Upload the configuration file and user authentication file to the corresponding path specified in [Step 1](#).
  - b. Switch to the running directory and run the following command to run the JAR package:  
**java -jar hbase-springboot-\*jar**
- If no, go to [Step 3](#).

**Step 3** Prepare for the required JAR packages and configuration files.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories **lib** and **conf**. Export the JAR package that the sample project depends on. For details about how to export the JAR package, see [Step 3](#) in [1.4.2.1 Compiling and Running an Application When a Client Is Installed](#). Upload this JAR file and that exported in [Step 1](#) to the **lib** directory on the Linux server. Upload the configuration file and authentication file obtained in section [Preparing for Development Environment](#) to the **conf** directory on Linux.

2. In **/opt/test**, create the **run.sh** script, modify the following content, and save the file:

```
#!/bin/sh  
BASEDIR=`cd $(dirname $0);pwd`  
cd ${BASEDIR}  
for file in ${BASEDIR}/lib/*.jar  
do  
i_cp=$i_cp:$file  
echo "$file"  
done  
for file in ${BASEDIR}/conf/*
```

```
do  
i_cp=$i_cp:$file  
done  
  
java -cp ${i_cp} com.huawei.bigdata.hbase.examples.TestMain
```

In the preceding content, *com.huawei.bigdata.hbase.examples.TestMain* is used as an example.

**Step 4** Go to **/opt/test** and run the following commands to run the JAR packages:

**sh run.sh**

**----End**

#### 1.10.4.2.3 Viewing Linux Commissioning Results

##### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see **More Information > External Interfaces > WebUI**.
- Using HBase Shell command, see **More Information > External Interfaces > Shell**.

##### Procedure

- You can view the execution details of the submitted application in the run logs. For example, after the **hbase-sample** is successfully executed, the following information is displayed:

```
2280 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Entering testCreateTable.  
3091 [main] WARNcom.huawei.hadoop.hbase.example.HBaseSample- table already exists  
3091 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Exiting testCreateTable.  
3091 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Entering testPut.  
3264 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Put successfully.  
3264 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Exiting testPut.  
3264 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Entering testGet.  
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample-  
012005000201:info,address,Shenzhen, Guangdong  
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample-  
012005000201:info,name,yugeZhang San  
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Get data successfully.  
3283 [main] INFOcom.huawei.hadoop.hbase.example.HBaseSample- Exiting testGet.  
3283 [main] INFOorg.apache.hadoop.hbase.client.ConnectionManager$HConnectionImplementation-  
Closing zookeeper sessionid=0xd000035eba278e9  
3297 [main] INFOorg.apache.zookeeper.ZooKeeper- Session: 0xd000035eba278e9 closed  
3297 [main-EventThread] INFOorg.apache.zookeeper.ClientCnxn- EventThread shut down  
-----finish HBase -----
```

- Result of interconnecting HBase/Phoenix with SpringBoot:

Open a browser and access **http://IP address of the sample running node:8080/hbase/HBaseDemo** and **http://IP address of the sample running node:8080/hbase/PhoenixDemo**. IDEA prints logs normally, and "finish HBase" and "finish Phoenix" are returned.

Figure 1-137 "finish HBase" displayed

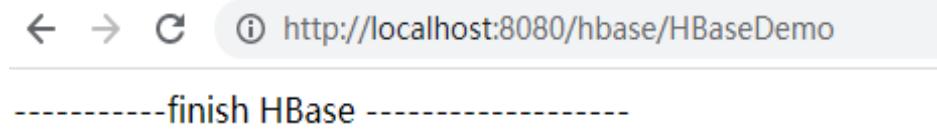
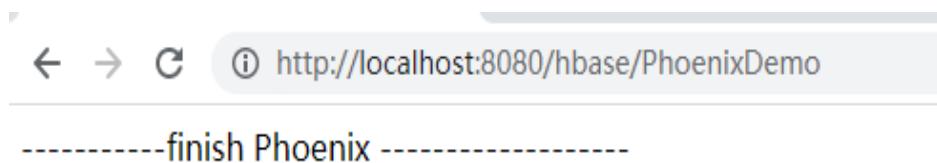


Figure 1-138 "finish Phoenix" returned



## 1.10.5 More Information

### 1.10.5.1 SQL Query

#### Function

Phoenix is an intermediate structured query language (SQL) layer built on HBase. Phoenix provides a JDBC driver that can be embedded in a client. The Phoenix query engine converts input SQL statements to one or multiple HBase scans, and compiles and executes the scan tasks to generate a standard JDBC result set.

#### Example Code

- The **hbase-example/conf/hbase-site.xml** file on the client is used to configure the temporary directory for storing query results. If the client program configures the temporary directory in a Linux environment, configure a Linux path. If the client program configures the temporary directory in a Windows environment, configure a Windows path.

```
<property>
    <name>phoenix.spool.directory</name>
    <value>[1] Temporary directory for storing intermediate query results</value>
</property>
```

- JAVA Example: Use the JDBC interface to access HBase.

```
public String getURL(Configuration conf)
{
    String phoenix_jdbc = "jdbc:phoenix";
    String zkQuorum = conf.get("hbase.zookeeper.quorum");
    return phoenix_jdbc + ":" + zkQuorum;
}

public void testSQL()
{
    String tableName = "TEST";
    // Create table
    String createTableSQL = "CREATE TABLE IF NOT EXISTS TEST(id integer not null primary key,
name varchar, account char(6), birth date)";

    // Delete table
    String dropTableSQL = "DROP TABLE TEST";

    // Insert
```

```
String upsertSQL = "UPSERT INTO TEST VALUES(1,'John','100000',  
TO_DATE('1980-01-01','yyyy-MM-dd'))";  
  
// Query  
String querySQL = "SELECT * FROM TEST WHERE id = ?";  
  
// Create the Configuration instance  
Configuration conf = getConfiguration();  
  
// Get URL  
String URL = getURL(conf);  
  
Connection conn = null;  
PreparedStatement preStat = null;  
Statement stat = null;  
ResultSet result = null;  
  
try  
{  
    // Create Connection  
    conn = DriverManager.getConnection(URL);  
    // Create Statement  
    stat = conn.createStatement();  
    // Execute Create SQL  
    stat.executeUpdate(createTableSQL);  
    // Execute Update SQL  
    stat.executeUpdate(upsertSQL);  
    // Create PrepareStatement  
    preStat = conn.prepareStatement(querySQL);  
    conn.commit();  
    // Execute query  
    preStat.setInt(1,1);  
    result = preStat.executeQuery();  
    // Get result  
    while (result.next())  
    {  
        int id = result.getInt("id");  
        String name = result.getString(1);  
    }  
}  
catch (Exception e)  
{  
    // handler exception  
}  
finally  
{  
    if(null != result){  
        try {  
            result.close();  
        } catch (Exception e2) {  
            // handler exception  
        }  
    }  
    if(null != stat){  
        try {  
            stat.close();  
        } catch (Exception e2) {  
            // handler exception  
        }  
    }  
    if(null != conn){  
        try {  
            conn.close();  
        } catch (Exception e2) {  
            // handler exception  
        }  
    }  
}
```

## Precaution

- You need to configure a temporary directory for storing intermediate query results in **hbase-site.xml**. The size of the query result set is restricted by the directory size.
- Phoenix provides most **java.sql** interfaces and follows the ANSI SQL standard.

### 1.10.5.2 HBase Dual-Read Configuration Items

This section provides the details of all the configurations required for the HBase dual-read feature.

## HBase Dual-Read Operations

**Table 1-83** Configuration items in hbase-dual.xml

Configuration Item	Description	Default Value	Level
hbase.dualclient.active.cluster.configuration.path	HBase client configuration directory of the active cluster	None	Mandatory
hbase.dualclient.standby.cluster.configuration.path	HBase client configuration directory of the standby cluster	None	Mandatory
dual.client.schedule.update.table.delay.second	DR table update interval	5	Optional
hbase.dualclient.gitchtimeout.ms	Maximum glitch time can be tolerated in the active cluster	50	Optional
hbase.dualclient.slow.query.timeout.ms	Slow query alarm log	180000	Optional
hbase.dualclient.active.cluster.id	Active cluster ID	ACTIVE	Optional
hbase.dualclient.standby.cluster.id	Standby cluster ID	STANDBY	Optional
hbase.dualclient.active.executor.thread.max	Maximum size of the thread pool for processing requests to the active cluster	100	Optional

Configuration Item	Description	Default Value	Level
hbase.dualclient.active.executor.thread.core	Core size of the thread pool for processing requests to the active cluster	100	Optional
hbase.dualclient.active.executor.queue	Queue size of the thread pool for processing requests to the active cluster	256	Optional
hbase.dualclient.standby.executor.thread.max	Maximum size of the thread pool for processing requests to the standby cluster	100	Optional
hbase.dualclient.standby.executor.thread.core	Core size of the thread pool for processing requests to the standby cluster	100	Optional
hbase.dualclient.standby.executor.queue	Queue size of the thread pool for processing requests to the standby cluster	256	Optional
hbase.dualclient.clear.executor.thread.max	Maximum size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.thread.core	Core size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.queue	Queue size of the thread pool for clearing resources	Integer.MAX_VALUE	Optional
dual.client.metrics.enable	Whether to print client metric information	true	Optional
dual.client.schedule.metrics.second	Interval for printing client metric information	300	Optional

Configuration Item	Description	Default Value	Level
dual.client.asyncronous.enable	Whether to asynchronously request the active and standby clusters	false	Optional

## Printing Metric Information

**Table 1-84** Basic specifications

Metric Name	Description	Log level
total_request_count	Total number of queries in a period	INFO
active_success_count	Number of successful queries in the active cluster in a period	INFO
active_error_count	Number of failed queries in the active cluster in a period	INFO
active_timeout_count	Number of query timeouts in the active cluster in a period	INFO
standby_success_count	Number of successful queries in the standby cluster in a period	INFO
standby_error_count	Number of failed queries in the standby cluster in a period	INFO
Active Thread pool	Periodically printed information about the thread pool for processing requests to the active cluster	DEBUG
Standby Thread pool	Periodically printed information about the thread pool for processing requests to the standby cluster	DEBUG

Metric Name	Description	Log level
Clear Thread pool	Periodically printed information about the thread pool for releasing resources	DEBUG

**Table 1-85** Histogram indicators for **GET**, **BatchGET**, and **SCAN** requests

Metric Name	Description	Log level
averageLatency(ms)	Average latency	INFO
minLatency(ms)	Minimum latency	INFO
maxLatency(ms)	Maximum latency	INFO
95thPercentileLatency(ms)	Maximum latency of 95% requests	INFO
99thPercentileLatency(ms)	Maximum latency of 99% requests	INFO
99.9PercentileLatency(ms)	Maximum latency of 99.9% requests	INFO
99.99PercentileLatency(ms)	Maximum latency of 99.99% requests	INFO

### 1.10.5.3 External Interfaces

#### 1.10.5.3.1 Shell

You can directly perform operations on HBase using Shell on the server. Versions of Shell interfaces of HBase need to be consistent with those in the open-source community. For details, see <http://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/>.

Methods of running the Shell command:

Go to any directory on the HBase client and run the following command:

**hbase shell**

Go to the running mode of the HBase command (also called CLI client connection).

```
hbase(main):001:0>
```

Run the **help** command to obtain the help information of the HBase command parameters.

## Precautions

The **count** command does not support conditional statistics. It supports only full table statistics.

## Command to retrieve HBase replication metrics

All the required metrics will be added for the shell command "status".

- Command to view replication source metrics.

**hbase(main):019:0> status 'replication', 'source'**

The output is as follows:

```
version 2.4.14
1 live servers
BLR1000006595:
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0 PeerID=3,
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0,
FailedReplicationAttempts=0
```

- Command to view replication sink metrics.

**hbase(main):020:0> status 'replication', 'sink'**

The output is as follows:

```
version 2.4.14
1 live servers
BLR1000006595:
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

- Command to view both replication source and replication sink metrics.

**hbase(main):018:0> status 'replication'**

The output is as follows:

```
version 2.4.14
1 live servers
BLR1000006595:
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0 PeerID=3,
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0,
FailedReplicationAttempts=0
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

### 1.10.5.3.2 Java API

For details about HBase APIs, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

### API Usage Suggestions

- **org.apache.hadoop.hbase.Cell** rather than **org.apache.hadoop.hbase.KeyValue** is recommended as the KV data object.

- It is recommended that `Connection connection = ConnectionFactory.createConnection(conf)` be used to create a connection. The HTablePool is abandoned.
- `org.apache.hadoop.hbase.mapreduce` rather than `org.apache.hadoop.hbase.mapred` is recommended.
- You are advised to obtain the HBase client operation object using the `getAdmin()` method of the constructed Connection object.

## Common HBase APIs

Common HBase Java classes are as follows:

- Interface class Admin: It is the core class of HBase client applications, which encapsulates APIs for HBase management, such as table creation and table deletion. For details, see [Table 1-86](#).
- Interface class Table: It is a class for HBase read/write, which encapsulates APIs for reading and writing HBase tables. For details, see [Table 1-87](#).

**Table 1-86** org.apache.hadoop.hbase.client.Admin

Method	Description
<code>boolean tableExists(final TableName tableName)</code>	This method is used to check whether a specified table exists. If the table exists in the <b>hbase:meta</b> table, <b>true</b> is returned. Otherwise, <b>false</b> is returned.
<code>HTableDescriptor[] listTables(String regex)</code>	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default.
<code>HTableDescriptor[] listTables(final Pattern pattern, final boolean includeSysTables)</code>	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.

Method	Description
TableName[] listTableNames(String regex)	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default. The function of this method is similar to that of listTables. The only difference is that this method returns TableName[].
TableName[] listTableNames(final Pattern pattern, final boolean includeSysTables)	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.
void createTable(HTableDescriptor desc)	This method is used to create a table with only one region.
void createTable(HTableDescriptor desc, byte[] startKey, byte[] endKey, int numRegions)	This method is used to create a table with a specified number of regions. The endKey of the first region is the startKey, and the StartKey of the last region is the endKey. If there are a large number of regions, the invoking of this method may time out.
void createTable(final HTableDescriptor desc, byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the startKey of each region are determined by splitKeys. If there are a large number of regions, the invoking of this method may time out.
void createTable(final HTableDescriptor desc, final byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the startKey of each region are determined by splitKeys. This method uses asynchronous invoking, and does not wait for the created table to be online.
void deleteTable(final TableName tableName)	This method is used to delete a specified table.

Method	Description
public void truncateTable(final TableName tableName, final boolean preserveSplits)	This method is used to re-create a specified table. If the second parameter is set to <b>true</b> , the region of the reconstructed table is the same as that of the previous table. Otherwise, there is only one region.
void enableTable(final TableName tableName)	This method is used to enable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void enableTableAsync(final TableName tableName)	This method is used to enable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be online.
void disableTable(final TableName tableName)	This method is used to disable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void disableTableAsync(final TableName tableName)	This method is used to disable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be offline.
boolean isTableEnabled(TableName tableName)	This method is used to check whether a table is enabled. This method can be used with the enableTableAsync method to check whether the operation of enabling a table is complete.
boolean isTableDisabled(TableName tableName)	This method is used to check whether a table is disabled. This method can be used with the disableTableAsync method to check whether the operation of disabling a table is complete.
void addColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to add a column family to a specified table.
void deleteColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to delete a specified column family from a specified table.
void modifyColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to modify a specified column family.

**Table 1-87** org.apache.hadoop.hbase.client.Table

Method	Description
boolean exists(Get get)	This method is used to check whether the specified rowkey exists in the table.
boolean[] existsAll(List<Get> gets)	This method is used to check whether the specified rowkey exists in the table. The returned Boolean array result corresponds to the input parameter position.
Result get(Get get)	This method is used to read data based on the specified rowkey.
Result[] get(List<Get> gets)	This method is used to read data in batches by specifying a batch of rowkeys.
ResultScanner getScanner(Scan scan)	This method is used to obtain a scanner object in the table. The query parameters can be specified by the input parameter scan, including <b>StartRow</b> , <b>StopRow</b> , and <b>caching</b> .
void put(Put put)	This method is used to write a data record to the table.
void put(List<Put> puts)	This method is used to write a batch of data records to the table.
void close()	This method is used to release the resources held by the table object.

 NOTE

The table [org.apache.hadoop.hbase.client.Admin](#) and [org.apache.hadoop.hbase.client.Table](#) list only some common methods.

#### 1.10.5.3.3 SQLLine

You can run **sqlline.py** on the client to perform SQL operations on HBase. The SQLLine APIs of Phoenix are the same as those of the open-source community. For details, see <http://phoenix.apache.org/>.

**Table 1-88** lists common SQLline syntax, **Table 1-89** lists common functions, and [Phoenix Command Line](#) describes how to use commands.

**Table 1-88** Common syntax

Command	Description	Example
<b>CREATE TABLE</b>	Creates a table.	CREATE TABLE MY_TABLE(id BIGINT not null primary key, name VARCHAR);
<b>ALTER</b>	Alters a table or a view.	ALTER TABLE MY_TABLE DROP COLUMN name;
<b>DROP TABLE</b>	Deletes a table.	DROP TABLE MY_TABLE;
<b>UPSERT VALUES</b>	Inserts or changes data.	UPSERT INTO MY_TABLE VALUES(1,'abc');
<b>SELECT</b>	Queries data.	SELECT * FROM MY_TABLE;
<b>CREATE INDEX</b>	Creates a global index.	CREATE INDEX MY_IDX ON MY_TABLE(name);
<b>CREATE LOCAL INDEX</b>	Creates a local index.	CREATE LOCAL INDEX MY_LOCAL_IDX ON MY_TABLE(id,name);
<b>ALTER INDEX</b>	Changes the index status.	ALTER INDEX MY_IDX ON MY_TABLE DISABLE;
<b>DROP INDEX</b>	Deletes an index.	DROP INDEX MY_IDX ON MY_TABLE;
<b>EXPLAIN</b>	Displays an execution plan.	EXPLAIN SELECT name FROM MY_TABLE;
<b>CREATE SEQUENCE</b>	Creates a sequence.	CREATE SEQUENCE MY_SEQUENCE;
<b>DROP SEQUENCE</b>	Deletes a sequence.	DROP SEQUENCE MY_SEQUENCE;
<b>CREATE VIEW</b>	Creates a view.	CREATE VIEW MY_VIEW AS SELECT * FROM MY_TABLE;
<b>DROP VIEW</b>	Deletes a view.	DROP VIEW MY_VIEW;
<b>CREATE SCHEMA</b>	Creates a schema.	CREATE SCHEMA MY_SCHEMA;
<b>USE</b>	Modifies the default schema.	USE MY_SCHEMA;
<b>DROP SCHEMA</b>	Deletes a schema.	DROP SCHEMA MY_SCHEMA;

**Table 1-89** Common functions

Function Type	Function	Description	Example
String functions	<b>SUBSTR</b>	Extracts a part of a string.	SUBSTR('[Hello]', 2, 5)
	<b>INSTR</b>	Queries the position of the first occurrence of a string in another string.	INSTR('Hello World', 'World')
	<b>TRIM</b>	Removes whitespaces from both ends of a string.	TRIM(' Hello ')
	<b>LTRIM</b>	Removes whitespaces found on the left-hand side of the string.	LTRIM(' Hello')
	<b>RTRIM</b>	Removes whitespaces found on the right-hand side of the string.	RTRIM('Hello ')
	<b>LPAD</b>	Left-pads a string with another string.	LPAD('John',30,'*')
	<b>LENGTH</b>	Gets the length of a given string.	LENGTH('Hello')
	<b>UPPER</b>	Converts all the characters in a string to uppercase characters.	UPPER('Hello')
	<b>LOWER</b>	Converts all the characters in a string to lowercase characters.	LOWER('HELLO')
	<b>REVERSE</b>	Reverses a string.	REVERSE('Hello')
	<b>REGEXP_SPLIT</b>	Splits a string using a regular expression.	REGEXP_SPLIT('ONE,TWO,THREE', ',')

Function Type	Function	Description	Example
	<b>REGEXP_REPLACE</b>	Returns a string by applying a regular expression and replacing the matches with the replacement string.	REGEXP_REPLACE('abc123ABC', '[0-9]+', '#')
	<b>REGEXP_SUBSTR</b>	Returns a substring from a string based on a regular expression pattern.	REGEXP_SUBSTR('na1-appsrv35-sj35', '[^-]+')
Aggregate functions	<b>AVG</b>	Gets the average value.	AVG(X)
	<b>COUNT</b>	Gets the number of data records.	COUNT(*)
	<b>MAX</b>	Gets the maximum value.	MAX(NAME)
	<b>MIN</b>	Gets the minimum value.	MIN(NAME)
	<b>SUM</b>	Gets the sum of all values.	SUM(X)
	<b>STDDEV_POP</b>	Gets the population standard deviation of all values.	STDDEV_POP( X )
	<b>STDDEV_SAMP</b>	Gets the sample standard deviation of all values.	STDDEV_SAMP( X )
	<b>NTH_VALUE</b>	Gets the Nth value in each distinct group ordered according to the <b>ORDER BY</b> specification.	NTH_VALUE( name, 2 ) WITHIN GROUP ( ORDER BY salary DESC )

Function Type	Function	Description	Example
Time and date functions	<b>NOW</b>	Returns the current date.	NOW()
	<b>CURRENT_TIME</b>	Returns the current time.	CURRENT_TIME()
	<b>CURRENT_DATE</b>	Returns the current date.	CURRENT_DATE()
	<b>TO_DATE</b>	Parses a string and returns a date.	TO_DATE('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIME</b>	Converts the given string into a <b>TIME</b> instance.	TO_TIME('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIMESTAMP</b>	Converts the given string into a <b>TIMESTAMP</b> instance.	TO_TIMESTAMP('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>YEAR</b>	Returns the year of the specified date.	YEAR(TO_DATE('2015-6-05'))
	<b>MONTH</b>	Returns the month of the specified date.	MONTH(TO_TIMESTAMP('2015-6-05'))
	<b>WEEK</b>	Returns the week of the specified date.	WEEK(TO_TIME('2010-6-15'))
	<b>HOUR</b>	Returns the hour of the specified date.	HOUR(TO_TIMESTAMP('2015-6-05'))
	<b>MINUTE</b>	Returns the minute of the specified date.	MINUTE(TO_TIME('2015-6-05'))
	<b>SECOND</b>	Returns the second of the specified date.	SECOND(TO_DATE('2015-6-05'))

Function Type	Function	Description	Example
Numeric functions	<b>ROUND</b>	Rounds the numeric or timestamp expression to the nearest scale or time unit specified.	ROUND(2.56)
	<b>CEIL</b>	Rounds any fractional value up to the next even multiple.	CEIL(2.34)
	<b>FLOOR</b>	Rounds any fractional value down to the previous even multiple.	FLOOR(2.34)
	<b>TRUNC</b>	Rounds any fractional value down to the previous even multiple (same as <b>FLOOR</b> ).	TRUNC(2.34)
	<b>TO_NUMBER</b>	Formats a string as a number.	TO_NUMBER('-123.33')
	<b>RAND</b>	Returns a random number.	RAND()
Math functions	<b>ABS</b>	Returns the absolute value of the given numeric expression.	ABS(-1)
	<b>SQRT</b>	$\sqrt{x}$	SQRT(1.1)
	<b>EXP</b>	$e^x$	EXP(-1)
	<b>POWER</b>	$x^y$	POWER(2, 3)
	<b>LN</b>	$\ln(x)$	LN(3)

Function Type	Function	Description	Example
	<b>LOG</b>	$\log_x(y)$	LOG(2, 3)
Array functions	<b>ARRAY_ELEM</b>	Uses the array subscript notation to access an array element.	ARRAY_ELEM(ARRAY[1,2,3], 1)
	<b>ARRAY-prepend</b>	Appends the given element to the beginning of the array.	ARRAY_APPEND(ARRAY[1,2,3], 4)
	<b>ARRAY_CAT</b>	Concatenates the input arrays and returns the result.	ARRAY_CAT(ARRAY[1,2], ARRAY[3,4])
	<b>ARRAY_FILL</b>	Fills an array with values.	ARRAY_FILL(1, 3)
	<b>ARRAY_TO_STRING</b>	Converts the elements of an array to a string and joins them using the specified delimiter.	ARRAY_TO_STRING(ARRAY['a','b','c'], ',')
	<b>ANY</b>	Used on the right-hand side of a comparison expression to test that any array element satisfies the comparison expression against the left-hand side.	10 > ANY(my_array)
	<b>ALL</b>	Used on the right-hand side of a comparison expression to test that all array elements satisfy the comparison expression against the left-hand side.	10 > ALL(my_array)

Function Type	Function	Description	Example
Other functions	<b>MD5</b>	Computes the MD5 hash of the argument.	MD5(my_column)
	<b>ENCODE</b>	Encodes the expression according to the encoding format provided.	ENCODE(myNumber, 'BASE62')
	<b>DECODE</b>	Decodes the expression according to the encoding format provided.	DECODE('000000008512af277fff fff8', 'HEX')

#### 1.10.5.3.4 JDBC APIs

Phoenix implements most **java.sql** interfaces. The SQL syntax follows the ANSI SQL standard.

For details about the functions that are supported, visit:

<http://phoenix.apache.org/language/functions.html>

For details about the syntax that is supported, visit:

<http://phoenix.apache.org/language/index.html>

#### 1.10.5.3.5 Web UI

##### Scenario

The Web UI displays information about the HBase cluster, such as cluster information, RegionServer and Master, snapshot, and running processes. The information helps you understand the status of the entire HBase cluster.



Contact the administrator to obtain a service account that has the permission to access the web UI and its password.

##### Procedure

1. Log in to FusionInsight Manager, choose **Cluster > Services > HBase**, and click **HMaster (Active)** to open the HBase WebUI.
2. On the Home page of the HBase Web UI, the following information about the HBase is displayed:
  - a. The **Region Servers** page displays basic server information.

**Figure 1-139** Basic information

Region Servers			
ServerName	Start time	Requests Per Second	Num. Regions
VM-3,21302,1415117599935	Wed Nov 05 00:13:19 CST 2014	0	1
VM-4,21302,1415117602775	Wed Nov 05 00:13:22 CST 2014	0	3
VM-5,21302,1415117607877	Wed Nov 05 00:13:27 CST 2014	0	0
Total:3		0	4

- b. The **Backup Masters** page displays information about backup master nodes.

**Figure 1-140** Basic information

Backup Masters		
ServerName	Port	Start Time
VM-4	21300	Wed Nov 05 00:13:08 CST 2014
Total:1		

- c. The **Tables** page displays information about tables in HBase, including user tables, catalog tables, and snapshots.

**Figure 1-141** Basic information

Tables		
User Tables	Catalog Tables	Snapshots
1 table(s) in set. [Details]		
Table Name	Online Regions	Description
t	1	't', {NAME => 'd'}

- d. The **Tasks** page displays information about tasks running on HBase, including the start time and status.

## **Figure 1-142 Task basic information**

Tasks				
Show All Monitored Tasks		Show non-RPC Tasks	Show All RPC Handler Tasks	Show Active RPC Calls
View as JSON				
Start Time	Description		State	Status
Wed Nov 05 03:06:35 CST 2014	Closing region availabilityCheck_VM-5_1415127943,1415127994683.1d82d088fbf4ba672ee2235fd98ae695.		COMPLETE (since 44sec ago)	Closed (since 44sec ago)
Wed Nov 05 00:20:13 CST 2014	RpcServer.reader=0,port=21300		WAITING (since 2mins, 36sec ago)	Waiting for a call (since 2mins, 36sec ago)
Wed Nov 05 00:19:49 CST 2014	RpcServer.reader=9,port=21300		WAITING (since 2mins, 53sec ago)	Waiting for a call (since 2mins, 53sec ago)
Wed Nov 05 00:19:17 CST 2014	RpcServer.reader=8,port=21300		WAITING (since 3mins, 8sec ago)	Waiting for a call (since 3mins, 8sec ago)
Wed Nov 05 00:15:10 CST 2014	RpcServer.reader=7,port=21300		WAITING (since 3mins, 46sec ago)	Waiting for a call (since 3mins, 46sec ago)
Wed Nov 05 00:14:52 CST 2014	RpcServer.reader=6,port=21300		WAITING (since 5mins, 2sec ago)	Waiting for a call (since 5mins, 2sec ago)
Wed Nov 05 00:14:36 CST 2014	RpcServer.reader=5,port=21300		WAITING (since 10sec ago)	Waiting for a call (since 10sec ago)
Wed Nov 05 00:14:01 CST 2014	RpcServer.reader=4,port=21300		WAITING (since 0sec ago)	Waiting for a call (since 0sec ago)

3. The details page of HBase tables displays general information of storage tables.

**Figure 1-143** Table details

User Tables	
Table	Description
t	't', {NAME => 'd', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}

4. The debug dump page displays debugging information.

**Figure 1-144** Debug dump

5. The HBase configuration page displays control information.

**Figure 1-145 HBase Configuration**

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    - <property>
        <name>dfs.journalnode.rpc-address</name>
        <value>0.0.0.0:8485</value>
        <source>hdfs-default.xml</source>
    </property>
    - <property>
        <name>io.storefile.bloom.block.size</name>
        <value>131072</value>
        <source>hbase-default.xml</source>
    </property>
    - <property>
        <name>hbase.sessioncontrol.maxSessions</name>
        <value>65535</value>
        <source>hbase-site.xml</source>
    </property>
    - <property>
        <name>yarn.ipc.rpc.class</name>
        <value>org.apache.hadoop.yarn.ipc.HadoopYarnProtoRPC</value>
        <source>yarn-default.xml</source>
    </property>
    - <property>
        <name>mapreduce.job.maxtaskfailures.per.tracker</name>
        <value>3</value>
        <source>mapred-default.xml</source>
    </property>
    - <property>
        <name>hbase.rest.threads.min</name>
        <value>2</value>
        <source>hbase-default.xml</source>
    </property>
    - <property>
        <name>hbase.rs.cacheblocksonwrite</name>
        <value>false</value>
        <source>hbase-default.xml</source>
    </property>
    - <property>
        <name>ha.health-monitor.connect-retry-interval.ms</name>
        <value>1000</value>
        <source>core-default.xml</source>
    </property>
    - <property>
        <name>yarn.resourcemanager.work-preserving-recovery.enabled</name>
        <value>false</value>
        <source>yarn-default.xml</source>
    </property>
    - <property>
        <name>dfs.client.mmap.cache.size</name>
        <value>256</value>
        <source>hdfs-default.xml</source>
    </property>
```

#### 1.10.5.4 Phoenix Command Line

Phoenix supports SQL statements to operate HBase. The following describes how to use SQL statements to create tables, insert data, query data, and delete tables.

#### Prerequisites

The HBase client has been installed. For details, see [Installing a Client](#). For example, the client has been installed in the `/opt/hadoopclient` directory. The client directory in the following operations is only an example. Change it based on the actual installation directory onsite. Before using the client, download and update the client configuration file, and ensure that the active management node of Manager is available.

## Procedure

**Step 1 Optional:** Log in to the node where the client is installed as the client installation user.

Access the HBase client installation directory:

```
cd /opt/hadoopclient
```

**Step 2** Run the following command to configure environment variables:

```
source bigdata_env
```

**Step 3** If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission to create HBase tables. If Kerberos authentication is disabled for the current cluster, skip this step:

```
kinit MRS cluster user
```

For example, **kinit hbaseuser**.

**Step 4** Running Commands on the Phoenix Client.

```
sqlline.py
```

**Step 5** Create a table:

```
CREATE TABLE TEST (id VARCHAR PRIMARY KEY, name VARCHAR);
```

**Step 6** Insert data:

```
UPSERT INTO TEST(id,name) VALUES ('1','jamee');
```

**Step 7** Query data:

```
SELECT * FROM TEST;
```

**Step 8** Deleting a table:

```
DROP TABLE TEST;
```

**Step 9** Exit the Phoenix CLI:

```
!quit
```

```
----End
```

### 1.10.5.5 FAQs

#### 1.10.5.5.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?

**Step 1** Check whether the **hbase.rpc.controllerfactory.class** configuration item is included in the **hbase-site.xml** configuration file.

```
<name>hbase.rpc.controllerfactory.class</name>
<value>org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory</value>
```

**Step 2** If this configuration item is included in the current application development project, you need to import the **phoenix-core-\*jar** package. You can obtain this package from **HBase/hbase/lib** in the HBase client installation directory.

**Step 3** If you do not import this JAR package, you need to delete **hbase.rpc.controllerfactory.class** from the **hbase-site.xml** configuration file.

----End

### 1.10.5.5.2 What Are the Application Scenarios of the Bulkload and put Data-loading Modes?

#### Question

Both the bulkload and put data-loading modes can be used to load data to HBase. Though the bulkload mode loads data faster than the put mode, the bulkload mode has its own disadvantages. The following describes the application scenarios of these two data-loading modes.

#### Answer

The bulkload starts MapReduce tasks to generate HFile files, and then registers HFile files with HBase. Incorrect use of the bulkload mode will consume more cluster memory and CPU resources due to started MapReduce tasks. The large number of HFile files may frequently trigger Compaction, decreasing the query speed drastically.

Incorrect use of the put mode may cause a slow data loading rate. If the memory allocated to RegionServer is not sufficient, the process may exit.

The application scenarios of the bulkload and put modes are as follows:

- bulkload:
  - Load a large amount of data to HBase in the one-off manner.
  - Load data to HBase with low reliability requirements and without generating WAL files.
  - Low loading and query speed if the put mode is used.
  - The size of the HFile generated after data loading is similar to the size of HDFS block.
- put:
  - The size of the data loaded to one Region at a time is smaller than half the size of an HDFS block.
  - Load data to HBase in real time.
  - The query speed does not decrease wildly during data loading.

### 1.10.5.5.3 An Error Occurred When Building a JAR Package

#### Question

When the sample code is compiled using Maven to build a JAR package, the error message "Could not transfer artifact org.apache.commons:commons-crypto:pom:\${commons-crypto.version}" is displayed and the package building fails.

#### Answer

The hbase-common module depends on commons-crypto. In the **pom.xml** file of hbase-common, the **\${commons-crypto.version}** variable is used to introduce

commons-crypto. The parsing logic of this variable is as follows: If the OS is AArch64, the value is **1.0.0-hw-aarch64**. If the OS is x86\_64, the value is **1.0.0**. If Maven fails to parse the variable using the OS due to incorrect configuration in the compilation environment, you can manually modify the **pom.xml** file to prevent correct compilation.

In the **pom.xml** file, manually change the dependency of the hbase-common module to the following to exclude the commons-crypto dependency:

```
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-common</artifactId>
    <version>${hbase.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-crypto</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Manually add the commons-crypto dependency of the specified version. Enter a correct version based on the OS architecture (x86\_64 or AArch64).

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-crypto</artifactId>
    <version>1.0.0</version>
</dependency>
```

#### 1.10.5.5.4 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "java.lang.OutOfMemoryError: Direct buffer memory" Is Displayed in the Error Information?

**Step 1** Run the **java -version** command to check whether the JDK version is JDK1.8u102 or later.

- If yes, go to [Step 2](#).
- If no, the client for secondary development supports only the built-in OpenJDK 1.8 (jdk-8u181-linux-x64). If the JDK version is not JDK1.8u102 or later, upgrade it to JDK1.8u102 or later, and then go to [Step 2](#).

**Step 2** When running the program, set the JVM parameter **-Djdk.nio.maxCachedBufferSize=262144** to limit the size of the direct memory used by each thread.

----End

#### 1.10.5.5.5 Why Does the Error Information Contain "GSSEException" When an IBM JDK Is Used on a Client to Connect to an HBase Cluster?

##### Question

After an IBM JDK is used on a client to connect to an HBase cluster and the debug log function is enabled, the following error information is displayed:

```
javax.security.sasl.SaslException: Failure to initialize security context [Caused by org.ietf.jgss.GSSEException,
major code: 13, minor code: 0]
```

The error stack is as follows:

```
Caused by: javax.security.sasl.SaslException: Failure to initialize security context [Caused by  
org.ietf.jgss.GSSEException, major code: 13, minor code: 0  
major string: Invalid credentials  
minor string: SubjectCredFinder: no JAAS Subject]  
at com.ibm.security.sasl.gsskerb.GssKrb5Client.<init>(GssKrb5Client.java:161)  
at com.ibm.security.sasl.gsskerb.FactoryImpl.createSaslClient(FactoryImpl.java:79)  
at javax.security.sasl.Sasl.createSaslClient(Sasl.java:400)  
at  
org.apache.hadoop.hbase.security.AbstractHBaseSaslRpcClient.createKerberosSaslClient(AbstractHBaseSaslRp  
cClient.java:125)  
at  
org.apache.hadoop.hbase.security.AbstractHBaseSaslRpcClient.<init>(AbstractHBaseSaslRpcClient.java:106)  
at org.apache.hadoop.hbase.security.NettyHBaseSaslRpcClient.<init>(NettyHBaseSaslRpcClient.java:43)  
at  
org.apache.hadoop.hbase.security.NettyHBaseSaslRpcClientHandler.<init>(NettyHBaseSaslRpcClientHandler.j  
ava:70)  
at org.apache.hadoop.hbase.ipc.NettyRpcConnection.saslNegotiate(NettyRpcConnection.java:194)  
... 20 more
```

## Answer

IBM JDKs and OpenJDK have different SASL implementation methods. When NettyRpcClient is used as the client by default, authentication issues may occur. As a result, the SASL client cannot be created. In this case, you need to modify the client configuration to not to use NettyRpcClient. For details, see [Modifying the Client Configuration](#).

# 1.11 HDFS Development Guide

## 1.11.1 Overview

### 1.11.1.1 Introduction to HDFS

#### Introduction to HDFS

Hadoop distribute file system (HDFS) is a distributed file system with high fault tolerance. HDFS supports data access with high throughput and applies to processing of large data sets.

HDFS applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require high reliability.
- Scenarios that require good scalability.

#### Introduction to HDFS Interface

HDFS can be developed by using Java language. For details of API Reference, see [Java API](#).

### 1.11.1.2 Basic Concepts

#### DataNode

A DataNode is used to store data blocks of each file and periodically report the storage status to the NameNode.

#### NameNode

A NameNode is used to manage the namespace, directory structure, and metadata information of a file system and provide the backup mechanism. NameNodes are classified into the following two types:

- Active NameNode: manages the file system namespace, maintains the directory structure tree and metadata information of a file system, and records the relationship between each data block and the file to which the data block belongs.
- Standby NameNode: Data in a standby NameNode is synchronous with those in an active NameNode. A standby NameNode takes over services from the active NameNode if the active NameNode is exception.

#### JournalNode

A JournalNode synchronizes metadata between the active and standby NameNodes in the High Availability (HA) cluster.

#### ZKFC

ZKFC must be deployed for each NameNode. It is responsible for monitoring NameNode status and writing status information to the ZooKeeper. ZKFC also has permission to select the active NameNode.

#### Colocation

Colocation is used to store associated data or the data to be associated on the same storage node. The HDFS Colocation stores files to be associated on a same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

#### Federation

Federation is adopted to solve most problems in the HDFS architecture that allows only a single NameNode.

Federation allows multiple NameServices. The NameServices can be configured in the HA mode. In HA mode, a hot standby NameNode is provided for the active NameNode to address the single node failure. The standby NameNode is able to quickly take over services from the active NameNode when the active NameNode breaks down. The administrator can start the failover during maintenance.



This function applies only to MRS physical machine clusters.

## Client

The HDFS can be accessed from the Java application programming interface (API), C API, Shell, HTTP REST API and web user interface (WebUI). For details, see [Common API Introduction](#) and [Shell Command Introduce](#).

- JAVA API  
Provides an application interface for the HDFS. This guide describes how to use the Java API to develop HDFS applications.
- C API  
Provides an application interface for the HDFS. This guide describes how to use the C API to develop HDFS applications.
- Shell  
Provides shell commands to perform operations on the HDFS.
- HTTP REST API  
Additional interfaces except Shell, Java API and C API. You can use the interfaces to monitor HDFS status.
- WEB UI  
Provides a visualized management web page.

## keytab file

The keytab file is a key file that stores user information. Applications use the key file for API authentication on MRS.

### 1.11.1.3 Development Process

All stages of the development process are shown and described in [Figure 1-146](#) and [Table 1-90](#).

Figure 1-146 HDFS development process

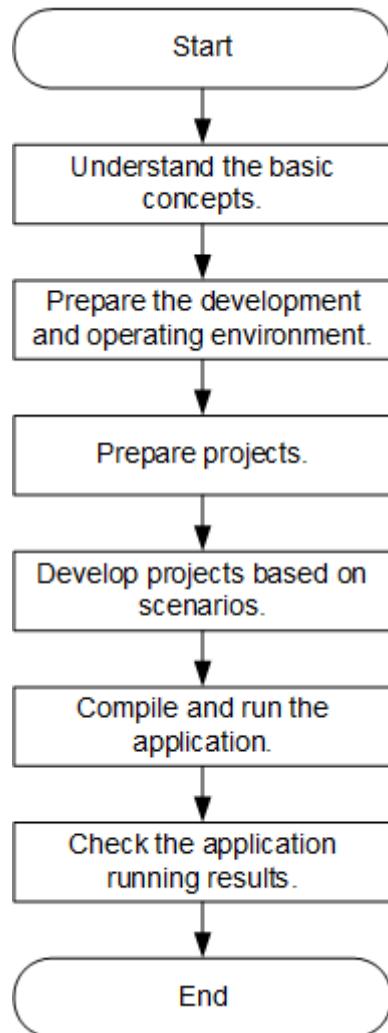


Table 1-90 Description of HDFS development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of HDFS are required to be understood.	<a href="#">Basic Concepts</a>
Prepare the development and operating environment.	Use IntelliJ IDEA and configure the development environment based on the reference. The running environment of HDFS is the HDFS client. Install and configure the client based on the reference.	<a href="#">Preparing Development and Operating Environment</a>
Prepare projects.	HDFS provides sample projects in various scenarios. Sample projects can be imported for studying.	<a href="#">Configuring and Importing Sample Projects</a>

Stage	Description	Reference
Prepare for security authentication.	If a safe cluster is used, the safety certification must be performed.	<a href="#">Preparing the Authentication Mechanism</a>
Develop projects based on scenarios.	Provide the sample project. This helps users to learn about the programming interfaces of all HDFS components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Commissioning the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Commissioning the Application</a>

## 1.11.2 Environment Preparation

### 1.11.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

[Table 1-91](#) describes the environment required for application development.

**Table 1-91** Development Environment

Preparation	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS.</li></ul> <p>If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</p>

Preparation	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <code>com.ibm.jsse2.overrideDefaultTLS</code> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
User development preparation	See <a href="#">Preparing a Developer Account</a> for configuration.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. **Accessing FusionInsight Manager of an MRS Cluster.** In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HDFS\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the HDFS sample project.  
The keytab file obtained during the **Preparing a Developer Account** is also stored in this directory.
- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see **Installing a Client**.
  - b. **Accessing FusionInsight Manager of an MRS Cluster.** Download the cluster client software package to the active management node and

decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/HDFS/config** directory and save them to the **conf** folder of the project code.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp HDFS/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-92](#) describes the main configuration files.

**Table 1-92** Configuration files

File	Function
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

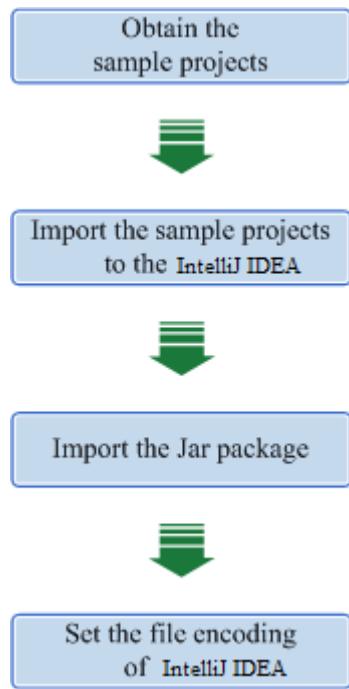
### 1.11.2.2 Configuring and Importing Sample Projects

#### Scenario

HDFS provides sample projects for multiple scenarios to help you quickly learn HDFS projects.

The procedure for importing HDFS example codes is described as follows: [Figure 1-147](#) shows the procedure.

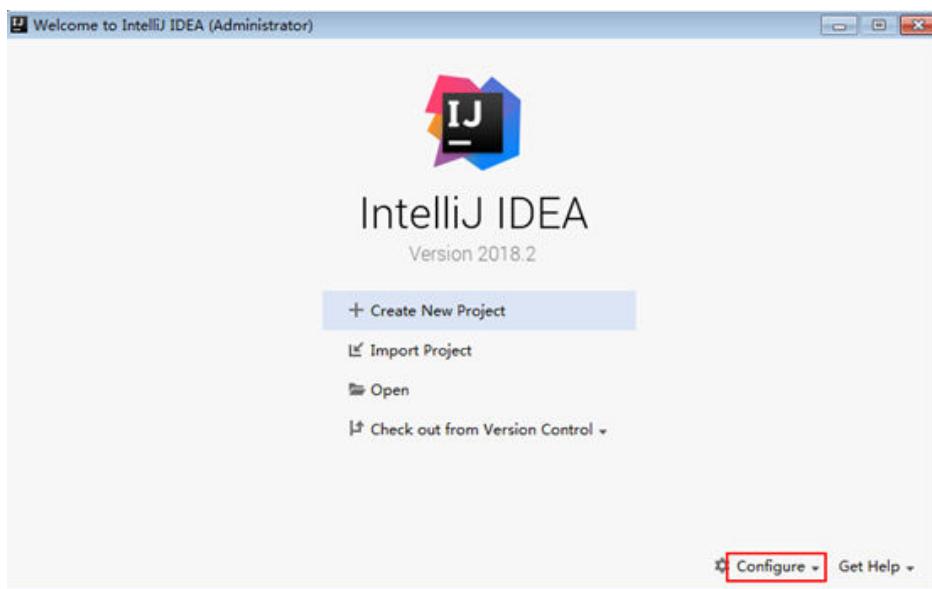
Figure 1-147 Sample project importing procedure



## Procedure

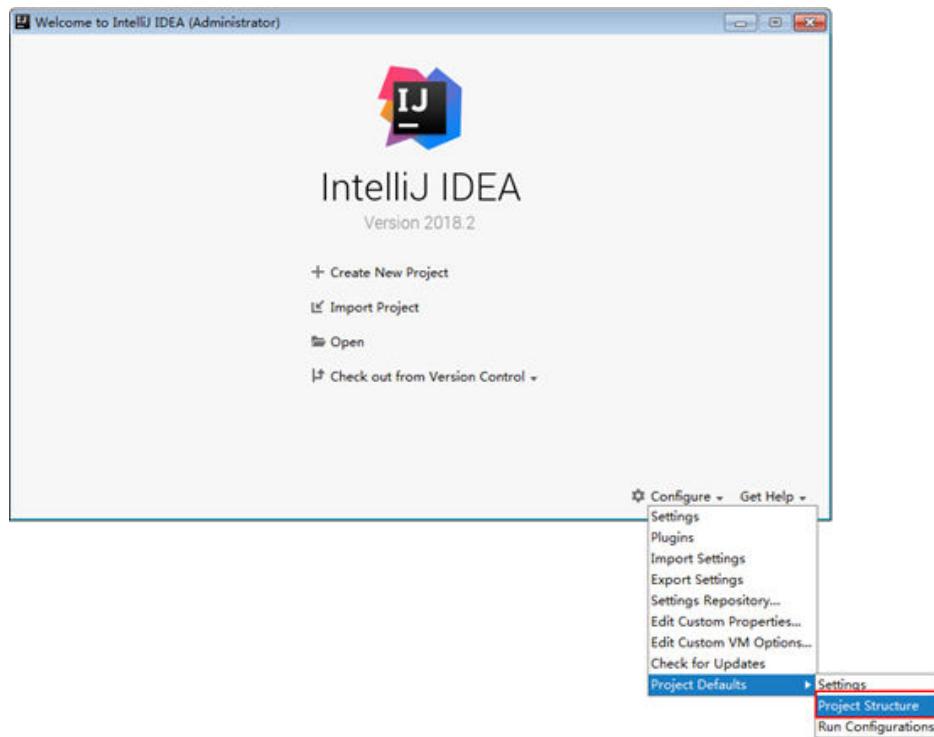
- Step 1** Obtain the sample project folder **hdfs-example-security** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) section to the **conf** directory of the sample project.
- Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.
  1. Open IntelliJ IDEA and click **Configure**.

Figure 1-148 Quick Start



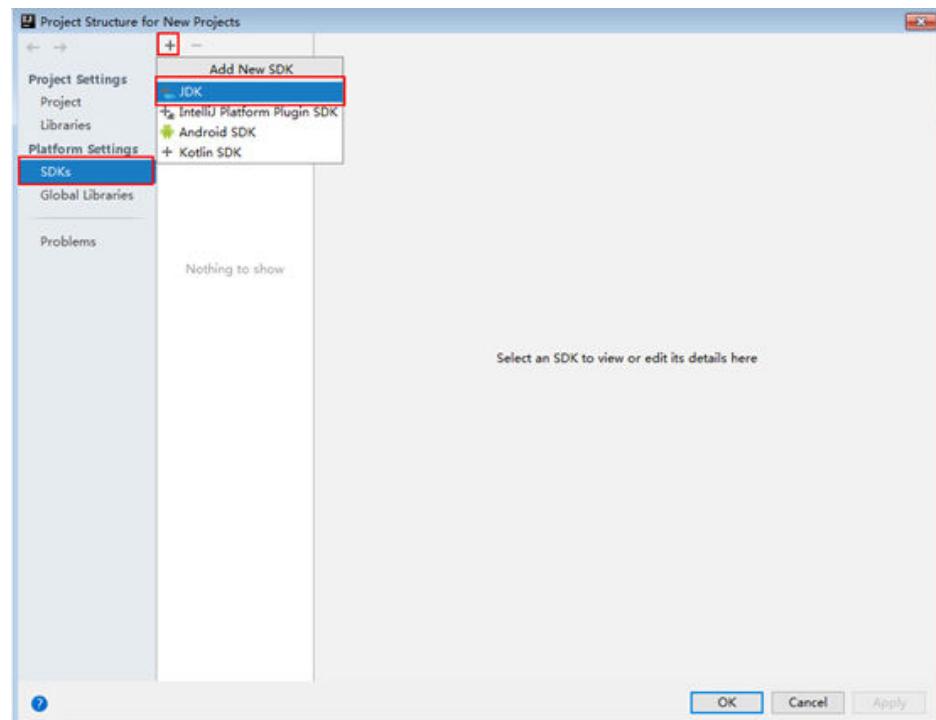
2. Choose **Project Defaults > Project Structure**.

**Figure 1-149** Configure



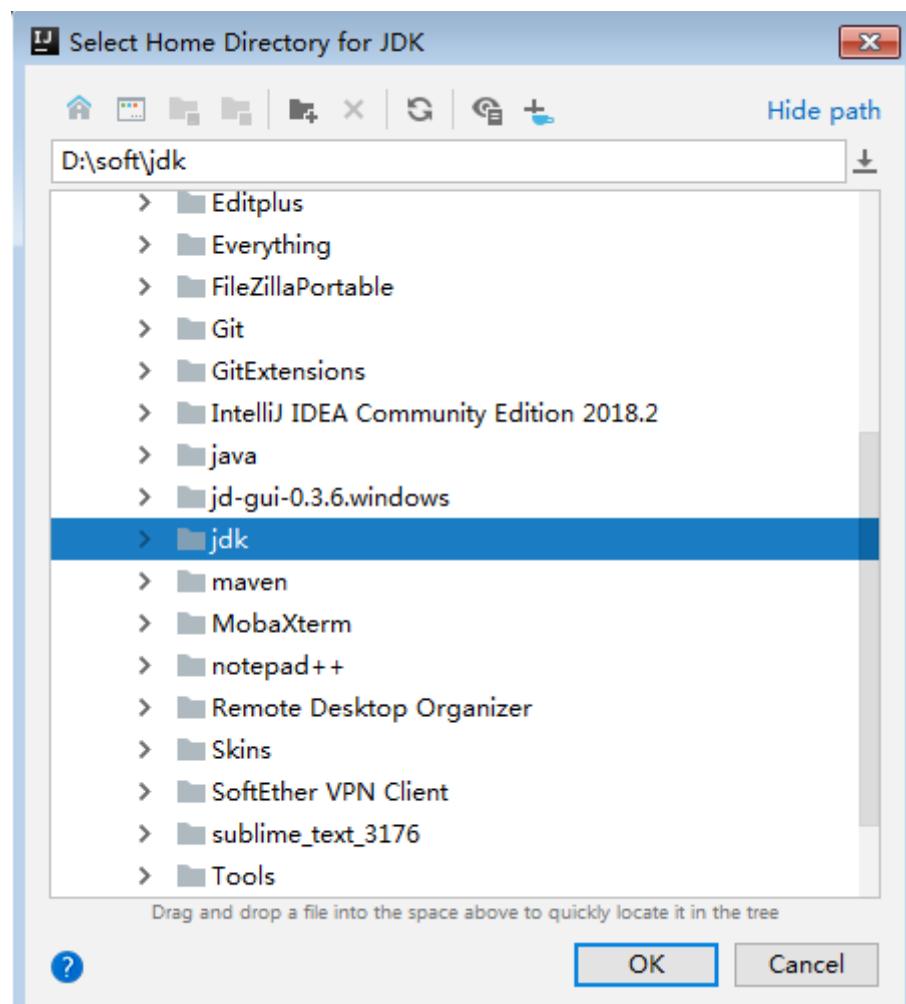
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 1-150 Project Structure for New Projects



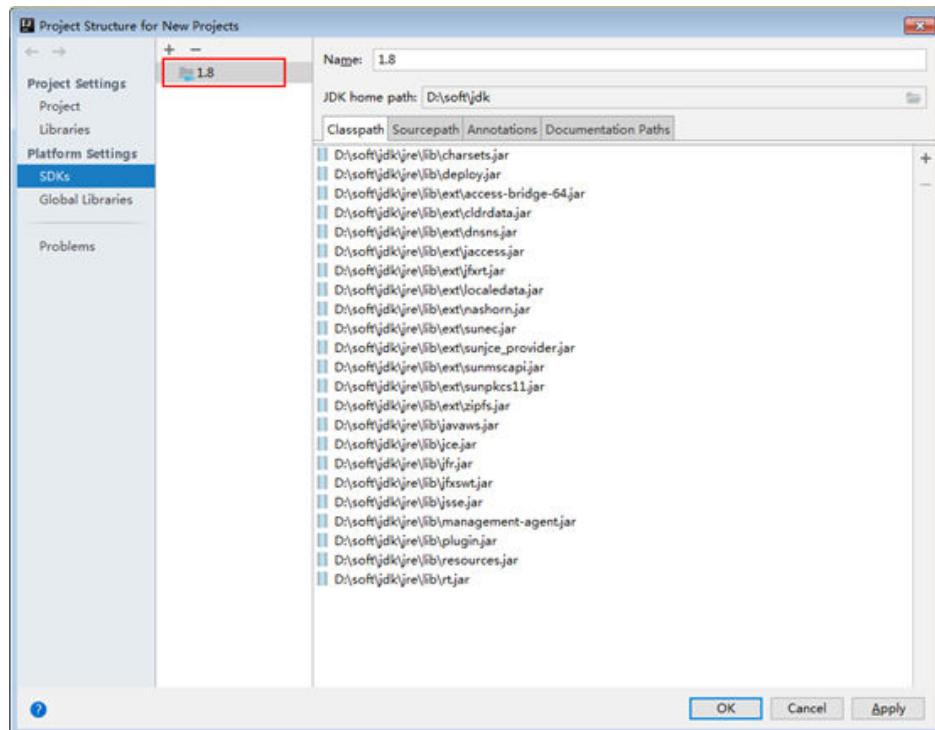
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 1-151 Select Home Directory for JDK



5. Click OK.

Figure 1-152 Completing the JDK configuration



**Step 4** Import the example project to the IntelliJ IDEA development environment.

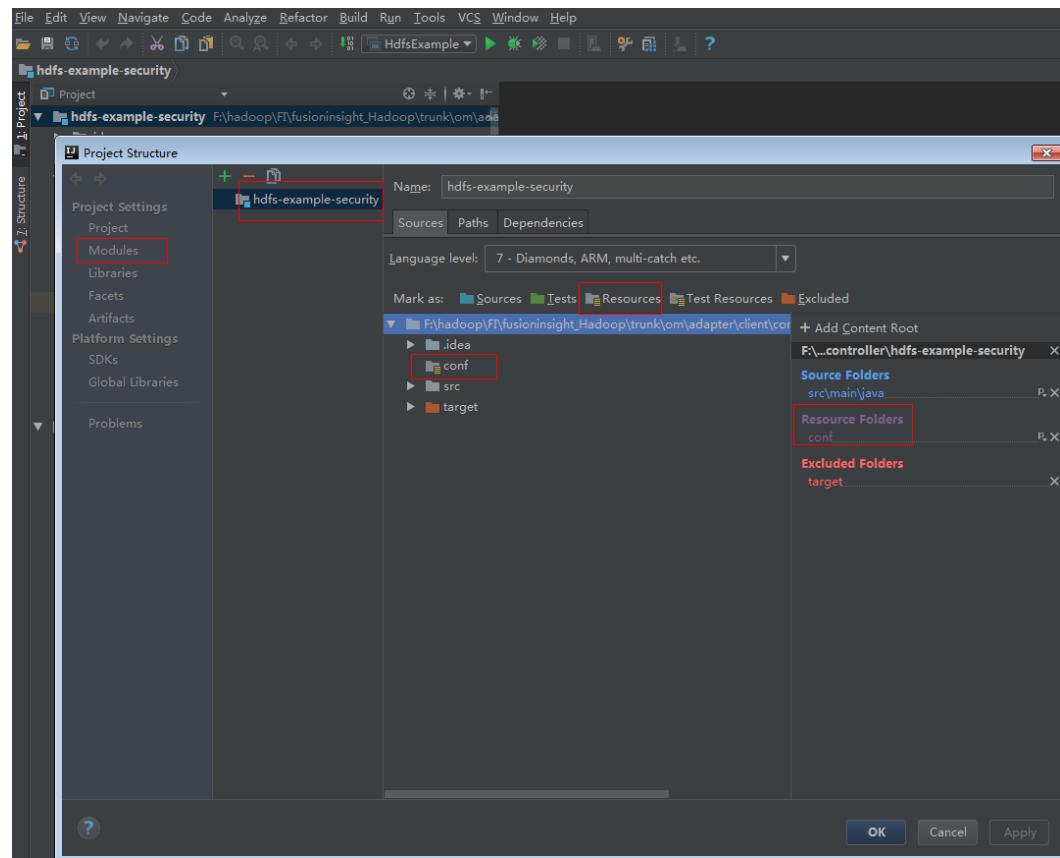
1. Open IntelliJ IDEA and choose **File > Open**.
2. Choose the directory of the example project **hdfs-example-security**. Click **OK**.

**Step 5** Add the related jar files the example project depending on into classpath.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured (for details, see [Configuring Huawei Open-Source Mirrors](#)).

**Step 6** Add the conf directory in the project to the resource path. On the menu bar of the IntelliJ IDEA, choose **File > Project Structure**. In the dialog box that is displayed, click **Modules**, select the current project and click **Resources > conf > OK** to set the resource directory. [Figure 1-153](#) shows the directory for setting the project resource.

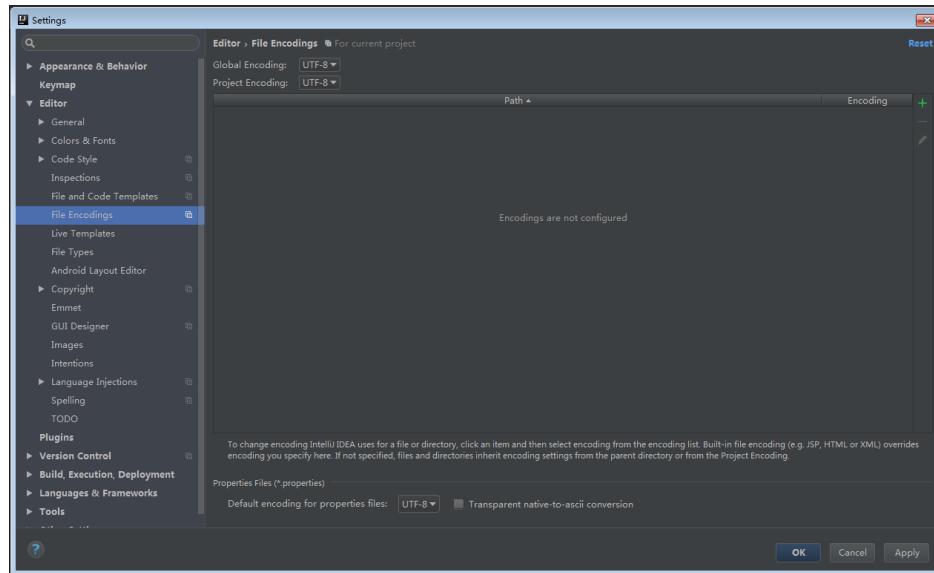
Figure 1-153 Setting the Project Resource Directory



**Step 7** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 1-154](#)

Figure 1-154 Setting the IntelliJ IDEA coding format



----End

### 1.11.2.3 Preparing the Authentication Mechanism

#### Scenario

Before accessing services in the secure cluster environment, you must be authorized by Kerberos. Codes for security authentication need to be written into the HDFS applications to ensure that the applications can work properly.

Two security authentication methods are described as follows:

- Authentication by running command lines:

Before submitting the HDFS application for running, run the following command in the HDFS client to obtain authentication:

**kinit component service user**

#### NOTE

This method applies only to the Linux OS that is installed with the HDFS client.

- Authentication by adding codes:

Authenticate by obtaining the principal and keytab files of the client.

Change the value of **PRINCIPAL\_NAME** in the code to the actual value.

```
private static final String PRINCIPAL_NAME = "hdfsDeveloper";
```

#### Safety Security Code

The safety authentication of the example codes is completed by invoking the LoginUtil class.

In the HDFS sample project code, different sample projects use different authentication codes which are basic safety authentication and the basic safety authentication with the ZooKeeper authentication.

- Basic safety authentication:

Sample projects of the **HdfsExample** class in the **com.huawei.bigdata.hdfs.examples** package need only the basic safety authentication codes because these sample projects do not need to access the HBase or ZooKeeper. Add the following codes in the program:

```
...
    private static final String PATH_TO_HDFS_SITE_XML =
HdfsExample.class.getClassLoader().getResource("hdfs-site.xml").getPath();
    private static final String PATH_TO_CORE_SITE_XML =
HdfsExample.class.getClassLoader().getResource("core-site.xml").getPath();
    private static final String PRINCIPAL_NAME = "hdfsDeveloper";
    private static final String PATH_TO_KEYTAB =
HdfsExample.class.getClassLoader().getResource("user.keytab").getPath();
    private static final String PATH_TO_KRB5_CONF =
HdfsExample.class.getClassLoader().getResource("krb5.conf").getPath();
    private static Configuration conf = null;
...
    private static void authentication() throws IOException {
        // security mode
        if ("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))) {
            System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);
            LoginUtil.login(PRINCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);
        }
    }
}
```

- Basic safety authentication with ZooKeeper Authentication:

Sample projects of the **ColocationExample** class in the **com.huawei.bigdata.hdfs.examples** package require not only the basic safety authentication, but also the Principal of the server in ZooKeeper to complete the safety authentication. Add the following codes in the program:

```
...
    private static final String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";
    private static final String PRINCIPAL = "username.client.kerberos.principal";
    private static final String KEYTAB = "username.client.keytab.file";
    private static final String PRINCIPAL_NAME = "hdfsDeveloper";
    private static final String LOGIN_CONTEXT_NAME = "Client";
    private static final String PATH_TO_KEYTAB = System.getProperty("user.dir") + File.separator +
"conf" + File.separator + "user.keytab";
    private static final String PATH_TO_KRB5_CONF =
ColocationExample.class.getClassLoader().getResource("krb5.conf").getPath();
    private static String zookeeperDefaultServerPrincipal = null;
    private static Configuration conf = new Configuration();
    private static DFSColocationAdmin dfsAdmin;
    private static DFSColocationClient dfs;
    private static void init() throws IOException {
        LoginUtil.login(PRINCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);
        LoginUtil.setJaasConf(LOGIN_CONTEXT_NAME, PRINCIPAL_NAME, PATH_TO_KEYTAB);
        zookeeperDefaultServerPrincipal = "zookeeper/hadoop." +
KerberosUtil.getKrb5DomainRealm().toLowerCase();
        LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
zookeeperDefaultServerPrincipal);
    }
...
}
```

#### NOTE

- The **HdfsDeveloper** user and the user's **user.keytab** and **krb5.conf** in the safety authentication codes are used as an example. In practical operations, contact the administrator to obtain the corresponding account and the keytab and krb5 files related to the account.
- You can log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**, which is the current system domain name.
- **zookeeper/hadoop.<system domain name>** is the user name. All letters in the system domain name contained in the user name of the system are lowercase letters. For example, if **Local domain** is set to **9427068F-6EFA-4833-B43E-60CB641E5B6C.COM**, the user name is **zookeeper/hadoop.9427068f-6efa-4833-b43e-60cb641e5b6c.com**.

## 1.11.3 Developing the Project

### 1.11.3.1 Scenario

#### Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the HDFS development process and know important interface functions.

#### Scenario

Service operation objects of HDFS are files. File operations in example codes include creating a folder, writing data into a file, appending data to a file, reading data from a file, and deleting a file or folder. HDFS also supports other services including setting file permission. You can learn how to perform other operations on HDFS after learning the example codes in this chapter.

The example codes are described in the following order:

1. Initializing the HDFS.
2. Creating directories.
3. Writing data into a file.
4. Appending data to a file.
5. Reading data from a file.
6. Deleting a file.
7. Deleting directories.
8. Multi-thread tasks.
9. Setting storage policies.
10. Colocation.

### 1.11.3.2 Development Idea

#### Development Idea

According to the previous scenario description, the following provides the basic operations for HDFS files with read, write, and delete operations on the **/user/hdfs-examples/test.txt** file as an example:

1. Pass the security certification.
2. Create a FileSystem object: fSystem.
3. Call the mkdir interface in fSystem to create a directory.
4. Call the create interface in fSystem to create an FSDataOutputStream object: out. Use the write method to write data into the object out.
5. Call the append interface in fSystem to create an FSDataOutputStream object: out. Use the write method to append data into the object out.
6. Call the open interface in fSystem to create an FSDataInputStream object: in. Use the read method to read files of the object in.
7. Call the delete interface in fSystem to delete the file.
8. Call the delete interface in fSystem to delete the folder.

### 1.11.3.3 Declare the Example Codes

#### 1.11.3.3.1 Initializing the HDFS

##### Function

Hadoop distributed file system (HDFS) initialization is a prerequisite for using application programming interfaces (APIs) provided by the HDFS. The process of initializing the HDFS is

1. Load the HDFS service configuration file.
2. Instantiate a FileSystem.



Obtain the keytab file for Kerberos security authentication in advance.

##### Configuration File Description

**Table 1-93** lists the configuration files to be used during the login to the HDFS. These files already imported to the **conf** directory of the **hdfs-example-security** project.

**Table 1-93** Configuration files

File	Function
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.

 NOTE

- Different clusters cannot share the same **user.keytab** and **krb5.conf** files.
- The **log4j.properties** file under the **conf** directory can be configured based on your needs.

## Example Codes

The following is code snippets. For complete codes, see the HdfsExample class in com.huawei.bigdata.hdfs.examples.

The initialization codes used when applications are run in Linux and the codes used when applications are run in Windows are the same. The example codes are as follows:

```
// Complete initialization and authentication.  
confLoad();  
authentication();  
// Creating a sample project  
HdfsExample hdfs_examples = new HdfsExample("/user/hdfs-examples", "test.txt");  
  
/**  
 *  
 * If the application is running in the Linux OS, the path of core-site.xml, hdfs-site.xml must be modified to  
the absolute path of the client file in the Linux OS.  
 *  
 */  
private static void confLoad() throws IOException {  
    conf = new Configuration();  
    // conf file  
    conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));  
    conf.addResource(new Path(PATH_TO_CORE_SITE_XML));  
}  
  
/**  
 *Safety authentication  
 */  
private static void authentication() throws IOException {  
    // security mode  
    if ("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))){  
        System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);  
        LoginUtil.login(PRINCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);  
    }  
}  
/**  
 *Create a sample project.  
 */  
public HdfsExample(String path, String fileName) throws IOException {  
    this.DEST_PATH = path;  
    this.FILE_NAME = fileName;  
    instanceBuild();  
}  
  
private void instanceBuild() throws IOException {  
    fSystem = FileSystem.get(conf);
```

The login example codes need to be added for the first login when applications are run in both Windows and Linux. For details on the example codes, see the LoginUtil class in com.huawei.hadoop.security.

```
public synchronized static void login(String userPrincipal, String userKeytabPath, String krb5ConfPath,  
Configuration conf)  
throws IOException  
{
```

```
// 1.Check the input parameters.

if ((userPrincipal == null) || (userPrincipal.length() <= 0))
{
    LOG.error("input userPrincipal is invalid.");
    throw new IOException("input userPrincipal is invalid.");
}

if ((userKeytabPath == null) || (userKeytabPath.length() <= 0))
{
    LOG.error("input userKeytabPath is invalid.");
    throw new IOException("input userKeytabPath is invalid.");
}

if ((krb5ConfPath == null) || (krb5ConfPath.length() <= 0))
{
    LOG.error("input krb5ConfPath is invalid.");
    throw new IOException("input krb5ConfPath is invalid.");
}

if ((conf == null))
{
    LOG.error("input conf is invalid.");
    throw new IOException("input conf is invalid.");
}

// 2.Check whether the file exists.

File userKeytabFile = new File(userKeytabPath);
if (!userKeytabFile.exists())
{
    LOG.error("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") does not exist.");
    throw new IOException("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") does not exist.");
}
if (!userKeytabFile.isFile())
{
    LOG.error("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") is not a file.");
    throw new IOException("userKeytabFile(" + userKeytabFile.getAbsolutePath() + ") is not a file.");
}

File krb5ConfFile = new File(krb5ConfPath);
if (!krb5ConfFile.exists())
{
    LOG.error("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") does not exist.");
    throw new IOException("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") does not exist.");
}
if (!krb5ConfFile.isFile())
{
    LOG.error("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") is not a file.");
    throw new IOException("krb5ConfFile(" + krb5ConfFile.getAbsolutePath() + ") is not a file.");
}

// 3.Set and check krb5config.

setKrb5Config(krb5ConfFile.getAbsolutePath());
setConfiguration(conf);

// 4.Log in to Hadoop to check items.

loginHadoop(userPrincipal, userKeytabFile.getAbsolutePath());
LOG.info("Login success!!!!!!!!!!!!!!!");
}
```

### 1.11.3.3.2 Creating Directories

#### Function

Process of creating a directory:

1. Call the exists method of the FileSystem instance to check whether the directory exists.
2. If yes, the method stops.
3. If no, call the mkdirs method in the FileSystem instance to create a directory.

## Example Codes

The following is a code snippet. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
/**  
 * Create a directory.  
 *  
 * @throws java.io.IOException  
 */  
private void mkdir() throws IOException {  
    Path destPath = new Path(DEST_PATH);  
    if (!createPath(destPath)) {LOG.error("failed to create destPath " + DEST_PATH);  
    return;  
}  
  
    LOG.info("success to create path " + DEST_PATH);  
}  
  
/**  
 * create file path  
 *  
 * @param filePath  
 * @return  
 * @throws java.io.IOException  
 */  
private boolean createPath(final Path filePath) throws IOException {  
    if (!fSystem.exists(filePath)) {  
        fSystem.mkdirs(filePath);  
    }  
    return true;  
}
```

### 1.11.3.3.3 Writing Data into a File

#### Function

The process of writing data into a file is

1. Use the create method in the FileSystem instance to obtain the output stream of writing files.
2. Uses this data stream to write content into a specified file in the HDFS.



Close all requested resources after writing files.

## Example Codes

The following is code snippets. For complete codes, see HdfsExample class in **com.huawei.bigdata.hdfs.examples**.

```
/**  
 * Create a file and write data into the file.  
 *
```

```
* @throws java.io.IOException
* @throws com.huawei.bigdata.hdfs.examples.ParameterException
*/
private void write() throws IOException {
    final String content = "hi, I am bigdata. It is successful if you can see me.";
    FSDataOutputStream out = null;
    try {
        out = fSystem.create(new Path(DEST_PATH + File.separator + FILE_NAME));
        out.write(content.getBytes());
        out.hsync();
        LOG.info("success to write.");
    } finally {
        // make sure the stream is closed finally.
        IOUtils.closeStream(out);
    }
}
```

#### 1.11.3.3.4 Appending Data to a File

### Function

Append data to a specified file in the Hadoop distributed file system (HDFS). The process of appending data to a file is

1. Use the append method in the FileSystem instance to obtain the output stream of the appended data.
2. Use the output stream to add the content to be appended behind the specified file in the HDFS.



Close all requested resources after appending data.

### Example Codes

The following is code snippets. For complete codes, see `HdfsExample` class in `com.huawei.bigdata.hdfs.examples`.

```
/*
 * Append data to a file
 *
 * @throws java.io.IOException
 */
private void append() throws IOException {
    final String content = "I append this content.";
    FSDataOutputStream out = null;
    try {
        out = fSystem.append(new Path(DEST_PATH + File.separator + FILE_NAME));
        out.write(content.getBytes());
        out.hsync();
        LOG.info("success to append.");
    } finally {
        // make sure the stream is closed finally.
        IOUtils.closeStream(out);
    }
}
```

### 1.11.3.3.5 Reading Data from a File

#### Function

Read data from a specified file in the Hadoop distributed file system (HDFS). The process is:

1. Use the open method in the FileSystem instance to obtain the input stream of writing files.
2. Use the input stream to read the content of the specified file in the HDFS.



Close all requested resources after reading files.

#### Example Codes

The following is code snippets. For complete codes, see the `HdfsExample` class in `com.huawei.bigdata.hdfs.examples`.

```
/**  
 * Read s file.  
 *  
 * @throws java.io.IOException  
 */  
private void read() throws IOException {  
    String strPath = DEST_PATH + File.separator + FILE_NAME;  
    Path path = new Path(strPath);  
    FSDataInputStream in = null;  
    BufferedReader reader = null;  
    StringBuffer strBuffer = new StringBuffer();  
    try {  
        in = fSystem.open(path);  
        reader = new BufferedReader(new InputStreamReader(in));  
        String sTempOneLine;  
        // write file  
        while ((sTempOneLine = reader.readLine()) != null) {  
            strBuffer.append(sTempOneLine);  
        }  
        LOG.info("result is : " + strBuffer.toString());  
        LOG.info("success to read.");  
    } finally {  
        // make sure the streams are closed finally.  
        IOUtils.closeStream(reader);  
        IOUtils.closeStream(in);  
    }  
}
```

### 1.11.3.3.6 Deleting a File

#### Function

Delete a file from the Hadoop distributed file system (HDFS).



Exercise caution when you delete files because the deletion is irreversible.

## Example Codes

The following is code snippets. For complete codes, see the `HdfsExample` class in `com.huawei.bigdata.hdfs.examples`.

```
/*
 * Delete a file.
 *
 * @throws java.io.IOException
 */
private void delete() throws IOException {
    Path beDeletedPath = new Path(DEST_PATH + File.separator + FILE_NAME);
    if (fSystem.delete(beDeletedPath, true)) {
        LOG.info("success to delete the file " + DEST_PATH + File.separator + FILE_NAME);
    } else {
        LOG.warn("failed to delete the file " + DEST_PATH + File.separator + FILE_NAME);
    }
}
```

### 1.11.3.3.7 Deleting Directories

## Function

Delete a specified directory from the HDFS.



Files in the deleted directory will be stored in the **Trash/Current** folder of the directory of the current user. In the event of mis-deletion, you can restore the folder from the directory.

## Example Codes

The following is a code snippet of file deletion. For complete codes, see the `HdfsExample` class in `com.huawei.bigdata.hdfs.examples`.

```
/*
 * delete the directory
 *
 * @throws java.io.IOException
 */
private void rmdir() throws IOException {
    Path destPath = new Path(DEST_PATH);
    if (!deletePath(destPath)) {
        LOG.error("failed to delete destPath " + DEST_PATH);
        return;
    }
    LOG.info("success to delete path " + DEST_PATH);
}

/*
 * delete file path
 *
 * @param filePath
 * @return
 * @throws java.io.IOException
 */
private boolean deletePath(final Path filePath) throws IOException {
    if (!fSystem.exists(filePath)) {
        return false;
    }
    // fSystem.delete(filePath, true);
    return fSystem.delete(filePath, true);
}
```

### 1.11.3.3.8 Multi-Thread Tasks

#### Function

Create multi-threaded tasks and initiate multiple instances to perform file operations.

#### Example Codes

The following is a code snippet of file deletion. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
// Service example 2: multi-thread tasks
final int THREAD_COUNT = 2;
for (int threadNum = 0; threadNum < THREAD_COUNT; threadNum++) {
    HdfsExampleThread example_thread = new HdfsExampleThread("hdfs_example_" + threadNum);
    example_thread.start();
}

class HdfsExampleThread extends Thread {
    private final static Log LOG = LogFactory.getLog(HdfsExampleThread.class.getName());
    /**
     *
     * @param threadName
     */
    public HdfsExampleThread(String threadName) {
        super(threadName);
    }
    public void run() {
        HdfsExample example;
        try {
            example = new HdfsExample("/user/hdfs-examples/" + getName(), "test.txt");
            example.test();
        } catch (IOException e) {
            LOG.error(e);
        }
    }
}
```

The **example.test()** method is the operation on files. The code is as follows:

```
/*
 * HDFS operation instance
 *
 * @throws IOException
 * @throws ParameterException
 *
 * @throws Exception
 */
public void test() throws IOException {
    // Create a directory.
    mkdir();

    // Write data into a file.
    write();

    // Append data to a file.
    append();

    // Read a file.
    read();

    // Delete a file.
    delete();
```

```
// Delete a directory.  
rmdir();  
}
```

### 1.11.3.3.9 Setting Storage Policies

#### Function

Specify storage policies for a file or folder in the HDFS.

#### Example Code

1. [Accessing FusionInsight Manager of an MRS Cluster](#), and choose **Cluster > Name of the desired cluster > Services > HDFS > Configurations > All Configurations**.
2. Check whether the value of **dfs.storage.policy.enabled** is the default value **true**. If not, modify the value to **true**, click **Save**, and restart HDFS.
3. Check the code.

The following code segment is only an example. For details, see the `HdfsExample` class in `com.huawei.bigdata.hdfs.examples`.

```
/**  
 * set storage policy to path  
 * @param policyName  
 * Policy Name can be accepted:  
 * <li>HOT  
 * <li>WARM  
 * <li>COLD  
 * <li>LAZY_PERSIST  
 * <li>ALL_SSD  
 * <li>ONE_SSD  
 * @throws IOException  
 */  
private void setStoragePolicy(String policyName) throws IOException {  
    if (fSystem instanceof DistributedFileSystem) {  
        DistributedFileSystem dfs = (DistributedFileSystem) fSystem;  
        Path destPath = new Path(DEST_PATH);  
        Boolean flag = false;  
        mkdir();  
        BlockStoragePolicySpi[] storage = dfs.getStoragePolicies();  
        for (BlockStoragePolicySpi bs : storage) {  
            if (bs.getName().equals(policyName)) {  
                flag = true;  
            }  
            LOG.info("StoragePolicy:" + bs.getName());  
        }  
        if (!flag) {  
            policyName = storage[0].getName();  
        }  
        dfs.setStoragePolicy(destPath, policyName);  
        LOG.info("success to set Storage Policy path " + DEST_PATH);  
        rmdir();  
    } else {  
        LOG.info("SmallFile not support to set Storage Policy !!!");  
    }  
}
```

### 1.11.3.3.10 Colocation

#### Function Description

Colocation is to store associated data or data on which associated operations are performed on the same storage node. The HDFS Colocation feature stores files on which associated operations are performed on the same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

Before using the Colocation function, you are advised to be familiar with the internal mechanisms of Colocation, including:

**Colocation node allocation principle**

**Capacity expansion and Colocation allocation**

**Colocation and data node capacity**

- **Colocation node allocation principle**

Colocation allocates data nodes to locators evenly according to the allocation node quantity.



The allocation algorithm principle is as follows: Colocation queries all locators, reads the data nodes allocated to the locators, and records the number of times. Based on the number of times, Colocation sorts the data nodes. The data nodes that are rarely used are placed at the beginning and selected first. The count increase by 1 each time after a node is selected. The nodes are shorted again, and the subsequent node will be selected.

- **Capacity expansion and Colocation allocation**

After cluster capacity expansion, you can select one of the following two policies shown in [Table 1-94](#) to balance the usage of data nodes and ensure that the allocation frequency of the newly added nodes is consistent with that of the old data nodes.

**Table 1-94** Allocation policies

No.	Policy	Description
1	Delete the original locators and create new locators for all data nodes in the cluster.	<ol style="list-style-type: none"><li>1. The original locator before the capacity expansion evenly uses all data nodes. After the capacity expansion, the newly added nodes are not allocated to existing locators, so Colocation stores data only to the old data nodes.</li><li>2. Data nodes are allocated to specific locators. Therefore, after capacity expansion, Colocation needs to reallocated data nodes to locators.</li></ol>

No.	Policy	Description
2	Create new locators and plan the data storage mode again.	The old locators use the old data nodes, while the newly created locators mainly use the new data nodes. Therefore, locators need to be planned again based on the actual service requirements on data.

 NOTE

Generally, you are advised to use the policy to reallocate data nodes to locators after capacity expansion to prevent data from being stored only to the new data nodes.

- **Colocation and data node capacity**

When Colocation is used to store data, the data is stored to the data node of a specified locator. If no locator planning is performed, the data node capacity will be uneven. [Table 1-95](#) summarizes the two usage principles to ensure even data node capacity.

**Table 1-95** Usage Principle

No.	Usage Principle	Description
1	All the data nodes are used in the same frequency in locators.	Assume that there are N data nodes, the number of locators must be an integral multiple of N (N, 2 N, ...).
2	A proper data storage plan must be made for all locators to ensure that data is evenly stored in the locators.	None

During HDFS secondary development, you can obtain the DFSColocatinAdmin and DFSColocatinClient instances to create groups, delete groups, write files, and delete files in or from the location.

 NOTE

- When the Colocation function is enabled and users specify DataNodes, the data volume will be large on some nodes. Serious data skew will result in HDFS data write failures.
- Because of data skew, MapReduce accesses only several nodes. In this case, the load is heavy on these nodes, while other nodes are idle.
- For a single application task, the DFSColocatinAdmin and DFSColocatinClient instances can be used only once. If the instances are used for many time, excessive HDFS links will be created and use up HDFS resources.
- If you need to perform the balance operation for a file uploaded by colocation, you can set the `oi.dfs.colocation.file.pattern` parameter on FusionInsight Manager to the file path to avoid invalid colocation. If there are multiple files, use commas (,) to separate the file paths, for example, `/test1,/test2`.
- Colocation stores associated data or data on which associated operations are performed on the same storage. When Balancer- or Mover-related operations are performed, data blocks will be moved. As a result, the Colocation function becomes unavailable. Therefore, when using the Colocation function, you are advised to set the HDFS configuration item `dfs.datanode.block-pinning.enabled` to `true`. In this case, files written by Colocation will not be moved when Balancer- or Mover-related operations are performed in the cluster, ensuring file colocation.

## Example Codes

For complete example codes, see  
`com.huawei.bigdata.hdfs.examples.ColocationExample`.

 NOTE

- Before using the Colocation function, add HDFS users to the supergroup group.
- When the Colocation project is run, the HDFS parameter `fs.defaultFS` cannot be set to `viewfs://ClusterX`.

### 1. Initialization

Kerberos security authentication is required before using Colocation.

```
private static void init() throws IOException {  
  
    conf.set(KEYTAB, PATH_TO_KEYTAB);  
    conf.set(PRINCIPAL, PRNCIPAL_NAME);  
  
    LoginUtil.setJaasConf(LOGIN_CONTEXT_NAME, PRNCIPAL_NAME, PATH_TO_KEYTAB);  
    LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,  
    ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);  
    LoginUtil.login(PRNCIPAL_NAME, PATH_TO_KEYTAB, PATH_TO_KRB5_CONF, conf);  
}
```

### 2. Obtain instances.

Example: Colocation operations require the DFSColocatinAdmin and DFSColocatinClient instances. Therefore, the instances must be obtained before operations, such as creating a group.

```
dfsAdmin = new DFSColocatinAdmin(conf);  
dfs = new DFSColocatinClient();  
dfs.initialize(URI.create(conf.get("fs.defaultFS")), conf);
```

### 3. Create a group.

Example: Create a `gid01` group, which contains three locators.

```
/**  
 * create group  
 *  
 * @throws java.io.IOException
```

```
/*
private static void createGroup() throws IOException {
    dfsAdmin.createColocationGroup(COLOCATION_GROUP_GROUP01,
        Arrays.asList(new String[] { "lid01", "lid02", "lid03" }));
}
```

4. Write data into a file. The related group must be created before writing data into the file.

Example: Write data into the testfile.txt file.

```
/**
 * create and write file
 *
 * @throws java.io.IOException
 */
private static void put() throws IOException {
    FSDatOutputStream out = dfs.create(new Path(TESTFILE_TXT), true,
COLOCATION_GROUP_GROUP01, "lid01");
    // the data to be written to the hdfs.
    byte[] readBuf = "Hello World".getBytes("UTF-8");
    out.write(readBuf, 0, readBuf.length);
    out.close();
}
```

5. Delete a file.

Example: Delete the testfile.txt file.

```
/**
 * delete file
 *
 * @throws java.io.IOException
 */
@SuppressWarnings("deprecation")
private static void delete() throws IOException {
    dfs.delete(new Path(TESTFILE_TXT));
}
```

6. Delete a group.

Example: Delete gid01.

```
/**
 * delete group
 *
 * @throws java.io.IOException
 */
private static void deleteGroup() throws IOException {
    dfsAdmin.deleteColocationGroup(COLOCATION_GROUP_GROUP01);
}
```

## 1.11.4 Commissioning the Application

### 1.11.4.1 Commissioning an Application in the Windows Environment

#### 1.11.4.1.1 Compiling and Running an Application

##### Scenario

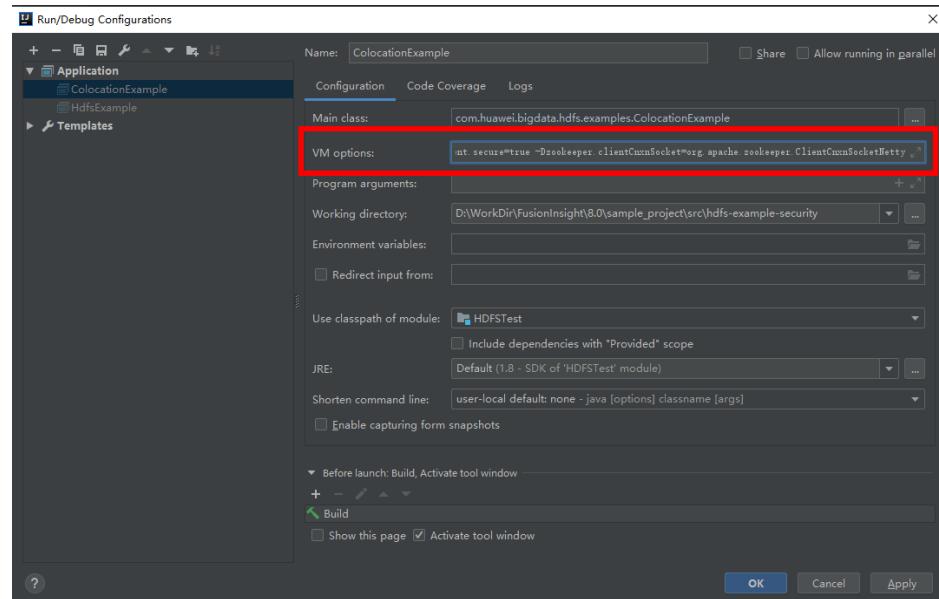
After the code development is complete, you can run an application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

## Procedure

**Step 1** If SSL is enabled for the ZooKeeper in the cluster, the **ColocationExample sample** code needs to access the ZooKeeper. Therefore, you need to set the JVM parameter `-Dzookeeper.client.secure=true -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty` in the **Idea** compilation tool before running `ColocationExample.java`.

As shown in the following figure:



**Step 2** In a development environment (such as IntelliJ IDEA), choose the following two projects separately and run the projects:

- Choose **HdfsExample.java** and right-click the project and choose **Run 'HdfsExample.main()'** from the shortcut menu to run the project.
- Choose **ColocationExample.java** and right-click the project and choose **Run 'ColocationExample.main()'** from the shortcut menu to run the project.

### NOTE

- Do not restart HDFS service while HDFS application is in running status, otherwise the application will fail.
- When the Colocation project is run, the HDFS parameter `fs.defaultFS` cannot be set to `viewfs://ClusterX`.

----End

## Note

During security authentication, Hadoop needs to obtain the domain name of the host where the client is located (**Default Realm**, which is obtained from environment variable **USERDNSDOMAIN**). If the host does not have a domain name, the following error message is displayed when you run the sample program:

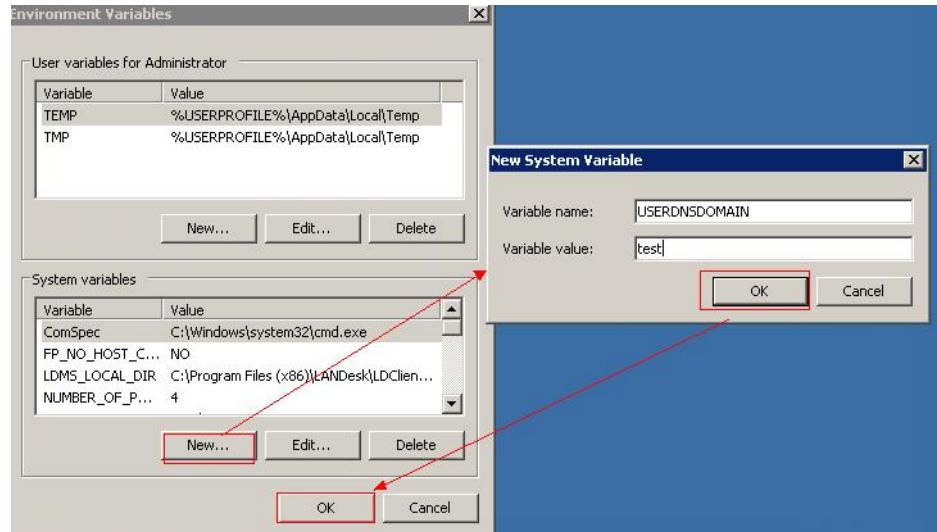
```

Exception in thread "main" java.lang.IllegalArgumentException: Can't get Kerberos realm
at org.apache.hadoop.security.HadoopKerberosName.setConfiguration(HadoopKerberosName.java:65)
at org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.java:288)
at org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformation.java:273)
at org.apache.hadoop.security.UserGroupInformation.loginUserFromSubject(UserGroupInformation.java:832)
at org.apache.hadoop.security.UserGroupInformation.getLoginUser(UserGroupInformation.java:802)
at org.apache.hadoop.security.UserGroupInformation.getCurrentUser(UserGroupInformation.java:675)
at org.apache.hadoop.conf.Configuration$Resource.getRestrictParserDefault(Configuration.java:243)
at org.apache.hadoop.conf.Configuration$Resource.<init>(Configuration.java:211)
at org.apache.hadoop.conf.Configuration$Resource.<init>(Configuration.java:203)
at org.apache.hadoop.conf.Configuration.addResource(Configuration.java:851)
at com.huawei.bigdata.hdfs.examples.HdfsExample.confLoad(HdfsExample.java:307)
at com.huawei.bigdata.hdfs.examples.HdfsExample.main(HdfsExample.java:277)
Caused by: java.lang.reflect.InvocationTargetException
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
at java.lang.reflect.Method.invoke(Unknown Source)
at org.apache.hadoop.security.authentication.util.KerberosUtil.getDefaultRealm(KerberosUtil.java:88)
at org.apache.hadoop.security.HadoopKerberosName.setConfiguration(HadoopKerberosName.java:63)
... 11 more
Caused by: KrbException: Cannot locate default realm
at sun.security.krb5.Config.getDefaultRealm(Unknown Source)
... 17 more

```

You can set environment variable **USERDNSDOMAIN** of the system to prevent this problem. The details are as follows:

1. Right-click **Computer** and choose **Properties** from the shortcut menu. The dialog box shown in the following figure is displayed. Click **Advanced system settings > Advanced > Environment Variables**.
2. Set the system environment variable and click "New". The New System Variable dialog box is displayed. Enter **USERDNSDOMAIN** in Variable name and set the Variable value to a non-null character string, for example, test. Click OK twice. The system environment variable is set.



3. Close the sample project, open it again, and run it.

#### 1.11.4.1.2 Checking the Commissioning Result

##### Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

##### Procedure

- Learn the application running conditions by viewing the running result.

- The running result of the HDFS windows example application is shown as follows:

```
1308 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user hdfsDeveloper using keytab file
1308 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!
2040 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-circuit local reads feature cannot be used because UNIX Domain sockets are not available on Windows.
3006 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples
3131 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
3598 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
4408 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
5015 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am bigdata. It is successful if you can see me.I append this content.
5015 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.
5077 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file /user/hdfs-examples\test.txt
5186 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /user/hdfs-examples
5311 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples/hdfs_example_0
5311 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples/hdfs_example_1
5669 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
5669 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
7258 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
7741 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
7896 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am bigdata. It is successful if you can see me.I append this content.
7896 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.
7959 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file /user/hdfs-examples/hdfs_example_1\test.txt
8068 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /user/hdfs-examples/hdfs_example_1
8364 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am bigdata. It is successful if you can see me.I append this content.
8364 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.
8426 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file /user/hdfs-examples/hdfs_example_0\test.txt
8535 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /user/hdfs-examples/hdfs_example_0
```

### NOTE

In the Windows environment, the following exception occurs but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

- The running result of the Colocation windows example application is shown as follows:

```
...
945 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user hdfsDeveloper using keytab file user.keytab
945 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!
945 [main] INFO com.huawei.hadoop.security.LoginUtil - JaasConfiguration loginContextName=Client principal=hdfsDeveloper useTicketCache=false keytabFile=XXX \sample_project\src\hdfs-example-security\conf\user.keytab
946 [main] INFO com.huawei.hadoop.security.KerberosUtil - Get default realm successfully, the realm is : HADOOP.COM
```

```
...
Create Group has finished.
Put file is running...
Put file has finished.
Delete file is running...
Delete file has finished.
Delete Group is running...
Delete Group has finished.
4946 [main-EventThread] INFO org.apache.zookeeper.ClientCnxn - EventThread shut down for
connection: 0x440751cb41a4d415
4946 [main] INFO org.apache.zookeeper.ZooKeeper - Connection: 0x440751cb41a4d415 closed
...
```

- **Learn the application running conditions by viewing HDFS logs.**  
The NameNode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 1.11.4.2 Commissioning an Application in the Linux Environment

### 1.11.4.2.1 Compiling and Running an Application With the Client Installed

#### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client installed. After the application code has been developed, you can upload the jar packages to the HDFS client and run the application.

#### Prerequisite

- The HDFS client has been installed.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the client is installed. The host name must be correctly mapped to the IP address.

#### Procedure

**Step 1** Go to the local root directory of the project, copy the required configuration file to the conf folder of the local project, and run the following command in Windows cmd to compress the package:

**mvn -s "{maven\_setting\_path}" clean package**

##### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package, for example, **HDFSTest-XXX.jar**, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

**Step 2** Upload the exported jarpackages to any directory in the running environment of the client, for example, **/opt/hadoopclient**.

**Step 3** Configure the environment variables:

**cd /opt/hadoopclient**

**source bigdata\_env**

**Step 4** Run the following commands to execute the jar packages.

**hadoop jar HDFSTest-XXX.jar com.huawei.bigdata.hdfs.examples.HdfsExample**

**hadoop jar HDFSTest-XXX.jar  
com.huawei.bigdata.hdfs.examples.ColocationExample**



When **com.huawei.bigdata.hdfs.examples.ColocationExample** is run, the HDFS parameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

#### 1.11.4.2.2 Compiling and Running an Application With the Client Not Installed

##### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client not installed. After the application code has been developed, you can upload the jar packages to the Linux OS and run the application.

##### Prerequisite

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by IDEA to export the JAR package.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

##### Procedure

**Step 1** Go to the local root directory of the project, copy the required configuration file to the conf folder of the local project and run the following command in Windows cmd to compress the package:

**mvn -s "{maven\_setting\_path}" clean package**



- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package from the target subdirectory in the root directory of the project.

**Step 2** Upload the exported jar packages to any directory in the running environment of the Linux OS, for example, **/opt/hadoop\_client**.

**Step 3** Upload the **lib** and **conf** folders of the project to the same directory that stores the jar packages (the **lib** Create a **lib** folder in the Linux operating environment directory (for example, **/opt/hadoopclient**) and upload the required **JAR**

packages. The **lib** folder contains all the JAR packages that the project depends on. For details, see section [Preparing an Operating Environment](#).

**Step 4** Run the following commands to execute the jar packages.

```
java -cp HDFSTest-XXX.jar:conf/:lib/*
com.huawei.bigdata.hdfs.examples.HdfsExample
```

```
java -cp HDFSTest-XXX.jar:conf/:lib/*
com.huawei.bigdata.hdfs.examples.ColocationExample
```

#### NOTE

When **com.huawei.bigdata.hdfs.examples.ColocationExample** is run, the HDFS parameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

If SSL is enabled for the ZooKeeper in the cluster, the sample code **ColocationExample** needs to access the ZooKeeper. Therefore, you need to add ZooKeeper SSL-related parameters to the command line.

```
java -cp HDFSTest-XXX.jar:conf/:lib/* -Dzookeeper.client.secure=true -
Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty
com.huawei.bigdata.hdfs.examples.ColocationExample
```

----End

### 1.11.4.2.3 Checking the Commissioning Result

#### Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

#### Procedure

- **Learn the application running conditions by viewing the running result.**

- The running result of the HDFS example application is shown as follows:  
0 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user hdfsDevelop using keytab file user:keytab  
1 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!  
568 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
582 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.  
793 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path / user/hdfs-examples  
969 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.  
1068 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.  
1191 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am bigdata. It is successful if you can see me.I append this content.  
1191 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.  
1202 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file / user/hdfs-examples/test.txt  
1210 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path / user/hdfs-examples  
1223 [hdfs\_example\_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples/hdfs\_example\_0  
1224 [hdfs\_example\_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples/hdfs\_example\_1  
1261 [hdfs\_example\_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.  
1264 [hdfs\_example\_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.  
2807 [hdfs\_example\_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to

```
append.  
2810 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
append.  
2861 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I  
am bigdata. It is successful if you can see me.I append this content.  
2861 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
read.  
2866 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete the file /user/hdfs-examples/hdfs_example_0/test.txt  
2874 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete path /user/hdfs-examples/hdfs_example_0  
2874 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I  
am bigdata. It is successful if you can see me.I append this content.  
2874 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
read.  
2879 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete the file /user/hdfs-examples/hdfs_example_1/test.txt  
2885 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete path /user/hdfs-examples/hdfs_example_1
```

- The running result of the Colocation example application is shown as follows:

```
0 [main] INFO com.huawei.hadoop.security.LoginUtil - JaasConfiguration  
loginContextName=Client principal=hdfsDevelop useTicketCache=false keytabFile=/opt/  
hdfsDemo/conf/user.keytab  
817 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user  
hdfsDevelop using keytab file user.keytab  
817 [main] INFO com.huawei.hadoop.security.LoginUtil - Login success!!!!!!!!!!!!!!  
1380 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop  
library for your platform... using builtin-java classes where applicable  
...  
Create Group has finished.  
Put file is running...  
Put file has finished.  
Delete file is running...  
Delete file has finished.  
Delete Group is running...  
Delete Group has finished.  
...
```

- **Learn the application running conditions by viewing HDFS logs.**

The namenode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 1.11.5 More Information

### 1.11.5.1 Common API Introduction

#### 1.11.5.1.1 Java API

For details about Hadoop distributed file system (HDFS) APIs, see:

<http://hadoop.apache.org/docs/r3.3.1/api/index.html>

#### HDFS Common API

Common HDFS Java classes are as follows:

- **FileSystem:** the core class of client applications. For details about common APIs, see [Table 1-96](#).
- **FileStatus:** record the status of files and directories. For details about common APIs, see [Table 1-97](#).

- DFSColocationAdmin: API used to manage colocation group information. For details about common APIs, see [Table 1-98](#).
- DFSColocationClient: API used to manage colocation files. For details about common APIs, see [Table 1-99](#).

 NOTE

- The system reserves only the mapping between nodes and locator IDs, but does not reserve the mapping between files and locator IDs. When a file is created using a Colocation interface, the file is created on the node that corresponds to a locator ID. File creation and writing must be performed using Colocation interfaces.
- After the file is written, subsequent operations on the file can use other open-source interfaces in addition to Colocation interfaces.
- The DFSColocationClient class inherits from the open-source DistributedFileSystem class and contains common file operation functions. If a user uses the DFSColocationClient class to create a Colocation file, the user is advanced to use the functions of this class in file operations.

**Table 1-96** Common FileSystem APIs

API	Description
public static FileSystem get(Configuration conf)	FileSystem is the API class provided for users in the Hadoop class library. FileSystem is an abstract class. Concrete classes can be obtained only using the get method. The get method has multiple overload versions and is commonly used.
public FSDataOutputStream create(Path f)	This API is used to create files in the HDFS. <i>f</i> indicates a complete file path.
public void copyFromLocalFile(Pat h src, Path dst)	This API is used to upload local files to a specified directory in the HDFS. <i>src</i> and <i>dst</i> indicate complete file paths.
public boolean mkdirs(Path f)	This API is used to create folders in the HDFS. <i>f</i> indicates a complete folder path.
public abstract boolean rename(Path src, Path dst)	This API is used to rename a specified HDFS file. <i>src</i> and <i>dst</i> indicate complete file paths.
public abstract boolean delete(Path f, boolean recursive)	This API is used to delete a specified HDFS file. <i>f</i> indicates the complete path of the file to be deleted, and <b>recursive</b> specifies recursive deletion.
public boolean exists(Path f)	This API is used to query a specified HDFS file. <i>f</i> indicates a complete file path.
public FileStatus getFileStatus(Path f)	This API is used to obtain the FileStatus object of a file or folder. The FsStatus object records status information of the file or folder, including the modification time and file directory.

API	Description
public BlockLocation[] getFileBlockLocations(FileStatus file, long start, long len)	This API is used to query the block location of a specified file in an HDFS cluster. <i>file</i> indicates a complete file path, and <i>start</i> and <i>len</i> specify the block scope.
public FSDataInputStream open(Path f)	This API is used to open the output stream of a specified file in the HDFS and read the file using the API provided by the FSDataInputStream class. <i>f</i> indicates a complete file path.
public FSDataOutputStream create(Path f, boolean overwrite)	This API is used to create the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class. <i>f</i> indicates a complete file path. If <b>overwrite</b> is <b>true</b> , the file is rewritten if it exists; if <b>overwrite</b> is <b>false</b> , an error is reported if the file exists.
public FSDataOutputStream append(Path f)	This API is used to open the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class. <i>f</i> indicates a complete file path.

**Table 1-97** Common FileStatus APIs

API	Description
public long getModificationTime()	This API is used to query the modification time of a specified HDFS file.
public Path getPath()	This API is used to query all files in an HDFS directory.

**Table 1-98** Common DFSColocationAdmin APIs

API	Description
public Map<String, List<DatanodeInfo>> createColocationGroup(String groupId, String file)	This API is used to create a group based on the locatorIds information in the file. <i>file</i> indicates the file path.
public Map<String, List<DatanodeInfo>> createColocationGroup(String groupId, List<String> locators)	This API is used to create a group based on the locatorIds information in the list in the memory.
public void deleteColocationGroup(String groupId)	This API is used to delete a group.

API	Description
public List<String> listColocationGroups()	This API is used to return all group information of Colocation. The returned group ID arrays are sorted by the creation time.
public List<DatanodeInfo> getNodesForLocator(String groupID, String locatorID)	This API is used to obtain the list of all nodes in the locator.

**Table 1-99** Common DFSColocationAdmin APIs

API	Description
public FSDataOutputStream create(Path f, boolean overwrite, String groupID, String locatorID)	This API is used to create a FSDataOutputStream in colocation mode to allow users to write files in f.  f is the HDFS path.  overwrite indicates whether an existing file can be overwritten.  groupID and locatorID of the file specified by a user must exist.
public FSDataOutputStream create(final Path f, final FsPermission permission, final EnumSet<CreateFlag> cflags, final int bufferSize, final short replication, final long blockSize, final Progressable progress, final ChecksumOpt checksumOpt, final String groupID, final String locatorID)	The function of this API is the same as that of FSDataOutputStream create(Path f, boolean overwrite, String groupID, String locatorID), except that users are allowed to customize checksum.
public void close()	This API is used to close the connection.

**Table 1-100** HDFS client WebHdfsFileSystem API

API	Description
public Remotelterator<FileSta- tus> listStatusItera- tor(final Path)	This API will help in fetching the child files and folders information through multiple requests using remote iterator, thus avoiding the user interface from becoming slow when there is a large amount of child information to be fetched.

## Glob path pattern based API to get LocatedFileStatus and Open file from FileStatus

Following APIs are added in DistributedFileSystem to get the FileStatus with block location and open file from FileStatus object. These APIs will reduce the number of RPC calls from client to Namenodes.

**Table 1-101** FileSystem APIs

Interface	Description
public LocatedFileStatus[] globLocatedStatus(Path, PathFilter, boolean) throws IOException	Return an array of LocatedFileStatus objects whose path names match pathPattern and pass the in path filter.
public FSDataInputStream open(FileStatus stat) throws IOException	If the stat is an instance of LocatedFileStatusHdfs that already have the location information, the InputStream is created without contacting NameNode.

### 1.11.5.1.2 C API

#### Function Description

Users can use the C application programming interface (API) to create, read and write, append, and delete files. For details of the C API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/LibHdfs.html>

#### Code Sample

The following code snippets are used as an example. For complete code, see the HDFS C sample code **HDFS/hdfs-c-example/hdfs\_test.c** in the HDFS sample code decompression directory.

1. Configure the HDFS NameNode parameter and create the link connecting to the HDFS files.

```
hdfsFS fs = hdfsConnect("default", 0);
fprintf(stderr, "hdfsConnect- SUCCESS!\n");
```

2. Create the HDFS directory.

```
const char* dir = "/tmp/nativeTest";
int exitCode = hdfsCreateDirectory(fs, dir);
if( exitCode == -1 ){
    fprintf(stderr, "Failed to create directory %s \n", dir );
    exit(-1);
}
fprintf(stderr, "hdfsCreateDirectory- SUCCESS! : %s\n", dir);
```

3. Write files.

```
const char* file = "/tmp/nativeTest/testfile.txt";
hdfsFile writeFile = openFile(fs, (char*)file, O_WRONLY | O_CREAT, 0, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for write : %s\n", file);
```

```

if(!hdfsFileIsOpenForWrite(writeFile)){
    fprintf(stderr, "Failed to open %s for writing.\n", file);
    exit(-1);
}

char* buffer = "Hadoop HDFS Native file write!";

hdfsWrite(fs, writeFile, (void*)buffer, strlen(buffer)+1);
fprintf(stderr, "hdfsWrite- SUCCESS! : %s\n", file);

printf("Flushing file data ....\n");
if (hdfsFlush(fs, writeFile)) {
    fprintf(stderr, "Failed to 'flush' %s\n", file);
    exit(-1);
}
hdfsCloseFile(fs, writeFile);
fprintf(stderr, "hdfsCloseFile- SUCCESS! : %s\n", file);

```

**4. Read files.**

```

hdfsFile readFile = openFile(fs, (char*)file, O_RDONLY, 100, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for read : %s\n", file);

if(!hdfsFileIsOpenForRead(readFile)){
    fprintf(stderr, "Failed to open %s for reading.\n", file);
    exit(-1);
}

buffer = (char *) malloc(100);
tSize num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsRead- SUCCESS!, Byte read : %d, File content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);

```

**5. From the specified location to read the file.**

```

buffer = (char *) malloc(100);
readFile = openFile(fs, file, O_RDONLY, 100, 0, 0);
if (hdfsSeek(fs, readFile, 10)) {
    fprintf(stderr, "Failed to 'seek' %s\n", file);
    exit(-1);
}
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsSeek- SUCCESS!, Byte read : %d, File seek content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);

```

**6. Copy the file.**

```

const char* destfile = "/tmp/nativeTest/testfile1.txt";
if (hdfsCopy(fs, file, fs, destfile)) {
    fprintf(stderr, "File copy failed, src : %s, des : %s \n", file, destfile);
    exit(-1);
}
fprintf(stderr, "hdfsCopy- SUCCESS!, File copied, src : %s, des : %s \n", file, destfile);

```

**7. Move the file.**

```

const char* mvfile = "/tmp/nativeTest/testfile2.txt";
if (hdfsMove(fs, destfile, fs, mvfile )) {
    fprintf(stderr, "File move failed, src : %s, des : %s \n", destfile , mvfile);
    exit(-1);
}
fprintf(stderr, "hdfsMove- SUCCESS!, File moved, src : %s, des : %s \n", destfile , mvfile);

```

**8. Rename the file.**

```

const char* renamefile = "/tmp/nativeTest/testfile3.txt";
if (hdfsRename(fs, mvfile, renamefile)) {
    fprintf(stderr, "File rename failed, Old name : %s, New name : %s \n", mvfile, renamefile);
    exit(-1);
}
fprintf(stderr, "hdfsRename- SUCCESS!, File renamed, Old name : %s, New name : %s \n", mvfile, renamefile);

```

**9. Delete Files.**

```

if (hdfsDelete(fs, renamefile, 0)) {
    fprintf(stderr, "File delete failed : %s \n", renamefile);
    exit(-1);
}

```

```

    }
    fprintf(stderr, "hdfsDelete- SUCCESS!, File deleted : %s\n", renamefile);
}

10. Set the number of replications.
if (hdfsSetReplication(fs, file, 10)) {
    fprintf(stderr, "Failed to set replication : %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsSetReplication- SUCCESS!, Set replication 10 for %s\n",file);

11. Set users, user groups.
if (hdfsChown(fs, file, "root", "root")) {
    fprintf(stderr, "Failed to set chown : %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsChown- SUCCESS!, Chown success for %s\n",file);

12. Set permissions.
if (hdfsChmod(fs, file, S_IRWXU | S_IRWXG | S_IROTH)) {
    fprintf(stderr, "Failed to set chmod: %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsChmod- SUCCESS!, Chmod success for %s\n",file);

13. Set the file time.
struct timeval now;
gettimeofday(&now, NULL);
if (hdfsUtime(fs, file, now.tv_sec, now.tv_usec)) {
    fprintf(stderr, "Failed to set time: %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsUtime- SUCCESS!, Set time success for %s\n",file);

14. Get file information.
hdfsFileInfo *fileInfo = NULL;
if((fileInfo = hdfsGetPathInfo(fs, file)) != NULL) {
    printFileInfo(fileInfo);
    hdfsFreeFileInfo(fileInfo, 1);
    fprintf(stderr, "hdfsGetPathInfo - SUCCESS!\n");
}

15. Variable directory.
hdfsFileInfo *fileList = 0;
int numEntries = 0;
if((fileList = hdfsListDirectory(fs, dir, &numEntries)) != NULL) {
    int i = 0;
    for(i=0; i < numEntries; ++i) {
        printFileInfo(fileList+i);
    }
    hdfsFreeFileInfo(fileList, numEntries);
}
fprintf(stderr, "hdfsListDirectory- SUCCESS!, %s\n", dir);

16. Stream builder interfaces.
buffer = (char *) malloc(100);
struct hdfsStreamBuilder *builder= hdfsStreamBuilderAlloc(fs, (char*)file, O_RDONLY);
hdfsStreamBuilderSetBufferSize(builder,100);
hdfsStreamBuilderSetReplication(builder,20);
hdfsStreamBuilderSetDefaultBlockSize(builder,10485760);
readFile = hdfsStreamBuilderBuild(builder);
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : %d, File content : %s
\n", num_read ,buffer);
free(buffer);

struct hdfsReadStatistics *stats = NULL;
hdfsFileGetReadStatistics(readFile, &stats);
fprintf(stderr, "hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : %" PRId64 ",
totalLocalBytesRead : %" PRId64 ", totalShortCircuitBytesRead : %" PRId64 ",
totalZeroCopyBytesRead : %" PRId64 "\n", stats->totalBytesRead , stats->totalLocalBytesRead, stats-
>totalShortCircuitBytesRead, stats->totalZeroCopyBytesRead);
hdfsFileFreeReadStatistics(stats);

```

17. Disconnect the HDFS links.  
`hdfsDisconnect(fs);`

## Preparing Running Environment

Install a client on the node. For example, install a client in the **/opt/hadoopclient** directory.

## Compiling and Running Applications in Linux

1. Go to the **/opt/hadoopclient** directory and run the following command to import the environment variables of the C client:

```
cd /opt/hadoopclient  
source bigdata_env
```

2. In the directory, run the following command as the hdfs user. For the user password, contact the cluster administrator.

```
kinit hdfs
```



The validity duration of kinit authentication is 24 hours. After 24 hours, you need to re-authenticate the sample with the kinit to restart the sample.

3. Go to the **/opt/hadoopclient/HDFS/hadoop/hdfs-c-example** directory and run the following command to import the environment variables of the C client:

```
cd /opt/hadoopclient/HDFS/hadoop/hdfs-c-example  
source component_env_C_example
```

4. Run the following command to clean the object files and executable files that are generated before:

```
make clean
```

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make clean  
rm -f hdfs_test.o  
rm -f hdfs_test
```

5. Run the following command to compile the new object files and executable files:

```
make (or make all)
```

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make  
cc -c -I/opt/hadoopclient/HDFS/hadoop/include -Wall -o hdfs_test.o hdfs_test.c  
cc -o hdfs_test hdfs_test.o -lhdfs
```

6. Run the following command to create, write, read, append, and delete files:

```
make run
```

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make run  
.hdfs_test  
hdfsConnect- SUCCESS!  
hdfsCreateDirectory- SUCCESS! : /tmp/nativeTest  
hdfsOpenFile- SUCCESS! for write : /tmp/nativeTest/testfile.txt  
hdfsWrite- SUCCESS! : /tmp/nativeTest/testfile.txt  
Flushing file data ....  
hdfsCloseFile- SUCCESS! : /tmp/nativeTest/testfile.txt
```

```
hdfsOpenFile- SUCCESS! for read : /tmp/nativeTest/testfile.txt
hdfsRead- SUCCESS!, Byte read : 31, File content : Hadoop HDFS Native file write!
hdfsSeek- SUCCESS!, Byte read : 21, File seek content : S Native file write!
hdfsPread- SUCCESS!, Byte read : 10, File read content : S Native f
hdfsCopy- SUCCESS!, File copied, src : /tmp/nativeTest/testfile.txt, des : /tmp/nativeTest/testfile1.txt
hdfsMove- SUCCESS!, File moved, src : /tmp/nativeTest/testfile1.txt, des : /tmp/nativeTest/testfile2.txt
hdfsRename- SUCCESS!, File renamed, Old name : /tmp/nativeTest/testfile2.txt, New name : /tmp/
nativeTest/testfile3.txt
hdfsDelete- SUCCESS!, File deleted : /tmp/nativeTest/testfile3.txt
hdfsSetReplication- SUCCESS!, Set replication 10 for /tmp/nativeTest/testfile.txt
hdfsChown- SUCCESS!, Chown success for /tmp/nativeTest/testfile.txt
hdfsChmod- SUCCESS!, Chmod success for /tmp/nativeTest/testfile.txt
hdfsUtime- SUCCESS!, Set time success for /tmp/nativeTest/testfile.txt

Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsGetPathInfo - SUCCESS!

Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsListDirectory- SUCCESS!, /tmp/nativeTest
hdfsTruncateFile- SUCCESS!, /tmp/nativeTest/testfile.txt
Block Size : 134217728
hdfsGetDefaultBlockSize- SUCCESS!
Block Size : 134217728 for file /tmp/nativeTest/testfile.txt
hdfsGetDefaultBlockSizeAtPath- SUCCESS!
HDFS Capacity : 102726873909
hdfsGetCapacity- SUCCESS!
HDFS Used : 4767076324
hdfsGetCapacity- SUCCESS!
hdfsExists- SUCCESS! /tmp/nativeTest/testfile.txt
hdfsConfGetStr- SUCCESS : hdfs://hacluster
hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : 31, File content : Hadoop HDFS
Native file write!
hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : 31, totalLocalBytesRead : 0,
totalShortCircuitBytesRead : 0, totalZeroCopyBytesRead : 0
```

## 7. Enter the debug mode (optional)

**make gdb**

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make gdb
gdb hdfs_test
GNU gdb (GDB) SUSE (7.5.1-0.7.29)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-suse-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /opt/hadoopclient/HDFS/hadoop/hdfs-c-example/hdfs_test...done.
(gdb)
```

### 1.11.5.1.3 HTTP REST API

#### Function Description

Users can use the application programming interface (API) of Representational State Transfer (REST) to create, read and write, append, and delete files. For details of the REST API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html>.

## Preparing Running Environment

**Step 1** Install the client. Install the client on the node. For example, install the client in the `/opt/hadoopclient` directory.

1. Run the following command to for the user authentication. In the command, the hdfs is taken as an example and can be defined by users.

**kinit hdfs**



### NOTE

The validity duration of kinit authentication is 24 hours. After 24 hours, you need to re-authenticate the sample with the kinit to restart the sample.

2. Prepare files **testFile** and **testFileAppend** and write content 'Hello, webhdfs user!' and 'Welcome back to webhdfs!'. Run the following command to prepare **testFile** and **testFileAppend** files:

**touch testFile**

**vi testFile**

```
Hello, webhdfs user!
```

**touch testFileAppend**

**vi testFileAppend**

```
Welcome back to webhdfs!
```

**Step 2** The MRS cluster supports only the HTTPS access by default. If access by using the HTTPS service, perform **Step 3**. If access by using the HTTP service, perform **Step 4**.

**Step 3** HTTPS-based access is different from HTTP-based access. When you access HDFS using HTTPS, you must ensure that the SSL protocol supported by the **curl** command is supported by the cluster because SSL security encryption is used. If the cluster does not support the SSL protocol, change the SSL protocol in the cluster. For example, if the **Curl** command only supports the TLSv1 protocol, modify the protocol configuration performing following measures:

Log in to FusionInsight Manager and choose **Cluster > Name of the desired cluster > Services > HDFS > Configurations > All Configurations**. Type **hadoop.ssl.enabled.protocols** in the research box, check whether the parameter value contains **TLSv1**. If the parameter value does not contain **TLSv1**, add **TLSv1** in the **hadoop.ssl.enabled.protocols** configuration item, and clear the value of **ssl.server.exclude.cipher.list**. Otherwise, HDFS cannot be accessed by using HTTPS. Then click **Save** and click **More > Restart Service** to restart the HDFS service.



TLSv1 has security vulnerabilities. Exercise caution when using it.

**Step 4** [Accessing FusionInsight Manager of an MRS Cluster](#), choose **Cluster > Name of the desired cluster > Services > HDFS > Configurations > All Configurations**. Type **dfs.http.policy** in the research box, select **HTTP\_AND\_HTTPS**, click **Save**, and select **More >Restart Service** to restart the HDFS service.

----End

## Procedure

**Step 1** Accessing FusionInsight Manager of an MRS Cluster, click **Cluster > Name of the desired cluster > Services**, and then select **HDFS**. The HDFS page is displayed.



Because webhdfs is accessed through HTTP/HTTPS, you need to obtain the IP address of the active NameNode and the HTTP/HTTPS port.

1. Click **Instance**, and in HDFS Instances page, view the host name and IP address of the active NameNode.
2. Click **Configurations**, and in HDFS Service Configuration page, find **namenode.https.port** (25002) and **namenode.https.port** (25003).

**Step 2** Create a directory by referring to the following link:

[http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make\\_a\\_Directory](http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make_a_Directory).

Click the link, **Figure 1-155** is displayed:

**Figure 1-155** Example code of creating a directory

The client receives a response with a boolean JSON object:

```
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
[{"boolean": true}]
```

Go to the **/opt/hadoopclient** directory, the installation directory of the client, and create the **huawei** directory.

1. Run the following command to check whether the **huawei** directory exists in the current path.

**hdfs dfs -ls /**

The running results are as follows:

```
linux1:/opt/hadoopclient # hdfs dfs -ls /
16/04/22 16:10:02 INFO hdfs.PeerCache: SocketCache disabled.
Found 7 items
-rw-r--r-- 3 hdfs supergroup      0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS
drwxr-x---  - flume hadoop       0 2016-04-20 18:02 /flume
drwx-----  - hbase hadoop      0 2016-04-22 15:19 /hbase
drwxrwxrwx   - mapred hadoop    0 2016-04-20 18:02 /mr-history
drwxrwxrwx   - spark supergroup 0 2016-04-22 15:19 /sparkJobHistory
drwxrwxrwx   - hdfs hadoop     0 2016-04-22 14:51 /tmp
drwxrwxrwx   - hdfs hadoop     0 2016-04-22 14:50 /user
```

The **huawei** directory does not exist in the current path.

2. Run the command in **Figure 1-155** that is named with **huawei**. Replace the <HOST> and <PORT> in the command with the host name or IP address and port number that are obtained in **Step 1**. Type the **huawei** as the directory in the <PATH>.

 NOTE

The <HOST> can be replaced by the host name or IP address. It is noted that the port of HTTP is different from the port of HTTPS.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X PUT --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei?op=MKDIRS"
```

In the command, the <HOST> is replaced by linux 1 and the <PORT> is replaced by 25002.

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly
Content-Length: 0
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 05 May 2016 03:10:09 GMT
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 03:10:09 GMT
Date: Thu, 05 May 2016 03:10:09 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAgIAb1swWaADAgEFoQMCQ
+iTTBLoAMCARKiRARCARhuv39Ttp6lhBLG3B0JAmFjv9weLp+SGFI+t2HSEHN6p4UVWKKy/
kd9dKEgNMlyDu/o7yts0cqMxNsI69WbN5H
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462453809395&s=wiRF4rdTWpm3tDST+a/Sy0lwgA4="; Path=/
Expires=Thu, 05-May-2016 13:10:09 GMT; HttpOnly
Transfer-Encoding: chunked
{"boolean":true}linux1:/opt/hadoopclient #
```

If {"boolean":true} returns, the **huawei** directory is successfully created.

- Run the following command to access HTTPS:

```
linux1:/opt/hadoopclient # curl -i -k -X PUT --negotiate -u: "https://10.120.172.109:25003/webhdfs/v1/huawei?op=MKDIRS"
```

In the command, the <HOST> is replaced by IP address (10.120.172.109) and the <PORT> is replaced by 25003.

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly
Content-Length: 0
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Fri, 22 Apr 2016 08:13:37 GMT
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
Expires: Fri, 22 Apr 2016 08:13:37 GMT
Date: Fri, 22 Apr 2016 08:13:37 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
```

```
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCugB+yT3Y+z8YCRMJHXF84o1cyCfjq157+NZN1gu7D7yhMULnjr
+7BuUdEcZKewFR7uD+DRIMY3akg3OgU45xQ9R
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1461348817963&s=sh57G7iVccX/Aknoz410yJPTLHg="; Path=/;
Expires=Fri, 22-Apr-2016 18:13:37 GMT; Secure; HttpOnly
Transfer-Encoding: chunked

>{"boolean":true}linux1:/opt/hadoopclient #
```

If {"boolean":true} returns, the **huawei** directory is successfully created.

3. Run the following command to check the **huawei** directory in the path.

```
linux1:/opt/hadoopclient # hdfs dfs -ls /
16/04/22 16:14:25 INFO hdfs.PeerCache: SocketCache disabled.
Found 8 items
-rw-r--r-- 3 hdfs supergroup      0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS
drwxr-x--- - flume hadoop        0 2016-04-20 18:02 /flume
drwx----- - hbase hadoop        0 2016-04-22 15:19 /hbase
drwxr-xr-x - hdfs supergroup    0 2016-04-22 16:13 /huawei
drwxrwxrwx - mapred hadoop      0 2016-04-20 18:02 /mr-history
drwxrwxrwx - spark supergroup   0 2016-04-22 16:12 /sparkJobHistory
drwxrwxrwx - hdfs hadoop        0 2016-04-22 14:51 /tmp
drwxrwxrwx - hdfs hadoop        0 2016-04-22 16:10 /user
```

**Step 3** Create a command of the upload request to obtain the information about Location where the DataNode IP address is written in.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X PUT --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/
testHdfs?op=CREATE"
```

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 06:09:48 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 06:09:48 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly
Content-Length: 0

HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 05 May 2016 06:09:48 GMT
Date: Thu, 05 May 2016 06:09:48 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 06:09:48 GMT
Date: Thu, 05 May 2016 06:09:48 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARKiRARCzQ6w
+9pNzWCTJEdoU3z9xEyg1JQNka0nYaB9TndvrL5S0neAoK2usnictTFnqlincAjwB6SnTht8Q16WDlHJX/
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462464588403&s=qry87vAyYzSn9Vs6Rm6vKLhKeU="; Path=/; Expires=Thu,
05-May-2016 16:09:48 GMT; HttpOnly
Location: http://linux1:25010/webhdfs/v1/huawei/testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUf4lZdloBVKOV3XQOCBSyXvFAp92alcRs4j-
KNulnN6wUoBJXRUIREZTIGRlbGVnYXRpb24UMTAuMTlwLjE3Mi4xMDk6MjUwMDA&namenoderpcadd
ress=hacluster&overwrite=false
Content-Length: 0
```

- Run the following command to access HTTPS:

```
linux1:/opt/hadoopclient # curl -i -k -X PUT --negotiate -u: "https://linux1:25003/webhdfs/v1/huawei/
testHdfs?op=CREATE"
```

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly
Content-Length: 0

HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 05 May 2016 03:46:18 GMT
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 03:46:18 GMT
Date: Thu, 05 May 2016 03:46:18 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAgIAb1swWaADAgEFoQMCAQ
+iTTBLoAMCARCKiRARCZMYR8GGUkn7pPZaoOYZD5HxzLTRZ71angUHKubW2wC/18m9/
OOZstGQ6M1wH2pGriipuCNsKIfwP93eO2Co0FQF3
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462455978166&s=F4rXUwEevHZze3PR8TxkzcV7RQQ="; Path=/; Expires=Thu,
05-May-2016 13:46:18 GMT; Secure; HttpOnly
Location: https://linux1:25011/webhdfs/v1/huawei/testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFTi9j7X64QwnSz59T
GFPKff7GhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzluMTA5Ojl1MDAw&namenoderpcaddr
ess=hacluster&overwrite=false
Content-Length: 0
```

**Step 4** According to the Location information, create the **testHdfs** file in the **/huawei/testHdfs** file on the HDFS and upload the content in the local **testFile** file into the **testHdfs** file.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X PUT -T testFile --negotiate -u: "http://linux1:250109864/
webhdfs/v1/huawei/testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFTi9j7X64QwnSz59T
KNuLnN6wUoBJXRUIREZTIGRlbGVnYXRpb24UMTAuMTlwLjE3Mi4xMDk6MjUwMDA&namenoderpcadd
ress=hacluster&overwrite=false"
```

- The running result is displayed as follows:

```
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Location: hdfs://hacluster/huawei/testHdfs
Content-Length: 0
Connection: close
```

- Run the following command to access HTTPS:

```
linux1:/opt/hadoopclient # curl -i -k -X PUT -T testFile --negotiate -u: "https://linux1:25011/
webhdfs/v1/huawei/testHdfs?
op=CREATE&delegation=HgAFYWRtaW4FYWRtaW4AigFUfwX3t4oBVKMSe7cCCBSFTi9j7X64QwnSz59T
GFPKff7GhNTV0VCSERGUyBkZWxlZ2F0aW9uFDEwLjEyMC4xNzluMTA5Ojl1MDAw&namenoderpcaddr
ess=hacluster&overwrite=false"
```

- The running result is displayed as follows:

```
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Location: hdfs://hacluster/huawei/testHdfs
Content-Length: 0
Connection: close
```

**Step 5** Go to the **/huawei/testHdfs** directory and read the content of **testHdfs** file.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -L --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?
op=OPEN"
```

- The running result is displayed as follows:  
Hello, webhdfs user!
- Run the following command to access HTTPS:  
linux1:/opt/hadoopclient # curl -k -L --negotiate -u: "https://linux1:25003/webhdfs/v1/huawei/testHdfs?op=OPEN"
- The running result is displayed as follows:  
Hello, webhdfs user!

**Step 6** Create a command of the upload request to obtain the information about Location where the DataNode IP address of **testHdfs** file is written in.

- Run the following command to access HTTP:  
linux1:/opt/hadoopclient # curl -i -X POST --negotiate -u: "http://linux1:25002//webhdfs/v1/huawei/testHdfs?op=APPEND"

- The running result is displayed as follows:  
HTTP/1.1 401 Authentication required  
Cache-Control: must-revalidate,no-cache,no-store  
Date: Thu, 05 May 2016 05:35:02 GMT  
Pragma: no-cache  
Date: Thu, 05 May 2016 05:35:02 GMT  
Pragma: no-cache  
Content-Type: text/html; charset=iso-8859-1  
X-Frame-Options: SAMEORIGIN  
WWW-Authenticate: Negotiate  
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly  
Content-Length: 1349  
  
HTTP/1.1 307 TEMPORARY\_REDIRECT  
Cache-Control: no-cache  
Expires: Thu, 05 May 2016 05:35:02 GMT  
Date: Thu, 05 May 2016 05:35:02 GMT  
Pragma: no-cache  
Expires: Thu, 05 May 2016 05:35:02 GMT  
Date: Thu, 05 May 2016 05:35:02 GMT  
Pragma: no-cache  
Content-Type: application/octet-stream  
X-Frame-Options: SAMEORIGIN  
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAgIAb1swWaADAgEFoQMCAQ  
+iTBLLoAMCARKiRARCTYvNX/2JMXhzsVPTw3Sluox6s/gEroHH980xMBkkYLcnO3W+0fM32c4/  
F98U5bl5dzgoolQoBvqq/EYivvvR12WX  
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain  
name>&t=kerberos&e=1462462502626&s=et1okvI0d7DWJ/LdhzNeS2wQEEY="; Path=/; Expires=Thu,  
05-May-2016 15:35:02 GMT; HttpOnly  
Location: http://linux1:25010/webhdfs/v1/huawei/testHdfs?  
op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUF2mGHooBVKN2Ch4KCBRzjM3jwSMIAowXb  
4dhqfkB5rT-8hjXRUJIREZTIGrlbGVnYXRpb24UMTAuMTlwLjE3Mi4xMDk6MjUwMDA&namenoderpcadd  
ress=hacluster  
Content-Length: 0

- Run the following command to access HTTPS:  
linux1:/opt/hadoopclient # curl -i -k -X POST --negotiate -u: "https://linux1:25003/webhdfs/v1/huawei/testHdfs?op=APPEND"

- The running result is displayed as follows:  
HTTP/1.1 401 Authentication required  
Cache-Control: must-revalidate,no-cache,no-store  
Date: Thu, 05 May 2016 05:20:41 GMT  
Pragma: no-cache  
Date: Thu, 05 May 2016 05:20:41 GMT  
Pragma: no-cache  
Content-Type: text/html; charset=iso-8859-1  
X-Frame-Options: SAMEORIGIN  
WWW-Authenticate: Negotiate  
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly  
Content-Length: 1349  
  
HTTP/1.1 307 TEMPORARY\_REDIRECT  
Cache-Control: no-cache

```
Expires: Thu, 05 May 2016 05:20:41 GMT
Date: Thu, 05 May 2016 05:20:41 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 05:20:41 GMT
Date: Thu, 05 May 2016 05:20:41 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGS1b3EgECAgIAb1swWaADAgEFoQMCAQ
+iTBLoAMCARKiRARCXgdjZuoxLHGtM1oyrPcXk95/
Y869eMfxIQV5UdEwBZ0iQiYaOdf5+Vk7a7FezhmzCABOWYXPxEQPNugbZ/yD5VLT
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462461641713&s=tGwwOH9scmnNtxPjlnu28SFtex0="; Path=/; Expires=Thu,
05-May-2016 15:20:41 GMT; Secure; HttpOnly
Location: https://linux1:25011/webhdfs/v1/huawei/testHdfs?
op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUF1xi_4oBVKN0v8HCBS3Fg0f_EwtFKKLODK
QSM2t32CjhNTV0VCSERGUyBkZWxIz2F0aW9uFDEwLjEyMC4xNzluMTA5OjI1MDAw&namenoderpcadd
ress=hacluster
```

**Step 7** According to the Location information, add the content in the local **testFileAppend** file to the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X POST -T testFileAppend --negotiate -u: "http://linux1:250109864/
webhdfs/v1/huawei/testHdfs?
op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUF2mGHooBVKN2Ch4KCBRzM3jwSMIAowXb
4dhqfKB5rT-8hJXRUIREZTIGRlbGVnYXRpb24UMTAuMTIwLjE3Mi4xMDk6MjUwMDAw&namenoderpcadd
ress=hacluster"
```

- The running result is displayed as follows:

```
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
Content-Length: 0
Connection: close
```

- Run the following command to access HTTPS:

```
linux1:/opt/hadoopclient # curl -i -k -X POST -T testFileAppend --negotiate -u: "https://linux1:25011/
webhdfs/v1/huawei/testHdfs?
op=APPEND&delegation=HgAFYWRtaW4FYWRtaW4AigFUF1xi_4oBVKN0v8HCBS3Fg0f_EwtFKKLODK
QSM2t32CjhNTV0VCSERGUyBkZWxIz2F0aW9uFDEwLjEyMC4xNzluMTA5OjI1MDAw&namenoderpcadd
ress=hacluster"
```

- The running result is displayed as follows:

```
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
Content-Length: 0
Connection: close
```

**Step 8** Go to the **/huawei/testHdfs** directory and read all content in the **testHdfs** file.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -L --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?
op=OPEN"
```

- The running result is displayed as follows:

```
Hello, webhdfs user!
Welcome back to webhdfs!
```

- Run the following command to access HTTPS:

```
linux1:/opt/hadoopclient # curl -k -L --negotiate -u: "https://linux1:25003/webhdfs/v1/huawei/
testHdfs?op=OPEN"
```

- The running result is displayed as follows:

```
Hello, webhdfs user!
Welcome back to webhdfs!
```

**Step 9** List details of all directory and file information in the **huawei** directory of the HDFS.

**LISTSTATUS** will return all child files and folders information in a single request.

- Run the following command to access HTTP.

```
linux1:/opt/hadoopclient # curl --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?op=LISTSTATUS"
```

- The result is displayed as follows:

```
{"FileStatuses":[{"FileStatus": [{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr oup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}]
```

- Run the following command to access HTTPS.

```
linux1:/opt/hadoopclient # curl -k --negotiate -u: "https://linux1:25003/webhdfs/v1/huawei/testHdfs?op=LISTSTATUS"
```

- The result is displayed as follows:

```
{"FileStatuses":[{"FileStatus": [{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr oup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"","permission":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}]
```

**LISTSTATUS** along with **size** and **startafter** param will help in fetching the child files and folders information through multiple requests, thus avoiding the user interface from becoming slow when there is a large amount of child information to be fetched.

- Run the following command to access HTTP.

```
linux1:/opt/hadoopclient # curl --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/?op=LISTSTATUS&startafter=sparkJobHistory&size=1"
```

- The result is displayed as follows:

```
{"FileStatuses":[{"FileStatus": [{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr oup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permmissio n":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}]
```

- Run the following command to access HTTPS.

```
linux1:/opt/hadoopclient # curl -k --negotiate -u: "https://linux1:25003/webhdfs/v1/huawei/?op=LISTSTATUS&startafter=sparkJobHistory&size=1"
```

- The result is displayed as follows:

```
{"FileStatuses":[{"FileStatus": [{"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr oup","length":70,"modificationTime":1462426678379,"owner":"hdfs","pathSuffix":"testHdfs","permmissio n":"755","replication":3,"storagePolicy":0,"type":"FILE"}]}]
```

## Step 10 Delete the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X DELETE --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?op=DELETE"
```

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required  
Date: Thu, 05 May 2016 05:54:37 GMT  
Pragma: no-cache  
Date: Thu, 05 May 2016 05:54:37 GMT  
Pragma: no-cache  
X-Frame-Options: SAMEORIGIN  
WWW-Authenticate: Negotiate  
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; HttpOnly  
Content-Length: 0  
HTTP/1.1 200 OK  
Cache-Control: no-cache  
Expires: Thu, 05 May 2016 05:54:37 GMT  
Date: Thu, 05 May 2016 05:54:37 GMT  
Pragma: no-cache  
Expires: Thu, 05 May 2016 05:54:37 GMT
```

```
Date: Thu, 05 May 2016 05:54:37 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGSib3EgECAgIAb1swWaADAgEFoQMCAQ
+iTBLLoAMCARKiRARC9k0/v6Ed8VlUBy3kuT0b4RkqkNMCrDevsLGQOUQRORKzWI3Wu
+XLJUMKlmZaWpP+bPzpx8O2Od81mLBgdi8sOkLw
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462463677153&s=Pwxe5UlqaULjFb9R6ZwlSX85Goi="; Path=/; Expires=Thu,
05-May-2016 15:54:37 GMT; HttpOnly
Transfer-Encoding: chunked
{"boolean":true}linux1:/opt/hadoopclient #
```

- Run the following command to access HTTPS:

```
linux1:/opt/hadoopclient # curl -i -k -X DELETE --negotiate -u: "https://linux1:25003/webhdfs/v1/
huawei/testHdfs?op=DELETE"
```

- The running result is displayed as follows:

```
HTTP/1.1 401 Authentication required
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate
Set-Cookie: hadoop.auth=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT; Secure; HttpOnly
Content-Length: 0
HTTP/1.1 200 OK
Cache-Control: no-cache
Expires: Thu, 05 May 2016 06:20:10 GMT
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
Expires: Thu, 05 May 2016 06:20:10 GMT
Date: Thu, 05 May 2016 06:20:10 GMT
Pragma: no-cache
Content-Type: application/json
X-Frame-Options: SAMEORIGIN
WWW-Authenticate: Negotiate YGoGCSqGSib3EgECAgIAb1swWaADAgEFoQMCAQ
+iTBLLoAMCARKiRARCLY5vrVmgsiH2VWRypc30iZGffRUF4nXNaHCWni3TIDUOTl+S+hfjatSbo/+uayQI/
6k9jAfajrvFlfxqppFtofpp
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs@<system domain
name>&t=kerberos&e=1462465210180&s=KGd2SbH/EUSaaeVKCb5zPzGBRKo="; Path=/; Expires=Thu,
05-May-2016 16:20:10 GMT; Secure; HttpOnly
Transfer-Encoding: chunked
{"boolean":true}linux1:/opt/hadoopclient #
```

----End

The Key Management Server (KMS) uses the HTTP REST API to provide key management services for external systems. For details about the API, see:

<https://hadoop.apache.org/docs/r3.3.1/hadoop-kms/index.html>.



As REST API reference has done security hardening to prevent script injection attack.  
Through REST API reference, it cannot create directory and file name which contain those key words "<script ", "<iframe", "<frame", "javascript:".

## 1.11.5.2 Shell Command Introduce

### HDFS Shell

You can use the Hadoop Distributed File System (HDFS) Shell command to perform operations on the HDFS, such as reading and writing files.

To run the HDFS Shell:

Go to the directory of HDFS client and enter the command. An example is shown as follows:

```
cd /opt/hadoopclient/HDFS/hadoop/bin
```

```
hdfs dfs -mkdir /tmp/input
```

You can run the following command to seek help about HDFS commands:

```
hdfs --help
```

For details about the shell, see

[http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-common/  
FileSystemShell.html](http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-common/FileSystemShell.html)

**Table 1-102** Transparent encryption-related commands

Scenario	Operation	Command	Description
hadoop shell command management key	Create keys.	<b><i>hadoop key create &lt;keyname&gt; [-cipher &lt;cipher&gt;] [-size &lt;size&gt;] [-description &lt;description&gt;] [-attr &lt;attribute=value&gt;] [-provider &lt;provider&gt;] [-help]</i></b>	The create subcommand creates a key for the name specified by the <keyname> argument within the provider specified by the -provider argument. You may specify a cipher with the -cipher argument.  The default keysize is 128. You may specify the requested key length using the -size argument. Arbitrary attribute=value style attributes may be specified using the -attr argument. The -attr may be specified for multiple times, once per attribute.
	Rollback	<b><i>hadoop key roll &lt;keyname&gt; [-provider &lt;provider&gt;] [-help]</i></b>	The roll subcommand creates a new version for the specified key within the provider indicated using the -provider argument.
	Delete keys	<b><i>hadoop key delete &lt;keyname&gt; [-provider &lt;provider&gt;] [-f] [-help]</i></b>	The delete subcommand deletes all versions of the key specified by the <keyname> argument within the provider specified by the -provider argument. The command asks for user confirmation unless -f is specified.
	View keys	<b><i>hadoop key list [-provider &lt;provider&gt;] [-metadata] [-help]</i></b>	The list subcommand displays the keynames contained in a particular provider as configured in core-site.xml or specified with the -provider argument. The -metadata argument displays the metadata.

**Table 1-103** Shell commands of Colocation client

Operation	Command	Description
Group creation	hdfs colocationadmin -createGroup -groupID <groupID> -locatorIDs <comma separated locatorIDs> or -file <path of the file contains all of locatorIDs>	Used to create a group. In the command, groupID is the group name and locatorID is the locator name. You can enter comma-separated locator IDs using command lines. You can also write locator IDs into a file so that the system can obtain the locator IDs by reading the file.
Group deletion	hdfs colocationadmin -deleteGroup <groupID>	Used to delete the specified group.
Group query	hdfs colocationadmin -queryGroup <groupID>	Used to query details about a specified group, including locators in the group and information about each locator and its corresponding DataNode.
Viewing all groups	hdfs colocationadmin -listGroups	Used to list all groups and their creation time.
Setting ACL permissions on Colocation directories	hdfs colocationadmin -setAcl	Used to set ACL permissions on Colocation directories in ZooKeeper. The default root directory of Colocation in ZooKeeper is <b>/hadoop/colocationDetails</b> .

## 1.12 HetuEngine Development Guide

### 1.12.1 Overview

### 1.12.1.1 Introduction to HetuEngine

#### Introduction to HetuEngine

HetuEngine is a high-performance interactive SQL analysis and data virtualization engine developed by Huawei. It seamlessly integrates with the big data ecosystem to implement interactive query of massive amounts of data within seconds, and supports cross-source and cross-domain unified data access to enable one-stop SQL convergence analysis in the data lake, between lakes, and between lakehouses.

### 1.12.1.2 Concepts

#### Basic Concepts

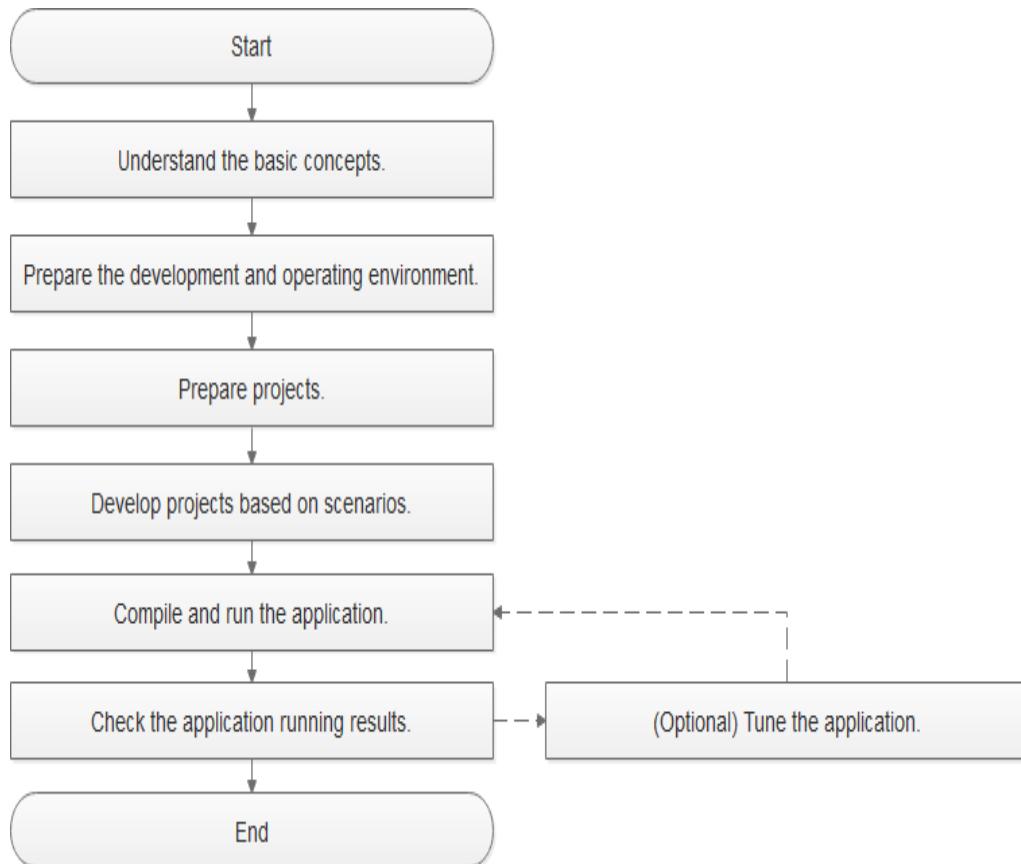
- HSFabric: unified entry for the HetuEngine services. It receives external requests and supports cross-network access to the HetuEngine services.
- HSBroker: service management of the HetuEngine services. It is used for resource management verification, health monitoring, and automatic maintenance of compute instances.
- Coordinator: coordinator of the HetuEngine services. It is responsible for SQL parsing and optimization.
- Worker: compute node of the HetuEngine compute instances. It executes tasks and processes data.
- Connector: an API for HetuEngine to access the database. Through the connector driver, HetuEngine connects to the data sources, reads data source metadata, and operates data (adding, deleting, modification, and query).
- Catalog: the catalog configuration file corresponds to a data source in HetuEngine. A data source can have multiple catalog configurations, which can be configured in the **properties** file of a data source.
- Schema: schema name of the database.
- Table: table name of the database.

### 1.12.1.3 Connection Modes

Connection Mode	Support for Username/Password Authentication	Support for Keytab Authentication	Support for Cross-Network-Segment Client Access	Prerequisite
HSFabric	Supported	Supported	Supported	<ul style="list-style-type: none"><li>The node running user services can communicate with the service nodes where HSFabric resides on the HetuEngine server side.</li><li>Dual-plane network scenarios are supported.</li><li>Only fixed IP addresses and ports need to be opened for HSFabric.</li><li>Supported version: MRS 3.1.3 or later.</li></ul>
HSBroker	Supported	Unsupported	Unsupported	<ul style="list-style-type: none"><li>The node running user services can communicate with the service nodes where HSBroker and Coordinator (randomly distributed in Yarn NodeManager) reside on the HetuEngine server side.</li><li>A large number of IP addresses and ports need to be opened for coordinators.</li><li>Supported version: MRS 3.1.0 or later</li></ul>

### 1.12.1.4 Development Process

This section describes the development process, as shown in [Figure 1-156](#).

**Figure 1-156** HetuEngine application development process**Table 1-104** Description of the HetuEngine application development process

Phase	Description	Reference Documents
Understand basic concepts.	Before application development, learn basic concepts of HetuEngine, and understand the scenario requirements.	<a href="#">Concepts</a>
Prepare the development and running environment.	The HetuEngine application can invoke the JDBC interface in any language for development. The current example uses the Java language. You are advised to use the IDEA tool to configure development environments in different languages according to the guide. The HetuEngine running environment is the client. Install and configure the client according to the guide.	<a href="#">Preparing the Development and Running Environment</a>

Phase	Description	Reference Documents
Prepare a project.	HetuEngine provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create a HetuEngine project according to the guide.	<a href="#">Configuring and Importing Sample Projects</a> <a href="#">Configuring and Importing Sample Projects</a>
Prepare for security authentication.	If a safe cluster is used, the safety certification must be performed.	<a href="#">Preparing for Security Authentication</a>
Develop a project based on the scenario.	A sample project in the Java language is provided, including an example project that connects the HetuEngine, SQL statement execution, result parsing, and disconnection.	<a href="#">Application Development</a>
Compile and run applications .	You can compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
Check the application running results.	The program running result is displayed in the expected display according to the implementation of the result parsing part.	<a href="#">Application Commissioning</a>

## 1.12.2 Environment Preparation

### 1.12.2.1 Preparing the Development and Running Environment

#### Preparing the Development Environment

This section describes the development and running environment to be prepared for application development, as listed in [Table 1-105](#).

**Table 1-105** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows 7 or later version is recommended.</li><li>• Operating environment: Windows or Linux OS If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane.</li></ul>
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows: The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstanceTLS-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstanceTLS-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>

Item	Description
Installation and configuration of IntelliJ IDEA	<p>Basic configuration of the development environment. The version must be 2019.1 or other compatible versions.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you are using an IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If you are using an Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If you are using an OpenJDK, ensure that the JDK configured in IntelliJ IDEA is the OpenJDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Maven installation	Basic configurations of the development environment. Maven is used for project management throughout the lifecycle of software development.
Development user	Cluster user for application development. Create the user and grant permissions to the user by referring to <a href="#">Preparing a Developer Account</a> .
7-zip	Tool used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

**Table 1-106** Python3 environment (required when the Python sample project is used)

Item	Description
Python3	Tool used to develop HetuEngine Python applications. The version must range from 3.6 to 3.9.
Setupools	Basic configuration of the Python3 development environment. Version: 47.3.1
jaydebeapi	Basic configuration of the Python3 development environment. You can use this module to connect to the database using Java JDBC.

## Preparing Operating Environment

During application development, prepare the environment for running and commissioning the program to verify that the application can run properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host;

obtain the cluster configuration file required for commissioning; configure the network connection, and commission the application in Windows.

- a. Log in to FusionInsight Manager. In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**, select the correct platform type (**x86\_64** for x86 and **aarch64** for ARM) based on the platform type of the node where the client is to be installed, and click **OK**. After the client file package is generated, download it to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar**. Then, decompress this file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HetuEngine\config** and place the configuration file to the **resources** directory of the sample project.

Save the **keytab** file obtained in [Preparing a Developer Account](#) to the **resources** directory.

Log in to the node where the HSBroker role is deployed as user **omm**, go to the  **\${BIGDATA\_HOME}/FusionInsight\_Hetu\_XXX/XXX\_HSBroker/etc/** directory, download the **hetuserver.jks** file, and save the file to the **resources** directory. [Table 1-107](#) describes the main configuration files.

**Table 1-107** Configuration files

File	Description
hdfs-site.xml	HDFS configuration parameters of the cluster
hetuserver-client.properties	HetuEngine client connection parameters
hetuserver-client-logging.properties	HetuEngine client log parameters
user.keytab	User information for Kerberos security authentication
krb5.conf	Kerberos Server configuration information
hetuserver.jks	Java TrustStore file
jaas-zk.conf	Jaas authentication file

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
  - The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
- a. Install the client on the node. For example, install the client in the **/opt/hadoopclient** directory.

The difference between the client time and the cluster time must be less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster.** Download the cluster client software package to the active management node and decompress it. Then log in to the active management node as user **root**. Go to the decompression directory of the cluster client, and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/HetuEngine/config** directory to the client node to the **conf** directory where the compiled JAR package is stored, for example, **/opt/hadoopclient/conf**, for subsequent commissioning.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active OMS node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp HetuEngine/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

Log in to the node where the HSBroker role is deployed as user **omm**, go to the  **\${BIGDATA\_HOME}/FusionInsight\_Hetu\_xxx/xxx\_HSBroker/etc/** directory, download the **hetuserver.jks** file, and place the file in the directory.

The keytab file obtained in [Preparing a Developer Account](#) is also stored in this directory. **Table 1-108** describes the main configuration files. (Obtain the required files you need.)

**Table 1-108** Configuration files

File	Description
hdfs-site.xml	HDFS parameters
hetuserver-client.properties	HetuEngine client connection parameters

File	Description
hetuserver-client-logging.properties	HetuEngine client log parameters
user.keytab	User information for Kerberos security authentication
krb5.conf	Kerberos Server configuration information
hetuserver.jks	Java TrustStore file
jaas-zk.conf	Jaas authentication file

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

### 1.12.2 Configuring and Importing Sample ProjectsConfiguring and Importing Sample Projects

#### Scenario

The client installation program directory contains a HetuEngine development sample project. You can start the sample project learning by importing the project. This document uses IntelliJ IDEA 2020.1.3 (Community Edition) as an example.

#### Prerequisites

Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the MRS cluster in the upper-right corner on the FusionInsight Manager page.

#### Procedure

- Step 1** Obtain the sample project folder **hetu-examples\hetu-examples-security** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
1. Open IntelliJ IDEA and choose **File > New > General > Project from Existing Sources > Select File or Directory to Import** to go to the **Browse Folder** dialog box is displayed.
  2. Select the sample project folder, choose **Import project from external model > Maven** during import, and click **Next** and then **Finish**.

 NOTE

The sample code is a Maven project. You can adjust the project configuration based on the site requirements.

----End

### 1.12.2.3 Setting Up a Python3 Project

#### Scenario

The following content describes the operations you need to do to run the python3 sample code of HetuEngine on FusionInsight MRS.

#### Procedure

**Step 1** Install Python 3.6 or a later version (before 3.9) on the client node.

The Python version can be viewed by running the **python3** command on the CLI of the client. The following information indicates that the Python version is 3.8.2.  
Python 3.8.2 (default, Jun 23 2020, 10:26:03)  
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux  
Type "help", "copyright", "credits" or "license" for more information.

**Step 2** Setuptools 47.3.1 must be installed on the client.

Download the software from the official website <https://pypi.org/project/setuptools/#files>.

Copy the downloaded setuptools package to the client, decompress the package, go to the setuptools project directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

Finished processing dependencies for setuptools==47.3.1

 NOTE

If the system displays a message indicating that setuptools 47.3.1 fails to be installed, check whether the environment is faulty or Python is faulty.

**Step 3** The JayDeBeApi module must be installed on the client. You can use the Java JDBC to connect to the database through this module.

You can install it either of the following ways:

- pip:  
Run the **pip install JayDeBeApi** command on the client node.
- Script:
  - a. Download the **JayDeBeApi** project file from the official website at <https://pypi.org/project/JayDeBeApi/>
  - b. Go to the **JayDeBeApi** project directory and run the **python3 setup.py install** command. During the installation, if the system displays a message indicating that the python3 module or package is missing, you need to add the module or package.

Take JayDeBeApi-1.2.3 as an example. If **Successfully installed JayDeBeApi-1.2.3** is displayed, the installation is successful.

**Step 4** Install Java on the client. For details about the supported Java versions, see "JDK Installation" in the [Table 1-105](#).

**Step 5** Obtain the Python3 sample code.

1. Obtain the sample project folder **python3-examples** in the **src\hetu-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
  - **normal** folder: Python3 sample code for interconnecting with HetuEngine in common mode
  - **security** folder: Python3 sample code for interconnecting with HetuEngine in security mode

**Step 6** Obtain the **hetu-jdbc** JAR package.

- Download the client file to the local PC through Manager.
  - a. Log in to FusionInsight Manager and choose **Cluster > Services > HetuEngine > More > Download Client**.
  - b. Select **Complete Client**, select the correct platform type (**x86\_64** for x86 and **aarch64** for ARM) for the node where the client is to be installed, deselect **Save to Path**, and click **OK**. Wait until the client file package is generated and download it.
  - c. Decompress the downloaded software package to obtain the **hetu-jdbc** package and decompress it to obtain the JDBC package.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress the package to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then decompress the package to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig** file. (The path cannot contain spaces.) Decompress **\FusionInsight\_Cluster\_1\_Services\_ClientConfig\HetuEngine\x86\hetu-jdbc.tar.gz** to obtain **hetu-jdbc-XXX.jar** and copy it to the user-defined path on the host where the sample code will run.

- Obtain the cluster client node.

Log in to the node where the HetuEngine client has been installed. For example, if the client installation path is **/opt/hadoopclient**, obtain **hetu-jdbc-XXX.jar** from **/opt/hadoopclient/HetuEngine/hetuserver/jars/**, and copy the file to the user-defined path on the node where the sample code will run.

----End

#### 1.12.2.4 Setting Up a Spring Boot Sample Project

##### Scenario

This topic describes how to run the Spring Boot sample code of the HetuEngine component in MRS.

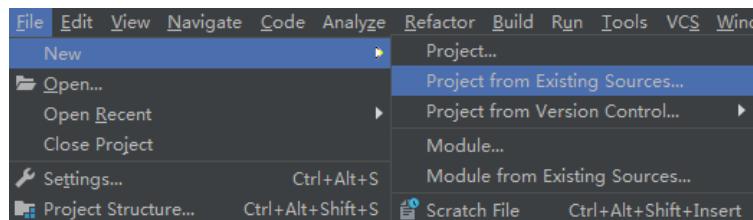
This example develops an application for connecting to the HetuEngine service using Spring Boot in a Windows environment.

## Procedure

**Step 1** Obtain the **src/springboot/hetu-examples** directory in the directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to IntelliJ IDEA.

1. Choose **File > New > Project from Existing Sources....**



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **hetu-examples** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.

Select the recommended JDK version and click **Finish**.

----End

### 1.12.2.5 Preparing for Security Authentication

#### 1.12.2.5.1 KeyTab File Authentication Using HSFabric

The KeyTab file authentication requires the **jaas-zk.conf**, **krb5.conf** and **user.keytab** files.

For details about how to obtain the **krb5.conf** and **user.keytab** files, see [Security Authentication](#).

The **jaas-zk.conf** file is defined as follows:

```
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    keyTab="/opt/hadoopclient/user.keytab"
    principal="hivetest@System domain name"
    useTicketCache=false
    storeKey=true
    debug=true;
};
```

 NOTE

- **keyTab** is the path of the **user.keytab** file. Change the value of **keyTab** in the **jaas-zk.conf** file based on the site requirements.  
For example:
  - Windows Path: "D:\\hetu-examples\\hetu-examples-security\\src\\main\\resources\\user.keytab".
  - Linux Path: "/opt/hadoopclient/user.keytab".
- **principal** is *the username added for authentication in section **Security Authentication**@domain name*. The domain name is the value of the **default\_realm** field in the **krb5.conf** file (For example, HADOOP.COM).

### 1.12.2.5.2 Username and Password Authentication Using HSFabric

Only the username and password are required to implement this authentication using HSFabric.

### 1.12.2.5.3 Username and Password Authentication Using HSBroker

Only the username and password are required to implement this authentication using HSBroker.

## 1.12.3 Application Development

### 1.12.3.1 Typical Application Scenario

This section describes the application development in a typical scenario, helping you quickly learn and master the HetuEngine development process and know key functions.

#### Scenario

Assume that a user develops an application and needs to perform the join operation on table A of the Hive data source and table B of the MPPDB data source. In this case, HetuEngine can be used to implement data query of the Hive data source, the process is as follows:

1. Connect to the HetuEngine JDBC server.
2. Assemble SQL statements.
3. Execute SQL statements.
4. Parse the returned result.
5. Close the HetuEngine JDBC Server connection.

### 1.12.3.2 Java Sample Code

### 1.12.3.2.1 KeyTab File Authentication Using HSFabric

#### Description

This section describes how to use the KeyTab file and HSFabric to connect to HetuEngine, assemble SQL statements, and send the SQL statements to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
public class JDBCExampleKeytabByFabric {  
    private static Properties properties = new Properties();  
    private final static String PATH_TO_JAAS_ZK_CONF = JDBCExample.class.getClassLoader()  
        .getResource("jaas-zk.conf")  
        .getPath();  
    private final static String PATH_TO_KRB5_CONF = JDBCExample.class.getClassLoader()  
        .getResource("krb5.conf")  
        .getPath();  
    private final static String PATH_TO_USER_KEYTAB = JDBCExample.class.getClassLoader()  
        .getResource("user.keytab")  
        .getPath();  
  
    private static void init() throws ClassNotFoundException {  
        System.setProperty("user.timezone", "UTC");  
        System.setProperty("java.security.auth.login.config", PATH_TO_JAAS_ZK_CONF);  
        System.setProperty("java.security.krb5.conf", PATH_TO_KRB5_CONF);  
        properties.setProperty("user", "hivetest");  
        properties.setProperty("SSL", "true");  
        properties.setProperty("KerberosConfigPath", PATH_TO_KRB5_CONF);  
        properties.setProperty("KerberosPrincipal", "hivetest");  
        properties.setProperty("KerberosKeytabPath", PATH_TO_USER_KEYTAB);  
        properties.setProperty("KerberosRemoteServiceName", "HTTP");  
        properties.setProperty("tenant", "default");  
        properties.setProperty("deploymentMode", "on_yarn");  
        properties.setProperty("ZooKeeperAuthType", "kerberos");  
        properties.setProperty("ZooKeeperSaslClientConfig", "Client");  
        Class.forName("io.trino.jdbc.TrinoDriver");  
    }  
    /**  
     * Program entry  
     *  
     * @param args no need program parameter  
     */  
    public static void main(String[] args) {  
        Connection connection = null;  
        ResultSet result = null;  
        PreparedStatement statement = null;  
        String url = "jdbc:trino://192.168.1.130:29902,192.168.1.131:29902,192.168.1.132:29902/hive/default?"  
            + "serviceDiscoveryMode=hsfabric";  
        try {  
            init();  
            String sql = "show tables";  
            connection = DriverManager.getConnection(url, properties);  
            statement = connection.prepareStatement(sql.trim());  
            result = statement.executeQuery();  
            ResultSetMetaData metaData = result.getMetaData();  
            Integer colNum = metaData.getColumnCount();  
            for (int j = 1; j <= colNum; j++) {  
                System.out.print(metaData.getColumnName(j) + "\t");  
            }  
            System.out.println();  
            while (result.next()) {  
                for (int j = 1; j <= colNum; j++) {  
                    System.out.print(result.getString(j) + "\t");  
                }  
                System.out.println();  
            }  
        } catch (SQLException | ClassNotFoundException e) {  
            e.printStackTrace();  
        } finally {  
    }
```

```
if (result != null) {
    try {
        result.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
if (statement != null) {
    try {
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
if (connection != null) {
    try {
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

The parameters in the preceding code are described in the following table.

**Table 1-109** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSFabric1_IP.HSFabric1_Port,HSFabric2_IP.HSFabric2_Port, HSFabric3_IP.HSFabric3_Port/catalog/schema?</i></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <b>catalog</b> and <b>schema</b> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <b>HSFabric_IP:HSFabric_Port</b> indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.1.130:29902,192.168.1.131:29902,192.168.1.132:29902. On FusionInsight Manager, choose <b>Cluster &gt; Services &gt; HetuEngine</b> and click <b>Instance</b> to obtain the service IP addresses of all HSFabric instances. On the <b>Configurations</b> page, search for <b>gateway.port</b> to obtain the port number of HSFabric.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
SSL	Whether to use the HTTPS connection. The default value is <b>false</b> .

Parameter	Description
SSLTrustStorePath	Java TrustStore file path
KerberosRemoteServiceName	Kerberos service name, which is fixed to <b>HTTP</b> .
KerberosPrincipal	Username corresponding to <b>keytab</b> specified by <b>KerberosKeytabPath</b>
KerberosConfigPath	The <b>krb5</b> configuration file to access the data source user. For details, see <a href="#">Preparing for Security Authentication</a> .
KerberosKeytabPath	The <b>keytab</b> configuration file of the data source user, which can be obtained by following the instructions in <a href="#">Preparing for Security Authentication</a> .
java.security.auth.login.config	Path of the <b>jaas-zk.conf</b> configuration file, which is used to access ZooKeeper in security mode.
java.security.krb5.conf	The <b>krb5</b> configuration file. For details, see <a href="#">Preparing for Security Authentication</a> .
ZooKeeperAuthType	ZooKeeper authentication mode. The value is <b>kerberos</b> in security mode.
ZooKeeperSaslClientConfig	Item name in the <b>jaas-zk.conf</b> configuration file
tenant	Tenant to which a user belongs
deploymentMode	Only <b>on_yarn</b> is supported.

### 1.12.3.2.2 Username and Password Authentication Using HSFabric

#### Description

An HSFabric connection is used. The username and password are used to connect to HetuEngine, and SQL statements are assembled and sent to HetuEngine for execution.

```
public class JDBCExampleFabric {  
    private static Properties properties = new Properties();  
    private static void init() throws ClassNotFoundException {  
        // Hard-coded password or plaintext password in code poses significant security risks. Encrypt and  
        // store them in configuration files or environment variables and decrypt them when needed.  
        // The password is stored in environment variables for identity authentication. Before running this  
        // example, set the environment variable HETUENGINE_PASSWORD.  
        properties.setProperty("user", "YourUserName");  
        String password = System.getenv("HETUENGINE_PASSWORD");  
        properties.setProperty("password", password);  
        Class.forName("io.trino.jdbc.TrinoDriver");  
    }  
    public static void main(String[] args){  
        Connection connection = null;  
        ResultSet resultSet = null;  
        PreparedStatement statement = null;  
        String url = "jdbc:trino://192.168.1.130:29902,192.168.1.131:29902/hive/default?  
        serviceDiscoveryMode=hsfabric";  
    }  
}
```

```
try {
    init();
    String sql = "show tables";
    connection = DriverManager.getConnection(url, properties);
    statement = connection.prepareStatement(sql.trim());
    resultSet = statement.executeQuery();
    Integer colNum = resultSet.getMetaData().getColumnCount();
    while (resultSet.next()) {
        for (int i = 1; i <= colNum; i++) {
            System.out.println(resultSet.getString(i) + "\t");
        }
    }
} catch (SQLException | ClassNotFoundException e) {
    e.printStackTrace();
} finally {
    if (resultSet != null) {
        try {
            resultSet.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (statement != null) {
        try {
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

**Table 1-110** describes the parameters in the preceding code.

**Table 1-110** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSFabric1_IP.HSFabric1_Port,HSFabric2_IP.HSFabric2_Port,HSFabric3_IP.HSFabric3_Port/catalog/schema?</i>serviceDiscoveryMode=hsfabric</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>catalog and schema indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>HSFabric_IP:HSFabric_Port indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29902,192.168.195.232:29902,192.168.169.84:29902. On FusionInsight Manager, choose Cluster &gt; Services &gt; HetuEngine and click Instance to obtain the service IP addresses of all HSFabric instances. On the Configurations page, search for gateway.port to obtain the port number of HSFabric.</li><li>(Optional) auditAddition: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>Name of the image. The name can contain 1 to 255 characters.</li><li>Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.

### 1.12.3.2.3 Querying SQL Tasks using JDBC

#### Description

A JDBC connection is used. You can use the username and password to connect to HetuEngine, assemble SQL statements, send them to HetuEngine for execution, and query the execution progress and status of the SQL statements.

```
import io.trino.jdbc.TrinoResultSet;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Properties;
import java.util.Timer;
import java.util.TimerTask;

public class JDBCExampleStatementProgressPercentage{

    private static Properties properties = new Properties();
    public static Connection connection = null;
    public static ResultSet result = null;
    public static PreparedStatement statement = null;
    private static void init() throws ClassNotFoundException {
        // Hard-coded password or plaintext password in code poses significant security risks. Encrypt and
        // store them in configuration files or environment variables and decrypt them when needed.
        // The password is stored in environment variables for identity authentication. Before running this
```

```
example, set the environment variable HETUENGINE_PASSWORD.
properties.setProperty("user", "YourUserName");
String password = System.getenv("HETUENGINE_PASSWORD");
properties.setProperty("password", password);
Class.forName("io.trino.jdbc.TrinoDriver");
}

/**
 * Program entry
 *
 * @param args no need program parameter
 */
public static void main(String[] args) {
    String url = "jdbc:trino://192.168.81.37:29860,192.168.195.232:29860,192.168.169.84:29860/hive/
default?serviceDiscoveryMode=hsbroker";
    try {
        init();
        String sql = "show tables";
        connection = DriverManager.getConnection(url, properties);
        statement = connection.prepareStatement(sql.trim());
        result = statement.executeQuery();

        TrinoResultSet rs = (TrinoResultSet) result;
        new Thread() {
            public void run() {
                Timer timer = new Timer();
                // Indicates that the task will be executed three seconds later and will be executed every two
seconds.
                timer.schedule(new TimerTask() {
                    @Override
                    public void run() {
                        double statementProgressPercentage = rs.getProgressPercentage().orElse(0.0);
                        System.out.println("The Current Query Progress Percentage is " +
statementProgressPercentage*100 + "%");
                        if("FINISHED".equals(rs.getStatementStatus().orElse(""))) {
                            System.out.println("The Current Query Progress Percentage is 100%");
                            timer.cancel();
                            Thread.currentThread().interrupt();
                        }
                    }
                }, 3000, 2000);
            }
        }.start();

        ResultSetMetaData resultMetaData = result.getMetaData();
        int colNum = resultMetaData.getColumnCount();
        for (int j = 1; j <= colNum; j++) {
            try {
                System.out.print(resultMetaData.getColumnLabel(j) + "\t");
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        }

        while (result.next()) {
            for (int j = 1; j <= colNum; j++) {
                System.out.print(result.getString(j) + "\t");
            }
            System.out.println();
        }
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        if (result != null) {
            try {
                result.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

**Table 1-111** describes the parameters in the preceding code.

**Table 1-111** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSBroker1_IP.HSBroker1_Port,HSBroker2_IP.HSBroker2_Port,HSBroker3_IP.HSBroker3_Port/catalog/schema?serviceDiscovery-Mode=hsbroker</i></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <b>catalog</b> and <b>schema</b> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <i>HSBroker_IP:HSBroker_Port</i> indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29860,192.168.195.232:29860,192.168.169.84:29860.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.
getStatementStatus()	Execution status of the SQL statement. There are 11 states: {'RUNNING', 'FAILED', 'FINISHED', 'QUEUED', 'WAITING_FOR_RESOURCES', 'DISPATCHING', 'PLANNING', 'STARTING', 'RESCHEDULING', 'RESUMING', 'FINISHING' }
getProgressPercentage()	Execution progress of the SQL statement. Value range: [0,1].

### 1.12.3.2.4 Username and Password Authentication Using HSBroker

#### Description

This section describes how to use HSBroker to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
public class JDBCExampleBroker {  
    private static Properties properties = new Properties();  
    private static void init() throws ClassNotFoundException {  
        // Hard-coded password or plaintext password in code poses significant security risks. Encrypt and  
        // store them in configuration files or environment variables and decrypt them when needed.  
        // The password is stored in environment variables for identity authentication. Before running this  
        // example, set the environment variable HETUENGINE_PASSWORD.  
        properties.setProperty("user", "YourUserName");  
        String password = System.getenv("HETUENGINE_PASSWORD");  
        properties.setProperty("password", password);  
        Class.forName("io.trino.jdbc.TrinoDriver");  
    }  
    public static void main(String[] args){  
        Connection connection = null;  
        ResultSet resultSet = null;  
        PreparedStatement statement = null;  
        String url = "jdbc:trino://192.168.1.130:29860,192.168.1.131:29860/hive/default?  
serviceDiscoveryMode=hsbroker";  
        try {  
            init();  
            String sql = "show tables";  
            connection = DriverManager.getConnection(url, properties);  
            statement = connection.prepareStatement(sql.trim());  
            resultSet = statement.executeQuery();  
            Integer colNum = resultSet.getMetaData().getColumnCount();  
            while (resultSet.next()) {  
                for (int i = 1; i <= colNum; i++) {  
                    System.out.println(resultSet.getString(i) + "\t");  
                }  
            }  
        } catch (SQLException | ClassNotFoundException e) {  
            e.printStackTrace();  
        } finally {  
            if (resultSet != null) {  
                try {  
                    resultSet.close();  
                } catch (SQLException e) {  
                    e.printStackTrace();  
                }  
            }  
            if (statement != null) {  
                try {  
                    statement.close();  
                } catch (SQLException e) {  
                    e.printStackTrace();  
                }  
            }  
            if (connection != null) {  
                try {  
                    connection.close();  
                } catch (SQLException e) {  
                    e.printStackTrace();  
                }  
            }  
        }  
    }  
}
```

**Table 1-112** describes the parameters in the preceding code.

**Table 1-112** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSBroker1_IP.HSBroker1_Port,HSBroker2_IP.HSBroker2_Port,HSBroker3_IP.HSBroker3_Port/catalog/schema?serviceDiscoveryMode=hsbroker</i></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <i>catalog</i> and <i>schema</i> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <i>HSBroker_IP:HSBroker_Port</i> indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29860,192.168.195.232:29860,192.168.169.84:29860.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.

### 1.12.3.3 Python3 Sample Code

#### 1.12.3.3.1 Username and Password Authentication Using HSBroker

This section describes how to use HSBroker to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
import jaydebeapi
import os

driver = "io.trino.jdbc.TrinoDriver"

# need to change the value based on the cluster information
url = "jdbc:trino://192.168.43.223:29860,192.168.43.244:29860/hive/default?serviceDiscoveryMode=hsbroker"
user = "YourUserName"
# Hard-coded or plaintext password is risky. For security, encrypt your passwords and store them in the configuration file or environment variables.
# The password is stored in environment variables for identity authentication. Before running this example, set the environment variable HETUENGINE_PASSWORD.
password = os.getenv('HETUENGINE_PASSWORD')
tenant = "YourTenant"
jdbc_location = "Your file path of the jdbc jar"

sql = "show tables"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {"user": user,
   "password": password,
   "tenant": tenant},
                             [jdbc_location])
    curs = conn.cursor()
```

```
curs.execute(sql)
result = curs.fetchall()
print(result)
curs.close()
conn.close()
```

The following table describes the parameters in the preceding code.

**Table 1-113** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSBroker1_IP.HSBroker1_Port,HSBroker2_IP.HSBroker2_Port,HSB roker3_IP.HSBroker3_Port/catalog/schema?serviceDiscovery- Mode=hsbroker</i></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <i>catalog</i> and <i>schema</i> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <i>HSBroker_IP:HSBroker_Port</i> indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29860,192.168.195.232:29860,192.168.169.84:29860.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances
jdbc_location	<p>Full path of the <b>hetu-jdbc-XXX.jar</b> package obtained in <a href="#">Setting Up a Python3 Project</a>.</p> <ul style="list-style-type: none"><li>• Example path in Windows: D:\\hetu-examples-python3\\ \\hetu-jdbc-XXX.jar</li><li>• Example path in Linux: /opt/hetu-examples-python3/hetu- jdbc-XXX.jar</li></ul>

### 1.12.3.3.2 Username and Password Authentication Using HSFabric

This section describes how to use HSFabric to connect to HetuEngine with the username and password, and assemble and send the SQL statements to HetuEngine for execution.

```
import jaydebeapi
driver = "io.trino.jdbc.TrinoDriver"
# need to change the value based on the cluster information
```

```
url = "jdbc:trino://192.168.43.244:29902/hive/default?serviceDiscoveryMode=hsfabric"
user = "YourUserName"
# Hard-coded or plaintext password is risky. For security, encrypt your passwords and store them in the
configuration file or environment variables.
# The password is stored in environment variables for identity authentication. Before running this example,
set the environment variable HETUENGINE_PASSWORD.
password = os.getenv('HETUENGINE_PASSWORD')
tenant = "YourTenant"
jdbc_location = "Your file path of the jdbc jar"

sql = "show tables"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {"user": user,
   "SSL": "true",
   "password": password,
   "tenant": tenant},
                               [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
    curs.close()
    conn.close()
```

The following table describes the parameters in the preceding code.

**Table 1-114** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSFabric1_IP:HSFabric1_Port,HSFabric2_IP:HSFabric2_Port,HSFabric3_IP:HSFabric3_Port/catalog/schema?serviceDiscoveryMode=hsfabric</i></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <b>catalog</b> and <b>schema</b> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <b>HSFabric_IP:HSFabric_Port</b> indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29902,192.168.195.232:29902,192.168.169.84:29902. On FusionInsight Manager, choose <b>Cluster &gt; Services &gt; HetuEngine</b> and click <b>Instance</b> to obtain the service IP addresses of all HSFabric instances. On the <b>Configurations</b> page, search for <b>gateway.port</b> to obtain the port number of HSFabric.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
password	Password of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances

Parameter	Description
jdbc_location	Full path of the <b>hetu-jdbc-XXX.jar</b> package obtained in <a href="#">Setting Up a Python3 Project</a> . <ul style="list-style-type: none"><li>• Example path in Windows: <b>D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar</b></li><li>• Example path in Linux: <b>/opt/hetu-examples-python3/hetu-jdbc-XXX.jar</b></li></ul>

### 1.12.3.3.3 KeyTab File Authentication Using HSFabric

This section describes how to use the KeyTab file and HSFabric to connect to HetuEngine, assemble SQL statements, and send the SQL statements to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
import jaydebeapi

driver = "io.trino.jdbc.TrinoDriver"

# need to change the value based on the cluster information
url = "jdbc:trino://192.168.43.244:29902/hive/default?serviceDiscoveryMode=hsfabric"
user = "YourUserName"
KerberosPrincipal = "YourUserName"
tenant = "YourTenant"
KerberosConfigPath = "Your file path of krb5.conf"
KerberosKeytabPath = "Your file path of user.keytab"
jdbc_location = "Your file path of the jdbc jar"

sql = "show tables"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {'user': user,
   'SSL': "true",
   'KerberosPrincipal': KerberosPrincipal,
   'KerberosConfigPath': KerberosConfigPath,
   'KerberosRemoteServiceName': "HTTP",
   'KerberosKeytabPath': KerberosKeytabPath,
   'tenant': tenant,
   'deploymentMode': "on_yarn",
   "ZooKeeperSaslClientConfig": "Client"},
                               [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
    curs.close()
    conn.close()
```

The following table describes the parameters in the preceding code.

**Table 1-115** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSFabric1_IP.HSFabric1_Port,HSFabric2_IP.HSFabric2_Port,HSFabric3_IP.HSFabric3_Port/catalog/schema?serviceDiscoveryMode=hsfabric</i></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>catalog and schema indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>HSFabric_IP:HSFabric_Port indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.1.130:29902,192.168.1.131:29902,192.168.1.132:29902. On FusionInsight Manager, choose <b>Cluster &gt; Services &gt; HetuEngine</b> and click <b>Instance</b> to obtain the service IP addresses of all HSFabric instances. On the <b>Configurations</b> page, search for <b>gateway.port</b> to obtain the port number of HSFabric.</li><li>(Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.</li><li>Name of the image. The name can contain 1 to 255 characters.</li><li>Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
KerberosPrincipal	Username corresponding to <b>keytab</b> specified by <b>KerberosKeytabPath</b>
tenant	Tenant resource queue for accessing HetuEngine compute instances
KerberosConfigPath	The <b>krb5</b> configuration file to access the data source user. For details, see <a href="#">Preparing for Security Authentication</a> .
KerberosKeytabPath	The <b>keytab</b> configuration file of the data source user, which can be obtained by following the instructions in <a href="#">Preparing for Security Authentication</a> .
jdbc_location	<p>Full path of the <b>hetu-jdbc-XXX.jar</b> package obtained in <a href="#">Setting Up a Python3 Project</a>.</p> <ul style="list-style-type: none"><li>Example path in Windows: <b>D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar</b></li><li>Example path in Linux: <b>/opt/hetu-examples-python3/hetu-jdbc-XXX.jar</b></li></ul>

## 1.12.4 Application Commissioning

### 1.12.4.1 Commissioning Applications on Windows

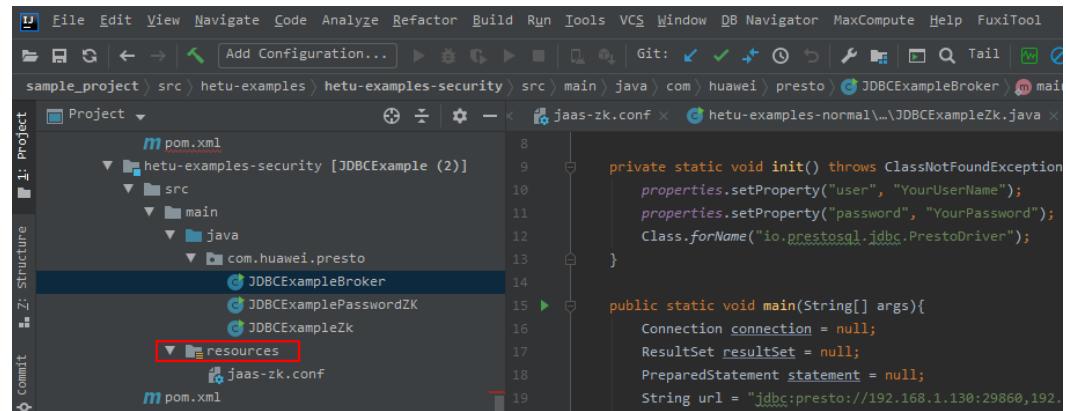
#### Scenario

After the program code is developed, you can compile the program in the Windows environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

#### Procedure

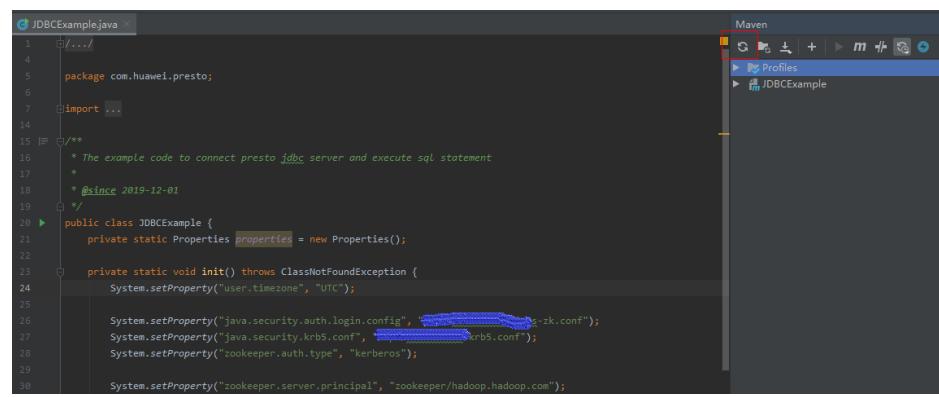
- Step 1** In the IntelliJ IDEA development environment on Windows, check that the **user.keytab** and **krb5.conf** files obtained in [Preparing for Security Authentication](#) are saved to the **resources** directory, and modify the parameters in the **jaas-zk.conf** file based on the actual path and username.

**Figure 1-157** Saving the authentication file to the resources directory



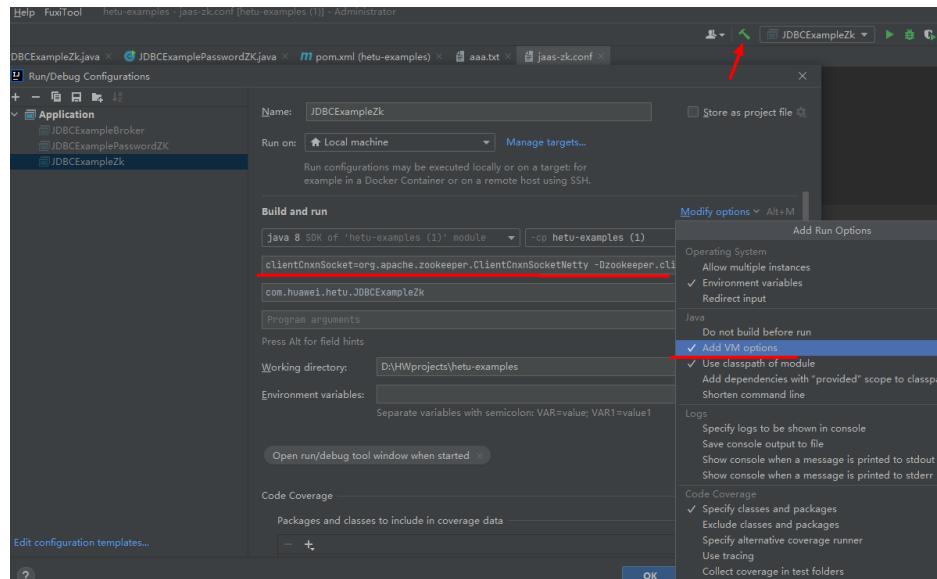
- Step 2** Click **Maven** on the right of IDEA to import the dependency.

**Figure 1-158** Importing the dependency



- Step 3** (Optional) If the SSL authentication communication function of ZooKeeper is enabled for the interconnected cluster, add the JVM configuration

**-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true** when you run JDBCExampleZk and JDBCExamplePasswordZK.



**Step 4** Right-click the **\*.java** file of the sample project and choose **Run'\*.main()'** from the shortcut menu. Wait until the execution is successful. (The default sample is to query a Hive table.)

- The running result of the JDBCExampleZk example application is shown as follows:

```
...
principal is hivetest@HADOOP.COM
Will use keytab
Commit Succeeded
...
The final connection url is: trino://192.168.1.189:29896/hive/default
Table
user_info
user_info2
```

- The running result of the JDBCExamplePasswordZK example application is as follows:

```
...
The final connection url is: trino://192.168.1.189:29896/hive/default
Table
user_info
user_info2
```

- The running result of the JDBCExampleBroker example application is as follows:

```
...
The final connection url is: trino://192.168.1.189:29896/hive/default
coordinator uri is trino://192.168.1.189:29896/hive/default
user_info
user_info2
```

----End

### 1.12.4.2 Commissioning Applications on Linux

#### Scenario

After the code is developed, you can compile the code into a JAR file and upload it to a Linux environment for application function commissioning.

 NOTE

Before commissioning applications on Linux, you need to pre-install a client on the Linux node.

## Procedure

- Step 1** Change the path for storing the KeyTab file in the **jaas-zk.conf** file in the Linux environment as required. for example, **/opt/hadoopclient/conf/user.keytab**.
- Step 2** Modify the configuration file path of the sample code, as shown in the following example:

```
private final static String PATH_TO_KRB5_CONF = "/opt/hadoopclient/krb5.conf"
```
- Step 3** In the Windows development environment IntelliJ IDEA, choose **Maven Projects > Example project name > Lifecycle** and perform the clean and package operations. After the compilation is complete, the **hetu-examples-XXX.jar** file is generated in the **target** directory.
- Step 4** Upload the **hetu-examples-XXX.jar** file to the **/opt/hadoopclient** directory in the Linux environment.
- Step 5** Download and decompress the client file **FusionInsight\_Cluster\_Cluster\_ID\_HetuEngine\_Client.tar** by referring to [Preparing Operating Environment](#), obtain the JDBC drive package, and upload it to the **/opt/hadoopclient** directory in the Linux environment.

 NOTE

The JDBC driver package **hetu-jdbc-\*jar** can be obtained from the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HetuEngine\XXX\** directory where the cluster client software package is decompressed. In the directory, **XXX** indicates ARM or x86.

- Step 6** Upload the **jaas-zk.conf**, **user.keytab**, and **krb5.conf** files obtained in [Preparing for Security Authentication](#) to the **/opt/hadoopclient** directory in the Linux environment.
- Step 7** Run the following command to go to the client installation directory:  
**cd /opt/hadoopclient**
- Step 8** Run the following command to configure environment variables:  
**source bigdata\_env**
- Step 9** Run the following command to debug the development program:  
**java -classpath hetu-examples-\*jar:hetu-jdbc-\*jar com.huawei.hetu.className**

 NOTE

- Replace the JDBC driver package name and class name with the actual ones, for example, `java -classpath hetu-examples-*.jar:hetu-jdbc-*.jar com.huawei.hetu.JDBCExampleBroker`.
- If SSL authentication of ZooKeeper is enabled for the interconnected cluster, you need to add the following JVM configuration

-  
`Dzookeeper.clientCnxnSocket=io.trino.jdbc.$internal.org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true.`

Using JDBCExampleZK as an example, the corresponding debugging command is as follows:

`java -cp hetu-examples-*.jar:hetu-jdbc-*.jar -Dzookeeper.clientCnxnSocket=io.trino.jdbc.$internal.org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true com.huawei.hetu.JDBCExampleZK.`

**Step 10** Check whether the command output is normal.

```
Jul 01, 2021 8:41:23 PM io.trino.jdbc.$internal.airlift.log.Logger info  
INFO: hsbroker finalUri is https://192.168.1.150:29860  
Jul 01, 2021 8:41:24 PM io.trino.jdbc.$internal.airlift.log.Logger info  
INFO: The final connection url is: trino://192.168.1.189:29896/hive/default  
Jul 01, 2021 8:41:24 PM io.trino.jdbc.$internal.airlift.log.Logger info  
INFO: coordinator uri is trino://192.168.1.189:29896/hive/default  
user_info  
user_info2
```

----End

### 1.12.4.3 Commissioning a Python3 Application

#### Scenario

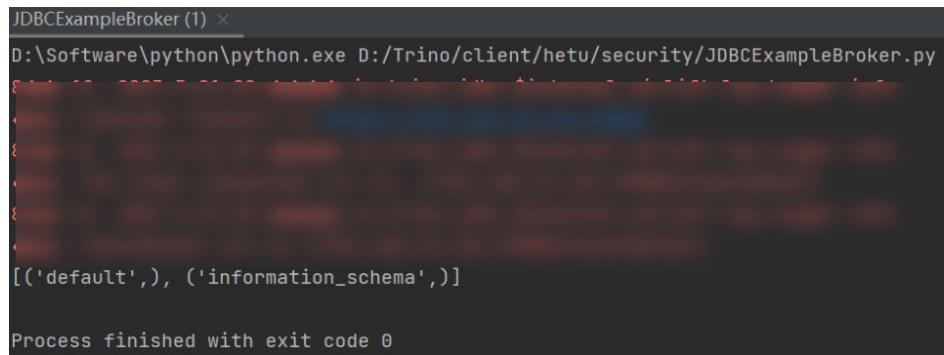
After the program code is written, you can commission the code in the Windows environment or upload the code to the Linux environment. If the local environment can communicate with the cluster service plane network, you can commission the program locally.

#### Procedure

1. Refer to [Setting Up a Python3 Project](#) to obtain the sample code, obtain the **hetu-jdbc-XXX.jar** file, and copy it to the user-defined directory.
2. Refer to [KeyTab File Authentication Using HSFabric](#) to obtain the **user.keytab** and **krb5.conf** files and save them to the customized directory.
3. Edit the sample code, modify URLs, user information, and passwords based on the cluster requirements, and modify **jdbc\_location** based on the actual path.
  - Example path in Windows: **D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar**
  - Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**
4. Run the python3 sample code.
  - In Windows, run the **py** script through pycharm or Python IDLE.
  - To run the sample code on Linux, install Java first.  
Go to the sample code path and run the **py** script. An example of the sample code path is **/opt/hetu-examples-python3**.  
**cd /opt/hetu-examples-python3**

**python3 JDBCExampleBroker.py**

5. View the command output.
  - In Windows, view the running result on the console.

**Figure 1-159** Running result

```
JDBCExampleBroker (1) >
D:\Software\python\python.exe D:/Trino/client/hetu/security/JDBCExampleBroker.py
[('default',), ('information_schema',)]
Process finished with exit code 0
```

- The following is an example of the running result in Linux:

```
Aug 12, 2023 5:19:53 PM io.trino.jdbc.$internal.airlift.log.Logger info
INFO: hsbroker finalUri is https://192.168.43.223:29860
Aug 12, 2023 5:19:53 PM io.trino.jdbc.$internal.airlift.log.Logger info
INFO: The final connection url is: //192.168.43.161:29888/hive/default
Aug 12, 2023 5:19:53 PM io.trino.jdbc.$internal.airlift.log.Logger info
INFO: coordinator uri is //192.168.43.161:29888/hive/default
[('default',), ('information_schema',)]
```

#### 1.12.4.4 Commissioning a Spring Boot Application

**Step 1** Configure the properties required for connecting the Spring Boot sample program to HetuEngine. Set the following parameters in the **springboot\hetu-examples\src\main\resources\application.yml** configuration file:

**Table 1-116** Configuring HetuEngine

Parameter	Mandatory	Description
host	Yes	HS Fabric address list of the HetuEngine service. The format is <i>HSFabric_IP1:HSFabric_Port,HSFabric_IP2:HSFabric_Port,HSFabric_IP3:HSFabric_Port</i> . Log in to FusionInsight Manager, choose <b>Cluster &gt; Services &gt; HetuEngine &gt; Instances</b> , and obtain the service IP addresses of all HSFabric instances. In the <b>Configurations</b> tab, search for <b>gateway.port</b> to obtain the HSFabric port.
catalog	Yes	Catalog name of the HetuEngine compute instance
schema	Yes	Schema name of the HetuEngine compute instance
user	Yes	Username for connecting to the HetuEngine compute instance

Parameter	Mandatory	Description
password	No	Password of the user for connecting to the HetuEngine compute instance in a cluster in security mode
ssl	Yes	Whether to use SSL to connect to the HetuEngine compute instance <ul style="list-style-type: none"> <li>For a cluster in security mode, set this parameter to <b>true</b>.</li> <li>For a cluster in normal mode, set this parameter to <b>false</b>.</li> </ul>

**Step 2** Click **Terminal** on the lower left part of the IDEA page to access the terminal. Run the following command to compile the file:

**mvn clean package**

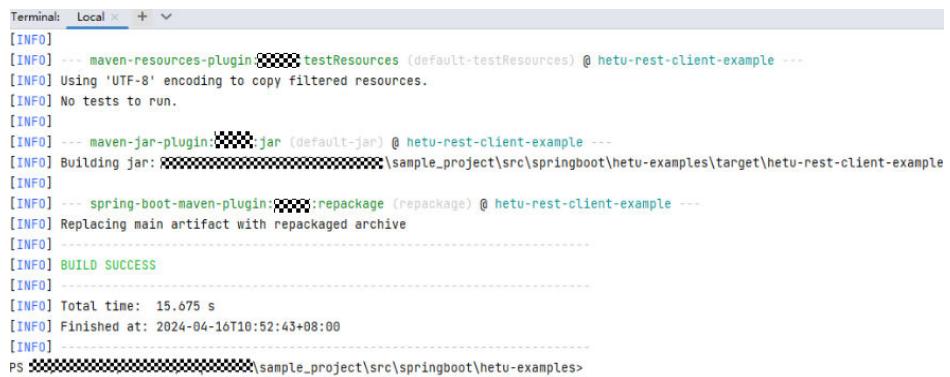
**Figure 1-160** Compile command



```
Terminal: Local + 
PowerShell
PS C:\sample_project\src\springboot\hetu-examples> mvn clean package
```

If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR package containing the **hetu-rest-client-example** field is generated in the target directory of the sample project.

**Figure 1-161** Successful compilation



```
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ hetu-rest-client-example ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:3.1.0:jar (default-jar) @ hetu-rest-client-example ---
[INFO] Building jar: C:\sample_project\src\springboot\hetu-examples\target\hetu-rest-client-example.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.1:repackage (repackage) @ hetu-rest-client-example ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 15.075 s
[INFO] Finished at: 2024-04-16T10:52:43+08:00
[INFO]
```

**Step 3** Create a directory in the system as the run directory to save the JAR package that contains the **hetu-rest-client-example** field generated in **Step 2**.

- Create a directory on Windows, for example, **D:\hetu-rest-client-example**.
- Create a directory on Linux, for example, **/opt/hetu-rest-client-example**.

**Step 4** Run the following command to start the SpringBoot service:

- Commands for Windows:

**cd /d D:\hetu-rest-client-example**

```
java -jar hetu-rest-client-example-*-SNAPSHOT.jar
```

- Commands for Linux:

```
chmod +x /opt/hetu-rest-client-example -R
```

```
cd /opt/hetu-rest-client-example
```

```
java -jar hetu-rest-client-example-*-SNAPSHOT.jar
```



The preceding JAR package names are for reference only. Use the actual names.

**Step 5** Call a Spring Boot sample API of HetuEngine to run the sample code.

- For the Windows environment:

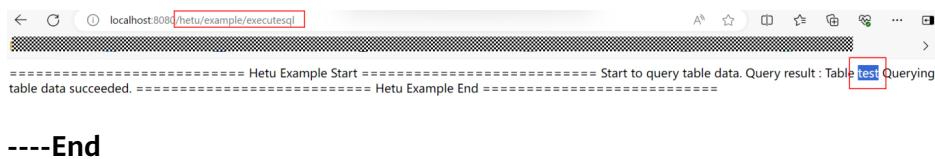
Open a browser and enter **http://localhost:8080/hetu/example/executesql** in the address box.

- For the Linux environment:

Run the **curl http://localhost:8080/hetu/example/executesql** command on the node where the JAR package is stored in **Step 3**.

**Step 6** View the query result of the SQL statement used in the sample code.

**Figure 1-162** Query results



## 1.13 Hive Development Guide

### 1.13.1 Overview

#### 1.13.1.1 Application Development Overview

##### Hive Introduction

Hive is an open-source data warehouse built on Hadoop. It stores structured data and provides basic data analysis services using the Hive query language (HQL), a language like the SQL. Hive converts HQL statements to Mapreduce or Spark jobs for querying and analyzing massive data stored in Hadoop clusters.

Hive provides the following features:

- Extracts, transforms, and loads (ETL) data using HQL.
- Analyzes massive structured data using HQL.
- Supports flexible data storage formats, including JavaScript object notation (JSON), comma separated values (CSV), TextFile, RCFile, ORCFILE, and SequenceFile, and supports custom extensions.
- Multiple client connection modes. Interfaces, such as JDBC and Thrift interfaces are supported.

Hive applies to offline massive data analysis (such as log and cluster status analysis), large-scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

To ensure Hive high availability (HA), user data security, and service access security, MRS incorporates the following features based on Hive 3.1.0:

- Kerberos security authentication
- Data file encryption
- Complete rights management

For Hive features in the Open Source Community, see <https://cwiki.apache.org/confluence/display/hive/designdocs>.

### 1.13.1.2 Common Concepts

- **keytab file**

The keytab file is a key file that stores user information. Applications use the key file for API authentication on MRS.

- **Client**

Users can access the server from the client through the Java API and Thrift API to perform Hive-related operations.

- **HQL**

Similar to SQL

- **HCatalog**

HCatalog is a table information management layer created based on Hive metadata and absorbs DDL commands of Hive. HCatalog provides read/write interfaces for Mapreduce and provides Hive command line interfaces (CLIs) for defining data and querying metadata. Hive and Mapreduce development personnel can share metadata based on the HCatalog component of MRS, preventing intermediate conversion and adjustment and improving the data processing efficiency.

- **WebHCat**

WebHCat running users use Rest APIs to perform operations, such as running Hive DDL commands, submitting Mapreduce tasks, and querying Mapreduce task execution results.

### 1.13.1.3 Required Permissions

Before developing an application based on this guide, ensure that the basic permissions of the users belong to a Hive group and obtain additional operation permissions from the system administrator. **Table 1-117** describes the permission required for each operation. To run example programs, you must have the create permission for the default database.

**Table 1-117** Required permissions

Operation Type/ Functional Object	Operation	Required Permission
DATABASE	CREATE DATABASE <i>dbname</i> [LOCATION "hdfs_path"]	If the HDFS path <b>hdfs_path</b> is specified, the ownership and RWX permission of <b>hdfs_path</b> are required.
	DROP DATABASE <i>dbname</i>	The database <b>dbname</b> ownership is required.
	ALTER DATABASE <i>dbname</i> SET OWNER <i>user_or_role</i>	The admin permission is required.
TABLE	CREATE TABLE <i>table_a</i>	The create permission for the database is required.
	CREATE TABLE <i>table_a</i> AS SELECT <i>table_b</i>	The create permission for the database and the select permission for <b>table_b</b> are required.
	CREATE TABLE <i>table_a</i> LIKE <i>table_b</i>	The create permission for the database is required.
	CREATE [EXTERNAL] TABLE <i>table_a</i> LOCATION "hdfs_path"	The create permission for the database, and the ownership and RWX permission of <b>hdfs_path</b> on HDFS are required.
	DROP TABLE <i>table_a</i>	The ownership of <b>table_a</b> is required.
	ALTER TABLE <i>table_a</i> SET LOCATION "hdfs_path"	The ownership of <b>table_a</b> , and the ownership and RWX permission of <b>hdfs_path</b> on HDFS are required.
	ALTER TABLE <i>table_a</i> SET FILEFORMAT	The ownership of <b>table_a</b> is required.
	TRUNCATE TABLE <i>table_a</i>	The ownership of <b>table_a</b> is required.
	ANALYZE TABLE <i>table_a</i> COMPUTE STATISTICS	The select and insert permission for <b>table_a</b> is required.
	SHOW TBLPROPERTIES <i>table_a</i>	The select permission for <b>table_a</b> is required.

Operation Type/ Functional Object	Operation	Required Permission
	SHOW CREATE TABLE <i>table_a</i>	The select permission with grant option for <b>table_a</b> is required.
Alter	ALTER TABLE <i>table_a</i> ADD COLUMN	The ownership of <b>table_a</b> is required.
	ALTER TABLE <i>table_a</i> REPLACE COLUMN	The ownership of <b>table_a</b> is required.
	ALTER TABLE <i>table_a</i> RENAME	The ownership of <b>table_a</b> is required.
	ALTER TABLE <i>table_a</i> SET SERDE	The ownership of <b>table_a</b> is required.
	ALTER TABLE <i>table_a</i> CLUSTER BY	The ownership of <b>table_a</b> is required.
PARTITION	ALTER TABLE <i>table_a</i> ADD PARTITION <i>partition_spec</i> LOCATION "hdfs_path"	The insert permission for <b>table_a</b> , and the ownership and RWX permission of <b>hdfs_path</b> on HDFS are required.
	ALTER TABLE <i>table_a</i> DROP PARTITION <i>partition_spec</i>	The delete permission for <b>table_a</b> is required.
	ALTER TABLE <i>table_a</i> PARTITION <i>partition_spec</i> SET LOCATION "hdfs_path"	The ownership of <b>table_a</b> , and the ownership and RWX permission of <b>hdfs_path</b> on HDFS are required.
	ALTER TABLE <i>table_a</i> PARTITION <i>partition_spec</i> SET FILEFORMAT	The ownership of <b>table_a</b> is required.
LOAD	LOAD INPATH 'hdfs_path' INTO TABLE <i>table_a</i>	The insert permission for <b>table_a</b> , and the ownership and RWX permission of <b>hdfs_path</b> on HDFS are required.
INSERT	INSERT TABLE <i>table_a</i> SELECT FROM <i>table_b</i>	The update permission for <b>table_a</b> and select permission for <b>table_b</b> are required.
SELECT	SELECT * FROM <i>table_a</i>	The select permission for <b>table_a</b> is required.

Operation Type/ Functional Object	Operation	Required Permission
	SELECT FROM <i>table_a</i> JOIN <i>table_b</i>	The select permission for <b>table_a</b> and <b>table_b</b> , the Submit permission of the default Yarn queue is required.
	SELECT FROM (SELECT FROM <i>table_a</i> UNION ALL SELECT FROM <i>table_b</i> )	The select permission for <b>table_a</b> and <b>table_b</b> , the Submit permission of the default Yarn queue is required.
EXPLAIN	EXPLAIN [EXTENDED] DEPENDENCY] <i>query</i>	The RX permissions for related table directory is required.
VIEW	CREATE VIEW <i>view_name</i> AS SELECT ...	The select permission with grant option for related tables is required.
	ALTER VIEW <i>view_name</i> RENAME TO <i>new_view_name</i>	The ownership of <b>view_name</b> is required.
	DROP VIEW <i>view_name</i>	The ownership of <b>view_name</b> is required.
INDEX	CREATE INDEX <i>index_name</i> ON TABLE <i>base_table_name</i> ( <i>col_name</i> , ...) AS <i>index_type</i>	The ownership of <b>table_a</b> is required.
	DROP INDEX <i>index_name</i> ON <i>table_name</i>	The ownership of <b>index_name</b> is required.
	ALTER INDEX <i>index_name</i> ON <i>table_name</i> REBUILD	The ownership of <b>index_name</b> is required.
FUNCTION	CREATE [TEMPORARY] FUNCTION <i>function_name</i> AS ' <i>class_name</i> '	The admin permission is required.
	DROP [TEMPORARY] <i>function_name</i>	The admin permission is required.
MACRO	CREATE TEMPORARY MACRO <i>macro_name</i> ...	The admin permission is required.

Operation Type/ Functional Object	Operation	Required Permission
	DROP TEMPORARY MACRO <i>macro_name</i>	The admin permission is required.

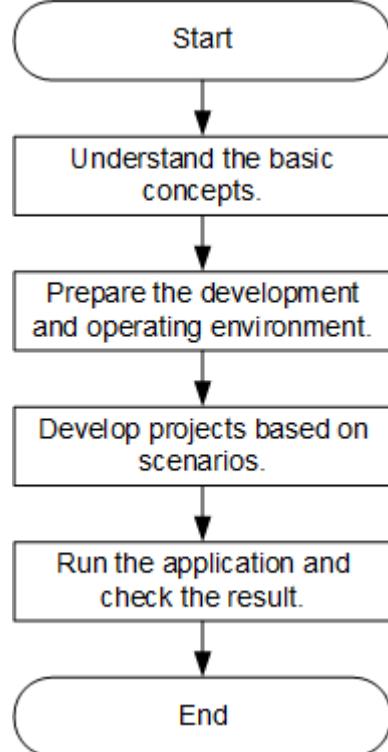
 NOTE

- You can perform all the previous operations when owning the **admin** permission of Hive and the corresponding directory permission of HDFS.
- If the current component uses Ranger for permission control, you need to configure permission management policies based on Ranger. For details, see "Component Operation Guide > Using Ranger > Adding a Ranger Access Permission Policy for Hive" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

#### 1.13.1.4 Development Process

[Figure 1-163](#) and [Table 1-118](#) illustrates each phase of the development process.

**Figure 1-163** Hive application development process



**Table 1-118** Description of the Hive application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, understand common concepts about Hive.	<a href="#">Common Concepts</a>
Prepare the development and operating environment.	Hive applications support Java and Python development languages. You are advised to use the IntelliJ IDEA tool to configure the development environment based on the language.	<a href="#">Preparing the Environment</a>
Develop projects based on scenarios.	Provides example projects of the Java and Python languages as well as example projects covering table creation, data load, and data query.	<a href="#">Developing an Application</a>
Run the application and view results.	Describes how to submit a developed application for compiling and view the result.	<a href="#">Commissioning Applications</a>

## 1.13.2 Preparing the Environment

### 1.13.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

Hive applications can be developed by using the JDBC/Metastore/Python/Python3 API. The following table describes the development and operating environment to be prepared.

**Table 1-119** JDBC/Metastore development environment

Item	Description
OS	<ul style="list-style-type: none"><li>Development environment: Windows OS. Windows 7 or later is supported.</li><li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Tool used for developing Hive applications. Requirements are as follows:</p> <p>JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li></ul>

Item	Description
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
Developer account preparation	See <a href="#">Preparing a Developer Account</a> for configuration.
7-zip	Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

**Table 1-120** Python development environment

Item	Description
OS	Development and operating environment: Linux OS
Python installation	Python is a tool used to develop Hive applications. Its version must be 2.6.6 or later but should not be later than 2.7.13.
setuptools installation	Basic configuration for the Python development environment. The version must be 5.0 or later.
Developer account preparation	See <a href="#">Preparing a Developer Account</a> for configuration.

 NOTE

For details about how to install and configure the Python development tool, see the [Configuring the Python Sample Project](#).

**Table 1-121** Python3 development environment

Item	Description
OS	Development and operating environment: Linux OS
Python3 installation	Python3 is a tool used to develop Hive applications. Its version must be 3.6 or later but should not be later than 3.9.
setuptools installation	Basic configuration for the Python3 development environment. The version must be 47.3.1.
Developer account preparation	See <a href="#">Preparing a Developer Account</a> for configuration.

 NOTE

For details about how to install and configure the Python3 development tool, see the [Configuring the Python3 Sample Project](#).

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. Log in to the FusionInsight Manager, click **Download Client** in the upper right corner of the Homepage, set **Select Client Type** to **Configuration Files Only** and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\Hive\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Hive sample project.

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. **Table 1-122** describes the main configuration files.

**Table 1-122** Configuration file

Document Name	Function
hive metastore-site.xml	Configures Hive parameters.
hiveclient.properties	Configures Hive parameters.
user.keytab	Provides Hive user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
core-site.xml	Configures Hive parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where

the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- When configuring IPv6 hosts on the local PC running Windows, add % and InterfaceIndex of the network interface card (NIC) to the end of the IPv6 address. The InterfaceIndex can be obtained by running the **ipconfig** command.

Example:

fec1:0:0:e505:8:99:5:1%10

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Hive/config** directory to the **src/main/resources** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/src/main/resources**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Hive/config/* root@IP address of the client node:/opt/  
hadoopclient/src/main/resources
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-123](#) describes the main configuration files.

**Table 1-123 Configuration files**

File	Function
hive metastore-site.xml	Configures Hive parameters.
hiveclient.properties	Configures Hive parameters.
user.keytab	Provides Hive user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
core-site.xml	Configures Hive parameters.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

### 1.13.2.2 Configuring the JDBC Sample Project

#### Scenario

To run the JDBC API example codes of the Hive component of MRS, perform the following operations.



#### NOTE

- The following uses the development of an application for connecting the Hive service in JDBC mode on Windows as an example.
- After importing the jdbc-example sample project, you need to change xxx of `USER_NAME = "xxx"` in the code to the development user created in [Preparing a Developer Account](#).

#### Procedure

**Step 1** Obtain the sample project folder **hive-jdbc-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** Copy the **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) to the **hive-jdbc-example\src\main\resources** directory of the sample project.

**Step 3** Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\Hive\config** and manually import the **core-site.xml** and **hiveclient.properties** files to the **hive-jdbc-example\src\main\resources** directory of the sample project.

### NOTE

Hive allows you to add extension identifiers to JDBC connection strings. These extension identifiers are printed in HiveServer audit logs to distinguish SQL sources. If you need to configure an identifier, add the following content to the **hive-jdbc-example\src\main\resources\hiveclient.properties** file:

```
auditAddition=xxx
```

*xxx* is the custom identifier. The identifier can contain a maximum of 256 bytes and only letters, digits, underscores (\_), commas (,), and colons (:) are allowed.

After the SQL statement is executed, the configured extension identifier is printed in the HiveServer audit log **/var/log/Bigdata/audit/hive/hiveserver/hive-audit.log**. The following is an example.

```
2024-02-27 11:55:56,501 | INFO | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserID=192.168.64.239 Time=2024/02/27 11:55:56 Operation=OpenSession Result= Data
1=Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:56,501 | INFO | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserID=192.168.64.239 Time=2024/02/27 11:55:56 Operation=OpenSession Result=SUCCE
2=Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:59,312 | INFO | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserID=192.168.64.239 Time=2024/02/27 11:55:59 Operation=ExecuteStatement Stmt
=(show tables) Result= Details=Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:59,316 | INFO | HiveServer2-Handler-Pool: Thread-402 | OperationId=718760c-bb8d-42f0-80d3-760baef5282ef UserName=test UserID=192.168.64.239 Time=2024/02/27 11:55:59 Operation=ExecuteStatement Stmt=[show tables] Result=SUCCESS Detail=Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
```

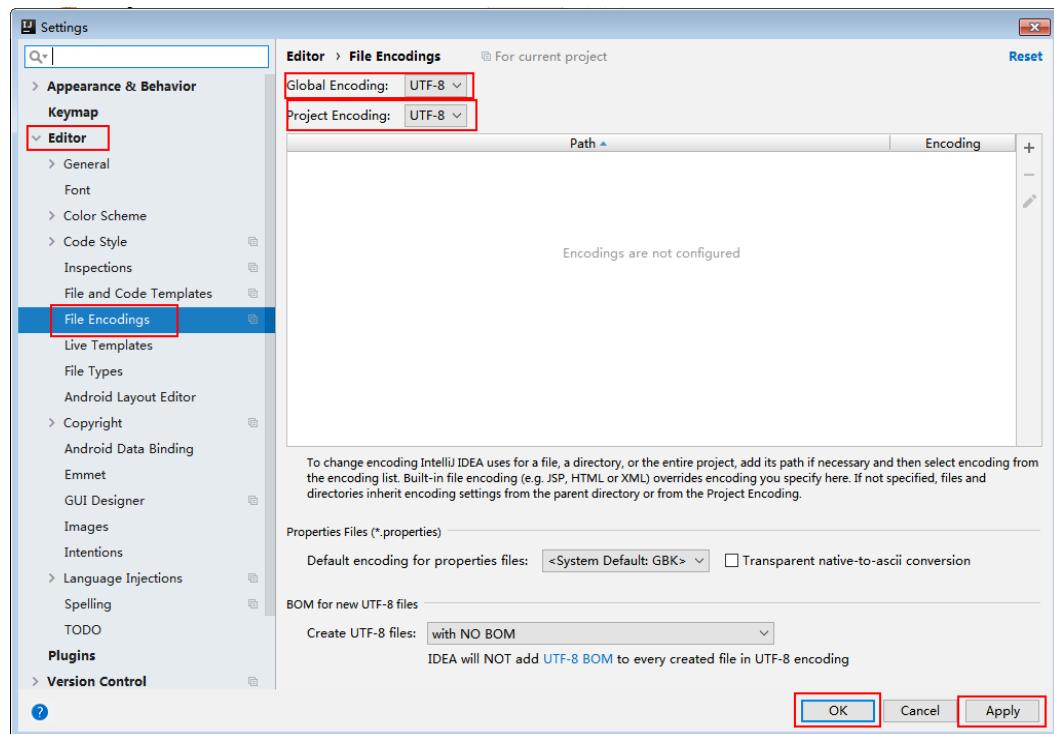
### Step 4 Import the example project to the IntelliJ IDEA development environment.

- On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
- In the displayed window, select the folder **hive-jdbc-example**, and click **OK**. On Windows, the path cannot contain any space.

### Step 5 Set the IntelliJ IDEA text file coding format to prevent garbled characters.

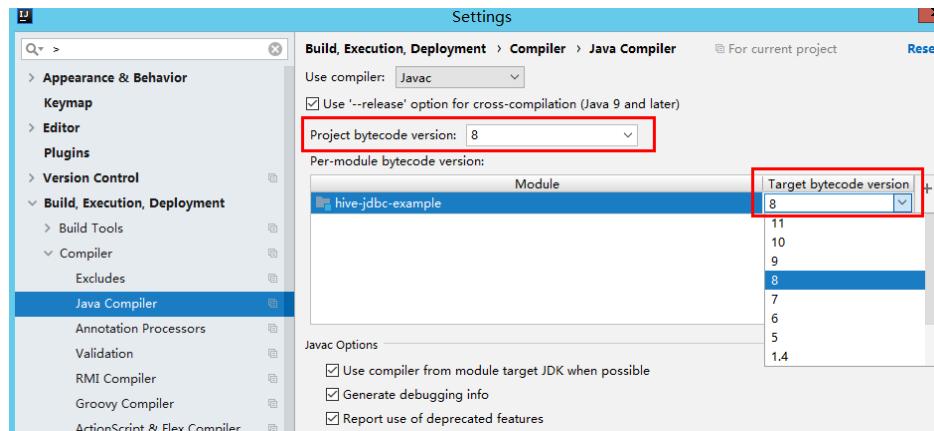
- On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.
- In the navigating tree on the left, choose **Editor > File Encodings**. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, and click **Apply** and **OK**, as shown in [Figure 1-164](#).

**Figure 1-164** Setting the IntelliJ IDEA coding format

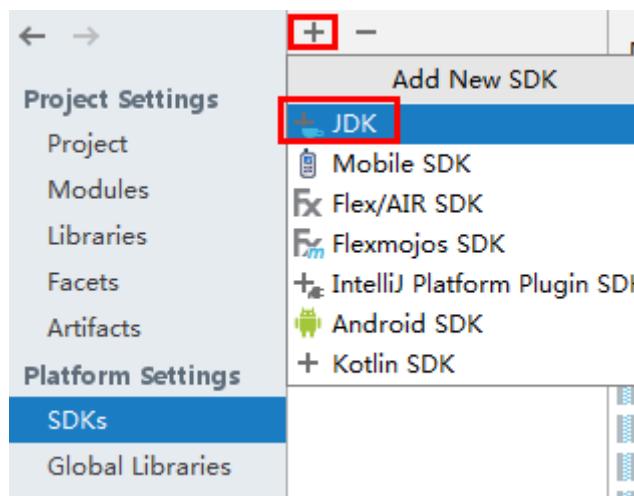


**Step 6** Set the JDK of the project.

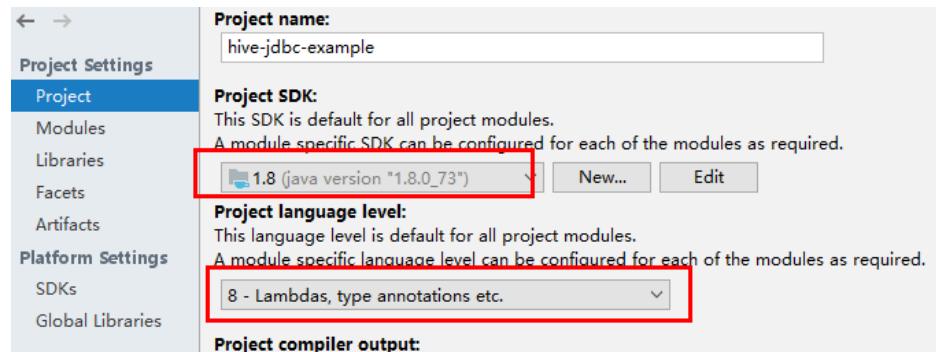
1. On the IntelliJ IDEA menu bar, choose **File > Settings....** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



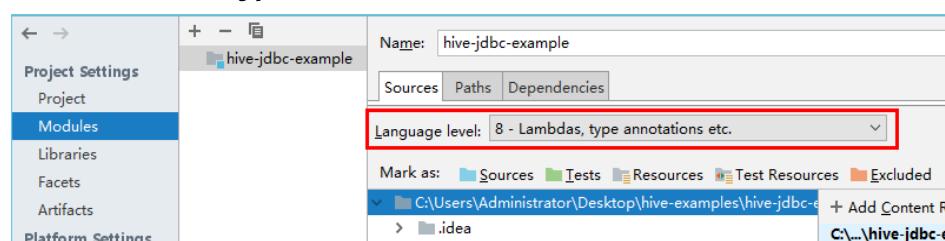
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



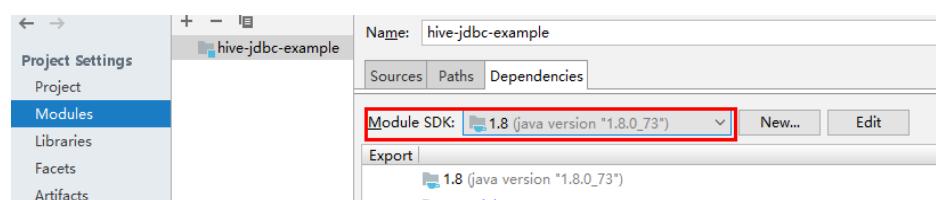
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in **SDKs**.

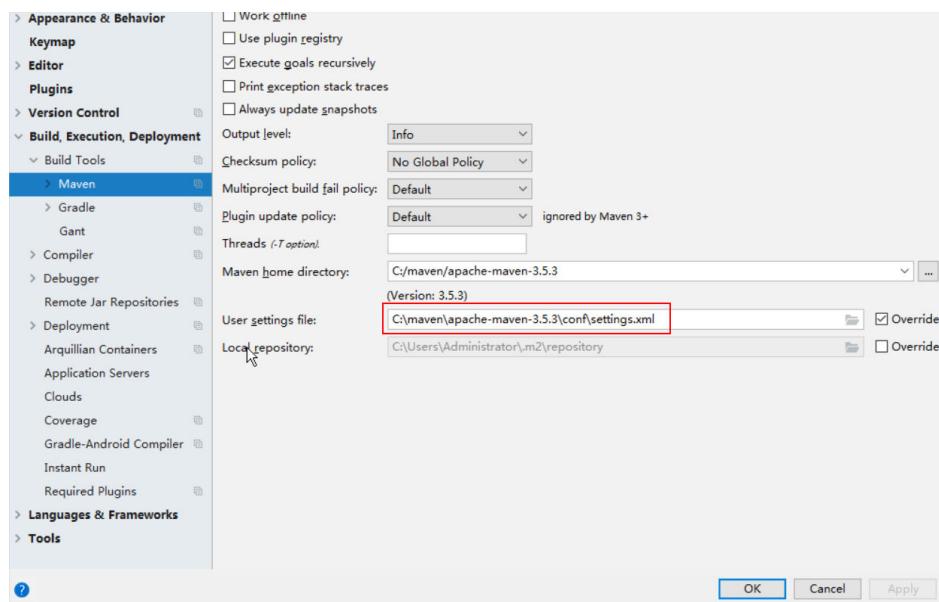


11. Click **Apply** and **OK**.

#### Step 7 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

Figure 1-165 Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

### 1.13.2.3 Configuring the Hcatalog Sample Project

#### Scenario

To run the HCatalog interface example codes of the Hive component of MRS, perform the following operations.

##### NOTE

The following uses the development of an application for connecting the Hive service in HCatalog mode on Windows as an example.

#### Procedure

**Step 1** Obtain the sample project folder **hcatalog-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** Import the example project to the IntelliJ IDEA development environment.

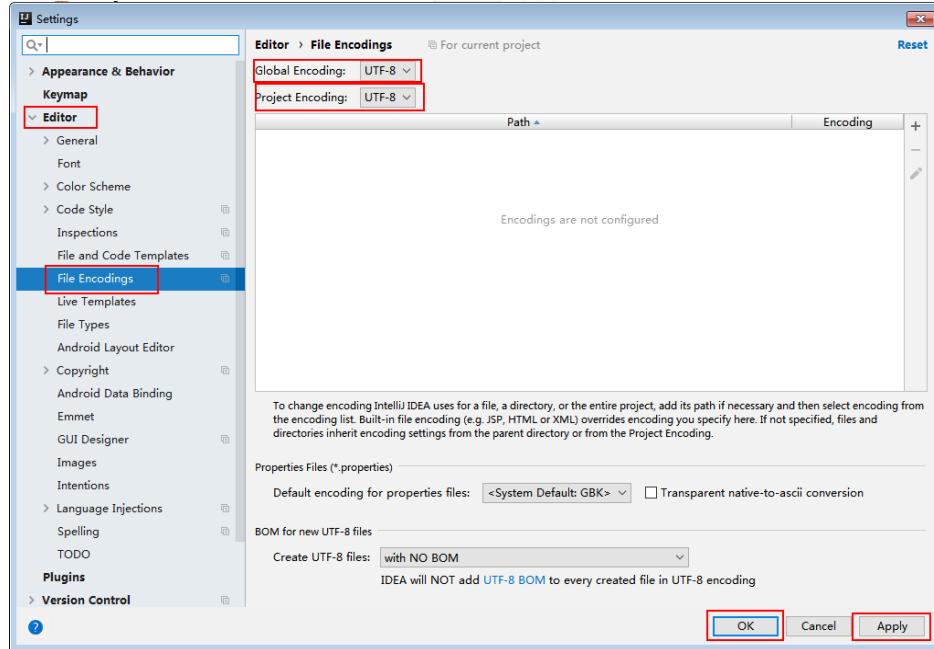
1. On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
2. In the displayed window, select the folder **hcatalog-example**, and click **OK**. On Windows, the path cannot contain any space.

**Step 3** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.

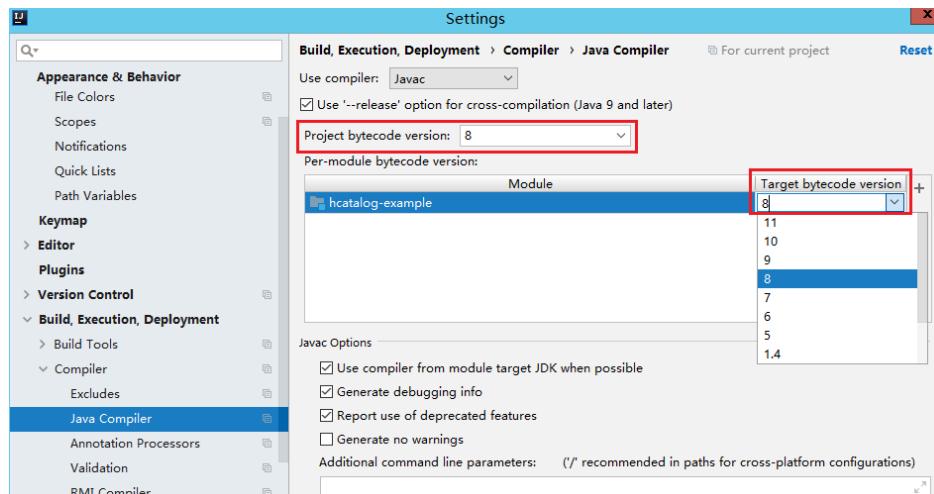
2. Choose **Editor > File Encodings** from the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 1-166](#).

**Figure 1-166** Setting the IntelliJ IDEA coding format

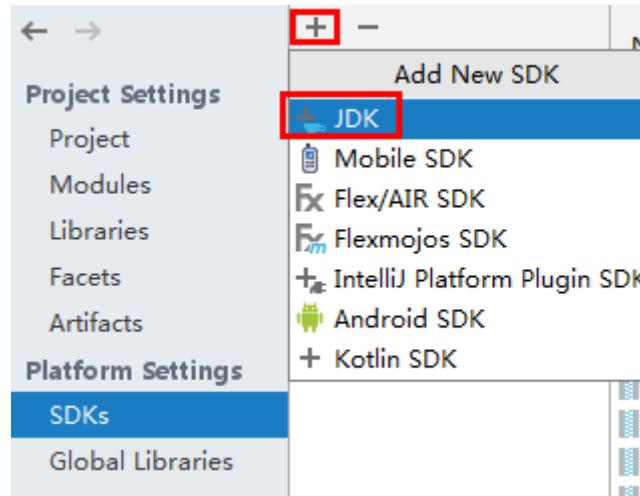


#### Step 4 Set the JDK of the project.

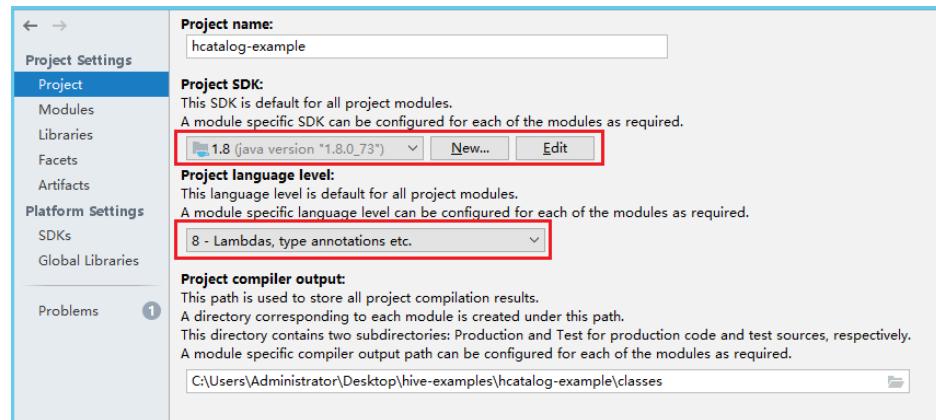
1. On the IntelliJ IDEA menu bar, choose **File > Settings....** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



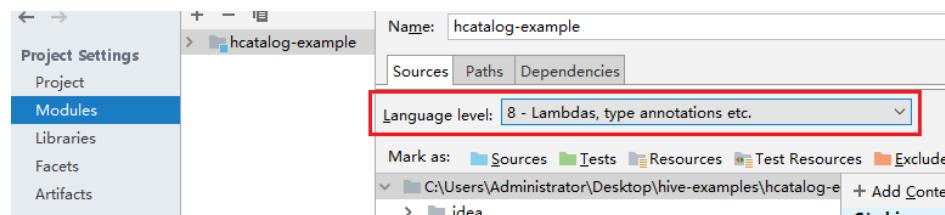
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



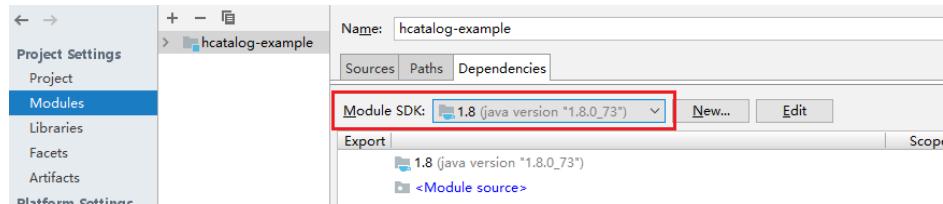
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



11. On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

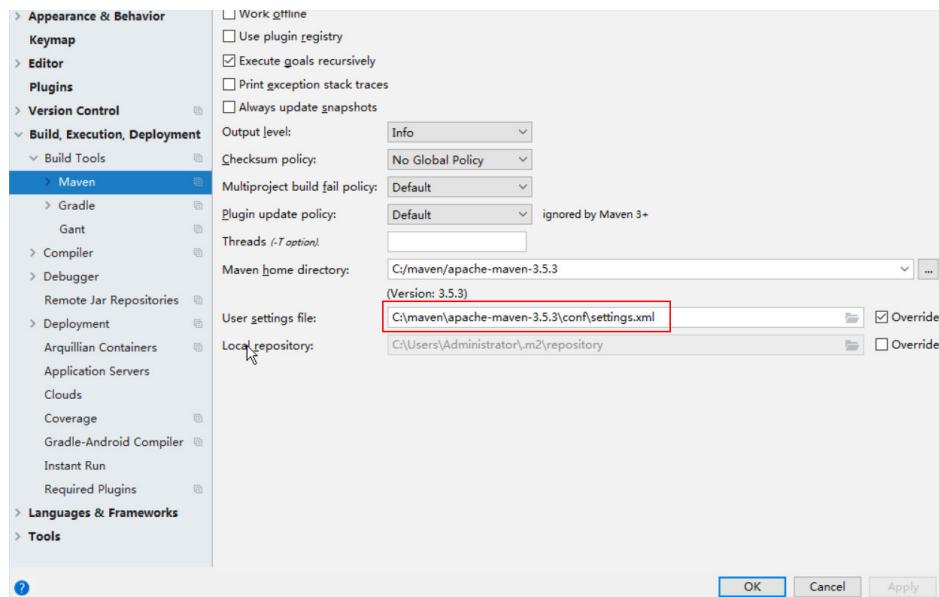


12. Click **Apply** and **OK**.

#### Step 5 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

Figure 1-167 Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

#### 1.13.2.4 Configuring the Python Sample Project

##### Scenario

To run the Python interface example codes of the Hive component of MRS, perform the following operations.

## Procedure

- Step 1** Python of 2.6.6 or a higher version has been installed on a client. The Python version cannot be higher than 2.7.13.

The Python version can be viewed by running the **python** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 2.6.6.

```
Python 2.6.6 (r266:84292, Oct 12 2012, 14:23:48)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

- Step 2** Setuptools of 5.0 or a higher version has been installed on a client. The setuptools version cannot be higher than 36.8.0.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python setup.py install** command in the CLI of the client.

The following information indicates that setuptools 5.7 is installed successfully.

```
Finished processing dependencies for setuptools==5.7
```

- Step 3** Install Python on the client.

1. Obtain the sample project folder **python-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python-examples** folder.
3. Go to the **python-examples** folder.
4. Run the **python setup.py install** command in the CLI.

The following information indicates that Python is installed successfully.

```
Finished processing dependencies for pyhs2==0.5.0
```

- Step 4** After the installation is successful, the following files are generated. **python-examples/pyCLI\_sec.py** is the Python client example codes. **python-examples/pyhs2/haconnection.py** is the Python client API. Run **hive\_python\_client** scripts to execute the SQL functions, for example, **sh hive\_python\_client 'show tables'**.



### NOTE

This function applies to only simple SQL statements and depends on the ZooKeeper client.

----End

## 1.13.2.5 Configuring the Python3 Sample Project

### Scenario

To run the Python3 interface example codes of the Hive component of MRS, perform the following operations.

## Procedure

- Step 1** Python3 of 3.6 or a higher version has been installed on a client. The Python3 version cannot be higher than 3.9.

The Python version can be viewed by running the **python3** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

- Step 2** Setuptools of 47.3.1 version has been installed on a client.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```



If the system displays a message indicating that the installation of setuptools of 47.3.1 fails, check whether the environment is faulty or whether the problem is caused by Python.

- Step 3** Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency\_python3.6**, **dependency\_python3.7**, or **dependency\_python3.8**, or **dependency\_python3.9** folder based on the Python3 version.
4. Run the **whereis easy\_install** command to find the path of the **easy\_install** program. If there are multiple paths, run the **easy\_install --version** command to select the **easy\_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy\_install**.
5. Run the **easy\_install** command to install the egg files in the **dependency\_python3.x** folder. If the egg file has dependencies, you can use wildcards to install the egg file. For example:
  - **dependency\_python3.6** directory:  
**/usr/local/bin/easy\_install future\*egg six\*egg python\*egg sasl-\*linux-\$(uname -p).egg thrift-\*egg thrift\_sasl\*egg**
  - **dependency\_python3.7** directory:  
**/usr/local/bin/easy\_install future\*egg six\*egg sasl-\*linux-\$(uname -p).egg thrift-\*egg thrift\_sasl\*egg**
  - **dependency\_python3.8** directory:  
**/usr/local/bin/easy\_install future\*egg six\*egg python\*egg sasl-\*linux-\$(uname -p).egg thrift-\*linux-\$(uname -p).egg thrift\_sasl\*egg**

- dependency\_python3.9 directory:

```
/usr/local/bin/easy_install future*egg six*egg sasl-*linux-$(uname -p).egg six-*.egg thrift-*linux-$(uname -p).egg thrift_sasl*egg
```

If the following information is displayed for each egg file, the installation is successful:

```
Finished processing dependencies for ***
```

**Step 4** After the installation is successful, the following files are generated. **python3-examples/pyCLI\_sec.py** is the Python client example codes. **python3-examples/pyhive/hive.py** is the Python client API.

----End

### 1.13.2.6 Configuring a Spring Boot Sample Project

#### Scenario

Run the Spring Boot interface sample code of MRS Hive.

The following example develops an application that uses SpringBoot to connect to Hive in the Windows environment.

#### Procedure

- Step 1** Obtain the sample project folder **hive-rest-client-example** in the **src\springboot\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Save the keytab files **user.keytab** and **krb5.conf** obtained when a developer account is prepared by referring to [Preparing a Developer Account](#) to the **hive-rest-client-example\src\main\resources** directory of the sample project.
- Step 3** Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Hive\config** and copy the **core-site.xml** and **hiveclient.properties** files to the **hive-jdbc-example\src\main\resources** directory of the sample project.



#### NOTE

Hive allows you to add extension identifiers to JDBC connection strings. These extension identifiers are printed in HiveServer audit logs to distinguish SQL sources. If you need to configure an identifier, add the following content to the **hive-jdbc-example\src\main\resources\hiveclient.properties** file:

```
auditAddition=xxx
```

**xxx** is the custom identifier. The identifier can contain a maximum of 256 bytes and only letters, digits, underscores (\_), commas (,), and colons (:) are allowed.

After the SQL statement is executed, the configured extension identifier is printed in the HiveServer audit log **/var/log/Bigdata/audit/hive/hiveserver/hive-audit.log**. The following is an example.

```
2024-02-27 11:55:56,501 | INFO  | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserId=192.168.64.230 Time=2024/02/27 11:55:56      Operation=OpenSession Result= Data
11=Addition=xx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:56,501 | INFO  | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserId=192.168.64.230 Time=2024/02/27 11:55:56      Operation=OpenSession Result=SUCCE
SS Detail= Addition=xx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:59,212 | INFO  | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserId=192.168.64.230 Time=2024/02/27 11:55:59      Operation=ExecuteStatement  stmt
=(show tables)  Result= Detail= Addition=xx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:59,336 | INFO  | HiveServer2-Handler-Pool: Thread-402 | OperationId=7187e0c-b8d3-42f0-80d3-760babf5202ef  UserName=test UserId=192.168.64.230 Time=2024/02/27 11:55:59      Operation=ExecuteStatement  stmt
=(ThriftCLIService.java:512)
```

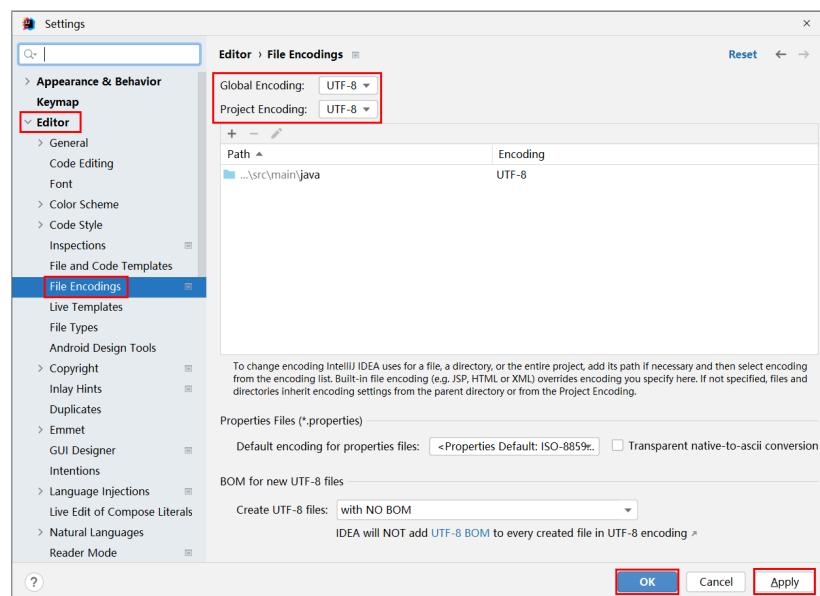
**Step 4** Import the sample project to the IntelliJ IDEA development environment.

1. On the menu bar of IntelliJ IDEA, choose **File > Open....**
2. In the **Open File or Project** dialog box that is displayed, select the **hive-rest-client-example** folder and click **OK**. In Windows, the folder path cannot contain any space.

**Step 5** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

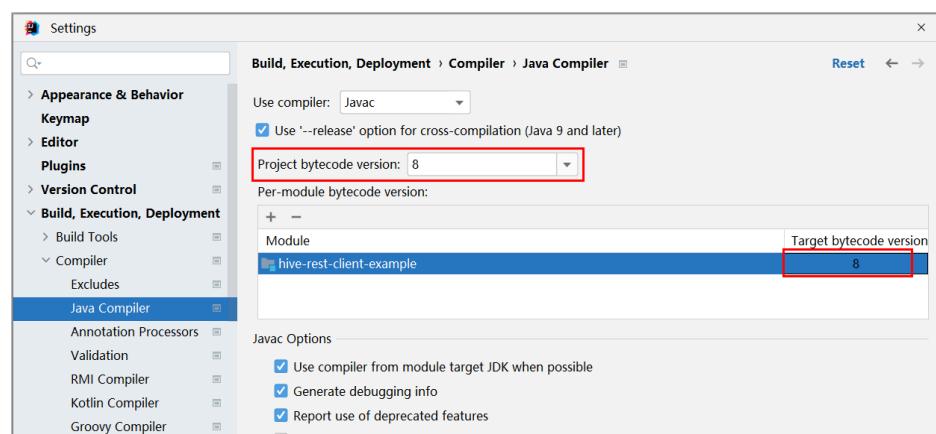
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. In the left navigation pane, choose **Editor > File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**.

**Figure 1-168** Setting the IntelliJ IDEA coding format

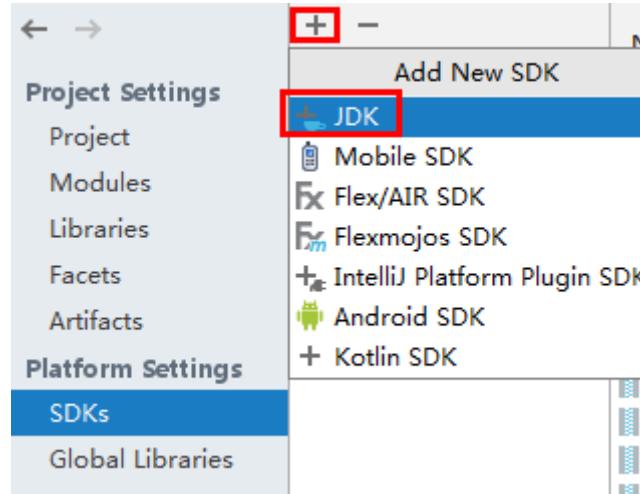


**Step 6** Set the JDK of the project.

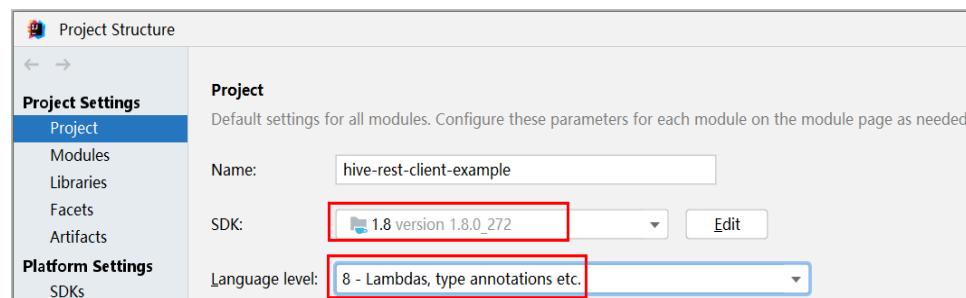
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **hive-rest-client-example** to **8**.



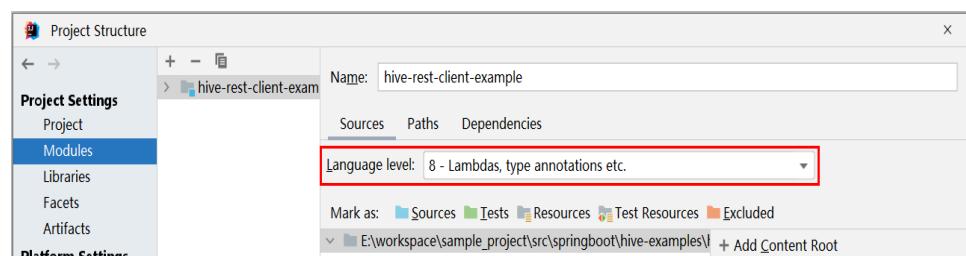
3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.



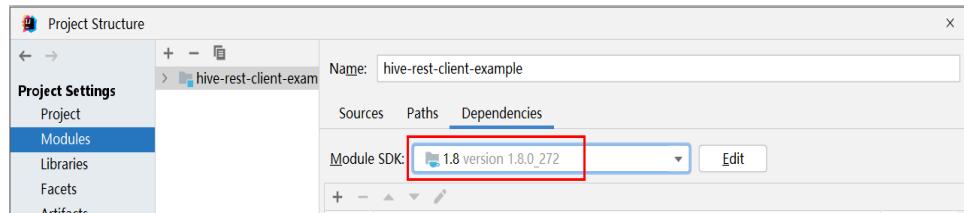
6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc..**



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.

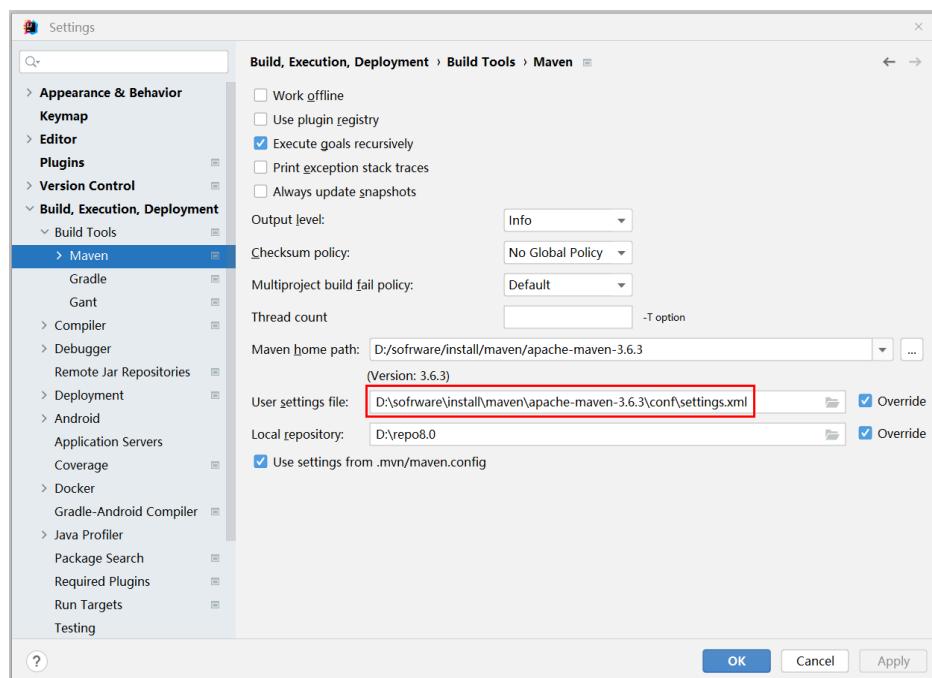


11. Click **Apply** then **OK**.

**Step 7** Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

**Figure 1-169** Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply** then **OK**.

**Step 8** Find the **hiveclient.properties** configuration file in **src\main\resources** and add the following content to the file. Replace **xxx** with the development user created during development account preparation.

```
user.name=xxx
```

----End

## 1.13.3 Developing an Application

### 1.13.3.1 Typical Scenario Description

#### Scenarios

Assume that you need to develop a Hive data analysis application to manage the employee information described in [Table 1-124](#) and [Table 1-125](#).

#### Development Guidelines

##### Step 1 Prepare data.

1. Create three tables: employee information table **employees\_info**, employee contact table **employees\_contact**, and extended employee information table **employees\_info\_extended**.
  - Employee information table **employees\_info** contains fields such as employee ID, name, salary currency, salary, tax category, work place, and hire date. In salary currency, **R** indicates RMB and **D** indicates USD.
  - Fields in the **employees\_contact** table include the employee ID, phone number, and email address.
  - Fields in the **employees\_info\_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hire date.
2. Load employee information to **employees\_info**.  
For data loading codes, see [Loading Data](#).

[Table 1-124](#) describes employee information.

**Table 1-124** Employee information

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wang	R	8000.01	personal income tax&0.05	Country 1:City1	2014
3	Tom	D	12000.02	personal income tax&0.09	Country 2:City2	2014
4	Jack	D	24000.03	personal income tax&0.09	Country 3:City3	2014
6	Linda	D	36000.04	personal income tax&0.09	Country 4:City4	2014

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
8	Zhang	R	9000.05	personal income tax&0.05	Country5:City5	2014

3. Load employee contact information to **employees\_contact**.

**Table 1-125** describes employee contact information.

**Table 1-125** Employee contact information

ID	Mobile Phone Number	E-mail Address
1	135 XXXX XXXX	xxxx@xx.com
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

4. Load extended employee information to **employees\_info\_extended**.

**Table 1-126** describes the extended employee information.

**Table 1-126** Extended employee information

ID	Name	Mobile Phone Number	E-mail Address	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wa ng	135 XXXX XXXX	xxxx@x x.com	R	8000 .01	personal income tax&0.0 5	Country1 :City1	201 4
3	To m	159 XXXX XXXX	xxxxx@ xx.com. cn	D	1200 0.02	personal income tax&0.0 9	Country2 :City2	201 4
4	Jack	186 XXXX XXXX	xxxx@x x.org	D	2400 0.03	personal income tax&0.0 9	Country3 :City3	201 4

ID	Name	Mobile Phone Number	E-mail Address	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
6	Linda	189 XXXX XXXX	xxxx@xx.cn	D	3600 0.04	personal income tax&0.09	Country4 :City4	2014
8	Zhang	134 XXXX XXXX	xxxx@xxx.cn	R	9000 .05	personal income tax&0.05	Country5 :City5	2014

## Step 2 Analyze data.

For data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load the query results to the partition with the hiring date of 2014 in **employees\_info\_extended**.
- Collect the number of records in the **employees\_info** table.
- Query information about employees whose email addresses end with "cn".

## Step 3 Submit a data analysis task to collect the number of records in the **employees\_info** table. For details about the implementation, see [Example Program Guide](#).

----End

### 1.13.3.2 Example Codes

#### 1.13.3.2.1 Creating a Table

##### Function

This section describes how to use Hive Query Language (HQL) to create internal and external tables. You can create a table in three modes:

- Customize the table structure, and use the key word **EXTERNAL** to differentiate between internal and external tables.
  - If data is to be processed by Hive, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted together.
  - If data is to be processed by multiple tools, create an external table. When an external table is deleted, only metadata is deleted.
- Create a table based on existing tables. Use **CREATE LIKE** to fully copy the original table structure, including the storage format.

- Create a table based on query results using CREATE AS SELECT.

In this mode, you can specify fields (except for the storage format) that you want to copy to the new table when copying the structure of the existing table.

#### NOTE

- To perform the following operations on a cluster enabled with the security service, you must have the CREATE permission for databases. To create a table using the CREATE AS SELECT statement, you must have the SELECT permission for tables. For details about the permission requirements, see [Overview](#).
- Both the table name and field name can contain a maximum of 128 bytes. Both the field comment and value can contain a maximum of 4,000 bytes. The key in **WITH SERDEPROPERTIES** can contain a maximum of 256 bytes.

## Sample Codes

```
-- Create an external table employees_info.  
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info  
(  
    id INT,  
    name STRING,  
    usd_flag STRING,  
    salary DOUBLE,  
    deductions MAP<STRING, DOUBLE>,  
    address STRING,  
    entrytime STRING  
)  
-- Specify the field delimiter.  
-- "delimited fields terminated by" indicates that the column delimiter is ',' and "MAP KEYS TERMINATED BY" indicates that the delimiter of key values in the MAP is '&'.  
ROW FORMAT delimited fields terminated by ',' MAP KEYS TERMINATED BY '&'  
-- Set the storage format to TEXTFILE.  
STORED AS TEXTFILE;  
  
-- Create an external table employees_contact.  
CREATE EXTERNAL TABLE IF NOT EXISTS employees_contact  
(  
    id INT,  
    tel_phone STRING,  
    email STRING  
)  
ROW FORMAT delimited fields terminated by ','  
STORED AS TEXTFILE;  
  
-- Create an external table employees_info_extended.  
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended(id INT, name STRING, usd_flag STRING,  
salary DOUBLE, deductions MAP<STRING, DOUBLE>, address STRING)  
-- A table may have one or multiple partitions. Each partition is saved as an independent folder in the table directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data based on certain criteria.  
-- Use PARTITIONED BY to specify the column name and data type of the partition.  
PARTITIONED BY(entrytime STRING)  
ROW FORMAT delimited fields terminated by ',' MAP KEYS TERMINATED BY '&'  
STORED AS TEXTFILE;  
-- After a table is created, you can use ALTER TABLE to add or delete fields to or from the table, modify table attributes, and add partitions.  
-- Add the tel_phone and email fields to the employees_info_extended table.  
ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
```

### 1.13.3.2.2 Loading Data

#### Function

This section describes how to use HQL to load data to the existing **employees\_info** table. You can learn how to load data from a local file system and MRS cluster. LOCAL is used to differentiate between local and non-local data sources.

##### NOTE

To perform the following operations on a cluster enabled with the security service, you must have the UPDATE permission for databases, and owner permission and read/write permission for files to be loaded. For details about permission requirements, see [Overview](#).

If LOCAL is used in data loading statements, data is loaded from a local directory. In addition to the UPDATE permission for tables, you must have the read permission for the data path. It is also required that the data can be accessed by user **omm** on the active HiveServer.

If OVERWRITE is used in data loading statements, the existing data in a table will be overwritten by new data. If OVERWRITE does not exist, data will be added to the table.

#### Example Codes

```
-- Load the employee_info.txt file from the /opt/hive_examples_data/ directory of the local file system to the employees_info table.  
---- Overwrite the original data using the new data.  
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE employees_info;  
---- Retain the original data and add the new data to the table.  
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' INTO TABLE employees_info;  
  
-- Load /user/hive_examples_data/employee_info.txt from HDFS to the employees_info table.  
---- Overwrite the original data using the new data.  
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE employees_info;  
---- Retain the original data and add the new data to the table.  
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' INTO TABLE employees_info;
```

##### NOTE

The essence of loading data is to copy the data to the specified table directory in HDFS.

#### Sample Data

Data in **employees\_info** is as follows:

```
1,Wang,R,8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014  
3,Tom,D,12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014  
4,Jack,D,24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014  
6,Linda,D,36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014  
8,Zhang,R,9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

Data in **employees\_contact** is as follows:

```
1,135 XXXX XXXX,xxxx@xx.com  
3,159 XXXX XXXX,xxxx@xx.com.cn  
4,186 XXXX XXXX,xxxx@xx.org  
6,189 XXXX XXXX,xxxx@xxx.cn  
8,134 XXXX XXXX,xxxx@xxxx.cn
```

Data in **employees\_info\_extended** is as follows:

```
1,Wang,135 XXXX  
XXXX,xxxx@xx.com,R,8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014
```

```
3,Tom,159 XXXX  
XXXX,xxxx@xx.com.cn,D,12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014  
4,Jack,186 XXXX  
XXXX,xxx@xx.org,D,24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014  
6,Linda,189 XXXX  
XXXX,xxx@xxx.cn,D,36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014  
8,Zhang,134 XXXX  
XXXX,xxx@xxx.cn,R,9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

### 1.13.3.2.3 Querying Data

#### Function

This topic describes how to use Hive Query Language (HQL) to query and analyze data. You can query and analyze data using the following methods:

- Use common features for SELECT query, such as JOIN.
- Load data to a specified partition.
- Use Hive-provided functions.
- Query and analyze data using user-defined functions (UDFs). For details about how to create and define UDFs, see [UDF](#).

#### NOTE

To perform the following operations on a cluster enabled with the security service, you must have related permissions for tables. For details about permission requirements, see [Overview](#).

#### Example Codes

```
-- Query the contact information of employees whose salaries are paid in USD.  
SELECT  
a.name,  
b.tel_phone,  
b.email  
FROM employees_info a JOIN employees_contact b ON(a.id = b.id) WHERE usd_flag='D';  
  
-- Query the IDs and names of employees who were hired in 2014, and load query results to the partition  
with the hiring time of 2014 in the employees_info_extended table.  
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')  
SELECT  
a.id,  
a.name,  
a.usd_flag,  
a.salary,  
a.deductions,  
a.address,  
b.tel_phone,  
b.email  
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';  
  
-- Use the existing Hive function COUNT() to count the number of records in the employees_info table.  
SELECT COUNT(*) FROM employees_info;  
  
-- Query information about employees whose email addresses end with "cn".  
SELECT a.name, b.tel_phone FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE  
b.email like '%cn';
```

#### Extensions

- Configure intermediate Hive data encryption.  
Set the table format to RCFile (recommended) or SequenceFile, and the encryption algorithm to ARC4Codec. SequenceFile is a unique Hadoop file

format, and RCFile is a Hive file format with optimized column storage. When a big table is queried, RCFile provides higher performance than SequenceFile.

```
set hive.exec.compress.output=true;
set hive.exec.compress.intermediate=true;
set hive.intermediate.compression.codec=org.apache.hadoop.io.encryption.arc4.ARC4Codec;
```

- For details about UDFs, see [UDF](#).

#### 1.13.3.2.4 UDF

When internal functions of Hive cannot meet requirements, you can compile user-defined functions (UDFs) and use them for query.

According to implementation methods, UDFs are classified as follows:

- Common UDFs: used to perform operations on a single data row and export a single data row.
- User-defined aggregating functions (UDAFs): used to input multiple data rows and export a single data row.
- User-defined table-generating functions (UDTFs): used to perform operations on a single data row and export multiple data rows.

According to usage methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent function: used in multiple sessions and do not have to be created every time a session restarts.

The following uses AddDoublesUDF as an example to describe how to compile and use UDFs.

## Function

AddDoublesUDF is used to add two or multiple floating point values. The following example describes how to compile and use UDFs.

### NOTE

- The normal UDF must be originated from [org.apache.hadoop.hive.ql.exec.UDF](#).
- The normal UDF must implement at least one evaluate(). The evaluate function supports overloading.
- To develop a customized function, you need to add the hive-exec-3.1.0.jar dependency package to the project. The package can be obtained from the Hive installation directory.

## Example Codes

The following is a UDF example:

```
package com.huawei.bigdata.hive.example.udf;
import org.apache.hadoop.hive.ql.exec.UDF;

public class AddDoublesUDF extends UDF {
    public Double evaluate(Double... a) {
        Double total = 0.0;
        // Processing logic
        for (int i = 0; i < a.length; i++)
```

```
    if (a[i] != null)
        total += a[i];
    return total;
}
```

## How to Use

- Step 1** Package the preceding program into **AddDoublesUDF.jar**, and upload it to a directory on the HDFS (such as `/user/hive_examples_jars/`). The user who creates the UDF and the user who uses the UDF function must have read right on this JAR file. Example statements:

```
hdfs dfs -put ./hive_examples_jars /user/hive_examples_jars
```

```
hdfs dfs -chmod 777 /user/hive_examples_jars
```

- Step 2** Use a user with admin rights to log in to the Beeline client and run the following command:

```
kinit Hive service user
```

```
beeline
```

```
set role admin;
```

- Step 3** Define the function in Hive Server. Run the following SQL statement to create a permanent function:

```
CREATE FUNCTION addDoubles AS
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

*addDoubles* indicates the function alias that is used for SELECT query.

Run the following statement to create a temporary function:

```
CREATE TEMPORARY FUNCTION addDoubles AS
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

- *addDoubles* indicates the function alias that is used for SELECT query.
- TEMPORARY indicates that the function is used only in the current session with the Hive server.

- Step 4** Run the following SQL statement to use the function in the Hive server:

```
SELECT addDoubles(1,2,3);
```



If an **[Error 10011]** error is displayed when you log in to the client again, run the **reload function**; command and then use this function.

- Step 5** Run the following SQL statement to delete the function from the Hive server:

```
DROP FUNCTION addDoubles;
```

----End

## Extensions

None

### 1.13.3.2.5 Example Program Guide

## Function

This section describes how to use an example program to complete an analysis task. An example program can submit a task by using the following methods:

- Submitting a data analysis task by using JDBC interfaces
- Submitting a data analysis task by using Python

## Example Codes

- Submit a data analysis task using the Hive Java database connectivity (JDBC) interface, that is, JDBCExample.java.
  - Read the **property** file of the HiveServer client. The **hiveclient.properties** file is saved in the **resources** directory of the JDBC example program provided by Hive.

```
Properties clientInfo = null;
String userdir = System.getProperty("user.dir") + File.separator
+ "conf" + File.separator;
InputStream fileInputStream = null;
try{
    clientInfo = new Properties();
    //hiveclient.properties indicates the client configuration file. If the multiple-service feature is
    used, the file must be replaced with the hiveclient.properties file on the instance client.
    //hiveclient.properties is located under the config directory of the directory where the instance
    client installation package is decompressed.
    String hiveclientProp = userdir + "hiveclient.properties";
    File propertiesFile = new File(hiveclientProp);
    fileInputStream = new FileInputStream(propertiesFile);
    clientInfo.load(fileInputStream);
} catch (Exception e) {
    throw new IOException(e);
} finally{
    if(fileInputStream != null){
        fileInputStream.close();
        fileInputStream = null;
    }
}
```

- Obtain the IP address and port list of ZooKeeper, the cluster authentication mode, the SASL configuration of HiveServers, node names of HiveServers in ZooKeeper, the discovery mode from the client to the server, and the principal server process for user authentication. You can read all these configurations from the **hiveclient.properties** file.

```
//The format of zkQuorum is
xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002";
//xxx.xxx.xxx.xxx is the IP address of the node where ZooKeeper resides. The default port is
24002.
```

```
zkQuorum = clientInfo.getProperty("zk.quorum");
auth = clientInfo.getProperty("auth");
sasl_qop = clientInfo.getProperty("sasl.qop");
zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
principal = clientInfo.getProperty("principal");
auditAddition = clientInfo.getProperty("auditAddition");
```

- c. In security mode, the kerberos user and keytab file path are required for login authentication. For details about how to obtain **USER\_NAME**, **USER\_KEYTAB\_FILE**, and **KRB5\_FILE**, see [Running JDBC and Viewing Results](#).

```
// Set the userName of new user.
USER_NAME = "xxx";
// Set the keytab and krb5 files location of client.
String userdir = System.getProperty("user.dir") + File.separator
+ "conf" + File.separator;
USER_KEYTAB_FILE = userdir + "user.keytab";
KRB5_FILE = userdir + "krb5.conf";
```

- d. Define HQL. HQL must be a single statement and cannot contain ";".

```
// Define HQL. HQL cannot contain ";"
String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
"SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};
```

- e. Build JDBC URL.

#### NOTE

You can also implement pre-authentication without the need of providing the account and keytab file path. For details, see JDBC code example 2 in [Examples](#). If IBM JDK is used to run Hive applications, pre-authentication in JDBC sample code 2 must be implemented.

The following is an example of the JDBC URL composed of code snippets:

```
jdbc:hive2://
xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx:24002,xxx.xxx.xxx:24002;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2;sasl.qop=auth-conf;auth=KERBEROS;principal=hive/hadoop.<system domain name>@<system domain name>

// Concat JDBC URL
StringBuilder sBuilder = new StringBuilder(
    "jdbc:hive2://"").append(zkQuorum).append("/");

if ("KERBEROS".equalsIgnoreCase(auth)) {
    sBuilder.append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";sasl.qop=")
        .append(sasl_qop)
        .append(";auth=")
        .append(auth)
        .append(";principal=")
        .append(principal)
        .append(";user.principal=")
        .append(USER_NAME)
        .append(";user.keytab=")
        .append(USER_KEYTAB_FILE);
}

} else {
    // Normal mode
    sBuilder.append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";auth=none;");
}
if (auditAddition != null && !auditAddition.isEmpty()) {
    strBuilder.append(";auditAddition=").append(auditAddition);
}
String url = sBuilder.toString();
```

- f. Load the Hive JDBC driver.

```
// Load the Hive JDBC driver.  
Class.forName(HIVE_DRIVER);
```

- g. Obtain the JDBC connection, confirm the HQL type (DDL/DML), call ports to run the HQL statement, return the queried column name and results to the console, and close the JDBC connection.

```
Connection connection = null;  
try {  
    // Obtain the JDBC connection.  
    // If the normal mode is used, the second parameter needs to be set to a correct username.  
    // Otherwise, the anonymous user will be used for login.  
    connection = DriverManager.getConnection(url, "", "");  
  
    // Create a table  
    // To import data to a table after the table is created, you can use the LOAD statement. For  
    // example, import data from the HDFS to the table.  
    //load data inpath '/tmp/employees.txt' overwrite into table employees_info;  
    execDDL(connection,sqls[0]);  
    System.out.println("Create table success!");  
  
    // Query  
    execDML(connection,sqls[1]);  
  
    // Delete the table  
    execDDL(connection,sqls[2]);  
    System.out.println("Delete table success!");  
}  
finally {  
    // Close the JDBC connection.  
    if (null != connection) {  
        connection.close();  
    }  
  
public static void execDDL(Connection connection, String sql)  
throws SQLException {  
    PreparedStatement statement = null;  
    try {  
        statement = connection.prepareStatement(sql);  
        statement.execute();  
    }  
    finally {  
        if (null != statement) {  
            statement.close();  
        }  
    }  
}  
  
public static void execDML(Connection connection, String sql) throws SQLException {  
    PreparedStatement statement = null;  
    ResultSet resultSet = null;  
    ResultSetMetaData metaData = null;  
  
    try {  
        // Run the HQL statement.  
        statement = connection.prepareStatement(sql);  
        resultSet = statement.executeQuery();  
  
        // Return the queried column name to the console.  
        metaData = resultSet.getMetaData();  
        int columnCount = metaData.getColumnCount();  
        for (int i = 1; i <= columnCount; i++) {  
            System.out.print(metaData.getColumnName(i) + '\t');  
        }  
        System.out.println();  
  
        // Return the results to the console.  
    }
```

```
        while (resultSet.next()) {
            for (int i = 1; i <= columnCount; i++) {
                System.out.print(resultSet.getString(i) + '\t');
            }
            System.out.println();
        }
    } finally {
    if (null != resultSet) {
        resultSet.close();
    }

    if (null != statement) {
        statement.close();
    }
}
```

- Submit a data analysis task using the Python interface, that is, **python-examples/pyCLI\_sec.py**. The authentication mode of the cluster to which the example program connects is the secure mode. Before running the example program, run the kinit command to authenticate the kerberos user with related rights.

- Import the HAConnection class.

```
from pyhs2.haconnection import HAConnection
```

- Declare the HiveServer IP address list. In this example, **hosts** indicate the nodes of HiveServer, and **xxx.xxx.xxx.xxx** indicates the service IP address.

```
hosts = ["xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"]
```

#### NOTE

- If the HiveServer instance is migrated, the original sample program is invalid. You need to update the IP address of the HiveServer after the migration of the HiveServer instance used in the sample program.
- If multiple Hive instances are used, update the IP address based on the address of the instance that is actually connected.
- Configure the kerberos host name and service name. In this example, the kerberos host name is hadoop and service name is hive.

```
conf = {"krb_host": "hadoop.<system domain name>", "krb_service": "hive"}
```

#### NOTE

If multiple Hive instances are used, change the service name based on the cluster that is actually connected. For example, if the Hive1 instance is connected, change the service name to hive1.

- Create a connection, run the HQL statement, and return the queried column name and results to the console.

```
try:
    with HAConnection(hosts = hosts,
                      port = 21066,
                      authMechanism = "KERBEROS",
                      configuration = conf) as haConn:
        with haConn.getConnection() as conn:
            with conn.cursor() as cur:
                # show databases
                print cur.getdatabases()

                # execute query
```

```
cur.execute("show tables")

# return column info from query
print cur.getschema()

# fetch table results
for i in cur.fetch():
    print i

except exception, e:
    print e
```

#### NOTE

If multiple Hive instances are used, you need to modify hosts according to the description in **b** and change the port number based on to the actual port number. The default ports of Hive to Hive4 are 21066 to 21070, respectively.

### 1.13.3.2.6 Accessing Multiple ZooKeepers

#### Description

This section describes how to access both FusionInsight ZooKeeper and the third-party ZooKeeper in the same client process using the `testConnectHive` and `testConnectApacheZK` methods respectively.

In the `JDBCExample` class of the `hive-jdbc-example-multizk` package, the code structure of the `main` method is as follows:

```
public static void main(String[] args) throws InstantiationException, IllegalAccessException,
ClassNotFoundException, SQLException, IOException{
    testConnectHive(); // Method of accessing FusionInsight ZooKeeper
    testConnectApacheZk(); // Method of accessing the open-source ZooKeeper
}
```

#### Accessing FusionInsight ZooKeeper

If only the method of accessing FusionInsight ZooKeeper needs to be executed, comment out the `testConnectApacheZk` method in the `main` function.

Before using the `testConnectHive` method to access FusionInsight ZooKeeper, perform the following operations:

- Step 1** Change the value of `USER_NAME` in the `init` method in `JDBCExample`.  
`USER_NAME` indicates the user that is used to access FusionInsight ZooKeeper and has permissions of the FusionInsight Hive and Hadoop common user groups.
- Step 2** Go to the client decompression path  
`FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\Hive\config` and manually import the `core-site.xml` and `hiveclient.properties` files to the `hive-jdbc-example-multizk\src\main\resources` directory of the sample project.
- Step 3** Download the `krb5.conf` and `user.keytab` files of the user to the `resources` directory in the `hive-jdbc-example-multizk` package.
- Step 4** Check and change the values of `zk.port` and `zk.quorum` in the `hiveclient.properties` file in the `resources` directory.
  - `zk.port`: indicates the port for accessing FusionInsight ZooKeeper. Generally, the default value is used. Change the value as required.

- **zk.quorum:** indicates the IP address for accessing ZooKeeper quorumpeer. Set it to the IP address of the cluster deployed with the FusionInsight ZooKeeper service.

----End

## Accessing Open-Source ZooKeeper

To use **testConnectApacheZk** to connect to the open-source ZooKeeper code, change xxx.xxx.xxx.xxx in the following code to the IP address of the open-source ZooKeeper to be connected. Change the port number as required. After the connection is used, run the try statement in try-with-resources mode to automatically close the ZooKeeper connection. If only the sample for accessing the third-party ZooKeeper needs to be executed, comment out the **testConnectHive** method in the **main** function.

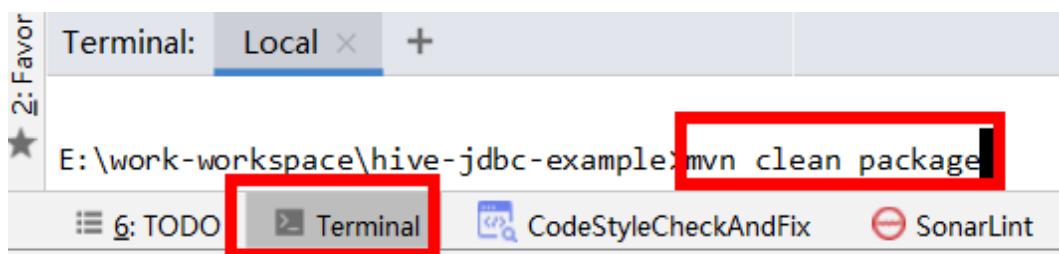
```
private static void testConnectApacheZk() {  
    //After the try statement is executed, the ZooKeeper connection is automatically disabled to prevent  
    //connection leakage.  
    try (org.apache.zookeeper.ZooKeeper digestZk =  
        new org.apache.zookeeper.ZooKeeper("xxx.xxx.xxx.xxx:port", 600000, null)) {  
        while (true) {  
            if (digestZk.getState().isConnected()) {  
                List<String> nodes = digestZk.getChildren("/", null);  
                logger.info("digest root path:{}", nodes);  
                for (String node : nodes) {  
                    List<String> subNodes = digestZk.getChildren "/" + node, null);  
                    logger.info("{} child path:{}", node, subNodes);  
                    for (String subNode : subNodes) {  
                        logger.info(  
                            "child patch:{}", subNode, digestZk.getChildren "/" + node + "/" + subNode, null));  
                    }  
                }  
                break;  
            }  
            Thread.sleep(1000);  
        }  
    } catch (IOException | KeeperException | InterruptedException e) {  
        logger.error("failed to connect zookeeper", e);  
    }  
}
```

## 1.13.4 Commissioning Applications

### 1.13.4.1 Running JDBC and Viewing Results

#### Running JDBC in CLI Mode

**Step 1** In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean package** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. A JAR file containing the **-with-dependencies** field is generated in the **target** directory of the sample project.

```
Terminal: Local +  
[INFO] com/ already added, skipping  
[INFO] com/huawei/ already added, skipping  
[INFO] com/huawei/bigdata/ already added, skipping  
[INFO] META-INF/maven/ already added, skipping  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 32.933 s  
[INFO] Finished at: 2020-11-23T16:18:08+08:00  
[INFO] -----
```

**Step 2** Create a directory on Windows or Linux as the running directory, for example, **D:\jdbc\_example** (Windows) or **/opt/jdbc\_example** (Linux). Place the JAR file whose name contains **-with-dependencies** in the **target** directory generated in **Step 1** to this directory. Create the **src/main/resources** subdirectory in the directory. Copy all files in the **resources** directory of the **jdbc-examples** project to the **resources** directory.

**Step 3** In Windows, run the following command:

```
cd /d d:\jdbc_example  
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```

In Linux, run the following command:

```
chmod +x /opt/jdbc_example -R  
cd /opt/jdbc_example  
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```

#### NOTE

The preceding JAR file names are for reference only. The actual names may vary.

**Step 4** In the CLI, view the HQL query results in the example codes.

The following information is displayed if the sample project is successful in Windows:

```
Create table success!  
_c0  
0  
Delete table success!
```

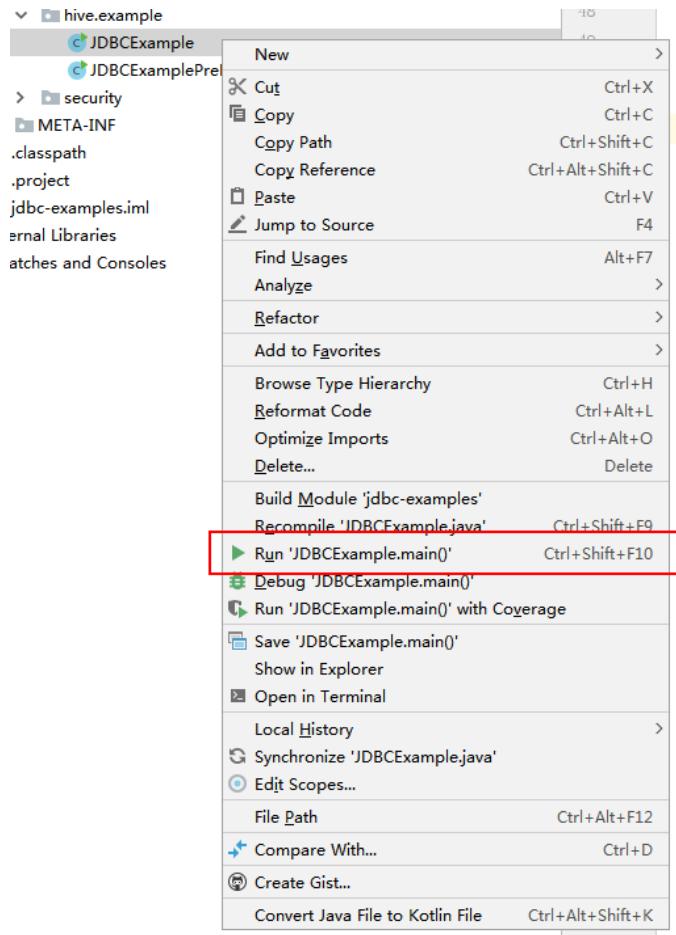
The following information is displayed if the sample project is successful in Linux:

```
Create table success!  
_c0  
0  
Delete table success!
```

----End

## Running JDBC in IntelliJ IDEA Mode

**Step 1** Right-click the JDBCExample class in the IntelliJ IDEA jdbc-examples project, and choose **Run JDBCExample.main()** from the shortcut menu. As shown in the following figure.



**Step 2** In the IntelliJ IDEA output window, view the HQL query results in the example codes.

```
Create table success!
```

```
_c0
```

```
0
```

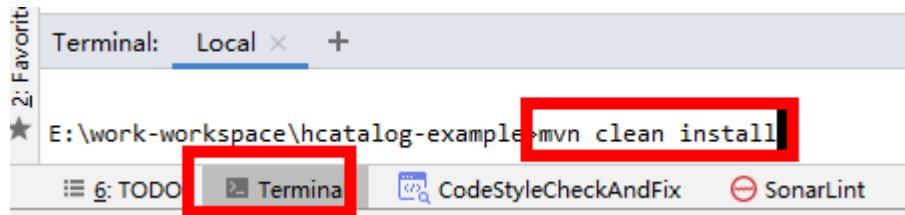
```
Delete table success!
```

----End

### 1.13.4.2 Running HCatalog and Viewing Results

#### Running HCatalog Example Projects

**Step 1** In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean install** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. The **hcatalog-example-\*.jar** package is generated in the **target** directory of the sample project.

```
Terminal: Local +  
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ hcatalog-example ---  
[INFO] Building jar: E:\other-workspase\sample_project\src\hive-examples\hcata...  
[INFO]  
[INFO] --- maven-install-plugin:2.4:install (default-install) @ hcatalog-exa...  
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcata...  
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcata...  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 4.916 s  
[INFO] Finished at: 2020-11-23T16:25:57+08:00  
[INFO] -----
```

#### NOTE

The preceding JAR file names are for reference only. The actual names may vary.

- Step 2** Upload the **hcatalog-example-\*jar** file generated in the **target** directory in the previous step to the specified directory on Linux, for example, **/opt/hive\_client**, marked as **\$HCAT\_CLIENT**, and ensure that the Hive and YARN clients have been installed. Execute environment variables for the HCAT\_CLIENT to take effect.
- ```
export HCAT_CLIENT=/opt/hive_client
```

- Step 3** Run the following command to configure environment parameters (client installation path **/opt/hadoopclient** is used as an example):

```
export HADOOP_HOME=/opt/hadoopclient/HDFS/hadoop  
export HIVE_HOME=/opt/hadoopclient/Hive/Beeline  
export HCAT_HOME=$HIVE_HOME/..../HCatalog  
export LIB_JARS=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar,$HCAT_HOME/lib/hive-metastore-xxx.jar,$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar,$HIVE_HOME/lib/hive-exec-xxx.jar,$HCAT_HOME/lib/libfb303-xxx.jar,$HCAT_HOME/lib/slf4j-api-xxx.jar,$HCAT_HOME/lib/jdo-api-xxx.jar,$HCAT_HOME/lib/antlr-runtime-xxx.jar,$HCAT_HOME/lib/datanucleus-api-jdo-xxx.jar,$HCAT_HOME/lib/datanucleus-core-xxx.jar,$HCAT_HOME/lib/datanucleus-rdbms-fi-xxx.jar,$HCAT_HOME/lib/log4j-api-xxx.jar,$HCAT_HOME/lib/log4j-core-xxx.jar,$HIVE_HOME/lib/commons-lang-xxx.jar,$HIVE_HOME/lib/hive-exec-xxx.jar  
export HADOOP_CLASSPATH=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar:$HCAT_HOME/lib/hive-metastore-xxx.jar:$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar:$HIVE_HOME/lib/hive-exec-xxx.jar:$HCAT_HOME/lib/libfb303-xxx.jar:$HADOOP_HOME/etc/hadoop:$HCAT_HOME/conf:$HCAT_HOME/lib/slf4j-api-xxx.jar:$HCAT_HOME/lib/jdo-api-xxx.jar:$HCAT_HOME/lib/antlr-runtime-xxx.jar:$HCAT_HOME/lib/datanucleus-api-jdo-xxx.jar:$HCAT_HOME/lib/datanucleus-core-xxx.jar:$HCAT_HOME/lib/datanucleus-rdbms-fi-xxx.jar:$HCAT_HOME/lib/log4j-api-xxx.jar:$HCAT_HOME/lib/log4j-core-xxx.jar:$HIVE_HOME/lib/commons-lang-xxx.jar:$HIVE_HOME/lib/hive-exec-xxx.jar
```

 NOTE

- xxx: Indicates the version number of the JAR package. **Change the version numbers of the JAR files specified in LIB\_JARS and HADOOP\_CLASSPATH based on the actual environment.**
- If the multi-instance function is enabled for Hive, perform the configuration in **export HIVE\_HOME=/opt/hadoopclient/Hive/Beeline**. For example, if Hive 1 is used, ensure that the Hive 1 client has been installed before using it. Change the value of **export HIVE\_HOME** to **/opt/hadoopclient/Hive1/Beeline**.

**Step 4** Prepare for the running:

1. Use the Hive client to create source table t1 in beeline: **create table t1(col1 int);**

Insert the following data into t1:

```
+-----+  
| t1.col1 |  
+-----+  
| 1      |  
| 1      |  
| 1      |  
| 2      |  
| 2      |  
| 3      |
```

2. Create destination table t2: **create table t2(col1 int,col2 int);**

**Step 5** Use the Yarn client to submit tasks:

```
yarn --config $HADOOP_HOME/etc/hadoop jar $HCAT_CLIENT/hcatalog-example-1.0-SNAPSHOT.jar com.huawei.bigdata.HCatalogExample -libjars $LIB_JARS t1 t2
```

**Step 6** View the running result. The data in t2 is as follows:

```
0: jdbc:hive2://192.168.1.18:24002,192.168.1.> select * from t2;  
+-----+-----+  
| t2.col1 | t2.col2 |  
+-----+-----+  
1	3
2	2
3	1
+-----+-----+
```

----End

### 1.13.4.3 Running Python and Viewing Results

#### Running Python in CLI Mode

**Step 1** Grant the execution permission for the script of **python-examples** folder. Run the following command in the CLI:

```
chmod +x python-examples -R.
```

**Step 2** Enter the service plane IP address of the node where HiveServer is installed in the hosts array of **python-examples/pyCLI\_sec.py**.

 NOTE

When the multi-instance is enabled: In **python-examples/pyCLI\_sec.py**, the hosts array must be modified. In addition, the port must be set according to the used instance. The port (**hive.server2.thrift.port**) is used for Hive to provide the Thrift service. For example, if the Hive 1 instance is used, port must be set to 21067. (The default ports of Hive to Hive 4 is 21066 to 21070, respectively).

**Step 3** Change `hadoop.hadoop.com` in the `conf` array of `python-examples/pyCLI_sec.py` and `python-examples/pyline.py` to `hadoop.the actual domain name`. To view the actual domain name, log in to FusionInsight Manager and choose **System > Permission > Domain and Mutual Trust > Local Domain**.

**Step 4** Run the `kinit` command to obtain the cache for Kerberos authentication.

Use the developer account created in [Preparing a Developer Account](#) to run the following commands to run the client program:

`kinit -kt keytabstorage path username`

`cd python-examples`

`python pyCLI_sec.py`

**Step 5** In the CLI, view the HQL query results in the example codes. For example:

```
[[{'default': ''}]]  
[{'comment': 'from deserializer', 'columnName': 'tab_name', 'type': 'STRING_TYPE'}]  
['xx']
```

 NOTE

If the following exception occurs:

```
importError: libsasl2.so.2: cannot open shared object file: No such file or directory
```

You can handle the problem as follows:

1. Run the following command to check the LibSASL version in the installed operating system.

```
ldconfig -p|grep sasl
```

If the following is displayed, the current operating system only has the 3.x version.

```
libsasl2.so.3 (libc6,x86-64) => /usr/lib64/libsasl2.so.3  
libsasl2.so.3 (libc6) => /usr/lib/libsasl2.so.3
```

2. If only the 3.x version exists, run the following command to create a soft link.

```
ln -s /usr/lib64/libsasl2.so.3.0.0 /usr/lib64/libsasl2.so.2
```

----End

#### 1.13.4.4 Running Python3 and Viewing Results

##### Running Python in CLI Mode

**Step 1** Grant the execution permission for the script of `python3-examples` folder. Run the following command in the CLI:

`chmod +x python3-examples -R.`

**Step 2** In `python3-examples/pyCLI_nosec.py`, change the value of host to the service plane IP address of the node where HiveServer is installed, and change the value of port to the port (**hive.server2.thrift.port**) used by Hive to provide the Thrift service. The default value is 21066.

 NOTE

When the multi-instance is enabled: In **python3-examples/pyCLI\_sec.py**, the hosts must be modified. In addition, the port must be set according to the used instance. The port (`hive.server2.thrift.port`) is used for Hive to provide the Thrift service. For example, if the Hive 1 instance is used, port must be set to 21067. (The default ports of Hive to Hive 4 is 21066 to 21070, respectively).

**Step 3** Change `hadoop.hadoop.com` in the conf of `python-examples/pyCLI_sec.py` to `hadoop.the actual domain name`. To view the actual domain name, log in to FusionInsight Manager and choose **System > Permission > Domain and Mutual Trust > Local Domain**.

**Step 4** Run the **kinit** command to obtain the cache for Kerberos authentication.

Use the developer account created in [Preparing a Developer Account](#) to run the following commands to run the client program:

```
kinit -kt keytabstorage path username
```

```
cd python3-examples
```

```
python3 pyCLI_sec.py
```

**Step 5** In the CLI, view the HQL query results in the example codes. For example:

```
('table_name1',)  
(‘table_name2’,)  
(‘table_name3’,)  
(‘table_name4’,)  
(‘table_name5’,)
```

In the preceding command, `table_nameX` indicates the actual table name.

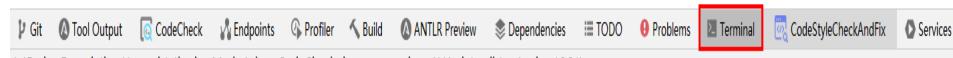
----End

#### 1.13.4.5 Running a Spring Boot Sample Project and Viewing Results

##### Running the Spring Boot Sample Project in CLI

**Step 1** Click **Terminal** in the lower left corner of the IDEA page to access the terminal. Run the **mvn clean package** command to perform compilation.

```
PS E:\workspace\sample_project\src\springboot\hive-examples\hive-rest-client-example> mvn clean package
```



If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR file containing the **-with-dependencies** field is generated in the **target** directory of the sample project.

```
Terminal: Local × +  
[INFO] com/ already added, skipping  
[INFO] com/huawei/ already added, skipping  
[INFO] com/huawei/bigdata/ already added, skipping  
[INFO] META-INF/maven/ already added, skipping  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 32.933 s  
[INFO] Finished at: 2020-11-23T16:18:08+08:00  
[INFO] -----
```

**Step 2** Create a directory on Windows or Linux as the running directory, for example, **D:\hive-rest-client-example** (Windows) or **/opt/hive-rest-client-example** (Linux). Place the JAR file in **Step 1** to this directory, and create the **src/main/resources** subdirectory in the directory. Copy all files in the **resources** directory of the **hive-rest-client-example** project to **resources**.

**Step 3** Run the following commands to start the Spring Boot service:

- Commands for Windows:  
`cd /d D:\hive-rest-client-example  
java -jar hive-rest-client-example-8.3.1-3.3.1-SNAPSHOT-jar-with-dependencies.jar`
- Commands for Linux:  
`chmod +x /opt/hive-rest-client-example -R  
cd /opt/hive-rest-client-example  
java -jar hive-rest-client-example-8.3.1-3.3.1-SNAPSHOT-jar-with-dependencies.jar`

#### BOOK NOTE

The preceding JAR file name is for reference only. Replace it with the actual one.

**Step 4** Call the Spring Boot sample API of Hive to trigger running the sample code.

- Running method in Windows:  
Open the browser and enter **http://localhost:8080/hive/example/executesql** in the address box.
- Running method in Linux:  
Run the **curl http://localhost:8080/hive/example/executesql** command on the node where the JAR file is stored in **Step 2**.

#### BOOK NOTE

The following information may be printed in the log when the sample code is executed. Although the log level is ERROR, the execution result is not affected.

```
ERROR 51320 --- [c-8-EventThread] o.a.c.framework.ims.EnsembleTracker : Invalid config event  
received: {version=100000000, server48=IP address of the ZooKeeper node:ZooKeeper port  
number.ZooKeeper port number:participant...}
```

**Step 5** View the HQL query results in the sample code.

- The following information is displayed if the sample project is successful in Windows:

```
===== Hive Example Start =====
Start create table.
Table created successfully.
Start to insert data into the table.
Inserting data to the table succeeded.
Start to query table data.
Query result :
employees_infoa.id    employees_infoa.age    employees_infoa.name
1      31     SJK
2      25     HS
3      28     HT

Querying table data succeeded.
Start to delete the table.
Table deleted successfully.
===== Hive Example End =====
```

- The following information is displayed if the sample project is successful in Linux:

```
===== Hive Example Start =====
Start create table.
Table created successfully.
Start to insert data into the table.
Inserting data to the table succeeded.
Start to query table data.
Query result :
employees_infoa.id    employees_infoa.age    employees_infoa.name
1      31     SJK
2      25     HS
3      28     HT

Querying table data succeeded.
Start to delete the table.
Table deleted successfully.
===== Hive Example End =====
```

----End

## 1.13.5 More Information

### 1.13.5.1 Interface Reference

#### 1.13.5.1.1 JDBC

The Hive Java database connectivity (JDBC) interface complies with the Java JDBC driver standard.



As a data warehouse, Hive does not support all JDBC APIs. For example, if transactional operations, such as rollback and setAutoCommit, are performed, SQL exceptions like **Method not supported** will occur.

#### 1.13.5.1.2 Hive SQL

Hive SQL supports all features in Hive-3.1.0. For details, visit <https://cwiki.apache.org/confluence/display/hive/languagemanual>.

**Table 1-127** describes the extended Hive statements provided by MRS.

**Table 1-127** Extended Hive statements

| Extended Syntax   | Syntax Description  | Syntax Example   | Example Description   |
|---|---|--|---|
| <pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format]   STORED BY 'storage.handler.cl ass.name' [WITH SERDEPROPERTIE S (...) ] ..... [TBLPROPERTIES ("groupId"= group1 ,"locatorId"="loc ator1")] ...;</pre> | <p>The statement is used to create a Hive table and specify locators on which table data files locate. For details, see "Component Operation Guide" &gt; "Using Hive" &gt; "Using HDFS Colocation to Store Hive Tables" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i>.</p> | <pre>CREATE TABLE tab1 (id INT, name STRING) row format delimited fields terminated by '\t' stored as RCFILE TBLPROPERTIES( "groupId"= group1 ,"locatorId"="loc ator1");</pre> | <p>The statement is used to create table <b>tab1</b> and specify locator1 on which the table data of <b>tab1</b> locates.</p> |

| Extended Syntax  | Syntax Description  | Syntax Example  | Example Description  |
|--|---|---|--|
| <pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format]   STORED BY 'storage.handler.cl ass.name' [WITH SERDEPROPERTIE S (...) ] ... [TBLPROPERTIES ('column.encode. columns'=col_na me1,col_name2')  'column.encode.i ndices'=col_id1,c ol_id2', 'column.encode.c lassname'=encod e_classname')]...;</pre> | <p>The statement is used to create a hive table and specify the table encryption column and encryption algorithm. For details, see "Component Operation Guide" &gt; "Using Hive" &gt; "Using the Hive Column Encryption Function" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i>.</p> | <pre>create table encode_test(id INT, name STRING, phone STRING, address STRING) ROW FORMAT SERDE 'org.apache.hadoo p.hive.serde2.lazy. LazySimpleSerDe' WITH SERDEPROPERTIE S ('column.encode.i ndices'=2,3', 'column.encode.cl assname='org.apa che.hadoop.hive.s erde2.SMS4Rewrit er') STORED AS TEXTFILE;</pre> | <p>The statement is used to create table <b>encode_test</b> and specify that column 2 and column 3 will be encrypted using the <b>org.apache.hadoop.hive.serde2.SMS4Rewriter</b> encryption algorithm class during data insertion.</p> |

| Extended Syntax  | Syntax Description  | Syntax Example   | Example Description  |
|--|---|--|--|
| <code>REMOVE TABLE<br/>hbase_tablename<br/>[WHERE<br/>where_condition];</code>   | The statement is used to delete data that meets criteria from the Hive on HBase table. For details, see "Component Operation Guide" > "Using Hive" > "Deleting Single-Row Records from Hive on HBase" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i> .  | <code>remove table<br/>hbase_table1<br/>where id = 1;</code>   | The statement is used to delete data that meets the criterion of "id = 1" from the table.  |
| <code>CREATE<br/>[TEMPORARY]<br/>[EXTERNAL]<br/>TABLE [IF NOT<br/>EXISTS]<br/>[db_name.]table_<br/>name (col_name<br/>data_type<br/>[COMMENT<br/>col_comment], ...)<br/>[ROW FORMAT<br/>row_format]<br/>STORED AS<br/>inputformat<br/>'org.apache.hadoop.hive.contrib.format.SpecifiedDelimiterInputFormat'<br/>outputformat<br/>'org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat';</code> | The statement is used to create a hive table and specify that the table supports customized row delimiters. For details, see "Component Operation Guide" > "Using Hive" > "Customizing Row Separators" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i> . | <code>create table<br/>blu(time string,<br/>num string, msg<br/>string) row<br/>format delimited<br/>fields terminated<br/>by ',' stored as<br/>inputformat<br/>'org.apache.hadoop.hive.contrib.format.SpecifiedDelimiterInputFormat'<br/>outputformat<br/>'org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat';</code> | The statement is used to create table <b>blu</b> and set <b>inputformat</b> to <b>SpecifiedDelimiterInputFormat</b> so that the query row delimiter can be specified during the query. |

### 1.13.5.1.3 WebHCat

#### NOTE

- The following uses the service IP address of WebHCat and the WebHCat HTTP port configured during the installation as an example.
- You can perform the example operations only after **kinit** authentication is implemented on the installed client.
- The examples of the HTTPS protocol are as follows. To use the HTTP protocol, you need to perform the following operations:
  1. Switch the RESTful API to the HTTP protocol. For details, see "Component Operation Guide" > "Using Hive" > "Configuring HTTPS/HTTP-based REST APIs" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.
  2. Delete **--insecure** from the example and replace HTTPS with HTTP, for example:  
`curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/status'`  
is changed to  
`curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/status'`
- Before performing the operation, ensure that the curl in use is later than 7.34.0. You can run the following command to view the curl version:  
`curl -V`

#### 1. :version(GET)

##### - Description

Queries a list of response types supported by WebHCat.

##### - URL

`https://www.myserver.com/templeton/:version`

##### - Parameter

| Parameter             | Description   |
|-----------------------|---|
| <code>:version</code> | WebHCat version number. Currently, the version number must be v1. |

##### - Returned result

| Parameter                  | Description                                  |
|----------------------------|--|
| <code>responseTypes</code> | List of response types supported by WebHCat. |

##### - Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1'
```

#### 2. status (GET)

##### - Description

Obtains the status of the current server.

- URL  
`https://www.myserver.com/templeton/v1/status`
- Parameter  
None
- Returned result

| Parameter | Description  |
|-----------|--|
| status    | If the WebHCat connection is normal, <b>OK</b> is returned.      |
| version   | Character string, including the version number, for example, v1. |

- Example  
`curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/status'`

### 3. version (GET)

- Description  
Obtains the WebHCat version of the server.
- URL  
`https://www.myserver.com/templeton/v1/version`
- Parameter  
None
- Returned result

| Parameter         | Description                    |
|-------------------|--------------------------------|
| supportedVersions | All supported versions.        |
| version           | WebHCat version of the server. |

- Example  
`curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/version'`

### 4. version/hive (GET)

- Description  
Obtains the Hive version of the server.
- URL  
`https://www.myserver.com/templeton/v1/version/hive`
- Parameter  
None
- Returned result

| Parameter | Description   |
|-----------|---------------|
| module    | Hive.         |
| version   | Hive version. |

- Example  

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/version/hive'
```

## 5. version/hadoop (GET)

- Description  

Obtains the Hadoop version of the server.
- URL  

<https://www.myserver.com/templeton/v1/version/hadoop>
- Parameter  

None
- Returned result

| Parameter | Description     |
|-----------|-----------------|
| module    | Hadoop.         |
| version   | Hadoop version. |

- Example  

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/version/hadoop'
```

## 6. ddl (POST)

- Description  

Executes a DDL statement.
- URL  

<https://www.myserver.com/templeton/v1/ddl>
- Parameter

| Parameter   | Description  |
|-------------|--|
| exec        | HCatalog DDL statement to be executed.   |
| group       | User group used when DDL is used to create a table.                                  |
| permissions | Permission used when DDL is used to create a table. The format is <b>rwxr-xr-x</b> . |

- Returned result

| Parameter | Description  |
|-----------|--|
| stdout    | Standard output value during HCatalog execution. The value may be empty. |
| stderr    | Error output during HCatalog execution. The value may be empty.          |

| Parameter | Description               |
|-----------|---------------------------|
| exitcode  | Return value of HCatalog. |

- Example

```
curl -i -u : --insecure --negotiate -d exec="show tables" 'https://10.64.35.144:21055/templeton/v1/ddl'
```

#### 7. ddl/database (GET)

- Description  
Lists all databases.
- URL  
<https://www.myserver.com/templeton/v1/ddl/database>
- Parameter

| Parameter | Description   |
|-----------|---|
| like      | Regular expression used to match the database name. |

- Returned result

| Parameter | Description    |
|-----------|----------------|
| databases | Database name. |

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/ddl/database'
```

#### 8. ddl/database/:db (GET)

- Description  
Obtains details about a specified database.
- URL  
<https://www.myserver.com/templeton/v1/ddl/database/:db>
- Parameter

| Parameter | Description    |
|-----------|----------------|
| :db       | Database name. |

- Returned result

| Parameter | Description  |
|-----------|--|
| location  | Database location.   |
| comment   | Database remarks. If there are no database remarks, the value is null. |
| database  | Database name.   |

| Parameter | Description                 |
|-----------|-----------------------------|
| owner     | Database owner.             |
| owertype  | Type of the database owner. |

- Example  

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/ddl/database/default'
```
9. **ddl/database/:db (PUT)**
- Description  

Creates a database.
  - URL  

<https://www.myserver.com/templeton/v1/ddl/database/:db>
  - Parameter
- | Parameter  | Description                                 |
|------------|---|
| :db        | Database name.                              |
| group      | User group used for creating the database.  |
| permission | Permission used for creating the database.  |
| location   | Database location.                          |
| comment    | Database remarks, for example, description. |
| properties | Database properties.                        |
- Returned result
- | Parameter | Description                         |
|-----------|-------------------------------------|
| database  | Name of the newly created database. |
- Example  

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"location": "/tmp/a", "comment": "my db", "properties": {"a": "b"}}' 'https://10.64.35.144:21055/templeton/v1/ddl/database/db2'
```
10. **ddl/database/:db (DELETE)**
- Description  

Deletes a database.
  - URL  

<https://www.myserver.com/templeton/v1/ddl/database/:db>
  - Parameter

| Parameter | Description   |
|-----------|---|
| :db       | Database name.  |
| IfExists  | If the specified database does not exist, Hive returns an error unless <b>IfExists</b> is set to <b>true</b> .  |
| option    | Set the parameter to <b>cascade</b> or <b>restrict</b> . If you set it to <b>cascade</b> , all data and definitions are cleared. If you set it to <b>restrict</b> , the table content is empty and the mode does not exist. |

- Returned result

| Parameter | Description                   |
|-----------|-------------------------------|
| database  | Name of the deleted database. |

- Example

```
curl -i -u : --insecure --negotiate -X DELETE 'https://10.64.35.144:21055/templeton/v1/ddl/database/db3?IfExists=true'
```

## 11. ddl/database/:db/table (GET)

- Description

Lists all tables in the database.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table>

- Parameter

| Parameter | Description                                    |
|-----------|--|
| :db       | Database name.                                 |
| like      | Regular expression used to match a table name. |

- Returned result

| Parameter | Description                     |
|-----------|---------------------------------|
| database  | Database name.                  |
| tables    | List of tables in the database. |

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/ddl/database/default/table'
```

## 12. ddl/database/:db/table/:table (GET)

- Description  
Obtains details about a table.
- URL  
<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>
- Parameter

| Parameter | Description  |
|-----------|--|
| :db       | Database name.   |
| :table    | Table name.  |
| format    | The format is "format=extended". If you want to see additional information, use "table extended like". |

- Returned result

| Parameter        | Description   |
|------------------|---|
| columns          | Column name and type.   |
| database         | Database name.  |
| table            | Table name.   |
| partitioned      | Whether a table is a partition table. This parameter is available only when the table format is <b>extended</b> . |
| location         | Table location. This parameter is available only when the table format is <b>extended</b> .                       |
| outputformat     | Output format. This parameter is available only when the table format is <b>extended</b> .                        |
| inputformat      | Input format. This parameter is available only when the table format is <b>extended</b> .                         |
| owner            | Table owner. This parameter is available only when the table format is <b>extended</b> .                          |
| partitionColumns | Partition column. This parameter is available only when the table format is <b>extended</b> .                     |

- Example

```
curl -i -u : --insecure --negotiate 'https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1?format=extended'
```

13. `ddl/database/:db/table/:table (PUT)`

- Description  
Creates a table.
- URL  
`https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table`
- Parameter

| Parameter     | Description   |
|---------------|---|
| :db           | Database name.  |
| :table        | New table name.   |
| group         | User group used for creating the table.   |
| permissions   | Permission used for creating the table.   |
| external      | Allows you to specify a location so that Hive does not use the default location for this table.   |
| ifNotExists   | If this parameter is set to <b>true</b> , no error is reported if a table exists.   |
| comment       | Remarks.  |
| columns       | Column description, including the column name, type, and optional remarks.  |
| partitionedBy | Partition column description, which is used to partition tables. The <b>columns</b> parameter is used to list the column name, type, and optional remarks.  |
| clusteredBy   | Bucket column description, including the <b>columnNames</b> , <b>sortedBy</b> , and <b>numberOfBuckets</b> parameters. The <b>columnNames</b> parameter includes <b>columnName</b> and sorting sequence ( <b>ASC</b> indicates an ascending order, and <b>DESC</b> indicates a descending order). |
| format        | Storage format. The parameters include <b>rowFormat</b> , <b>storedAs</b> , and <b>storedBy</b> .   |
| location      | HDFS path.  |

| Parameter       | Description   |
|-----------------|---|
| tableProperties | Table property names and values (name-value pairs). |

- Returned result

| Parameter | Description    |
|-----------|----------------|
| database  | Database name. |
| table     | Table name.    |

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"columns": [{"name": "id", "type": "int"}, {"name": "name", "type": "string"}], "comment": "hello", "format": {"storedAs": "orc"} }' 'https://10.64.35.144:21055/templeton/v1/ddl/database/db3/table/tbl1'
```

#### 14. ddl/database/:db/table/:table (POST)

- Description  
Renames a table.
- URL  
<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>
- Parameter

| Parameter | Description          |
|-----------|----------------------|
| :db       | Database name.       |
| :table    | Existing table name. |
| rename    | New table name.      |

- Returned result

| Parameter | Description     |
|-----------|-----------------|
| database  | Database name.  |
| table     | New table name. |

- Example

```
curl -i -u : --insecure --negotiate -d rename=table1 'https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/tbl1'
```

#### 15. ddl/database/:db/table/:table (DELETE)

- Description  
Deletes a table.
- URL  
<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>
- Parameter

| Parameter | Description   |
|-----------|---|
| :db       | Database name.  |
| :table    | Table name.   |
| IfExists  | If this parameter is set to <b>true</b> , no error is reported. |

- Returned result

| Parameter | Description    |
|-----------|----------------|
| database  | Database name. |
| table     | Table name.    |

- Example

```
curl -i -u : --insecure --negotiate -X DELETE 'https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/table2?IfExists=true'
```

## 16. ddl/database/:db/table/:existingtable/like/:newtable (PUT)

- Description

Creates a table that is the same as an existing table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:existingtable/like/:newtable>

- Parameter

| Parameter      | Description  |
|----------------|--|
| :db            | Database name.   |
| :existingtable | Existing table name.   |
| :newtable      | New table name.  |
| group          | User group used for creating the table.  |
| permissions    | Permission used for creating the table.  |
| external       | Allows you to specify a location so that Hive does not use the default location for this table.        |
| ifNotExists    | If this parameter is set to <b>true</b> , the Hive does not report an error if a table already exists. |
| location       | HDFS path.   |

- Returned result

| Parameter | Description    |
|-----------|----------------|
| database  | Database name. |
| table     | Table name.    |

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"ifNotExists": "true"}' 'https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/like/tt1'
```

17. `ddl/database/:db/table/:table/partition(GET)`

- Description

Lists information about all partitions of a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition>

- Parameter

| Parameter | Description    |
|-----------|----------------|
| :db       | Database name. |
| :table    | Table name.    |

- Returned result

| Parameter  | Description   |
|------------|---|
| database   | Database name.  |
| table      | Table name.   |
| partitions | List of partition attribute values and partition names. |

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition
```

18. `ddl/database/:db/table/:table/partition/:partition(GET)`

- Description

Lists information about a specific partition of a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition>

- Parameter

| Parameter | Description    |
|-----------|----------------|
| :db       | Database name. |
| :table    | Table name.    |

| Parameter  | Description  |
|------------|--|
| :partition | Partition name. Exercise caution when decoding HTTP quote, for example, <b>country=%27algeria%27</b> . |

- Returned result

| Parameter        | Description   |
|------------------|---|
| database         | Database name.  |
| table            | Table name.   |
| partition        | Partition name.   |
| partitioned      | If this parameter is set to <b>true</b> , the table is a partitioned table. |
| location         | Storage path of the table.  |
| outputFormat     | Output format.  |
| columns          | Column name, type, and remarks.   |
| owner            | Owner.  |
| partitionColumns | Partition column.   |
| inputFormat      | Input format.   |
| totalNumberFiles | Number of files in a partition.   |
| totalFileSize    | Total size of files in a partition.   |
| maxFileSize      | Maximum file size.  |
| minFileSize      | Minimum file size.  |
| lastAccessTime   | Last access time.   |
| lastUpdateTime   | Last update time.   |

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition/dt=1
```

## 19. ddl/database/:db/table/:table/partition/:partition(PUT)

- Description  
Adds a table partition.
- URL  
<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition>
- Parameter

| Parameter   | Description  |
|-------------|--|
| :db         | Database name.   |
| :table      | Table name.  |
| group       | User group used for creating a partition.  |
| permissions | User permission used for creating a partition.   |
| location    | Storage location of the new partition.   |
| ifNotExists | If this parameter is set to <b>true</b> , the system reports an error when the partition already exists. |

- Returned result

| Parameter  | Description     |
|------------|-----------------|
| database   | Database name.  |
| table      | Table name.     |
| partitions | Partition name. |

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{}' https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition/dt=10
```

## 20. ddl/database/:db/table/:table/partition/:partition(DELETE)

- Description

Deletes a table partition.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition>

- Parameter

| Parameter   | Description   |
|-------------|---|
| :db         | Database name.  |
| :table      | Table name.   |
| group       | User group used for deleting a new partition.                                       |
| permissions | User permission used for deleting a new partition. The format is <b>rwxrw-r-x</b> . |

| Parameter | Description   |
|-----------|---|
| IfExists  | If the specified partition does not exist, the Hive reports an error, unless this parameter is set to <b>true</b> . |

- Returned result

| Parameter  | Description     |
|------------|-----------------|
| database   | Database name.  |
| table      | Table name.     |
| partitions | Partition name. |

- Example

```
curl -i -u : --insecure --negotiate -X DELETE -HContent-type:application/json -d '{}' https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition/dt=10
```

## 21. `ddl/database/:db/table/:table/column(GET)`

- Description

Obtains a column list of a table.

- URL

`https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column`

- Parameter

| Parameter | Description    |
|-----------|----------------|
| :db       | Database name. |
| :table    | Table name.    |

- Returned result

| Parameter | Description           |
|-----------|-----------------------|
| database  | Database name.        |
| table     | Table name.           |
| columns   | Column name and type. |

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/column
```

## 22. `ddl/database/:db/table/:table/column/:column(GET)`

- Description

Obtains details about a column in a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column>

- Parameter

| Parameter | Description    |
|-----------|----------------|
| :db       | Database name. |
| :table    | Table name.    |
| :column   | Column name.   |

- Returned result

| Parameter | Description           |
|-----------|-----------------------|
| database  | Database name.        |
| table     | Table name.           |
| column    | Column name and type. |

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/column/id
```

### 23. `ddl/database/:db/table/:table/column/:column(PUT)`

- Description

Adds a column to a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column>

- Parameter

| Parameter | Description                               |
|-----------|---|
| :db       | Database name.                            |
| :table    | Table name.                               |
| :column   | Column name.                              |
| type      | Column type, for example, string and int. |
| comment   | Column remarks, for example, description. |

- Returned result

| Parameter | Description    |
|-----------|----------------|
| database  | Database name. |
| table     | Table name.    |
| column    | Column name.   |

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"type": "string", "comment": "new column"}' https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/column/name
```

#### 24. ddl/database/:db/table/:table/property(GET)

- Description

Obtains properties of a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property>

- Parameter

| Parameter | Description    |
|-----------|----------------|
| :db       | Database name. |
| :table    | Table name.    |

- Returned result

| Parameter  | Description    |
|------------|----------------|
| database   | Database name. |
| table      | Table name.    |
| properties | Property.      |

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/property
```

#### 25. ddl/database/:db/table/:table/property/:property(GET)

- Description

Obtains a specific property value of a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property/:property>

- Parameter

| Parameter | Description    |
|-----------|----------------|
| :db       | Database name. |

| Parameter | Description    |
|-----------|----------------|
| :table    | Table name.    |
| :property | Property name. |

- Returned result

| Parameter | Description    |
|-----------|----------------|
| database  | Database name. |
| table     | Table name.    |
| property  | Property list. |

- Example

```
curl -i -u : --insecure --negotiate https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/property/last_modified_by
```

## 26. ddl/database/:db/table/:table/property/:property(PUT)

- Description

Adds a property value to a table.

- URL

<https://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property/:property>

- Parameter

| Parameter | Description     |
|-----------|-----------------|
| :db       | Database name.  |
| :table    | Table name.     |
| :property | Property name.  |
| value     | Property value. |

- Returned result

| Parameter | Description    |
|-----------|----------------|
| database  | Database name. |
| table     | Table name.    |
| property  | Property name. |

- Example

```
curl -i -u : --insecure --negotiate -X PUT -HContent-type:application/json -d '{"value": "my value"}' https://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/property/mykey
```

## 27. mapreduce/jar(POST)

- Description

Executes a MapReduce job. Before executing a MapReduce job, upload the JAR file of the MapReduce job to HDFS.

- URL

<https://www.myserver.com/templeton/v1/mapreduce/jar>

- Parameter

| Parameter | Description   |
|-----------|---|
| jar       | JAR file of the MapReduce job to be executed.   |
| class     | Class of the MapReduce job to be executed.  |
| libjars   | JAR file names of <b>classpath</b> to be added, separated by commas (,).  |
| files     | Names of files to be copied to the MapReduce cluster, separated by commas (,).  |
| arg       | Input parameter received by the Main class.   |
| define    | This parameter is used to configure Hadoop in the <b>define=NAME=VALUE</b> format.  |
| statusdir | WebHCat writes the status of the MapReduce task to <b>statusdir</b> . If this parameter is set, you need to manually delete it.   |
| enablelog | If <b>statusdir</b> is set and <b>enablelog</b> is set to <b>true</b> , Hadoop task configurations and logs are collected to <b>\$statusdir/logs</b> . Then, successful and failed attempts are recorded in the logs. The layout of the subdirectories in <b>\$statusdir/logs</b> is as follows:<br><br>logs/\$job_id (directory for \$job_id)<br>logs/\$job_id/job.xml.html<br>logs/\$job_id/\$attempt_id (directory for \$attempt_id)<br>logs/\$job_id/\$attempt_id/stderr<br>logs/\$job_id/\$attempt_id/stdout<br>logs/\$job_id/\$attempt_id/syslog<br>Only Hadoop 1.X is supported. |

| Parameter | Description  |
|-----------|--|
| callback  | Callback address after MapReduce job execution. Use <b>\$jobId</b> to embed the job ID in the callback address. In the callback address, replace the <b>\$jobId</b> with the job ID. |

- Returned result

| Parameter | Description                                       |
|-----------|---|
| id        | Job ID, similar to <b>job_201110132141_0001</b> . |

- Example

```
curl -i -u : --insecure --negotiate -d jar="/tmp/word.count-0.0.1-SNAPSHOT.jar" -d class=com.huawei.word.count.WD -d statusdir="/output" -d enablelog=true "https://10.64.35.144:21055/templeton/v1/mapreduce/jar"
```

## 28. mapreduce/streaming(POST)

- Description

Submits a MapReduce job in Streaming mode.

- URL

<https://www.myserver.com/templeton/v1/mapreduce/streaming>

- Parameter

| Parameter | Description   |
|-----------|---|
| input     | Input path of Hadoop.   |
| output    | Output save path. If this parameter is not specified, WebHCat will store the output in a path that can be found by using queue resources. |
| mapper    | Location of the mapper program.   |
| reducer   | Location of the reducer program.  |
| files     | Add HDFS files to the distributed cache.  |
| arg       | Set an argument.  |
| define    | Set Hadoop configuration variables in the <b>define=NAME=VALUE</b> format.  |
| cmdenv    | Set environment variables in the <b>cmdenv=NAME=VALUE</b> format.   |

| Parameter | Description   |
|-----------|---|
| statusdir | WebHCat writes the status of the MapReduce task to <b>statusdir</b> . If this parameter is set, you need to manually delete it.   |
| enablelog | If <b>statusdir</b> is set and <b>enablelog</b> is set to <b>true</b> , Hadoop task configurations and logs are collected to <b>\$statusdir/logs</b> . Then, successful and failed attempts are recorded in the logs. The layout of the subdirectories in <b>\$statusdir/logs</b> is as follows:<br><br>logs/\$job_id (directory for \$job_id)<br>logs/\$job_id/job.xml.html<br>logs/\$job_id/\$attempt_id (directory for \$attempt_id)<br>logs/\$job_id/\$attempt_id/stderr<br>logs/\$job_id/\$attempt_id/stdout<br>logs/\$job_id/\$attempt_id/syslog<br>Only Hadoop 1.X is supported. |
| callback  | Callback address after MapReduce job execution. Use <b>\$jobId</b> to embed the job ID in the callback address. In the callback address, replace the <b>\$jobId</b> with the job ID.  |

- Returned result

| Parameter | Description                                       |
|-----------|---|
| id        | Job ID, similar to <b>job_201110132141_0001</b> . |

- Example

```
curl -i -u : --insecure --negotiate -d input=/input -d output=/oooo -d mapper=/bin/cat -d reducer="/usr/bin/wc -w" -d statusdir="/output" 'https://10.64.35.144:21055/templeton/v1/mapreduce/streaming'
```

#### NOTE

Before using this API, ensure that the prerequisites are met. For details, see [Rules](#).

### 29. /hive(POST)

- Description  
Runs Hive commands.
- URL  
<https://www.myserver.com/templeton/v1/hive>

## - Parameter

| Parameter | Description  |
|-----------|--|
| execute   | Hive commands, including entire and short Hive commands.   |
| file      | HDFS file containing Hive commands.  |
| files     | Names of files to be copied to the MapReduce cluster, separated by commas (,).   |
| arg       | Set an argument.   |
| define    | Hive configuration. The format is <b>define=key=value</b> . When using the post statement, you need to configure the scratch dir of the instance. The WebHCat instance uses <b>define=hive.exec.scratchdir=/tmp/hive-scratch</b> , and the WebHCat1 instance uses <b>define=hive.exec.scratchdir=/tmp/hive1-scratch</b> . The same rule applies to other instances.  |
| statusdir | WebHCat writes the status of the MapReduce task to <b>statusdir</b> . If this parameter is set, you need to manually delete it.  |
| enablelog | If <b>statusdir</b> is set and <b>enablelog</b> is set to <b>true</b> , Hadoop task configurations and logs are collected to <b>\$statusdir/logs</b> . Then, successful and failed attempts are recorded in the logs. The layout of the subdirectories in <b>\$statusdir/logs</b> is as follows:<br><b>logs/\$job_id</b> (directory for \$job_id)<br><b>logs/\$job_id/job.xml.html</b><br><b>logs/\$job_id/\$attempt_id</b> (directory for \$attempt_id)<br><b>logs/\$job_id/\$attempt_id/stderr</b><br><b>logs/\$job_id/\$attempt_id/stdout</b><br><b>logs/\$job_id/\$attempt_id/syslog</b> |

| Parameter | Description  |
|-----------|--|
| callback  | Callback address after MapReduce job execution. Use <b>\$jobId</b> to embed the job ID in the callback address. In the callback address, replace the <b>\$jobId</b> with the job ID. |

- Returned result

| Parameter | Description                                       |
|-----------|---|
| id        | Job ID, similar to <b>job_201110132141_0001</b> . |

- Example

```
curl -i -u : --insecure --negotiate -d execute="select count(*) from t1" -d
define=hive.exec.scratchdir=/tmp/hive-scratch -d statusdir="/output" "https://10.64.35.144:21055/
templeton/v1/hive"
```

### 30. jobs(GET)

- Description  
Obtains all job IDs.
- URL  
<https://www.myserver.com/templeton/v1/jobs>
- Parameter

| Parameter | Description  |
|-----------|--|
| fields    | If this parameter is set to *, details about each job are returned. If this parameter is not set, only a job ID is returned. The parameter can only be set to *. If the parameter is set to another value, an exception occurs.  |
| jobid     | If <b>jobid</b> is set, only jobs whose lexicographic order is greater than <b>jobid</b> are returned. For example, if the value of <b>jobid</b> is <b>job_201312091733_0001</b> , only the job whose value is greater than the value can be returned. The number of returned jobs depends on the value of <b>numrecords</b> . |

| Parameter  | Description  |
|------------|--|
| numrecords | If <b>numrecords</b> and <b>jobid</b> are set, the <b>jobid</b> list is sorted lexicographically. After <b>jobid</b> is returned, the maximum value of <b>numrecords</b> can be obtained. If <b>jobid</b> is not set but <b>numrecords</b> is set, the maximum value of <b>numrecords</b> can be obtained after the <b>jobid</b> list is sorted lexicographically. In contrast, if <b>numrecords</b> is not set but <b>jobid</b> is set, all jobs whose lexicographic orders are greater than <b>jobid</b> will be returned. |
| showall    | If <b>showall</b> is set to <b>true</b> , the request will return all jobs the user has permission to view, not only the jobs belonging to the user.   |

- Returned result

| Parameter | Description  |
|-----------|--|
| id        | Job id   |
| detail    | If the value of <b>showall</b> is <b>true</b> , details are displayed. Otherwise, the value is null. |

- Example

```
curl -i -u : --insecure --negotiate "https://10.64.35.144:21055/templeton/v1/jobs"
```

### 31. jobs/:jobid(GET)

- Description

Obtains information about a specified job.

- URL

<https://www.myserver.com/templeton/v1/jobs/:jobid>

- Parameter

| Parameter | Description                              |
|-----------|--|
| jobid     | Job ID, received after a job is created. |

- Returned result

| Parameter       | Description  |
|-----------------|--|
| status          | JSON object containing job status information.   |
| profile         | JSON object containing job information. WebHCat parses information in the <b>JobProfile</b> object. The object varies according to the Hadoop version. |
| id              | Job ID.  |
| percentComplete | Job completion percentage, for example, 75%. If the job is complete, the value is null.  |
| user            | User who created the job.  |
| callback        | Callback URL (if any).   |
| userargs        | Parameter <b>argument</b> and parameter value when a user submits a job.   |
| exitValue       | Exit value of the job.   |

- Example

```
curl -i -u : --insecure --negotiate "https://10.64.35.144:21055/templeton/v1/jobs/  
job_1440386556001_0255"
```

### 32. jobs/:jobid(DELETE)

- Description

Kills a job.

- URL

<https://www.myserver.com/templeton/v1/jobs/:jobid>

- Parameter

| Parameter | Description                  |
|-----------|------------------------------|
| :jobid    | ID of the job to be deleted. |

- Returned result

| Parameter | Description                                    |
|-----------|--|
| user      | User who submits a job.                        |
| status    | JSON object containing job status information. |

| Parameter | Description  |
|-----------|--|
| profile   | JSON object containing job information. WebHCat parses information in the <b>JobProfile</b> object. The object varies according to the Hadoop version. |
| id        | Job ID.  |
| callback  | Callback URL (if any).   |

- Example

```
curl -i -u : --insecure --negotiate -X DELETE "https://10.64.35.143:21055/templeton/v1/jobs/job_1440386556001_0265"
```

## 1.13.5.2 FAQ

### 1.13.5.2.1 A Message Is Displayed Stating "Unable to read HiveServer2 configs from ZooKeeper" During the Use of the Secondary Development Program

#### Question

A Message Is Displayed Stating "Unable to read HiveServer2 configs from ZooKeeper" During the Use of the Secondary Development Program.

#### Answer

- Possible Causes
  - The used **krb5.conf** and **user.keytab** may not be the latest ones, or the user names in the files do not match with those in example codes.
  - The difference between the time of the client and that of the connected cluster is greater than 5 minutes.
- Possible Causes
  - Check the code and download the latest credential file for user authentication.
  - Check whether the difference between the time of the client and that of the connected cluster is greater than 5 minutes. If yes, adjust the time of the client.

### 1.13.5.2.2 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions

#### Question

IBM JDK is abnormal, and the **Problem performing GSS wrap** message is displayed.

#### Answer

##### Possible Causes

The Hive connection duration exceeds the user authentication timeout duration (one day by default), causing authentication failure.

 **NOTE**

The IBM JDK mechanism is different from the Oracle JDK mechanism. IBM JDK executes time check after authentication and login, but does not check external time update. This results in refreshing failure even Hive relogin is invoked explicitly.

**Solution**

When one Hive connection fails, disable this connection, and create a connection to continue performing previous operations.

### 1.13.5.2.3 Hive SQL Is Incompatible with SQL2003 Standards

This document describes the incompatibility issues between Hive SQL and SQL2003.

- **The view cannot be written in having.**

For example:

```
select c_last_name
      ,c_first_name
      ,s_store_name
      ,sum(netpaid) paid
  from ssales
 where i_color = 'chiffon'
 group by c_last_name
      ,c_first_name
      ,s_store_name
 having sum(netpaid) > (select 0.05*avg(netpaid) from ssales);
```

**Error message:**

Error: Error while compiling statement: FAILED:ParseException line 46:23 cannot recognize input near 'select' '0.05' '\*' in expression specification (state=42000,code=40000)

- **The Having does not support sub-query.**

For example:

```
select
      ps_partkey,
      sum(ps_supplycost * ps_availqty) as value
  from
      partsupp,
      supplier,
      nation
 where
      ps_suppkey = s_suppkey
      and s_nationkey = n_nationkey
      and n_name = 'SAUDI ARABIA'
 group by
      ps_partkey having
      sum(ps_supplycost * ps_availqty) > (
          select
              sum(ps_supplycost * ps_availqty) * 0.000100000
          from
              partsupp,
              supplier,
              nation
          where
              ps_suppkey = s_suppkey
              and s_nationkey = n_nationkey
              and n_name = 'SAUDI ARABIA'
      )
 order by
      value desc;
```

### Error message:

Error: Error while compiling statement: FAILED: SemanticException Line 0:-1 Unsupported SubQuery Expression "SAUDI ARABIA": Only SubQuery expressions that are top level conjuncts are allowed (state=42000,code=40000)

- **Multiple query results cannot be displayed as multiple fields.**

For example:

```
select
    c_count,
    count(*) as custdist
from
(
    select
        c_custkey,
        count(o_orderkey)
    from
        customer left outer join orders on
            c_custkey = o_custkey
            and o_comment not like '%pending%requests%'
    group by
        c_custkey
) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;
```

### Error message:

Error: Error while compiling statement: FAILED: ParseException line 1:213 missing EOF at '(' near 'c\_orders' (state=42000,code=40000)

- **The query results cannot be compared as fields.**

For example:

```
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#25'
    and p_container = 'MED JAR'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );
```

### Error message:

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 14:4 Unsupported SubQuery Expression 'ps\_suppkey': Correlating expression contains ambiguous column references. (state=42000,code=10249)

- **The multi-table association query does not support the sub-query filter by not in or in.**

For example:

```
select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
```

```
partsupp,  
part  
where  
p_partkey = ps_partkey  
and p_brand <> 'Brand#12'  
and p_type not like 'PROMO PLATED%'  
and p_size in (25, 2, 43, 9, 35, 36, 48, 24)  
and ps_suppkey in (  
select  
s_suppkey as ps_suppkey  
from  
supplier  
where  
s_comment like '%Customer%Complaints%'  
)  
group by  
p_brand,  
p_type,  
p_size  
order by  
supplier_cnt desc,  
p_brand,  
p_type,  
p_size;
```

**Error message:**

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 14:4  
Unsupported SubQuery Expression 'ps\_suppkey': Correlating expression contains ambiguous column  
references. (state=42000,code=10249)

- **The multi-table association does not support filtering of not in and in sub-queries.**

For example:

```
select  
c_name,  
c_custkey,  
o_orderkey,  
o_orderdate,  
o_totalprice,  
sum(l_quantity)  
from  
customer,  
orders,  
lineitem  
where  
o_orderkey in (  
select  
l_orderkey  
from  
lineitem  
group by  
l_orderkey having  
sum(l_quantity) > 315  
)  
and c_custkey = o_custkey  
and o_orderkey = l_orderkey  
group by  
c_name,  
c_custkey,  
o_orderkey,  
o_orderdate,  
o_totalprice  
order by  
o_totalprice desc,  
o_orderdate  
limit 100;
```

**Error message:**

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 13:0  
Unsupported SubQuery Expression 'o\_orderkey': Correlating expression contains ambiguous column references. (state=42000,code=10249)

- **The association conditions do not support multiple exists query entities.**

For example:

```
select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
        select
            *
        from
            lineitem l2
        where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey <> l1.l_suppkey
    )
    and not exists (
        select
            *
        from
            lineitem l3
        where
            l3.l_orderkey = l1.l_orderkey
            and l3.l_suppkey <> l1.l_suppkey
            and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and n_name = 'VIETNAM'
group by
    s_name
order by
    numwait desc,
    s_name
limit 100;
```

**Error message:**

Error: Error while compiling statement: FAILED: SemanticException [Error 10249]: Line 23:8  
Unsupported SubQuery Expression 'l\_commitdate': Only 1 SubQuery expression is supported.  
(state=42000,code=10249)

- **The multi-level in-nested sub-query is not supported.**

For example:

```
select i_item_id item_id,
       sum(sr_return_quantity) sr_item_qty
  from store_returns,
       item,
       date_dim
 where sr_item_sk = i_item_sk
   and d_date in
       (select d_date
          from date_dim
         where d_week_seq in
              (select d_week_seq
                 from date_dim
                where d_date in ('1998-01-02','1998-10-15','1998-11-10'))
   and sr_returned_date_sk = d_date_sk
```

```
group by i_item_id),
cr_items as
(select i_item_id item_id,
       sum(cr_return_quantity) cr_item_qty
  from catalog_returns,
       item,
       date_dim
 where cr_item_sk = i_item_sk);
```

**Error message:**

Unsupported SubQuery Expression 'd\_week\_seq': SubQuery cannot use the table alias: date\_dim; this is also an alias in the Outer Query and SubQuery contains a unqualified column reference  
(state=42000,code=10249)

### 1.13.5.2.4 "version 'GLIBCXX\_3.4.21' not found" Exception Occurs When the Python 3 Secondary Development Program Is Used to Access Hive

#### Question

"version 'GLIBCXX\_3.4.21' not found" exception occurs when the Python 3 secondary development program is used to access Hive.

#### Answer

- **Possible Causes**

The default GCC version of the OS used by the client is too early, and the dynamic link library (DLL) does not contain **GLIBCXX\_3.4.21**.

- **Solution**

Upgrade the GCC version of the client OS to 5.4.0.

## 1.14 IoTDB Development Guide

### 1.14.1 Overview

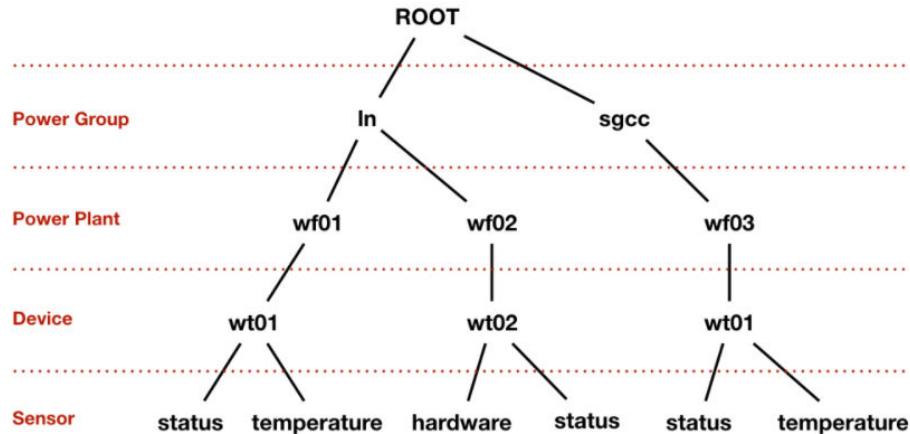
#### 1.14.1.1 Application Development Overview

##### Introduction to IoTDB

IoTDB is a data management engine that integrates collection, storage, and analysis of time series data. It features lightweight, high performance, and ease of use. It perfectly interconnects with the Hadoop and Spark ecosystems and meets the requirements of high-speed write and complex analysis and query on massive time series data in industrial IoT applications.

##### 1.14.1.2 Basic Concepts

The following uses an electric power scenario as an example to describe how to create a correct data model in IoTDB.

**Figure 1-170** Hierarchical structure of attributes in an electric power scenario

**Figure 1-170** shows the power group layer, power plant layer, device layer, and sensor layer. **ROOT** is a root node, and each node at the sensor layer is a leaf node. According to the IoTDB syntax, the path from **ROOT** to a leaf node is separated by a dot (.). The complete path is used to name a time series in the IoTDB. For example, the time series name corresponding to the path on the left in **Figure 1-170** is **ROOT.In.wf01.wt01.status**.

## Basic Concepts

- **Device**  
A device is a machine equipped with sensors in actual scenarios. In IoTDB, all sensors must have their corresponding devices.
- **Sensor**  
A sensor is a detection machine in actual scenarios. It can sense the information to be measured, and can transform the sensed information into an electrical signal or other desired form of information output and send it to IoTDB. In IoTDB, all data and paths stored are organized in units of sensors.
- **Database**  
A database is also called a storage group. You can set any prefixed path as a database. If there are four time series, for example, **root.vehicle.d1.s1**, **root.vehicle.d1.s2**, **root.vehicle.d2.s1**, and **root.vehicle.d2.s2**, two devices **d1** and **d2** in the path **root.vehicle** may belong to the same owner or manufacturer, so **d1** and **d2** are closely related. In this case, the prefixed path **root.vehicle** can be designated as a database, which will enable IoTDB to store the data of all devices under it in the same folder. Newly added devices in **root.vehicle** will also belong to this storage group.

### NOTE

- A proper number of databases can improve performance. There is neither the slowdown of the system due to frequent I/O switching (which will also take up excessive memory and result in frequent memory-file switching) caused by too many storage files or folders, nor the block of write commands caused by too few storage files or folders (which reduces concurrency).
- You need to balance the storage group settings of storage files according to their own data size and usage scenarios to achieve better system performance.

- **Time series**

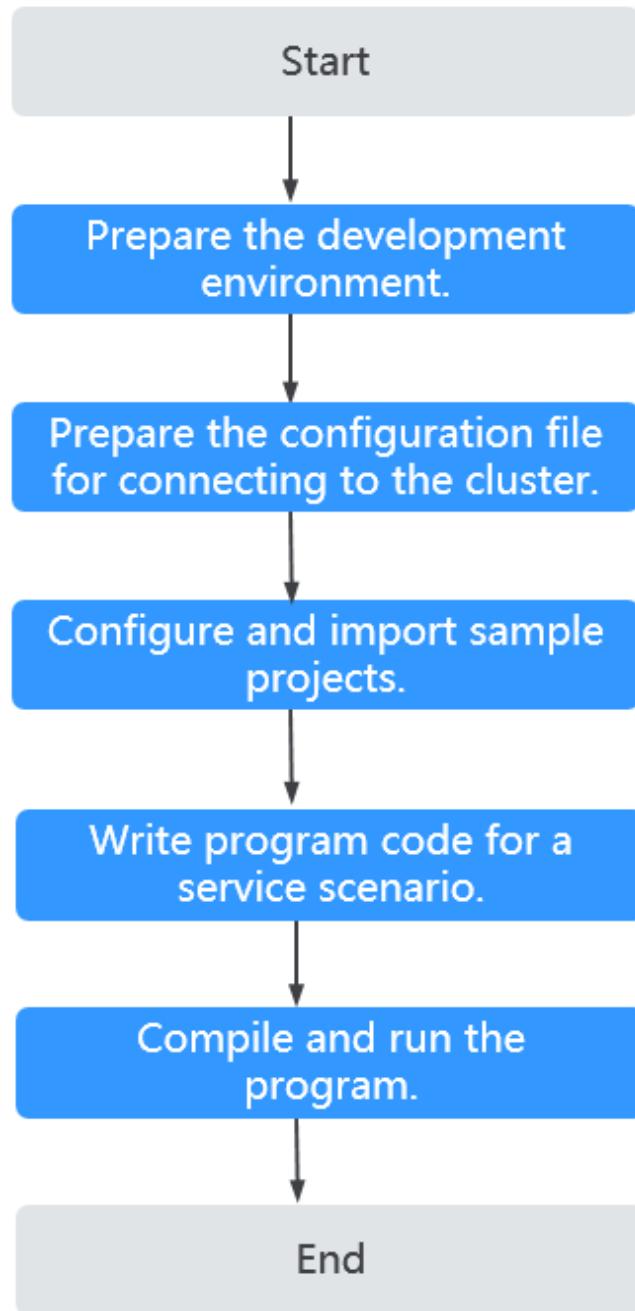
Time series is a core concept in IoTDB. The time series can be considered as the complete path of the sensor that generates the time series data. In IoTDB, all time series must start with root and end with the sensor.

### 1.14.1.3 Development Process

This section describes how to use Java APIs to develop IoTDB applications.

[Figure 1-171](#) and [Table 1-128](#) describe the phases in the development process.

[Figure 1-171](#) IoTDB application development process



**Table 1-128** Description of the IoTDB application development process

| Phase   | Description   | Reference   |
|---|---|---|
| Prepare the development environment.                          | Before developing an application, you need to prepare the development environment. You are advised to use the Java language and IntelliJ IDEA tool for development. In addition, you need to complete the initial configuration of the JDK and Maven.   | <a href="#">Preparing the Environment</a>                                       |
| Prepare the Configuration File for Connecting to the Cluster. | During application development or running, you need to connect to the MRS cluster through cluster configuration files. Configuration files include cluster component information files and user files used for security authentication. You can obtain related content from the created MRS cluster.<br><br>Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured. | <a href="#">Preparing the Configuration Files for Connecting to the Cluster</a> |
| Configure and import the sample project.                      | IoTDB provides multiple sample programs in different scenarios. You can obtain sample projects and import them to the local development environment for program learning.   | <a href="#">Configuring and Importing a Sample Project</a>                      |
| Develop programs based on business scenarios.                 | Sample projects of the Java language are provided, including JDBC and Session connection modes. A sample project covers the entire process from creating a storage group, creating a time sequence, inserting data, to deleting a storage group.  | <a href="#">Application Development</a>   |
| Compile and run the application.                              | Compile the developed application and submit it for running.  | <a href="#">Application Commissioning</a>                                       |

#### 1.14.1.4 IoTDB Sample Project

To obtain an MRS sample project, visit <https://github.com/huaweicloud/huaweicloud-mrs-example> and switch to the branch that matches the MRS cluster version. Download the package to the local PC and decompress it to obtain the sample project of each component.

MRS provides the following IoTDB sample projects.

**Table 1-129** IoTDB sample projects

| Sample Project Location              | Description  |
|--------------------------------------|--|
| iotdb-examples/iotdb-flink-example   | Program for using Flink to access IoTDB data, including FlinkIoTDBSink and FlinkIoTDBSource data.<br>FlinkIoTDBSink can use Flink jobs to write time series data to IoTDB. FlinkIoTDBSource reads time series data from IoTDB through Flink jobs and prints the data. For details, see <a href="#">IoTDB Flink</a> . |
| iotdb-examples/iotdb-jdbc-example    | Java sample program for IoTDB JDBC to process data<br>This example demonstrates how to use JDBC APIs to connect to IoTDB and run IoTDB SQL statements. For details, see <a href="#">IoTDB JDBC</a> .   |
| iotdb-examples/iotdb-kafka-example   | Sample program for accessing IoTDB data through Kafka<br>This program demonstrates how to send time series data to Kafka and then use multiple threads to write the data to IoTDB. For details, see <a href="#">IoTDB Kafka</a> .  |
| iotdb-examples/iotdb-session-example | Java sample program for IoTDB Session to process data<br>This example demonstrates how to use Session to connect to IoTDB and run IoTDB SQL statements. For details, see <a href="#">IoTDB Session</a> .   |
| iotdb-examples/iotdb-udf-exmaple     | This sample program describes how to implement a simple IoTDB user-defined function (UDF). For details, see <a href="#">IoTDB UDF Program</a> .  |

## 1.14.2 Environment Preparations

### 1.14.2.1 Preparing the Environment

[Table 1-130](#) describes the environment required for application development.

**Table 1-130** Development environment

| Item | Description  |
|------|--|
| OS   | <ul style="list-style-type: none"><li>Development environment: Windows 7 or later versions</li><li>Operating environment: Windows or Linux<br/>If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.</li></ul> |

| Item          | Description   |
|---------------|---|
| JDK           | <p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>● For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>● For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● For security purposes, the server supports only TLS V1.2 or later.</li><li>● By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetin-stancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetin-stancetls-oracle#matchsslcontext_tls</a>.</li><li>● For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul> |
| IntelliJ IDEA | <p>Tool used for developing IoTDB applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li><li>● If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li><li>● If you use OpenJDK, ensure that the JDK configured in IntelliJ IDEA is OpenJDK.</li><li>● Do not use the same workspace or the sample project in the same path for different IntelliJ IDEA programs.</li></ul>   |
| Maven         | <p>Basic configurations of the development environment. Maven is used for project management throughout the lifecycle of software development.</p> <p>Huawei provides an open-source mirror site, Huawei Mirrors. You can download the dependent JAR packages of the sample projects from this site. You can download the rest open-source JAR packages from the Maven central repository or other user-defined repositories. For details, see <a href="#">Configuring Huawei Open-Source Mirrors</a>.</p>  |

| Item  | Description   |
|-------|---|
| 7-zip | This tool is used to decompress *.zip and *.rar files. <b>7-zip 16.04</b> is supported. |

### 1.14.2.2 Preparing the Configuration Files for Connecting to the Cluster

#### Preparing for User Authentication

For an MRS cluster with Kerberos authentication enabled, you need to prepare a user who has the operation permission on related components for program authentication.

The following IoTDB permission configuration example is for reference only. You can modify the configuration as you need.

**Step 1** Log in to FusionInsight Manager.

**Step 2** Choose **System > Permission > Role**. On the displayed page, click **Create Role**.

1. Enter the role name, for example, **developrole**.
2. In the **Configure Resource Permission** table, choose *Name of the desired cluster > IoTDB > Common User Permission*, and select **Set Database** for the **root** directory.
3. Click **root**, select the desired storage group, select **Create, Modify, Write, Read, and Delete**, and click **OK**.

**Step 3** Choose **User** in the navigation pane and click **Create** on the displayed page. Create a machine-machine user, for example, **developuser**.

- Add the **iotdbgroup** user group to **User Group**.
- Add the new role created in **Step 2** to **Role**.

**Step 4** Log in to FusionInsight Manager as user **admin** and choose **System > Permission > User**. In the **Operation** column of **developuser**, choose **More > Download Authentication Credential**. Save the file and decompress it to obtain the **user.keytab** and **krb5.conf** files of the user.

----End

#### Generating an SSL Certificate for the IoTDB Client

If SSL encrypted transmission is enabled for the cluster and this is your first time running IoTDB sample code in the local Windows or Linux environment, perform the following operations to generate a client SSL certificate:

**Step 1** Log in to the node where the client is installed as the client installation user.

**Step 2** Switch to the IoTDB client installation directory, for example, **/opt/hadoopclient**.

```
cd /opt/hadoopclient
```

**Step 3** Configure environment variables.

```
source bigdata_env
```

**Step 4** Generate the client SSL certificate **truststore.jks**.

```
keytool -noprompt -import -alias myservercert -file ca.crt -keystore  
truststore.jks
```

You are required to set a password.

**Step 5** To run the sample code in the Linux environment, copy the generated **truststore.jks** file to the *Client installation directory/iotDB/iotdb/conf* directory.

```
cp truststore.jks Client installation directory/iotDB/iotdb/conf
```

----End

## Preparing the Configuration Files of the Running Environment

During the development or a test run of the program, you need to use cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file and user files used for security authentication. You can obtain the required information from the created MRS cluster.

Nodes used for program debugging or running must be able to communicate with nodes within the MRS cluster, and the hosts domain name must be configured.

- Scenario 1: Preparing the configuration files required for debugging in the local Windows development environment
  - a. Log in to FusionInsight Manager. In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Configuration Files Only**, and click **OK**. After the client file package is generated, download it to the local PC as prompted and decompress it.  
For example, if the client configuration file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Decompress the package to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC.
  - b. Copy all items from the **hosts** file in the decompression directory to the **hosts** file on the node where the client is installed. Ensure that the network communication between the local PC and hosts listed in the **hosts** file is normal.

### NOTE

- If the client host is outside the cluster, configure network connections to the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- Scenario 2: Preparing the configuration files required for running the program in a Linux environment
  - a. Install the client. For example, install the client in the **/opt/hadoopclient** directory.

- Ensure that the difference between the client time and the cluster time is less than 5 minutes.
- b. Log in to FusionInsight Manager, click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.
  - c. Log in to the active OMS node as user **root**, go to the directory where the client configuration files are stored (**/tmp/FusionInsight-Client** by default), decompress the software package, and obtain all configuration files in the **IoTDB/config** directory. Place the files in the **conf** directory where the compiled JAR package is stored on the client node, for example, **/opt/hadoopclient/conf**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and it is downloaded to the **/tmp/FusionInsight-Client** directory on the active management node, run the following commands:  

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp IoTDB/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```
  - d. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

### 1.14.2.3 Configuring and Importing a Sample Project

#### Scenario

Obtain the IoTDB development sample project and import the project to IntelliJ IDEA to learn the sample project.

#### Procedure

- Step 1** Obtain the sample project from the **src/iotdb-examples** directory in the directory where the sample code is decompressed by referring to [Obtaining Sample Projects from Huawei Mirrors](#). You can select a sample based on the actual service scenario. For details about the samples, see [IoTDB Sample Project](#).
- Step 2** To debug **iotdb-flink-example**, **iotdb-jdbc-example**, **iotdb-kafka-example**, or **iotdb-session-example** sample code on the local Windows environment, perform the following operations:

- Store the authentication files **user.keytab** and **krb5.conf** obtained in [Preparing for User Authentication](#) and SSL certificate file **truststore.jks** to the ..\src\main\resources directory of each sample project.
- Configure the **com.huawei.bigdata.iotdb.IoTDBProperties** class in the ..\src\main\resources directory of each sample project, change the value of **proPath** in the **IoTDBProperties()** method to the absolute path of the **iotdb-example.properties** file.

**Figure 1-172** Configuring the proPath parameter

```
private IoTDBProperties() {
    // If Windows environment, proPath is "iotdb-example.properties" absolute file path.
    // eg:D:\\sample_project\\sample_project\\src\\iotdb-examples\\iotdb-session-example\\src\\main\\resources\\iotdb-example.properties
    String proPath = "D:\\code\\CodeHub\\sample_project\\sample_project\\src\\iotdb-examples\\iotdb-session-example\\src\\main\\resources\\iotdb-example.properties";
    // String proPath = System.getProperty("user.dir") + File.separator + "src" + File.separator + "main"
    //           + File.separator + "resources" + File.separator + "iotdb-example.properties";
    try {
        iotdbProps.load(new FileInputStream(new File(proPath)));
    } catch (IOException e) {
        LOG.info(s:"The Exception occurred.", e);
    }
}
```

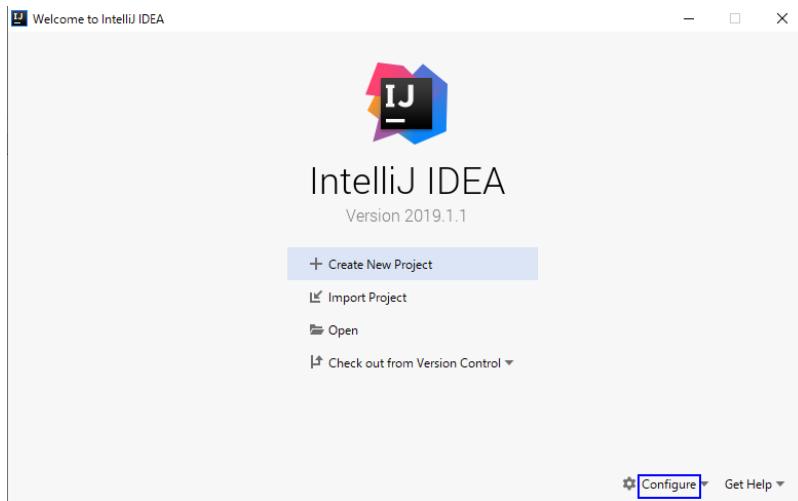
**Step 3** Modify the **iotdb-example.properties** file in the ..\src\main\resources directory of each sample project.

```
iotdb_ssl_enable=true
jdbc_url=jdbc:iotdb://IP address of the IoTDBServer instance:IoTDBServer RPC port
# Username for authentication
username=developuser
# User password for authentication. It is recommended that the password be stored in ciphertext and decrypted when being used.
password=xxx
# Location of the krb5.conf file in the downloaded authentication credential. In Linux, you are advised to use the krb5.conf file that has been uploaded to the IoTDB client installation directory/IoTDB/iotdb/conf directory. In Windows, use \\ to specify the absolute path of the file.
krb5_conf_dest=/opt/hadoopclient/IoTDB/iotdb/conf/krb5.conf
# Location of the user.keytab file in the downloaded authentication credential. In Linux, you are advised to use the user.keytab file that has been uploaded to the IoTDB client installation directory/IoTDB/iotdb/conf directory. In Windows, use \\ to specify the absolute path of the file.
key_tab_dest=/opt/hadoopclient/IoTDB/iotdb/conf/user.keytab
# Username corresponding to user.keytab followed by @HADOOP.COM
client_principal=developuser@HADOOP.COM
# Location of the truststore file. In Linux, you are advised to use the truststore.jks file that has been uploaded to the IoTDB client installation directory/IoTDB/iotdb/conf directory. In Windows, use \\ to specify the absolute path of the file.
iotdb_ssl_truststore=/opt/hadoopclient/IoTDB/iotdb/conf/truststore.jks
```

**Step 4** After the IntelliJ IDEA and JDK tools are installed, configure the JDK in IntelliJ IDEA.

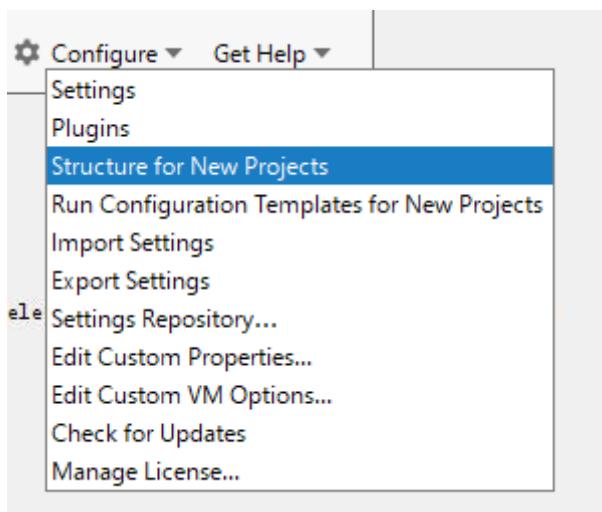
1. Start IntelliJ IDEA and choose **Configure**.

Figure 1-173 Quick Start



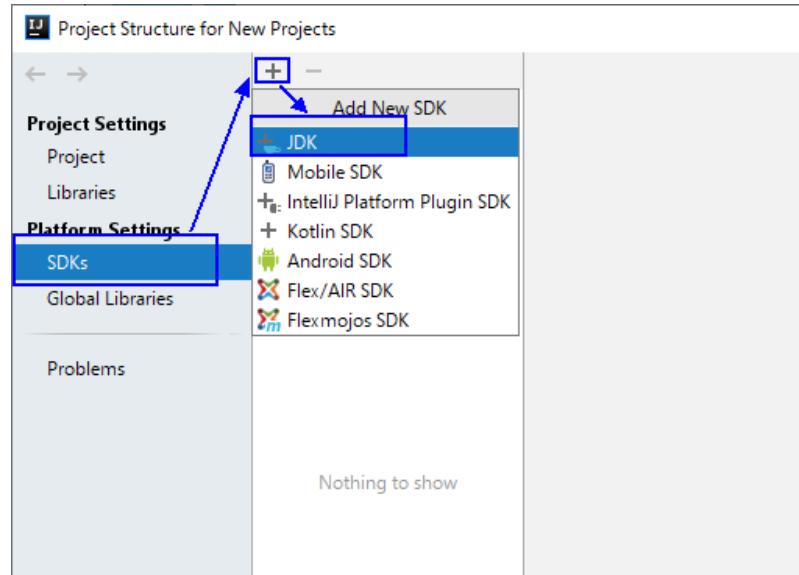
2. Select **Structure for New Projects** from the drop-down list.

Figure 1-174 Configure



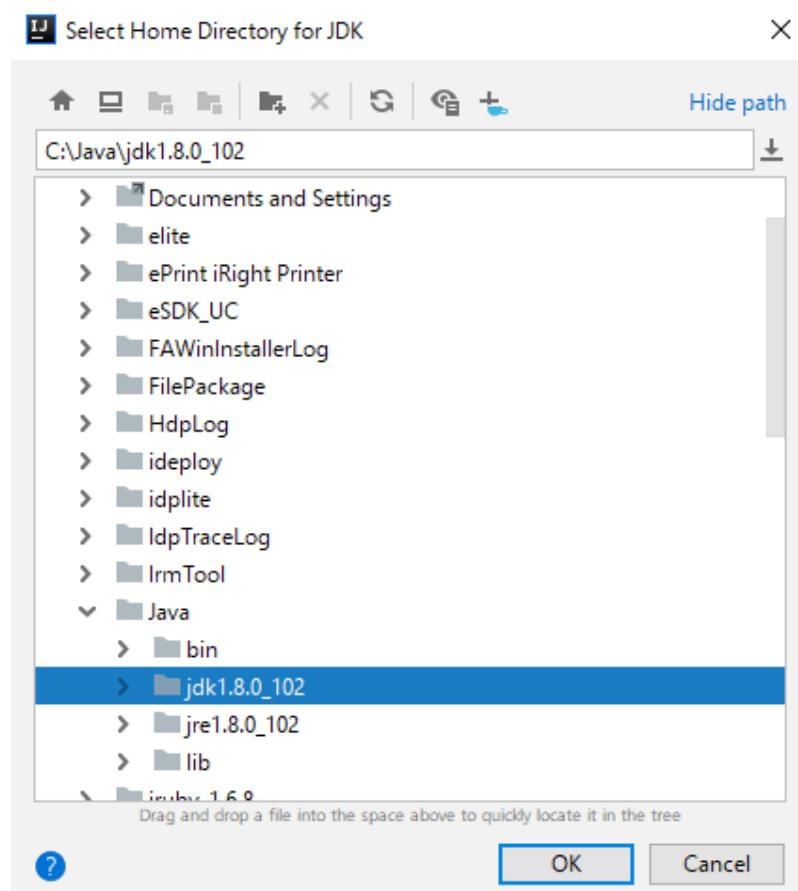
3. On the displayed **Project Structure for New Projects** page, select **SDKs**, click the plus sign (+), and click **JDK**.

Figure 1-175 Project Structure for New Projects



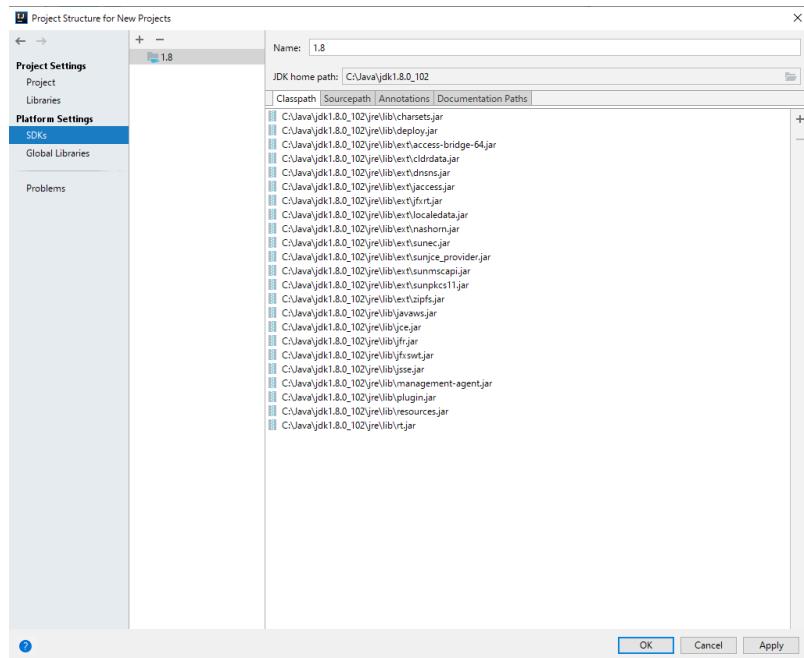
4. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 1-176 Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.

Figure 1-177 Completing the JDK configuration



**NOTE**

The operation procedure may vary according to the IDEA version.

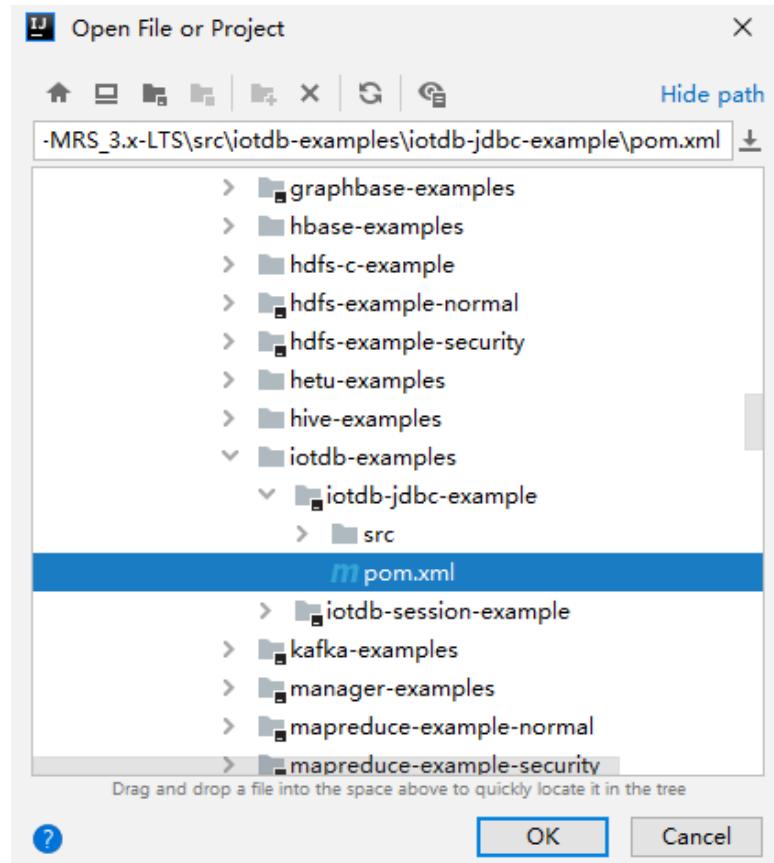
**Step 5** Import the sample project to the IntelliJ IDEA development environment.

1. Start the IntelliJ IDEA, and click **Open or Import** on the quick start page.  
For the IDEA tool that has been used, you can choose **File > Import project...** on the main interface to import the sample project.

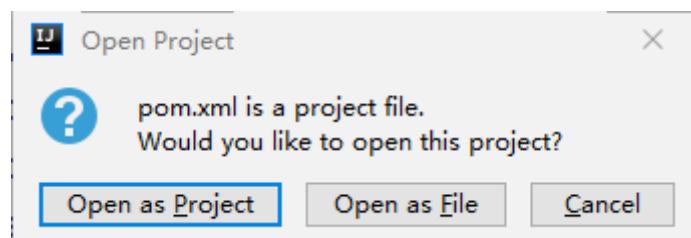
Figure 1-178 Open or Import (quick start page)



2. Select the **pom.xml** file of **iotdb-jdbc-example** in the sample project folder **iotdb-examples**, and click **OK**.

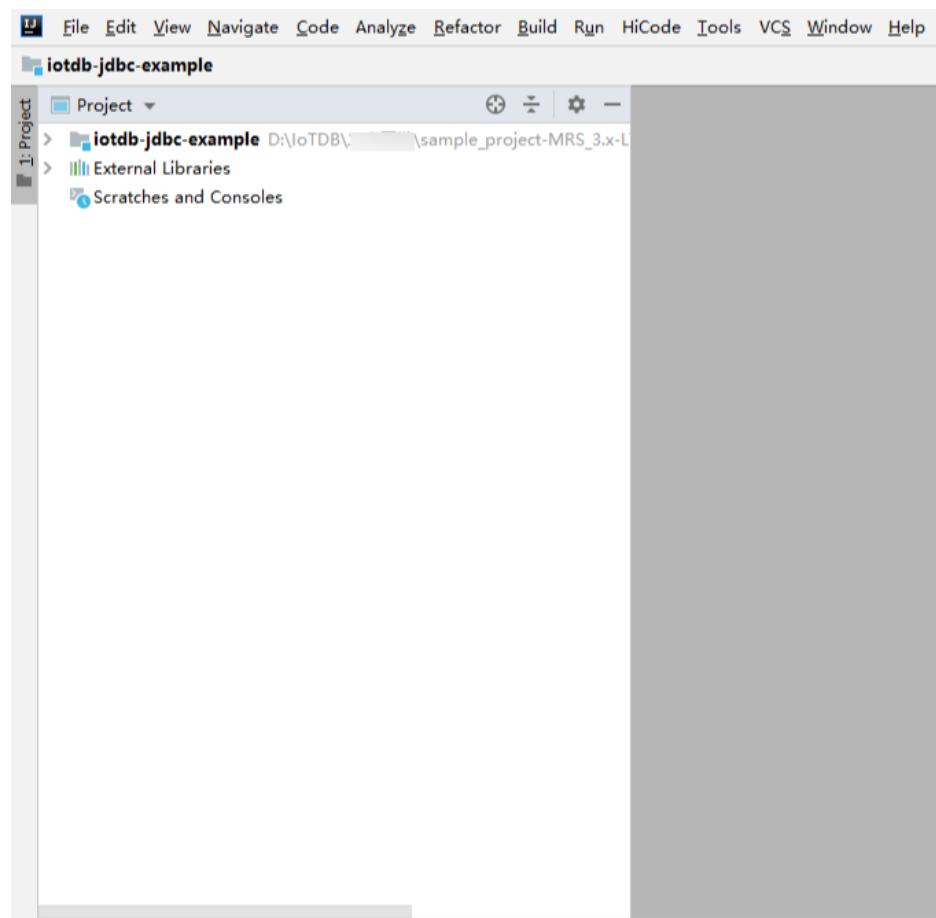


Select Open as Project.



3. After the import is complete, check that the imported sample projects are displayed on the IDEA home page.

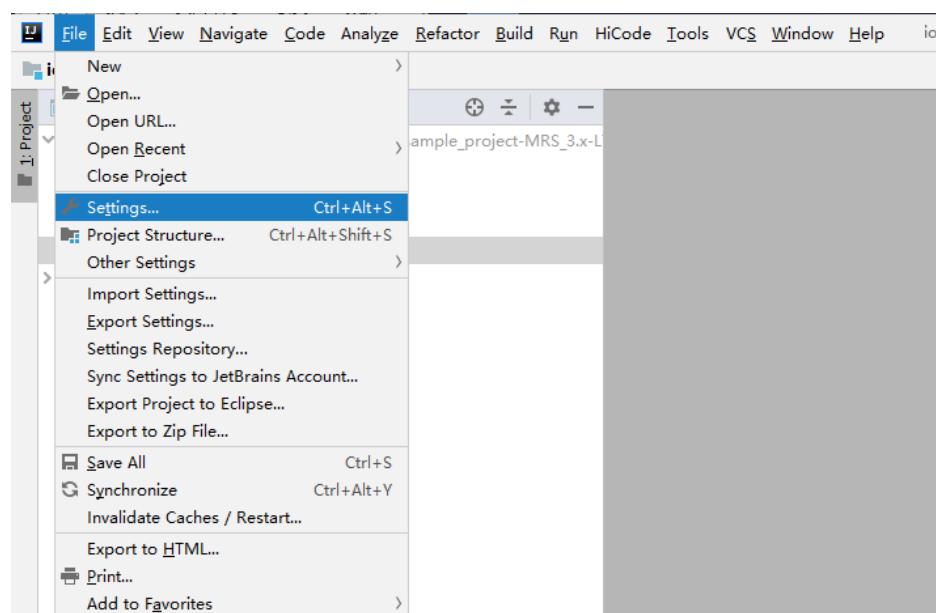
Figure 1-179 Successfully importing sample projects



**Step 6** Set the Maven version used by the project.

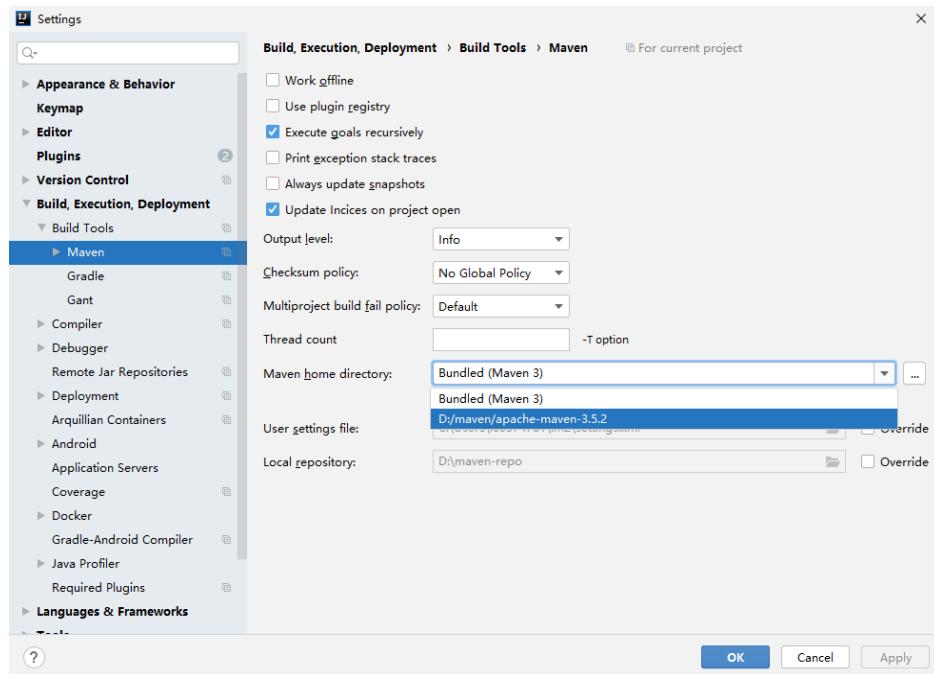
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

Figure 1-180 Settings



2. Choose **Build, Execution, Deployment > Maven** and set **Maven home directory** to the Maven version installed on the local host.  
Configure **User settings file** and **Local repository** based on the site requirements.

**Figure 1-181** Selecting the local Maven installation directory



3. Click **Apply** and **OK** to complete the configuration.

----End

## 1.14.3 Application Development

### 1.14.3.1 IoTDB JDBC

#### 1.14.3.1.1 Java Example Code

##### Description

Run the IoTDB SQL statement in JDBC connection mode.

##### Sample Code

The following code snippet is used as an example. For complete code, see the **com.huawei.bigdata.iotdb.JDBCExample** class.

**jdbc url** includes the IP address, RPC port number, username, and password of the node where the IoTDBServer to be connected is located.

### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see [Preparing for User Authentication](#).
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - Authentication username* is the username for accessing IoTDB.
  - Password* is the password for accessing IoTDB.

```
/**  
 * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.  
 * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/iotDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"  
*/  
private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.  
public static void main(String[] args) throws ClassNotFoundException, SQLException {  
    Class.forName("org.apache.iotdb.jdbc.IoTDBDriver");  
    // set iotdb_ssl_enable  
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);  
    if ("true".equals(IOTDB_SSL_ENABLE)) {  
        // set truststore.jks path  
        System.setProperty("iotdb_ssl_truststore", "truststore file path");  
    }  
    try (Connection connection =  
        DriverManager.getConnection("jdbc:iotdb://IP address of the IoTDBServer instance node:Port",  
        "authentication username", "authentication user password")) {  
        Statement statement = connection.createStatement()  
  
        // set JDBC fetchSize  
        statement.setFetchSize(10000);  
        for (int i = 0; i <= 100; i++) {  
            statement.addBatch(prepareInsertStatement(i));  
        }  
        statement.executeBatch();  
        statement.clearBatch();  
  
        ResultSet resultSet = statement.executeQuery("select ** from root where time <= 10");  
        outputResult(resultSet);  
        resultSet = statement.executeQuery("select count(**) from root");  
        outputResult(resultSet);  
        resultSet =  
            statement.executeQuery(  
                "select count(**) from root where time >= 1 and time <= 100 group by ([0, 100), 20ms, 20ms]");  
        outputResult(resultSet);  
    } catch (IoTDBSQLException e) {  
        System.out.println(e.getMessage());  
    }  
}
```

#### 1.14.3.1.2 Using the keytab File for JDBC Authentication

##### Description

Use the keytab file for JDBC authentication.

## Preparations

Log in to FusionInsight Manager, choose **System > Permission > User**, and download the user credential prepared in [Preparing for User Authentication](#).

## Sample Code

The following code snippet is used as an example. For complete code, see the **com.huawei.bigdata.iotdb.JDBCbyKerberosExample** class.

### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see [Preparing for User Authentication](#).
- Log in to FusionInsight Manager and choose **System > Permission > Domain and Mutual Trust**. On the page that is displayed, the value of **Local Domain** is the domain name.

```
package com.huawei.bigdata.iotdb;

import com.huawei.iotdb.client.security.IoTDBClientKerberosFactory;
import org.apache.iotdb.jdbc.IoTDBSQLException;
import org.ietf.jgss.GSException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Base64;
import javax.security.auth.login.LoginException;

public class JDBCbyKerberosExample {

    /**
     * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
     * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/iotdb/conf directory on the client: iotdb_ssl_enable="false"
     */
    private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.

    private static final String JAVA_KRB5_CONF = "java.security.krb5.conf";
    /**
     * Location of krb5.conf file
     */
    private static final String KRB5_CONF_DEST = "Location of the krb5.conf file in the downloaded authentication credential";
    /**
     * Location of keytab file
     */
    private static final String KEY_TAB_DEST = "Location of the user.keytab file in the downloaded authentication credential";
    /**
     * User principal
     */
    private static final String CLIENT_PRINCIPAL = "User Principal (usually in the format of username@local domain, for example, iotdb_admin@HADOOP.COM)";
```

```
/*
 * Server principal, 'iotdb_server_kerberos_principal' in iotdb-datanode.properties
 */
private static final String SERVER_PRINCIPAL = "iotdb/hadoop.hadoop.com@HADOOP.COM";// You can
obtain this parameter value by searching for iotdb_server_kerberos_principal in ${BIGDATA_HOME}/
FusionInsight_IoTDB_/*_*_IoTDBServer/etc/iotdb-datanode.properties on any node with the IoTDB service
installed.

/**
 * Get kerberos token as password
 * @return kerberos token
 * @throws IOException loginException
 * @throws GSSEException GSSEException
 */
public static String getAuthToken() throws IOException, GSSEException {
    IoTDBClientKerberosFactory kerberosHandler = IoTDBClientKerberosFactory.getInstance();
    System.setProperty(JAVA_KRB5_CONF, KRB5_CONF_DEST);
    kerberosHandler.loginSubjectFromKeytab(PRINCIPAL, KEY_TAB_DEST);
    byte[] tokens = kerberosHandler.generateServiceToken(PRINCIPAL);
    return Base64.getEncoder().encodeToString(tokens);
}

public static void main(String[] args) throws SQLException {
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
        // set truststore.jks path
        System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }

    try (Connection connection =
        DriverManager.getConnection("jdbc:iotdb://IP address of the IoTDBServer instance
node:port number/", "Authentication username", getAuthToken());
        Statement statement = connection.createStatement()) {
        // set JDBC fetchSize
        statement.setFetchSize(10000);

        try {
            statement.execute("SET STORAGE GROUP TO root.sg1");
            statement.execute(
                "CREATE TIMESERIES root.sg1.d1.s1 WITH DATATYPE=INT64, ENCODING=RLE,
COMPRESSOR=SNAPPY");
            statement.execute(
                "CREATE TIMESERIES root.sg1.d1.s2 WITH DATATYPE=INT64, ENCODING=RLE,
COMPRESSOR=SNAPPY");
            statement.execute(
                "CREATE TIMESERIES root.sg1.d1.s3 WITH DATATYPE=INT64, ENCODING=RLE,
COMPRESSOR=SNAPPY");
        } catch (IoTDBSQLException e) {
            System.out.println(e.getMessage());
        }
    } catch (GSSEException | IOException e) {
        System.out.println(e.getMessage());
    }
}
}
```

### 1.14.3.2 IoTDB Session

#### 1.14.3.2.1 Java Example Code

##### Description

Run the IoTDB SQL statement in Session connection mode.

## Sample Code

The following code snippet is used as an example. For complete code, see [com.huawei.bigdata.SessionExample](#).

Change **HOST\_1**, **HOST\_2**, and **HOST\_3** to the service IP addresses of the nodes accommodating IoTDBServer. Set the username and password in the **Session** object.

### NOTE

- On Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP addresses of the nodes where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see [Preparing for User Authentication](#).
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication.username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
/**  
 * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.  
 * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"  
 */  
private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.  
public static void main(String[] args)  
    throws IoTDBConnectionException, StatementExecutionException {  
    // set iotdb_ssl_enable  
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);  
    if ("true".equals(IOTDB_SSL_ENABLE)) {  
        // set truststore.jks path  
        System.setProperty("iotdb_ssl_truststore", "truststore file path");  
    }  
  
    session = new Session(nodeUrls, Port, Authentication.username, Authentication.user.password);  
    session.open(false);  
  
    // set session fetchSize  
    session.setFetchSize(10000);  
  
    try {  
        session.setStorageGroup("root.sg1");  
    } catch (StatementExecutionException e) {  
        if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {  
            throw e;  
        }  
    }  
  
    createTimeseries();  
    createMultiTimeseries();  
    insertRecord();  
    insertTablet();  
    insertTablets();  
    insertRecords();  
}
```

```
nonQuery();
query();
queryWithTimeout();
rawDataQuery();
queryByIterator();
deleteData();
deleteTimeseries();
setTimeout();

sessionEnableRedirect = new Session(nodeUrls, Authentication.username, Authentication.user.password);
sessionEnableRedirect.setEnableQueryRedirection(true);
sessionEnableRedirect.open(false);

// set session fetchSize
sessionEnableRedirect.setFetchSize(10000);

insertRecord4Redirect();
query4Redirect();
sessionEnableRedirect.close();
session.close();
}
```

### 1.14.3.2.2 Using the Keytab File for Session Authentication

#### Description

Use the Keytab file for session authentication.

#### Preparations

Log in to FusionInsight Manager, choose **System > Permission > User**, and download the user credential prepared in [Preparing for User Authentication](#).

#### Sample Code

The following code snippet is used as an example. For complete code, see the **com.huawei.bigdata.iotdb.SessionbyKerberosExample** class.

##### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see [Preparing for User Authentication](#).
- Log in to FusionInsight Manager and choose **System > Permission > Domain and Mutual Trust**. On the page that is displayed, the value of **Local Domain** is the domain name.

```
package com.huawei.bigdata.iotdb;

import org.apache.iotdb.rpc.IoTDBConnectionException;
import org.apache.iotdb.rpc.StatementExecutionException;
import org.apache.iotdb.rpc.TSStatusCode;
import org.apache.iotdb.session.Session;
import org.apache.iotdb.tsfile.file.metadata.enums.CompressionType;
import org.apache.iotdb.tsfile.file.metadata.enums.TSDDataType;
import org.apache.iotdb.tsfile.file.metadata.enums.TSEncoding;
```

```
import org.ietf.jgss.GSSEException;
import java.util.Base64;
import javax.security.auth.login.LoginException;

public class SessionbyKerberosExample{
    private static Session session;
    private static final String ROOT_SG1_D1_S1 = "root.sg1.d1.s1";
    private static final String ROOT_SG1_D1_S2 = "root.sg1.d1.s2";
    private static final String ROOT_SG1_D1_S3 = "root.sg1.d1.s3";
    private static final String HOST = "127.0.0.1";
    private static final String PORT = "22260";

    /**
     * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
     * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
     */
    private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.

    private static final String JAVA_KRB5_CONF = "java.security.krb5.conf";
    /**
     * Location of krb5.conf file
     */
    private static final String KRB5_CONF_DEST = "Location of the krb5.conf file in the downloaded authentication credential";
    /**
     * Location of keytab file
     */
    private static final String KEY_TAB_DEST = "Location of the user.keytab file in the downloaded authentication credential";
    /**
     * User principal
     */

    private static final String CLIENT_PRINCIPAL = "User Principal (usually in the format of username@local domain, for example, iotdb_admin@HADOOP.COM)";
    /**
     * Server principal, 'iotdb_server_kerberos_principal' in iotdb-datanode.properties
     */
    private static final String SERVER_PRINCIPAL = "iotdb/hadoop.hadoop.com@HADOOP.COM";// You can obtain this parameter value by searching for iotdb_server_kerberos_principal in /opt/huawei/Bigdata/FusionInsight_IoTDB_*/_*_IoTDBServer/etc/iotdb-datanode.properties on any node with the IoTDB service installed.

    /**
     * Get kerberos token as password
     * @return kerberos token
     * @throws LoginException loginException
     * @throws GSSEException GSSEException
     */
    public static String getAuthToken() throws LoginException, GSSEException {
        IoTDBClientKerberosFactory kerberosHandler = IoTDBClientKerberosFactory.getInstance();
        System.setProperty(JAVA_KRB5_CONF, KRB5_CONF_DEST);
        kerberosHandler.loginSubjectFromKeytab(PRINCIPAL, KEY_TAB_DEST);
        byte[] tokens = kerberosHandler.generateServiceToken(PRINCIPAL);
        return Base64.getEncoder().encodeToString(tokens);
    }

    public static void main(String[] args)
    throws IoTDBConnectionException, StatementExecutionException, GSSEException, LoginException {
        // set iotdb_ssl_enable
        System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
        if ("true".equals(IOTDB_SSL_ENABLE)) {
            // set truststore.jks path
            System.setProperty("iotdb_ssl_truststore", "truststore file path");
        }
    }
}
```

```
session = new Session(HOST, PORT, "Authentication username", getAuthToken());  
session.open(false);  
  
// set session fetchSize  
session.setFetchSize(10000);  
  
try {  
    session.setStorageGroup("root.sg1");  
} catch (StatementExecutionException e) {  
    if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {  
        throw e;  
    }  
}  
createTimeseries();  
  
private static void createTimeseries() throws IoTDBConnectionException, StatementExecutionException {  
if (!session.checkTimeseriesExists(ROOT_SG1_D1_S1)) {  
    session.createTimeseries(  
        ROOT_SG1_D1_S1, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);  
}  
if (!session.checkTimeseriesExists(ROOT_SG1_D1_S2)) {  
    session.createTimeseries(  
        ROOT_SG1_D1_S2, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);  
}  
if (!session.checkTimeseriesExists(ROOT_SG1_D1_S3)) {  
    session.createTimeseries(  
        ROOT_SG1_D1_S3, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);  
}  
}
```

### 1.14.3.3 IoTDB Flink

#### 1.14.3.3.1 FlinkIoTDBSink

##### Description

It is an integration of IoTDB and Flink. This module contains IoTDBSink and writes time series data into IoTDB using a Flink job.

##### Sample Code

This example shows how to send data from a Flink job to an IoTDB server.

- A simulated source SensorSource generates a data point every second.
- Flink uses IoTDBSink to consume produced data and write the data into IoTDB.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

## NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permissions to operate the IoTDB and Flink services. For details, see [Preparing for User Authentication](#).
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
public class FlinkIoTDBSink {  
    public static void main(String[] args) throws Exception {  
        // run the flink job on local mini cluster  
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
  
        IoTDBSinkOptions options = new IoTDBSinkOptions();  
        options.setHost("127.0.0.1");  
        options.setPort(22260);  
        options.setUser ("Authentication username");  
        options.setPassword ("Password");  
  
        // If the server enables auto_create_schema, then we do not need to register all timeseries  
        // here.  
        options.setTimeseriesOptionList(  
            Lists.newArrayList(  
                new IoTDBSinkOptions.TimeseriesOption(  
                    "root.sg.d1.s1", TSDDataType.DOUBLE, TSEncoding.GORILLA, CompressionType.SNAPPY)));  
  
        IoTSerializationSchema serializationSchema = new DefaultIoTSerializationSchema();  
        IoTDBSink ioTDBSink =  
            new IoTDBSink(options, serializationSchema)  
                // enable batching  
                .withBatchSize(10)  
                // how many connections to the server will be created for each parallelism  
                .withSessionPoolSize(3);  
  
        env.addSource(new SensorSource())  
            .name("sensor-source")  
            .setParallelism(1)  
            .addSink(ioTDBSink)  
            .name("iotdb-sink");  
  
        env.execute("iotdb-flink-example");  
    }  
  
    private static class SensorSource implements SourceFunction<Map<String, String>> {  
        boolean running = true;  
        Random random = new SecureRandom();  
  
        @Override  
        public void run(SourceContext context) throws Exception {  
            while (running) {  
                Map<String, String> tuple = new HashMap();  
                tuple.put("device", "root.sg.d1");  
                tuple.put("timestamp", String.valueOf(System.currentTimeMillis()));  
                tuple.put("measurements", "s1");  
                tuple.put("types", "DOUBLE");  
                tuple.put("values", String.valueOf(random.nextDouble()));  
            }  
        }  
    }  
}
```

```
        context.collect(tuple);
        Thread.sleep(1000);
    }
}

@Override
public void cancel() {
    running = false;
}
}
```

### 1.14.3.3.2 FlinkIoTDBSource

#### Description

It is an integration of IoTDB and Flink. This module contains IoTDBSource and reads time series data from IoTDB and prints the data using a Flink job.

#### Sample Code

This example shows how a Flink job reads time series data from a IoTDB server.

- Flink uses IoTDBSource to read data from a IoTDB server.
- To use IoTDBSource, you need to construct an IoTDBSource instance by specifying **IoTDBSourceOptions** and implementing the abstract method **convert()** in IoTDBSource. The **convert()** method defines how you want to convert row data.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

#### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instance** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permissions to operate the IoTDB and Flink services. For details, see [Preparing for User Authentication](#).
- If SSL is enabled for a cluster in security mode, copy the **truststore.jks** file to all nodes where NodeManager instances of Yarn reside in the current cluster. The copy path must be the same as **truststore file path** in the following code, for example, **/opt/truststore.jks**.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
public class FlinkIoTDBSource {
    /**
     * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
     * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following
```

```
configuration in the iotdb-client.env file in the Client installation directory/loTDB/iotdb/conf directory  
on the client: iotdb_ssl_enable="false"  
*/  
private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.  
static final String LOCAL_HOST = "127.0.0.1";  
static final String ROOT_SG1_D1_S1 = "root.sg1.d1.s1";  
static final String ROOT_SG1_D1 = "root.sg1.d1";  
  
public static void main(String[] args) throws Exception {  
    // use session api to create data in loTDB  
    prepareData();  
  
    // run the flink job on local mini cluster  
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
  
    IoTDBSourceOptions ioTDBSourceOptions =  
        new IoTDBSourceOptions()  
            .setHost(LOCAL_HOST, 22260, "Authentication username", "Password", "select s1 from "+ ROOT_SG1_D1 +"  
align by device");  
  
    IoTDBSource<RowRecord> source =  
        new IoTDBSource<RowRecord>(ioTDBSourceOptions) {  
            @Override  
            public RowRecord convert(RowRecord rowRecord) {  
                return rowRecord;  
            }  
        };  
    env.addSource(source).name("sensor-source").print().setParallelism(2);  
    env.execute();  
}  
  
private static void prepareData()  
    throws IoTDBConnectionException, StatementExecutionException, TTransportException {  
    // set iotdb_ssl_enable  
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);  
    if ("true".equals(IOTDB_SSL_ENABLE)) {  
        // set truststore.jks path  
        System.setProperty("iotdb_ssl_truststore", "truststore file path");  
    }  
    Session session = new Session(LOCAL_HOST, 22260, "Authentication username", "Password");  
    session.open(false);  
    try {  
        session.setStorageGroup("root.sg1");  
        if (!session.checkTimeseriesExists(ROOT_SG1_D1_S1)) {  
            session.createTimeseries(  
                ROOT_SG1_D1_S1, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);  
            List<String> measurements = new ArrayList<>();  
            measurements.add("s1");  
            measurements.add("s2");  
            measurements.add("s3");  
            List<TSDataType> types = new ArrayList<>();  
            types.add(TSDataType.INT64);  
            types.add(TSDataType.INT64);  
            types.add(TSDataType.INT64);  
  
            for (long time = 0; time < 1000; time++) {  
                List<Object> values = new ArrayList<>();  
                values.add(1L);  
                values.add(2L);  
                values.add(3L);  
                session.insertRecord(ROOT_SG1_D1, time, measurements, types, values);  
            }  
        }  
    } catch (StatementExecutionException e) {  
        if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {  
            throw e;  
        }  
    }  
}
```

```
}
```

### 1.14.3.4 IoTDB Kafka

#### 1.14.3.4.1 Java Example Code

##### Description

This section describes how to use Kafka to send data to IoTDB.

##### Sample Code

- Producer.java:

This example shows how to send time series data to a Kafka cluster.

- Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.
- The default time series data format of this sample is *Device name, Timestamp, Value*, for example, **sensor\_1,1642215835758,1.0**. You can change the value of **IOTDB\_DATA\_SAMPLE\_TEMPLATE** in the **Constant.java** file based on the site requirements.

```
public static void main(String[] args) {  
    // whether use security mode  
    final boolean isSecurityModel = true;  
  
    if (isSecurityModel) {  
        try {  
            LOG.info("Security mode start.");  
  
            // Note: During security authentication, you need to manually change the account to a  
            // machine-machine one that you have applied for.  
            LoginUtil.securityPrepare(Constant.USER_PRINCIPAL, Constant.USER_KEYTAB_FILE);  
        } catch (IOException e) {  
            LOG.error("Security prepare failure.", e);  
            return;  
        }  
        LOG.info("Security prepare success.");  
    }  
  
    // whether to use the asynchronous sending mode  
    final boolean asyncEnable = false;  
    Producer producerThread = new Producer(KafkaProperties.TOPIC, asyncEnable);  
}
```

##### NOTE

For details about the Kafka producer code, see [Producer API Usage Sample](#).

- KafkaConsumerMultThread.java:

This example shows how to write data from a Kafka cluster to IoTDB using multiple threads. Kafka cluster data is generated by **Producer.java**.

- Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.

- b. Change the value of **CONCURRENCY\_THREAD\_NUM** in the **KafkaConsumerMultThread.java** file to adjust the number of consumer threads.

#### NOTICE

- If multiple threads are used to consume Kafka cluster data, ensure that the number of consumed topic partitions is greater than 1.
- The Kafka client configuration files **client.properties** and **log4j.properties** must be stored in the configuration file directory of the program.

- c. Set the IP address, port number, username, and password of the node where the IoTDBServer is located in the IoTDBSessionPool object parameters.

#### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB** and click the **Instance** tab to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port number is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, choose **Configurations > All Configurations**, and search for **rpc\_port**.
- In security mode, the username and password for logging in to the node where IoTDBServer resides are controlled by FusionInsight Manager. Ensure that the user has the permission to operate the IoTDB service. For details, see [Preparing for User Authentication](#).
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
/*
 * In security mode, the default value of SSL_ENABLE is true. You need to import the truststore.jks file.
 * In security mode, you can also log in to FusionInsight Manager, choose Cluster > Services > IoTDB > Configuration, search for SSL in the search box, and change the value of SSL_ENABLE to false. After saving the configuration, restart the IoTDB service for the configuration to take effect. Modify the following configuration in the iotdb-client.env file in the Client installation directory/IoTDB/iotdb/conf directory on the client: iotdb_ssl_enable="false"
 */
private static final String IOTDB_SSL_ENABLE = "true"; // Set it to the SSL_ENABLE value.
public static void main(String[] args) {
    // whether use security mode
    final boolean isSecurityModel = true;

    if (isSecurityModel) {
        try {
            LOG.info("Securitymode start.");

            // Note: During security authentication, you need to manually change the account to a machine-machine one.
            LoginUtil.securityPrepare(Constant.USER_PRINCIPAL, Constant.USER_KEYTAB_FILE);
        } catch (IOException e) {
            LOG.error("Security prepare failure.", e);
            return;
        }
        LOG.info("Security prepare success.");
    }
}
```

```
        }
        // set iotdb_ssl_enable
        System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
        if ("true".equals(IOTDB_SSL_ENABLE)) {
            // set truststore.jks path
            System.setProperty("iotdb_ssl_truststore", "truststore file path");
        }

        // create IoTDB session connection pool
        IoTDBSessionPool sessionPool = new IoTDBSessionPool("127.0.0.1", 22260, "Authentication
username", "Password", 3);

        // start consumer thread
        KafkaConsumerMultThread consumerThread = new
        KafkaConsumerMultThread(KafkaProperties.TOPIC, sessionPool);
        consumerThread.start();
    }

    /**
     * consumer thread
     */
    private class ConsumerThread extends ShutdownableThread {
        private int threadNum;
        private String topic;
        private KafkaConsumer<String, String> consumer;
        private IoTDBSessionPool sessionPool;

        public ConsumerThread(int threadNum, String topic, Properties props, IoTDBSessionPool
sessionPool) {
            super("ConsumerThread" + threadNum, true);
            this.threadNum = threadNum;
            this.topic = topic;
            this.sessionPool = sessionPool;
            this.consumer = new KafkaConsumer<>(props);
            consumer.subscribe(Collections.singleton(this.topic));
        }

        public void doWork() {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(waitTime));
            for (ConsumerRecord<String, String> record : records) {
                LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + "
record: "
+ record.value() + " offsets: " + record.offset());

                // insert kafka consumer data to iotdb
                sessionPool.insertRecord(record.value());
            }
        }
    }
}
```

#### NOTE

For details about the Kafka consumer code, see [Multi-thread Producer Sample](#).

### 1.14.3.5 IoTDB UDF Program

#### 1.14.3.5.1 IoTDB UDF Sample Code

#### Description

This section describes how to implement a simple IoTDB user-defined function (UDF). For details, see "UDFs" in the *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

## Sample Code

```
package com.huawei.bigdata.iotdb;
import org.apache.iotdb.udf.api.UDTF;
import org.apache.iotdb.udf.api.access.Row;
import org.apache.iotdb.udf.api.collector.PointCollector;
import org.apache.iotdb.udf.api.customizer.config.UDTFConfigurations;
import org.apache.iotdb.udf.api.customizer.parameter.UDFParameters;
import org.apache.iotdb.udf.api.customizer.strategy.RowByRowAccessStrategy;
import org.apache.iotdb.udf.api.type.Type;
import java.io.IOException;

public class UDTFExample implements UDTF {
    @Override
    public void beforeStart(UDFParameters parameters, UDTFConfigurations configurations) {
        configurations.setAccessStrategy(new RowByRowAccessStrategy()).setOutputDataType(Type.INT32);
    }

    @Override
    public void transform(Row row, PointCollector collector) throws IOException {
        collector.putInt(row.getTime(), -row.getInt(0));
    }
}
```

## 1.14.4 Application Commissioning

### 1.14.4.1 Commissioning Applications on Windows

#### 1.14.4.1.1 Compiling and Running Applications

##### Scenario

Run applications in a Windows development environment after application code is developed. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

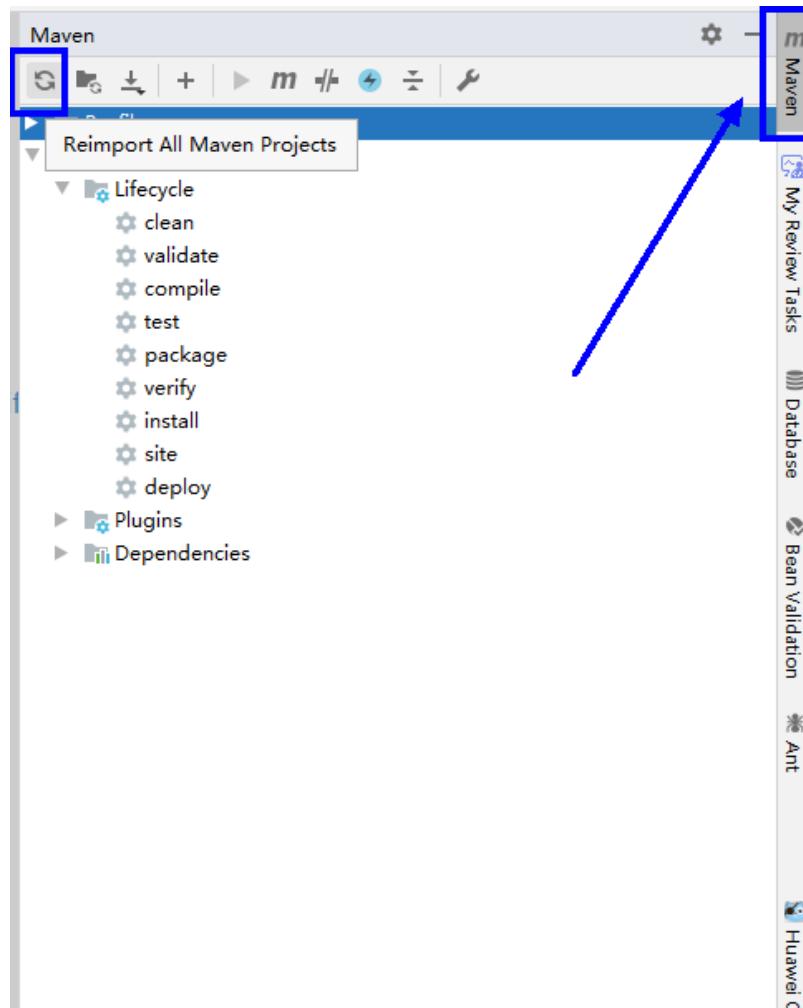
##### NOTE

- If the IBM JDK is used in the Windows environment, applications cannot be directly run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the **hosts** file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

##### Procedure

**Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

Figure 1-182 reimport projects



**Step 2** Compile and run the application.

Modify the IP address, port number, login username, and password of the IoTDBServer node that matches the code.

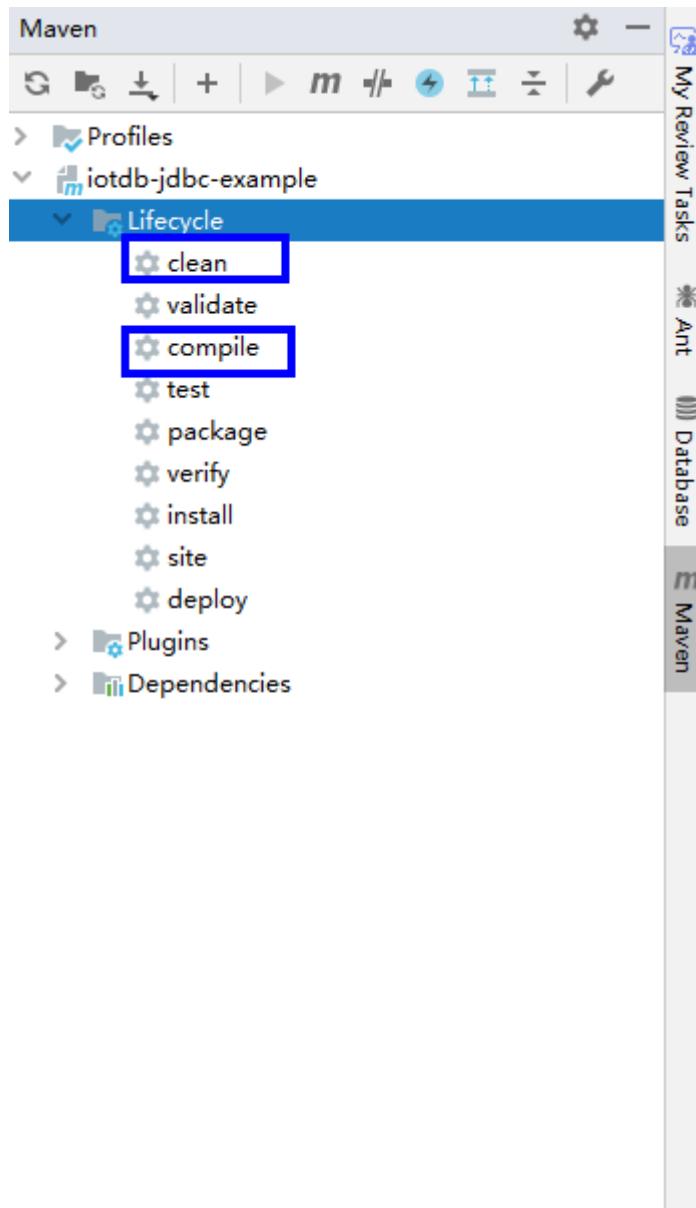
1. Use either of the following two methods:

- Method 1

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile** and double-click **compile** to run the **compile** command of Maven.

Figure 1-183 Maven tools clean and compile



- Method 2

Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 1-184 Enter **mvn clean compile** in the IDEA **Terminal** text box.

```
Terminal: Local +  
Microsoft Windows [Version 10.0.18362.592]  
(c) 2019 Microsoft Corporation. All rights reserved.  
D:\IoTDB\develop\sample_project-MRS_3.x-LTS\src\iotdb-examples\iotdb-jdbc-example>mvn clean compile
```

After the compilation is complete, the message "BUILD SUCCESS" is displayed.

**Figure 1-185** Compilation completed



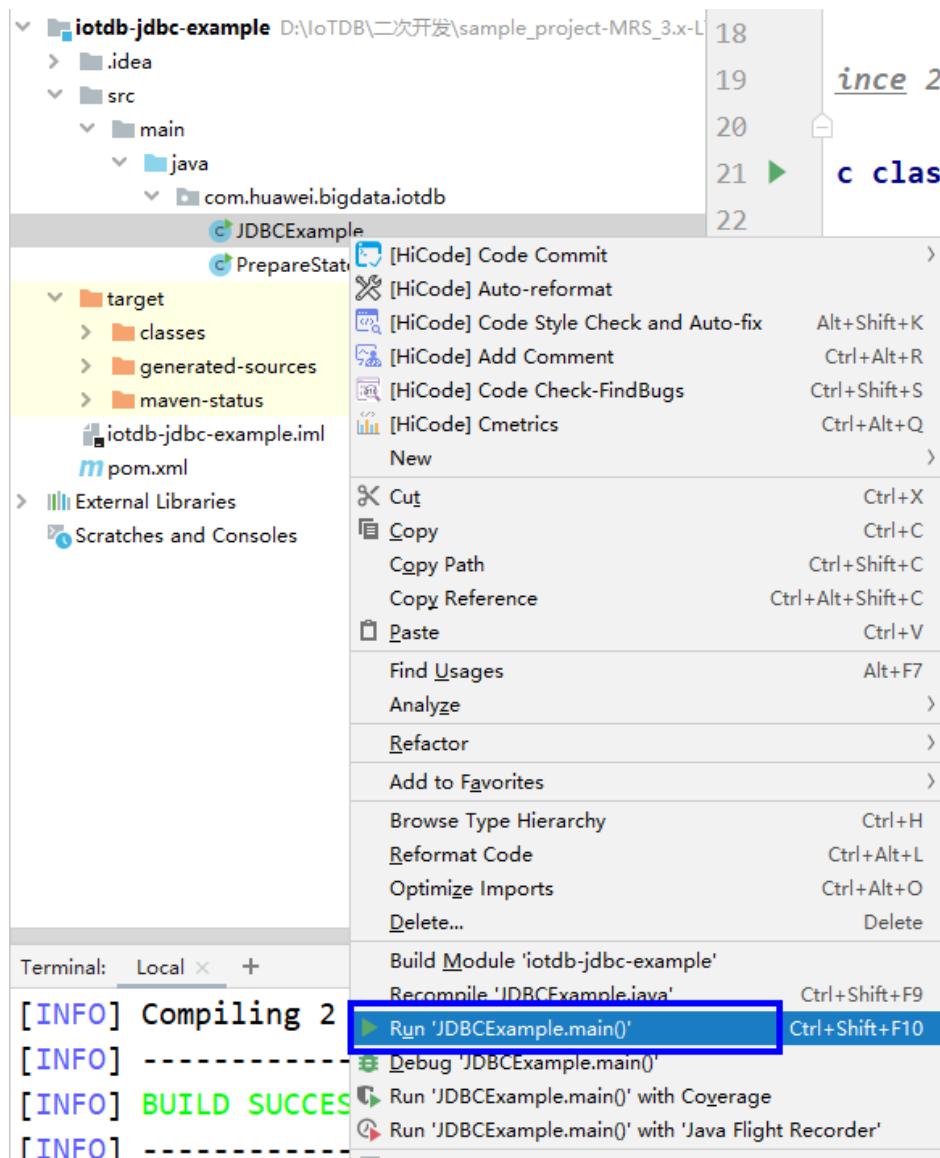
```
Terminal: Local +  
[INFO] Compiling 2 source files to D:\IoTDB\develop\sample_project-MRS_3.x-LTS\src\io  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.156 s  
[INFO] Finished at: 2021-06-16T14:29:36+08:00  
[INFO] -----  
D:\IoTDB\develop\sample_project-MRS_3.x-LTS\src\iotdb-examples\iotdb-jdbc-example>[]
```

The screenshot shows a terminal window titled "Local". The output of the compilation process is displayed, including the message "BUILD SUCCESS". The terminal window has a tab bar with several icons and labels: Cmetrics, CodeCheck-FindBugs, CodeReview, CodeStyleCheckAndFix, Terminal, Build, Messages, Run, and TODO. The "Terminal" tab is currently selected.

2. Run the application. A JDBC application is used as an example. The operations for running other applications are the same.

Right-click the **JDBCExample.java** file and choose **Run 'JDBCExample.main()'** from the shortcut menu.

Figure 1-186 Running the application



----End

#### 1.14.4.1.2 Viewing Commissioning Results

After the IoTDB application is run, you can view the running results in IntelliJ IDEA.

- The running result of the JDBCExample example application is as follows:

```

-----
Time root.sg.d1.s1 root.company.line2.device1.temperature root.company.line2.device1.speed
root.company.line2.device2.speed root.company.line2.device2.status root.company.line1.device1.spin
root.company.line1.device1.status root.company.line1.device2.temperature
root.company.line1.device2.power root.sg1.d1.s3 root.sg1.d1.s1 root.sg1.d1.s2
0, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
1, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
2, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
3, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
4, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0
5, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0

```

```
6, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0  
7, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0  
8, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0  
9, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0  
10, null, null, null, null, null, null, null, null, null, 1.0, 1.0, 1.0  
-----  
-----  
count(root.sg.d1.s1) count(root.company.line2.device1.temperature)  
count(root.company.line2.device1.speed) count(root.company.line2.device2.speed)  
count(root.company.line2.device2.status) count(root.company.line1.device1.spin)  
count(root.company.line1.device1.status) count(root.company.line1.device2.temperature)  
count(root.company.line1.device2.power) count(root.sg1.d1.s3) count(root.sg1.d1.s1)  
count(root.sg1.d1.s2)  
8237, 1, 1, 1, 1, 1, 1, 1, 101, 101, 101  
-----  
-----  
Time count(root.sg.d1.s1) count(root.company.line2.device1.temperature)  
count(root.company.line2.device1.speed) count(root.company.line2.device2.speed)  
count(root.company.line2.device2.status) count(root.company.line1.device1.spin)  
count(root.company.line1.device1.status) count(root.company.line1.device2.temperature)  
count(root.company.line1.device2.power) count(root.sg1.d1.s3) count(root.sg1.d1.s1)  
count(root.sg1.d1.s2)  
0, 0, 0, 0, 0, 0, 0, 0, 19, 19, 19  
20, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20  
40, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20  
60, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20  
80, 0, 0, 0, 0, 0, 0, 0, 20, 20, 20
```

- The running result of the FlinkIoTDBSink example application is as follows:

```
...  
19:53:41.532 [flink-akka.actor.default-dispatcher-9] DEBUG  
org.apache.flink.runtime.resourcemanager.StandaloneResourceManager - Received heartbeat from  
5153e4ff24b25b13225f1bf67a4312d8.  
19:53:41.800 [flink-akka.actor.default-dispatcher-9] DEBUG  
org.apache.flink.runtime.jobmaster.JobMaster - Trigger heartbeat request.  
19:53:41.800 [flink-akka.actor.default-dispatcher-10] DEBUG  
org.apache.flink.runtime.taskexecutor.TaskExecutor - Received heartbeat request from  
5153e4ff24b25b13225f1bf67a4312d8.  
19:53:41.802 [flink-akka.actor.default-dispatcher-9] DEBUG  
org.apache.flink.runtime.jobmaster.JobMaster - Received heartbeat from 7d6ef313-3f78-4cee-bbb1-e234dcac6d30.  
19:53:42.988 [pool-3-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully  
19:53:42.988 [pool-6-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully  
19:53:42.990 [pool-4-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully  
19:53:45.990 [pool-7-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully  
19:53:45.992 [pool-9-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully  
19:53:45.994 [pool-5-thread-1] DEBUG org.apache.iotdb.flink.IoTDBSink - send event successfully
```

- The running result of the IoTDB Kafka example application is as follows:

- Producer.java

```
...  
[2022-01-15 15:12:34,221] INFO New Producer: start. (com.huawei.bigdata.iotdb.Producer)  
[2022-01-15 15:12:39,369] INFO [Producer clientId=DemoProducer] Cluster ID:  
uDtuaWS_QUK02EtuZQ4Xew (org.apache.kafka.clients.Metadata)  
[2022-01-15 15:12:57,077] INFO The Producer have send 100 messages.  
(com.huawei.bigdata.iotdb.Producer)  
[2022-01-15 15:13:04,691] INFO The Producer have send 200 messages.  
(com.huawei.bigdata.iotdb.Producer)  
[2022-01-15 15:13:11,355] INFO The Producer have send 300 messages.  
(com.huawei.bigdata.iotdb.Producer)  
[2022-01-15 15:13:17,758] INFO The Producer have send 400 messages.  
(com.huawei.bigdata.iotdb.Producer)  
[2022-01-15 15:13:24,335] INFO The Producer have send 500 messages.  
(com.huawei.bigdata.iotdb.Producer)  
[2022-01-15 15:13:30,739] INFO The Producer have send 600 messages.  
(com.huawei.bigdata.iotdb.Producer)
```

```
[2022-01-15 15:13:37,267] INFO The Producer have send 700 messages.  
(com.huawei.bigdata.iotdb.Producer)
```

- KafkaConsumerMultThread.java

```
...  
[2022-01-15 15:19:27,563] INFO Consumer Thread-1 partitions:1 record:  
sensor_29,1642231023769,1.000000 offsets: 828  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,612] INFO Consumer Thread-1 partitions:1 record:  
sensor_31,1642231023769,1.000000 offsets: 829  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,612] INFO Consumer Thread-0 partitions:0 record:  
sensor_8,1642231023769,1.000000 offsets: 842  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,665] INFO Consumer Thread-1 partitions:1 record:  
sensor_32,1642231023769,1.000000 offsets: 830  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,665] INFO Consumer Thread-0 partitions:0 record:  
sensor_9,1642231023769,1.000000 offsets: 843  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,732] INFO Consumer Thread-1 partitions:1 record:  
sensor_33,1642231023769,1.000000 offsets: 831  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,732] INFO Consumer Thread-0 partitions:0 record:  
sensor_11,1642231023769,1.000000 offsets: 844  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,786] INFO Consumer Thread-0 partitions:0 record:  
sensor_12,1642231023769,1.000000 offsets: 845  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-15 15:19:27,786] INFO Consumer Thread-1 partitions:1 record:  
sensor_35,1642231023769,1.000000 offsets: 832  
(com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
```

## 1.14.4.2 Commissioning JDBC and Session Applications on Linux

### 1.14.4.2.1 Compiling and Running Applications

#### Scenario

IoTDB applications can run in a Linux environment where an IoTDB client is installed. After the application code is developed, you can upload the JAR file to the prepared Linux environment. A Session application is used as an example. Operations for JDBC applications are the same as those for Session applications.

#### Prerequisites

- You have installed the IoTDB client.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

#### Procedure

##### Step 1 Export a JAR file.

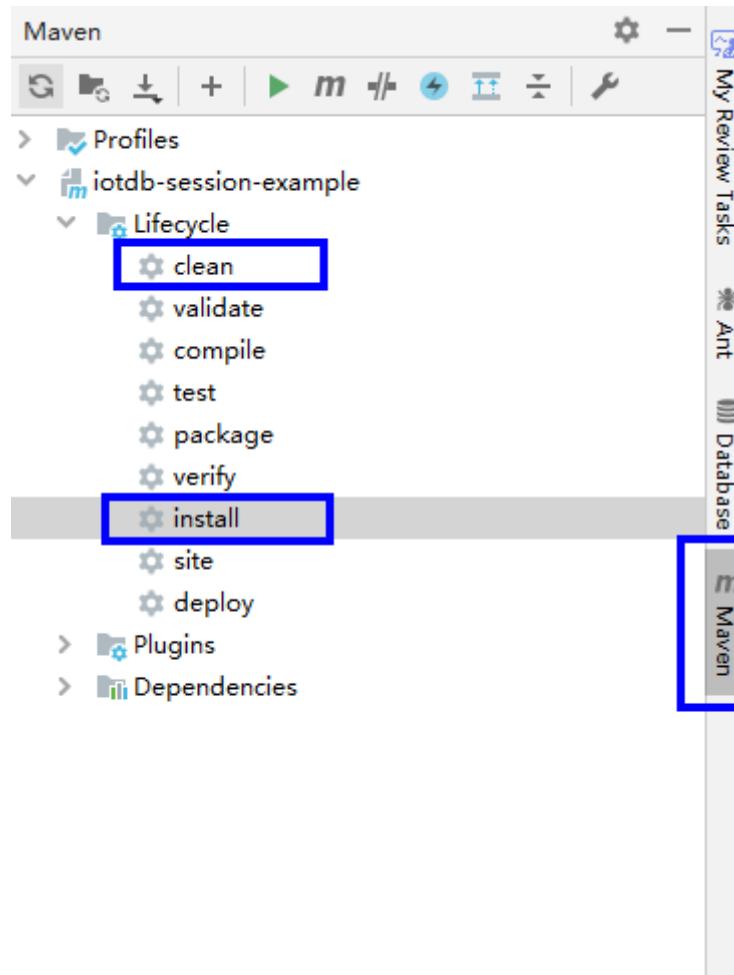
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

**Figure 1-187** Maven tools clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the file.

**Figure 1-188** Entering mvn clean install in the IDEA Terminal text box

```
Terminal: Local +  
1006\iotdb-session-example-0.12.0-hw-ei-311006.pom  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.401 s  
[INFO] Finished at: 2021-07-31T11:22:12+08:00  
[INFO] -----  
D:\IoTDB\1006\sample_project\src\iotdb-examples\iotdb-session-example>mvn clean install
```

After the compilation is completed, the message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

**Step 2** Prepare the dependent JAR file.

1. Log in to the IoTDB client and import the JAR file generated in [Step 1](#) to the **lib** directory of the IoTDB client, for example, **/opt/hadoopclient/IoTDB/iotdb/lib**.
2. Upload the user authentication files **user.keytab** and **krb5.conf** obtained in [Preparing for User Authentication](#) to *Client installation directory/IoTDB/iotdb/conf*.
3. In the root directory of the IoTDB client, for example, **/opt/hadoopclient/IoTDB/iotdb**, create the **run.sh** script, modify the content as follows, and save the modification.

```
#!/bin/sh
BASEDIR=`cd $(dirname $0);pwd`
cd ${BASEDIR}
for file in ${BASEDIR}/lib/*.jar
do
i_cp=$i_cp:$file
echo "$file"
done
for file in ${BASEDIR}/conf/*
do
i_cp=$i_cp:$file
done

java -cp ${i_cp} com.huawei.bigdata.iotdb.JDBCExample
```

*com.huawei.bigdata.iotdb.JDBCExample* is an example. Use the actual code instead.

4. Run the **run.sh** script to run the JAR file.

```
sh /opt/hadoopclient/IoTDB/iotdb/run.sh
```

----End

#### 1.14.4.2.2 Viewing Commissioning Results

If the application running is successful, the following information is displayed.

```
11:53:47.586 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect2.d1 where time >= 1 and time < 10 and root.redirect2.d1.s1 > 1
[Time, root.redirect2.d1.s1, root.redirect2.d1.s2]
1 5 3 4
2 6 5 5
3 7 6 6
11:53:47.590 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect3.d1 where time >= 1 and time < 10 and root.redirect3.d1.s1 > 1
[Time, root.redirect3.d1.s1, root.redirect3.d1.s2]
1 5 3 4
2 6 4 5
3 7 5 6
11:53:47.596 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect4.d1 where time >= 1 and time < 10 and root.redirect4.d1.s1 > 1
[Time, root.redirect4.d1.s1, root.redirect4.d1.s2]
1 5 2 3
2 6 3 4
3 7 4 5
4 8 5 6
11:53:47.601 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect5.d1 where time >= 1 and time < 10 and root.redirect5.d1.s1 > 1
[Time, root.redirect5.d1.s1, root.redirect5.d1.s2]
1 4 2 3
2 5 3 4
3 6 4 5
4 7 5 6
[root@8.5.248.2 iotdb]
```

#### 1.14.4.3 Commissioning Flink Applications on Flink Web UI and Linux

##### 1.14.4.3.1 Compiling and Running Applications

###### Scenario

IoTDB applications can run in a Linux environment where the Flink client is installed and in an environment where the Flink web UI is installed. After the application code is developed, you can upload the JAR file to the prepared environment.

## Prerequisites

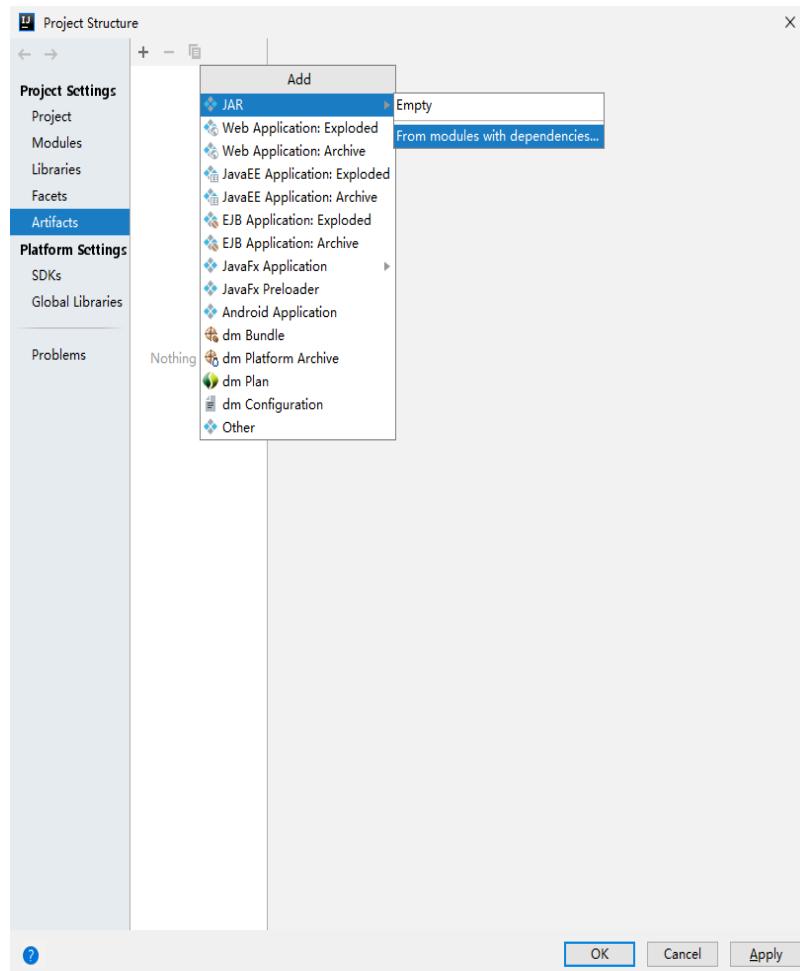
- The Flink component has been installed in the cluster and the FlinkServer instance has been added.
- The cluster client that contains the Flink service has been installed, for example, in the **/opt/hadoopclient** directory.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

## Procedure

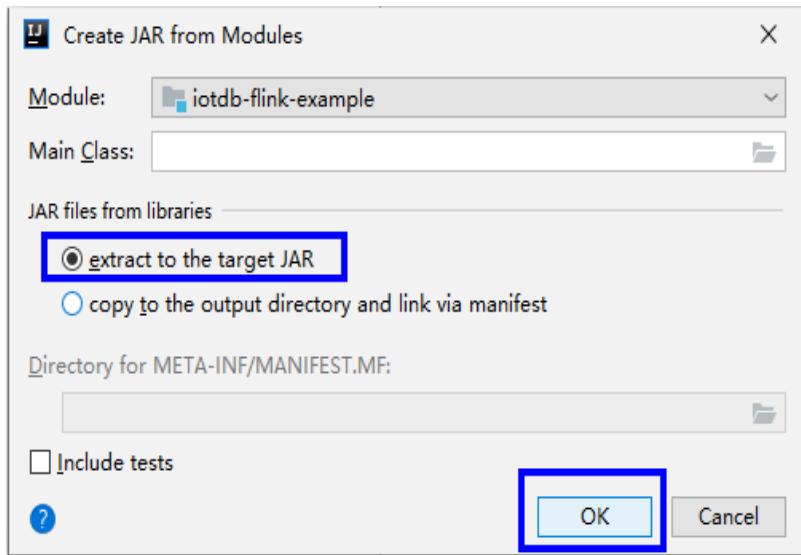
### Step 1 Build a JAR file.

- In IntelliJ IDEA, configure **Artifacts** of the project before generating a JAR file.
  - a. On the IDEA homepage, choose **File > Project Structures...** to go to the **Project Structure** page.
  - b. On the **Project Structure** page, select **Artifacts**, click **+**, and select **From modules with dependencies....**

Figure 1-189 Adding Artifacts



- c. Select **extract to the target JAR** and click **OK**.



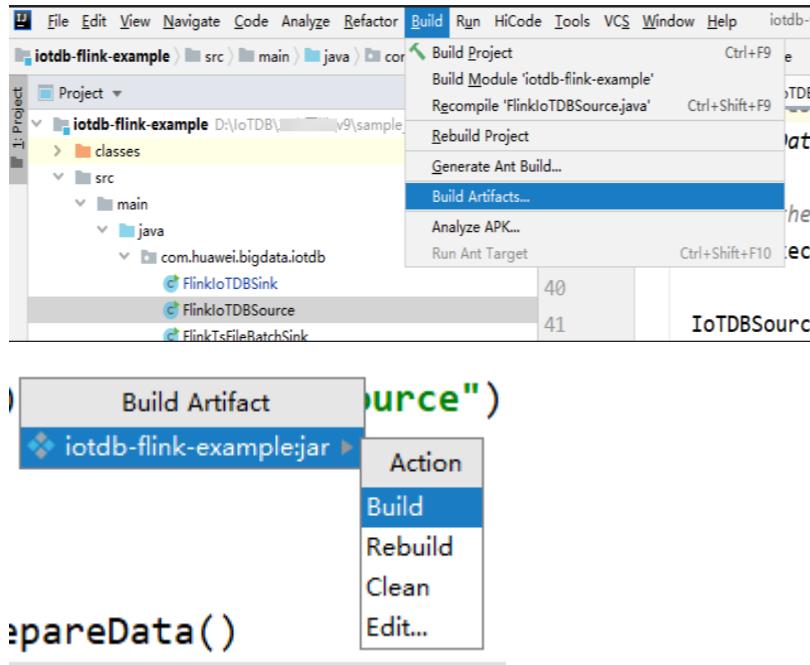
- d. Set the name, type, and output path of the JAR file based on the site requirements.

To avoid JAR file conflicts caused by unnecessary JAR files, you only need to load the following basic JAR files related to IoTDB:

- flink-iotdb-connector-\*
- flink-tsfile-connector-\*
- hdoop-tsfile-\*
- influxdb-thrift-\*
- iotdb-antlr-
- iotdb-session-\*
- iotdb-thrift-\*
- iotdb-thrift-commons-\*
- isession-\*
- libthrift-\*
- service-rpc-\*
- tsfile-\*

Click **OK**.

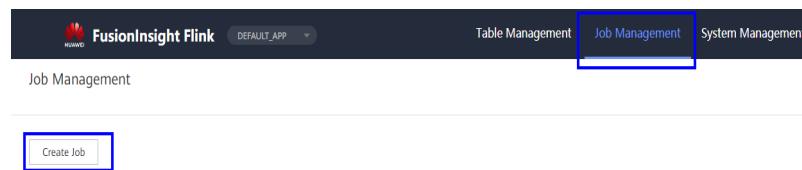
- e. On the IDEA home page, choose **Build > Build Artifacts....** On the **Build Artifact** page that is displayed, choose **Action > Build**.



- f. After the build is successful, the message "Build completed successfully" is displayed in the lower right corner, and the corresponding JAR file is generated in the **Output Directory** directory.

**Step 2** (Scenario 1) Run a Flink job on the Flink web UI.

1. Log in to FusionInsight Manager of the cluster as a user who has the FlinkServer web UI management permission, choose **Cluster > Services > Flink**, and click the hyperlink next to **Flink WebUI** on the dashboard page to go to the FlinkServer web UI.
2. On the FusionInsight Flink web UI, choose **Job Management > Create Job** to create a job.



3. Set **Type** to **Flink Jar**, enter the name of the job to be created, select a task type, and click **OK**.

Create Job

\*Type Flink SQL Flink Jar

\*Name iotdb\_flink

\*Task Type Stream job Batch Job

Description Enter the description.

OK Cancel

- Upload the JAR file generated in **Step 1**, set **Main Class** to **Specify**, enter the class to be executed in **Class Parameter**, and click **Submit**.

For example, set **Type** to **com.huawei.bigdata.iotdb.FlinkIoTDBSink** (development program that executes [FlinkIoTDBSink](#)) or **com.huawei.bigdata.iotdb.FlinkIoTDBSource** (development program that executes [FlinkIoTDBSource](#)).

FusionInsight Flink DEFAULT\_APP

Table Management Job Management System Management

administ

Job Management > iotdb\_flink > Back

Upload jar File Local jar File Select flink-example.jar flink-example.jar 2.08MB File uplo... Default Specify com.huawei.bigdata.iotdb.FlinkIoTDB

Configure Parameters Parallelism EnterParallelism JobManager Memory(MB) Enter the memory

Save Submit

### Step 3 (Scenario 2) Submit a Flink job on the Flink client in the Linux environment.

- Log in to the MRS client as a client installation user.
- Run the following command to initialize environment variables:  
`source /opt/hadoopclient/bigdata_env`
- If Kerberos authentication is enabled for the cluster, perform [Step 3.4](#) to [Step 3.11](#). If Kerberos authentication is disabled for the cluster, skip these steps.
- Prepare a user for submitting Flink jobs.  
For details, see [Preparing a Developer Account](#).

5. Log in to Manager using the created user, choose **System > Permission > User**. Locate the row that contains the new user and choose **More > Download Authentication Credential** in the **Operation** column.
6. Decompress the downloaded authentication credential package and copy the **user.keytab** file to the client node, for example, to the **/opt/hadoopclient/Flink/flink/conf** directory on the client node. If the client is installed on a node outside the cluster, copy the **krb5.conf** file to the **/etc/** directory on this node.
7. In security mode, append the service IP address of the node where the client is installed and floating IP address of Manager to the **jobmanager.web.allow-access-address** configuration item in the **/opt/hadoopclient/Flink/flink/conf/flink-conf.yaml** file. Use commas (,) to separate IP addresses.
8. Run the following commands to configure security authentication by adding the **keytab** path and username to the **/opt/hadoopclient/Flink/flink/conf/flink-conf.yaml** configuration file.  
**security.kerberos.login.keytab: <user.keytab file path>**  
**security.kerberos.login.principal: <Username>**

Example:

```
security.kerberos.login.keytab: /opt/hadoopclient/Flink/flink/conf/user.keytab  
security.kerberos.login.principal: test
```

9. In the **bin** directory of the Flink client, run the following command to perform security hardening. For details, see "Component Operation Guide" > "Using Flink" > "Security Hardening" > "Authentication and Encryption" in the *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

**sh generate\_keystore.sh**

After the script is executed, enter a password for submitting jobs. Then, the value of SSL in **/opt/hadoopclient/Flink/flink/conf/flink-conf.yaml** is automatically replaced.

#### NOTE

- After the operations in "Component Operation Guide" > "Using Flink" > "Security Hardening" > "Authentication and Encryption" in the *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)* are performed, the generated **flink.keystore**, **flink.truststore**, and **security.cookie** are automatically filled in the corresponding configuration items in the **flink-conf.yaml** file.
- The values of **security.ssl.key-password**, **security.ssl.keystore-password**, and **security.ssl.truststore-password** must be obtained using the Manager plaintext encryption API by running the following command:  
**curl -k -i -u <user name>:<password> -X POST -HContent-type:application/json -d '{"plainText":"<password>"}' 'https://x.x.x.x:28443/web/api/v2/tools/encrypt'**; in the preceding command, **<password>** must be the same as the password used for issuing the certificate, and **x.x.x.x** indicates the floating IP address of Manager in the cluster.

10. Configure paths for the **flink.keystore** and **flink.truststore** files.
  - Absolute path: After the script is executed, the **flink.keystore** and **flink.truststore** file paths are automatically set to absolute paths in the **flink-conf.yaml** file. In this case, you need to place the **flink.keystore** and **flink.truststore** files in the **conf** directory to the absolute paths of the Flink client and each Yarn node in the cluster, respectively.

- Relative path: Perform the following operations to set the **flink.keystore** and **flink.truststore** files to relative paths:
  - i. In the **/opt/hadoopclient/Flink/flink/conf/** directory, create a directory, for example, **ssl**.  
**cd /opt/hadoopclient/Flink/flink/conf**  
**mkdir ssl**
  - ii. Move the **flink.keystore** and **flink.truststore** files to the new folder.  
**mv flink.keystore flink.truststore ssl/**
  - iii. Change the values of the following parameters to relative paths in the **flink-conf.yaml** file:  
security.ssl.keystore: **ssl/flink.keystore**  
security.ssl.truststore: **ssl/flink.truststore**

11. Add the IP addresses of the nodes where the clients are located to the following configuration items in the **flink-conf.yaml** file. Use commas (,) to separate IP addresses.  
web.access-control-allow-origin: **xx.xx.xxx.xxx**  
jobmanager.web.allow-access-address: **xx.xx.xxx.xxx**
12. Upload the JAR file generated in **Step 1** to the Flink client node, for example, **/opt/hadoopclient/Flink/flink**, and submit the job.

---

#### NOTICE

To submit or run jobs on Flink, the user must have the following permissions:

- If Ranger authentication is enabled, the current user must belong to the **hadoop** group or the user has been granted the **/flink** read and write permissions in Ranger.
- If Ranger authentication is disabled, the current user must belong to the **hadoop** group.
- If the **flink.keystore** and **flink.truststore** files are stored in the absolute path:  
Run the following commands to start a session and submit a job in the session: **com.huawei.bigdata.iotdb.FlinkIoTDBSink** is the application in **FlinkIoTDBSink**.  
**yarn-session.sh -nm "session-name"**  
**flink run --class com.huawei.bigdata.iotdb.FlinkIoTDBSink /opt/hadoopclient/Flink/flink/flink-example.jar**
- If the **flink.keystore** and **flink.truststore** files are stored in the relative path:  
In the same directory of SSL, run the following commands to start a session and submit a job in the session. The SSL directory is a relative path. For example, if the SSL directory is **/opt/hadoopclient/Flink/flink/conf/**, then run the following commands in this directory.  
**com.huawei.bigdata.iotdb.FlinkIoTDBSink** is the application in **FlinkIoTDBSink**.  
**yarn-session.sh -t ssl/ -nm "session-name"**

```
flink run --class com.huawei.bigdata.iotdb.FlinkIoTDBSink /opt/
hadoopclient/Flink/flink-example.jar
```

---End

### 1.14.4.3.2 Viewing Commissioning Results

- Check whether the job is executed.

- Using the Flink web UI

If a success message is returned on the Flink web UI, the execution is successful. You can choose **More > Job Monitoring** in the **Operation** column to view detailed logs.



- Using the Flink client

Log in to FusionInsight Manager as a running user, go to the native page of the Yarn service, find the application of the corresponding job, and click the application name to go to the job details page.

- If the job is not complete, click **Tracking URL** to go to the native Flink service page and view the job running information.
- If the job submitted in a session has been completed, you can click **Tracking URL** to log in to the native Flink service page to view job information.

The screenshot shows the Yarn service native page with the following details:

- Application Overview:**
  - User: administ
  - Name: session-name
  - Application Type: Apache Flink
  - Application ID: application\_1627029644621\_0039
  - Yarn Application State: RUNNING (AM has registered with RM and started running)
  - Queue: default
  - Final Status Reported by AM: Application has not completed yet.
  - Started: Sat Jul 31 10:28:43 +0800 2021
  - Elapsed: 48m, 57m, 29s
  - Tracking URL: ApplicationMaster
  - Log Aggregate Status: NOT START
  - Application Timeout (Remaining Time): Unlimited
  - Diagnostics:
  - Unmanaged Application: false
  - Application Node Label expression: <Not set>
  - AM container Node Label expression: <DEFAULT\_PARTITION>
- Application Metrics:**
  - Total Resource Preempted: <memory0, vCores0>
  - Total Number of Non-AM Containers Preempted: 0
  - Total Number of AM Containers Preempted: 0
  - Resource Preempted from Current Attempt: <memory0, vCores0>
  - Number of Non-AM Containers Preempted from Current Attempt: 0
  - Aggregate Resource Allocation: 362854138 MB-seconds, 354346 vcore-seconds, 0 yarn.io/gpu-seconds
  - Aggregate Preempted Resource Allocation: 0 MB-seconds, 0 vcore-seconds

- Verify the job execution result.

- FlinkIoTDBSink execution result:

- Run the following command on the IoTDB client and check whether the data has been written from Flink to IoTDB:

```
select * from root.sg.d1
```

```
IoTDB@...:22260> select * from root.sg.d1
+-----+
| Time |      root.sg.d1.s1 |
+-----+
2022-09-24T16:19:13.076+08:00	0.48050481096976925
2022-09-24T16:19:15.404+08:00	0.5917976517293774
2022-09-24T16:19:16.404+08:00	0.735936612423717
2022-09-24T16:19:23.869+08:00	0.8386843031417548
2022-09-24T16:19:24.869+08:00	0.6463236193893684
2022-09-24T16:19:25.870+08:00	0.5954449837992902
2022-09-24T16:19:26.870+08:00	0.5945272607886377
2022-09-24T16:19:27.871+08:00	0.4929779102387244
2022-09-24T16:19:28.871+08:00	0.34395251562673457
2022-09-24T16:19:29.872+08:00	0.6177739062963853
2022-09-24T16:19:30.872+08:00	0.7498794851618589
2022-09-24T16:19:31.873+08:00	0.3148741420873038
2022-09-24T16:19:32.873+08:00	0.6684900825420255
2022-09-24T16:19:33.874+08:00	0.45938194501820595
2022-09-24T16:19:34.874+08:00	0.24293321093096187
2022-09-24T16:19:35.875+08:00	0.7094903559027376
2022-09-24T16:19:36.875+08:00	0.934997431381603
2022-09-24T16:19:37.876+08:00	0.9151513474234568
2022-09-24T16:19:38.876+08:00	0.5625493998872735
2022-09-24T16:19:39.876+08:00	0.991484539263849
2022-09-24T16:19:40.877+08:00	0.8925889658869346
2022-09-24T16:19:41.878+08:00	0.8873826211498665
2022-09-24T16:19:42.878+08:00	0.44325532800273826
2022-09-24T16:19:43.879+08:00	0.48251227562595245
2022-09-24T16:19:44.879+08:00	0.1367102977099991
2022-09-24T16:19:45.880+08:00	0.06494944420823234
2022-09-24T16:19:46.880+08:00	0.4026376592496197
2022-09-24T16:19:47.881+08:00	0.6740267414029771
2022-09-24T16:19:48.881+08:00	0.7480089974928386
```

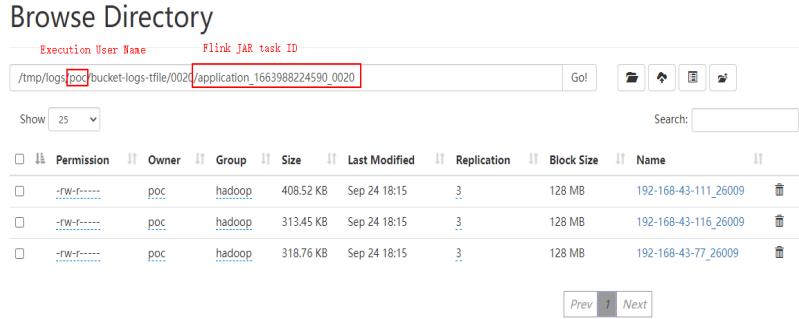
- FlinkIoTDBSource execution result:

- 1) Log in to FusionInsight Manager as a running user and choose **Cluster > Services > HDFS**. Click the hyperlink next to **NameNode WebUI** to access the HDFS web UI.
- 2) Choose **Utilities > Browse the file system**.

The screenshot shows the 'Startup Progress' section of the HDFS Utilities page. At the top, there's a green navigation bar with 'Utilities' and a dropdown menu. The dropdown menu has an option 'Browse the file system' highlighted with a red box and a red arrow pointing to it. Below the dropdown, the 'Startup Progress' section displays the progress of various startup phases. The first phase listed is 'Loading fsimage /srv/BigData/namenode/current/fsimage\_00000000000000000000 399 B', which is at 100% completion and took 1 sec.

| Phase  | Completion | Elapsed Time |
|--|------------|--------------|
| Loading fsimage /srv/BigData/namenode/current/fsimage_00000000000000000000 399 B | 100%       | 1 sec        |
| erasure coding policies (0/0)  | 100%       |              |
| inodes (1/1)   | 100%       |              |
| delegation tokens (0/0)  | 100%       |              |
| cache pools (0/0)  | 100%       |              |
| Loading edits  | 100%       | 0 sec        |
| Saving checkpoint  | 100%       | 0 sec        |
| Safe mode  | 100%       | 0 sec        |

- 3) Go to the **/tmp/logs/Execution username/bucket-logs-tfile/Task ID/Flink task ID** directory and download all files in the directory to the local PC.



- 4) Search for **root.sg.d1** in the files downloaded in [2.3](#)). If the following information is displayed, data is successfully read from IoTDB.

#### 1.14.4.4 Commissioning Kafka Applications on Linux

#### **1.14.4.4.1 Compiling and Running Applications**

# Scenario

After code development is complete, you can run an IoTDB-Kafka sample application in the Linux environment.

## Prerequisites

- You have installed the IoTDB and Kafka clients.
  - If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

## Procedure

## **Step 1 Export a JAR file.**

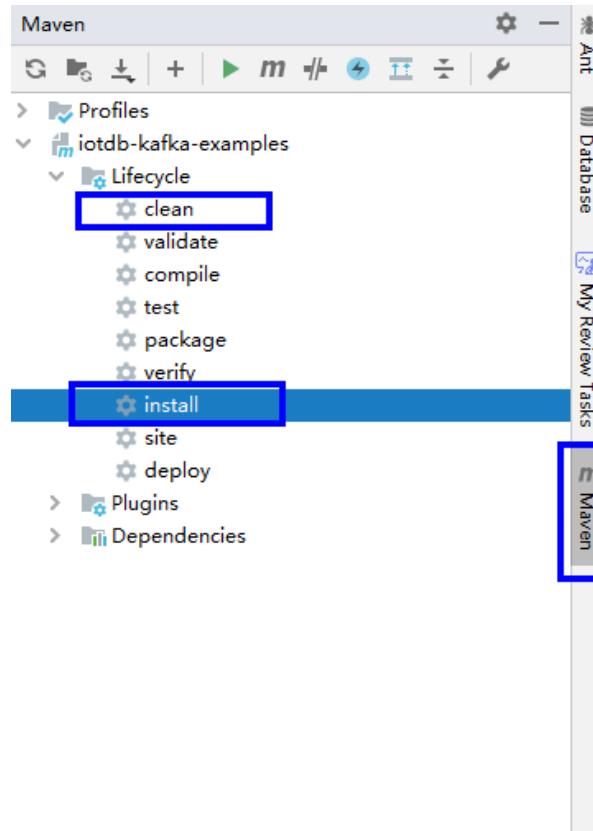
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

**Figure 1-190** Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

**Figure 1-191** Entering **mvn clean install** in the IDEA Terminal text box

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.594 s  
[INFO] Finished at: 2022-01-15T17:58:02+08:00  
[INFO] -----  
D:\IoTDB\二次开发\v18\sample_project\src\iotdb-examples\iotdb-kafka-example>mvn clean install
```

After the build is completed, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

## Step 2 Prepare the dependent JAR file.

- Go to the client installation directory, create the **lib** directory, and import the JAR package generated in **Step 1** to the **lib** directory, for example, **/opt/hadoopclient/lib**.
- Go to the Kafka client and copy the JAR package on which Kafka depends to the **lib** directory in **Step 2.1**. The following is an example directory:

```
cp /opt/hadoopclient/Kafka/kafka/libs/*.jar /opt/hadoopclient/lib
```

3. Go to the IoTDB client and copy the JAR package on which IoTDB depends to the **lib** directory in **Step 2.1**. The following is an example directory:  

```
cp /opt/hadoopclient/IoTDB/iotdb/lib/*.jar/opt/hadoopclient/lib
```
4. Copy all files in the **src/main/resources** directory of the IntelliJ IDEA project to the **src/main/resources** directory at the same level as the **lib** folder, that is, **/opt/hadoopclient/src/main/resources**.

**Step 3** Go to the **/opt/hadoopclient** directory and ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependency library folder. In addition, ensure that JDK has been installed and Java environment variables have been set. Then, run the following command to run the sample project:

```
java -cp /opt/hadoopclient/lib/*:/opt/hadoopclient/src/main/resources  
com.huawei.bigdata.iotdb.KafkaConsumerMultThread
```

----End

#### 1.14.4.4.2 Viewing Commissioning Results

If the application running is successful, the following information is displayed.

```
[2022-01-17 14:43:57,511] INFO Consumer Thread-1 partitions:1 record:sensor_89,1642401919971,1.000000  
offsets:6951 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,511] INFO Consumer Thread-0 partitions:0 record:sensor_97,1642401919971,1.000000  
offsets:7129 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,512] INFO Consumer Thread-0 partitions:0 record:sensor_98,1642401919971,1.000000  
offsets:7130 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,512] INFO Consumer Thread-1 partitions:1 record:sensor_92,1642401919971,1.000000  
offsets:6952 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,513] INFO Consumer Thread-0 partitions:0 record:sensor_99,1642401919971,1.000000  
offsets:7131 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,513] INFO Consumer Thread-1 partitions:1 record:sensor_93,1642401919971,1.000000  
offsets:6953 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,513] INFO Consumer Thread-1 partitions:1 record:sensor_95,1642401919971,1.000000  
offsets:6954 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
```

#### 1.14.4.5 Using a UDF

##### 1.14.4.5.1 Registering a UDF

1. Build a JAR file.

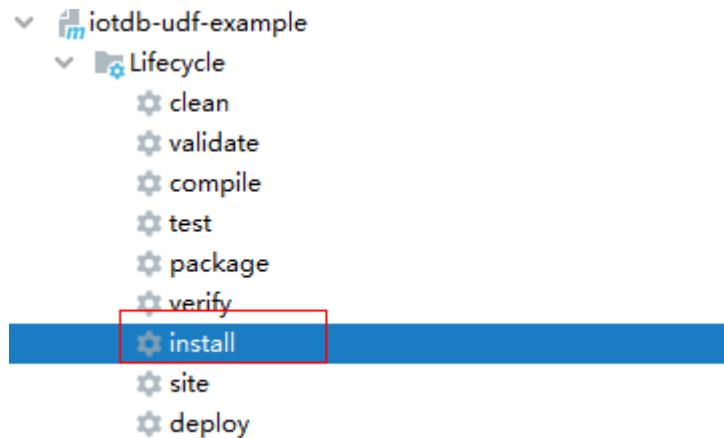
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 1-192 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 1-193 Building result after mvn clean install is entered

```
[INFO] --- maven-install-plugin:2.4:install (default-install) @ iotdb-udf-example ---
[INFO] Installing D:\code\lib\sample_project\src\iotdb-examples\iotdb-udf-example\target\iotdb-udf-example-0.12.0-hw-ei-312005.jar to D:\repo1\com\huawei\mrs\iotdb-udf-example\0.12.0-hw-ei-312005\iot
[INFO] Installing D:\code\lib\sample_project\src\iotdb-examples\iotdb-udf-example\pom.xml to D:\repo1\com\huawei\mrs\iotdb-udf-example\0.12.0-hw-ei-312005\iotdb-udf-example-0.12.0-hw-ei-312005.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:19 min
[INFO] Finished at: 2022-02-14T15:51:35+08:00
[INFO] -----
```

After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

2. Import the dependent JAR file.

Log in to the node where IoTDBServer is located as user **root**, run **su - omm** to switch to user **omm**, and import the JAR file generated in 1 to the **\$BIGDATA\_HOME/FusionInsight\_IoTDB\_\*/install/FusionInsight-IoTDB-\*/iotdb/ext/udf** directory.

**NOTICE**

During cluster deployment, ensure that a corresponding JAR package exists in the UDF JAR package path of each IoTDBServer node. You can modify IoTDB configuration **udf\_root\_dir** to specify the root path for the UDF to load JAR files.

3. Run the following SQL statement to register the UDF:

**CREATE FUNCTION <UDF-NAME> AS '<UDF-CLASS-FULL-PATHNAME>'**

For example, you can run the following statement to register a UDF named **example**:

```
CREATE FUNCTION example AS 'com.huawei.bigdata.iotdb.UDTFExample'
```

### 1.14.4.5.2 Querying a UDF

#### Basic SQL Syntax Supported

- SLIMIT / SOFFSET
- LIMIT / OFFSET
- NON ALIGN
- Queries with value filters
- Queries with time filters



Currently, aligned time series are not supported in UDF queries. An error message is reported if you use UDF queries with aligned time series selected.

#### Queries with an Asterisk (\*)

Assume that there are two time series (`root.sg.d1.s1` and `root.sg.d1.s2`).

- Execute the **SELECT example(s1) from root.sg.d1** statement.  
The result set contains the result of **example(root.sg.d1.s1)**.
- Execute the **SELECT example(s1, s2) from root.sg.d1** statement.  
The result set contains the result of **example(root.sg.d1.s1, root.sg.d1.s2)**.

#### Queries with key-value pair attributes in UDF parameters

You can pass any number of key-value pair parameters to the UDF when constructing a UDF query. The key and value in a key-value pair must be enclosed in single or double quotation marks.

---

#### NOTICE

The key-value pair parameters can be passed in only after the time series have been passed in.

---

For example:

```
SELECT example(s1, 'key1'='value1', 'key2'='value2'), example(*, 'key3'='value3') FROM root.sg.d1;  
SELECT example(s1, s2, 'key1'='value1', 'key2'='value2') FROM root.sg.d1;
```

#### Showing All Registered UDFs

Run the following SQL statement on the IoTDB client to view the registered UDFs:

```
SHOW FUNCTIONS
```

### 1.14.4.5.3 Deregistering a UDF

#### Syntax

```
DROP FUNCTION <UDF-NAME>
```

## Example

Run the following command on the IoTDB client to deregister the UDF named **example**:

```
DROP FUNCTION example
```

## 1.14.5 More Information

### 1.14.5.1 Common APIs

#### 1.14.5.1.1 Java API

IoTDB provides a connection pool (SessionPool) for native APIs. When using the APIs, you only need to specify the pool size to obtain connections from the pool. If you cannot get a connection in 60 seconds, a warning log will be printed, but the program continues to wait.

When a connection is used, it automatically returns to the pool and waits to be used next time. When a connection is damaged, it is deleted from the pool and a new connection is created to perform user operations again.

For query operations:

1. When SessionPool is used for query, the result set is SessionDataSetWrapper, the encapsulation class of SessionDataSet.
2. If a query result set is not traversed and you do not want to continue traversing, you need to call **closeResultSet** to release the connection.
3. If an exception is reported when you traverse a query result set, you need to call **closeResultSet** to release the connection.
4. You can call the **getColumnName()** method of SessionDataSetWrapper to obtain the column names in the result set.

**Table 1-131** Sessions APIs and corresponding parameters

| Method   | Description            |
|--|------------------------|
| <ul style="list-style-type: none"><li>• Session(String host, int rpcPort)</li><li>• Session(String host, String rpcPort, String username, String password)</li><li>• Session(String host, int rpcPort, String username, String password)</li></ul> | Initializes a session. |
| Session.open()   | Opens a session.       |
| Session.close()  | Closes a session.      |
| void createDatabase(String database)   | Creates a database.    |

| Method   | Description   |
|--|---|
| <ul style="list-style-type: none"><li>• void deleteDatabase(String storageGroup)</li><li>• void deleteDatabases(List&lt;String&gt; storageGroups)</li></ul>  | Deletes one or more databases.  |
| <ul style="list-style-type: none"><li>• void createTimeseries(String path, TSDataType dataType, TSEncoding encoding, CompressionType compressor, Map&lt;String, String&gt; props, Map&lt;String, String&gt; tags, Map&lt;String, String&gt; attributes, String measurementAlias)</li><li>• void createMultiTimeseries(List&lt;String&gt; paths, List&lt;TSDataType&gt; dataTypes, List&lt;TSEncoding&gt; encodings, List&lt;CompressionType&gt; compressors, List&lt;Map&lt;String, String&gt;&gt; propsList, List&lt;Map&lt;String, String&gt;&gt; tagsList, List&lt;Map&lt;String, String&gt;&gt; attributesList, List&lt;String&gt; measurementAliasList)</li></ul> | Creates one or more time series.  |
| <ul style="list-style-type: none"><li>• void deleteTimeseries(String path)</li><li>• void deleteTimeseries(List&lt;String&gt; paths)</li></ul>   | Deletes one or more time series.  |
| <ul style="list-style-type: none"><li>• void deleteData(String path, long time)</li><li>• void deleteData(List&lt;String&gt; paths, long time)</li></ul>   | Deletes the data of one or more time series before or at a specific time point.   |
| void insertRecord(String deviceId, long time, List<String> measurements, List<String> values)  | Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. Servers need to perform type inference, which may take extra time. |
| void insertTablet(Tablet tablet)   | Inserts a tablet, which contains multiple rows of non-empty data blocks. The columns in each row are the same.  |
| void insertTablets(Map<String, Tablet> tablet)   | Inserts multiple tablets.   |
| void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<String>> valuesList)   | Inserts multiple records. Servers need to perform type inference, which may take extra time.  |

| Method  | Description  |
|---|--|
| void insertRecord(String deviceld, long time, List<String> measurements, List<TSDDataType> types, List<Object> values)  | Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. With the data type information, servers do not need to perform type inference, improving the performance. |
| void insertRecords(List<String> devicelds, List<Long> times, List<List<String>> measurementsList, List<List<TSDDataType>> typesList, List<List<Object>> valuesList)     | Inserts multiple records. With the data type information, servers do not need to perform type inference, improving the performance.  |
| submitApplication(SubmitApplicationRequest request)   | Used by the client to submit a new application to ResourceManager.   |
| void insertRecordsOfOneDevice(String deviceld, List<Long> times, List<List<String>> measurementsList, List<List<TSDDataType>> typesList, List<List<Object>> valuesList) | Inserts multiple records of the same device.   |
| SessionDataSet executeRawDataQuery(List<String> paths, long startTime, long endTime)  | Queries raw data. The interval includes the start time but does not include the end time.  |
| SessionDataSet executeQueryStatement(String sql)  | Executes query statements.   |
| void executeNonQueryStatement(String sql)   | Executes non-query statements.   |

**Table 1-132** Test APIs

| Method   | Description  |
|--|--|
| <ul style="list-style-type: none"><li>• void testInsertRecords(List&lt;String&gt; devicelds, List&lt;Long&gt; times, List&lt;List&lt;String&gt;&gt; measurementsList, List&lt;List&lt;String&gt;&gt; valuesList)</li><li>• void testInsertRecords(List&lt;String&gt; devicelds, List&lt;Long&gt; times, List&lt;List&lt;String&gt;&gt; measurementsList, List&lt;List&lt;TSDDataType&gt;&gt; typesList, List&lt;List&lt;Object&gt;&gt; valuesList)</li></ul> | Tests testInsertRecords. A response is returned immediately after data is transmitted to the server. No data is written. |

| Method  | Description   |
|---|---|
| <ul style="list-style-type: none"><li>• void testInsertRecord(String deviceld, long time, List&lt;String&gt; measurements, List&lt;String&gt; values)</li><li>• void testInsertRecord(String deviceld, long time, List&lt;String&gt; measurements, List&lt;TSDataType&gt; types, List&lt;Object&gt; values)</li></ul> | Tests insertRecord. A response is returned immediately after data is transmitted to the server. No data is written. |
| void testInsertTablet(Tablet tablet)  | Tests insertTablet. A response is returned immediately after data is transmitted to the server. No data is written. |

## 1.15 Kafka Development Guide

### 1.15.1 Overview

#### 1.15.1.1 Development Environment Preparation

##### Kafka Introduction

Kafka is a distributed message release and subscription system. With features similar to JMS, Kafka processes active streaming data.

Kafka is applicable to message queuing, behavior tracing, operation & maintenance (O&M) data monitoring, log collection, streaming processing, event tracing, and log persistence.

Kafka features:

- High throughput
- Message persistence to disks
- Scalable distributed system
- Fault-tolerant
- Support for online and offline scenarios

##### Interface Type Introduction

APIs provided by Kafka can be divided into two types: Producer API and Consumer API. Both the types of APIs contain Java API. For details, see section [Java API](#).

#### 1.15.1.2 Common Concepts

- **Topic**

A same type of messages maintained by Kafka.

- **Partition**  
One topic can be divided into multiple partitions, and each partition corresponds to an appendant and log file whose sequence is fixed.
- **Producer**  
Role in a Kafka topic to which messages are sent.
- **Consumer**  
Role that obtains messages from Kafka topics.
- **Broker**  
A node server in a Kafka cluster.
- **keytab file**  
A key file for storing user information. Applications use the key file for API authentication in the cluster.

### 1.15.1.3 Development Process

Kafka client roles include Producer and Consumer, which share the same application development process.

[Figure 1-194](#) and [Table 1-133](#) show each phase of the development process.

Figure 1-194 Kafka client application development process

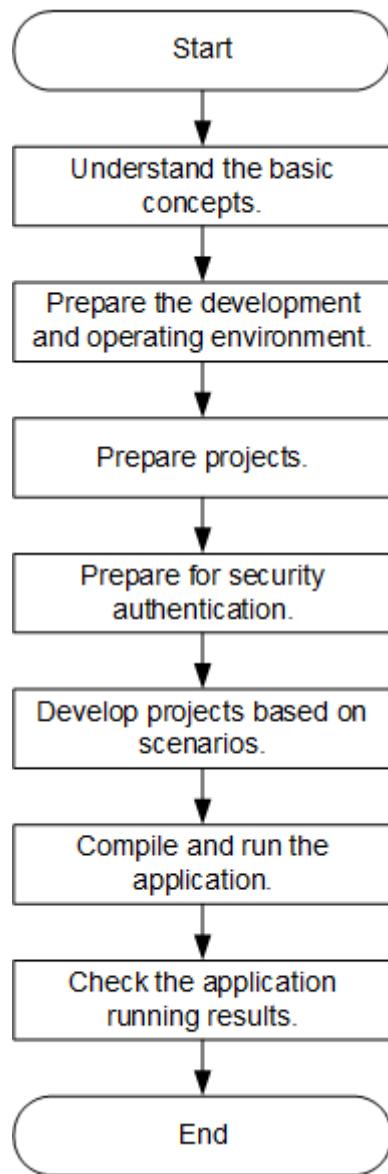


Table 1-133 Kafka client application development process description

| Phase                          | Description   | Reference Document              |
|--------------------------------|---|---------------------------------|
| Understand the basic concepts. | Before developing an application, learn basic concepts of Kafka, and determine whether the desired role is Producer or Consumer based on the actual scenario. | <a href="#">Common Concepts</a> |

| Phase  | Description   | Reference Document  |
|--|---|---|
| Prepare the development and operating environment. | The Java language is recommended for the development of Java applications. The IntelliJ IDEA tool can be used.<br><br>The Kafka running environment is the Kafka client. Install and configure the client according to the guide. | <a href="#">Preparing for Development and Operating Environment</a> |
| Prepare projects                                   | Kafka provides program samples for different scenarios. You can import the program samples for learning.  | <a href="#">Configuring and Importing Sample Projects</a>           |
| Prepare for security authentication.               | If you use a security cluster, you need to perform security authentication.   | <a href="#">Preparing for Security Authentication</a>               |
| Develop projects based on scenarios.               | Producer and Consumer API usage samples are provided and cover API, and multi-thread usage scenarios, helping you quickly know Kafka interfaces well.   | <a href="#">Developing an Application</a>                           |
| Compile and run the application.                   | Compile the developed application and submit it for running.  | <a href="#">Application Commissioning</a>                           |
| View application running results.                  | Program running results will be written to and printed on the console. You can use a Linux client to consume topic data to check whether data has been successfully written.  | <a href="#">Application Commissioning</a>                           |

## 1.15.2 Environment Preparation

### 1.15.2.1 Preparing for Development and Operating Environment

#### Preparing Development Environment

[Table 1-134](#) describes the environment required for secondary development.

**Table 1-134** Development environment

| Item   | Description   |
|--|---|
| OS   | <ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>   |
| IntelliJ IDEA installation and configuration | <p>It is the basic configuration for the development environment. JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li></ul> |

| Item                          | Description   |
|-------------------------------|---|
| Installing JDK                | <p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <code>com.ibm.jsse2.overrideDefaultTLS</code> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul> |
| Maven installation            | Basic configuration of the development environment for project management throughout the lifecycle of software development.   |
| Developer account preparation | See <a href="#">Preparing a Developer Account</a> for configuration.  |
| 7-zip                         | It is a tool used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.   |

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) and choose **Homepage > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the

client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Kafka\config**. Obtain the Kafka-related configuration file.

**Table 1-135** describes the main configuration files.

**Table 1-135** Configuration file

| Document Name       | Function  |
|---------------------|---|
| server.properties   | Configures the Kafka server.  |
| client.properties   | Configures the Kafka client.  |
| producer.properties | Configures producer of kafka parameters.                              |
| consumer.properties | Configures consumer of kafka parameters.                              |
| user.keytab         | Provides user information for Kerberos security authentication.       |
| krb5.conf           | Kerberos server configuration information                             |
| kafkaSecurityMode   | Indicates whether the security mode is enabled for the Kafka service. |

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

#### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. **Accessing FusionInsight Manager of an MRS Cluster.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all JAR package in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Kafka/install\_files/kafka/libs** directory to the **lib** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/Kafka/kafka/libs**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Kafka/install_files/kafka/libs/* root@IP address of the client node:/opt/hadoopclient/Kafka/kafka/libs
```

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

### 1.15.2.2 Configuring and Importing Sample Projects

**Step 1** Obtain the sample project folder **kafka-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

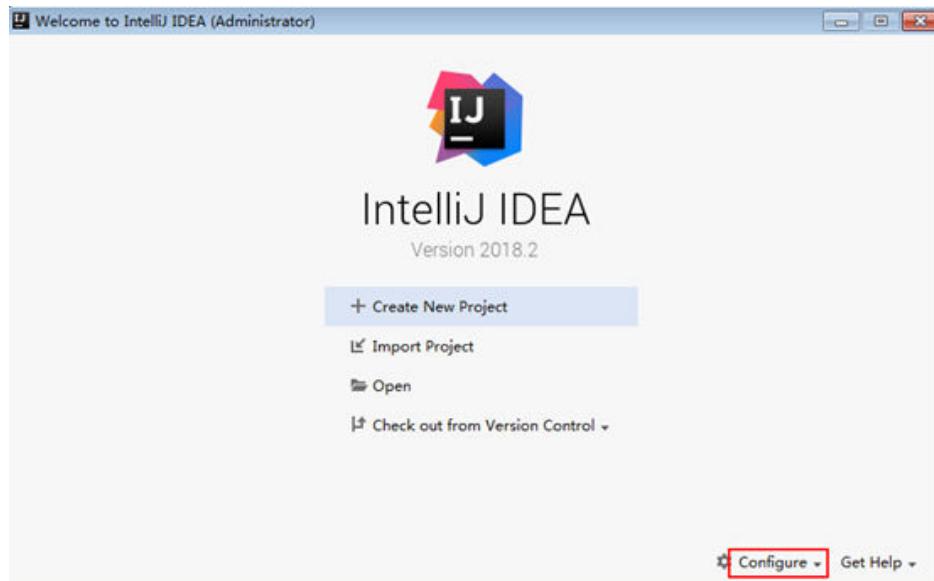
**Step 2** Configure the **krb5.conf** and **user.keytab** files.

Copy the **krb5.conf** and **keytab** files obtained in [Preparing a Developer Account](#) section and all the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the **kafka-examples\src\main\resources** directory of the sample project.

**Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.

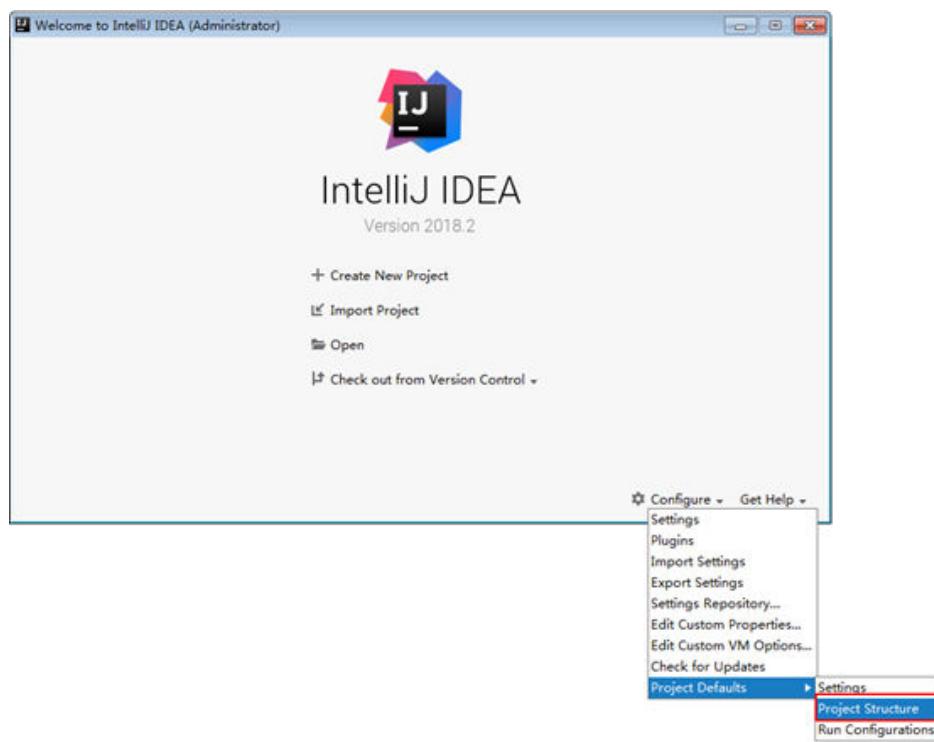
1. Open IntelliJ IDEA and click **Configure**.

Figure 1-195 Quick Start



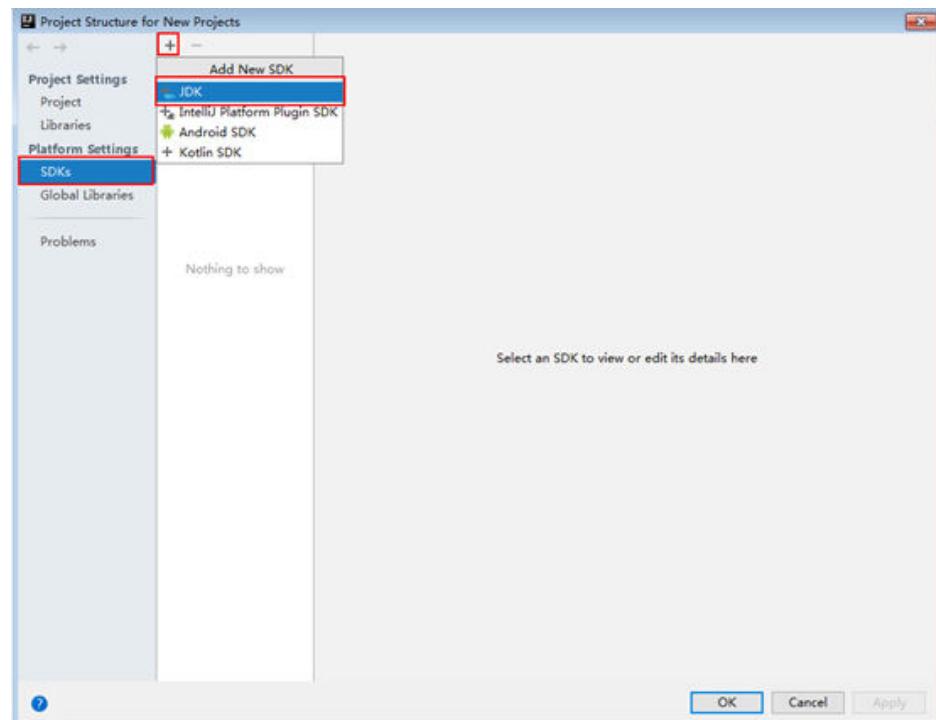
2. Choose **Project Defaults > Project Structure**.

Figure 1-196 Configure



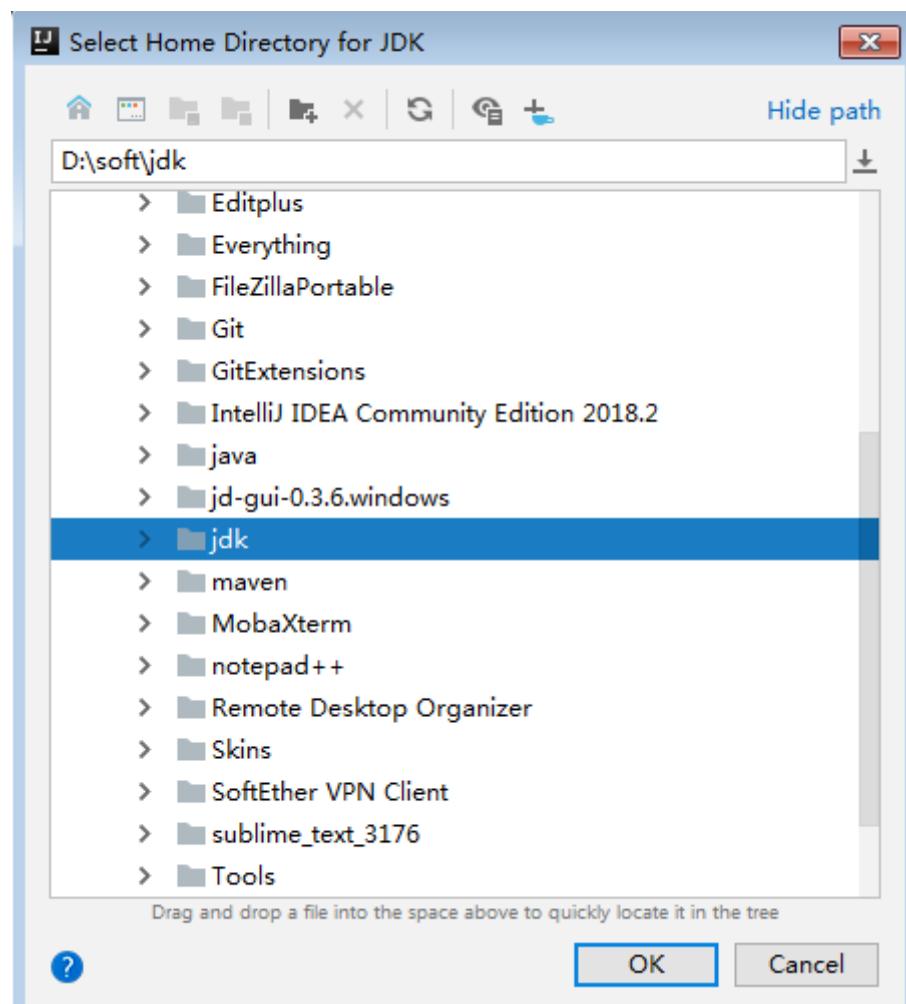
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 1-197 Project Structure for New Projects



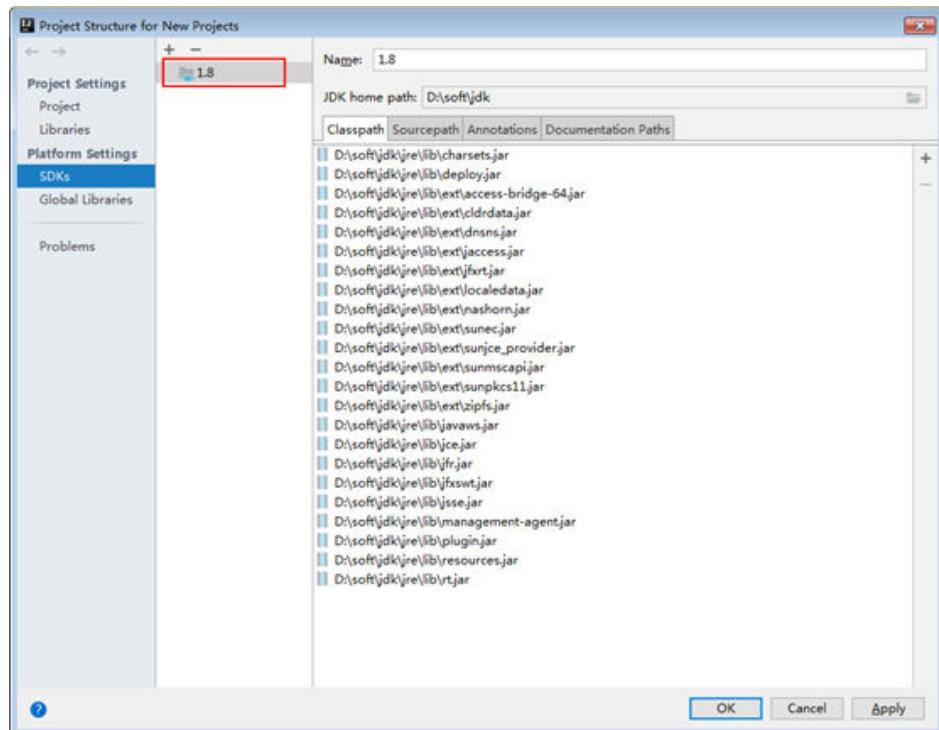
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 1-198 Select Home Directory for JDK



5. Click OK.

Figure 1-199 Completing the JDK configuration



**Step 4** Import the sample project into the IntelliJ IDEA development environment.

1. Choose **Open**.

The dialog box for browsing directories is displayed.

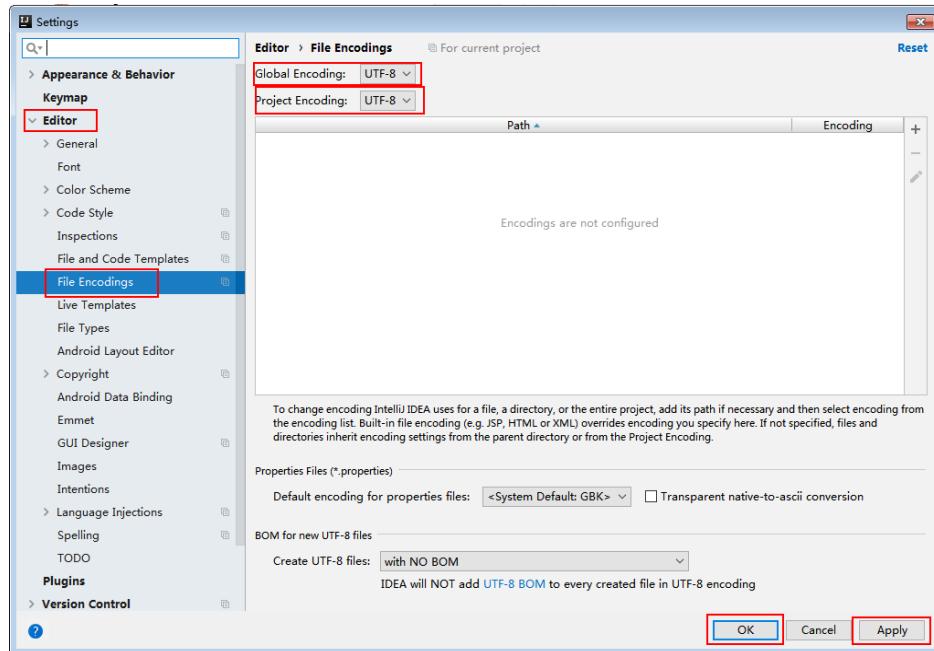
2. Select the sample project folder and click **OK**.

**Step 5** Set the IntelliJ IDEA text file coding format to prevent invalid characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.

The **Settings** window is displayed.

2. Choose **Editor > File Encodings** in the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the parameter to **UTF-8**, click **Apply**, and then click **OK**, as shown in [Figure 1-200](#).

**Figure 1-200** Setting the IntelliJ IDEA coding format

----End

### 1.15.2.3 Preparing for Security Authentication

#### 1.15.2.3.1 Sasl Kerberos authentication

In a secure cluster environment, components must perform mutual authentication before communicating with each other to ensure communication security. Kafka application development requires Kafka, ZooKeeper and Kerberos security authentication. To perform the security authentication, a **jaas** file needs to be generated, and related environment variables need to be configured. **LoginUtil** related interfaces are provided for the configuration. As shown in the following code sample, only the machine-machine user account applied by the user and the keytab file name need to be configured. For details about the code, reference may be made to the **LoginUtil** class in **com.huawei.bigdata.kafka.example.security** of the sample project.

Code sample:

```
/**  
 * keytab file name of the machine-machine account that a user applies for  
 */  
private static final String USER_KEYTAB_FILE = "keytab file name of the machine-machine account that a user applies for";  
  
/**  
 * Machine-machine account that a user applies for  
 */  
private static final String USER_PRINCIPAL = "Machine-machine account that a user applies for ";  
  
public static void securityPrepare() throws IOException  
{  
    String filePath = System.getProperty("user.dir") + File.separator + "src" + File.separator + "main" +  
    File.separator + "resources" + File.separator;  
    String krbFile = filePath + "krb5.conf";  
    String userKeyTableFile = filePath + USER_KEYTAB_FILE;
```

```
//Replace separators in Windows.  
userKeyTableFile = userKeyTableFile.replace("\\", "\\\\";  
krbFile = krbFile.replace("\\", "\\\\";  
  
LoginUtil.setKrb5Config(krbFile);  
LoginUtil.setZookeeperServerPrincipal("zookeeper/hadoop.<system domain name>");  
LoginUtil.setJaasFile(USER_PRINCIPAL, userKeyTableFile);  
}
```

#### NOTE

You can log in to the FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**, which is the current system domain name.

### 1.15.2.3.2 SASL/PLAINTEXT Authentication

#### Scenario

Kafka supports SASL/PLAINTEXT authentication for clusters in security mode.

#### Configuring SASL/PLAINTEXT Authentication on the Kafka Server

- Step 1 Log in to FusionInsight Manager.
  - Step 2 Choose **Cluster > Services > Kafka** and choose **Configurations > All Configurations**. Search for **sasl.enabled.mechanisms**, change the value to **GSSAPI,PLAIN**, and click **Save**.
  - Step 3 Click **Dashboard**, click **More**, and select **Restart Service** to make the configuration take effect.
- End

#### Configuring SASL/PLAINTEXT Authentication on the Kafka Client

You only need to configure dynamic **jaas.conf** and set related authentication attributes on the Kafka client. For example, only set the username and password of the human-machine account that you have applied for in the following sample code. For details, see the authentication sample code in **Producer** of the **com.huawei.bigdata.kafka.example.security** package.

```
public static Properties initProperties() {  
    .....  
    props.put("sasl.mechanism", "PLAIN");  
    props.put("sasl.jaas.config", "org.apache.kafka.common.security.plain.PlainLoginModule required  
username=manager_user password=Password");  
}
```

#### NOTE

- *manager\_user* is a human-machine user created on FusionInsight Manager and must have the production and consumption permissions for the topic that is being used.
- *Password* is the password of *manager\_user*.
  - If the open-source **kafka-client** JAR package is used, the special characters in the password can only be the dollar sign (\$).
  - If the MRS **kafka-client** JAR package is used, the special characters in the password are those supported by FusionInsight Manager (for example, ~`!?,.;-\_()'[]/<>@#\$%^&\*+|=).

## 1.15.3 Developing an Application

### 1.15.3.1 Typical Scenario Description

#### Scenario

Kafka is a distributed message system, in which messages can be publicized or subscribed. A Producer is to be developed to send a message to a topic of a Kafka cluster every second, and a Consumer is to be implemented to ensure that the topic is subscribed and that messages of the topic are consumed in real time.

#### Development Idea

1. Use a Linux client to create a topic. For details, see [Shell](#).
2. Develop a Producer to produce data to the topic.
3. Develop a Consumer to consume the data of the topic.

#### Suggestions on Performance Tuning

1. Create a topic and plan its partitions based on service requirements. The number of partitions limits the number of concurrent consumers.
2. The key value of a message must be variable to ensure even message distribution.
3. Consumers are advised to proactively submit the offset to avoid repeated consumption.
4. For details about other tuning suggestions, see section "Kafka" in MapReduce Service (MRS) 3.3.1-LTS Performance Tuning Guide (for Huawei Cloud Stack 8.3.1).

### 1.15.3.2 Typical Scenario Sample Code Description

#### 1.15.3.2.1 Producer API Usage Sample

#### Function Description

The following code snippet belongs to the **run** method of the **com.huawei.bigdata.kafka.example.Producer** class. It is used by the Producer APIs to produce messages for the security topic.

#### Code Sample

```
/*
 * The Producer thread executes a function to send messages periodically.
 */
public void run() {
    LOG.info("New Producer: start.");
    int messageNo = 1;

    while (messageNo <= MESSAGE_NUM) {
        String messageStr = "Message_" + messageNo;
        long startTime = System.currentTimeMillis();
```

```
// Construct message records.  
ProducerRecord<Integer, String> record = new ProducerRecord<Integer, String>(topic, messageNo,  
messageStr);  
  
if (isAsync) {  
    // Sending in asynchronous mode  
    producer.send(record, new DemoCallBack(startTime, messageNo, messageStr));  
} else {  
    try {  
        // Sending in synchronous mode  
        producer.send(record).get();  
        long elapsedTime = System.currentTimeMillis() - startTime;  
        LOG.info("message(" + messageNo + ", " + messageStr + ") sent to topic(" + topic + ") in " +  
elapsedTime + " ms.");  
    } catch (InterruptedException ie) {  
        LOG.info("The InterruptedException occurred : {}.", ie);  
    } catch (ExecutionException ee) {  
        LOG.info("The ExecutionException occurred : {}.", ee);  
    }  
}  
messageNo++;  
}
```

### 1.15.3.2.2 Consumer API Usage Sample

#### Function Description

The following code sample belongs to the **com.huawei.bigdata.kafka.example.Consumer** class. It is used to enable the Consumer API to subscribe a secure topic and consume messages.

#### Code Sample

```
/**  
 * Consumer constructor.  
 * @param topic Name of the subscribed topic  
 */  
  
public Consumer(String topic) {  
    super("KafkaConsumerExample", false);  
    // Initializes the configuration parameters required for starting the consumer. For details, see the code.  
    Properties props = initProperties();  
    consumer = new KafkaConsumer<Integer, String>(props);  
    this.topic = topic;  
}  
  
public void doWork() {  
    // Subscribe  
    consumer.subscribe(Collections.singletonList(this.topic));  
    // Message consumption request  
    ConsumerRecords<Integer, String> records = consumer.poll(waitTime);  
    // Message Processing  
    for (ConsumerRecord<Integer, String> record : records) {  
        LOG.info("[ConsumerExample], Received message: (" + record.key() + ", " + record.value() + ") at  
offset " + record.offset());  
    }  
}
```

### 1.15.3.2.3 Multi-thread Producer Sample

#### Function Description

The multi-thread producer function is implemented based on the code sample described in section [Producer API Usage Sample](#). Multiple producer threads can

be started. Each thread sends messages to the partition whose key is the same as the thread ID.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ProducerMultThread** class, and these code snippets are used to enable multiple threads to produce data.

## Code Sample

```
/*
 * Specify the current ThreadID as the key value and send data.
 */
public void run()
{
LOG.info("Producer: start.");

    // Record the number of messages.
int messageCount = 1;

    // Specify the number of messages sent by each thread.
int messagesPerThread = 5;
while (messageCount <= messagesPerThread)
{
    // Specify the content of messages to be sent.
    String messageStr = new String("Message_" + sendThreadId + "_" + messageCount);

    // Specify a key value for each thread to enable the thread to send messages to only a specified
partition.
    String key = String.valueOf(sendThreadId);

    // Send messages.
    producer.send(new KeyedMessage<String, String>(sendTopic, key, messageStr));
LOG.info("Producer: send " + messageStr + " to " + sendTopic + " with key: " + key);
    messageCount++;
}
}
```

### 1.15.3.2.4 Multi-thread Consumer Sample

## Function Description

The multi-thread consumer function is implemented based on the code sample described in section [Consumer API Usage Sample](#). The number of consumer threads that can be started to consume the messages in partitions is the same as the number of partitions in the topic.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ConsumerMultThread** class, and these code snippets are used to implement concurrent consumption of messages in a specified topic.

## Code Sample

```
/*
 * Start the concurrent multi-thread consumer.
 */
public void run() {
    LOG.info("Consumer: start.");
    Properties props = Consumer.initProperties();
    // Start a specified number of consumer threads to consume.
    // Note: When this parameter is larger than the number of partitions of the topic to be consumed, the
extra threads fails to consume data.
}
```

```
for (int threadNum = 0; threadNum < CONCURRENCY_THREAD_NUM; threadNum++) {
    new ConsumerThread(threadNum, topic, props).start();
    LOG.info("Consumer Thread " + threadNum + " Start.");
}
}

private class ConsumerThread extends ShutdownableThread {
    private int threadNum = 0;
    private String topic;
    private Properties props;
    private KafkaConsumer<String, String> consumer = null;

    /**
     * Construction method of the consumer thread class
     *
     * @param threadNum Thread number
     * @param topic      topic
     */
    public ConsumerThread(int threadNum, String topic, Properties props) {
        super("ConsumerThread" + threadNum, true);
        this.threadNum = threadNum;
        this.topic = topic;
        this.props = props;
        this.consumer = new KafkaConsumer<String, String>(props);
    }

    public void doWork() {
        consumer.subscribe(Collections.singleton(this.topic));
        ConsumerRecords<String, String> records = consumer.poll(waitTime);
        for (ConsumerRecord<String, String> record : records) {
            LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + " record: "
                    + record.value() + " offsets: " + record.offset());
        }
    }
}
```

### 1.15.3.3 Kafka Streams Scenario Description

#### Scenario Description

Kafka Streams is a lightweight stream processing framework provided by Apache Kafka. The input and output of Kafka Streams are stored in the Kafka cluster.

The following describes the most common WordCount samples.

#### Development Guideline

1. Create two topics on the Linux client to serve as the input and output topics.
2. Develop a Kafka Streams to implement the word count function. The system collects statistics on the number of words in each message by reading the message in the input topic, consumes data from the output topic, and outputs the statistical result in the form of a key-value pair.

### 1.15.3.4 Kafka Streams Sample Code Description

### 1.15.3.4.1 High level KafkaStreams API Usage Sample

#### Function Description

The following code snippets are used in the **createWordCountStream** method of the **com.huawei.bigdata.kafka.example.WordCountDemo** class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

#### Code Sample

```
static void createWordCountStream(final StreamsBuilder builder) {  
    // Receive the input records from input-topic.  
    final KStream<String, String> source = builder.stream(INPUT_TOPIC_NAME);  
  
    // Aggregate the calculation result of the key-value pair.  
    final KTable<String, Long> counts = source  
        // Process the received records and split according to the regular expression REGEX_STRING.  
        .flatMapValues(value ->  
            Arrays.asList(value.toLowerCase(Locale.getDefault()).split(REGEX_STRING)))  
        // Aggregate the calculation result of the key-value pair.  
        .groupBy((key, value) -> value)  
        // The final calculation result  
        .count();  
  
    // Output the key-value pair of the calculation result from the output topic.  
    counts.toStream().to(OUTPUT_TOPIC_NAME, Produced.with(Serdes.String(), Serdes.Long()));  
}
```

### 1.15.3.4.2 Low level KafkaStreams API Usage Sample

#### Function Description

The following code snippets are used in the **com.huawei.bigdata.kafka.example.WordCountProcessorDemo** class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

#### Code Sample

```
private static class MyProcessorSupplier implements ProcessorSupplier<String, String> {  
    @Override  
    public Processor<String, String> get() {  
        return new Processor<String, String>() {  
            // ProcessorContext instance, which provides the access of the metadata of the records being  
            processed  
            private ProcessorContext context;  
            private KeyValueStore<String, Integer> kvStore;  
  
            @Override  
            @SuppressWarnings("unchecked")  
            public void init(ProcessorContext context) {  
                // Save processor context in the local host, because it will be used for punctuate() and commit().  
                this.context = context;  
                // Execute punctuate() once every second.  
                this.context.schedule(Duration.ofSeconds(1), PunctuationType.STREAM_TIME, timestamp -> {
```

```
try (final KeyValueIterator<String, Integer> iter = kvStore.all()) {
    System.out.println("----- " + timestamp + " -----");
    while (iter.hasNext()) {
        final KeyValue<String, Integer> entry = iter.next();
        System.out.println("[" + entry.key + ", " + entry.value + "]");
        // Send the new records to the downstream processor as key-value pairs.
        context.forward(entry.key, entry.value.toString());
    }
}
});
// Search for the key-value states storage area named KEY_VALUE_STATE_STORE_NAME to
memorize the recently received input records.
this.kvStore = (KeyValueStore<String, Integer>)
context.getStateStore(KEY_VALUE_STATE_STORE_NAME);
}

// Process the receiving records of input topic. Split the records into words, and count the words.
@Override
public void process(String dummy, String line) {
    String[] words = line.toLowerCase(Locale.getDefault()).split(REGEX_STRING);

    for (String word : words) {
        Integer oldValue = this.kvStore.get(word);

        if (oldValue == null) {
            this.kvStore.put(word, 1);
        } else {
            this.kvStore.put(word, oldValue + 1);
        }
    }
}

@Override
public void close() {
}
};

}
```

### 1.15.3.5 Kafka Token Authentication Mechanism Scenario

#### Scenario

The token authentication mechanism is a lightweight authentication mechanism that does not require Kerberos authentication. It can be used in APIs.

#### Development Guideline

1. Use a Linux client to create a token. For details, see "Kafka Token Authentication Mechanism Tool Usage" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).
2. Use the token mechanism for authentication in the secondary development sample.

### 1.15.3.6 Sample Code for Kafka Token Authentication

#### Function

This example shows how to use Kafka user token for authentication.

## Prerequisite

The following parameters of the user token have been added to the *Sample project folder\src\main\resources\producer.properties* file:

- **sasl.mechanism**: SASL framework for user authentication. Set this parameter to **SCRAM-SHA-512**.
- **sasl.jaas.config**: token configuration used by the user

For example, add the following content. Set the parameters in the **producer.properties** file. Ensure that the semicolons (;) and colons (:) are correct.

```
sasl.mechanism = SCRAM-SHA-512
sasl.jaas.config = org.apache.kafka.common.security.scram.ScramLoginModule required username="Token
ID generated by the user" password="HMAC of the token generated by the user" tokenauth=true;
```

## Sample Code

Initialize the configuration in the **Producer()** method and load the token configuration from a file. Do not hardcode the password.

```
public static Properties initProperties() {
    Properties props = new Properties();
    KafkaProperties kafkaProc = KafkaProperties.getInstance();

    // Broker address list
    props.put(BOOTSTRAP_SERVER, kafkaProc.getValues(BOOTSTRAP_SERVER, "localhost:21007"));
    // Client ID
    props.put(CLIENT_ID, kafkaProc.getValues(CLIENT_ID, "DemoProducer"));
    // Key serialization class
    props.put(KEY_SERIALIZER,
        kafkaProc.getValues(KEY_SERIALIZER, "org.apache.kafka.common.serialization.StringSerializer"));
    // Value serialization class
    props.put(VALUE_SERIALIZER,
        kafkaProc.getValues(VALUE_SERIALIZER, "org.apache.kafka.common.serialization.StringSerializer"));
    // Protocol type: Currently, the SASL_PLAINTEXT or PLAINTEXT protocol types can be used.
    props.put(SECURITY_PROTOCOL, kafkaProc.getValues(SECURITY_PROTOCOL, "SASL_PLAINTEXT"));
    // Service name
    props.put(SASL_KERBEROS_SERVICE_NAME, "kafka");
    // Domain name
    props.put(KERBEROS_DOMAIN_NAME, kafkaProc.getValues(KERBEROS_DOMAIN_NAME,
    "hadoop.hadoop.com"));
    // Partition class name
    props.put(PARTITIONER_NAME,
        kafkaProc.getValues(PARTITIONER_NAME, "com.huawei.bigdata.kafka.example.SimplePartitioner"));
    return props;
}
```

 NOTE

When using the token authentication mechanism, you need to comment out the Kerberos authentication mechanism to ensure that only one authentication mechanism is used during code running, as shown in the following:

```
public static void main(String[] args)
{
    if (isSecurityModel())
    {
        // try
        //{
        //     LOG.info("Securitymode start.");
        //
        //     //!!!Note: When using security authentication, you need to change the account to a
        // machine-machine one.
        //     securityPrepare();
        //}
        // catch (IOException e)
        //{
        //     LOG.error("Security prepare failure.");
        //     LOG.error("The IOException occurred.", e);
        //     return;
        //}
        // LOG.info("Security prepare success.");
    }

    // Specify whether to use the asynchronous sending mode.
    final boolean asyncEnable = false;
    Producer producerThread = new Producer(KafkaProperties.TOPIC, asyncEnable);
    producerThread.start();
}
```

### 1.15.3.7 Sample Code for Connecting Kafka to Spring Boot

#### Function

Spring Boot is used to produce and consume Kafka clusters.

#### Sample Code

Use Spring Boot to implement Kafka production and consumption.

```
@RestController
public class MessageController {
    private final static Logger LOG = LoggerFactory.getLogger(MessageController.class);
    @Autowired
    private KafkaProperties kafkaProperties;
    @GetMapping("/produce")
    public String produce() {
        Producer producerThread = new Producer();
        producerThread.init(this.kafkaProperties);
        producerThread.start();
        String message = "Start to produce messages";
        LOG.info(message);
        return message;
    }
    @GetMapping("/consume")
    public String consume() {
        Consumer consumerThread = new Consumer();
        consumerThread.init(this.kafkaProperties);
        consumerThread.start();
        LOG.info("Start to consume messages");
        // Wait for 180s and close the consumer. Modification can be made during actual execution.
        try {
            Thread.sleep(consumerThread.getThreadAliveTime());
        }
```

```
        } catch (InterruptedException e) {
            LOG.info("Occurred InterruptedException: ", e);
        } finally {
            consumerThread.close();
        }
        return String.format("Finished consume messages");
    }
}
```

**Produce** indicates the message production interface, **consume** indicates the message consumption interface, and **KafkaProperties** indicates the client parameter. The parameters need to be modified based on the actual service.

#### NOTE

The **KafkaProperties** parameter in the sample code can be configured in **springboot > kafka-examples > src > main > resources > application.properties** or manually compiled in the **application.properties** file in the sample running environment. If no default value is specified, the configuration is mandatory.

- **bootstrap.servers**: Broker address list of the Kafka cluster. The format is **IP address: Port, IP address: Port, IP address: Port**. In the IPv6 environment, add **[]** to the IP address, for example, **[1:2:3:4:5:6:7:8]:21007**.
- **security.protocol**: authentication protocol used by the Kafka client. The default value is **SASL\_PLAINTEXT**. The **SASL\_SSL** protocol is supported.
- **sasl.mechanism**: authentication mechanism used by the client. The default value is **PLAIN**.
- **manager\_username**: user of the cluster. The value must be enclosed in double quotation marks, for example, **"username"**.
- **manager\_password**: password of the cluster user. Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables. The value must be enclosed in double quotation marks, for example, **"password"**.
- **topic**: name of the production and consumption topic. The default value is **example-metric1**.
- **isAsync**: whether to use asynchronous production. The default value is **false**.
- **consumer.alive.time**: lifetime of the consumer thread. The default value is **180000**, in milliseconds.
- **server.port**: port for accessing the Spring Boot server. The default value is **8080**, which can be customized.
- **server.address**: IP address bound to the Spring Boot server during the startup. The default value is **0.0.0.0**, which needs to be changed to the IP address of the node where Spring Boot is deployed.
- **is.security.mode**: whether the client connects to the cluster in security mode. The default value is **true**.

## Sample Running

### Step 1 Enable Kafka Plain authentication.

Log in to FusionInsight Manager, choose **Cluster > Services > Kafka** and choose **Configurations > All Configurations**. Search for **sasl.enabled.mechanisms**, change the value to **GSSAPI,PLAIN**, Click **Save**, click **Dashboard**, click **More**, and select **Restart Service** to make the configuration take effect.

### Step 2 Obtain the **huawei-spring-boot-kafka-examples-\*jar** package.

Find the POM file in the **springboot/kafka-examples** directory of the sample code, and use the Maven install tool to compile the Spring Boot sample in the

same directory as the POM file. A target folder is generated, and **huawei-spring-boot-kafka-examples-\*jar** is obtained from the **target** folder.

**Step 3** Create a directory on Windows or Linux as the running directory.

- Create the **D:\Spring** directory in the Windows OS and upload the **huawei-spring-boot-kafka-examples-\*jar** and **application.properties** files to the current directory.
- Create the **/opt/spring** directory on the Linux OS and upload the **huawei-spring-boot-kafka-examples-\*jar** and **application.properties** files to the current directory.

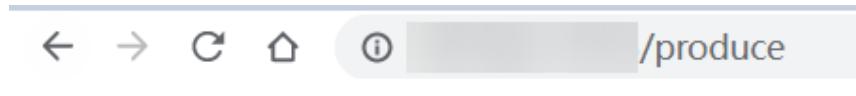
**Step 4** Run the corresponding command to start Spring Boot:

- In Windows, open the CLI in the **D:\Spring** directory and run the following command:  
**java -jar huawei-spring-boot-kafka-examples-\*jar**
- In Linux, run the following command in the **/opt/spring** directory:  
**java -jar huawei-spring-boot-kafka-examples-\*jar**

**Step 5** Produce data.

- For a Windows OS, open a browser and enter **http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/produce** to generate data to Broker, as shown in the following figure.

**Figure 1-201** Producing data

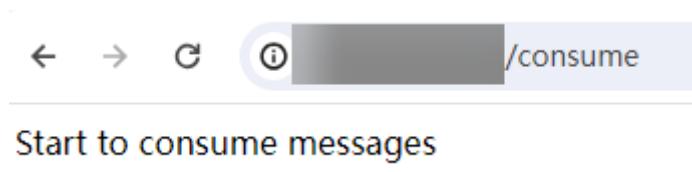


- For a Linux OS, run the **curl** command to access Spring Boot.  
**curl http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/produce**

**Step 6** Consume data.

- For a Windows OS, open a browser and enter **http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/consume** to consume data from broker, as shown in the following figure.

**Figure 1-202** Consuming data



- For a Linux OS, run the **curl** command to access Spring Boot.  
**curl http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/consume**

----End

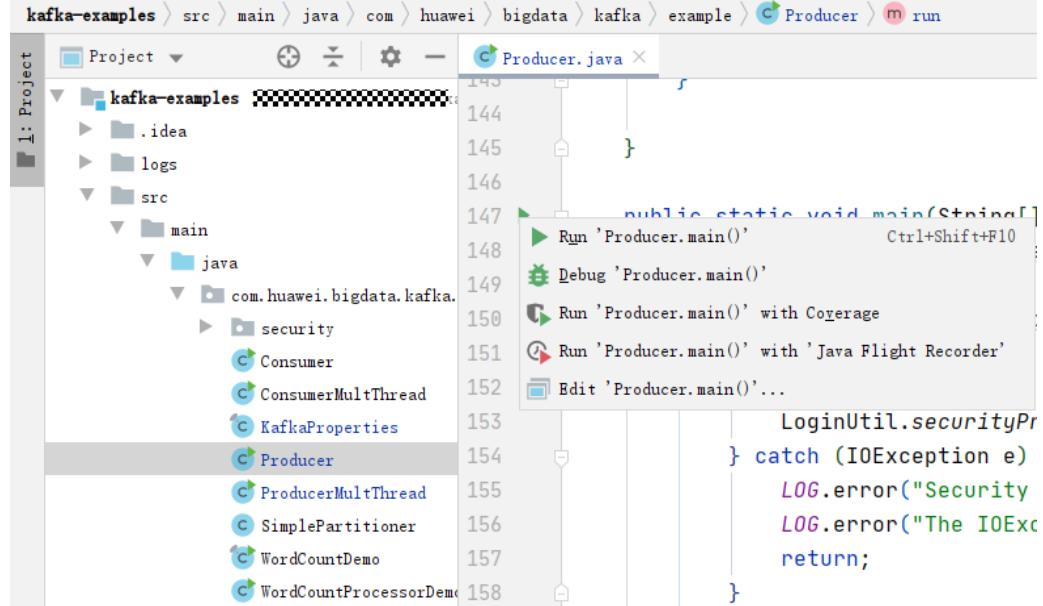
## 1.15.4 Application Commissioning

### 1.15.4.1 Commissioning an Application in Windows

#### Starting Producer

- Step 1** Ensure that the mappings between the host names and service IP addresses of all the hosts in the remote cluster are configured in the local **hosts** file.
- Step 2** Run **Producer.java** on IntelliJ IDEA, as shown in [Figure 1-203](#).

**Figure 1-203** Run **Producer.java** on IntelliJ IDEA



- Step 3** The console window appears. You can find that Producer is sending messages to the default topic (example-metric1). One piece of log is printed when every 10 messages are sent.

**Figure 1-204** Procedurer running window

```
[2019-06-12 10:31:37,865] INFO Updated cluster metadata version 2 to Cluster(id = 1cPugjn8Q4ernMH8RS_~JA, nodes = [187.
[2019-06-12 10:31:51,729] INFO Updated cluster metadata version 3 to Cluster(id = 1cPugjn8Q4ernMH8RS_~JA, nodes = [187.
[2019-06-12 10:31:53,140] INFO The Producer have send 10 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:54,516] INFO The Producer have send 20 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:55,906] INFO The Producer have send 30 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:57,299] INFO The Producer have send 40 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:58,686] INFO The Producer have send 50 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:32:00,070] INFO The Producer have send 60 messages. (com.huawei.bigdata.kafka.example.NewProducer)
```

----End

#### Starting Consumer

- Step 1** Run the **Consumer.java** file.

**Step 2 Click Run.** In the console window that appears, you can find that Producer starts after Consumer successfully starts, and then you can view messages received in real time.

**Figure 1-205 Consumer.java running window**

```
[2019-06-23 17:49:48,609] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] (Re-)joining group (org.apache.kafka.clients.consumer.internals.AbstractCoordinator)
[2019-06-23 17:49:51,865] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Successfully joined group with generation 5 (org.apache.kafka.clients.consumer.internals.ConsumerCoordinator)
[2019-06-23 17:49:51,866] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Setting newly assigned partitions [example-metric1-0, example-metric1-1] (org.apache.kafka.clients.consumer.internals.ConsumerCoordinator)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (1, Message_1) at offset 188 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (5, Message_5) at offset 189 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (8, Message_8) at offset 190 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (9, Message_9) at offset 191 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (15, Message_15) at offset 192 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (18, Message_18) at offset 193 (com.huawei.bigdata.kafka.example.NewConsumer)
```

----End

## Starting Other Code Samples

The procedures for starting other code samples are similar to the procedures for starting Producer and Consumer in this section.

### 1.15.4.2 Commissioning an Application in Linux

#### Scenario

Run a sample program in Linux after code development is complete.

#### Procedure

**Step 1 Export a JAR package.**

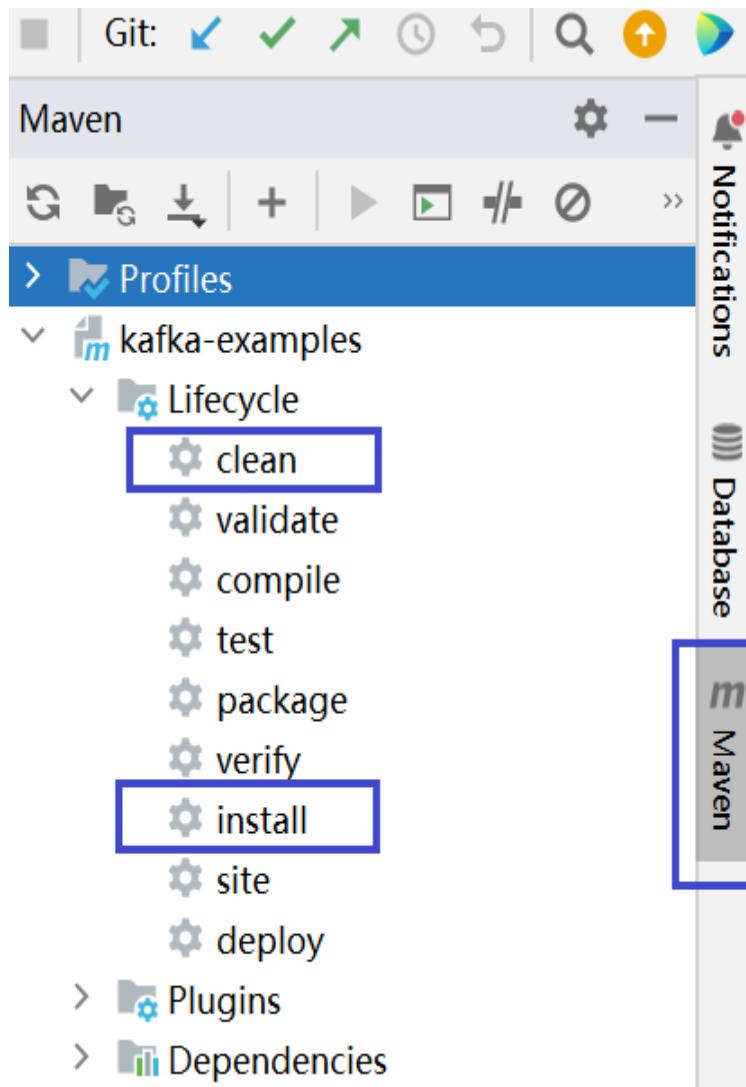
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 1-206 Maven tools: clean and install



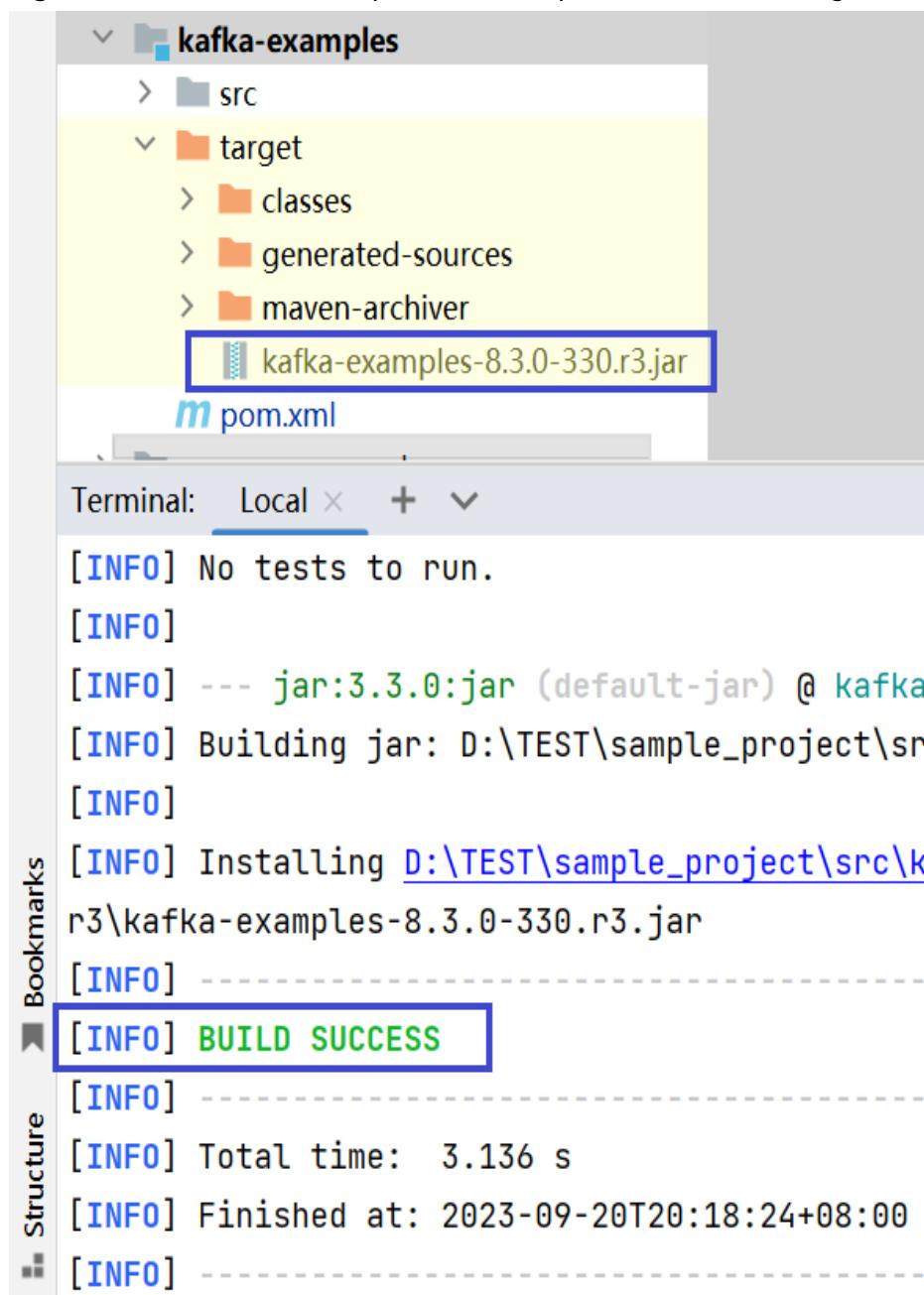
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 1-207 Enter mvn clean compile in the IDEA Terminal text box.

The screenshot shows the IntelliJ IDEA terminal window with the command `mvn clean install` entered. The terminal output area is currently empty.

After the compilation is completed, the message "Build Success" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

Figure 1-208 When the compilation is completed, the JAR file is generated.



- Step 2** Copy the JAR file generated during project compilation to the `/opt/hadoopclient/Kafka/kafka/libs` directory.
- Step 3** Copy all files in the `src/main/resources` directory of the IntelliJ IDEA project to the `src/main/resources` directory at the same level as the `lib` folder, that is, `/opt/hadoopclient/src/kafka-examples/src/main/resources`.
- Step 4** Ensure that the current user has read permission of all the files in the `src/main/resources` and `libs` folders in `/opt/hadoopclient/src/kafka-examples/`, jdk has been installed, and java environment variables are set. Then, run the command, for example, `java -cp /opt/hadoopclient/Kafka/kafka/libs/*:src/main/resources com.huawei.bigdata.kafka.example.Producer` to run the example project.

----End

## 1.15.4.3 Kafka Streams Sample Running Guide

### 1.15.4.3.1 High level Streams API Sample Usage Guide

1. Open sample code **WordCountDemo.java** in the IDE and change the following two variables to the machine-machine account name and keytab file that you apply for.

```
// Use the machine-machine account keytab file that you apply for.  
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name.;"  
// Use the machine-machine account name that you apply for.  
private static final String USER_PRINCIPAL = "Change it to the real-world username.;"
```

2. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic  
private static final String INPUT_TOPIC_NAME = "streams-wordcount-input";  
// sink-topic name created by the user, that is, output topic  
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-output";
```

- Create the input topic:

```
kafka-topics.sh --create --bootstrap-server  
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --  
command-config config/client.properties --replication-factor 1 --  
partitions 1 --topic streams-wordcount-input
```

- Create the output topic:

```
kafka-topics.sh --create --bootstrap-server  
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --  
command-config config/client.properties --replication-factor 1 --  
partitions 1 --topic streams-wordcount-output --config  
cleanup.policy=compact
```

- Run sample code **WordCountDemo.java**. For details, see [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).

3. Use the Linux client to write messages and view the statistics result.

Run the **kafka-console-producer.sh** command to write messages to the input topic.

```
# kafka-console-producer.sh --broker-list 192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --  
topic streams-wordcount-input --producer.config /opt/hadoopclient/Kafka/kafka/config/  
producer.properties  
>This is Kafka Streams test  
>test starting  
>now Kafka Streams is running  
>test end  
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-output --bootstrap-server  
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --consumer.config /opt/hadoopclient/  
Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property  
print.value=true --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --  
property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer --formatter  
kafka.tools.DefaultMessageFormatter  
this 1  
is 6  
kafka 12  
streams 8  
test 8
```

```
test 9
starting 1
now 1
kafka 13
streams 9
is 7
running 1
test 10
end 1
```

#### 1.15.4.3.2 Low level Streams API Sample Usage Guide

1. Open sample code **WordCountProcessorDemo.java** in the IDE and change the following two variables to the machine-machine account name and keytab file that you apply for.

```
// Use the machine-machine account keytab file that you apply for.
private static final String USER_KEYTAB_FILE = "Change it to the real-world keytab file name";
// Use the machine-machine account name that you apply for.
private static final String USER_PRINCIPAL = "Change it to the real-world username. ";
```

2. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic
private static final String INPUT_TOPIC_NAME = "streams-wordcount-processor-input";
```

```
// sink-topic name created by the user, that is, output topic
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-processor-output";
```

- Create the input topic:

```
kafka-topics.sh --create --bootstrap-server
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --
command-config config/client.properties --replication-factor 1 --
partitions 1 --topic streams-wordcount-processor-input
```

- Create the output topic:

```
kafka-topics.sh --create --bootstrap-server
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --
command-config config/client.properties --replication-factor 1 --
partitions 1 --topic streams-wordcount-processor-output --config
cleanup.policy=compact
```

- Run sample code **WordCountProcessorDemo.java**. For details, see [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).

3. Use the Linux client to write messages and view the statistics result.

Run the **kafka-console-producer.sh** command to write messages to the input topic.

```
# kafka-console-producer.sh --broker-list 192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --
topic streams-wordcount-processor-input --producer.config /opt/hadoopclient/Kafka/kafka/config/
producer.properties
>This is Kafka Streams test
>now Kafka Streams is running
>test end
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-processor-output --bootstrap-server
192.168.0.11:21007,192.168.0.12:21007,192.168.0.13:21007 --consumer.config /opt/hadoopclient/
Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property
print.value=true
```

```
is 1
kafka 1
streams 1
test 1
this 1
end 1
is 2
kafka 2
now 1
running 1
streams 2
test 2
this 1
```

#### 1.15.4.4 Sample Code Running Guide for the Kafka Token Authentication Mechanism

### Procedure

#### Kafka Server Configuration

**Step 1** On the FusionInsight Manager portal, choose **Cluster**, click the name of the desired cluster, and choose **Service > Kafka**. On the displayed page, click **Configuration** to go to the Kafka service configuration page.

**Step 2** Enable the token authentication mechanism.

Find the **delegation.token.master.key** parameter, which specifies the master key used to generate and verify tokens. Check whether the parameter has been configured. If it has been configured, and the value is not **null**, the token authentication mechanism has been enabled and does not need to be reconfigured. If the token authentication mechanism is configured again, the original token cannot be used.



The value of **delegation.token.master.key** can be customized, for example, **Tokentest**.

**Step 3** Specify the SASL authentication mechanism used for the service.

Find the **sasl.enabled.mechanisms** parameter and set it to **GSSAPI,SCRAM-SHA-256,SCRAM-SHA-512**. Use commas (,) to separate the three items.

**Step 4** Log in to a component using Scram.

Find the custom parameter **kafka.config.expandor** and set its name to **listener.name.sasl\_plaintext.scram-sha-512.sasl.jaas.config**. Set the value to **org.apache.kafka.common.security.scram.ScramLoginModule required;**

**Step 5** Log in to FusionInsight Manager and restart all Broker instances of the Kafka service.

#### Kafka Client Configuration

**Step 6** Generate a token for the user. For details, see "Kafka Token Authentication Mechanism Tool Usage" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

#### Secondary Development Sample Code Project Configuration

The token authentication mechanism can be used for APIs. Therefore, you can configure the token authentication mechanism in **Producer()** and **Consumer()** of the secondary development sample.

**Step 7** Enable the token authentication mechanism.

Set **tokenauth** to **true** on the client.

**Step 8** Specify the SASL authentication mechanism used for the service.

Set **sasl.mechanism** to **SCRAM-SHA-512** on the client.

**Step 9** Configure the Java Authentication and Authorization Service (JAAS) file.

Configure **sasl.jaas.config** on the client as follows:

```
org.apache.kafka.common.security.scram.ScramLoginModule required  
username="TOKENID"  
password="HMAC";
```

**TOKENID** and **HMAC** are generated when tokens are generated in **Step 6**.

#### NOTE

For details about how to enable the token authentication mechanism, specify the SASL authentication mechanism for a specified service, and configure the JAAS file, see [Sample Code for Kafka Token Authentication](#).

**Step 10** Run the sample code based on the running environment. For details, see [Commissioning an Application in Windows](#) and [Commissioning an Application in Linux](#).

----End

## 1.15.5 More Information

### 1.15.5.1 External Interfaces

#### 1.15.5.1.1 Shell

1. Query the list of topics in current clusters.

```
bin/kafka-topics.sh --list --bootstrap-server <Kafka cluster IP  
address:21007> --command-config config/client.properties
```

2. Query the details of a topic.

```
bin/kafka-topics.sh --describe --bootstrap-server <Kafka cluster IP  
address:21007> --command-config config/client.properties --topic <Topic  
name>
```

3. Delete a topic (this operation can be performed by an administrator).

```
bin/kafka-topics.sh --delete --bootstrap-server <Kafka cluster IP  
address:21007> --command-config config/client.properties --topic <Topic  
name>
```

4. Create a topic (this operation can be performed by an administrator).

```
bin/kafka-topics.sh --create --bootstrap-server <Kafka cluster IP  
address:21007> --command-config config/client.properties --partitions 6  
--replication-factor 2 --topic <Topic name>
```

5. Assign permissions to the Consumer (this operation is performed by an administrator).

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:24002/kafka> --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
```

```
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
```

6. Assign permissions to the Producer (this operation is performed by an administrator).

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:24002/kafka> --add --allow-principal User:<User name> --producer --topic <Topic name>
```

```
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21007> --command-config config/client.properties --add --allow-principal User:<User name> --producer --topic <Topic name>
```

7. Produce messages (this operation requires the producer permission of the desired topic).

```
bin/kafka-console-producer.sh --broker-list <KafkaCluster IP address:21007> --topic <Topic name> --producer.config config/producer.properties
```

8. Consume data (the producer permission of the topic is required).

```
bin/kafka-console-consumer.sh --topic <Topic name> --bootstrap-server <KafkaCluster IP address:21007> --consumer.config config/consumer.properties
```

For details, see the MapReduce Service (MRS) 3.3.1-LTS Shell O&M Commands (for Huawei Cloud Stack 8.3.1).

### 1.15.5.1.2 Java API

Versions of interfaces adopted by Kafka are consistent with those in Open Source Community. For details, see <https://kafka.apache.org/24/documentation.html>.

## Major Interfaces of a Producer

**Table 1-136** Major parameters of a Producer

| Parameter         | Description         | Remarks   |
|-------------------|---------------------|---|
| bootstrap.servers | Broker address list | The Producer creates connections with the Broker based on this parameter. |

| Parameter                  | Description                             | Remarks   |
|----------------------------|---|---|
| security.protocol          | Security protocol type                  | The Producer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as <b>SASL_PLAINTEXT</b> . |
| sasl.kerberos.service.name | Service name                            | This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as <b>kafka</b> .  |
| key.serializer             | Serialization of key values in messages | This parameter specifies how to serialize the key values in messages.   |
| value.serializer           | Serialization of messages               | This parameter specifies how to serialize transmitted messages.   |

**Table 1-137** Major interface functions of a Producer

| Return Value                                | Interface Function                                   | Description  |
|---|--|--|
| java.util.concurrent.Future<RecordMetadata> | send(ProducerRecord<K, V> record)                    | Indicates a TX interface without a callback function. Generally, the get() function of Future is used for synchronous transmission.                                      |
| java.util.concurrent.Future<RecordMetadata> | send(ProducerRecord<K, V> record, Callback callback) | Indicates a TX interface with a callback function. Generally, this interface uses the callback function to process transmission results after asynchronous transmission. |

| Return Value | Interface Function  | Description   |
|--------------|---|---|
| void         | onCompletion(RecordMe tadata metadata,<br>Exception exception); | Indicates the interface method for a callback function. This method is used to process asynchronous transmission results. |

## Major Interfaces of a Consumer

**Table 1-138** Major parameters of a Consumer

| Parameter                  | Description                               | Remarks   |
|----------------------------|---|---|
| bootstrap.servers          | Broker address list                       | The Consumer creates connections with the Broker based on this parameter.   |
| security.protocol          | Security protocol type                    | The Consumer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as <b>SASL_PLAINTEXT</b> . |
| sasl.kerberos.service.name | Service name                              | This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as <b>kafka</b> .  |
| key.deserializer           | Deserialization of key values in messages | This parameter specifies how to deserialize the key values in messages.   |
| value.deserializer         | Deserialization of messages               | This parameter specifies how to deserialize received messages.  |

**Table 1-139** Major interface functions of a Consumer

| Return Value         | Interface Function                                       | Description  |
|----------------------|--|--|
| void                 | close()  | Indicates the interface method for closing the Consumer. |
| void                 | subscribe(java.util.Collection<java.lang.String> topics) | Indicates the interface method for subscribing topics.   |
| ConsumerRecords<K,V> | poll(final Duration timeout)                             | Indicates the interface method for requesting messages.  |

### 1.15.5.1.3 Security Ports

1. By default, port 21007 is used for access to secure Kafka clusters, and port 21005 is used for access to normal Kafka clusters.
2. API supports access via both ports 21005 and 21007.

### 1.15.5.1.4 SSL Encryption Function Used by a Client

#### Prerequisites

1. Before enabling the SSL function on the client, ensure that the SSL service function on the server has been enabled (**ssl.mode.enable** of the server has been set to **true**).
2. The SSL function requires APIs. For details, see "Component Operation Guide > Using Kafka > Safety Instruction on Using Kafka" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).

#### Description

- **SSL used by a Linux client**
  - a. Change the value of **security.protocol** in the **client installation directory/Kafka/kafka/config/producer.properties** and **client installation directory/Kafka/kafka/config/consumer.properties** directories to **SASL\_SSL** or **SSL**.
  - b. When using the Shell commands, enter a port ID corresponding to the protocol set in Step 1. For example, if **security.protocol** is set to **SASL\_SSL**, an SASL\_SSL protocol port ID is required, which is **21009** by default:  
**bin/kafka-console-producer.sh --broker-list </P address of a Kafka cluster:21009> --topic <Topic name> --producer.config config/producer.properties**  
**bin/kafka-console-consumer.sh --topic <Topic name> --bootstrap-server </P address of a Kafka cluster:21009> --consumer.config config/consumer.properties**

- **SSL used by a Windows client**
  - a. Download the Kafka client, decompress the client, and find the **ca.crt** file in the root directory.
  - b. Use the **ca.crt** file to generate the TrustStore file of the client.  
Run the **keytool -noprompt -import -alias myservercert -file ca.crt -keystore truststore.jks** command in the Java running environment.
  - c. Create the **conf** directory in the **kafka-examples** directory of the IntelliJ IDEA project (at the same level as the **src** directory), and copy the generated **truststore.jks** file to the **conf** directory. Add the following codes to the client codes (**initProperties ()** method of **Producer.java** and **Consumer.java**):

```
//truststore file address  
props.put("ssl.truststore.location", System.getProperty("user.dir") + File.separator + "conf" +  
File.separator + "truststore.jks");  
//truststore file password (password when the TrustStore file is generated)  
props.put("ssl.truststore.password", "XXXXX");
```
  - d. Change the values of **security.protocol** in **producer.properties** and **consumer.properties** in the **src/main/resources** directory of the IntelliJ IDEA project as required, and change the value of **bootstrap.servers** in the **producer.properties** file to ensure that the type of **security.protocol** matches with the port ID of **bootstrap.servers**.

### 1.15.5.2 FAQ

#### 1.15.5.2.1 Topic Authentication Fails During Sample Running and "example-metric1=TOPIC\_AUTHORIZATION\_FAILED" Is Displayed

##### Troubleshooting Procedure

- Step 1** Apply to the administrator for the access permission for the related topic.
- Step 2** If the topic cannot be logged in after the access permission is assigned, log in to FusionInsight Manager, go to the Kafka service configuration page, search for **allow.everyone.if.no.acl.found**, and change the value to **true**. Then, run the sample again.
- End

#### 1.15.5.2.2 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic

##### Troubleshooting Procedure

- Step 1** Find **bootstrap.servers** in **producer.properties** under the **conf** directory of the project, and check whether the IP address and port ID are configured correctly.
- If the IP address is inconsistent with the service IP address of the Kafka cluster, change the IP address to the correct one.
  - If the port ID is 21007 (security mode port), change it to 21005 (normal mode port).

**Step 2** Check whether network connections are correct to ensure that the current device can access the Kafka cluster normally.

----End

## 1.16 MapReduce Development Guide

### 1.16.1 Overview

#### 1.16.1.1 MapReduce Overview

##### MapReduce Introduction

Hadoop MapReduce is an easy-to-use parallel computing software framework. Applications developed based on MapReduce can run on large clusters consisting of thousands of servers and process data sets larger than 1 TB in fault tolerance (FT) mode.

A MapReduce job (application or job) splits an input data set into several data blocks which then are processed by Map tasks in parallel mode. The framework sorts output results of the Map task, sends the results to Reduce tasks, and returns a result to the client. Input and output information is stored in the Hadoop Distributed File System (HDFS). The framework schedules and monitors tasks and re-executes failed tasks.

MapReduce supports the following features:

- Large-scale parallel computing
- Large data set processing
- High FT and reliability
- Reasonable resource scheduling

#### 1.16.1.2 Basic Concepts

##### Hadoop shell command

Basic hadoop shell commands include commands that are used to submit MapReduce jobs, kill MapReduce jobs, and perform operations on the HDFS.

##### MapReduce InputFormat and OutputFormat

Based on the specified InputFormat, the MapReduce framework splits data sets, reads data, provides key-value pairs for Map tasks, and determines the number of Map tasks that are started in parallel mode. Based on the OutputFormat, the MapReduce framework outputs the generated key-value pairs to data in a specific format.

Map and Reduce tasks are running based on <key,value> pairs. In other words, the framework regards the input information about a job as a group of key-value pairs and outputs a group of key-value pairs. Two groups of key-value pairs may

be of different types. For a single Map or Reduce task, key-value pairs are processed in single-thread serial mode.

The framework needs to perform serialized operations on key and value classes. Therefore, the classes must support the Writable interface. To facilitate sorting operations, key classes must support the WritableComparable interface.

The input and output types of a MapReduce job are as follows:

(input) <k1,v1> -> Map -> <k2,v2> -> Summary data -> <k2, List(v2)> -> Reduce -> <k3,v3> (output)

## Job Core

In normal cases, an application only needs to inherit Mapper and Reducer classes and rewrite map and reduce methods to implement service logic. The map and reduce methods constitute the core of jobs.

## MapReduce WebUI

Allows users to monitor running or historical MapReduce jobs, view logs, and implement fine-grained job development, configuration, and optimization.

## Keytab file

A key file for storing user information. Applications use the key file for application programming interface (API) authentication on product.

## Reduce

A processing model function that merges all intermediate values associated with the same intermediate key.

## Shuffle

A process of outputting data from a Map task to a Reduce task.

## Map

A method used to map a group of key-value pairs into a new group of key-value pairs.

### 1.16.1.3 Development Process

All stages of the development process are shown and described in [Figure 1-209](#) and [Table 1-140](#).

Figure 1-209 MapReduce development process

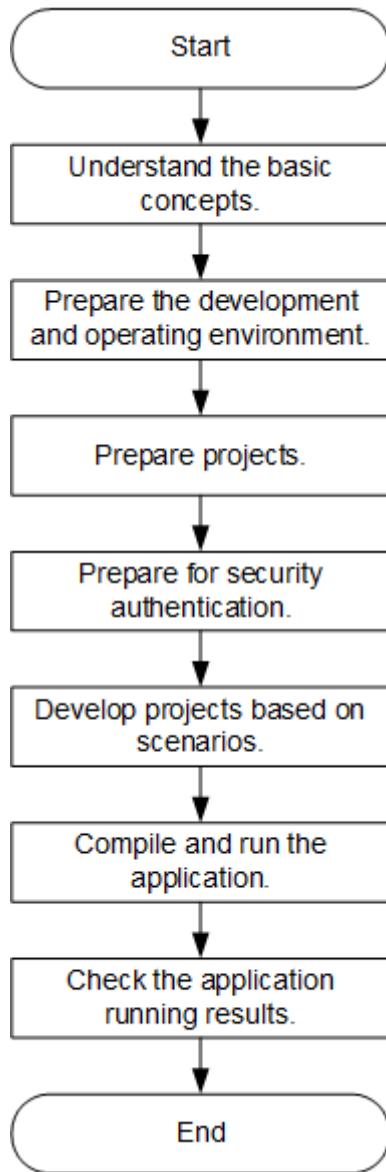


Table 1-140 Description of MapReduce development process

| Stage  | Description   | Reference   |
|--|---|---|
| Understand the basic concepts.                     | Before the application development, the basic concepts of MapReduce are required to be understood.  | <a href="#">Basic Concepts</a>                                      |
| Prepare the development and operating environment. | Use IntelliJ IDEA and configure the development environment based on the reference.<br>The running environment of MapReduce is the MapReduce client. Install and configure the client based on the reference. | <a href="#">Preparing for Development and Operating Environment</a> |

| Stage                                  | Description  | Reference  |
|--|--|--|
| Prepare projects                       | MapReduce provides sample projects in various scenarios. Sample projects can be imported for studying. Or you can create a new MapReduce project based on the reference. | <a href="#">Configuring and Importing Sample Projects</a><br><a href="#">Creating a New Project (Optional)</a> |
| Prepare for safety certification.      | If a safe cluster is used, the safety certification must be performed.   | <a href="#">Preparing the Authentication Mechanism</a>   |
| Develop projects based on scenarios.   | Provide the example project.<br>This helps users to learn about the programming interfaces of all Spark components quickly.  | <a href="#">Developing the Project</a>   |
| Compile and run the application.       | Users compile the developed application and deliver it for running based on the reference.   | <a href="#">Commissioning the Application</a>  |
| Check the application running results. | Application running results are stored in the directory specified by users. Users can also check the running results through the UI.                                     | <a href="#">Commissioning the Application</a>  |

## 1.16.2 Environment Preparation

### 1.16.2.1 Preparing for Development and Operating Environment

#### Preparing Development Environment

[Table 1-141](#) describes the environment required for application development.

**Table 1-141** Development environment

| Preparation Item | Description   |
|------------------|---|
| OS               | <ul style="list-style-type: none"><li>Development environment: Windows OS. Windows 7 or later is supported.</li><li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul> |

| Preparation Item                             | Description   |
|--|---|
| IntelliJ IDEA installation and configuration | <p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>  |
| Maven installation                           | Basic configuration of the development environment for project management throughout the lifecycle of software development.   |
| Installing JDK                               | <p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul> |
| Developer account preparation                | See <a href="#">Preparing a Developer Account</a> for configuration.  |
| 7-zip  | Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.  |

| Preparation Item          | Description   |
|---------------------------|---|
| checkJarsCrossPlatform.sh | In a cluster with both the x86 and TaiShan platforms, if a third-party JAR package not supported on the x86 and TaiShan platforms is required in the MapReduce program, you need to check whether this JAR package supports cross-platform deployment. If it does not support, resolve this problem following instructions provided in <a href="#">Cross-Platform Compatibility of JAR Packages</a> . |

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#). In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** and obtain the cluster configuration file. The configuration file will be imported to the configuration file directory (usually the **conf** folder) of the MapReduce sample project.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:  

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Yarn/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory.
  - c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

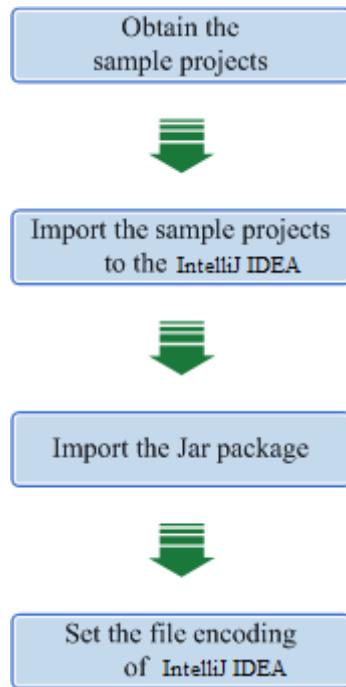
## 1.16.2.2 Configuring and Importing Sample Projects

### Scenario

MapReduce provides sample projects for multiple scenarios to help you quickly learn MapReduce projects.

The procedure of importing MapReduce example code is described as follows: [Figure 1-210](#) shows the procedure.

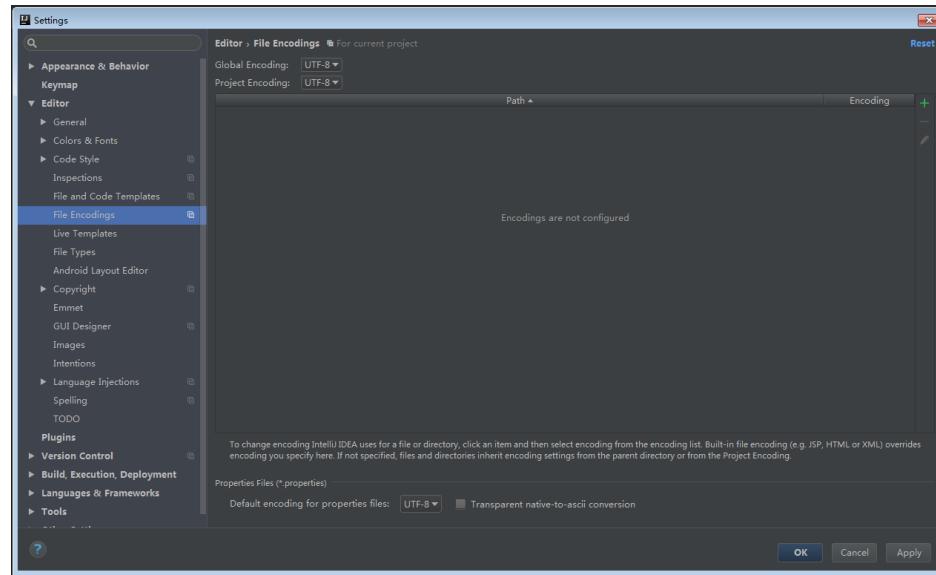
Figure 1-210 Procedure of importing sample projects



## Procedure

- Step 1** Obtain the sample project folder **mapreduce-example-security** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the **user.keytab** and **krb5.conf** files obtained during [Preparing a Developer Account](#) and the cluster configuration file obtained during running environment preparation in [Preparing an Operating Environment](#) to the **conf** directory of the sample project.
- Step 3** Import the sample project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > Open**.
  2. Choose the directory of the example project **mapreduce-example-security**. Click **OK**.
- Step 4** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
  1. On the IntelliJ IDEA menu bar, choose **File > Settings**.  
The **Settings** window is displayed.
  2. Choose **Editor > File Encodings** from the navigation tree. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 1-211](#)

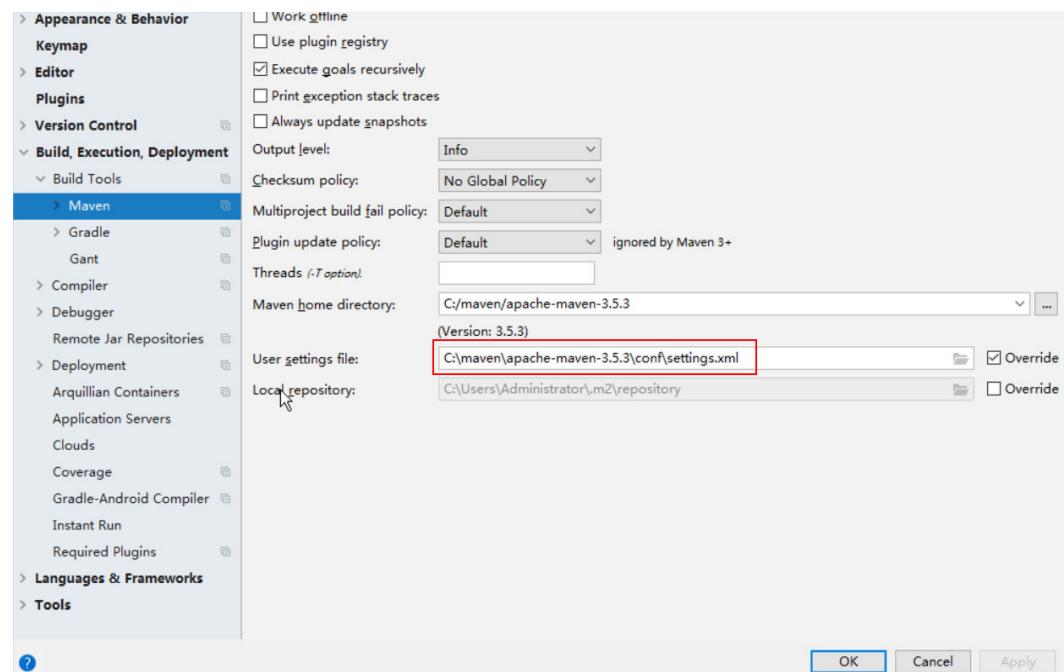
Figure 1-211 Setting the IntelliJ IDEA coding format



**Step 5** Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).

On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

Figure 1-212 Directory for storing the **settings.xml** file



----End

## References

The dependency packages mapped to the sample projects of MapReduce are as follows:

- MapReduce statistics sample project  
No additional jars.
- MapReduce accessing multi-components sample project

 NOTE

- If you want to use the multi-component accessing sample project after importing a sample project, ensure that the Hive and HBase services have been installed in the cluster.
- If you do not use the multi-components accessing sample project, you can ignore errors about the multi-components accessing sample project as long as the compilation of the statistics sample project is not affected. Otherwise, delete the files about the multi-components accessing sample project after importing the sample projects.

### 1.16.2.3 Creating a New Project (Optional)

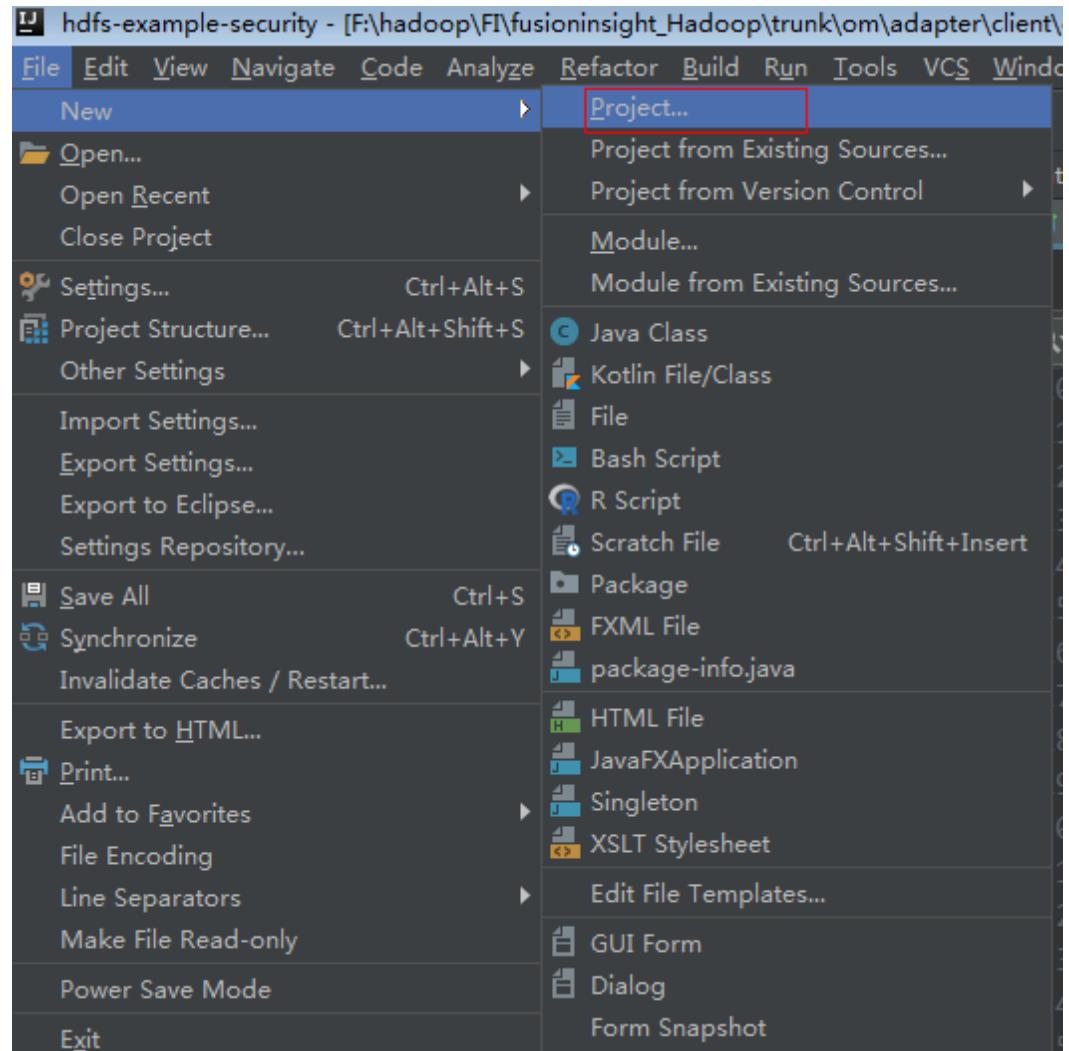
#### Scenario

Apart from importing MapReduce sample projects, you can also create a new MapReduce project using IntelliJ IDEA.

#### Procedure

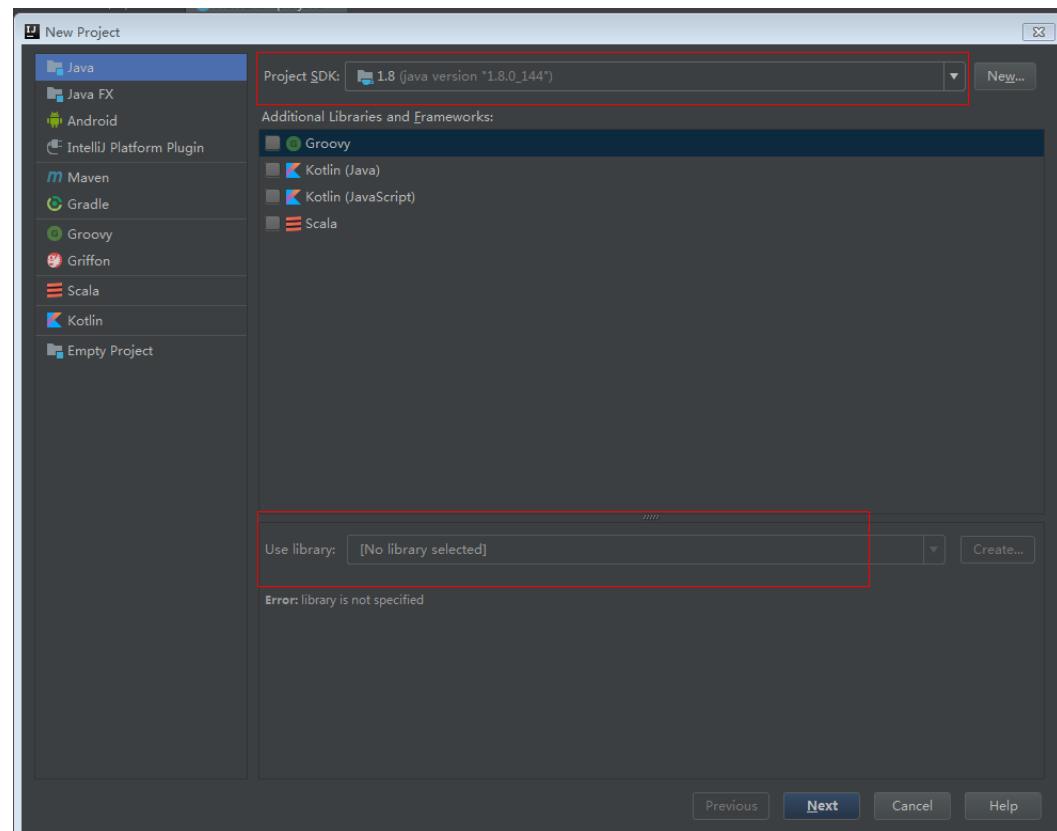
**Step 1** Open IntelliJ IDEA and choose **File > New > Project**, as shown in [Figure 1-213](#).

Figure 1-213 Creating a project



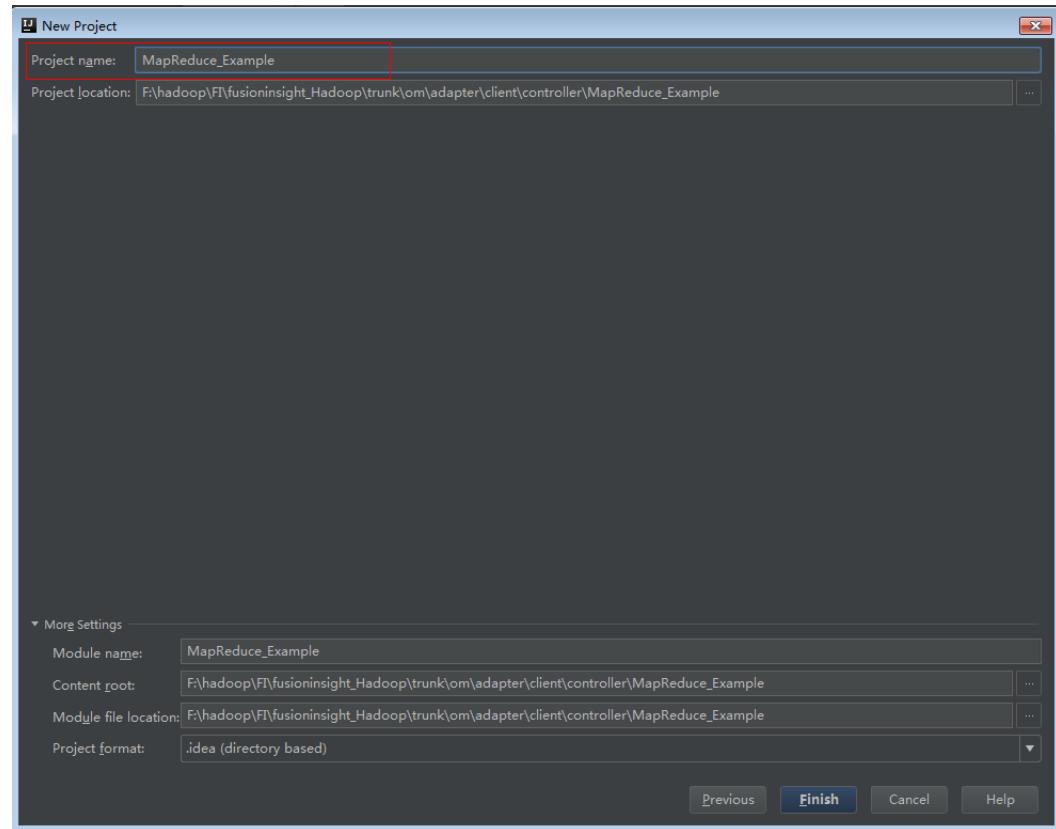
**Step 2** On the **New Project** page, select **Java**, and then configure the JDK and other Java libraries required by the project. As shown in the following figure. After the configuration is complete, click **Next**.

Figure 1-214 Configuring sdk information required by the project



**Step 3** Enter the name of the new project in the dialog box. Click **Finish**.

Figure 1-215 Enter the project name



----End

#### 1.16.2.4 Preparing the Authentication Mechanism

##### Scenario

In a safe cluster environment, the communication among components cannot be a simple communication. Components must be authorized by each other before the communication to ensure the security of the communication.

When users are developing the MapReduce application, the MapReduce is required to interwork with Yarn and HDFS in certain scenarios. Therefore, security authentication code must be written into the MapReduce application to ensure that the MapReduce application can run properly.

Two security authentication methods are described as follows:

- Authentication by running command lines:

Before submitting the MapReduce application for running, run the following command in the MapReduce client to obtain authentication:

***kinit component service user***

- Authentication by adding code:

Authenticate by obtaining principal and keytab files of the client.

## MapReduce Security Authentication Code

The security authentication is completed by calling the LoginUtil class.

In the code of the **FemaleInfoCollector** class in the **com.huawei.bigdata.mapreduce.examples** package of the sample project **MapReduce**, test@<system domain name>, user.keytab, and krb5.conf are examples. In practice, contact the admin to obtain your keytab and krb5.conf files and place them in the conf directory.):

```
public static final String PRINCIPAL= "test@<system domain name>";
public static final String KEYTAB =
FemaleInfoCollector.class.getClassLoader().getResource("user.keytab").getPath();
public static final String KRB =
FemaleInfoCollector.class.getClassLoader().getResource("krb5.conf").getPath();
...
// Security login
LoginUtil.login(PRINCIPAL, KEYTAB, KRB, conf);
```

## 1.16.3 Developing the Project

### 1.16.3.1 MapReduce Statistics Sample Project

#### 1.16.3.1.1 Typical Scenarios

##### Scenario

Develop a MapReduce application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records genders, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

**log1.txt:** logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

**log2.txt:** logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
```

```
FangBo,female,50  
LiuYang,female,20  
YuanJing,male,10  
FangBo,female,50  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files input\_data1.txt and input\_data2.txt on the Linux operating system, and copy log1.txt to input\_data1.txt and log2.txt to input\_data2.txt.
2. Create **/tmp/input** on the HDFS, and run the following commands to upload input\_data1.txt and input\_data2.txt to **/tmp/input**:
  - a. On the Linux client, run **hdfs dfs -mkdir /tmp/input**.
  - b. On the Linux client, run **hdfs dfs -put local\_file\_path /tmp/input**.

## Development Idea

Collects the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information about the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information about female netizens who spend more than 2 hours online.

### 1.16.3.1.2 Example Code

#### Function

Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.

The operation is performed in three steps:

- Filter the online time of female netizens in original files using the CollectionMapper class inherited from the Mapper abstract class.
- Count the online time of each female netizen, and output information about female netizens who dwell online for more than 2 hours using the CollectionReducer class inherited from the Reducer abstract class.
- The main method creates a MapReduce job and submits the MapReduce job to the Hadoop cluster.

#### Example Code

The following code snippets are used as an example. For complete code, see the com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector class.

Example 1: The CollectionMapper class defines the map() and setup() methods of the Mapper abstract class.

```
public static class CollectionMapper extends  
    Mapper<Object, Text, Text, IntWritable> {  
  
    // Delimiter.  
    String delim;  
    // Filter sex.  
    String sexFilter;  
  
    // Name.  
    private Text nameInfo = new Text();  
  
    // Output <key,value> must be serialized.  
    private IntWritable timeInfo = new IntWritable(1);  
  
    /**  
     * Distributed computing  
     *  
     * @param key Object: location offset of the source file.  
     * @param value Text: a row of characters in the source file.  
     * @param context Context: output parameter.  
     * @throws IOException , InterruptedException  
     */  
    public void map(Object key, Text value, Context context)  
        throws IOException, InterruptedException  
    {  
  
        String line = value.toString();  
  
        if (line.contains(sexFilter))  
        {  
  
            // A character string that has been read.  
            String name = line.substring(0, line.indexOf(delim));  
            nameInfo.set(name);  
            // Obtain the dwell duration.  
            String time = line.substring(line.lastIndexOf(delim) + 1,  
                line.length());  
            timeInfo.set(Integer.parseInt(time));  
  
            // The Map task outputs a key-value pair.  
            context.write(nameInfo, timeInfo);  
        }  
    }  
    /**  
     * map use to init.  
     *  
     * @param context Context.  
     */  
    public void setup(Context context) throws IOException,  
        InterruptedException  
    {  
  
        // Obtain configuration information using Context.  
        delim = context.getConfiguration().get("log.delimiter", ",");  
  
        sexFilter = delim  
            + context.getConfiguration()  
            .get("log.sex.filter", "female") + delim;  
    }  
}
```

Example 2: The CollectionReducer class defines the reduce() method of the Reducer abstract class.

```

public static class CollectionReducer extends
    Reducer<Text, IntWritable, Text, IntWritable>
{
    // Statistical results.
    private IntWritable result = new IntWritable();

    // Total time threshold.
    private int timeThreshold;

    /**
     * @param key Text : key after Mapper.
     * @param values Iterable : all statistical results with the same key.
     * @param context Context
     * @throws IOException , InterruptedException
     */
    public void reduce(Text key, Iterable<IntWritable> values,
                      Context context) throws IOException, InterruptedException
    {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }

        // No results are output if the time is less than the threshold.
        if (sum < timeThreshold)
        {
            return;
        }
        result.set(sum);

        // In the output information, key indicates netizen information, and value indicates the total online
        // time of the netizen.
        context.write(key, result);
    }

    /**
     * The setup() method is invoked for only once before the map() method or reduce() method.
     *
     * @param context Context
     * @throws IOException , InterruptedException
     */
    public void setup(Context context) throws IOException,
                           InterruptedException
    {
        // Context obtains configuration information.
        timeThreshold = context.getConfiguration().getInt(
            "log.time.threshold", 120);
    }
}

```

**Example 3:** Use the main() method to create a job, set parameters, and submit the job to the hadoop cluster.

```

public static void main(String[] args) throws Exception {
    // Initialize environment variables.
    Configuration conf = new Configuration();

    // Security login.
    LoginUtil.login(PRINCIPAL, KEYTAB, KRB, conf);

    // Obtain input parameters.
    String[] otherArgs = new GenericOptionsParser(conf, args)
        .getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: collect female info <in> <out>");
        System.exit(2);
    }
}

```

```
// Initialize the job object.  
@SuppressWarnings("deprecation")  
Job job = new Job(conf, "Collect Female Info");  
job.setJarByClass(FemaleInfoCollector.class);  
  
// Set map and reduce classes to be executed, or specify the map and reduce classes using configuration  
files.  
job.setMapperClass(CollectionMapper.class);  
job.setReducerClass(CollectionReducer.class);  
  
// Set the Combiner class. The combiner class is not used by default. Classes same as the reduce class are  
used.  
// Exercise caution when using the Combiner class. You can specify it using configuration files.  
job.setCombinerClass(CollectionReducer.class);  
  
// Set the output type of the job.  
job.setOutputKeyClass(Text.class);  
job.setOutputValueClass(IntWritable.class);  
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
  
// Submit the job to a remote environment for execution.  
System.exit(job.waitForCompletion(true) ? 0 : 1);  
}
```

Example 4: CollectionCombiner class combines the mapped data on the map side to reduce the amount of data transmitted from map to reduce.

```
/**  
 * Combiner class  
 */  
public static class CollectionCombiner extends  
Reducer<Text, IntWritable, Text, IntWritable> {  
  
    // Intermediate statistical results  
    private IntWritable intermediateResult = new IntWritable();  
  
    /**  
     * @param key    Text : key after Mapper  
     * @param values Iterable : all results with the same key in this map task  
     * @param context Context  
     * @throws IOException , InterruptedException  
     */  
    public void reduce(Text key, Iterable<IntWritable> values,  
Context context) throws IOException, InterruptedException {  
int sum = 0;  
for (IntWritable val : values) {  
sum += val.get();  
}  
  
intermediateResult.set(sum);  
  
// In the output information, key indicates netizen information,  
// and value indicates the total online time of the netizen in this map task.  
context.write(key, intermediateResult);  
}  
}
```

### 1.16.3.2 MapReduce Accessing Multi-Component Example Project

### 1.16.3.2.1 Instance

#### Scenario

The sample project illustrates how to compile MapReduce jobs to visit multiple service components in HDFS, HBase, and Hive, helping users to understand key actions such as certificating and configuration loading.

The logic of the sample project is as follows:

The input data is HDFS text file and the input file is **log1.txt**.

```
YuanJing,male,10  
GuoYijun,male,5
```

Map:

1. Obtain one row of the input data and extract the user name.
2. Query one piece of data from HBase.
3. Query one piece of data from Hive.
4. Combine the data queried from HBase and that from Hive as the output of Map as the output of Map.

Reduce:

1. Obtain the last piece of data from Map output.
2. Import the data to HBase.
3. Save the data to HDFS.

#### Data Planning

1. Create an HDFS data file.
  - a. Create a text file named **data.txt** in the Linux-based HDFS and copy the content of **log1.txt** to **data.txt**.
  - b. Run the following commands to create a directory **/tmp/examples/multi-components/mapreduce/input/** and copy the **data.txt** to the directory:
    - i. **hdfs dfs -mkdir -p /tmp/examples/multi-components/mapreduce/input/**
    - ii. **hdfs dfs -put data.txt /tmp/examples/multi-components/mapreduce/input/**
2. Create a HBase table and insert data into it.
  - a. Run the **source bigdata\_env** command on a Linux-based HBase client and run the **hbase shell** command.
  - b. Run the **create 'table1', 'cf'** command in the HBase shell to create table1 with column family cf.
  - c. Run the **put 'table1', '1', 'cf:cid', '123'** command to insert data whose rowkey is 1, column name is **cid**, and data value is **123**.
  - d. Run the **quit** command to exit the table.
3. Create a Hive table and load data to it.

- a. Run the **beeline** command on a Linux-based Hive client.
  - b. In the Hive beeline interaction window, run the **CREATE TABLE person(name STRING, gender STRING, stayTime INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as textfile;** command to create data table **person** with three fields.
  - c. In the Hive beeline interaction window, run the **LOAD DATA INPATH '/tmp/examples/multi-components/mapreduce/input/' OVERWRITE INTO TABLE person;** command to load data files to the **person** table.
  - d. Run **!q** to exit the table.
4. The data of HDFS is cleared in the preceding step. Therefore, perform 1 again.

### 1.16.3.2.2 Example Code

## Function

The functions of the sample project are as follows:

- Collect the name information from HDFS source files, query and combine data of HBase and Hive using the MultiComponentMapper class inherited from the Mapper abstract class.
- Obtain the last piece of mapped data and output to HBase and HDFS, using the MultiComponentMapper class inherited from the Reducer abstract class.
- The main function creates a MapReduce job and submits the MapReduce job to Hadoop clusters.

## Example Code

For details about code, see the class  
`com.huawei.bigdata.mapreduce.examples.MultiComponentExampl`.

Example code of the map function used by MultiComponentMapper class to define the Mapper abstract class.

```
private static class MultiComponentMapper extends Mapper<Object, Text, Text, Text> {  
    Configuration conf;  
  
    @Override protected void map(Object key, Text value, Context context) throws IOException,  
    InterruptedException {  
        conf = context.getConfiguration();  
  
        // for components that depend on Zookeeper, need provide the conf of jaas and krb5  
        // Notice, no need to login again here, will use the credentials in main function  
        String krb5 = "krb5.conf";  
        String jaas = "jaas_mr.conf";  
        // These files are uploaded at main function  
        File jaasFile = new File(jaas);  
        File krb5File = new File(krb5);  
        System.setProperty("java.security.auth.login.config", jaasFile.getCanonicalPath());  
        System.setProperty("java.security.krb5.conf", krb5File.getCanonicalPath());  
        System.setProperty("zookeeper.sasl.client", "true");  
  
        LOG.info("UGI :" + UserGroupInformation.getCurrentUser());  
  
        String name = "";  
        String line = value.toString();  
        if (line.contains("male")) {
```

```
// A character string that has been read
    name = line.substring(0, line.indexOf(","));
}
// 1. read from HBase
String hbaseData = readHBase();

// 2. read from Hive
String hiveData = readHive(name);

// The Map task outputs a key-value pair.
context.write(new Text(name), new Text("hbase:" + hbaseData + ", hive:" + hiveData));
}
```

Example code of the readHBase function.

```
private String readHBase() {
    String tableName = "table1";
    String columnFamily = "cf";
    String hbaseKey = "1";
    String hbaseValue;

    Configuration hbaseConfig = HBaseConfiguration.create(conf);
    org.apache.hadoop.hbase.client.Connection conn = null;
    try {
        // Create a HBase connection
        conn = ConnectionFactory.createConnection(hbaseConfig);
        // get table
        Table table = conn.getTable(Table.Name.valueOf(tableName));
        // Instantiate a Get object.
        Get get = new Get(hbaseKey.getBytes());
        // Submit a get request.
        Result result = table.get(get);
        hbaseValue = Bytes.toString(result.getValue(columnFamily.getBytes(), "cid".getBytes()));

        return hbaseValue;
    } catch (IOException e) {
        LOG.warn("Exception occur ", e);
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (Exception e1) {
                LOG.error("Failed to close the connection ", e1);
            }
        }
    }
    return "";
}
```

Example of the readHive function.

```
private String readHive(String name) throws IOException {
    //Load the configuration file
    Properties clientInfo = null;
    String userdir = System.getProperty("user.dir") + "/";
    InputStream fileInputStream = null;
    try {
        clientInfo = new Properties();
        String hiveclientProp = userdir + "hiveclient.properties";
        File propertiesFile = new File(hiveclientProp);
        fileInputStream = new FileInputStream(propertiesFile);
        clientInfo.load(fileInputStream);
    } catch (Exception e) {
        throw new IOException(e);
    } finally {
        if (fileInputStream != null) {
            fileInputStream.close();
        }
    }
}
```

```
        }
    }
String zkQuorum = clientInfo.getProperty("zk.quorum");
String zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
String serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
// Read this carefully:
// MapReduce can only use Hive through JDBC.
// Hive will submit another MapReduce Job to execute query.
// So we run Hive in MapReduce is not recommended.
final String driver = "org.apache.hive.jdbc.HiveDriver";

String sql = "select name,sum(stayTime) as " + "stayTime from person where name = '" + name + "'"
group by name";

StringBuilder sBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");
// in map or reduce, use 'auth=delegationToken'
sBuilder
.append(";serviceDiscoveryMode=")

.append(serviceDiscoveryMode)
.append(";zooKeeperNamespace=")
.append(zooKeeperNamespace)
.append(";auth=delegationToken;");
String url = sBuilder.toString();
Connection connection = null;
PreparedStatement statement = null;
ResultSet resultSet = null;
try {
    Class.forName(driver);
    connection = DriverManager.getConnection(url, "", "");
    statement = connection.prepareStatement(sql);
    resultSet = statement.executeQuery();

    if (resultSet.next()) {
        return resultSet.getString(1);
    }
} catch (ClassNotFoundException e) {
    LOG.warn("Exception occur ", e);
} catch (SQLException e) {
    LOG.warn("Exception occur ", e);
} finally {
    if (null != resultSet) {
        try {
            resultSet.close();
        } catch (SQLException e) {
            // handle exception
        }
    }
    if (null != statement) {
        try {
            statement.close();
        } catch (SQLException e) {
            // handle exception
        }
    }
    if (null != connection) {
        try {
            connection.close();
        } catch (SQLException e) {
            // handle exception
        }
    }
}
return "";
}
```

Example code of the reduce function used by MultiComponentReducer class to define the Reducer abstract class.

```
private static class MultiComponentReducer extends Reducer<Text, Text, Text, Text> {
    Configuration conf;

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
    InterruptedException {
        conf = context.getConfiguration();

        // for components that depend on Zookeeper, need provide the conf of jaas and krb5
        // Notice, no need to login again here, will use the credentials in main function
        String krb5 = "krb5.conf";
        String jaas = "jaas_mr.conf";
        // These files are uploaded at main function
        File jaasFile = new File(jaas);
        File krb5File = new File(krb5);
        System.setProperty("java.security.auth.login.config", jaasFile.getCanonicalPath());
        System.setProperty("java.security.krb5.conf", krb5File.getCanonicalPath());
        System.setProperty("zookeeper.sasl.client", "true");

        Text finalValue = new Text("");
        // just pick the last value as the data to save
        for (Text value : values) {
            finalValue = value;
        }

        // write data to HBase
        writeHBase(key.toString(), finalValue.toString());

        // save result to HDFS
        context.write(key, finalValue);
    }
}
```

Example of the writeHBase function.

```
private void writeHBase(String rowKey, String data) {
    String tableName = "table1";
    String columnFamily = "cf";

    try {
        LOG.info("UGI read :" + UserGroupInformation.getCurrentUser());
    } catch (IOException e1) {
        // handler exception
    }

    Configuration hbaseConfig = HBaseConfiguration.create(conf);
    org.apache.hadoop.hbase.client.Connection conn = null;
    try {
        // Create a HBase connection
        conn = ConnectionFactory.createConnection(hbaseConfig);
        // get table
        Table table = conn.getTable(TableName.valueOf(tableName));

        // create a Put to HBase
        List<Put> list = new ArrayList<Put>();
        byte[] row = Bytes.toBytes("row" + rowKey);
        Put put = new Put(row);
        byte[] family = Bytes.toBytes(columnFamily);
        byte[] qualifier = Bytes.toBytes("value");
        byte[] value = Bytes.toBytes(data);
        put.addColumn(family, qualifier, value);
        list.add(put);
        // execute Put
        table.put(list);
    } catch (IOException e) {
        LOG.warn("Exception occur ", e);
    } finally {
        if (conn != null) {
            try {

```

```
        conn.close();
    } catch (Exception e1) {
        LOG.error("Failed to close the connection ", e1);
    }
}
```

Example code: the main() function creates a job, configures the dependency and permission, and submits the job to Hadoop clusters.

```

public static void main(String[] args) throws Exception {
    //Load the hiveclient.properties configuration file
    Properties clientInfo = null;
    try {
        clientInfo = new Properties();

clientInfo.load(MultiComponentExample.class.getClassLoader().getResourceAsStream("hiveclient.properties"));
    } catch (Exception e) {
        throw new IOException(e);
    } finally {
    }

    String zkQuorum = clientInfo.getProperty("zk.quorum");
    String zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
    String serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
    String principal = clientInfo.getProperty("principal");
    String auth = clientInfo.getProperty("auth");
    String sasl_qop = clientInfo.getProperty("sasl.qop");
    String hbaseKeytab =
MultiComponentExample.class.getClassLoader().getResource("user.keytab").getPath();
    String hbaseJaas =
MultiComponentExample.class.getClassLoader().getResource("jaas_mr.conf").getPath();
    String hiveClientProperties =
MultiComponentExample.class.getClassLoader().getResource("hiveclient.properties").getPath();
    // a list of files, separated by comma
    String files = "file://" + KEYTAB + "," + "file://" + KRB + "," + "file://" + JAAS;
    files = files + "," + "file://" + hbaseKeytab;
    files = files + "," + "file://" + hbaseJaas;
    files = files + "," + "file://" + hiveClientProperties;
    // this setting will ask Job upload these files to HDFS
    config.set("tmpfiles", files);

    // clean control files before job submit
    MultiComponentExample.cleanupBeforeRun();

    // Security login
    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, PRINCIPAL, hbaseKeytab);
    LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
    LoginUtil.login(PRINCIPAL, KEYTAB, KRB, config);

    // find dependency jars for hive
    Class hiveDriverClass = Class.forName("org.apache.hive.jdbc.HiveDriver");
    Class thriftClass = Class.forName("org.apache.thrift.TException");
    Class serviceThriftCLIClass = Class.forName("org.apache.hive.service.rpc.thrift.TCLIService");
    Class hiveConfClass = Class.forName("org.apache.hadoop.hive.conf.HiveConf");
    Class hiveTransClass = Class.forName("org.apache.thrift.transport.HiveTSaslServerTransport");
    Class hiveMetaClass = Class.forName("org.apache.hadoop.hive.metastore.api.MetaException");
    Class hiveShimClass =
Class.forName("org.apache.hadoop.hive.metastore.security.HadoopThriftAuthBridge23");
    Class thriftCLIClass = Class.forName("org.apache.hive.service.cli.thrift.ThriftCLIService");

    // add dependency jars to Job
    JarFinderUtil.addDependencyJars(config, hiveDriverClass, serviceThriftCLIClass, thriftCLIClass, thriftClass,
        hiveConfClass, hiveTransClass, hiveMetaClass, hiveShimClass);

    // add hive config file
    config.addResource("hive-site.xml");
}

```

```
// add hbase config file
Configuration conf = HBaseConfiguration.create(config);

// Initialize the job object.
Job job = Job.getInstance(conf);
job.setJarByClass(MultiComponentExample.class);

// set mapper&reducer class
job.setMapperClass(MultiComponentMapper.class);
job.setReducerClass(MultiComponentReducer.class);

//set job input&output
FileInputFormat.addInputPath(job, new Path(baseDir, INPUT_DIR_NAME + File.separator + "data.txt"));
FileOutputFormat.setOutputPath(job, new Path(baseDir, OUTPUT_DIR_NAME));

// Set the output type of the job.
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);

// HBase use this utility class to add dependency jars of hbase to MR job
TableMapReduceUtil.addDependencyJars(job);

// this is mandatory when access to security HBase cluster
// HBase add security credentials to Job, and will use in map and reduce
TableMapReduceUtil.initCredentials(job);

// create Hive security credentials

StringBuilder sBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");

sBuilder.append(";serviceDiscoveryMode=").append(serviceDiscoveryMode).append(";zooKeeperNamespace=")
.append(zooKeeperNamespace)
.append(";sasl.qop=")
.append(sasl_qop)
.append(";auth=")
.append(auth)
.append(";principal=")
.append(principal)
.append(";");
String url = sBuilder.toString();
Connection connection = DriverManager.getConnection(url, "", "");
String tokenStr = ((HiveConnection) connection)
    .getDelegationToken(UserGroupInformation.getCurrentUser().getShortUserName(), PRINCIPAL);
connection.close();
Token<DelegationTokenIdentifier> hive2Token = new Token<DelegationTokenIdentifier>();
hive2Token.decodeFromUrlString(tokenStr);
// add Hive security credentials to Job
job.getCredentials().addToken(new Text("hive.server2.delegation.token"), hive2Token);
job.getCredentials().addToken(new Text(HiveAuthConstants.HS2_CLIENT_TOKEN), hive2Token);

// Submit the job to a remote environment for execution.
System.exit(job.waitForCompletion(true) ? 0 : 1);

}
```

 **NOTE**

Replace all the zkQuorum objects with the actual information about the ZooKeeper cluster nodes.

## 1.16.4 Commissioning the Application

### 1.16.4.1 Commissioning the Application in the Windows Environment

### 1.16.4.1.1 Compiling and Running the Application

#### Scenario

After the code development is complete, you can run an application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

##### NOTE

- If IBM JDK is used in the Windows environment, the application cannot be run in the Windows environment.
- Do not restart HDFS service while MapReduce application is in running status, otherwise the application will fail.

#### Running MapReduce Statistics Sample Project

**Step 1** Ensure that all JAR packages on which the sample project depends have been obtained.

**Step 2** In the IntelliJ IDEA development environment, select the LocalRunner.java project.

Right-click the project and choose **Run > LocalRunner.main()** from the shortcut menu to run the project. Click to run the related application project.

----End

#### Running MapReduce Accessing Multi-Component Example Project

**Step 1** Save the **user.keytab**, **hive-site.xml**, **hbase-site.xml**, and **hiveclient.properties** files to the **conf** directory of the project.

Create the **jaas\_mr.conf** file in the **conf** directory and add the following content (**test** is the user corresponding to **user.keytab**):

```
Client {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab="user.keytab"  
    principal="test@<system domain name>"  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

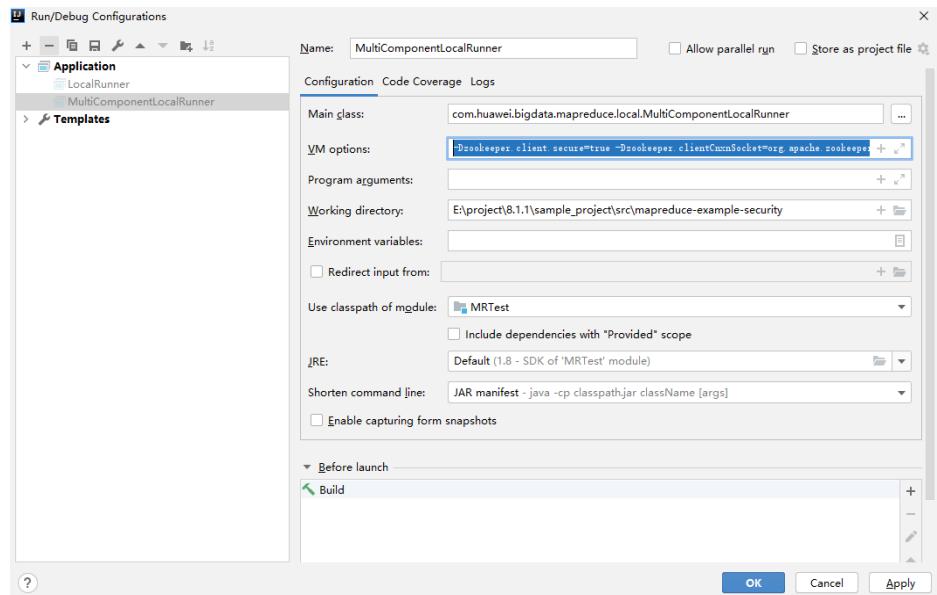
**Step 2** Ensure that all Hive and HBase JAR packages on which the sample project depends have been obtained.

**Step 3** In the IntelliJ IDEA development environment, click **MultiComponentLocalRunner.java** to run the application project. You can also right-click the project and choose **Run MultiComponentLocalRunner.main()** from the shortcut menu to run the project.

If ZooKeeper SSL is enabled for the cluster, add the -

**Dzookeeper.client.secure=true -**

**Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty** parameter to the position shown in the following figure before running this example.



----End

#### 1.16.4.1.2 Checking the Commissioning Result

##### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the program running status in IntelliJ IDEA.
- Viewing MapReduce logs.
- Logging in to the MapReduce WebUI.
- Logging in to the Yarn WebUI.

##### NOTE

You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

##### Procedure

- **Check the running result by obtaining the MapReduce log.**

As shown below, the console outputs the application running result.

```
1848 [main] INFO org.apache.hadoop.security.UserGroupInformation - Login successful for user
admin@<system domain name> using keytab file
Login success!!!!!!!!!!!!!!
7093 [main] INFO org.apache.hadoop.hdfs.PeerCache - SocketCache disabled.
9614 [main] INFO org.apache.hadoop.hdfs.DFSClient - Created HDFS_DELEGATION_TOKEN token 45
for admin on ha-hdfs:hacluster
9709 [main] INFO org.apache.hadoop.mapreduce.security.TokenCache - Got dt for hdfs://hacluster;
Kind: HDFS_DELEGATION_TOKEN, Service: ha-hdfs:hacluster, Ident:
(HDFS_DELEGATION_TOKEN token 45 for admin)
10914 [main] INFO org.apache.hadoop.yarn.client.ConfiguredRMFailoverProxyProvider - Failing over
to 53
12136 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to
process : 2
12731 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:2
13405 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job:
```

```
job_1456738266914_0006
13405 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - Kind: HDFS_DELEGATION_TOKEN,
Service: ha-hdfs:hacluster, Ident: (HDFS_DELEGATION_TOKEN token 45 for admin)
16019 [main] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Application submission is
not finished,
submitted application application_1456738266914_0006 is still in NEW
16975 [main] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application
application_1456738266914_0006
17069 [main] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: https://
linux2:26001/proxy/application_1456738266914_0006/
17086 [main] INFO org.apache.hadoop.mapreduce.Job - Running job: job_1456738266914_0006
29811 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1456738266914_0006 running in
uber mode : false
29811 [main] INFO org.apache.hadoop.mapreduce.Job - map 0% reduce 0%
41492 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 0%
53161 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 100%
53265 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1456738266914_0006 completed
successfully
53393 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 50
```

#### NOTE

In the Windows environment, the following exception occurs but does not affect services.

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

- **Check the running result by using MapReduce WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster > Name of the desired cluster > Services > Mapreduce > JobHistoryServer**. On the web page that is displayed, view the task execution status.

**Figure 1-216 JobHistory Web UI**



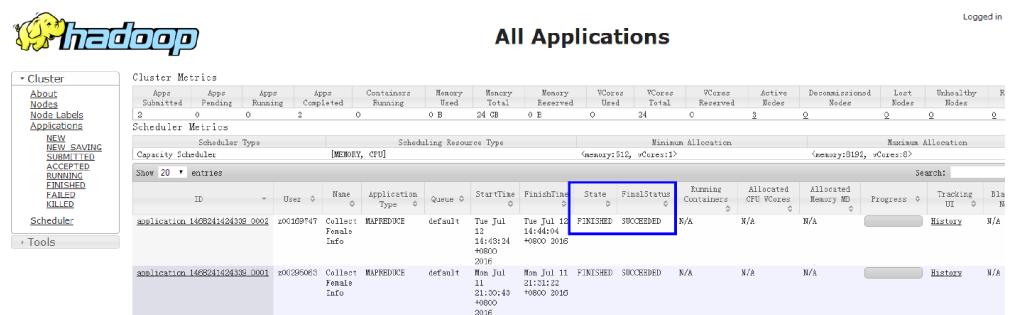
The screenshot shows a table of completed MapReduce jobs. The columns include Submit Time, Start Time, Finish Time, Job ID, Name, User, Queue, State, Maps Total, Maps Completed, Reduces Total, and Reduces Completed. Two rows are highlighted with blue boxes around their 'State' and 'Maps Completed' columns. The first row's state is 'SUCCEEDED' and maps completed are 2. The second row's state is also 'SUCCEEDED' and maps completed are 3.

| Submit Time             | Start Time              | Finish Time             | Job ID                 | Name           | User      | Queue   | State     | Maps Total | Maps Completed | Reduces Total | Reduces Completed |
|-------------------------|-------------------------|-------------------------|------------------------|----------------|-----------|---------|-----------|------------|----------------|---------------|-------------------|
| 2016-07-11 14:43:34 CST | 2016-07-11 14:43:37 CST | 2016-07-11 14:44:35 CST | job_1468241424339_0002 | Collect Female | x00109747 | default | SUCCEEDED | 2          | 2              | 1             | 1                 |
| 2016-07-11 23:30:15 CST | 2016-07-11 23:30:17 CST | 2016-07-11 23:31:22 CST | job_1468241424339_0001 | Collect Female | x00239069 | default | SUCCEEDED | 3          | 3              | 1             | 1                 |

- **Check the running result by using YARN WebUI.**

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager (Active)**. On the web page that is displayed, view the task execution status.

**Figure 1-217 ResourceManager Web UI**



The screenshot shows a table of completed applications. The columns include ID, User, Name, Application Type, Queue, Start Time, Finish Time, State, Final Status, Cores Used, Allocated CPU Minutes, Allocated Memory MB, Progress, Tracking UI, and Duration. Two rows are highlighted with blue boxes around their 'State' and 'Final Status' columns. Both rows show 'FINISHED' and 'SUCCEEDED'.

| ID                             | User      | Name           | Application Type | Queue   | Start Time               | Finish Time              | State    | Final Status | Cores Used | Allocated CPU Minutes | Allocated Memory MB | Progress | Tracking UI | Duration |
|--------------------------------|-----------|----------------|------------------|---------|--------------------------|--------------------------|----------|--------------|------------|-----------------------|---------------------|----------|-------------|----------|
| application_1468241424339_0002 | x00109747 | Collect Female | MAPREDUCE        | default | Fri Jul 15 14:43:24 2016 | Fri Jul 15 14:44:35 2016 | FINISHED | SUCCEEDED    | N/A        | N/A                   | N/A                 | N/A      | N/A         | N/A      |
| application_1468241424339_0001 | x00239069 | Collect Female | MAPREDUCE        | default | Fri Jul 15 21:00:43 2016 | Sat Jul 16 00:31:22 2016 | FINISHED | SUCCEEDED    | N/A        | N/A                   | N/A                 | N/A      | N/A         | N/A      |

- **View MapReduce logs to learn application running conditions.**

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 1.16.4.2 Commissioning an Application in the Linux Environment

### 1.16.4.2.1 Compiling and Running the Application

#### Scenario

After the code development is complete, you can run an application in the Linux development environment.

#### Prerequisites

You have installed the Yarn client.

#### Procedure

- Step 1** Go to the local root directory of the project and run the following command in Windows cmd to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```



- In the preceding command, {maven\_setting\_path} is the path of the setting.xml file of the local maven.  
• After the package is successfully packed, obtain the JAR package, for example, *MRTTest-XXX.jar*, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

- Step 2** Upload **MRTTest-XXX.jar** to the Linux client, such as **/opt/hadoopclient/conf**, the same directory where the configuration files are located in.

- Step 3** Run the sample project in the Linux environment.

- Run the following command for MapReduce statistics sample project to configure parameters and submit jobs:

```
yarn jar MRTTest-XXX.jar  
com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector  
<inputPath> <outputPath>  
<inputPath> indicates the input path and <outputPath> indicates the output path in HDFS.
```



- Before running the **yarn jar MRTTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>** command, upload the **log1.txt** and **log2.txt** files to the **<inputPath>** directory in HDFS.
- Before running the **yarn jar MRTTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>** command, ensure that the **<outputPath>** directory is deleted. Otherwise, an error will occur.
- Do not restart the HDFS service during the running of MapReduce tasks. Otherwise, the tasks may fail.

- In MapReduce accessing multi-components sample project, perform the following operations:
  - a. Obtain the **user.keytab**, **krb5.conf**, **hbase-site.xml**, **hiveclient.properties**, and **hive-site.xml** configuration files, create a folder for example **/opt/hadoopclient/conf**, and save the configurations files.

 NOTE

The account permission files **user.keytab** and **krb5.conf** are acquired from the administrator. The account permission file **hbase-site.xml** is acquired from the HBase client, **hiveclient.properties** and **hive-site.xml** are acquired from the Hive client.

- b. Create the **jaas\_mr.conf** file in the created folder. The content of the **jaas\_mr.conf** file is as follows:

```
Client {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab="user.keytab"  
    principal="test@<system domain name>"  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

 NOTE

The **test@<system domain name>** in the preceding file content is an example of user. Modify it as required.

- c. Add the classpath required for sample projects in the Linux environment, Example of classpath:

**export YARN\_USER\_CLASSPATH=/opt/hadoopclient/conf:/opt/hadoopclient/HBase/hbase/lib/\*:/opt/hadoopclient/HBase/hbase/lib/client-facing-thirdparty/\*:/opt/hadoopclient/Hive/Beeline/lib/\***

- d. Submit MapReduce jobs. Run the following command to run the sample project:

**yarn jar MRTTest-XXX.jar  
com.huawei.bigdata.mapreduce.examples.MultiComponentExample**

----End

### 1.16.4.2.2 Checking the Commissioning Result

#### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the MapReduce WebUI
- Logging in to the Yarn WebUI
- Viewing MapReduce logs

### NOTE

You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

## Procedure

- Check the running result by using MapReduce WebUI.

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster > Name of the desired cluster > Services > Mapreduce > JobHistoryServer**. On the web page that is displayed, view the task execution status.

Figure 1-218 JobHistory Web UI

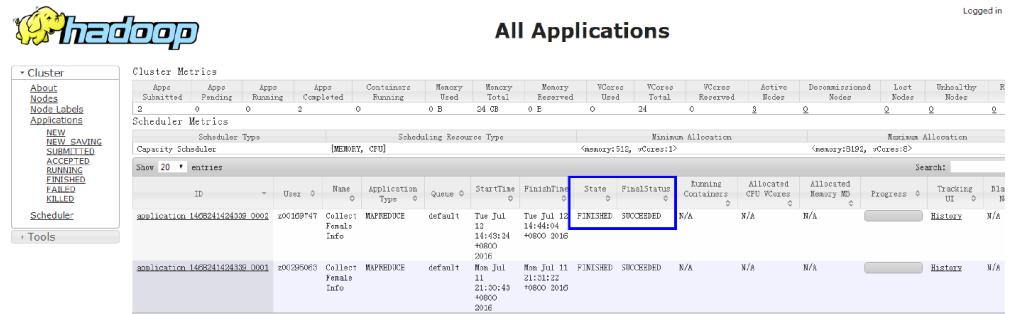


| Submit Time             | Start Time              | Finish Time             | Job ID                 | Name           | User      | Queue   | State     | Maps Total | Maps Completed | Reduces Total | Reduces Completed |
|-------------------------|-------------------------|-------------------------|------------------------|----------------|-----------|---------|-----------|------------|----------------|---------------|-------------------|
| 2016-07-11 14:14:34 CST | 2016-07-11 14:43:30 CST | 2016-07-11 14:43:30 CST | job_1408241424339_0002 | Collect Female | x00109747 | default | SUCCEEDED | 2          | 2              | 1             | 1                 |
| 2016-07-11 20:05:17 CST | 2016-07-11 20:05:17 CST | 2016-07-11 20:13:22 CST | job_1408241424339_0001 | Info           | x00295069 | default | SUCCEEDED | 2          | 2              | 1             | 1                 |

- Check the running result by using YARN WebUI.

Log in to FusionInsight Manager as a user who has the permission to view tasks and choose **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. On the web page that is displayed, view the task execution status.

Figure 1-219 ResourceManager Web UI



| ID                             | User      | Num Apps | Application Type | Queue   | StartTime                | FinishTime               | State    | FinalStatus | Running Containers | Allocated CPU | Allocated Memory MB | Progress | Tracking UI | Logs | History | W/A |
|--------------------------------|-----------|----------|------------------|---------|--------------------------|--------------------------|----------|-------------|--------------------|---------------|---------------------|----------|-------------|------|---------|-----|
| application_1408241424339_0002 | x00109747 | 0        | MAPREDUCE        | default | Tue Jul 12 14:43:24 2016 | Tue Jul 12 14:44:04 2016 | FINISHED | SHOURED     | N/A                | N/A           | N/A                 |          |             |      | History | W/A |
| application_1408241424339_0001 | x00295069 | 0        | MAPREDUCE        | default | Mon Jul 11 21:30:45 2016 | Mon Jul 11 21:31:22 2016 | FINISHED | SHOURED     | N/A                | N/A           | N/A                 |          |             |      | History | W/A |

- Check the running result of the MapReduce application.

- After running the **yarn jar MRTTest-XXX.jar** command in the Linux environment, you can check the running status of the application by the returned information about the command.

```
linux1:/opt # yarn jar MRTTest-XXXjar /user/mapred/example/input/ /output6
16/02/24 15:45:40 INFO security.UserGroupInformation: Login successful for user
admin@<system domain name> using keytab file user.keytab
Login success!!!!!!!!!!!!!!
16/02/24 15:45:40 INFO hdfs.PeerCache: SocketCache disabled.
16/02/24 15:45:41 INFO hdfs.DFSClient: Created HDFS_DELEGATION_TOKEN token 28 for
admin on ha-hdfs:hacluster
16/02/24 15:45:41 INFO security.TokenCache: Got dt for hdfs://hacluster; Kind:
HDFS_DELEGATION_TOKEN, Service: ha-hdfs:hacluster,
Ident: (HDFS_DELEGATION_TOKEN token 28 for admin)
16/02/24 15:45:41 INFO input.FileInputFormat: Total input files to process : 2
16/02/24 15:45:41 INFO mapreduce.JobSubmitter: number of splits:2
16/02/24 15:45:42 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1455853029114_0027
16/02/24 15:45:42 INFO mapreduce.JobSubmitter: Kind: HDFS_DELEGATION_TOKEN, Service: ha-
```

```
hdfs:hacluster,  
Ident: (HDFS_DELEGATION_TOKEN token 28 for admin)  
16/02/24 15:45:42 INFO impl.YarnClientImpl: Submitted application  
application_1455853029114_0027  
16/02/24 15:45:42 INFO mapreduce.Job: The url to track the job: https://linux1:26001/proxy/  
application_1455853029114_0027/  
16/02/24 15:45:42 INFO mapreduce.Job: Running job: job_1455853029114_0027  
16/02/24 15:45:50 INFO mapreduce.Job: Job job_1455853029114_0027 running in uber mode :  
false  
16/02/24 15:45:50 INFO mapreduce.Job: map 0% reduce 0%  
16/02/24 15:45:56 INFO mapreduce.Job: map 100% reduce 0%  
16/02/24 15:46:03 INFO mapreduce.Job: map 100% reduce 100%  
16/02/24 15:46:03 INFO mapreduce.Job: Job job_1455853029114_0027 completed successfully  
16/02/24 15:46:03 INFO mapreduce.Job: Counters: 49
```

- In the Linux environment, run the **yarn application -status <ApplicationID>** command to check the running result of the current application. Example:

```
linux1:/opt # yarn application -status application_1455853029114_0027  
Application Report:
```

```
Application-Id : application_1455853029114_0027  
Application-Name : Collect Female Info  
Application-Type : MAPREDUCE  
User : admin  
Queue : default  
Start-Time : 1456299942302  
Finish-Time : 1456299962343  
Progress : 100%  
State : FINISHED  
Final-State : SUCCEEDED  
Tracking-URL : https://linux1:26014/jobhistory/job/job_1455853029114_0027  
RPC Port : 27100  
AM Host : SZV1000044726  
Aggregate Resource Allocation : 114106 MB-seconds, 42 vcore-seconds  
Log Aggregation Status : SUCCEEDED  
Diagnostics : Application finished execution.  
Application Node Label Expression : <Not set>  
AM container Node Label Expression : <DEFAULT_PARTITION>
```

- **View MapReduce logs to learn application running conditions.**

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 1.16.5 More Information

### 1.16.5.1 Cross-Platform Compatibility of JAR Packages

#### Scenarios

This section describes how to configure a third-party JAR package in the MapReduce program if the package is not supported by x86 and TaiShan platforms in the cluster.

#### Procedure

- Step 1** On the Linux client, use the `checkJarsCrossPlatform.sh` script to check the cross-platform compatibility of all used JAR packages.

Run the following command to configure the cross-platform compatibility:

```
sh /opt/hadoopClient/checkJarsCrossPlatform.sh {path}
```

 NOTE

*path* is the client directory that contains all third-party JAR packages in the program.

```
sh /opt/hadoopClient/checkJarsCrossPlatformCompatibility.sh /opt/hadoopClient/lib/
4]: Succeed to check cross-platform compatibility for jars in dir.
4]: Not Compatibility For Cross-Platform Jars List is
4]: /opt/hadoopClient/lib/jansi-1.4.jar
```

- Step 2** Based on the command output in [Step 1](#), download the JAR packages not supported on the x86 and TaiShan platforms.

For example, as shown in the command output in [Step 1](#), the **jansi-1.4.jar** package is not supported by the x86 and TaiShan platforms and you need to download this package on both platforms.

- Step 3** Classify all used JAR packages shown in [Step 1](#) into following three types and compress these three types of JAR packages into .zip packages.
- public: JAR packages belonging to this type can run across platforms. Compress all JAR packages of this type to a **job-public.zip** package.
  - x86: JAR packages belonging to this type can run only on the x86 platform. Compress all JAR packages of this type to a **job-x86.zip** package.
  - TaiShan: JAR packages belonging to this type can run only on the TaiShan platform. Compress all JAR packages of this type to a **job-arm.zip** package.

- Step 4** [Accessing FusionInsight Manager of an MRS Cluster](#).

- Step 5** Choose **Cluster > Name of the desired cluster > Services > HDFS**.

- Step 6** Click **NameNode (Active)** to enter the WebUI page.

- Step 7** On the WebUI page, choose **Utilities > Browse the file system**.

- Step 8** Upload the three .zip packages **job-public.zip**, **job-x86.zip**, and **job-arm.zip** generated in [Step 3](#) to the **/tmp** directory of HDFS, as shown in [Figure 1-220](#).

**Figure 1-220** Uploading .zip packages

Browse Directory

| Browse Directory |        |        |           |               |             |            |                   |  |  |  |
|------------------|--------|--------|-----------|---------------|-------------|------------|-------------------|--|--|--|
| /tmp             |        |        |           |               |             |            |                   |  |  |  |
| Show 25 entries  |        |        |           |               |             |            |                   |  |  |  |
| Permission       | Owner  | Group  | Size      | Last Modified | Replication | Block Size | Name              |  |  |  |
| drwx-----        | hdfs   | hadoop | 0 B       | Jan 04 14:33  | 0           | 0 B        | .testHDFS         |  |  |  |
| drwxrwxrwx       | mapred | hadoop | 0 B       | Jan 02 10:12  | 0           | 0 B        | hadoop-yarn       |  |  |  |
| drwxw----        | hive   | hadoop | 0 B       | Jan 02 16:26  | 0           | 0 B        | hive              |  |  |  |
| drwxrwx---       | hive   | hive   | 0 B       | Jan 02 16:26  | 0           | 0 B        | hive-scratch      |  |  |  |
| -rw-r--r--       | super  | hadoop | 126.68 KB | Jan 04 14:33  | 3           | 128 MB     | job-arm.zip       |  |  |  |
| -rw-r--r--       | super  | hadoop | 444.18 KB | Jan 04 14:33  | 3           | 128 MB     | job-public.zip    |  |  |  |
| -rw-r--r--       | super  | hadoop | 147.26 KB | Jan 04 14:33  | 3           | 128 MB     | job-x86.zip       |  |  |  |
| drwxrwxrwt       | yarn   | hadoop | 0 B       | Jan 04 10:16  | 0           | 0 B        | logs              |  |  |  |
| drwxrwxrwx       | spark  | hadoop | 0 B       | Jan 02 16:28  | 0           | 0 B        | spark             |  |  |  |
| drwxrwxrwx       | spark  | hadoop | 0 B       | Jan 02 16:27  | 0           | 0 B        | sparkhive-scratch |  |  |  |

Showing 1 to 10 of 10 entries

Previous 1 Next

- Step 9** Add the **mapreduce.job.cache.archives** to the **mapred-site.xml** file in the **/conf** directory of the program.

```
<property>
<name>mapreduce.job.cache.archives</name>
<value>hdfs://hacluster/tmp/job-public.zip#public,hdfs://hacluster/tmp/job-x86.zip#x86,hdfs://
hacluster/tmp/job-arm.zip#arm</value>
</property>
```

- Step 10** Modify the **yarn.application.classpath** of the **yarn-site.xml** file in the **/conf** directory of the program. Specifically, add **\$PWD/public/\*,\$PWD/\$CPU\_TYPE/\*** to the value and separate them with a comma (,).

```
<property>
<name>yarn.application.classpath</name>
<value>$PWD/public/*,$PWD/$CPU_TYPE/*,$HADOOP_CONF_DIR,$HADOOP_COMMON_HOME/share/
hadoop/common/*,$HADOOP_COMMON_HOME/share/hadoop/common/lib/*,$HADOOP_HDFS_HOME/
share/hadoop/hdfs/*,$HADOOP_HDFS_HOME/share/hadoop/hdfs/lib/*,$HADOOP_YARN_HOME/share/
hadoop/mapreduce/*,$HADOOP_YARN_HOME/share/hadoop/mapreduce/lib/*,$HADOOP_YARN_HOME/
share/hadoop/yarn/*,$HADOOP_YARN_HOME/share/hadoop/yarn/lib/*,$HADOOP_YARN_HOME/share/
hadoop/tools/lib/*</value>
</property>
```

- Step 11** Compile and run the program.

----End

## 1.16.5.2 Common APIs

### 1.16.5.2.1 Java API

Directly consult official website for detailed API of MapReduce:

<http://hadoop.apache.org/docs/r3.3.1/api/index.html>

#### Common Interfaces

Common classes in MapReduce are as follows:

- org.apache.hadoop.mapreduce.Job: an interface for users to submit MR jobs and used to set job parameters, submit jobs, control job executions, and query job status.
- org.apache.hadoop.mapred.JobConf: configuration class of a MapReduce job and major configuration interface for users to submit jobs to Hadoop.

**Table 1-142** Common interfaces of org.apache.hadoop.mapreduce.Job

Interface	Description
Job(Configuration conf, String jobName), Job(Configuration conf)	Creates a MapReduce client for configuring job attributes and submitting the job.
setMapperClass(Class<extends Mapper> cls)	A core interface used to specify the Mapper class of a MapReduce job. The Mapper class is empty by default. You can also configure <b>mapreduce.job.map.class</b> in <b>mapred-site.xml</b> .
setReducerClass(Class<extends Reducer> cls)	A core interface used to specify the Reducer class of a MapReduce job. The Reducer class is empty by default. You can also configure <b>mapreduce.job.reduce.class</b> in <b>mapred-site.xml</b> .

Interface	Description
setCombinerClass(Class<extends Reducer> cls)	Specifies the Combiner class of a MapReduce job. The Combiner class is empty by default. You can also configure <b>mapreduce.job.combine.class</b> in <b>mapred-site.xml</b> . The Combiner class can be used only when the input and output key and value types of the reduce task are the same.
setInputFormatClass(Class <extends InputFormat> cls)	A core interface used to specify the InputFormat class of a MapReduce job. The default InputFormat class is <b>TextInputFormat</b> . You can also configure <b>mapreduce.job.inputformat.class</b> in <b>mapred-site.xml</b> . This interface can be used to specify the InputFormat class for processing data in different formats, reading data, and splitting data into data blocks.
setJarByClass(Class< > cls)	A core interface used to specify the local location of the JAR package of a class. Java locates the JAR package based on the class file and uploads the JAR package to the Hadoop distributed file system (HDFS).
setJar(String jar)	Specifies the local location of the JAR package of a class. You can directly set the location of a JAR package and upload the JAR package to the HDFS. Use either setJar(String jar) or setJarByClass(Class< > cls). You can also configure <b>mapreduce.job.jar</b> in <b>mapred-site.xml</b> .
setOutputFormatClass(Class<extends OutputFormat> theClass)	A core interface used to specify the OutputFormat class of a MapReduce job. The default OutputFormat class is <b>TextOutputFormat</b> . You can also configure <b>mapred.output.format.class</b> in <b>mapred-site.xml</b> . In the default <b>TextOutputFormat</b> , each key and value are recorded in text. <b>OutputFormat</b> is not specified usually.
setOutputKeyClass(Class< > theClass)	A core interface used to specify the output key type of a MapReduce job. You can also configure <b>mapreduce.job.output.key.class</b> in <b>mapred-site.xml</b> .
setOutputValueClass(Class < > theClass)	A core interface used to specify the output value type of a MapReduce job. You can also configure <b>mapreduce.job.output.value.class</b> in <b>mapred-site.xml</b> .

Interface	Description
setPartitionerClass(Class<extends Partitioner> theClass)	Specifies the Partitioner class of a MapReduce job. You can also configure <b>mapred.partitionner.class</b> in <b>mapred-site.xml</b> . This method is used to allocate Map output results to reduce classes. <b>HashPartitioner</b> is used by default, which evenly allocates the key-value pairs of a map task. For example, in HBase applications, different key-value pairs belong to different regions. In this case, you must specify the Partitioner class to allocate map output results.
setSortComparatorClass(Class<extends RawComparator> cls)	Specifies the compression class for output results of a map task. Compression is not implemented by default. You can also configure <b>mapreduce.map.output.compress</b> and <b>mapreduce.map.output.compress.codec</b> in <b>mapred-site.xml</b> . You can compress data for transmission when the map task outputs a large amount of data.
setPriority(JobPriority priority)	Specifies the priority of a MapReduce job. Five priorities can be set: <b>VERY_HIGH</b> , <b>HIGH</b> , <b>NORMAL</b> , <b>LOW</b> , and <b>VERY_LOW</b> . The default priority is <b>NORMAL</b> . You can also configure <b>mapreduce.job.priority</b> in <b>mapred-site.xml</b> .

**Table 1-143** Common interfaces of org.apache.hadoop.mapred.JobConf

Interface	Description
setNumMapTasks(int n)	A core interface used to specify the number of map tasks in a MapReduce job. You can also configure <b>mapreduce.job.maps</b> in <b>mapred-site.xml</b> . <b>NOTE</b> The InputFormat class controls the number of map tasks. Ensure that the InputFormat class supports setting the number of map tasks on the client.

Interface	Description
setNumReduceTasks(int n)	A core interface used to specify the number of reduce tasks in a MapReduce job. Only one reduce task is started by default. You can also configure <b>mapreduce.job.reduces</b> in <b>mapred-site.xml</b> . The number of reduce tasks is controlled by users. In most cases, the number of reduce tasks is one-fourth the number of map tasks.
setQueueName(String queueName)	Specifies the queue where a MapReduce job is submitted. The default queue is used by default. You can also configure <b>mapreduce.job.queuename</b> in <b>mapred-site.xml</b> .

### 1.16.5.2.2 REST API

#### Function Description

Use the HTTP REST API to view more information about MapReduce tasks. Currently, the REST API of MapResuce can be used to query the status of completed tasks. For details about the API, see the official website:

<http://hadoop.apache.org/docs/r3.3.1/hadoop-mapreduce-client/hadoop-mapreduce-client-hs/HistoryServerRest.html>

#### Preparing the Running Environment

1. Install the client, for example, to the **/opt/hadoopclient** directory on the node. For details, see section "Installing a Client."
2. Go the client installation directory and run the following commands to configure the environment variables:

```
source bigdata_env  
kinit service user
```

##### NOTE

The validity duration of kinit authentication is 24 hours. If you run the sample again 24 hours later, you need to run the kinit command again.

3. HTTPS-based access is different from HTTP-based access. When you access MapReduce using HTTPS, you must ensure that the SSL protocol supported by the curl command is supported by the cluster because SSL security encryption is used. If the cluster does not support the SSL protocol, change the SSL protocol in the cluster. For example, if the cURL supports only the TLSv1 protocol, perform the following steps:

Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Yarn > Configurations > All Configurations**, search for

**hadoop.ssl.enabled.protocols** in the search box, and check whether the parameter value contains **TLSv1**. If the parameter value does not contain **TLSv1**, add **TLSv1** in the **hadoop.ssl.enabled.protocols** configuration item. Clear the value of **ssl.server.exclude.cipher.list**. Otherwise, you cannot access Yarn using HTTPS. Click **Save**, and click **More > Restart Service** to restart the service.

 NOTE

- The values of MapReduce configuration items **hadoop.ssl.enabled.protocols** and **ssl.server.exclude.cipher.list** directly reference the values of the corresponding configuration items in Yarn. Therefore, you need to change the values of the corresponding configuration items in Yarn and restart the Yarn and MapReduce services.
- TLSv1 has security vulnerabilities. Exercise caution when using it.

## Procedure

Obtain detailed information about tasks that have been completed on MapReduce.

- Commands for the operation:

```
curl -k -i --negotiate -u : "https://10.120.85.2:26014/ws/v1/history/mapreduce/jobs"
```

In the preceding command, **10.120.85.2** indicates the value of **JHS\_FLOAT\_IP** for MapReduce, and **26014** indicates the port ID of the JobHistoryServer node.

 NOTE

In RedHat 6.x and CentOS 6.x, a compatibility problem occurs when the curl command is used to access the JobHistoryServer. As a result, the correct result cannot be returned.

- You can view the status information about historical tasks, such as the task IDs, start time, end time, and task execution status.

- Execution result

```
{
  "jobs": [
    "job": [
      {
        "submitTime":1525693184360,
        "startTime":1525693194840,
        "finishTime":1525693215540,
        "id":"job_1525686535456_0001",
        "name":"QuasiMonteCarlo",
        "queue":"default",
        "user":"mapred",
        "state":"SUCCEEDED",
        "mapsTotal":1,
        "mapsCompleted":1,
        "reducesTotal":1,
        "reducesCompleted":1
      }
    ]
  }
}
```

- Result analysis:

Using this API, you can query the completed MapReduce tasks in the current cluster and obtain information listed in **Table 1**.

**Table 1-144** Common information

Parameter	Description
submitTime	Time when a task is submitted
startTime	Start time
finishTime	End time
queue	Task queue
user	User who submits the task
state	Task state, success or failure

### 1.16.5.3 FAQ

#### 1.16.5.3.1 No Response from the Client When Submitting the MapReduce Application

##### Question

After the MapReduce task is submitted to the YARN server, the client is prompted with following information without response for a long time.

```
16/03/03 16:44:56 INFO hdfs.DFSClient: Created HDFS_DELEGATION_TOKEN token 44 for admin on ha-hdfs:hacluster
16/03/03 16:44:56 INFO security.TokenCache: Got dt for hdfs://hacluster; Kind: HDFS_DELEGATION_TOKEN, Service: ha-hdfs:hacluster, Ident: (HDFS_DELEGATION_TOKEN token 44 for admin)
16/03/03 16:44:56 INFO client.ConfiguredRMFailoverProxyProvider: Failing over to 53
16/03/03 16:44:57 INFO input.FileInputFormat: Total input files to process : 200
16/03/03 16:44:57 INFO mapreduce.JobSubmitter: number of splits:200
16/03/03 16:44:57 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1456738266914_0005
16/03/03 16:44:57 INFO mapreduce.JobSubmitter: Kind: HDFS_DELEGATION_TOKEN, Service: ha-hdfs:hacluster, Ident: (HDFS_DELEGATION_TOKEN token 44 for admin)
16/03/03 16:44:57 INFO impl.YarnClientImpl: Submitted application application_1456738266914_0005
16/03/03 16:44:57 INFO mapreduce.Job: The url to track the job: https://linux2:26001/proxy/application_1456738266914_0005/
16/03/03 16:44:57 INFO mapreduce.Job: Running job: job_1456738266914_0005
```

##### Answer

For the above problem, ResourceManager provides the key diagnosis information about MapReduce operating status on the WebUI. For the MapReduce application that is submitted to the YARN, users can obtain the current application status and the reason to be in the status with the diagnosis information.

Procedures:

Log in to FusionInsight Manager, and select **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager (Active)**. Enter the WebUI, click the submitted MapReduce application on the WebUI of ResourceManager (Active), check the diagnosis information on the WebUI, and take measures according to the diagnosis information.

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

### 1.16.5.3.2 When an Application Is Run, An Abnormality Occurs Due to Network Faults

#### Question

When an application is run in the Windows environment, the application fails to connect the cluster. While the application running in the Linux environment (the same network with the host that is installed with the MRS Cluster) works properly.

#### Answer

Kerberos authentication requires the UDP protocol. But the firewall disables the required UDP ports with special operations, resulting in that the network of the application running in the Windows environment fails to connect with the MRS Cluster. Reset the firewall, and open the UDP ports that are required, to ensure that the host running the application in the Windows environment can connect with the MRS.

### 1.16.5.3.3 How to Perform Remote Debugging During MapReduce Secondary Development?

#### Question

During the secondary development of MapReduce, how to perform remote debugging?

#### Answer

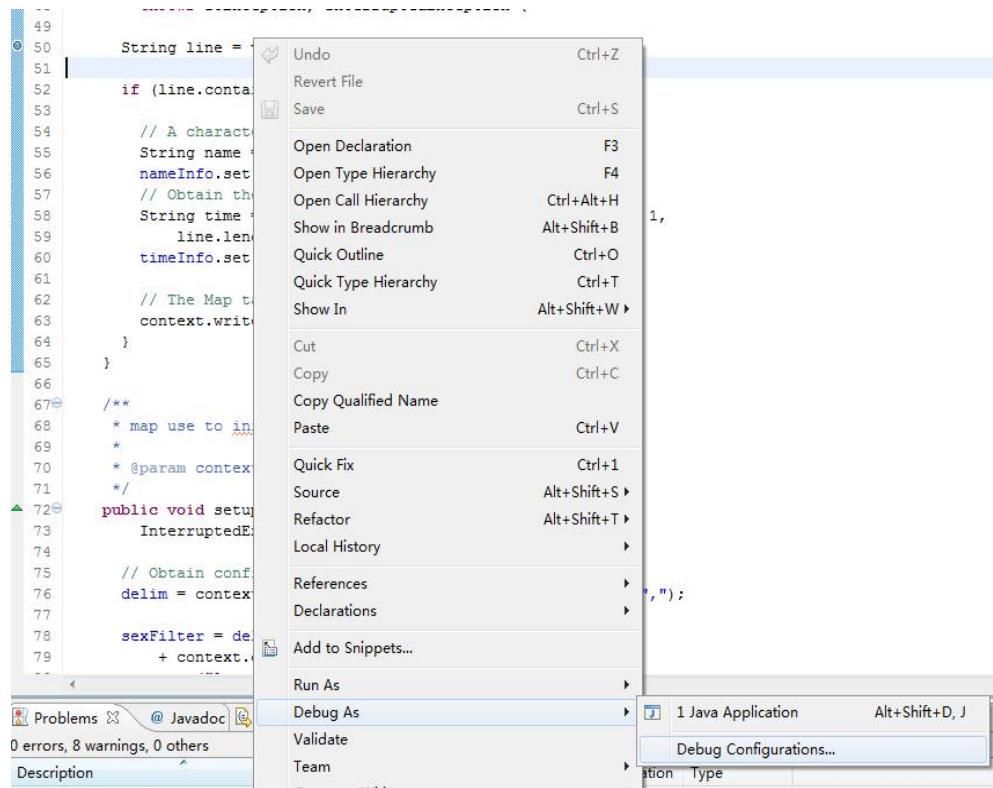
MapReduce adopts Java remote debugging mechanism. Run Java remote debugging commands when starting the Map or Reduce tasks.

- Step 1** The **mapreduce.map.java.opts** and **mapreduce.reduce.java.opts** parameters specify the JVM startup parameters of Map and Reduce tasks respectively. In the **mapred-site.xml** configuration file on the client, add the command **-agentlib:jdwp=transport=dt\_socket,server=y,suspend=y,address=8000** to the **mapreduce.map.java.opts** and **mapreduce.reduce.java.opts** parameters.
- Step 2** MapReduce is a distributed computing framework, and the node where Map or Reduce tasks are running are not fixed. It is advisable to keep only one NodeManager running in the cluster to ensure that the task is executed in the running NodeManager.
- Step 3** Submit MapReduce tasks on the client, then the Map or Reduce tasks will be suspended and listen to the port 8000 to wait for remote debugging.
- Step 4** In IDE, select the implementation class of MapReduce tasks and configure the remote debugging information to perform debugging.
  1. Double-click the area in the blue box to configure or cancel the break point.

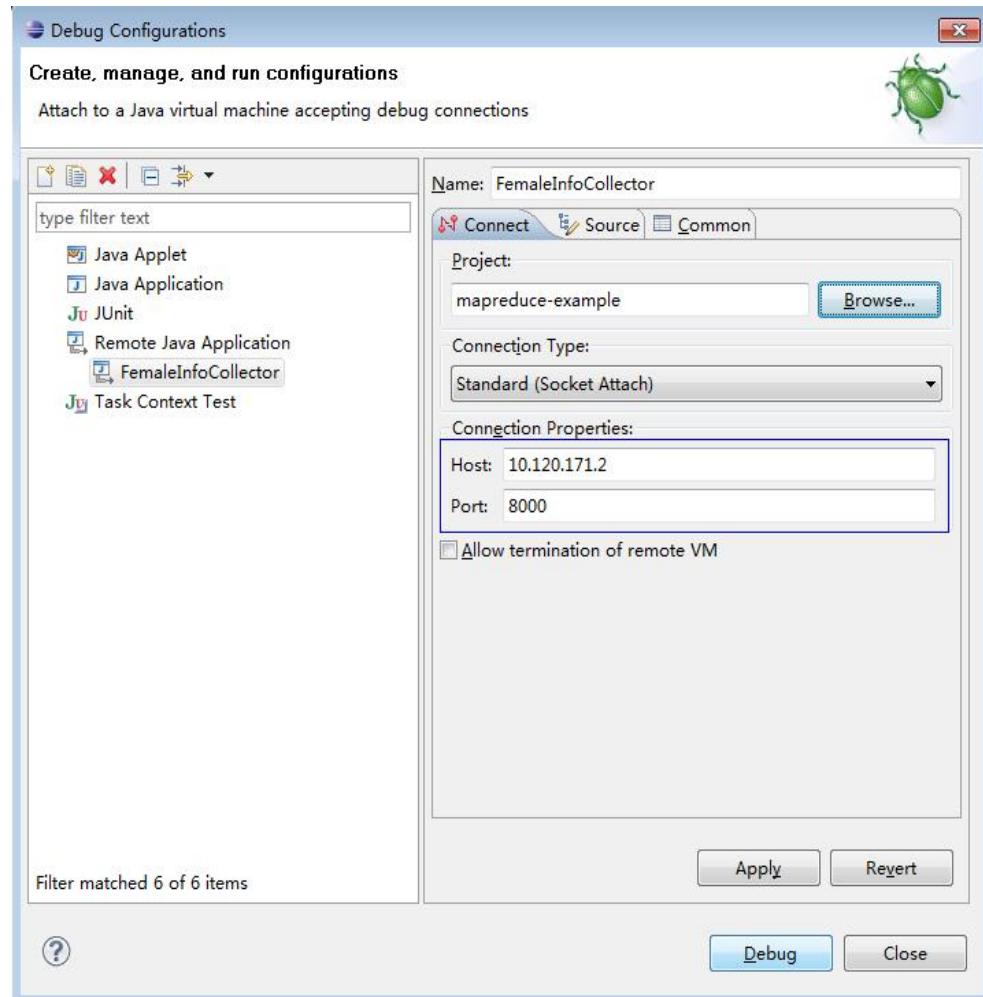
```

39    /**
40     * Distributed computing
41     *
42     * @param key      Object : location offset of the source file
43     * @param value    Text : a row of characters in the source file
44     * @param context Context : output parameter
45     * @throws IOException , InterruptedException
46     */
47     public void map(Object key, Text value, Context context)
48         throws IOException, InterruptedException {
49
50         String line = value.toString();
51
52         if (line.contains(sexFilter)) {
53
54             // A character string that has been read
55             String name = line.substring(0, line.indexOf(delim));
56             nameInfo.set(name);
57             // Obtain the dwell duration.
58             String time = line.substring(line.lastIndexOf(delim) + 1,
59                 line.length());
60             timeInfo.set(Integer.parseInt(time));
61
62             // The Map task outputs a key-value pair.
63             context.write(nameInfo, timeInfo);
64         }
65     }
66 
```

- Right-click the break point and choose **Debug As->Debug Configurations...** from the shortcut menu.



3. On the displayed page, double-click **Remote Java Application** and configure the **Connection Properties** area. Set **Host** to the IP address of the NodeManager and **Port** to **8000**, and then click **Debug**.



----End

**NOTE**

If you use IDE to submit MapReduce tasks, the IDE is the client. Modify the **mapred-site.xml** file of the secondary development project by referring to [Step 1](#).

## 1.17 Oozie Development Guide

### 1.17.1 Overview

#### 1.17.1.1 Application Development Overview

##### Oozie Introduction

Oozie is a workflow engine used to manage Hadoop jobs. Oozie workflows are defined and described based on the directed acyclic graph (DAG). Oozie supports

multiple types of workflow modes and workflow scheduled triggering mechanisms. Oozie features easy extensibility, convenient maintenance, and high reliability and works closely with each component in the Hadoop ecosystem.

Oozie workflows are classified into three types:

- Workflow  
Provides a complete basic service workflow.
- Coordinator  
Built on workflows, triggers workflows in a scheduled manner or based on conditions.
- Bundle  
Built on coordinators, centrally schedules, controls, and manages coordinators.

Oozie provides the following features:

- Supports distribution, merging, and choice workflow modes.
- Works closely with each component in the Hadoop ecosystem.
- Supports parameterized workflow variables.
- Supports scheduled workflow triggering.
- Provides a web console that allows users to view and monitor workflows and view logs.

### 1.17.1.2 Common Concepts

- **Workflow definition file**

Workflow definition files refer to XML files that describe service logic, including **workflow.xml**, **coordinator.xml**, and **bundle.xml**. Workflow definition files are parsed and executed by the Oozie engine.

- **Workflow property file**

The workflow property file refers to the parameter configuration file named **job.properties** for workflow execution. Each workflow has only one property file.

- **keytab file**

The keytab file is a key file that stores user information. In security mode, applications use the key file for API authentication.

- **Client**

Users can access the server from the client through the Java API, Shell API, REST API, or WebUI to access the Oozie server. The client in this document includes the example code used for accessing Oozie through the Java API.

- **Oozie WebUI**

Log in to the Oozie WebUI via <https://Oozie server IP address:21003/oozie>.

### 1.17.1.3 Development Process

This document describes Oozie application development based on the Java API.

For information about each phase in the development process, see [Figure 1-221](#) and [Table 1-145](#).

Figure 1-221 Oozie application development process

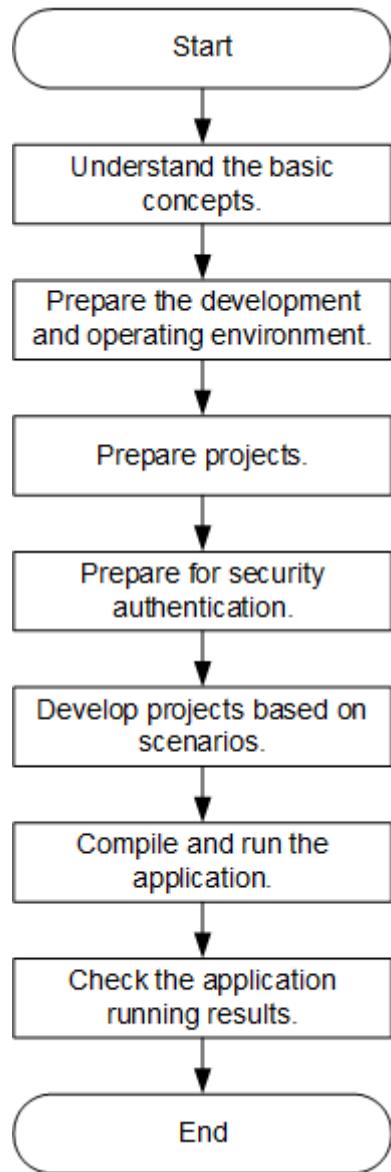


Table 1-145 Description of Oozie development process

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Oozie and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of Oozie applications. The IDEA tool can be used.	<a href="#">Preparing Development and Operating Environment</a>
Prepare projects.	Oozie provides example projects for different scenarios. You can import an example project to learn the application.	<a href="#">Downloading and Importing Sample Projects</a>
Prepare for security authentication.	If you use a security cluster, you need to perform security authentication.	<a href="#">Preparing Authentication Mechanism Code</a>
Develop projects based on scenarios.	An example project based on the Java language is provided.	<a href="#">Developing the Project</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Commissioning the Application</a>
View application running results.	Application running results are written into a specified path. You can also view the application running status on the UI.	<a href="#">Commissioning the Application</a>

## 1.17.2 Environment Preparation

### 1.17.2.1 Preparing Development and Operating Environment

[Table 1-146](#) describes the environment required for secondary development.

**Table 1-146** Development environment

Item	Description
OS	Windows OS. Windows 7 or later is supported. If the program needs to be commissioned locally, the development and running environment must be able to communicate with the cluster service plane network.

Item	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>● For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>● For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● For security purposes, the server supports only TLS V1.2 or later.</li><li>● By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>● For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IDEA installation and configuration	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● If the IBM JDK is used, ensure that the JDK configured in IDEA is the IBM JDK.</li><li>● If the Oracle JDK is used, ensure that the JDK configured in IDEA is the Oracle JDK.</li><li>● If the Open JDK is used, ensure that the JDK configured in IDEA is the Open JDK.</li><li>● Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
User development preparation	See <a href="#">Preparing a Developer Account</a> for configuration.

Item	Description
7-zip	Used to decompress .zip and .rar packages. The 7-zip 16.04 is supported.

### 1.17.2.2 Downloading and Importing Sample Projects

#### Scenario

Download sample projects and import them to the IDEA on Windows for learning.

#### Prerequisites

- If the active Oozie node and Solr are deployed on the same node, the Oozie IP address needs to be replaced with the host name to submit an Oozie task. Otherwise, the task fails to be submitted.
- A developer account, for example, **developuser**, has been prepared by following instructions in [Preparing Development and Operating Environment](#), and the user authentication credential file has been downloaded to the local host.  
The user has the common user permission of Oozie, HDFS access permission, Hive table read and write permission, HBase read and write permission, and Yarn queue submission permission.
- A complete cluster client has been installed in the Linux environment.
- The URL of a running Oozie server (any node) has been obtained. The URL is the target address to which the client submits a workflow job.

The URL format is `https://Oozie node service IP address:21003/oozie`. The port number is the value of **OOZIE\_HTTPS\_PORT**, which is **21003** by default.  
for example, `https://10.10.10.176:21003/oozie`.

#### Procedure

- Step 1** Obtain the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects from the sample project folder **ooziesecurity-examples** in the **src\oozie-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the keytab file **user.keytab** and user authentication credential file **krb5.conf** obtained in [Preparing a Developer Account](#) to the **\src\main\resources** directory of the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects.
- Step 3** In an application development environment, import the sample projects to the IDEA development environment.
1. Choose **File > Open**.  
The **Browse** dialog box is displayed.
  2. Select a sample project folder, and click **OK**.

**Step 4** Modify the parameters in the sample project. For details, see [Table 1-147](#).

**Table 1-147** Parameters to be modified

File Name	Parameter	Value	Example Value
\src\main\resources\job.properties	userName	User who submits a job.	developuser
\src\main\resources\application.properties	submit_user	User who submits a job.	developuser
	oozie_url_default	https://Oozie service IP address:21003/oozie	https://10.10.10.176:21003/oozie

**Step 5** Select the sample project to be run.

- For the **OozieMapReduceExample** sample project, go to [Step 6](#).
- For the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects, see [Scheduling Spark to Access HBase and Hive Using Oozie](#).

**Step 6** Use the client to upload the **examples** folder of Oozie to HDFS.

- Log in to the node where the client is installed, and switch to the directory of the client, for example **/opt/hadoopclient**.

**cd /opt/hadoopclient**

- Run the following command to configure environment variables:

**source bigdata\_env**

- Run the following command to perform user authentication. Change the password upon the first login.

**kinit developuser**

- Run the following commands to create a directory in HDFS and upload the sample project to the directory:

**hdfs dfs -mkdir /user/developuser**

**hdfs dfs -put -f /opt/hadoopclient/Oozie/oozie-client-\*/\* /examples /user/developuser**



In the preceding command, **oozie-client-\*** indicates the version number. Replace it with the actual version number.

----End

### 1.17.2.3 Preparing Authentication Mechanism Code

#### Scenario

In a secure cluster environment, components must perform mutual authentication before communicating with each other to ensure communication security.

In some cases, Oozie needs to communicate with Hadoop and Hive when users develop Oozie applications. Codes for security authentication need to be written into the Oozie applications to ensure that the applications can work properly.

Two security authentication modes are available:

- Command line authentication:

Before running the Oozie applications, run the following command on the Oozie client to obtain authentication:

**kinit Component service user**

- Code authentication (Kerberos security authentication):

You can contact the administrator to create and obtain keytab and krb5.conf files used for Kerberos security authentication. For details, see the sample codes.

The sample codes invoke the LoginUtil class for security authentication and support the Oracle JAVA platform and the IBM JAVA platform.

Set **userName** to the actual user name based on the actual situation, for example, **developeruser**.

```
private static void login(String keytabFilePath, String krb5FilePath, String user) throws IOException {
    Configuration conf = new Configuration();
    conf.set(KERBEROS_PRINCIPAL, user);
    conf.set(KEYTAB_FILE, keytabFilePath);
    conf.set(HADOOP_SECURITY_AUTHENTICATION, "kerberos");
    conf.set(HADOOP_SECURITY_AUTHORIZATION, "true");

    /*
     * if need to connect zk, please provide jaas info about zk. of course,
     * you can do it as below:
     * System.setProperty("java.security.auth.login.config", confDirPath +
     * "jaas.conf"); but the demo can help you more : Note: if this process
     * will connect more than one zk cluster, the demo may be not proper. you
     * can contact us for more help
     */
    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, user, keytabFilePath);
    LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
    LoginUtil.login(user, keytabFilePath, krb5FilePath, conf);
}
```

## 1.17.3 Developing the Project

### 1.17.3.1 Development of Configuration Files

#### 1.17.3.1.1 Description

#### Development Process

1. Configure the **workflow.xml** workflow configuration file (**coordinator.xml** schedules the workflow, and **bundle.xml** manages a pair of Coordinators) and **job.properties**.
2. If you want to implement codes, develop relevant jar packages, for example, Java Action. If you want to use Hive, develop SQL files.
3. Upload the configuration file and jar packages (including dependent jar packages) to the HDFS. The upload path is **oozie.wf.application.path** in **workflow.xml**.

4. The workflow can be implemented by using the following three methods. For details, see [More Information](#).
  - Shell command
  - Java API
  - Hue
5. The Oozie client provides examples for your reference, involving various Actions and how to use Coordinator and Bundle. For example, if the installation directory of the Oozie client is /opt/hadoopclient, the examples directory is **/opt/hadoopclient/Oozie/oozie-client-\*/examples**.

The following example shows you how to configure a configuration file by using the Mapreduce workflow and invoke the configuration file by running the Shell command.

## Description

Provides that a user needs to analyze website logs offline every day, and collect statistics on the access frequency of each module of the website. Log files are stored in the HDFS.

Jobs are submitted through templates and configuration files in the client.

### 1.17.3.1.2 Development Procedure

#### Step 1 Analyze the service.

1. Analyze and process logs using Mapreduce in the client example directory.
2. Move Mapreduce analysis results to the data analysis result directory, and set the data file access permission to **660**.
3. To analyze data every day, perform [Step 1.1](#) and [Step 1.2](#) every day.

#### Step 2 Implement the service.

1. Log in to the node where the client is located, and create the **dataLoad** directory, for example, **/opt/hadoopclient/Oozie/oozie-client-\*/examples/apps/dataLoad/**. This directory is used as a program running directory to store files that are edited subsequently.

#### NOTE

You can directly copy the content in the **map-reduce** directory of the example directory to the **dataLoad** directory and edit the content.

Replace **oozie-client-\*** in the directory with the actual version number.

2. Compile a workflow job property file **job.properties**.  
For details, see [job.properties](#).
3. Compile a workflow job using **workflow.xml**.

**Table 1-148** Actions in a Workflow

No.	Procedure	Description
1	Define the start action.	For details, see <a href="#">Start Action</a>

No.	Procedure	Description
2	Define the MapReduce action.	For details, see <a href="#">MapReduce Action</a>
3	Define the FS action.	For details, see <a href="#">FS Action</a>
4	Define the end action.	For details, see <a href="#">End Action</a>
5	Define the kill action.	For details, see <a href="#">Kill Action</a>

### NOTE

Dependent or newly developed JAR packages must be saved in **dataLoad/lib**.

The following provides an example workflow file:

```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">
  <start to="mr-dataLoad"/>
  <action name="mr-dataLoad">
    <map-reduce> <resource-manager>${resourceManager}</resource-manager>
      <name-node>${nameNode}</name-node>
      <prepare>
        <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce"/>
      </prepare>
      <configuration>
        <property>
          <name>mapred.job.queue.name</name>
          <value>${queueName}</value>
        </property>
        <property>
          <name>mapred.mapper.class</name>
          <value>org.apache.oozie.example.SampleMapper</value>
        </property>
        <property>
          <name>mapred.reducer.class</name>
          <value>org.apache.oozie.example.SampleReducer</value>
        </property>
        <property>
          <name>mapred.map.tasks</name>
          <value>1</value>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="copyData"/>
    <error to="fail"/>
  </action>

  <action name="copyData">
    <fs>
      <delete path='${nameNode}/user/oozie/${dataLoadRoot}/result' />
      <move source='${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce'
            target='${nameNode}/user/oozie/${dataLoadRoot}/result' />
      <chmod path='${nameNode}/user/oozie/${dataLoadRoot}/result' permissions='rwxrw-rw-' dir-files='true' />
    </fs>
  </action>
</workflow-app>
```

```
</fs>
<ok to="end"/>
<error to="fail"/>
</action>

<kill name="fail">
    <message>This workflow failed, error message[$wf:errorMessage(wf:lastErrorNode())]</
message>
    </kill>
    <end name="end"/>
</workflow-app>
```

#### 4. Compile a Coordinator job using **coordinator.xml**.

The Coordinator job is used to analyze data every day. For details, see [coordinator.xml](#).

#### Step 3 Upload the workflow file.

1. Use or switch to the user account that is granted with rights to upload files to the HDFS. For details about developer account preparation, see [Preparing Development and Operating Environment](#).
2. Implement Kerberos authentication for the user account.
3. Run the HDFS upload command to upload the **dataLoad** folder to a specified directory on the HDFS (user **developuser** must have the read/write permission for the directory).

##### NOTE

The specified directory must be the same as **oozie.coord.application.path** and **workflowAppUri** defined in **job.properties**.

#### Step 4 Execute the workflow file.

1. Log in to the client node, implement Kerberos authentication for user **developuser**.

```
cd /opt/hadoopclient
source bigdata_env
kinit developuser
```

2. Run the following command to start the workflow:

Command:

```
oozie job -oozie https://oozie server hostname:port/oozie -config
job.propertiesfile path -run
```

Parameter list:

**Table 1-149** Parameters

Parameter	Description
job	Indicates that a job is to be executed.
-oozie	Indicates the (any instance) Oozie server address.
-config	Indicates the path of <b>job.properties</b> .
-run	Indicates the starts workflow.

For example:

```
oozie job -oozie https://10-1-130-10:21003/oozie -config job.properties -run  
----End
```

### 1.17.3.2 Example Codes

#### 1.17.3.2.1 job.properties

##### Function

**job.properties** is a workflow property file that defines external parameters used for workflow execution.

##### Parameter Description

**Table 1-150** describes parameters in **job.properties**.

**Table 1-150** Parameters

Parameter	Meaning
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode cluster address.
resourceManager	Indicates the Yarn ResourceManager address.
queueName	Identifies the MapReduce queue where a workflow job is executed.
dataLoadRoot	Identifies the folder where the workflow job resides.
oozie.coord.application.path	Indicates the storage path of a Coordinator job in the HDFS.
start	Indicates the time when a scheduled workflow job is started.
end	Indicates the time when a scheduled workflow job is stopped.
workflowAppUri	Indicates the storage path of a workflow job in the HDFS.

##### NOTE

You can define parameters in the key=value format based on service requirements.

## Example Codes

```
nameNode=hdfs://hacluster  
  
resourceManager=10.1.130.10:26004  
queueName=QueueA  
dataLoadRoot=examples  
  
oozie.coord.application.path=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad  
start=2013-04-02T00:00Z  
end=2014-04-02T00:00Z  
workflowAppUri=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad
```

### 1.17.3.2.2 workflow.xml

#### Function

**workflow.xml** describes a complete service workflow. A workflow consists of a start node, an end node, and multiple action nodes.

#### Parameter Description

**Table 1-151** describes parameters in **workflow.xml**.

**Table 1-151** Parameters

Parameter	Meaning
name	Identifies a workflow file.
start	Indicates the workflow start node.
end	Indicates the workflow end node.
action	Indicates nodes (one or multiple) that are used to implement a service.

## Example Codes

```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">  
  <start to="copyData"/>  
  <action name="copyData">  
    </action>  
    .....  
  <end name="end"/>  
</workflow-app>
```

### 1.17.3.2.3 Start Action

#### Function

The Start Action node indicates the start point of a workflow job. Each workflow job has only one Start Action node.

#### Parameter Description

**Table 1-152** describes the parameter used on the Start Action node.

**Table 1-152** Parameters

Parameter	Meaning
to	Identifies a subsequent action node.

## Example Codes

```
<start to="mr-dataLoad"/>
```

### 1.17.3.2.4 End Action

#### Function

The End Action node indicates the end point of a workflow job. Each workflow job has only one End Action node.

#### Parameter Description

[Table 1-153](#) describes the parameter used on the End Action node.

**Table 1-153** Parameters

Parameter	Meaning
name	Identifies an end action.

## Example Codes

```
<end name="end"/>
```

### 1.17.3.2.5 Kill Action

#### Function

The Kill Action node indicates the end point of a workflow job when an error occurs.

#### Parameter Description

[Table 1-154](#) describes parameters used on the Kill Action node.

**Table 1-154** Parameters

Parameter	Meaning
name	Identifies a kill action.
message	Provides error messages.

Parameter	Meaning
\$ {wf:errorMessage(wf:lastErrorNode())}	Indicates the internal error message function in the Oozie system.

## Example Codes

```
<kill name="fail">
  <message>
    This workflow failed, error message[${wf:errorMessage(wf:lastErrorNode())}]
  </message>
</kill>
```

### 1.17.3.2.6 FS Action

#### Function

The FS Action node is a Hadoop distributed file system (HDFS) operation node. You can create and delete HDFS files and folders and grant permissions for HDFS files and folders using this node.

#### Parameter Description

[Table 1-155](#) describes parameters used on the FS Action node.

**Table 1-155** Parameters

Parameter	Meaning
name	Identifies an FS action.
delete	Deletes a specified file or folder.
move	Moves a file from the source directory to the target directory.
chmod	Modifies file or folder access rights.
path	Indicates the current file path.
source	Indicates the source file path.
target	Indicates the target file path.
permissions	Indicates permissions.



**\${variable name}**  indicates the value defined in  **job.properties** .

For example,  **\${nameNode}**  indicates  **hdfs://hacluster** . (See  [job.properties](#) .)

## Example Codes

```
<action name="copyData">
  <fs>
    <delete path='${nameNode}/user/oozie_cli/${dataLoadRoot}/result' />
    <move source='${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce' target='${nameNode}/user/oozie_cli/${dataLoadRoot}/result' />
    <chmod path='${nameNode}/user/oozie_cli/${dataLoadRoot}/result' permissions='rwxrw-rw-' dir-files='true' /></chmod>
  </fs>
  <ok to="end"/>
  <error to="fail"/>
</action>
```

### 1.17.3.2.7 MapReduce Action

#### Function

The MapReduce Action node is used to execute a map-reduce job.

#### Parameter Description

**Table 1-156** describes parameters used on the MapReduce Action node.

**Table 1-156** Parameters

Parameter	Meaning
name	Identifies a map-reduce action.
resourceManager	Indicates the MapReduce ResourceManager address.
name-node	Indicates the Hadoop distributed file system (HDFS) NameNode address.
queueName	Identifies the MapReduce queue where a job is executed.
mapred.mapper.class	Identifies the Mapper class.
mapred.reducer.class	Identifies the Reducer class.
mapred.input.dir	Indicates the input directory of MapReduce processed data.
mapred.output.dir	Indicates the output directory of MapReduce processing results.
mapred.map.tasks	Indicates the number of map tasks.

#### NOTE

**\${variable name}** indicates the value defined in **job.properties**.

For example,  **\${nameNode}** indicates **hdfs://hacluster**. (See [job.properties](#).)

## Example Codes

```
<action name="mr-dataLoad">
    <map-reduce>
        <resource-manager>${resourceManager}</resource-manager>
        <name-node>${nameNode}</name-node>
        <prepare>
            <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce"/>
        </prepare>
        <configuration>
            <property>
                <name>mapred.job.queue.name</name>
                <value>${queueName}</value>
            </property>
            <property>
                <name>mapred.mapper.class</name>
                <value>org.apache.oozie.example.SampleMapper</value>
            </property>
            <property>
                <name>mapred.reducer.class</name>
                <value>org.apache.oozie.example.SampleReducer</value>
            </property>
            <property>
                <name>mapred.map.tasks</name>
                <value>1</value>
            </property>
            <property>
                <name>mapred.input.dir</name>
                <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
            </property>
            <property>
                <name>mapred.output.dir</name>
                <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
            </property>
        </configuration>
    </map-reduce>
    <ok to="copyData"/>
    <error to="fail"/>
</action>
```

### 1.17.3.2.8 coordinator.xml

#### Function

**coordinator.xml** is used to periodically execute a workflow job.

#### Parameter Description

[Table 1-157](#) describes parameters in **coordinator.xml**.

**Table 1-157** Parameters

Parameter	Meaning
frequency	Indicates the workflow execution interval.
start	Indicates the time when a scheduled workflow job is started.
end	Indicates the time when a scheduled workflow job is stopped.

Parameter	Meaning
workflowAppUri	Indicates the storage path of a workflow job in the HDFS.
resourceManager	Indicates the MapReduce ResourceManager address.
queueName	Identifies the MapReduce queue where a job is executed.
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode address.

#### NOTE

**\${variable name}** indicates the value defined in **job.properties**.

For example,  **\${nameNode}** indicates **hdfs://hacluster**. (See [job.properties](#).)

## Example Codes

```
<coordinator-app name="cron-coord" frequency ="${coord:days(1)}" start ="${start}" end ="${end}"  
timeZone ="UTC" xmlns="uri:oozie:coordinator:0.2">  
    <action>  
        <workflow>  
            <app-path>${workflowAppUri}</app-path>  
            <configuration>  
                <property>  
                    <name>resourceManager</name>  
                    <value>${resourceManager}</value>  
                </property>  
                <property>  
                    <name>nameNode</name>  
                    <value>${nameNode}</value>  
                </property>  
                <property>  
                    <name>queueName</name>  
                    <value>${queueName}</value>  
                </property>  
            </configuration>  
        </workflow>  
    </action>  
</coordinator-app>
```

### 1.17.3.3 Development of Java

#### 1.17.3.3.1 Description

These typical scenarios help you quickly understand the development procedure of Oozie and learn key API functions.

This example shows you how to submit a MapReduce job and query the job status by using the Java API. The sample code relates to the MapReduce job only, but the API invocation codes of other jobs are the same. The difference is that the configuration of **job.properties** in a job and that of **workflow.xml** in a workflow is different.

 NOTE

After the operations in [Downloading and Importing Sample Projects](#) are complete, you can submit MapReduce jobs and query job status through Java APIs.

### 1.17.3.3.2 Sample Code

## Function

Oozie submits a job from the run method of `org.apache.oozie.client.OozieClient` and obtains job information from `getJobInfo`.

## Sample Code

Change `OOZIE_URL_DEFALUT` in the code example to the actual host name of any Oozie node, for example, `https://10-1-131-131:21003/oozie/`.

```
public void test(String filePath) {
    try {
        UserGroupInformation.getLoginUser()
            .doAs(
                new PrivilegedExceptionAction<Void>() {
                    /**
                     * run job
                     *
                     * @return null
                     * @throws Exception exception
                     */
                    public Void run() throws Exception {
                        runJob(filePath);
                        return null;
                    }
                });
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void runJob(String filePath) throws OozieClientException, InterruptedException {
    Properties conf = getJobProperties(filePath);

    // submit and start the workflow job
    String jobId = wc.run(conf);

    logger.info("Workflow job submitted : {}" , jobId);

    // wait until the workflow job finishes printing the status every 10 seconds
    while (wc.getJobInfo(jobId).getStatus() == WorkflowJob.Status.RUNNING) {
        logger.info("Workflow job running ... {}" , jobId);
        Thread.sleep(10 * 1000);
    }

    // print the final status of the workflow job
    logger.info("Workflow job completed ... {}" , jobId);
    logger.info(String.valueOf(wc.getJobInfo(jobId)));
}

/**
 * Get job.properties File in filePath
 *
 * @param filePath file path
 * @return job.properties
 * @since 2020-09-30
 */
```

```
public Properties getJobProperties(String filePath) {
    File configFile = new File(filePath);
    if (!configFile.exists()) {
        logger.info(filePath, "{} is not exist.");
    }

    InputStream inputStream = null;

    // create a workflow job configuration
    Properties properties = wc.createConfiguration();
    try {
        inputStream = new FileInputStream(filePath);
        properties.load(inputStream);

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (inputStream != null) {
            try {
                inputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    return properties;
}
```

## Precautions

Implement the security authentication when you use the Java API to access the Oozie. For details, see section "Preparing for the Development Environment". Upload the dependent configuration file (For details about how to develop the **Workflow.xml** configuration file, see [workflow.xml](#)) and the jar package to the HDFS, and ensure that users who have passed the security authentication are granted the rights to access the relevant directory on the HDFS. (The owner of the directory is the authenticated users, or is in the same user group with the users).

### 1.17.3.4 Scheduling Spark to Access HBase and Hive Using Oozie

#### Prerequisites

Prerequisites in [Downloading and Importing Sample Projects](#) have been met.

#### Preparing a Development Environment

**Step 1** Obtain the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects from the sample project folder **ooziesecurity-examples** in the **src\oozie-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).



Currently, MRS Spark uses Scala 2.12.14.

**Step 2** Copy the keytab file **user.keytab** and user authentication credential file **krb5.conf** obtained in [Preparing a Developer Account](#) to the **\src\main\resources** directory of the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects.

**Step 3** Modify the parameters in each sample project. For details, see [Table 1-158](#).

**Table 1-158** Parameters to be modified

File Name	Parameter	Value	Example Value
src\main\resources\application.properties	submit_user	User who submits a job.	developuser
	oozie_url_default	https://Oozie service IP address:21003/oozie/	https://10.10.10.176:21003/oozie/
src\main\resources\job.properties	userName	User who submits a job.	developuser
	examplesRoot	Use the default value or change the value based on the site requirements.	myjobs
	oozie.wf.application.path	Use the default value or change the value based on the site requirements.	<p><b>NOTICE</b> \${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark Ensure that the path is the same as the path with the &lt;jar&gt; and &lt;spark-opts&gt; tags in the src\main\resources\workflow.xml file.</p>
src\main\resources\workflow.xml	<jar> </jar>	Change <b>OoizeSparkHBase-1.0.jar</b> to the actual JAR package name.	<jar>\${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark/lib/OoizeSparkHBase-1.0.jar</jar>

#### NOTE

Go to the root directory of the project, for example, D:\sample\_project\src\oozie-examples\oozie-security-examples\OozieSparkHBaseExample, and run the **mvn clean package -DskipTests** command. After the operation is successful, the package is in the target directory.

**Step 4** Create the following folders on the HDFS client in the configured path:

**hdfs dfs -mkdir -p /user/developuser/myjobs/apps/spark/lib**

**hdfs dfs -mkdir -p /user/developuser/myjobs/apps/spark/hbase**

**hdfs dfs -mkdir -p /user/developuser/myjobs/apps/spark/hive**

**Step 5** Upload the files listed in **Table 1-159** to the corresponding path.

**Table 1-159** Files to be uploaded

Initial File Path	File	Destination Path
Spark client directory (for example, /opt/hadoopclient/Spark/spark/conf)	hive-site.xml	/user/developuser/myjobs/apps/spark directory in the HDFS.
	hbase-site.xml	
Keytab file obtained in <a href="#">Preparing a Developer Account</a>	user.keytab	
	krb5.conf	
Spark client directory (for example, /opt/hadoopclient/Spark/spark/jars)	JAR package	Share HDFS /user/oozie/share/lib/spark/ directory of Oozie.
JAR package of the sample projects to be used, for example, <b>OoizeSparkHBase-1.0.jar</b>	JAR package	/user/developuser/myjobs/apps/spark/lib/ directory in the HDFS.
OozieSparkHiveExample sample project directory <b>src\main\resources</b>	workflow.xml	/user/developuser/myjobs/apps/spark/hive/ directory in the HDFS. <b>NOTE</b> Change the path of <b>spark-archive.zip</b> in <spark-opts> based on the actual HDFS file path.
OozieSparkHBaseExample sample project directory <b>src\main\resources</b>	workflow.xml	/user/developuser/myjobs/apps/spark/hbase/ directory in the HDFS. <b>NOTE</b> Change the path of <b>spark-archive.zip</b> in <spark-opts> based on the actual HDFS file path.

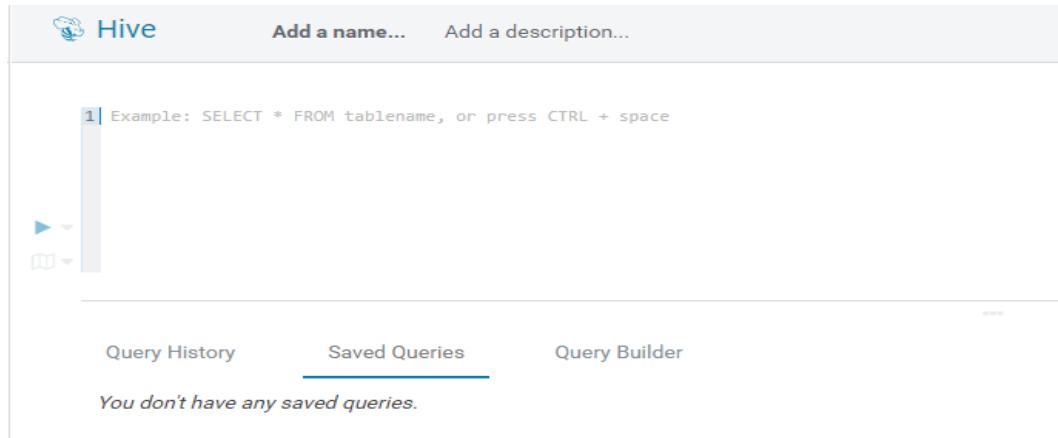
**Step 6** Change the value of **hive.security.authenticator.manager** in the **hive-site.xml** file in the **/user/developuser/myjobs/apps/spark** directory of HDFS from **org.apache.hadoop.hive ql.security.SessionStateUserMSGroupAuthenticator** to **org.apache.hadoop.hive ql.security.SessionStateUserGroupAuthenticator**.

**Step 7** If ZooKeeper SSL is enabled, add the following content to the **hbase-site.xml** file in the **/user/developuser/myjobs/apps/spark** directory of the HDFS:

```
<property>
<name>HBASE_ZK_SSL_ENABLED</name>
<value>true</value>
</property>
```

**Step 8** Run the following commands to create a Hive table:

You can enter the following SQL statements in the Hive panel on the Hue UI:



```
CREATE DATABASE test;  
  
CREATE TABLE IF NOT EXISTS `test`.`usr` (user_id int comment  
'userID',user_name string comment 'userName',age int comment  
'age')PARTITIONED BY (country string)STORED AS PARQUET;  
  
CREATE TABLE IF NOT EXISTS `test`.`usr2` (user_id int comment  
'userID',user_name string comment 'userName',age int comment  
'age')PARTITIONED BY (country string)STORED AS PARQUET;  
  
INSERT INTO TABLE test.usr partition(country='CN') VALUES(1,'maxwell',45),  
(2,'minwell',30),(3,'mike',22);  
  
INSERT INTO TABLE test.usr partition(country='USA') VALUES(4,'minbin',35);
```

**Step 9** Use HBase Shell to run the following commands to create an HBase table:

```
create 'SparkHBase',{NAME=>'cf1'}  
put 'SparkHBase','01','cf1:name','Max'  
put 'SparkHBase','01','cf1:age','23'  
----End
```

## 1.17.4 Commissioning the Application

### 1.17.4.1 Commissioning an Application in the Windows Environment

#### 1.17.4.1.1 Compiling and Running Applications

##### Scenario

After the code development is complete using Java APIs, you can run applications in the Windows development environment. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

## Procedure

- The HTTPS SSL certificate is required for running applications in Windows.
  - a. Log in to any node in the cluster and go to the following directory to download the **ca.crt** file.  
**cd \${BIGDATA\_HOME}/om-agent\_8.3.1/nodeagent/security/cert/subcert/certFile/**
  - b. Download the **ca.crt** file to a local directory and open the CLI as the administrator.  
Run the following command:  
**keytool -import -v -trustcacerts -alias ca -file "D:\xx\ca.crt" -storepass JRE truststore password -keystore "%JAVA\_HOME%\jre\lib\security\cacerts"**  
In the preceding command, **D:\xx\ca.crt** is the path for storing the **ca.crt** file. **%JAVA\_HOME %** indicates the JDK installation path. Obtain the JRE truststore password by following the instructions provided in [Huawei Cloud Stack 8.3.1 Account List](#).
- In the development environment (such as IDEA), right-click **OozieRestApiMain.java**, and choose **Run 'OozieRestApiMain.main()'** to run the application project.
- Run the following command on the Oozie client:  
**oozie job -oozie https://Oozie service IP:21003/oozie -config job.properties -run**  
You need to copy the **job.properties** file in the **src\main\resources** directory of the sample project to the directory where the Oozie client is located in advance.

### 1.17.4.1.2 Checking the Commissioning Result

#### Scenario

The results can be viewed on the console after the Oozie sample project is completed.

#### Procedure

The following information is displayed if the sample project is successful:

```
log4j:WARN No appenders could be found for logger (com.huawei.hadoop.security.LoginUtil).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/D:/temp/newClientSec/oozie-example/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/D:/temp/newClientSec/oozie-example/lib/slf4j-simple-1.7.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
current user is developuser@<system domain name> (auth:KERBEROS)
login user is developuser@<system domain name> (auth:KERBEROS)
cluset status is true
Warning: Could not get charToByteConverterClass!
Workflow job submitted: 0000071-160729120057089-oozie-omm-W
Workflow job running ...0000071-160729120057089-oozie-omm-W
```

```
Workflow job running ...0000071-160729120057089-oozie-omm-W
Workflow job completed ...0000071-160729120057089-oozie-omm-W
Workflow id[0000071-160729120057089-oozie-omm-W] status[SUCCEEDED]
-----finish Oozie -----
```

Directory **/user/developuser/examples/output-data/map-reduce** is generated on the HDFS. The directory contains the following two files:

- `_SUCCESS`
- `part-00000`

You can view the files by using the file browser of the Hue or running the following commands on the HDFS:

```
hdfs dfs -ls /user/developuser/examples/output-data/map-reduce
```

#### 📖 NOTE

In the Windows environment, the following exception may occur but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

## 1.17.5 More Information

### 1.17.5.1 Common API Introduce

#### 1.17.5.1.1 Shell

**Table 1-160** Interface parameters

Command	Parameter	Meaning
oozie version	-	The version information
oozie job	-config <arg>	Indicates the path to the job configuration file <b>job.properties</b> .
	-oozie <arg>	Indicates the Oozie server address.
	-haconfig <arg>	Indicates the path to the Oozie service configuration file <b>oozie-site.xml</b> .
	-run	Runs a job.
	-start <arg>	Starts a specified job.

Command	Parameter	Meaning
	-submit	Submits a job.
	-kill <arg>	Deletes a specified job.
	-suspend <arg>	Suspends a specified job.
	-resume <arg>	Resumes a specified job.
	-D <property=value>	Sets a property.
oozie admin	-oozie <arg>	Indicates the Oozie server address.
	-status	Indicates the service status of the Oozie service.

The Oozie command and other parameters can be found in the following address:  
[https://oozie.apache.org/docs/5.1.0/DG\\_CommandLineTool.html](https://oozie.apache.org/docs/5.1.0/DG_CommandLineTool.html).

#### 1.17.5.1.2 Java

Java APIs are provided by `org.apache.oozie.client.OozieClient`.

**Table 1-161 API**

Item	Description
public String run(Properties conf)	Runs a job.
public void start(String jobId)	Starts the specified job.
public String submit(Properties conf)	Submits a job.
public void kill(String jobId)	Delete the specified job.
public void suspend(String jobId)	Suspends the specified job.
public void resume(String jobId)	Resumes the specified job.
public WorkflowJob getJobInfo(String jobId)	Obtains job information.

#### 1.17.5.1.3 REST

The common APIs of REST are the same as the APIs of Java. For details, see  
<http://oozie.apache.org/docs/5.1.0/WebServicesAPI.html>.

## 1.18 Redis Development Guide

## 1.18.1 Overview

### 1.18.1.1 Application Development Overview

#### Redis Introduction

Redis is an open-source, high-performance key-value distributed storage database. It supports a variety of data types, making up storage limitations of Memcached and meeting requirements for real-time and high-concurrency processing. In certain application scenarios, Redis is a good supplement to relational databases.

Redis is similar to Memcached. Besides, it supports data persistence and diverse data types. Redis supports union, intersection, and complement of mathematical sets on the server side and a wide range of sorting functions.

Redis applies to the following application scenarios:

- High performance.
- Low latency.
- Enriching data structure.
- Supporting persistence.

#### Interface Type Introduction

Redis server is developed on the basis of C language. The community provides clients with common development languages and using Java language to develop Redis applications is recommended.

Redis employs the same interfaces as Jedis. For details, see <https://github.com/xetorthio/jedis>.

Redis can invoke the interfaces to provide diverse functions, as shown in **Table 1-162**.

**Table 1-162** Functions provided by Redis interfaces

Function	Description
String type access	Common operations of String in class java are: set, get, append, and strlen.
List type access	Common operations of List data structures are: lpush, rpush, lpop, rpop, llen, lindex, lrange, and lrem.
Hash type access	Common operations of Map data structures.
Set type access	Common operations of Set data structures.
Sorted Set type access	Sorted set operations

Function	Description
GEO (Geographical location)	Geographical location information can be added to the stored data

### 1.18.1.2 Common Concepts

- **keytab file**

The keytab file is a key file that stores user information. In security mode, applications use the key file for API authentication.

- **Client**

Users can access the server from the client through Java API and Redis-cli to read and write the data in Redis. The client in this document includes the example code used for accessing Redis through the Java API.

### 1.18.1.3 Development Process

This document describes Redis application development based on the Java API.

For information about each phase in the development process, see [Figure 1-222](#) and [Table 1-163](#).

Figure 1-222 Redis application development process

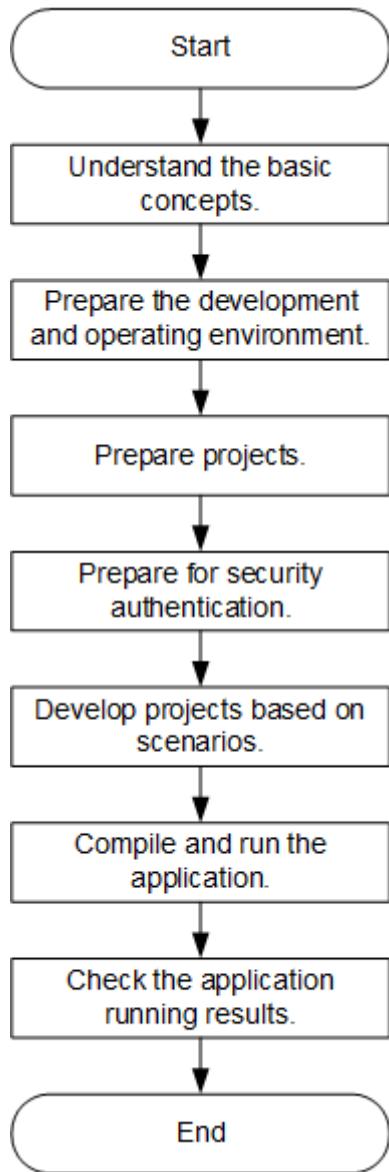


Table 1-163 Redis application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Redis and understand the scenario requirements.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of Redis applications. The IntelliJ IDEA tool can be used. The Redis running environment is the client. Install and configure the client according to the guide.	<a href="#">Development and Operating Environment</a>
Prepare projects.	Redis provides example projects for different scenarios. You can import an example project to learn the application.	<a href="#">Configuring and Importing Sample Projects</a>
Prepare for security authentication.	If you use a security cluster, you need to perform security authentication.	<a href="#">Preparing for Security Authentication</a>
Develop projects based on scenarios.	Providing the example projects of the Java language, covering Redis cluster initialization and a variety of data types access.	<a href="#">Developing an Application</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
View application running results.	Application running results will be written and output to the console.	<a href="#">Application Commissioning</a>

## 1.18.2 Environment Preparation

### 1.18.2.1 Development and Operating Environment

[Table 1-164](#) describes the development and running environment to be prepared for secondary development.

**Table 1-164** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the runtime environment must be able to communicate with the cluster service plane network.</li></ul>
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Basic configuration of the development environment. The version must be 2019.1 or other compatible versions.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you use the IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>

Item	Description
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
Redis cluster creation	<ol style="list-style-type: none"><li>1. Log in to FusionInsight Manager.</li><li>2. Click <b>Cluster</b>, click the name of the desired cluster, choose <b>Services &gt; Redis</b>, and click <b>Redis Manager</b>.</li><li>3. Click <b>Create Redis Cluster</b>.</li><li>4. Enter a cluster name and click <b>Next</b>.</li><li>5. Select all the available Redis instances contained in the Redis cluster.</li><li>6. Click <b>Submit</b>. In the displayed <b>Create Redis Cluster</b> window, enter the administrator password and click <b>OK</b>.</li></ol> <p><b>NOTE</b> Skip these steps if such a cluster containing all the Redis instances already exists.</p>
User development preparation	Prepare a cluster user for application development and grant permissions to the user by referring to <a href="#">Preparing a Developer Account</a> .
7-Zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 version is supported.

## Preparing Operating Environment

During application development, prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning application; configure the network connection, and commission the application in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) ,In the upper right corner of the home page, click **Download Client**.Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file, and then decompress it to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Redis\FusionInsight-redis-\*.tar.gz\redis\config** and manually import the configuration file to

the configuration file directory (usually the **resources\config** folder) of the Redis sample project.

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-165](#) describes the main configuration files.

**Table 1-165** Configuration files

File	Function
redis.conf	Configures Redis parameters.
log4j.xml	Configures logs.
auth.conf	Provides authentication configuration when Redis is connected.
aof-backup.properties	Provides configuration for backing up Redis data.
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Provides Kerberos server configuration information.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- When configuring IPv6 hosts on the local PC running Windows, add **%** and **InterfaceIndex** of the network interface card (NIC) to the end of the IPv6 address. The **InterfaceIndex** can be obtained by running the **ipconfig** command.

Example:

fec1:0:0:e505:8:99:5:1%10

- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client in a directory, for example, **/opt/hadoopclient**, on the node.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. **Accessing FusionInsight Manager of an MRS Cluster.** Download the cluster client software package to the active management node and decompress it. Then log in to the active management node as user **root**. Go to the decompression directory of the cluster client, and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Redis/FusionInsight-redis-\*.tar.gz/redis/config** directory to the client node to the **config** directory where the compiled JAR package is stored, for example, **/opt/hadoopclient/config**, for subsequent commissioning.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Redis/FusionInsight-redis-*.tar.gz/redis/config/* root@IP address of the client node::/opt/hadoopclient/config
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-166](#) describes the main configuration files.

**Table 1-166** Configuration files

File	Function
redis.conf	Configures Redis parameters.
log4j.xml	Configures logs.
auth.conf	Provides authentication configuration when jedisclient is connected.
aof-backup.properties	Provides configuration for backing up Redis data.
user.keytab	Provides user information for Kerberos security authentication.
krb5.conf	Provides Kerberos server configuration information.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client is located, to ensure that the local host can communicate with each host in the cluster.

## 1.18.2.2 Configuring and Importing Sample Projects

### Background

The Redis development example project is included in the client installation directory. Import the project to IntelliJ IDEA for learning.

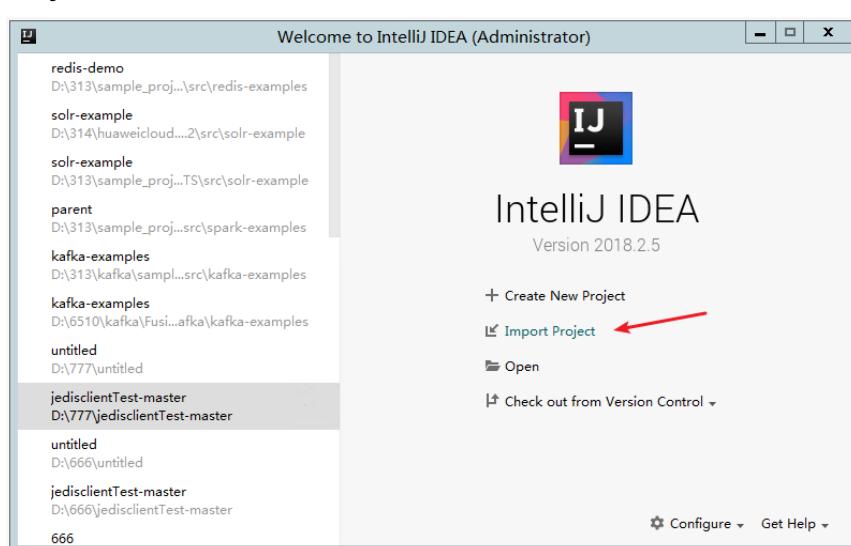
### Prerequisites

Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator.

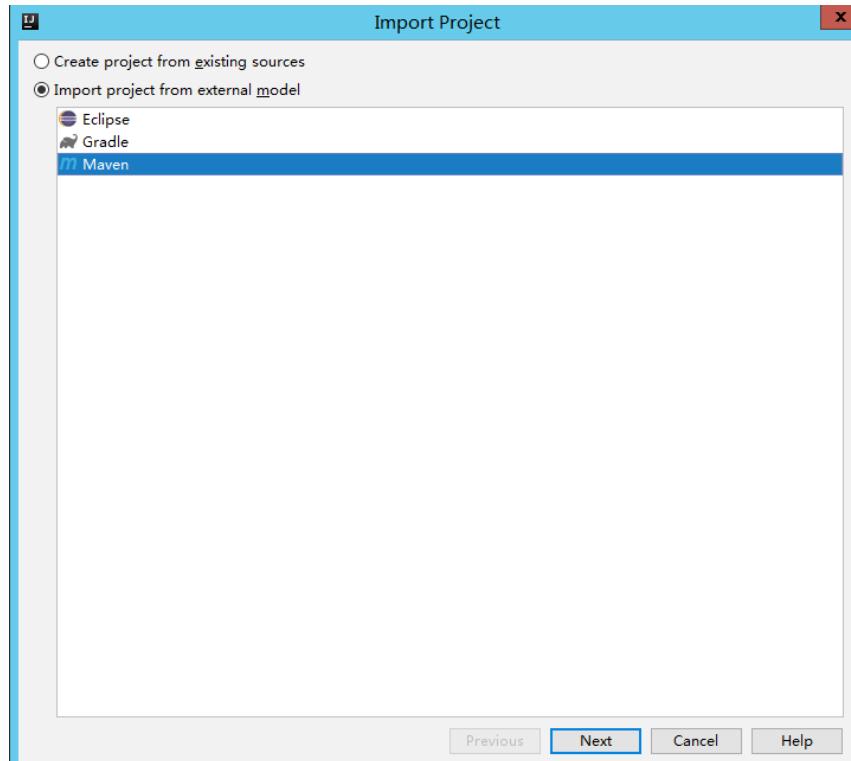
You can view the MRS cluster time in the lower-right corner on FusionInsight Manager.

### Procedure

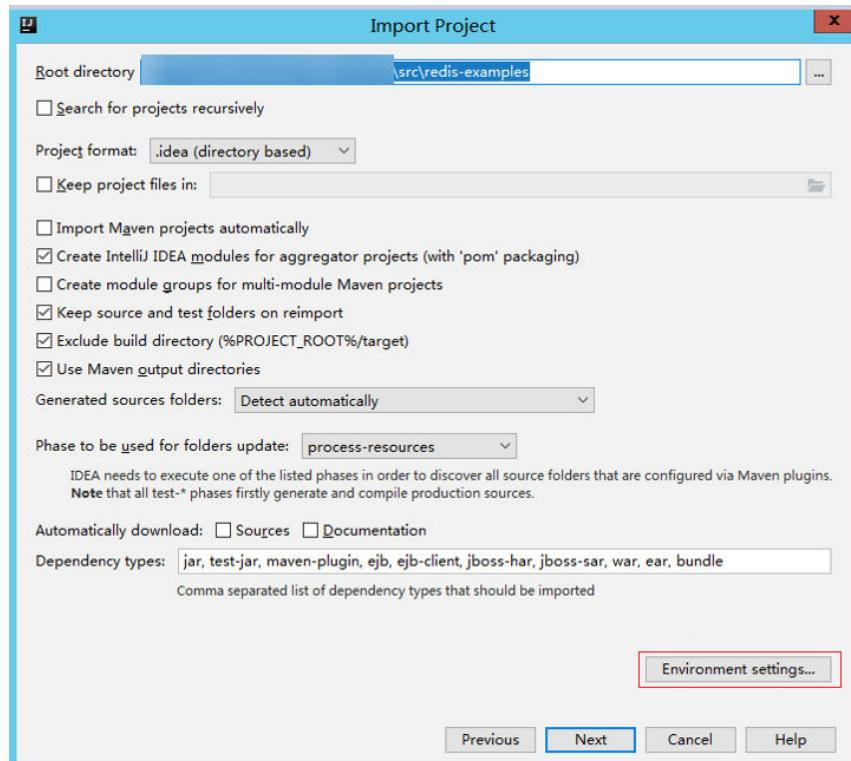
- Step 1** Obtain the sample project folder in the `/src/redis-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the `user.keytab` and `krb5.conf` files obtained in [Preparing a Developer Account](#) section to the `/src/main/resources/config` directory of the sample project.
- Step 3** Import the example project to the IntelliJ IDEA development environment.
  1. Start the IntelliJ IDEA, select **Import Project** on the **Quick Start** page.  
Or, for the used IDEA tool, add projects directly from the IDEA homepage. Select **File > Close Project** back to the **Quick Start** page and select **Import Project** .



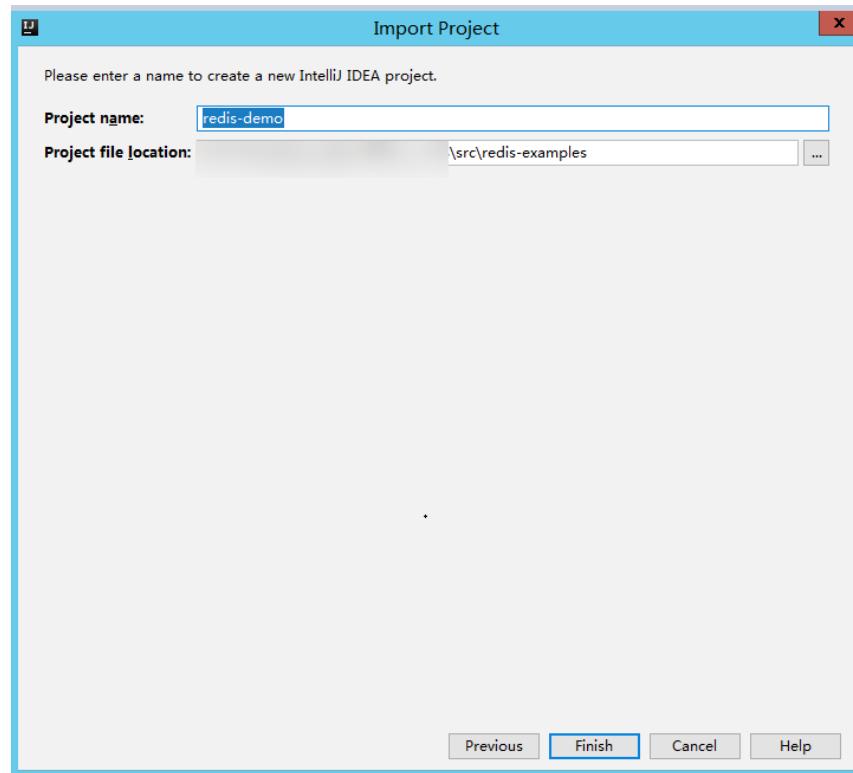
2. Select the example project folder, and click **OK**.
3. On the **Import Project** page that is displayed, select **Import project from external model and Maven**, and click **Next**.



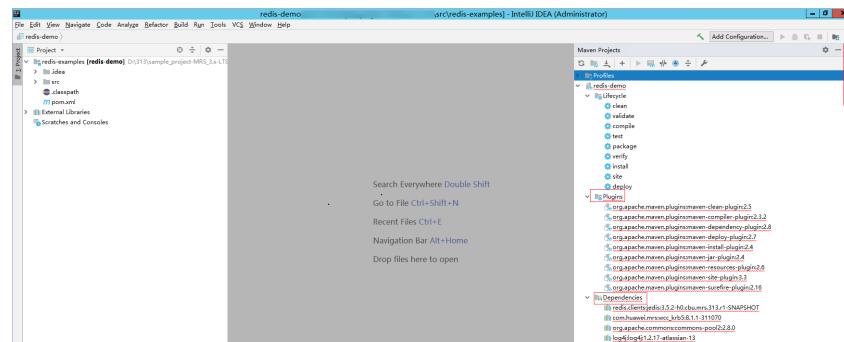
4. Click **Environment settings** to configure the environment information. Select the local Maven version. Set **User settings file** and **Local repository** based on the site requirements, click **OK**, and then click **Next**.



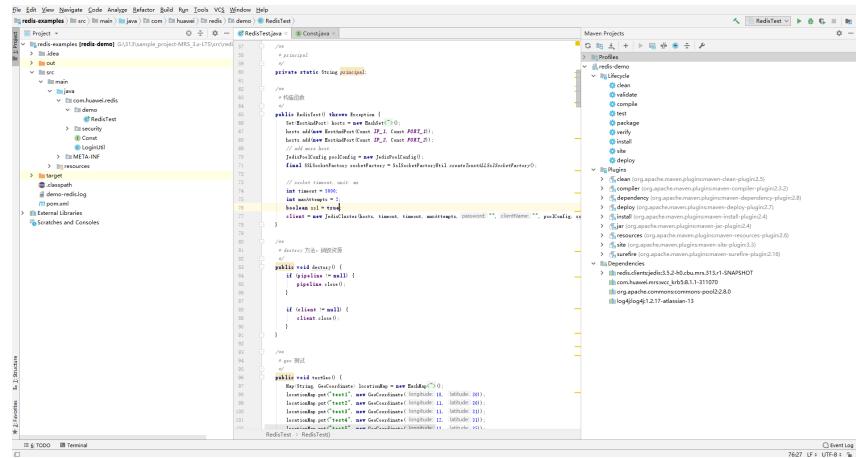
5. Confirm the subsequent configurations and click **Next**. Retain the default values unless otherwise required.
6. Enter the **project name**, select the **Project file location**, and click **Finish**.



**Step 4** In the menu bar on the right, click **Maven Project** and check whether the plug-in is loaded.



**Step 5** Click **Install** under the current project to execute the program.



----End

## 1.18.2.3 Preparing for Security Authentication

### 1.18.2.3.1 Preparing Authentication Mechanism Code

#### Scenario

In a secure cluster environment, components must perform mutual authentication before communicating with each other to ensure communication security. Redis application development requires Kerberos security authentication. You can contact the administrator to create and obtain keytab and krb5.conf files used for Kerberos security authentication. For details about how to use, see the description in the example codes.

Security authentication uses the code authentication mode. This example project applies to the Oracle Java platform and the IBM Java platform.

#### Initializing security configurations

Set system properties **redis.authentication.jaas** to **true**.

Set **principal** to the actual user name based on the actual situation, for example, **redisuser1@**.

The following code snippet belongs to the **init** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public static void init() throws IOException {
    System.setProperty("redis.authentication.jaas", "true");
    if (System.getProperty("redis.authentication.jaas", "false").equals("true")) {
        LoginUtil.setKrb5Config(getResource("config/krb5.conf"));
        principal = "redisuser1@" + KerberosUtil.getKrb5DomainRealm();
        LoginUtil.setJaasFile(principal, getResource("config/user.keytab"));
    }
}
```

## 1.18.3 Developing an Application

### 1.18.3.1 Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the Redis development process and know important interface functions.

## Scenarios

The service operation object of the Redis is key. Operations covered by example codes include access operations for keys of the String, List, Hash, Set, and Sorted Set types.

Example codes are described in the following sequence:

- Redis cluster initialization
- String type access
- List type access
- Hash type access
- Set type access
- Sorted Set type access
- Usage of Basic GEO APIs
- Use of Pipelines in the Single-thread Scenario
- Use of Pipelines in the Multi-thread Scenario
- Use of Pipelines in the bytes data Scenario

### 1.18.3.2 Example Code Description

#### 1.18.3.2.1 Redis Cluster Initialization

## Function

Create JedisCluster instances by specifying the IP addresses and port IDs of one or more instances in a cluster.

## Example Code

The following code snippet belongs to the **RedisTest** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public RedisTest() {  
  
    Set<HostAndPort> hosts = new HashSet<HostAndPort>();  
    hosts.add(new HostAndPort(Const.IP_1, Const.PORT_1));  
    hosts.add(new HostAndPort(Const.IP_2, Const.PORT_2));  
    // add more host...  
  
    // socket timeout(connect, read), unit: ms  
  
    JedisPoolConfig poolConfig = new JedisPoolConfig();  
    final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();  
    int timeout = 5000;  
    int maxAttempts = 2;  
    // Enabling channel encryption for Redis, the value of boolean SSL is changed to true  
    boolean ssl = false;  
    client = new JedisCluster(hosts, timeout, timeout, maxAttempts, "", "", poolConfig, ssl, socketFactory,
```

```
null, null, null);  
}
```

#### NOTE

- On FusionInsight Manager, users can view the Redis instances included in the Redis clusters.
- In the Redis cluster, the port ID corresponding to role Redis\_1 is 22400, and the port ID corresponding to role Redis\_2 is 22401. The others follow the same rule.
- Set the values, such as Const.IP\_1, Const.IP\_2, Const.PORT\_1, and Const.PORT\_2 in the **Const** class of the **com.huawei.redis** package, to the IP addresses and port IDs in the actual environment.
- Enabling channel encryption for Redis

Log in to FusionInsight Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for **REDIS\_SSL\_ON**, and set the parameter value to **true** to enable SSL channel encryption for Redis.

```
public interface Const {  
  
    String IP_1 = "Redis_1 node IP address";  
    String IP_2 = "Redis_2 node IP address";  
  
    int PORT_1 = 22400;  
    int PORT_2 = 22401;  
}
```

### 1.18.3.2.2 String Type Access

#### Function

Implement setting and obtaining data of simple String type in Redis, through the methods such as **set** and **get** in the JedisCluster instance.

#### Example Code

The following code snippets belong to the **testString** method in the RedisTest class of the **com.huawei.redis.demo** package.

```
public void testString() {  
    String key = "sid-user01";  
  
    // Save user's session ID, and set expire time  
    client.setex(key, 5, "A0BC9869FBC92933255A37A1D21167B2");  
    String sessionId = client.get(key);  
    LOGGER.info("User " + key + ", session id: " + sessionId);  
    try {  
        Thread.sleep(10000);  
    } catch (InterruptedException e) {  
        LOGGER.warn("InterruptedException");  
    }  
  
    sessionId = client.get(key);  
    LOGGER.info("User " + key + ", session id: " + sessionId);  
  
    key = "message";  
  
    client.set(key, "hello");  
    String value = client.get(key);  
    LOGGER.info("Value: " + value);  
  
    client.append(key, " world");  
    value = client.get(key);  
    LOGGER.info("After append, value: " + value);  
}
```

```
    client.del(key);  
}
```

## Precautions

- All types of key in Redis are case-sensitive.
- **sessionId** is a variable name. In actual use, sensitive data may be stored in Redis. Therefore, you are not advised to directly print the data read from Redis.

### 1.18.3.2.3 List Type Access

#### Function

Implement setting and obtaining data of List type in the Redis, through the methods such as rpush and lpop in the JedisCluster instance.

#### Example Code

The following code snippet belongs to the **testList** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testList() {  
    String key = "messages";  
  
    // Right push  
    client.rpush(key, "Hello how are you?");  
    client.rpush(key, "Fine thanks. I'm having fun with redis.");  
    client.rpush(key, "I should look into this NOSQL thing ASAP");  
  
    // Fetch all data  
    List<String> messages = client.lrange(key, 0, -1);  
    LOGGER.info("All messages: " + messages);  
  
    long len = client.llen(key);  
    LOGGER.info("Message count: " + len);  
  
    // Fetch the first element and delete it from list  
    String message = client.lpop(key);  
    LOGGER.info("First message: " + message);  
    len = client.llen(key);  
    LOGGER.info("After one pop, message count: " + len);  
  
    client.del(key);  
}
```

### 1.18.3.2.4 Hash Type Access

#### Function

Implement setting and obtaining data of Hash type in the Redis, through the methods such as hset, hget, hmset and hgetall in the JedisCluster instance.

#### Example Code

The following code snippet belongs to the **testHash** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testHash() {
    String key = "userinfo-001";

    // like Map.put()
    client.hset(key, "id", "J001");
    client.hset(key, "name", "John");
    client.hset(key, "gender", "male");
    client.hset(key, "age", "35");
    client.hset(key, "salary", "1000000");

    // like Map.get()
    String id = client.hget(key, "id");
    String name = client.hget(key, "name");
    LOGGER.info("User " + id + "'s name is " + name);

    Map<String, String> user = client.hgetAll(key);
    LOGGER.info(user);
    client.del(key);

    key = "userinfo-002";
    Map<String, String> user2 = new HashMap<String, String>();
    user2.put("id", "L002");
    user2.put("name", "Lucy");
    user2.put("gender", "female");
    user2.put("age", "25");
    user2.put("salary", "200000");
    client.hmset(key, user2);
    client.hincrBy(key, "salary", 50000);
    id = client.hget(key, "id");
    String salary = client.hget(key, "salary");
    LOGGER.info("User " + id + "'s salary is " + salary);

    // like Map.keySet()
    Set<String> keys = client.hkeys(key);
    LOGGER.info("all fields: " + keys);
    // like Map.values()
    List<String> values = client.hvals(key);
    LOGGER.info("all values: " + values);

    // Fetch some fields
    values = client.hmget(key, "id", "name");
    LOGGER.info("partial field values: " + values);

    // like Map.containsKey()
    boolean exist = client.hexists(key, "gender");
    LOGGER.info("Exist field gender? " + exist);

    // like Map.remove();
    client.hdel(key, "age");
    keys = client.hkeys(key);
    LOGGER.info("after del field age, rest fields: " + keys);

    client.del(key);
}
```

### 1.18.3.2.5 Set Type Access

#### Function

Implement setting and obtaining of Set type in the Redis, through the methods such as sadd and smember in the JedisCluster instance.

#### Example Code

The following code snippet belongs to the **testSet** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testSet() {
    String key = "sets";

    client.sadd(key, "HashSet");
    client.sadd(key, "SortedSet");
    client.sadd(key, "TreeSet");

    // like Set.size()
    long size = client.scard(key);
    LOGGER.info("Set size: " + size);

    client.sadd(key, "SortedSet");
    size = client.scard(key);
    LOGGER.info("Set size: " + size);

    Set<String> sets = client.smembers(key);
    LOGGER.info("Set: " + sets);

    client.srem(key, "SortedSet");
    sets = client.smembers(key);
    LOGGER.info("Set: " + sets);

    boolean ismember = client.sismember(key, "TreeSet");
    LOGGER.info("TreeSet is set's member: " + ismember);

    client.del(key);
}
```

### 1.18.3.2.6 Sorted Set Type Access

#### Function

Implement setting and obtaining data of Sorted Set type in the Redis, through the methods such as zadd and zrange in the JedisCluster instance.

#### Example Code

The following code snippet belongs to the **testSortedSet** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testSortedSet() {
    String key = "hackers";

    // Score: age
    client.zadd(key, 1940, "Alan Kay");
    client.zadd(key, 1953, "Richard Stallman");
    client.zadd(key, 1965, "Yukihiro Matsumoto");
    client.zadd(key, 1916, "Claude Shannon");
    client.zadd(key, 1969, "Linus Torvalds");
    client.zadd(key, 1912, "Alan Turing");

    // sort by score, ascending order
    Set<String> setValues = client.zrange(key, 0, -1);
    LOGGER.info("All hackers: " + setValues);

    long size = client.zcard(key);
    LOGGER.info("Size: " + size);

    Double score = client.zscore(key, "Linus Torvalds");
    LOGGER.info("Score: " + score);

    long count = client.zcount(key, 1960, 1969);
    LOGGER.info("Count: " + count);

    // sort by score, descending order
```

```
Set<String> setValues2 = client.zrevrange(key, 0, -1);
LOGGER.info("All hackers 2: " + setValues2);

client.zrem(key, "Linus Torvalds");
setValues = client.zrange(key, 0, -1);
LOGGER.info("All hackers: " + setValues);

client.del(key);
}
```

### 1.18.3.2.7 Usage of Basic GEO APIs

#### Function

This section describes how to use basic GEO APIs, such as the geoadd, geodist, and georadius methods of the `JedisCluster` instance.

#### Example Code

The following code snippet belongs to the `testGeo` method in the `RedisTest` class of the `com.huawei.redis.demo` package.

```
public void testGeo() {
    Map<String,GeoCoordinate> locationMap = new HashMap<String, GeoCoordinate>();
    locationMap.put("test1",new GeoCoordinate(10,30));
    locationMap.put("test2",new GeoCoordinate(11,30));
    locationMap.put("test3",new GeoCoordinate(11,31));
    locationMap.put("test4",new GeoCoordinate(12,32));
    locationMap.put("test5",new GeoCoordinate(13,35));

    client.geoadd("location",locationMap);

    double dis = client.geodist("location","test1","test2", GeoUnit.KM);
    LOGGER.info("The distance between test1 and test2 is " + dis + " KM.");

    List<GeoCoordinate> geoCoordinateList = client.geopos("location","test1","test2");
    LOGGER.info("Geo location info is " + geoCoordinateList.toString());

    List<String> hashInfo = client.geohash("location" , "test1" , "test2");
    LOGGER.info("geohash info of test1 and test2 is " + hashInfo.toString());

    List<GeoRadiusResponse> memberInfoByNumber = client.georadius("location" , 11 , 31 , 500 ,
GeoUnit.KM, GeoRadiusParam.geoRadiusParam().withDist());
    LOGGER.info("Get location info by longitude and latitude : ");
    for(GeoRadiusResponse geoRadiusResponse : memberInfoByNumber){
        LOGGER.info(geoRadiusResponse.getMemberByString() + ":" + geoRadiusResponse.getDistance());
    }

    List<GeoRadiusResponse> memberInfoByLocation = client.georadiusByMember("location" , "test3" ,
500 , GeoUnit.KM, GeoRadiusParam.geoRadiusParam().sortAscending().withDist().count(3));
    LOGGER.info("Get location info by member : ");
    for(GeoRadiusResponse geoRadiusResponse : memberInfoByLocation){
        LOGGER.info(geoRadiusResponse.getMemberByString() + ":" + geoRadiusResponse.getDistance());
    }
}
```

### 1.18.3.2.8 Use of Pipelines in the Single-thread Scenario

#### Function

The pipeline (pipeline) function of Redis can achieve the efficient operation in Redis, and `JedisCluster` can read and write cluster data.

#### Example Code

For scenarios that do not require return values and require all return values and some return values, use the corresponding interface in the sample code. The example code is applicable to single-thread scenarios.

```
public void testPipeline() {
    HostAndPort host1 = new HostAndPort(IP_1,PORT_1);
    HostAndPort host2 = new HostAndPort(IP_2,PORT_2);
    HashSet<HostAndPort> set = new HashSet<HostAndPort>();
    set.add(host1);
    set.add(host2);
    JedisCluster cluster = null;
    try{
        cluster = new JedisCluster(set);
        ClusterBatch pipeline = cluster.getPipeline();
        pipeline.set("key1","value1");
        pipeline.set("key2","value2");
        pipeline.set("key3","value3");

        // none returned value
        pipeline.sync();

        // all returned value
        List<Object> syncAndReturnAll = pipeline.syncAndReturnAll();

        //part returned value
        List<Object> syncAndReturn = pipeline.syncAndReturn(new String[]{"key1"});
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        if(cluster != null){
            cluster.close();
        }
    }
}
```

### 1.18.3.2.9 Use of Pipelines in the Multi-thread Scenario

#### Function

The pipeline (pipeline) function of Redis can achieve the efficient operation in Redis, and JedisCluster can read and write cluster data.

#### Example Code

For scenarios that do not require return values and require all return values and some return values, use the corresponding interface in the sample code. The example code is applicable to multi-thread scenarios.

```
public void testPipeline() {
    public static void main(String[] args){
        HostAndPort host1 = new HostAndPort(IP_1,PORT_1);
        HostAndPort host2 = new HostAndPort(IP_2,PORT_2);
        HashSet<HostAndPort> set = new HashSet<HostAndPort>();
        set.add(host1);
        set.add(host2);

        myThread t1 = new myThread(set);
        myThread t2 = new myThread(set);
        myThread t3 = new myThread(set);

        t1.start();
        t2.start();
        t3.start();
    }

    public static class myThread extends Thread{
        private JedisCluster cluster;
```

```
myThread(Set<HostAndPort> set){
    cluster = new JedisCluster(set);
}

public void run(){
    try{
        ClusterBatch pipeline = cluster.getPipeline();
        pipeline.set("key1","value1");
        pipeline.set("key2","value2");
        pipeline.set("key3","value3");

        // none returned value
        pipeline.sync();

        // all returned value
        List<Object> syncAndReturnAll = pipeline.syncAndReturnAll();

        //part returned value
        List<Object> syncAndReturn = pipeline.syncAndReturn(new String[]{"key1"});
    }catch (Exception e){
        e.printStackTrace();
    }finally {
        if(cluster != null){
            cluster.close();
        }
    }
}
```

### 1.18.3.2.10 Use of Pipelines in the bytes data Scenario

#### Function

The pipeline function of Redis can achieve the efficient operation in Redis, and JedisCluster can read and write cluster data.

#### Example Code

For scenarios that do not require return values and require all return values and some return values, use the corresponding interface in the sample code. This example applies to scenarios where input and output parameters are bytes data.

```
public void testBinaryPipeline() {
    HostAndPort host1 = new HostAndPort(IP_1,PORT_1);
    HostAndPort host2 = new HostAndPort(IP_2,PORT_2);
    HashSet<HostAndPort> set = new HashSet<HostAndPort>();
    set.add(host1);
    set.add(host2);
    JedisCluster cluster = null;
    try{
        cluster = new JedisCluster(set);
        BinaryClusterBatch pipeline = cluster.getBinaryPipeline();
        pipeline.set("key1".getBytes(),"value1".getBytes());
        pipeline.set("key2".getBytes(),"value2".getBytes());
        pipeline.set("key3".getBytes(),"value3".getBytes());

        // none returned value
        pipeline.sync();

        // all returned value
        List<Object> syncAndReturnAll = pipeline.syncAndReturnAll();

        //part returned value
        byte[][] part = new byte[][]{"key1".getBytes(), "key2".getBytes()};
        List<Object> syncAndReturn = pipeline.syncAndReturn(part);
    }
```

```
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            if(cluster != null){
                cluster.close();
            }
        }
    }
```

### 1.18.3.2.11 Connection to a Redis Logical ClusterUsing Spring Boot

#### Function

Spring Boot is used to connect to a Redis logical cluster and read and write data in this cluster.

#### Example Code

The example code is used to connect to a Redis logical cluster to realize read and write in Redis.

```
@Repository
public class CustomerRedisTemplateUtil {
    protected static final Logger logger =
LoggerFactory.getLogger(CustomerRedisTemplateUtil.class.getName());
    private static JedisCluster client;
    /**
     * Default connection timeout
     */
    private static final Integer TIMEOUT = 3000;
    /**
     * Maximum retries
     */
    private static final Integer MAX_ATTEMPTS = 1;
    @Value("${spring.redis.cluster.nodes}")
    private String nodes;
    @Value("${redis.keytab}")
    private String keytab;
    @Value("${redis.user}")
    private String user;
    @Value("${redis.krb5}")
    private String krb5;
    @Value("${redis.realm}")
    private String realm;
    @Value("${redis.ssl}")
    private boolean ssl;
    @PostConstruct
    private void init() throws NoSuchAlgorithmException, KeyManagementException {
        if (Objects.isNull(client)) {
            System.setProperty("redis.authentication.jaas", "true");
            AuthConfiguration authConfig = new AuthConfiguration(krb5, keytab, user);
            GlobalConfig.setAuthConfiguration(authConfig);
            authConfig.setServerRealm(realm);
            authConfig.setLocalRealm(realm);
            logger.info("user={}, realm={}", user, realm);
            //Configure the connection pool.
            JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
            jedisPoolConfig.setMaxTotal(5);
            jedisPoolConfig.setMaxIdle(3);
            jedisPoolConfig.setMinIdle(2);
            Set<HostAndPort> hosts = new HashSet<HostAndPort>();
            for (String node : nodes.split(",")) {
                HostAndPort hp = getIpAndPort(node);
                if (hp == null) {
                    logger.warn("{} not valid", node);
                    continue;
                }
                jedisPoolConfig.addServer(hp.getHost(), hp.getPort());
            }
        }
    }
}
```

```
        }
        hosts.add(hp);
    }
    final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustALLSslSocketFactory();
    client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, TIMEOUT, MAX_ATTEMPTS, jedisPoolConfig,
ssl,
        socketFactory, authConfig);
}
}
public static HostAndPort getIpAndPort(String hostAndPortStr) {
    if (StringUtils.isBlank(hostAndPortStr)) {
        return null;
    }
    String[] hostAndPort = hostAndPortStr.split(":");
    String ipAddress;
    String port;
    if (hostAndPort.length < 2) {
        return null;
    } else {
        port = hostAndPort[hostAndPort.length - 1];
        ipAddress = hostAndPortStr.substring(0, hostAndPortStr.lastIndexOf(":"));
    }
    return new HostAndPort(ipAddress, Integer.parseInt(port));
}
public void save(String key, String value) {
    client.set(key, value);
}
public String find(String id) {
    String s = client.get(id);
    System.out.println(s);
    return s;
}
}
```

The values of **spring.redis.cluster.nodes**, **redis.user**, **redis.keytab**, **redis.krb5** and **redis.realm** need to be read from the **/src/main/resources/application.properties** configuration file of the sample project. Modify the parameters based on the site requirements.

#### NOTE

- **spring.redis.cluster.nodes** indicates the IP address and port number of the logical cluster. Multiple IP addresses and ports are separated by commas (,), for example, **IP address 1:Port number 1,IP address 2:Port number 2,IP address 3:Port number 3**.
- Obtaining the instance service IP address: Log in to FusionInsight Manager, choose **Cluster > Services > Redis**, and click **Instance** to view the IP addresses of instances running in the cluster.
- Obtaining the port number: On Redis nodes, the port number used by role **Redis\_1** is **22400**, and that used by role **Redis\_2** is **22401**. The others follow the same rule.
- **redis.user** indicates the username for connecting to the Redis logical cluster. The recommended format is **Username@Domain name**, for example, **redis\_user@HADOOP.COM**.
- **redis.keytab** and **redis.krb5** indicate the authentication credentials of the user for connecting to the Redis logical cluster, which can be obtained from the downloaded authentication credentials.
- **redis.realm** indicates the current cluster domain name. You can log in to FusionInsight Manager and choose **System > Permission > Domain** and **Mutual Trust** to view the actual domain name of the cluster.
- **redis.ssl** indicates whether to enable the channel encryption. If the channel encryption is enabled, set this parameter to **true**. Otherwise, set this parameter to **false**. To check whether channel encryption has been enabled for a cluster, perform the following operations:  
Log in to FusionInsight Manager, choose **Cluster > Services > Redis**. On the page that is displayed, click the **Configurations** tab then the **All Configurations** sub-tab. On this sub-tab page, search for **REDIS\_SSL\_ON**. If its value is **true**, SSL channel encryption has been enabled for Redis. Otherwise, channel encryption has been disabled for the Redis cluster.
- **server.port** indicates the port for accessing the Spring Boot server. The default value is **9093** and can be customized.

### 1.18.3.3 Redis Client Authentication Mode

#### 1.18.3.3.1 Usage Instruction

You can access the Redis service in security mode through the Java client provided by Huawei. Currently, four authentication modes are provided. Before the authentication, you need to create a Redis role and user on the FusionInsight Manager management page, for example, create user **redisuser**. For details, see [Development and Operating Environment](#). Then, download the authentication file of the user.

#### 1.18.3.3.2 Authentication Through the auth.conf File

You can use the **auth.conf** configuration file to access the Redis service in security mode. The sample code is as follows:

```
public class SecureJedisClusterDemo {  
    /*  
     * Entry method  
     *  
     * @param args args  
     */  
    public static void main(String[] args) {  
        Set<HostAndPort> hosts = new HashSet<HostAndPort>();  
        hosts.add(new HostAndPort(Const.IP_1, Const.PORT_1));  
        hosts.add(new HostAndPort(Const.IP_2, Const.PORT_2));  
    }  
}
```

```
// add host...
// System.setProperty("SERVER_REALM","HADOOP.COM");

JedisCluster client = new JedisCluster(hosts, 15000);
System.out.println(client.set("SecureJedisClusterDemo", "value"));
System.out.println(client.get("SecureJedisClusterDemo"));
client.del("SecureJedisClusterDemo");
client.close();
}
```

#### NOTE

- Change the values of **Const.IP\_1**, **Const.IP\_2**, **Const.PORT\_1**, and **Const.PORT\_2** in the sample code to the actual IP addresses and port numbers.
- Log in to FusionInsight Manager, choose **Cluster > Services > Redis**, and click **Instance** to view the IP addresses of instances running in the cluster.
- On Redis nodes, the port number used by role **Redis\_1** is **22400**, and that used by role **Redis\_2** is **22401**. The others follow the same rule.
- **userName** indicates the cluster user. **redisuser** is for reference only.
- **realmsName** indicates the cluster domain name. **HADOOP.COM** is for reference only. You can log in to FusionInsight Manager and choose **System > Domain and Mutual Trust** to view the actual domain name of the cluster.
- **krbConfPath** and **keyTabFile** indicate the locations of the user authentication files decompressed from **krb5.conf** and **user.keytab** on the Linux server, respectively. **/opt/test** is used as an example.
- **principal** is in the format of **Username@Domain name**. **redisuser1@HADOOP.COM** is for reference only.

Perform the following steps to run the code:

**Step 1** Create a directory on the Linux node, for example, **/opt/test**, and create the **/config/auth.conf** file. The file content is as follows:

```
userName = redisuser
storeKey = true
realmsName = HADOOP.COM
useKeyTab = true
keyTabFile = config/user.keytab
krbConfPath = config/krb5.conf
```

**Step 2** Decompress the downloaded user authentication file to obtain **krb5.conf** and **user.keytab**, and move them to the **/opt/test/config** directory.

**Step 3** Change the values of **Const.IP\_1** and **Const.PORT\_1** in the preceding sample code based on the site requirements.

**Step 4** Package the application by referring to [Commissioning an Application in Linux](#).

**Step 5** Run the following command to run the application:

```
java -cp .:lib/* com.huawei.redis.security.SecureJedisClusterDemo
```

----End

### 1.18.3.3 Global Authentication Through APIs

You can access the Redis service by setting global authentication information. The sample codes are as follows:

```
public class SecureJedisClusterDemo2 {
    public static void main(String[] args) {
```

```
System.setProperty("java.security.krb5.conf", "krb5.conf file path");
AuthConfiguration authConfiguration = new AuthConfiguration("keytab file path", "principal");
GlobalConfig.setAuthConfiguration(authConfiguration);
// System.setProperty("SERVER_REALM","HADOOP.COM");

Set<HostAndPort> hosts = new HashSet<HostAndPort>();
hosts.add(new HostAndPort(Const.IP_1, Const.PORT_1));
JedisCluster client = new JedisCluster(hosts, 5000);

client.set("test-key", System.currentTimeMillis() + "");
System.out.println(client.get("test-key"));
client.del("test-key");
client.close();
}
}
```

Perform the following operations to run the application:

- Step 1** Decompress the downloaded user authentication files to obtain **krb5.conf** and **user.keytab**, and upload them to the Linux server where the sample runs, for example, **/opt/test**.
- Step 2** Change the values of **java.security.krb5.conf**, **krb5.conf file path**, **principal**, **Const.IP\_1** and **Const.PORT\_1** in the preceding sample codes based on the site requirements.
- Step 3** Refer to the packing operations described in [Commissioning an Application in Linux](#).
- Step 4** Run the following command to run the application:

```
java -cp ..lib/* com.huawei.redis.security.SecureJedisClusterDemo2
----End
```

#### 1.18.3.3.4 Authentication Through the jaas.conf File

Use the specified **jaas.conf** file to access the Redis server. The sample codes are as follows:

```
public class SecureJedisClusterDemo3 {
    public static void main(String[] args) {
        System.setProperty("redis.authentication.jaas", "true");
        System.setProperty("java.security.auth.login.config", "jaas.conf file path");
        System.setProperty("java.security.krb5.conf", "krb5.conf file path");
        // Section name of the jaas.conf file. The value is case sensitive. The default value is Client.
        // System.setProperty("redis.sasl.clientconfig", "redisClient");
        // System.setProperty("SERVER_REALM","HADOOP.COM");
        Set<HostAndPort> hosts = new HashSet<HostAndPort>();
        hosts.add(new HostAndPort(Const.IP_1, Const.PORT_1));
        JedisCluster client = new JedisCluster(hosts, 5000);

        client.set("test-key", System.currentTimeMillis() + "");
        System.out.println(client.get("test-key"));
        client.del("test-key");
        client.close();
    }
}
```

Perform the following operations to run the application:

- Step 1** Decompress the downloaded user authentication files to obtain **krb5.conf** and **user.keytab**, and upload them to the Linux server where the sample runs, for example, **/opt/test**. Generate the **jaas.conf** file by referring to [Preparing Authentication Mechanism Code](#).

**Step 2** Modify the preceding sample codes based on the content of the **jaas.conf** file generated in [Step 1](#).

**Step 3** Refer to the packing operations described in [Commissioning an Application in Linux](#).

**Step 4** Run the following command to start the application:

```
java -cp .:lib/* com.huawei.redis.security.SecureJedisClusterDemo3
```

----End

### 1.18.3.3.5 Independent Authentication Through APIs

You can use APIs to access Redis servers using different usernames in the same application. The following sample code is used to access two Redis logical clusters at the same time in the same application. You need to download the authentication information of the two Redis logical clusters.

```
public class SecureJedisClusterDemo4 {  
  
    /**  
     * Default connection timeout  
     */  
    private static final Integer TIMEOUT = 3000;  
  
    /**  
     * Maximum retries  
     */  
    private static final Integer MAX_ATTEMPTS = 1;  
  
    public static void main(String[] args) throws KeyManagementException, NoSuchAlgorithmException {  
        // Create a Kerberos authentication object.  
        AuthConfiguration authConfiguration = new AuthConfiguration("path1/krb5.conf", "path1/  
user.keytab",  
            "user@HADOOP1.COM");  
        authConfiguration.setServerRealm("HADOOP1.COM");  
        authConfiguration.setLocalRealm("HADOOP1.COM");  
        // Create a second Kerberos authentication object.  
        AuthConfiguration authConfiguration2 = new AuthConfiguration("path2/krb5.conf", "path2/  
user.keytab",  
            "user@HADOOP.COM");  
        authConfiguration.setServerRealm("HADOOP.COM");  
        authConfiguration.setLocalRealm("HADOOP.COM");  
        // Add the IP addresses and port numbers of the cluster.  
        Set<HostAndPort> hosts = new HashSet<>();  
        hosts.add(new HostAndPort("Const.IP_1", Const.PORT_1));  
        Set<HostAndPort> hosts2 = new HashSet<>();  
        hosts2.add(new HostAndPort("Const.IP_2", Const.PORT_2));  
  
        // Initialize the connection pool.  
        JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();  
        jedisPoolConfig.setMinIdle(3);  
        jedisPoolConfig.setMaxTotal(100);  
        jedisPoolConfig.setMaxIdle(100);  
  
        // Create SSL sockets.  
        boolean ssl = false;  
        final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();  
  
        // Initialize the jedisCluster connection.  
        writeCluster(hosts, jedisPoolConfig, socketFactory, ssl, authConfiguration);  
        // Connect to the second JedisCluster.  
        writeCluster(hosts2, jedisPoolConfig, socketFactory, ssl, authConfiguration2);  
    }  
}
```

```
* Connect to a Redis cluster.  
*  
* @param hosts      Indicates the primary instance list.  
* @param jedisPoolConfig  Indicates the connection pool configuration.  
* @param socketFactory  socket factory class  
* @param ssl          Specifies whether to enable SSL.  
* @param authConfiguration  Indicates the security authentication configuration.  
*/  
private static void writeCluster(Set<HostAndPort> hosts, JedisPoolConfig jedisPoolConfig,  
    SSLSocketFactory socketFactory, boolean ssl, AuthConfiguration authConfiguration) {  
    JedisCluster client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, TIMEOUT, MAX_ATTEMPTS,  
    jedisPoolConfig, ssl,  
        socketFactory, authConfiguration);  
    client.set("test-key", System.currentTimeMillis() + "");  
    System.out.println(client.get("test-key"));  
    client.del("test-key");  
    client.close();  
}  
}
```

Perform the following operations to run the application:

- Step 1** Decompress the downloaded user authentication files of the two Redis logical clusters to obtain **krb5.conf** and **user.keytab**, and upload them to the Linux server where the sample runs, for example, **/opt/test**.
- Step 2** Change the values of **krb5.conf**, **user.keytab**, **HADOOP1.COM**, **principal**, **Const.IP\_1**, **Const.PORT\_1**, **Const.IP\_2**, and **Const.PORT\_2** in the preceding sample code based on the site requirements.
- Step 3** Package the application by referring to [Commissioning an Application in Linux](#).
- Step 4** Run the following command to run the application:

```
java -cp ..:lib/* com.huawei.redis.security.SecureJedisClusterDemo4
```

----End

### 1.18.3.4 Connection to Redis in Security Mode Using Other Languages

#### 1.18.3.4.1 Connection to Redis in Security Mode Using Python 3.x

##### Overview

This section describes how to use Python 3.x in a Linux operating system to connect to a Redis instance in security mode. Currently, Windows operating systems are not supported.

##### Prerequisites

- SSL channel encryption has been disabled.  
Log in to FusionInsight Manager and choose **Cluster > Services > Redis**. Click **Configurations** then **All Configurations**, search for **REDIS\_SSL\_ON**, and set this parameter to **false**, and click **Save**. Restart the service for the configuration to take effect.
- Create a Redis role and user on FusionInsight Manager, for example, create **redisuser**. For details, see [Development and Operating Environment](#). Download the authentication file of the user.

- This section takes a CentOS 7.6 running on a x86 server as an example. The **python-gssapi** package for Python Kerberos authentication is required. You need to install the authentication package and set up other basic Python environments. To install Kerberos authentication packages, perform the following steps:

**Step 1** Run the following command to install the packages on the client to be authenticated:

```
yum install cyrus-sasl cyrus-sasl-lib cyrus-sasl-gssapi cyrus-sasl-plain krb5-devel krb5-libs
```

**Step 2** Run the following command to install the Python basic package on the client to be authenticated:

```
yum install python3-devel python-kadmin
```

**Step 3** Run the following command to install the Python packages required for Kerberos authentication on the client to be authenticated:

```
pip3 install gssapi subprocess struct redis
```

**Step 4** Install the client of the current cluster to a specified directory, for example, **/opt/hadoopclient**. For details, see [Preparing Operating Environment](#). Run the following command to add environment variables:

```
source /opt/hadoopclient/bigdata_env
```

**Step 5** Download the Redis user credential, decompress it to obtain the **krb5.conf** and **user.keytab** files, and upload the files to the **/srv/project/krb5-example** directory on the client server.

----End

## Kerberos Authentication

The following code is used as an example to connect to a Redis instance with Kerberos authentication enabled using Python 3.x. Modify the parameters in the example as needed by importing the dependency.

```
import gssapi
import os
import redis
import subprocess
import struct
#Change the values of KRB5CCNAME, KRB5_CONFIG, and KRB5_KTNAME as needed.
os.environ["KRB5_KTNAME"] = "/srv/project/krb5-example/user.keytab"
os.environ["KRB5_CONFIG"] = "/srv/project/krb5-example/krb5.conf"
os.environ["KRB5CCNAME"] = "/srv/project/krb5-example/krb5cc_0"
#Change the domain name.
realm = "HADOOP.COM"
#Change the username and domain name.
principal="redisuser@" + realm
store = {'ccache': os.environ["KRB5CCNAME"], 'keytab': os.environ["KRB5_KTNAME"]}
#Service IP address and port number for connecting to the Redis instance. You can change them as needed.
r = redis.Redis("192.168.20.238", 22400)
conn_pool = r.connection_pool
conn = conn_pool.get_connection("_")
subprocess.run(["kinit", "-kt", os.environ["KRB5_KTNAME"], principal])
server_principal = "redis/hadoop." + realm.lower() + "@" + realm
service_name = gssapi.Name(server_principal)
cname = service_name.canonicalize(gssapi.MechType.kerberos)
```

```
client_creds = gssapi.Credentials(usage='both', store=store)
client_ctx = gssapi.SecurityContext(name=cname,
                                    flags=gssapi.RequirementFlag.mutual_authentication,
                                    mech=gssapi.MechType.kerberos,
                                    creds=client_creds)
SASL_QOP_AUTH = 1
server_token = None
while not client_ctx.complete:
    client_token = client_ctx.step(server_token)
    client_token = client_token or b''
    conn.send_command("authext", client_token)
    auth_response=conn.read_response()
    server_token=auth_response[4:]

msg = client_ctx.unwrap(server_token).message
qop = struct.pack('!B', SASL_QOP_AUTH & msg[0])
msg = qop + msg[1:]
msg = client_ctx.wrap(msg + principal.encode(), False).message
conn.send_command("authext", msg)
auth_response=conn.read_response()
print(auth_response)

conn.send_command("config", "get", "maxmemory")
auth_response=conn.read_response()
print(auth_response)
```

#### NOTE

- **KRB5\_KTNAME** and **KRB5\_CONFIG** indicate the locations where the user authentication files decompressed from **user.keytab** and **krb5.conf** are uploaded, respectively, on the Linux server.
- **KRB5CCNAME** indicates the path of the cache file generated after you log in to Kerberos.
- **realm** indicates the cluster domain name. **HADOOP.COM** is for reference only. You can log in to FusionInsight Manager and choose **System > Permission > Domain** and **Mutual Trust** to view the actual domain name of the cluster.
- **principal** is in the format of *Username@Domain name*. **redisuser@" + realm** is for reference only.
- Obtaining the instance service IP address: Log in to FusionInsight Manager, choose **Cluster > Services > Redis**, and click **Instance** to view the IP addresses of instances running in the cluster.
- Obtaining the port number: On Redis nodes, the port number used by role **Redis\_1** is **22400**, and that used by role **Redis\_2** is **22401**. The others follow the same rule.

## Debugging Program

- Step 1** Log in to the server where the Redis client is deployed as the **root** user.
- Step 2** Save the modified Python sample code as **krb5test.py** and upload it to the **/srv/project/krb5-example** directory on the server.
- Step 3** Run the following commands to change the permission on the **krb5test.py** file to **700**:
- ```
cd /srv/project/krb5-example
chmod 700 krb5test.py
```
- Step 4** Run the following command to run the sample Python program:
- ```
python3 krb5test.py
```

**Step 5** The command output is as follows:

```
...  
b'authok'  
[b'maxmemory', b'1073741824']
```

----End

### 1.18.3.4.2 Connection to Redis in Security Mode Using C and C++

## Overview

This section describes how to use C and C++ to connect to Redis in security mode to implement Kerberos authentication. This section is implemented based on the environment listed in the following table. Other environments are similar and are not described. Note that some commands in the guide do not support the Debian OS.

**Table 1-167**

Environment	Version information
GCC	7.3.0
CMake	3.22.1

This chapter describes how to log in to the Kerberos service using **kinit**, whose login duration is 24 hours. You can periodically update the login information of **kinit** based on the site requirements to prevent Redis access failures caused by login information expiration.

## Prerequisites

- SSL channel encryption has been disabled.  
Log in to FusionInsight Manager and choose **Cluster > Services > Redis**. Click **Configurations** then **All Configurations**, search for **REDIS\_SSL\_ON**, and set this parameter to **false**, and click **Save**. Restart the service for the configuration to take effect.
- A Redis role and user has been created on FusionInsight Manager, for example, create **redisuser**. For details, see [Preparing Operating Environment](#). The authentication file of the user has been downloaded.
- The Hiredis source code package has been downloaded to the local environment in advance because it is used to access the Redis service in this test example. The download address is <https://github.com/redis/hiredis/tree/v1.0.2>. Click **Download ZIP** to download Hiredis 1.0.2 to the local environment.
- The Kerberos dependencies have been installed. EulerOS is used as an example here.

**yum install cyrus-sasl cyrus-sasl-devel cyrus-sasl-lib**

 NOTE

Debian OSs do not support the preceding commands. You need to install Cyrus and krb5 dependency packages.

## Kerberos Authentication

The following code is used as an example to connect to the Redis instance in security mode using C/C++. Modify the parameters in the example based on the site requirements.

```
#include <stdio.h>
#include "kerberos.h"
#include "hiredis/hiredis.h"
#include "hiredis/sds.h"

#define CRLF      "\r\n"
#define AUTH_OK    "authok"
// auth continue
#define ACT_PREFIX  "act:"
#define CRLF_LEN   2
#define AUTH_OK_LEN 6
#define ACT_PREFIX_LEN 4

/* Error codes */
#define C_OK        0
#define C_ERR       -1

int sds2str(char *s, long long value) {
    char *p, aux;
    unsigned long long v;
    size_t l;
    /* Generate the string representation, this method produces
     * a reversed string. */
    v = (value < 0) ? -value : value;
    p = s;
    do {
        *p++ = '0' + (v%10);
        v /= 10;
    } while(v);
    if (value < 0) *p++ = '-';
    /* Compute length and add null term. */
    l = p-s;
    *p = '\0';
    /* Reverse the string. */
    p--;
    while(s < p) {
        aux = *s;
        *s = *p;
        *p = aux;
        s++;
        p--;
    }
    return l;
}
/**
 * Send authentication information to the server.
 * @param c Redis connection information
 * @param datalen ticket length
 * @param data ticket content
 * @return Whether the message is sent successfully
 */
int sendAuthReq(redisContext *c, size_t datalen, char *data) {
    sds cmd = sdsnew("*2\r\n$7\r\nnauthext\r\n");
    char buf[32] = {0};
    long len = 0;
    len = sds2str(buf, (long long )datalen);
```

```
cmd = sdscatlen(cmd, "$", 1);
cmd = sdscatlen(cmd, buf, len);
cmd = sdscatlen(cmd, CRLF, CRLF_LEN);
cmd = sdscatlen(cmd, data, datalen);
cmd = sdscatlen(cmd, CRLF, CRLF_LEN);
if (redisAppendFormattedCommand(c, cmd, sdslen(cmd)) != REDIS_OK) {
    return C_ERR;
}
return C_OK;
}
/**
 * Obtain the returned authentication information.
 * @param c Redis connection information
 * @return Ticket information returned during authentication
 */
sds getAuthReply(redisContext *c) {
    redisReply *reply;
    if (redisGetReply(c, (void **)(&reply)) == REDIS_ERR) {
        printf("reply error\n");
        return NULL;
    }
    if (reply->type == REDIS_REPLY_ERROR) {
        printf("(error) %s\n", reply->str);
        freeReplyObject(reply);
        return NULL;
    }
    sds ticket = sdsnewlen(reply->str, reply->len);
    if (!strncmp(ticket, AUTH_OK, AUTH_OK_LEN)) {
        return ticket;
    }
    // Delete useless information returned by the server: act:
    sdsrange(ticket, ACT_PREFIX_LEN, -1);
    freeReplyObject(reply);
    return ticket;
}
int main() {
    // Initialize the Kerberos authentication information when the client is started.
    initializeClient();

    redisContext *c = redisConnect("Instance service IP address", Port number);
    int finished = 0;
    char* full_server_realm = "hadoop.Domain name";
    printf("full_server_realm=%s\n", full_server_realm);
    struct client_state state = {
        .auth_started = 0,
        .auth_complete = 0
    };
    // Start authentication.
    gss_result result = start(full_server_realm, &state);
    if (result.code != 0) {
        printf("error:%s", result.message);
        return -1;
    }
    while (!finished) {
        sendAuthReq(c, result.len, result.response);
        sds reply = getAuthReply(c);
        if (reply == NULL) {
            return -1;
        }
        if (!strncmp(reply, AUTH_OK, AUTH_OK_LEN)) {
            finished = 1;
            continue;
        }
        result = step(reply, sdslen(reply), &state);
        if (result.code == -1) {
            printf("error:%s\n", result.message);
            return -1;
        }
        if (result.response == NULL) {
```

```
        result.response = "";
    }
}
printf("auth ok%\n\n");
printf("authed user=%s\n", state.username);
redisCommand(c, "set %s %s", "b", "bbbbbb");
redisReply *reply = redisCommand(c, "get %s", "b");
printf("b=%s\n", reply->str);
return 0;
}
```

#### NOTE

- Obtaining the instance service IP address: Log in to FusionInsight Manager, choose **Cluster > Services > Redis**, and click **Instance** to view the IP addresses of instances running in the cluster.
- Obtaining the port number: On Redis nodes, the port number used by role **Redis\_1** is **22400**, and that used by role **Redis\_2** is **22401**. The others follow the same rule.
- Format of **full\_server\_realm**: "*hadoop.Domain name*". You can log in to FusionInsight Manager and choose **System > Permission > Domain** and **Mutual Trust** to view the cluster domain name of the cluster.

## Running the Sample

**Step 1** Upload the modified sample code to the **/srv/cKerberos** directory on the Linux server.

**Step 2** Decompress the downloaded Hiredis package and upload the decompressed code to **/srv/cKerberos/src/hiredis**.

```
[root@192-168-150-7 cKerberos]# ls src/hiredis/
adapters alloc.h      async.c  async_private.h CMakeLists.txt dict.c examples hiredis.c
hiredis.h   hiredis_ssl-config.cmake.in hiredis_ssl.pc.in net.c read.c README.md sds.c sockcompat.c
ssl.c test.sh
alloc.c appveyor.yml async.h CHANGELOG.md  COPYING      dict.h fmacros.h hiredis-
config.cmake.in hiredis.pc.in hiredis_ssl.h      Makefile    net.h read.h sdsalloc.h sds.h
sockcompat.h test.c win32.h
```

**Step 3** Create the **/srv/cKerberos/build** folder.

```
mkdir -p /srv/cKerberos/build
```

```
cd /srv/cKerberos/build
```

**Step 4** Compile the sample code.

```
cmake -DTEST_ENABLE=true .. && make
```

The compilation result is as follows:

```
Detected version: 1.0.2
-- Configuring done
-- Generating done
-- Build files have been written to: /srv/cKerberos/build
Consolidate compiler generated dependencies of target ckrb5
[ 8%] Built target ckrb5
[ 12%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/alloc.c.o
[ 16%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/async.c.o
[ 20%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/dict.c.o
[ 25%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/net.c.o
[ 29%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/read.c.o
[ 33%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/sds.c.o
[ 37%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/hiredis.c.o
[ 41%] Building C object CMakeFiles/kerberosTest.dir/src/hiredis/sockcompat.c.o
[ 45%] Building C object CMakeFiles/kerberosTest.dir/src/kerberos.c.o
```

```
[ 50%] Building C object CMakeFiles/kerberosTest.dir/src/testMain.c.o
[ 54%] Linking C executable kerberosTest
[ 54%] Built target kerberosTest
[ 58%] Building C object src/hiredis/CMakeFiles/hiredis.dir/alloc.c.o
[ 62%] Building C object src/hiredis/CMakeFiles/hiredis.dir/async.c.o
[ 66%] Building C object src/hiredis/CMakeFiles/hiredis.dir/dict.c.o
[ 70%] Building C object src/hiredis/CMakeFiles/hiredis.dir/hiredis.c.o
[ 75%] Building C object src/hiredis/CMakeFiles/hiredis.dir/net.c.o
[ 79%] Building C object src/hiredis/CMakeFiles/hiredis.dir/read.c.o
[ 83%] Building C object src/hiredis/CMakeFiles/hiredis.dir/sds.c.o
[ 87%] Building C object src/hiredis/CMakeFiles/hiredis.dir/sockcompat.c.o
[ 91%] Linking C shared library libhiredis.so
[ 91%] Built target hiredis
[ 95%] Building C object src/hiredis/CMakeFiles/hiredis-test.dir/test.c.o
[100%] Linking C executable hiredis-test
[100%] Built target hiredis-test
```

**Step 5** Run the sample code.

```
source Client path/ bigdata_env
kinit Redis user
./kerberosTest
```

**Step 6** Check the command output, as shown in the following figure.

```
[root@100-85-150-7 build]# ./kerberosTest
full_server_realm=hadoop.HADOOP.COM
auth ok

authed user=redisuser@HADOOP.COM

b=bbbbbb
[root@100-85-150-7 build]#
```

----End

#### 1.18.3.4.3 Connection to Redis in Security Mode Using Node.js

##### Overview

This section describes how to use Node.js to connect to a Redis instance in security mode. Currently, the development environment running Windows is not supported.

This chapter describes how to log in to the Kerberos service using **kinit**, whose login duration is 24 hours. You can periodically update the login information of **kinit** based on the site requirements to prevent Redis access failures caused by login information expiration.

##### Prerequisites

- SSL channel encryption has been disabled.  
Log in to FusionInsight Manager and choose **Cluster > Services > Redis**. Click **Configurations** then **All Configurations**, search for **REDIS\_SSL\_ON**, and set

this parameter to **false**, and click **Save**. Restart the service for the configuration to take effect.

- A Redis role and user has been created on FusionInsight Manager, for example, create **redisuser**. For details, see [Preparing Operating Environment](#). The authentication file of the user has been downloaded.
- This section takes a CentOS 7.6 running on a x86 server as an example. The basic Node.js environment has been installed. The current example uses Node.js to call the binary library and you need to install authentication-related packages before authentication.  
**yum install cyrus-sasl cyrus-sasl-devel cyrus-sasl-lib**
- The Node.js dependencies have been installed.  
**npm install node-gyp -g**

## Kerberos Authentication

The following code is used as an example to connect to the Redis instance in security mode using Node.js. Modify the parameters in the example based on the site requirements.

```
"use strict";

const stream = require("net");
const {genCommand} = require("./redisProtocol")
const Krb5 = require("./kerberosAuth")
const {initParser} = Krb5

let realm = "HADOOP.COM";
let username = "flinkuser" + "@" + realm;
// Initialize Krb5Auth.
let auth = new Krb5.Krb5Auth("./krb5.conf", "./user.keytab", realm, username);
auth.setCCname("/srv/kerberos-auth/flinkuser_10000");
auth.kinit();
let connectionOptions = {
    "port": Port number,
    "host": Service instance IP address
};
// Connect to the Redis service.
const client = stream.createConnection(connectionOptions);
// Initialize the Redis return value parser.
const authParser = initParser(auth);
// Start authentication.
auth.authStart(client);
let count = 1;
client.on("data", (buffer) => {
    authParser.execute(buffer);
    if (auth.error) {
        console.log("error");
        return;
    }
    if (count <= 0) {
        // After the authentication is successful, the system sends a password and exits. This parameter is invalid.
        process.exit();
    }
    if (auth.authed) {
        // After the authentication is successful, the service command starts to be sent. The following uses config get maxmemory as an example.
        console.log("send info");
        client.write(genCommand("config", ["get", "maxmemory"]));
        count--;
    } else {
        // Before the authentication is successful, obtain the ticket from the server, verify the ticket, and
```

generate a ticket from the client until the authentication is successful. This process takes about two to three times.

```
        auth.authTicket(client);
    }
});
```

#### NOTE

- Obtaining the instance service IP address: Log in to FusionInsight Manager, choose **Cluster > Services > Redis**, and click **Instance** to view the IP addresses of instances running in the cluster.
- Obtaining the port number: On Redis nodes, the port number used by role **Redis\_1** is **22400**, and that used by role **Redis\_2** is **22401**. The others follow the same rule.
- **realm** indicates the cluster domain name. **HADOOP.COM** is for reference only. You can log in to FusionInsight Manager and choose **System > Permission > Domain** and **Mutual Trust** to view the actual domain name of the cluster.
- **Krb5.Krb5Auth**: The first two parameters indicate the location where the user authentication files decompressed from **user.keytab** and **krb5.conf** are uploaded to the Linux server.
- **setCCname** indicates the path of the cache file generated after you log in to Kerberos.
- **username**: The value is in the format of *Username@Domain name*.

## Running the Sample

**Step 1** Upload modified sample code **test.js** to the **/srv/kerberos-auth** directory on the Linux server and run the following command to install the dependencies:

**npm install**

**Step 2** Decompress the downloaded authentication credential to obtain the **krb5.conf** and **user.keytab** files and upload them to the **/srv/kerberos-auth** directory on the server.

**Step 3** Run the following command to run the sample:

**node test.js**

**Step 4** Check the command output, as shown in the following figure.

```
[testuser@192-168-64-77 kerberos-auth]$ node test.js
kinitd
auth ok
send info
-----result begin-----
maxmemory,1073741824
-----result end-----
[testuser@192-168-64-77 kerberos-auth]$
```

----End

### 1.18.3.4.4 Connection to Redis in Security Mode Using the Go Language

## Overview

This section describes how to use the Go language on Linux to connect to Redis in security mode. Currently, the development environment running Windows is not supported.

The Go language is used to invoke the binary dependency library to implement Kerberos authentication. This chapter describes how to log in to the Kerberos service using **kinit**, whose login duration is 24 hours. You can periodically update the login information of **kinit** based on the site requirements to prevent Redis access failures caused by login information expiration.

## Prerequisites

- SSL channel encryption has been disabled.  
Log in to FusionInsight Manager and choose **Cluster > Services > Redis**. Click **Configurations** then **All Configurations**, search for **REDIS\_SSL\_ON**, and set this parameter to **false**, and click **Save**. Restart the service for the configuration to take effect.
- A Redis role and user has been created on FusionInsight Manager, for example, create **redisuser**. For details, see [Preparing Operating Environment](#). The authentication file of the user has been downloaded.
- The system packages related to Kerberos authentication has been installed. This section uses CentOS 7.6 on an x86 server as an example. This section does not describe how to set up the basic environment of the Go language. For details about how to install the system package, see the following command:  
**yum install cyrus-sasl cyrus-sasl-devel cyrus-sasl-lib**

## Kerberos Authentication

The following code is used as an example to connect to the Redis instance in security mode using the Go language. Modify the parameters in the example based on the site requirements.

```
package main
/*
// Location of the main package
#cgo CFLAGS: -I./include
//Load the path and file name of the dynamic library.
#cgo LDFLAGS: -L./lib ./lib/libcckrb5.a -lsasl2 -Wl,-rpath,lib
#include "kerberos.h"
*/
import "C"
import (
    "bufio"
    "encoding"
    "errors"
    "fmt"
    "io"
    "math"
    "math/big"
    "net"
    "strconv"
    "strings"
    "time"
```

```
"unsafe"
)
// BytesToString converts byte slice to string.
func BytesToString(b []byte) string {
    return *(*string)(unsafe.Pointer(&b))
}
// StringToBytes converts string to byte slice.
func StringToBytes(s string) []byte {
    return *(*[]byte)(unsafe.Pointer(
        &struct {
            string
            Cap int
        }{s, len(s)},
    ))
}
func Atoi(b []byte) (int, error) {
    return strconv.Atoi(BytesToString(b))
}
func ParseInt(b []byte, base int, bitSize int) (int64, error) {
    return strconv.ParseInt(BytesToString(b), base, bitSize)
}
type writer interface {
    io.Writer
    io.ByteWriter
    // WriteString implement io.StringWriter.
    WriteString(s string) (n int, err error)
}
type Writer struct {
    writer
    lenBuf []byte
    numBuf []byte
}
func NewWriter(wr writer) *Writer {
    return &Writer{
        writer: wr,
        lenBuf: make([]byte, 64),
        numBuf: make([]byte, 64),
    }
}
func (w *Writer) WriteArgs(args []interface{}) error {
    if err := w.WriteByte(RespArray); err != nil {
        return err
    }
    if err := w.writeLen(len(args)); err != nil {
        return err
    }
    for _, arg := range args {
        if err := w.WriteArg(arg); err != nil {
            return err
        }
    }
    return nil
}
func (w *Writer) writeLen(n int) error {
    w.lenBuf = strconv.AppendUint(w.lenBuf[:0], uint64(n), 10)
    w.lenBuf = append(w.lenBuf, '\r', '\n')
    err := w.Write(w.lenBuf)
    return err
}
func (w *Writer) WriteArg(v interface{}) error {
    switch v := v.(type) {
    case nil:
        return w.string("")
    case string:
        return w.string(v)
    case []byte:
        return w.bytes(v)
    case int:
        return w.int(int64(v))
    }
```

```
case int8:
    return w.int(int64(v))
case int16:
    return w.int(int64(v))
case int32:
    return w.int(int64(v))
case int64:
    return w.int(v)
case uint:
    return w.uint(uint64(v))
case uint8:
    return w.uint(uint64(v))
case uint16:
    return w.uint(uint64(v))
case uint32:
    return w.uint(uint64(v))
case uint64:
    return w.uint(v)
case float32:
    return w.float(float64(v))
case float64:
    return w.float(v)
case bool:
    if v {
        return w.int(1)
    }
    return w.int(0)
case time.Time:
    w.numBuf = v.AppendFormat(w.numBuf[:0], time.RFC3339Nano)
    return w.bytes(w.numBuf)
case time.Duration:
    return w.int(v.Nanoseconds())
case encoding.BinaryMarshaler:
    b, err := v.MarshalBinary()
    if err != nil {
        return err
    }
    return w.bytes(b)
case net.IP:
    return w.bytes(v)
default:
    return fmt.Errorf(
        "redis: can't marshal %T (implement encoding.BinaryMarshaler)", v
)
}
func (w *Writer) bytes(b []byte) error {
    if err := w.WriteByte(RespString); err != nil {
        return err
    }
    if err := w.writeLen(len(b)); err != nil {
        return err
    }
    if _, err := w.Write(b); err != nil {
        return err
    }
    return w.crlf()
}
func (w *Writer) string(s string) error {
    return w.bytes(StringToBytes(s))
}
func (w *Writer) uint(n uint64) error {
    w.numBuf = strconv.AppendUint(w.numBuf[:0], n, 10)
    return w.bytes(w.numBuf)
}
func (w *Writer) int(n int64) error {
    w.numBuf = strconv.AppendInt(w.numBuf[:0], n, 10)
    return w.bytes(w.numBuf)
}
func (w *Writer) float(f float64) error {
```

```
w.numBuf = strconv.AppendFloat(w.numBuf[:0], f, 'f', -1, 64)
return w.bytes(w.numBuf)
}
func (w *Writer) crlf() error {
    if err := w.WriteByte('\r'); err != nil {
        return err
    }
    return w.WriteByte('\n')
}
// redis resp protocol data type.
const (
    RespStatus  = '+' // +<string>\r\n
    RespError   = '-' // -<string>\r\n
    RespString  = '$' // $<length>\r\n<bytes>\r\n
    RespInt     = ':' // :<number>\r\n
    RespNil     = '_' // _\r\n
    RespFloat   = ';' // ,<floating-point-number>\r\n (golang float)
    RespBool    = '#' // true: #t\r\n false: #f\r\n
    RespBlobError = '!' // !<length>\r\n<bytes>\r\n
    RespVerbatim = '=' // =<length>\r\nFORMAT:<bytes>\r\n
    RespBigInt  = '(' // (<big number>\r\n
    RespArray   = '*' // *<len>\r\n... (same as resp2)
    RespMap     = '%' // %<len>\r\n(key)\r\n(value)\r\n... (golang map)
    RespSet     = '~' // ~<len>\r\n... (same as Array)
    RespAttr    = '|' // |<len>\r\n(key)\r\n(value)\r\n... + command reply
    RespPush    = '>' // ><len>\r\n... (same as Array)
)
// Not used temporarily.
// Redis has not used these two data types for the time being, and will implement them later.
// Streamed      = "EOF"
// StreamedAggregated = '?'
//-----
const Nil = RedisError("redis: nil") // nolint:errname
type RedisError string
func (e RedisError) Error() string { return string(e) }
func (RedisError) RedisError() {}
func ParseErrorReply(line []byte) error {
    return RedisError(line[1:])
}
//-----
type Reader struct {
    rd *bufio.Reader
}
func NewReader(rd io.Reader) *Reader {
    return &Reader{
        rd: bufio.NewReader(rd),
    }
}
func (r *Reader) Buffered() int {
    return r.rd.Buffered()
}
func (r *Reader) Peek(n int) ([]byte, error) {
    return r.rd.Peek(n)
}
func (r *Reader) Reset(rd io.Reader) {
    r.rd.Reset(rd)
}
// PeekReplyType returns the data type of the next response without advancing the Reader,
// and discard the attribute type.
func (r *Reader) PeekReplyType() (byte, error) {
    b, err := r.rd.Peek(1)
    if err != nil {
        return 0, err
    }
    if b[0] == RespAttr {
        if err = r.DiscardNext(); err != nil {
            return 0, err
        }
    }
    return r.PeekReplyType()
}
```

```
        }
        return b[0], nil
    }
    // ReadLine Return a valid reply, it will check the protocol or redis error,
    // and discard the attribute type.
    func (r *Reader) ReadLine() ([]byte, error) {
        line, err := r.readLine()
        if err != nil {
            return nil, err
        }
        switch line[0] {
        case RespError:
            return nil, ParseErrorReply(line)
        case RespNil:
            return nil, Nil
        case RespBlobError:
            var blobErr string
            blobErr, err = r.readStringReply(line)
            if err == nil {
                err = RedisError(blobErr)
            }
            return nil, err
        case RespAttr:
            if err = r.Discard(line); err != nil {
                return nil, err
            }
            return r.ReadLine()
        }
        // Compatible with RESP2
        if IsNilReply(line) {
            return nil, Nil
        }
        return line, nil
    }
    // readLine returns an error if:
    // - there is a pending read error;
    // - or line does not end with \r\n.
    func (r *Reader) readLine() ([]byte, error) {
        b, err := r.rd.ReadSlice('\n')
        if err != nil {
            if err != bufio.ErrBufferFull {
                return nil, err
            }
            full := make([]byte, len(b))
            copy(full, b)
            b, err = r.rd.ReadBytes('\n')
            if err != nil {
                return nil, err
            }
            full = append(full, b...) //nolint:makezero
            b = full
        }
        if len(b) <= 2 || b[len(b)-1] != '\n' || b[len(b)-2] != '\r' {
            return nil, fmt.Errorf("redis: invalid reply: %q", b)
        }
        return b[:len(b)-2], nil
    }
    func (r *Reader) ReadReply() (interface{}, error) {
        line, err := r.ReadLine()
        if err != nil {
            return nil, err
        }
        switch line[0] {
        case RespStatus:
            return string(line[1:]), nil
        case RespInt:
            return ParseInt(line[1:], 10, 64)
        case RespFloat:
            return r.readFloat(line)
```

```
case RespBool:
    return r.readBool(line)
case RespBigInt:
    return r.readBigInt(line)
case RespString:
    return r.readStringReply(line)
case RespVerbatim:
    return r.readVerb(line)
case RespArray, RespSet, RespPush:
    return r.readSlice(line)
case RespMap:
    return r.readMap(line)
}
return nil, fmt.Errorf("redis: can't parse %.100q", line)
}
func (r *Reader) readFloat(line []byte) (float64, error) {
    v := string(line[1:])
    switch string(line[1:]) {
    case "inf":
        return math.Inf(1), nil
    case "-inf":
        return math.Inf(-1), nil
    }
    return strconv.ParseFloat(v, 64)
}
func (r *Reader) readBool(line []byte) (bool, error) {
    switch string(line[1:]) {
    case "t":
        return true, nil
    case "f":
        return false, nil
    }
    return false, fmt.Errorf("redis: can't parse bool reply: %q", line)
}
func (r *Reader) readBigInt(line []byte) (*big.Int, error) {
    i := new(big.Int)
    if i, ok := i.SetString(string(line[1:])), 10); ok {
        return i, nil
    }
    return nil, fmt.Errorf("redis: can't parse bigint reply: %q", line)
}
func (r *Reader) readStringReply(line []byte) (string, error) {
    n, err := replyLen(line)
    if err != nil {
        return "", err
    }
    b := make([]byte, n+2)
    _, err = io.ReadFull(r.rd, b)
    if err != nil {
        return "", err
    }
    return BytesToString(b[:n]), nil
}
func (r *Reader) readVerb(line []byte) (string, error) {
    s, err := r.readStringReply(line)
    if err != nil {
        return "", err
    }
    if len(s) < 4 || s[3] != ':' {
        return "", fmt.Errorf("redis: can't parse verbatim string reply: %q", line)
    }
    return s[4:], nil
}
func (r *Reader) readSlice(line []byte) ([]interface{}, error) {
    n, err := replyLen(line)
    if err != nil {
        return nil, err
    }
    val := make([]interface{}, n)
```

```
for i := 0; i < len(val); i++ {
    v, err := r.ReadReply()
    if err != nil {
        if err == Nil {
            val[i] = nil
            continue
        }
        if err, ok := err.(RedisError); ok {
            val[i] = err
            continue
        }
        return nil, err
    }
    val[i] = v
}
return val, nil
}
func (r *Reader) readMap(line []byte) (map[interface{}]interface{}, error) {
    n, err := replyLen(line)
    if err != nil {
        return nil, err
    }
    m := make(map[interface{}]interface{}, n)
    for i := 0; i < n; i++ {
        k, err := r.ReadReply()
        if err != nil {
            return nil, err
        }
        v, err := r.ReadReply()
        if err != nil {
            if err == Nil {
                m[k] = nil
                continue
            }
            if err, ok := err.(RedisError); ok {
                m[k] = err
                continue
            }
            return nil, err
        }
        m[k] = v
    }
    return m, nil
}
// -----
func (r *Reader) ReadInt() (int64, error) {
    line, err := r.ReadLine()
    if err != nil {
        return 0, err
    }
    switch line[0] {
    case RespInt, RespStatus:
        return ParseInt(line[1:], 10, 64)
    case RespString:
        s, err := r.readStringReply(line)
        if err != nil {
            return 0, err
        }
        return ParseInt([]byte(s), 10, 64)
    case RespBigInt:
        b, err := r.readBigInt(line)
        if err != nil {
            return 0, err
        }
        if !b.lIsInt64() {
            return 0, fmt.Errorf("bigint(%s) value out of range", b.String())
        }
        return b.Int64(), nil
    }
```

```
    return 0, fmt.Errorf("redis: can't parse int reply: %.100q", line)
}
func (r *Reader) ReadFloat() (float64, error) {
    line, err := r.ReadLine()
    if err != nil {
        return 0, err
    }
    switch line[0] {
    case RespFloat:
        return r.readFloat(line)
    case RespStatus:
        return strconv.ParseFloat(string(line[1:]), 64)
    case RespString:
        s, err := r.readStringReply(line)
        if err != nil {
            return 0, err
        }
        return strconv.ParseFloat(s, 64)
    }
    return 0, fmt.Errorf("redis: can't parse float reply: %.100q", line)
}
func (r *Reader) ReadString() (string, error) {
    line, err := r.ReadLine()
    if err != nil {
        return "", err
    }
    switch line[0] {
    case RespStatus, RespInt, RespFloat:
        return string(line[1:]), nil
    case RespString:
        return r.readStringReply(line)
    case RespBool:
        b, err := r.readBool(line)
        return strconv.FormatBool(b), err
    case RespVerbatim:
        return r.readVerb(line)
    case RespBigInt:
        b, err := r.readBigInt(line)
        if err != nil {
            return "", err
        }
        return b.String(), nil
    }
    return "", fmt.Errorf("redis: can't parse reply=%s reading string", line)
}
func (r *Reader) ReadBool() (bool, error) {
    s, err := r.ReadString()
    if err != nil {
        return false, err
    }
    return s == "OK" || s == "1" || s == "true", nil
}
func (r *Reader) ReadSlice() ([]interface{}, error) {
    line, err := r.ReadLine()
    if err != nil {
        return nil, err
    }
    return r.readSlice(line)
}
// ReadFixedArrayLen read fixed array length.
func (r *Reader) ReadFixedArrayLen(fixedLen int) error {
    n, err := r.ReadArrayLen()
    if err != nil {
        return err
    }
    if n != fixedLen {
        return fmt.Errorf("redis: got %d elements in the array, wanted %d", n, fixedLen)
    }
    return nil
```

```
}

// ReadArrayLen Read and return the length of the array.
func (r *Reader) ReadArrayLen() (int, error) {
    line, err := r.ReadLine()
    if err != nil {
        return 0, err
    }
    switch line[0] {
    case RespArray, RespSet, RespPush:
        return replyLen(line)
    default:
        return 0, fmt.Errorf("redis: can't parse array/set/push reply: %.100q", line)
    }
}

// ReadFixedMapLen reads fixed map length.
func (r *Reader) ReadFixedMapLen(fixedLen int) error {
    n, err := r.ReadMapLen()
    if err != nil {
        return err
    }
    if n != fixedLen {
        return fmt.Errorf("redis: got %d elements in the map, wanted %d", n, fixedLen)
    }
    return nil
}

// ReadMapLen reads the length of the map type.
// If responding to the array type (RespArray/RespSet/RespPush),
// it must be a multiple of 2 and return n/2.
// Other types will return an error.
func (r *Reader) ReadMapLen() (int, error) {
    line, err := r.ReadLine()
    if err != nil {
        return 0, err
    }
    switch line[0] {
    case RespMap:
        return replyLen(line)
    case RespArray, RespSet, RespPush:
        // Some commands and RESP2 protocol may respond to array types.
        n, err := replyLen(line)
        if err != nil {
            return 0, err
        }
        if n%2 != 0 {
            return 0, fmt.Errorf("redis: the length of the array must be a multiple of 2, got: %d", n)
        }
        return n / 2, nil
    default:
        return 0, fmt.Errorf("redis: can't parse map reply: %.100q", line)
    }
}

// DiscardNext read and discard the data represented by the next line.
func (r *Reader) DiscardNext() error {
    line, err := r.ReadLine()
    if err != nil {
        return err
    }
    return r.Discard(line)
}

// Discard the data represented by line.
func (r *Reader) Discard(line []byte) (err error) {
    if len(line) == 0 {
        return errors.New("redis: invalid line")
    }
    switch line[0] {
    case RespStatus, RespError, RespInt, RespNil, RespFloat, RespBool, RespBigInt:
        return nil
    }
    n, err := replyLen(line)
```

```

if err != nil && err != Nil {
    return err
}
switch line[0] {
case RespBlobError, RespString, RespVerbatim:
    // +\r\n
    _ = err = r.rd.Discard(n + 2)
    return err
case RespArray, RespSet, RespPush:
    for i := 0; i < n; i++ {
        if err = r.DiscardNext(); err != nil {
            return err
        }
    }
    return nil
case RespMap, RespAttr:
    // Read key & value.
    for i := 0; i < n*2; i++ {
        if err = r.DiscardNext(); err != nil {
            return err
        }
    }
    return nil
}
return fmt.Errorf("redis: can't parse %.100q", line)
}
func replyLen(line []byte) (n int, err error) {
    n, err = Atoi(line[1:])
    if err != nil {
        return 0, err
    }
    if n < -1 {
        return 0, fmt.Errorf("redis: invalid reply: %q", line)
    }
    switch line[0] {
    case RespString, RespVerbatim, RespBlobError,
        RespArray, RespSet, RespPush, RespMap, RespAttr:
        if n == -1 {
            return 0, Nil
        }
    }
    return n, nil
}
// IsNilReply detects redis.Nil of RESP2.
func IsNilReply(line []byte) bool {
    return len(line) == 3 &&
        (line[0] == RespString || line[0] == RespArray) &&
        line[1] == '-' && line[2] == '1'
}
func sendAuth(wr *Writer, bw *bufio.Writer, conn net.Conn, ticket *C.char, ticketLen C.int) {
    if bw.Buffered() > 0 {
        bw.Reset(conn)
    }
    goLen := int(ticketLen)
    var goStr = ""
    if goLen != 0 {
        goStr = C.GoStringN(ticket, ticketLen)
    }
    _ = wr.WriteByte(RespArray)
    _ = wr.writeLen(2)
    _ = wr.bytes([]byte("authext"))
    _ = wr.WriteByte(RespString)
    _ = wr.writeLen(goLen)
    if goLen != 0 {
        _ = wr.Write([]byte(goStr))
    }
    _ = wr.crlf()
    err := bw.Flush()
    if err != nil {

```

```

        fmt.Println("send error", err)
    }
}

func sendCommand(wr *Writer, bw *bufio.Writer, conn net.Conn, params []string) {
    if bw.Buffered() > 0 {
        bw.Reset(conn)
    }
    _ = wr.WriteByte(RespArray)
    _ = wr.writeLen(len(params))
    var i int
    for i = 0; i < len(params); i++ {
        _ = wr.WriteByte(RespString)
        _ = wr.writeLen(len(params[i])))
        _ = wr.Write([]byte(params[i]))
        _ = wr.crlf()
    }
    err := bw.Flush()
    if err != nil {
        fmt.Println("send error", err)
    }
}

// This function needs to be called to start authentication.
func authServer(serverRealm string, wr *Writer, bw *bufio.Writer, conn net.Conn, reader *Reader) (string, error) {
    fullServerRealm := C.CString(serverRealm)
    // Invoke the start function to enable authentication and generate a client ticket.
    result := C.start(fullServerRealm, &state)
    // Send the client ticket to the server.
    sendAuth(wr, bw, conn, result.response, result.len)
    // Obtain the ticket from the server.
    serverMsg, err := reader.ReadReply()
    if err != nil {
        fmt.Println("reply:", err)
        return "", err
    }
    serverTicket := fmt.Sprintf("%v", serverMsg)
    // Continue the authentication.
    return authServerStep(serverTicket, wr, bw, conn, reader)
}

// This function needs to be invoked recursively. The authentication is successful only when the server
// returns authok.
func authServerStep(serverTicket string, wr *Writer, bw *bufio.Writer, conn net.Conn, reader *Reader) (string, error) {
    if serverTicket == "authok" {
        return "authok", nil
    }
    // By default, the prefix act: is added to the ticket on the server. By default, the prefix needs to be deleted.
    if strings.HasPrefix(serverTicket, "act:") {
        serverTicket = serverTicket[4:]
    }
    sTicket := C.CString(serverTicket)
    sLen := len(serverTicket)
    sTLen := (*C.ulong)(unsafe.Pointer(&sLen))
    // Invoke step to verify the ticket on the server.
    result := C.step(sTicket, sTLen, &state)
    if result.code == -1 {
        goStr := C.GoStringN(result.message, 10)
        return "", errors.New(goStr)
    }
    // Send the new client ticket to the server.
    sendAuth(wr, bw, conn, result.response, result.len)
    // Obtain the server verification result. If the authentication is successful, authok is returned. If the
    // authentication needs to be continued, the server ticket is returned. If the authentication fails, an error is
    // returned.
    serverMsg, err := reader.ReadReply()
    if err != nil {
        // Authentication failed.
        fmt.Println("reply:", err)
        return "", err
    }
}

```

```
        }
        newSTicket := fmt.Sprintf("%v", serverMsg)
        // Continue the authentication.
        return authServerStep(newSTicket, wr, bw, conn, reader)
    }
    // Authentication information
    var goState *C.struct_client_state = &C.struct_client_state{
        auth_started: 0,
        auth_complete: 0,
    }
    // Convert to the c pointer
    var state = (*C.struct_client_state)(unsafe.Pointer(goState))
    func main() {
        // Initialize the authentication environment during startup.
        C.initializeClient()
        var fullServerRealm = "hadoop.HADOOP.COM"
        conn, err := net.Dial("tcp", "Service instance IP address.Port number")
        if err != nil {
            fmt.Println("err:", err)
            return
        }
        bw := bufio.NewWriter(conn)
        wr := NewWriter(bw)
        reader := NewReader(bufio.NewReader(conn))
        // Start authentication.
        authRes, err := authServer(fullServerRealm, wr, bw, conn, reader)
        if err != nil {
            fmt.Println("error:", err)
        }
        fmt.Println("authRes=", authRes)
        // The authentication is complete and the command starts to be sent.
        var infoCommand = []string{"info", "Server"}
        sendCommand(wr, bw, conn, infoCommand)
        serverMsg, err := reader.ReadReply()
        if err != nil {
            fmt.Println("reply:", err)
            return
        }
        info := fmt.Sprintf("%v", serverMsg)
        fmt.Println("info=", info)
    }
}
```

#### NOTE

- Obtaining the instance service IP address: Log in to FusionInsight Manager, choose **Cluster** > **Services** > **Redis**, and click **Instance** to view the IP addresses of instances running in the cluster.
- Obtaining the port number: On Redis nodes, the port number used by role **Redis\_1** is **22400**, and that used by role **Redis\_2** is **22401**. The others follow the same rule.
- Format of **full\_server\_realm**: "*hadoop.Domain name*". You can log in to FusionInsight Manager and choose **System** > **Permission** > **Domain** and **Mutual Trust** to view the cluster domain name of the cluster.

## Debugging Program

**Step 1** Compile the code to obtain binary files. For details, see [Connection to Redis in Security Mode Using C and C++](#). The following is an example:

```
-rw----- 1 root root 17815 Dec 22 16:47 CMakeCache.txt
drwx----- 6 root root 4096 Dec 22 17:03 CMakeFiles
-rw----- 1 root root 1774 Dec 22 16:47 cmake_install.cmake
-rwx----- 1 root root 146784 Dec 22 17:03 kerberosTest
-rw----- 1 root root 4324 Dec 22 16:46 libckrb5.a
-rw----- 1 root root 17417 Dec 22 17:03 Makefile
drwx----- 3 root root 4096 Dec 22 16:47 src
```

**Step 2** Upload the Go language sample code to the **/srv/go-krb5** directory.

```
drwx----- 2 root root 4096 Oct 21 15:48 include  
-rw----- 1 root root 18058 Oct 24 17:28 krb.go
```

**Step 3** Create the **lib** folder in **/srv/go-krb5** and copy **libckrb5.a** obtained in **Step 1** to the **lib** folder.

```
-rw----- 1 root root 4180 Oct 24 19:02 libckrb5.a
```

**Step 4** Run the following commands to run the sample:

```
source Client path / bigdata_env  
kinit Redis user  
go run krb.go
```

**Step 5** Check the command output, as shown in the following figure.

```
[root@100-85-150-7 go-Krb5]# go run krb.go  
authRes= authok  
info= # Server  
redis_version:6.0.12  
redis_git_shal:b64ba0c4  
redis_git_dirty:1  
redis_build_id:58359862364c6757  
redis_mode:cluster  
os:Linux 4.19.90-vhulk2011.1.0.h382.eulerov2r9.aarch64 aarch64  
arch_bits:64  
multiplexing_api:epoll  
atomicvar_api:atomic-builtin  
gcc_version:7.3.0  
process_id:39315  
run_id:74a479335a83d3a50bc83be89bd1237cla5b063a  
tcp_port:22400  
uptime_in_seconds:214933  
uptime_in_days:2  
hz:10  
configured_hz:10  
lru_clock:5716105  
executable:/opt/huawei/Bigdata/FusionInsight_HD_8.1.2.5/install/  
config_file:/opt/huawei/Bigdata/FusionInsight_HD_8.1.2.5/install/  
io_threads_active:0
```

----End

## 1.18.4 Application Commissioning

### 1.18.4.1 Commissioning an Application in Windows

#### 1.18.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

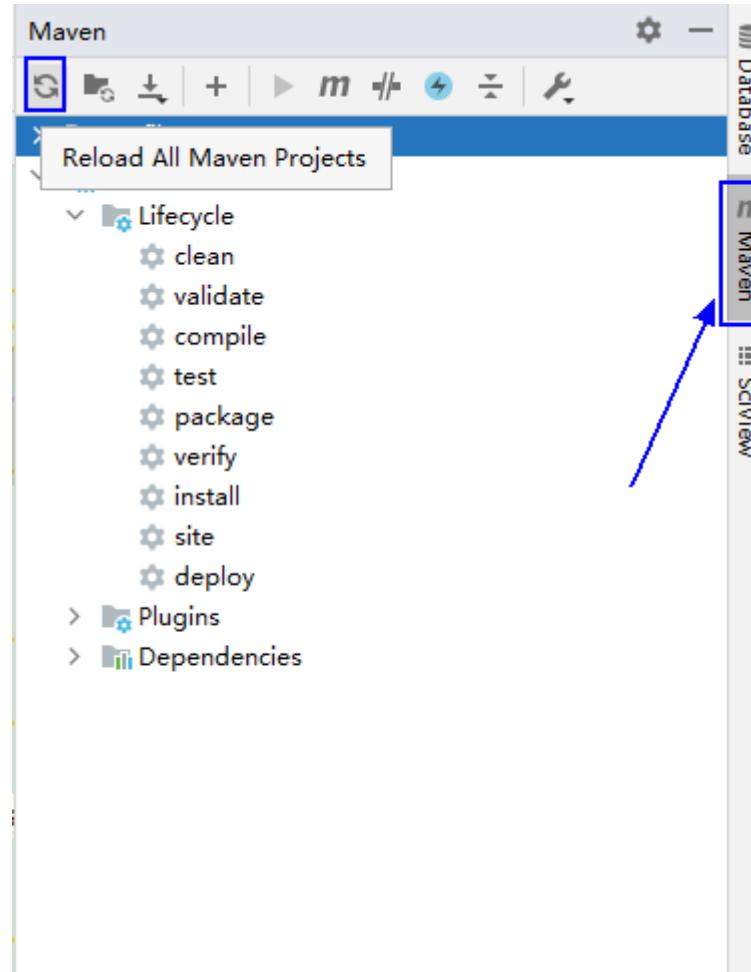
 NOTE

- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

## Procedure

**Step 1** Click **Reload All Maven Projects** in the Maven window on the right of the IDEA to reload the Maven project dependency.

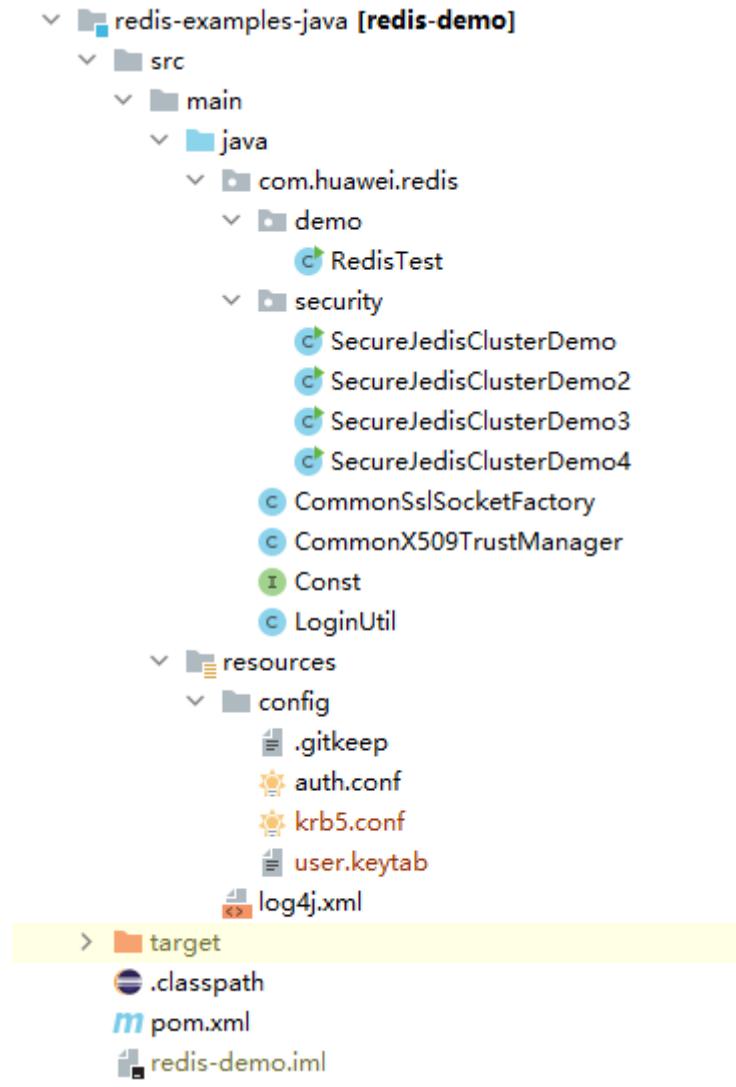
**Figure 1-223** reload projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user.  
See [Figure 1-224](#).

Figure 1-224 Directory list of **redis-demo** to be compiled



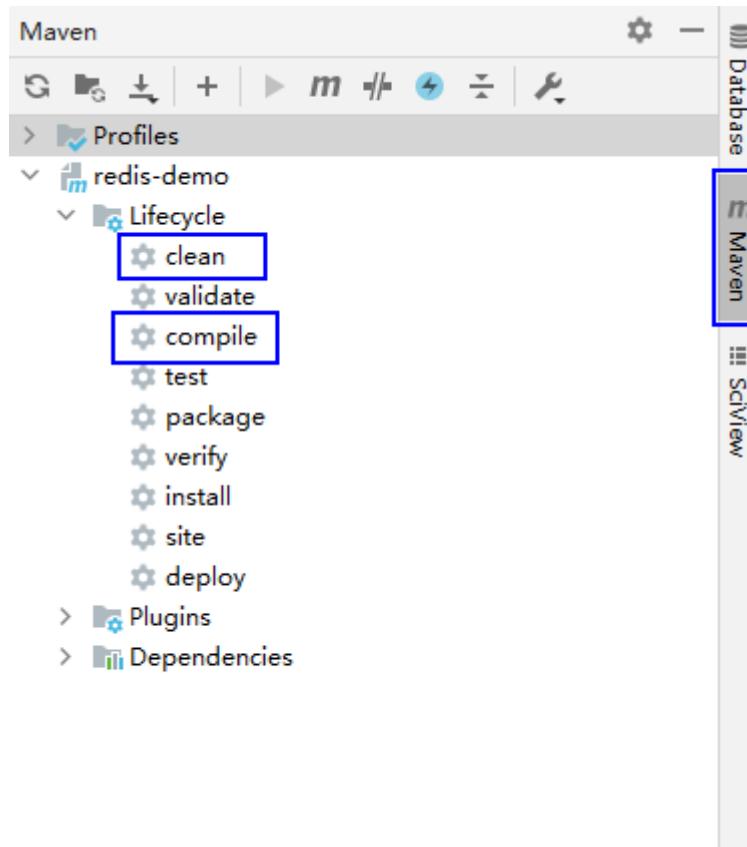
1. Compile an application.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

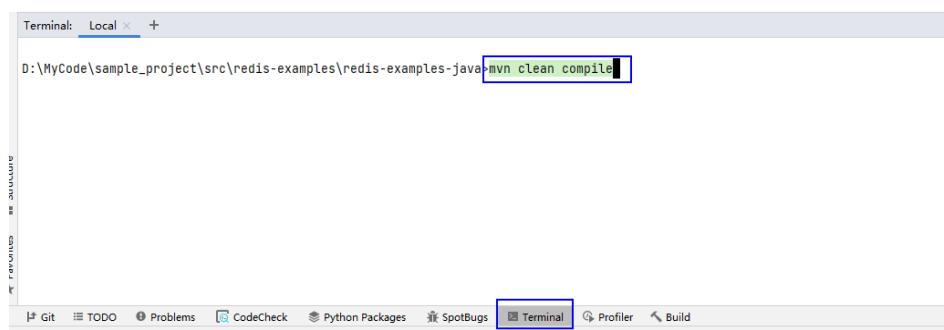
Figure 1-225 clean and compile tools of Maven



- Method 2:

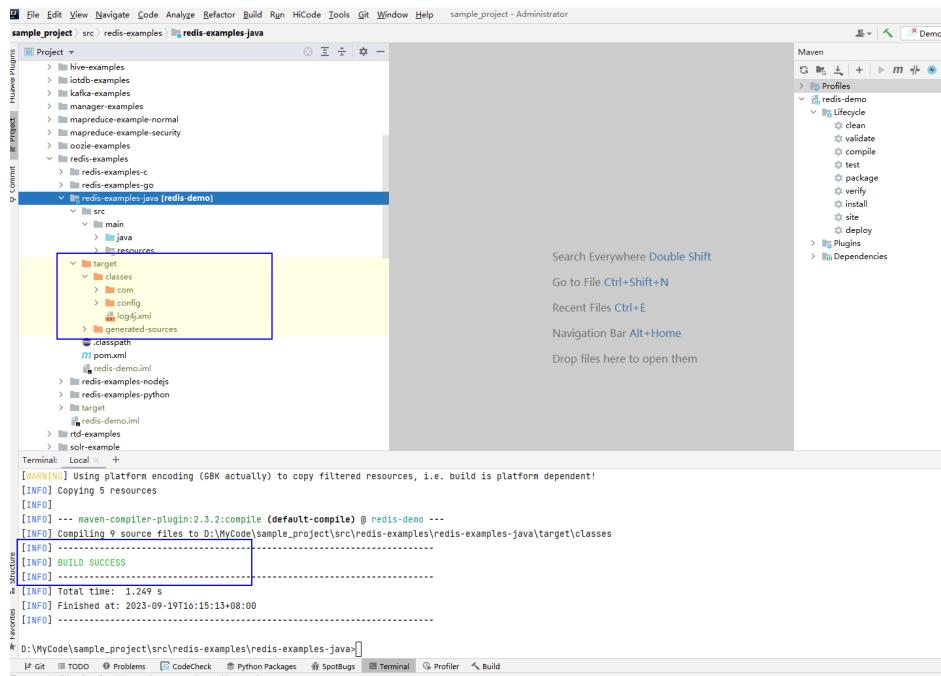
Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 1-226 Enter mvn clean compile in the IDEA Terminal text box



After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

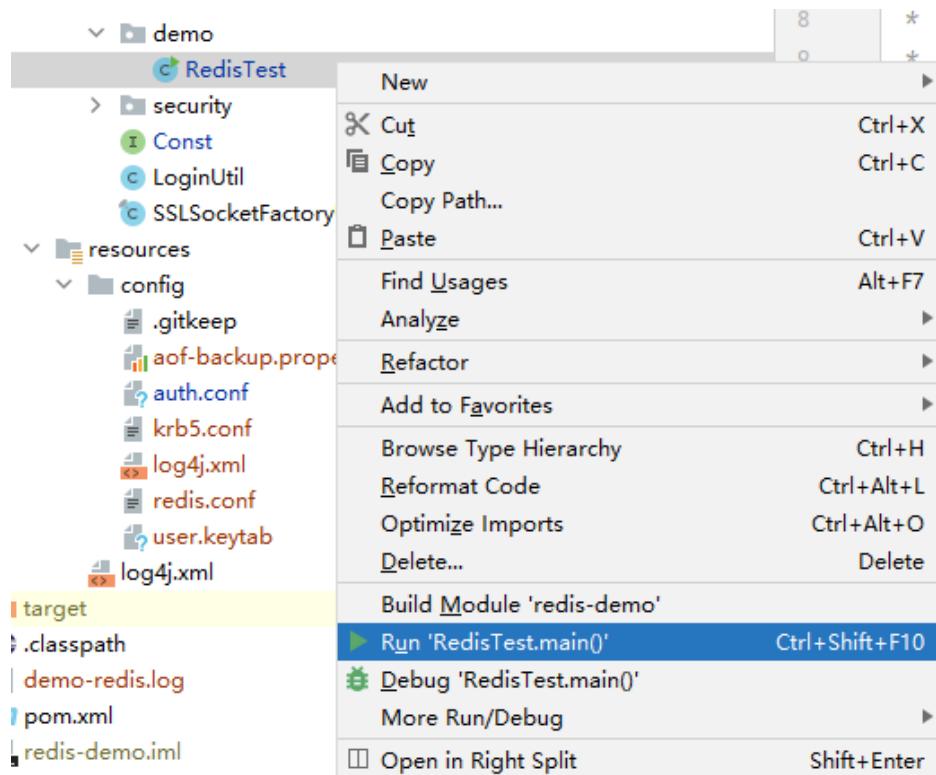
Figure 1-227 Compilation completed



## 2. Run the program.

Right-click **RedisTest.java**, and choose **Run 'RedisTest.main()'** from the shortcut menu to run the application project.

Figure 1-228 Run the application



----End

### 1.18.4.1.2 Viewing Commissioning Results

#### Scenario

After a Redis application is run, you can check the running result through one of the following methods:

- Viewing the IntelliJ IDEA running result.
- Logging in to the Redis Shell client to view the command output.

#### Procedure

- Viewing the IntelliJ IDEA running result

```
2019-06-22 16:23:27,519 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:93) - User sid-user01, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:37,921 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:101) - User sid-user01, session id: null
2019-06-22 16:23:38,084 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:107) - Value: hello
2019-06-22 16:23:38,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:111) - After append, value: hello world
2019-06-22 16:23:38,332 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:117) - User message, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:38,564 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:132) - All messages: [Hello how are you?, Fine thanks. I'm having fun with redis., I should look into this NOSQL thing ASAP]
2019-06-22 16:23:38,611 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:135) - Message count: 3
2019-06-22 16:23:38,665 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:139) - First message: Hello how are you?
2019-06-22 16:23:38,705 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:141) - After one pop, message count: 2
2019-06-22 16:23:39,016 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:157) - length of 1000000 is 7
2019-06-22 16:23:39,092 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:162) - User J001's name is John
2019-06-22 16:23:39,150 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:165) - {name=John, id=J001, gender=male, salary=1000000, age=35}
2019-06-22 16:23:39,430 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:179) - User L002's salary is 250000
2019-06-22 16:23:39,472 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:183) - all fields: [name, id, salary, gender, age]
2019-06-22 16:23:39,510 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:186) - all values: [L002, Lucy, 250000, 25, female]
2019-06-22 16:23:39,563 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:190) - partial field values: [L002, Lucy]
2019-06-22 16:23:39,605 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:194) - Exist field gender? true
2019-06-22 16:23:39,695 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:199) - after del field age, rest fields: [name, id, salary, gender]
2019-06-22 16:23:39,810 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:207) - key1's value is : value1
2019-06-22 16:23:39,888 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:211) - Can delete key by UNLINK.
2019-06-22 16:23:40,044 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:224) - Set size: 3
2019-06-22 16:23:40,122 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:228) - Set size: 3
2019-06-22 16:23:40,163 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:231) - Set: [TreeSet, HashSet, SortedSet]
2019-06-22 16:23:40,249 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:235) - Set: [TreeSet, HashSet]
2019-06-22 16:23:40,287 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:238) - TreeSet is set's member: true
2019-06-22 16:23:40,676 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:256) - All hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto, Linus
```

```
Torvalds]
2019-06-22 16:23:40,718 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:259) - Size: 6
2019-06-22 16:23:40,767 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:262) - Score: 1969.0
2019-06-22 16:23:40,808 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:265) - Count: 2
2019-06-22 16:23:40,860 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:269) - All
hackers 2: [Linus Torvalds, Yukihiro Matsumoto, Richard Stallman, Alan Kay, Claude Shannon, Alan
Turing]
2019-06-22 16:23:40,942 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:273) - All
hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto]
2019-06-22 16:23:41,106 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:284) - TTL: 5
2019-06-22 16:23:41,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:288) - KEY
type: string
2019-06-22 16:23:41,448 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:297) - List:
[1, 4, 6, 3, 8]
2019-06-22 16:23:41,490 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:300) - Sort
list: [1, 3, 4, 6, 8]
2019-06-22 16:23:42,129 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:343) - Result:
[www.google.cn, www.baidu.com, www.sina.com]
2019-06-22 16:23:42,301 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:60) - The
distance between test1 and test2 is 96.3242 KM.
2019-06-22 16:23:42,341 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:63) - Geo
location info is [(10.000002086162567,30.000000249977013),
(10.999999344348907,30.000000249977013)]
2019-06-22 16:23:42,380 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:66) -
geohash info of test1 and test2 is [sjr4et3f8v0, sjrfdm3fwt0]
2019-06-22 16:23:42,424 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:72) - Get
location info by longitude and latitude :
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test1 :
146.8169
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test2 :
111.2263
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test3 :
1.0E-4
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test4 :
95.3394
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test5 :
482.4041
2019-06-22 16:23:42,462 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:80) - Get
location info by member :
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test3 :
0.0
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test4 :
95.3395
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test2 :
111.2264
2019-06-22 16:23:42,657 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:312) - Byte
conversion can be done by dump and restore
```

- Logging in to the Redis Shell client to view the command output:

After the sample code is executed, keys are deleted. Therefore, you cannot directly use the shell command to view the application running result after the sample project is executed. The following uses sample code of **String Type Access** to explain how to use the shell command to view the application running result:

- Comment out **client.del(key);** in the last line of the **TestString()** method.

```
...
//client.del(key);
```

- Perform operations as instructed in **Compiling and Running an Application.**

- Install the Redis client and use the Redis user for authentication.

```
cd /opt/hadoopclient
```

```
source bigdata_env  
kinit redisuser
```

- d. Run the following command on the client node (<hostip> indicates any node in the Redis cluster):

```
cd /opt/client/Redis/bin  
redis-cli -c -p 22400 -h <hostip> get message
```

The value **A0BC9869FBC92933255A37A1D21167B2** set in the Java API is displayed:

```
"A0BC9869FBC92933255A37A1D21167B2"
```

#### NOTE

If channel encryption is enabled for Redis, run **redis-cli -c -p 22400 --tls -h <hostip> get message**.

To enable channel encryption for Redis, log in to Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for **REDIS\_SSL\_ON**, and change the parameter value to **true** to enable SSL channel encryption.

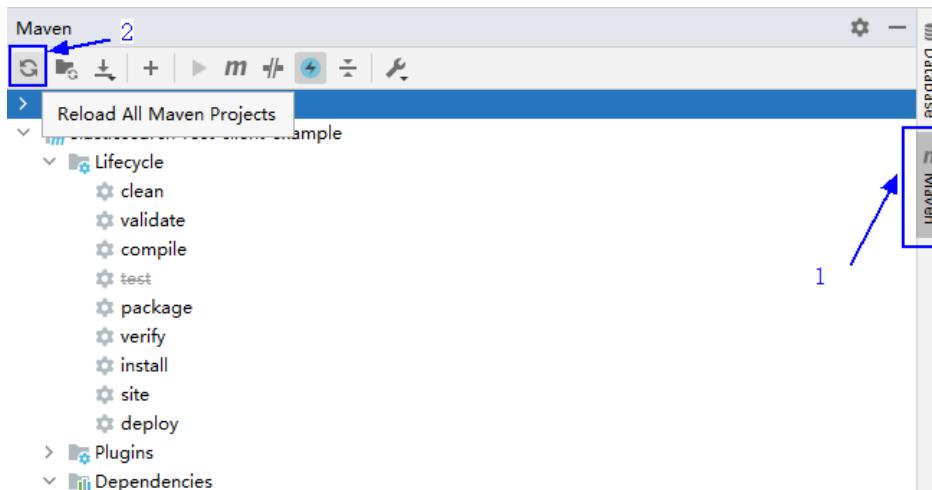
### 1.18.4.1.3 Commissioning a Spring Boot Program

You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

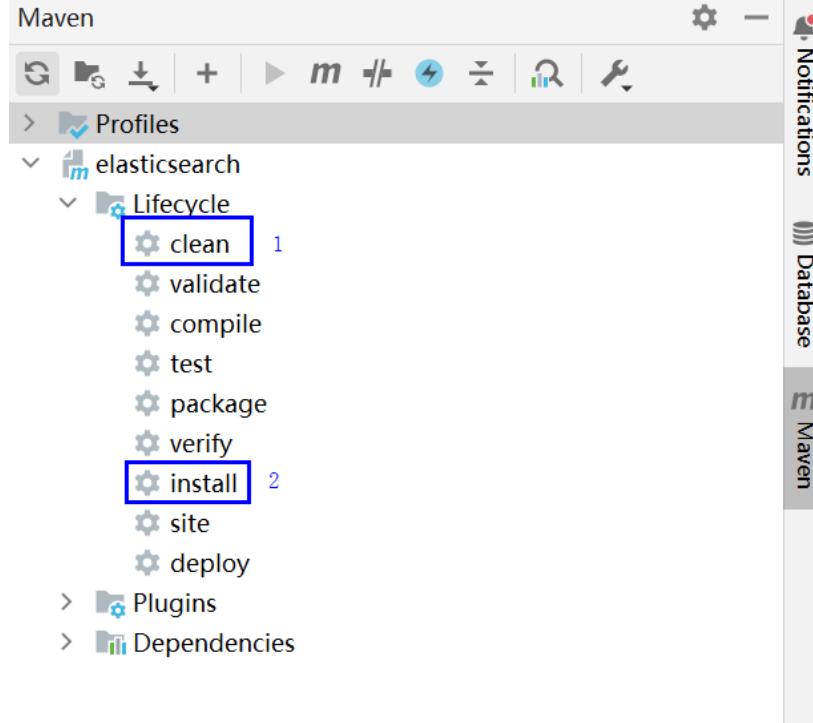
#### Step 1 Compile the Spring Boot sample to obtain **spring-boot-redis-1.0-SNAPSHOT.jar**.

1. Click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependency.

Figure 1-229 Importing dependencies



2. Double-click **clean** and **install** in sequence to run the **maven clean** and **maven install** commands. If "Build Success" is displayed, the compilation is successful.



## **Step 2** Read the configuration file.

1. Right-click **resource** in the sample code project and choose **Mark Directory as > Test Resource Root** from the shortcut menu.
  2. Decompress the user authentication file to obtain **krb5.conf** and **user.keytab**, and upload them to a local directory, for example, **D:\test**.
  3. Modify parameters in the **application.properties** file based on the site requirements.

### **Step 3** Run the applications.

Right-click **SpringDataRedisApplication** and choose **Run 'SpringDataRedisApplication...' from the shortcut menu** to start the Spring Boot service. If the Springboot service is started, the following information is displayed.

## Figure 1-230 Startup result

**Step 4** Open a browser, enter `http://localhost:9093/save` and save the data to Redis, as shown in the following figure.



Done

**Step 5** Open a browser, enter `http://localhost:9093/find?id=1` and save the data to Redis, as shown in the following figure.



a

----End

## 1.18.4.2 Commissioning an Application in Linux

### 1.18.4.2.1 Compiling and Running an Application

#### Scenario

In a Linux environment, you can upload the JAR package to the Linux and then run an application after the code development is complete.

#### Prerequisites

JDK has been installed.

#### Procedure

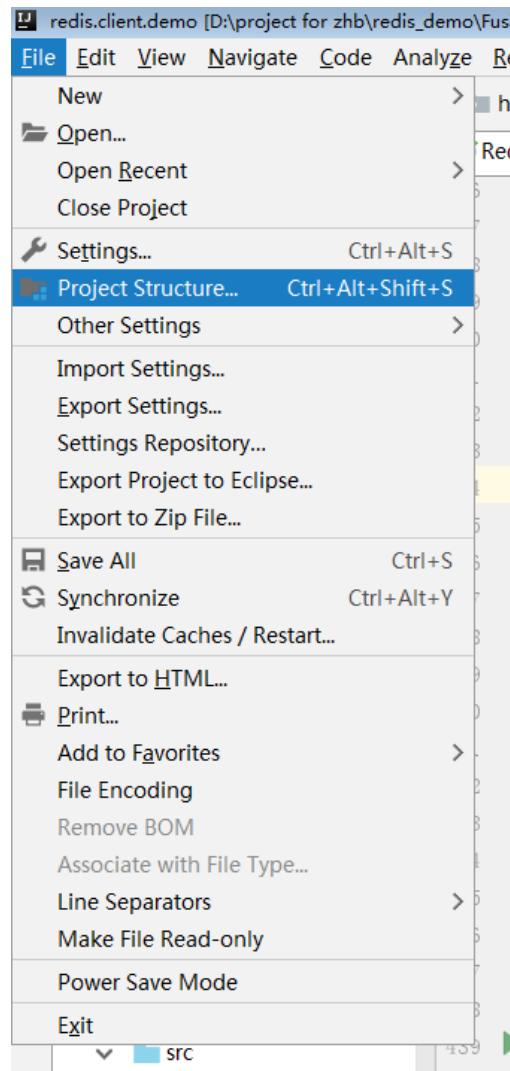
**Step 1** Export a JAR package.

1. On the IntelliJ IDEA redis-examples menu bar, choose **File > Project Structure....**



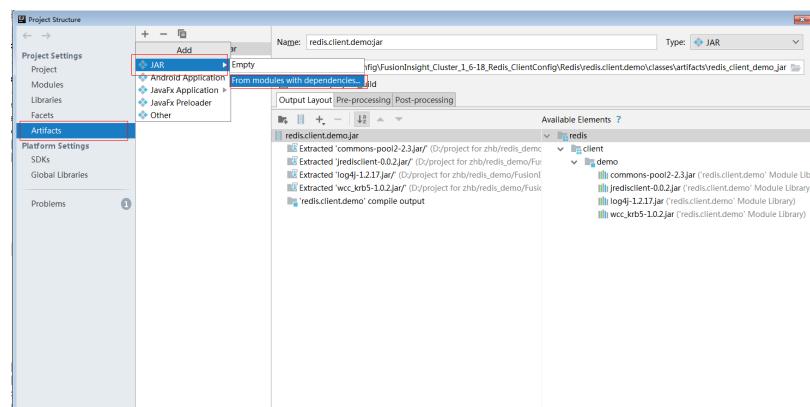
The preceding project name is for reference only. Use the actual project name in an actual scenario.

**Figure 1-231 "File > Project Structure..." shortcut menu**



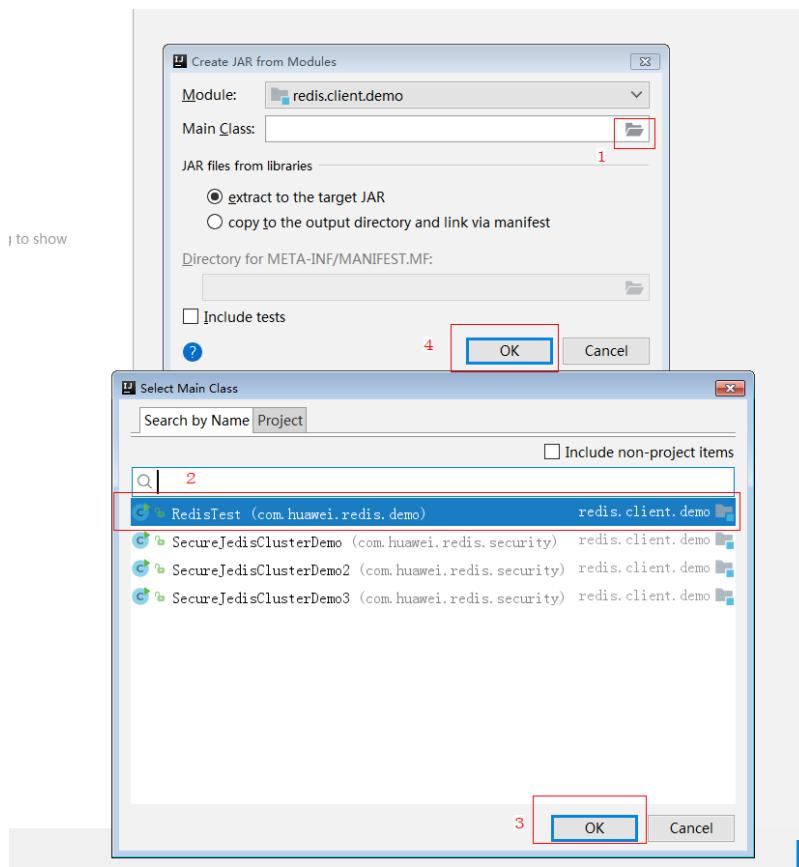
2. In the displayed **Project Structure** panel, select **Artifacts**. Then Click plus sign (+) and choose **JAR > From modules with dependencies**.

**Figure 1-232 Artifacts panel**



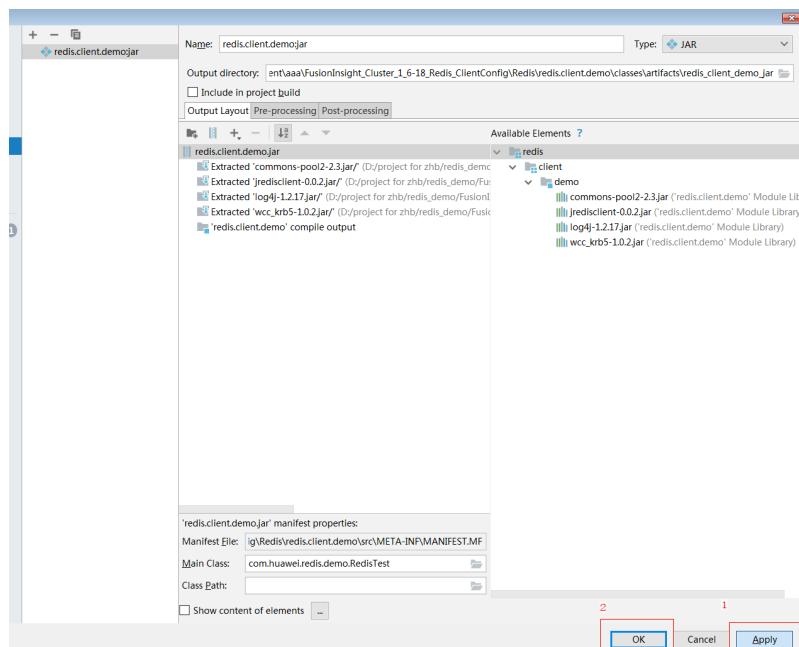
3. In the displayed **Create JAR from Modules** dialog box, click the folder icon, select **RedisTest** for **Main Class**, and click **OK**.

Figure 1-233 Specifying Main Class



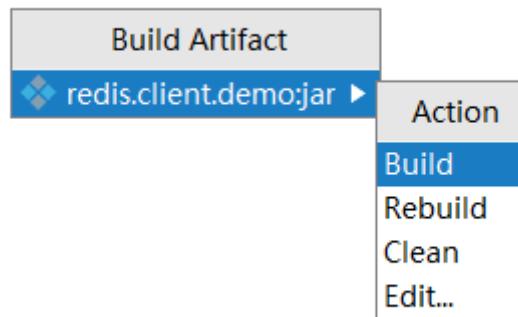
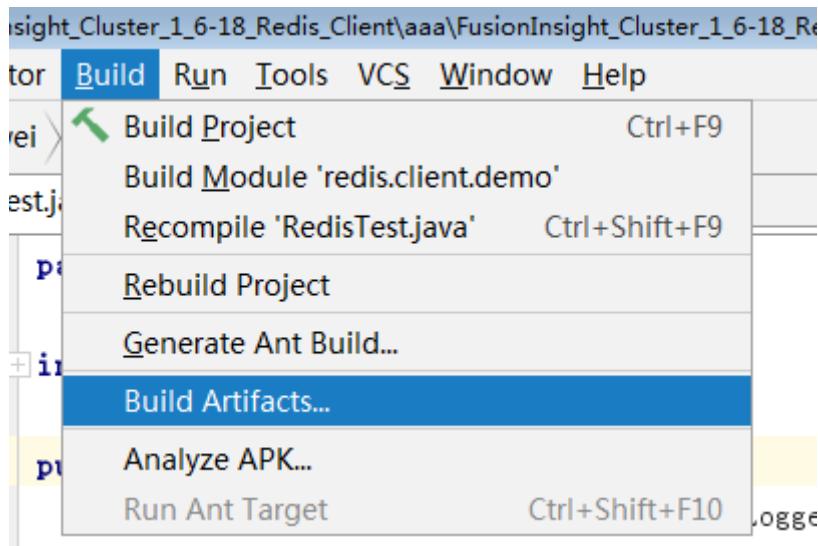
- Click the folder icon of **Output directory**, select the JAR package export path, and choose **Apply > OK**.

Figure 1-234 Selecting a JAR package export path



- Choose **Build > Build Artifacts**. In the displayed interface, choose **redis.client.demojar > Build** to export the JAR package.

Figure 1-235 export the JAR package



## Step 2 Prepare for the required JAR packages and configuration files.

In the Linux environment, create a directory, for example, `/opt/testone` and create subdirectories **config** and **lib**. Upload the JAR packages exported in **Step 1** to the **lib** directory in Linux. Upload the conf configuration files in the `/src/main/resources/config` of the sample project to the **config** directory in Linux.

## Step 3 Go to `/opt/testone`, Ensure that jdk has been installed, and java environment variables are set. Then, run the following commands to run the JAR packages:

```
java -cp .:lib/* com.huawei.redis.demo.RedisTest  
----End
```

### 1.18.4.2.2 Viewing Commissioning Results

#### Scenario

After a Redis application is run, you can check the running result through one of the following methods:<0>

- Viewing the Linux running result

- Logging in to the Redis Shell client to view the command output

## Procedure

- Viewing the Linux running result

The following success information will be displayed:

```
2019-06-22 16:23:27,519 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:93) - User sid-user01, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:37,921 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:101) - User sid-user01, session id: null
2019-06-22 16:23:38,084 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:107) - Value: hello
2019-06-22 16:23:38,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:111) - After append, value: hello world
2019-06-22 16:23:38,332 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:117) - User message, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:38,564 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:132) - All messages: [Hello how are you?, Fine thanks. I'm having fun with redis., I should look into this NOSQL thing ASAP]
2019-06-22 16:23:38,611 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:135) - Message count: 3
2019-06-22 16:23:38,665 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:139) - First message: Hello how are you?
2019-06-22 16:23:38,705 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:141) - After one pop, message count: 2
2019-06-22 16:23:39,016 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:157) - length of 1000000 is 7
2019-06-22 16:23:39,092 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:162) - User J001's name is John
2019-06-22 16:23:39,150 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:165) - {name=John, id=J001, gender=male, salary=1000000, age=35}
2019-06-22 16:23:39,430 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:179) - User L002's salary is 250000
2019-06-22 16:23:39,472 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:183) - all fields: [name, id, salary, gender, age]
2019-06-22 16:23:39,510 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:186) - all values: [L002, Lucy, 250000, 25, female]
2019-06-22 16:23:39,563 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:190) - partial field values: [L002, Lucy]
2019-06-22 16:23:39,605 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:194) - Exist field gender? true
2019-06-22 16:23:39,695 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:199) - after del field age, rest fields: [name, id, salary, gender]
2019-06-22 16:23:39,810 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:207) - key1's value is : value1
2019-06-22 16:23:39,888 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:211) - Can delete key by UNLINK.
2019-06-22 16:23:40,044 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:224) - Set size: 3
2019-06-22 16:23:40,122 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:228) - Set size: 3
2019-06-22 16:23:40,163 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:231) - Set: [TreeSet, HashSet, SortedSet]
2019-06-22 16:23:40,249 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:235) - Set: [TreeSet, HashSet]
2019-06-22 16:23:40,287 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:238) - TreeSet is set's member: true
2019-06-22 16:23:40,676 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:256) - All hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto, Linus Torvalds]
2019-06-22 16:23:40,718 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:259) - Size: 6
2019-06-22 16:23:40,767 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:262) - Score: 1969.0
2019-06-22 16:23:40,808 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:265) - Count: 2
2019-06-22 16:23:40,860 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:269) - All hackers 2: [Linus Torvalds, Yukihiro Matsumoto, Richard Stallman, Alan Kay, Claude Shannon, Alan Turing]
```

```
2019-06-22 16:23:40,942 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:273) - All  
hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto]  
2019-06-22 16:23:41,106 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:284) - TTL: 5  
2019-06-22 16:23:41,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:288) - KEY  
type: string  
2019-06-22 16:23:41,448 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:297) - List:  
[1, 4, 6, 3, 8]  
2019-06-22 16:23:41,490 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:300) - Sort  
list: [1, 3, 4, 6, 8]  
2019-06-22 16:23:42,129 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:343) - Result:  
[www.google.cn, www.baidu.com, www.sina.com]  
2019-06-22 16:23:42,301 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:60) - The  
distance between test1 and test2 is 96.3242 KM.  
2019-06-22 16:23:42,341 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:63) - Geo  
location info is [(10.000002086162567,30.000000249977013),  
(10.999999344348907,30.000000249977013)]  
2019-06-22 16:23:42,380 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:66) -  
geohash info of test1 and test2 is [sjr4et3f8v0, sjrfdm3fwto]  
2019-06-22 16:23:42,424 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:72) - Get  
location info by longitude and latitude :  
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test1 :  
146.8169  
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test2 :  
111.2263  
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test3 :  
1.0E-4  
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test4 :  
95.3394  
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test5 :  
482.4041  
2019-06-22 16:23:42,462 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:80) - Get  
location info by member :  
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test3 :  
0.0  
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test4 :  
95.3395  
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test2 :  
111.2264  
2019-06-22 16:23:42,657 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:312) - Byte  
conversion can be done by dump and restore
```

- Logging in to the Redis Shell client to view the command output  
After the sample code is executed, keys are deleted. Therefore, you cannot directly use the shell command to view the application running result after the sample project is executed. The following uses sample code of **String Type Access** to explain how to use the shell command to view the application running result:

- Comment out **client.del(key);** in the last line of the **TestString()** method.

```
...  
//client.del(key);
```

- Perform operations as instructed in [Compiling and Running an Application](#).

- Install the Redis client and use the Redis user for authentication.

```
cd /opt/hadoopclient  
source bigdata_env  
kinit redisuser
```

- Run the following command on the client node (<hostip> indicates any node in the Redis cluster):

```
cd /opt/client/Redis/bin  
redis-cli -c -p 22400 -h <hostip> get message
```

The value **A0BC9869FBC92933255A37A1D21167B2** set in the Java API is displayed:

"A0BC9869FBC92933255A37A1D21167B2"

 NOTE

If channel encryption is enabled for Redis, run `redis-cli -c -p 22400 --tls -h <hostip> get message`.

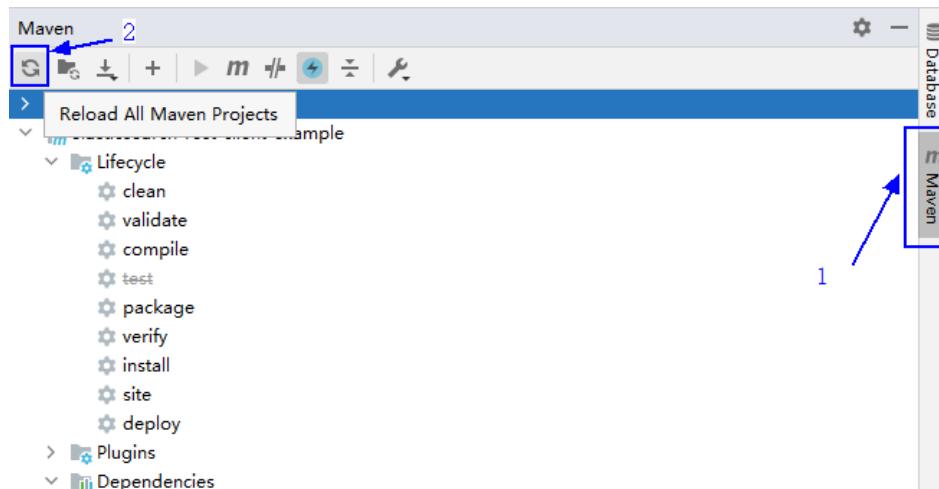
To enable channel encryption for Redis, log in to Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for **REDIS\_SSL\_ON**, and change the parameter value to **true** to enable SSL channel encryption.

#### 1.18.4.2.3 Commissioning the Spring Boot Program

**Step 1** Compile the Spring Boot sample to obtain **spring-boot-redis-1.0-SNAPSHOT.jar**.

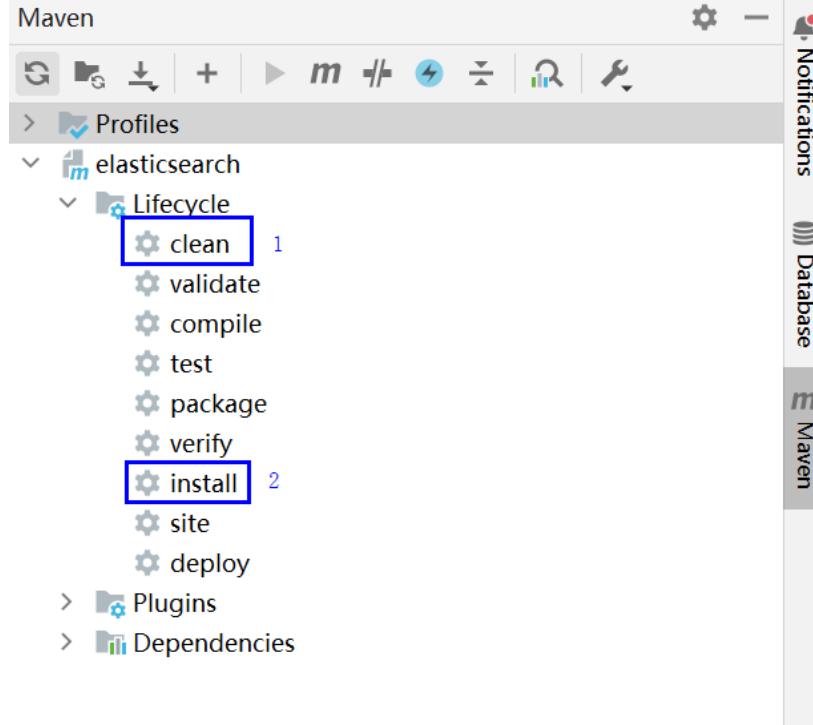
1. Click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependency.

**Figure 1-236** Importing dependencies



2. Double-click **clean** and **install** in sequence to run the **maven clean** and **maven install** commands. If "Build Success" is displayed, the compilation is successful.

A JAR file containing the **spring-boot-redis-** field is generated in the **target** directory of the sample project.



**Step 2** Create the `/opt/spring` directory on the Linux OS and upload the JAR packages whose names contain `spring-boot-redis-` in the target directory generated in **Step 1** to the directory.

**Step 3** Create the `/opt/spring/config` directory on Linux, modify the `application.properties` file based on the site requirements, and upload the file to the current directory.

**Step 4** Decompress the user authentication file to obtain **krb5.conf** and **user.keytab**, and upload them to the **/opt/spring/config** directory.

**Step 5** Run the following command in `/opt/spring/config` to start Spring Boot:

```
java -jar spring-boot-redis-1.0-SNAPSHOT.jar
```

The command output is as follows.

```
[root@宿主-192-168-64-138 ~]# spring -java -jar ./spring-boot-redis-1.0-SNAPSHOT.jar
```



Spring Boot (v2.4.2)

```
2022-12-16 14:37:37.722 INFO 10437 --- [ main] c.j.redis.SpringDataRedisApplication : Starting SpringDataRedisApplication v1.0-SNAPSHOT using Java 1.8.0_332 on 192-168-64-138 with PID 10437 (/opt/spring/spring-boot-redis-1.0-SNAPSHOT.jar)
```

```
2022-12-16 14:37:37.722 INFO 10437 --- [ main] c.j.redis.SpringDataRedisApplication : No active profile set, falling back to default profiles: default
```

```
2022-12-16 14:37:39.930 INFO 10437 --- [ main] o.e.j.s.p.ContainerInitializerFactory : Logging initialized at 450ms to org.eclipse.jetty.util.log.Slf4jLog
```

```
2022-12-16 14:37:39.930 INFO 10437 --- [ main] o.e.j.s.p.ContainerInitializerFactory : jetty-9.4.35.v20201210; built: 2020-11-20T21:21:07Z; hex: 0064; git: bdc54f03a5e07a087ab27f55c35c7eebd8d9fb; jvm: 1.8.0_332-Bisheng_3K_Enterprise
```

```
2022-12-16 14:37:39.930 INFO 10437 --- [ main] o.e.j.s.p.ContainerInitializerFactory : Root WebApplicationContext: initialization completed in 2115 ms
```

```
2022-12-16 14:37:39.930 INFO 10437 --- [ main] o.e.j.s.p.ContainerInitializerFactory : No SessionAware beans found, using defaults
```

```
2022-12-16 14:37:39.930 INFO 10437 --- [ main] o.e.j.s.p.ContainerInitializerFactory : Node Scanning every 50000ms
```

```
2022-12-16 14:37:39.930 INFO 10437 --- [ main] o.e.j.s.p.ContainerInitializerFactory : Started @630ms
```

```
2022-12-16 14:37:39.930 INFO 10437 --- [ main] o.a.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
```

```
2022-12-16 14:37:41.229 INFO 10437 --- [ main] o.a.s.c.a.ContextHandlerApplication : Initializing DispatcherServlet 'dispatcherServlet'
```

```
2022-12-16 14:37:41.229 INFO 10437 --- [ main] o.a.s.c.a.ContextHandlerApplication : Initializing ExecutorService 'applicationTaskExecutor'
```

```
2022-12-16 14:37:41.229 INFO 10437 --- [ main] o.a.s.c.a.ContextHandlerApplication : Completed initialization in 1 ms
```

```
2022-12-16 14:37:41.229 INFO 10437 --- [ main] o.a.s.c.a.ContextHandlerApplication : Jetty-9.4.35.v20201210, (http://0.0.0.0:9093)
```

```
2022-12-16 14:37:41.405 INFO 10437 --- [ main] o.a.s.c.a.ContextHandlerApplication : Jetty started on port(s) 9093 (http/1.1) with context path '/'
```

```
2022-12-16 14:37:41.405 INFO 10437 --- [ main] c.j.redis.SpringDataRedisApplication : Started SpringDataRedisApplication in 0.81 seconds (JVM running for 6.019)
```

```
"2022-12-16 14:37:40.461,949 [main] INFO 10437 --- [ extShutdownHook-0] o.a.j.s.p.ContainerInitializer : Stopped ServerConnector@704B704[HTTP/1.1, {/}] (0.0.0.0:9093)
```

```
2022-12-16 14:37:40.461,949 [main] INFO 10437 --- [ extShutdownHook-0] o.a.j.s.p.ContainerInitializer : Destroying Spring FrameworkServletDispatcherServlet
```

```
2022-12-16 14:37:40.461,949 [main] INFO 10437 --- [ extShutdownHook-0] o.a.j.s.p.ContainerInitializer : Stopped SpringDataRedisApplication v1.0-SNAPSHOT (/opt/spring/spring-boot-redis-1.0-SNAPSHOT.jar)
```

```
2022-12-16 14:37:40.461,949 [main] INFO 10437 --- [ extShutdownHook-0] o.a.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService 'applicationTaskExecutor'
```

**Step 6** Open a browser, enter `http://IP address of the Linux server:9093/save` in the address box, and save the data to Redis, as shown in the following figure.



**Step 7** Open a browser, enter `http://IP address of the Linux server:9093/find?id=1`, and save the data to Redis, as shown in the following figure.



## 1.18.5 More Information

### 1.18.5.1 External Interfaces

#### 1.18.5.1.1 Shell

You can directly perform operations on Redis using Shell on the server. Versions of Shell interfaces of Redis need to be consistent with those in the open-source community. For details, see <http://redis.io/commands>.

Methods of running the Shell command:

**Step 1** After the client is installed, go to the Redis client installation directory and run the `source bigdata_env` command. For a cluster in security mode, run the `kinit login user name`.

**Step 2** run the following command:

`redis-cli -h IP -p port`

#### NOTE

If channel encryption is enabled for Redis, run `redis-cli -c -h IP -p port --tls`

To enable channel encryption for Redis, log in to Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for **REDIS\_SSL\_ON**, and change the parameter value to **true** to enable SSL channel encryption.

**Step 3** Go to the running mode of the Redis command (also called CLI client connection).  
192.168.0.11:22400>

**Step 4** Run the `help` command to obtain the help information of the Redis command parameters.

----End

### 1.18.5.1.2 Java API

Redis APIs are consistent with community Jedis APIs. For details, see <https://github.com/xetorthio/jedis>.

### 1.18.5.2 Performance Optimization

#### 1.18.5.2.1 Optimized Redis Write/Read Performance in Security Mode

##### Disabling channel encryption

Check whether channel encryption is enabled in the Redis. If services have high requirements and no sensitive information is stored in the Redis, disable channel encryption.

- Step 1** Log in to FusionInsight Manager.
  - Step 2** choose **Cluster > Services > Redis**. On the page that is displayed, click the **Configurations** tab then the **All Configurations** sub-tab. On this sub-tab page, search for **REDIS\_SSL\_ON**. If its value is **true**, SSL channel encryption has been enabled for Redis. To disable channel encryption, change the value of **REDIS\_SSL\_ON** to **false**.
  - Step 3** Click the **Dashboard** tab, choose **More > Restart Service** to restart the Redis service.
- End

##### The number of client connections is optimized.

When a service application accesses the Redis cluster in security mode, security authentication is required for each connection to the Redis server. If connections are frequently set up in service applications, read/write performance is affected, leading to long delay. In this case, you are advised to use the permanent connection pool to connect service applications to the server. The following is an example:

```
JedisPoolConfig poolConfig = new JedisPoolConfig();
//Maximum number of connections in the connection pool, which is 8 by default.
poolConfig.setMaxTotal(50);

//Maximum number of idle connections in the connection pool, which is 8 by default.
poolConfig.setMaxIdle(50);
poolConfig.setMinIdle(0);

//Minimum idle time of the connection. When the time is reached, the idle connection is removed. The
recommended value is -1, indicating that the connection is not removed.
poolConfig.setMinEvictableIdleTimeMillis(-1);
poolConfig.setSoftMinEvictableIdleTimeMillis(-1);

JedisCluster client = new JedisCluster(hosts, 15000, poolConfig);
```

The client does not invoke the `close()` method after each read/write operation. The `close()` method is invoked when the service application exits.

### Adding the Client Environment to a Domain

In security mode, when accessing the Redis, ensure that the node where the client resides and the Redis service are in the same domain. For details about how to

install the client, see section [Development and Operating Environment](#) to ensure that the client and server are in the same domain.

## 1.19 RTD Development Guide

### 1.19.1 Overview

#### 1.19.1.1 Application Development Overview

FusionInsight RTD is an enterprise-level distributed real-time decision-making platform that suits for mass data processing in high-concurrency and low latency scenarios. It supports user-defined rules and horizontal expansion. It bridges the "last mile" towards business decision-making, providing enterprises with precise risk control and marketing.

#### 1.19.1.2 Common Concepts

FusionInsight RTD is a component of MRS. It provides the following functions:

- RTDService
  - As the core component of RTD, RTDService functions as the unified web definition entry of RTD and allows users to define tenants, event sources, dimensions, variables, models, and rules. RTDService is deployed in active/standby mode to ensure availability.
- Containers
  - The Containers service provides physical environments for the running of Business Logic Unit (BLU) instances and controls the start and stop of the BLUs.
  - The Containers service provides Access Load Balance (ALB) to connect to load balancers. ALB implements socket access. Specifically, it distributes requests of different projects to service instances on the platform based on different processing policies and implements conversion between protocol interfaces. ALB is not provided as an independent service but integrated in Containers.
- MOTService
  - MOTService provides fast and large-throughput access capabilities and uses stored procedures to quickly process service logic at the database layer. It is deployed in active/standby mode.

#### 1.19.1.3 Development Process

This section describes how to use Java APIs to develop RTD applications.

[Figure 1-237](#) and [Table 1-168](#) describe the phases in the development process.

Figure 1-237 RTD development process

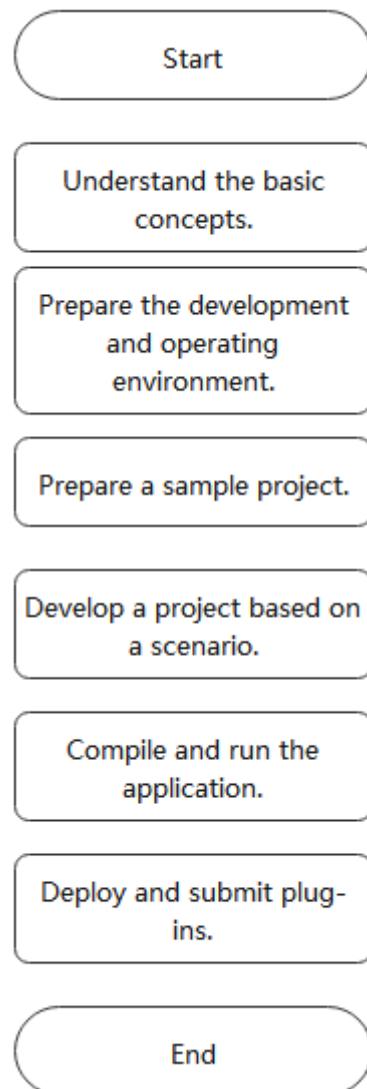


Table 1-168 Description of RTD development process

Step	Description	Reference
Understand the basic concepts.	Before developing an application, learn basic concepts of RTD and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>
Prepare the development and operating environment.	The Java language is recommended for RTD application development. You can use the IntelliJ IDEA tool.	<a href="#">Preparing the Development and Operating Environment</a>

Step	Description	Reference
Prepare a sample project.	RTD provides sample projects in various scenarios. Sample projects can be imported for studying.	<a href="#">Configuring and Importing Sample Projects</a>
Develop a project based on a scenario.	Decision engine and decision-making flow customization are supported, and dynamic variables can be extended.	<a href="#">Developing an Application</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Compiling and Packaging</a>

## 1.19.2 Environment Preparation

### 1.19.2.1 Preparing the Development and Operating Environment

[Table 1-169](#) describes the environment required for application development.

**Table 1-169** Items needed for environment preparation

Item	Description
OS	Windows 7 or later is supported. Network environment: The local development environment must interconnect with the cluster service-plane network.
JDK installation	Basic configuration of the development and running environment. The version requirements are as follows: The JDK version must be the same as the version of FusionInsight Manager to be accessed. For details about the JDK version, see the corresponding version document or contact the system administrator.
IntelliJ IDEA installation and configuration	Basic configuration of the development environment. The version must be 2019.1 or other compatible versions.

## 1.19.2.2 Configuring and Importing Sample Projects

### Procedure

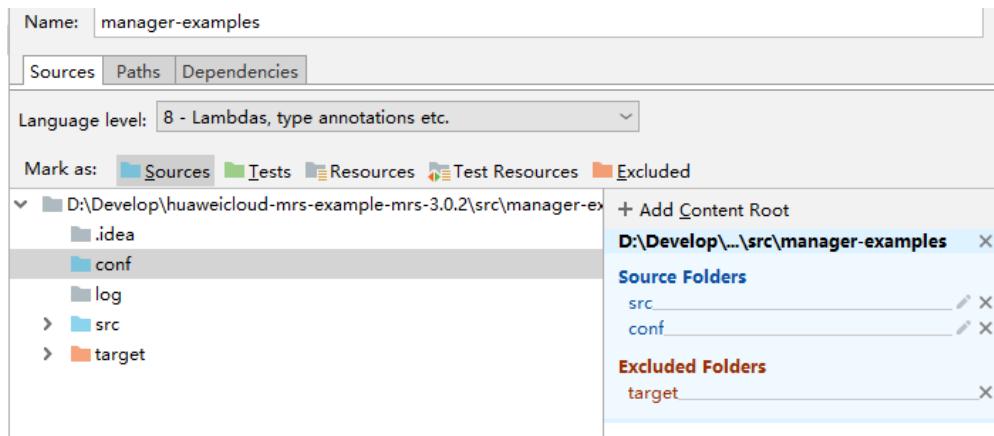
- Step 1** Obtain the sample project folder **rtd-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
1. Open IntelliJ IDEA and choose **File > New > Project from Existing Sources**. In the displayed **Select File or Directory to Import** window, select the sample code folder.
  2. Select the **pom.xml** file in the sample project folder, select **Import project from external model > Maven** as prompted, and click **Next** until **Finish** is displayed.

#### NOTE

The sample code is a Maven project. You can adjust the project configuration as required. The operations vary according to the IntelliJ IDEA version.

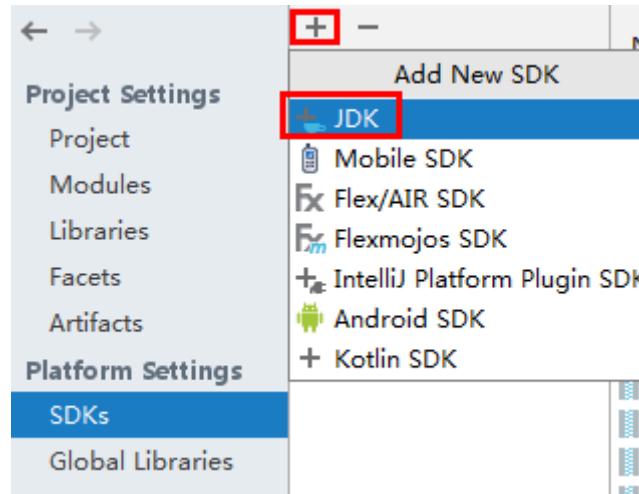
3. Add the **src** and **conf** directories in the project to the source file path. After the project is imported, choose **File > Project Structure** on the menu bar of IntelliJ IDEA. In the displayed window, choose **Project Settings > Modules**.

Select the current project, click the **src** folder, click **Sources** on the right of **Mark as**, click the **conf** folder, and click **Sources** on the right of **Mark as**. Click **Apply** and then **OK**.

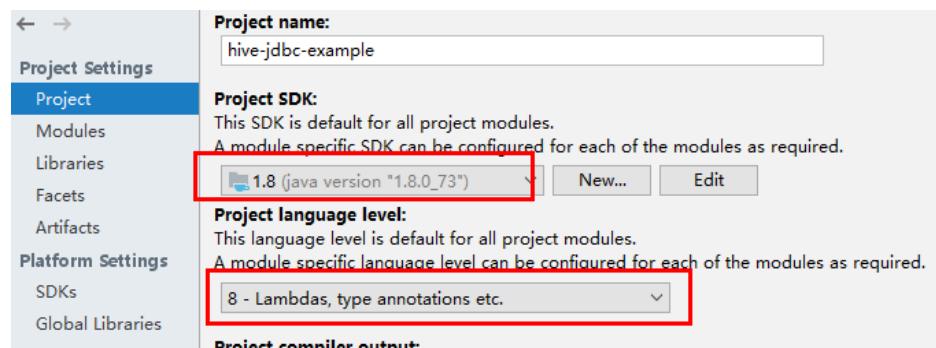


- Step 3** Set the JDK of the project.

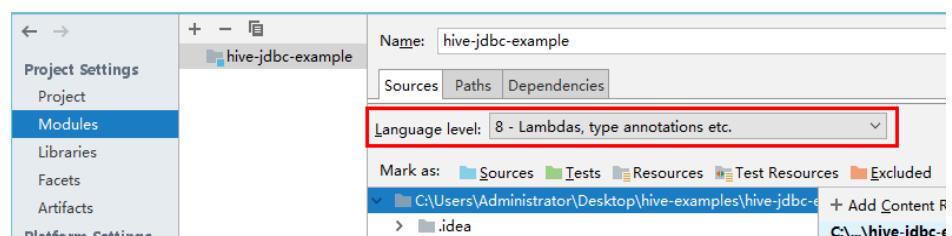
1. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
2. Choose **SDKs**, click the plus sign (+), and select **JDK**.



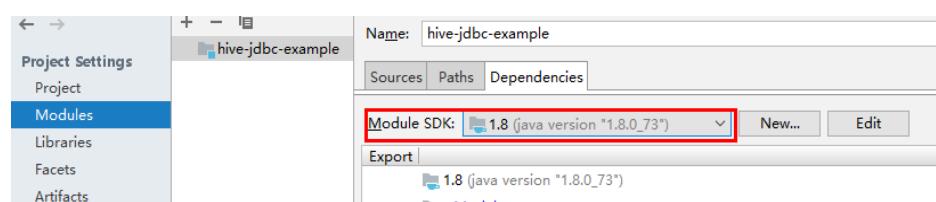
3. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
4. Click **Apply**.
5. Select **Project**, select the JDK added in **SDKs** from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



6. Click **Apply**.
7. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.

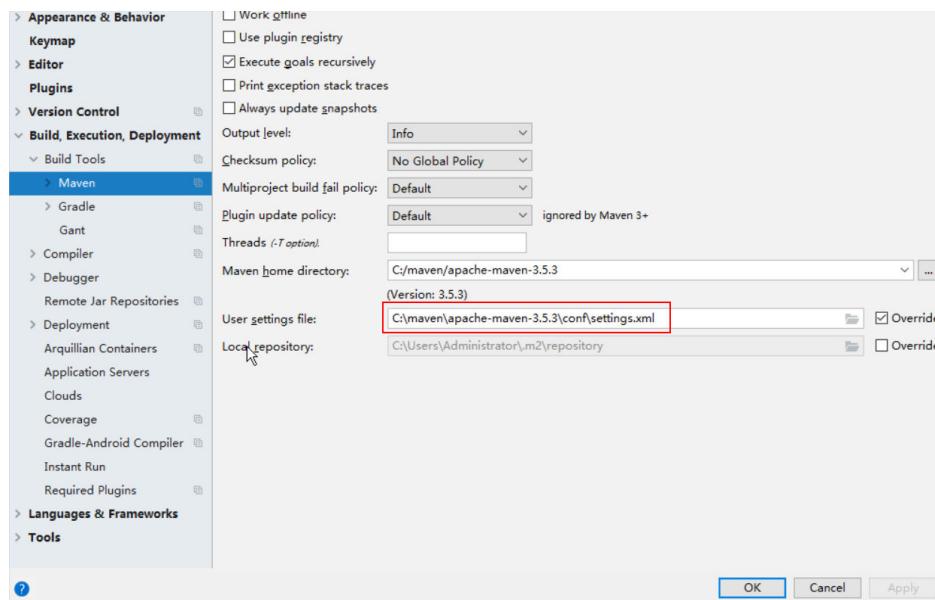


8. Click **Apply** and then **OK**.

**Step 4** Configure Maven.

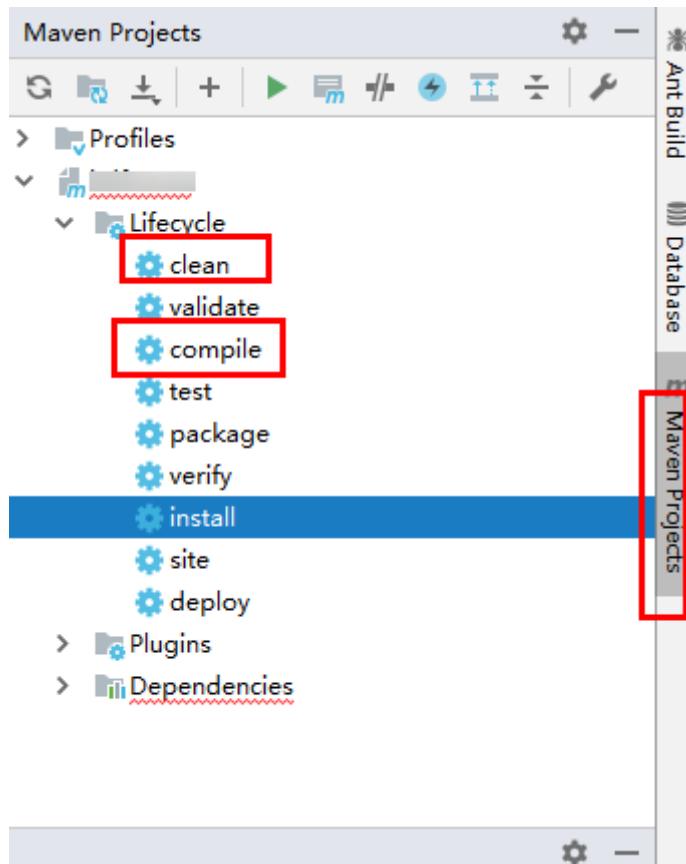
1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is *<Local Maven installation directory>\conf\settings.xml*.

**Figure 1-238** Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply** and then **OK**.
5. On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the displayed page, choose **Project name > Lifecycle** and run the **clean** and **compile** scripts.

Figure 1-239 Maven Projects page



----End

## 1.19.3 Developing an Application

### 1.19.3.1 Event Source Custom Plug-in Program

#### 1.19.3.1.1 Scenario

Users can customize and orchestrate decision-making flows and extended variables based on actual service requirements to meet more complex service requirements.

#### Data Preparation

Use Maven to package the entire project. By default, the following two files are generated:

- rtdexecutor-plugins-bundle-\*-.zip
  - This file contains only the content related to the DecisionFlowPlugin module.
  - If the content related to the DecisionFlowPlugin module is modified, the associated BLU will be restarted (rolling restart) after the update is performed on the RTD web UI.

- **rtdexecutor-plugins-bundle-\*-update.zip**
  - This file contains the contents of all modules of the current project.
  - If the content related to the VarsExtensionPlugin module is modified and updated on the RTD web UI, the update is loaded in hot loading mode and the associated BLU is not restarted.

 NOTE

You can modify the **assembly** configuration file as required.

## File Modification

Modify the **pom.xml** and **plugin.properties** files for each submodule in the project.

- **pom.xml**

```
<properties>
<plugin.id>DecisionFlowPlugin</plugin.id>
<plugin.class>com.huawei.fi.plugins.ExampleDecisionFlowPlugin</plugin.class>
<plugin.version>0.1.0</plugin.version>
<plugin.provider>cmb</plugin.provider>
</properties>
```

- **plugin.properties**

```
plugin.id=DecisionFlowPlugin
plugin.class=com.huawei.fi.plugins.decisionflow.ExampleDecisionFlowPlugin
plugin.version=0.1.0
plugin.provider=cmb
```

 NOTE

Modify all parameters except **plugin.id** based on the site requirements.

### 1.19.3.1.2 Development Guidelines

#### Project Description

- Project name: **rtdexecutor-plugins**
- The following sub-modules are included:
  - DecisionFlowPlugin: used for custom orchestration of decision-making flows and can contain the **RTDEventHandler** class implemented by users.
  - VarsExtensionPlugin: used for packet extension and variable extension. You can implement the subclasses of **EventFieldsExtension** and **DynamicVarsExtension** to customize packets and variables.
- The names of sub-modules in a project are fixed and cannot be customized.
- The project must contain the **DecisionFlowPlugin** submodule. The **VarsExtensionPlugin** submodule is optional.

### 1.19.3.1.3 Sample Code

#### DecisionFlowPlugin

The following code snippets are stored in **com/huawei/fi/plugins/decisionflow**.

- **ExampleDecisionFlowPlugin.java**

The custom plug-in needs to inherit the plug-in class of **pf4j** and overwrite the default construction method, as well as **start** and **stop** methods. When

the BLU associated with the RTD event source is started or stopped, the **start** or **stop** method is invoked.

```
public class ExampleDecisionFlowPlugin extends Plugin
{
    private static final Logger LOG = Logger.getLogger(ExampleDecisionFlowPlugin.class);

    public ExampleDecisionFlowPlugin(PluginWrapper wrapper)
    {
        super(wrapper);
    }

    @Override
    public void start()
    {
        LOG.info("ExampleDecisionFlowPlugin.start()");
    }

    @Override
    public void stop()
    {
        LOG.info("ExampleDecisionFlowPlugin.stop()");
    }
}
```

- **ExampleScoreEventHandler.java**

Implement the **RTDEventHandler** subclass based on the site requirements.  
This is optional.

```
public class ExampleScoreEventHandler extends RTDEventHandler
{
}
```

- **ExampleDecisionFlowExtension.java**

- Inherit **DecisionFlowExtension** and add **pf4j** annotations to the subclass so that the subclass can be loaded by **pf4j**.

```
@Extension
public class ExampleDecisionFlowExtension extends DecisionFlowExtension
{
}
```

- Overwrite the **getPipelineDecisionServiceMode** method of **DecisionFlowExtension** to set the decision-making type (real-time or quasi-real-time mode) based on the service scenario.

```
/**
 * Obtain the decision-making type and customize it based on the business scenario
 * requirements.
 * Real-time mode: Constants.DECISION_SERVICE_MODE_RT
 * Quasi-real-time mode: Constants.DECISION_SERVICE_MODE_NRT
 * @return int decision mode
 */
@Override
public int getPipelineDecisionServiceMode()
{
    return Constants.DECISION_SERVICE_MODE_RT;
}
```

- Override the **orchestrateDecisionFlow** method of **DecisionFlowExtension** to orchestrate the **handler** process of a custom decision-making flow based on service requirements.

```
@Override
public void orchestrateDecisionFlow(RTDPipeline pipeline)
{
    String runtimeModel = RTDEnvironment.getProperty("plugin.RuntimeMode");
    LOG.info("RuntimeMode: " + runtimeModel + ".");
    // for testing the development mode
    if (RuntimeMode.DEVELOPMENT.toString().equalsIgnoreCase(runtimeModel))
    {
```

```
        pipeline.setInboundEventHandler(new EventInboundEventHandler());
        pipeline.setOutboundEventHandler(new EventOutboundEventHandler());

        pipeline.addEventHandler(new DataPreEventHandler());
    }
else
{
    pipeline.setInboundEventHandler(new EventInboundEventHandler());
    pipeline.setOutboundEventHandler(new EventOutboundEventHandler());
    // Instructions:
    // 1. If the custom DataPreEventHandler is used to extend event variables,
    // pay attention to the package path of the DataPreEventHandler class.
    // Note: This mode does not support online dynamic update.
    // 2. If VarsExtensionPlugin is used to extend event variables,
    // DataPreEventHandler must be placed at the beginning of the entire pipeline,
    // DataPreEventHandler is a Huawei JAR built-in class.
    // The customer logic is in Extension of the VarsExtensionPlugin plug-in.
    // Note: This mode supports online dynamic update.
    pipeline.addEventHandler(new DataPreEventHandler());
    pipeline.addEventHandler(new InsertDataEventHandler());
    pipeline.addEventHandler(new CallFilterRulesEventHandler());
    pipeline.addEventHandler(new ComputeVarsEventHandler());
    // 1. If VarsExtensionPlugin is used to dynamically extend variables,
    // Add DynamicVarsEventHandler to the pipeline.
    // 2. If there is no dynamic extended variable, delete the handler from the pipeline.
    // 3. DynamicVarsEventHandler must be placed after ComputeVarsEventHandler.
    pipeline.addEventHandler(new DynamicVarsEventHandler());
    // Add the scoring model capability to the pipeline.
    pipeline.addEventHandler(new ScoreModelEventHandler());
    pipeline.addEventHandler(new CallProcRulesEventHandler());
    // The logic for adding custom decision-making is the same as the existing method.
    pipeline.addEventHandler(new ExampleScoreEventHandler());
}
}
```

- d. If you need to use the built-in packet extension and variable extension functions, import the class path of the handler when orchestrating the handler in the pipeline in 3.

```
import com.huawei.fi.rtdexecutor.RTDEnvironment;
import com.huawei.fi.rtdexecutor.common.Constants;
import com.huawei.fi.rtdexecutor.plugin.DecisionFlowExtension;
import com.huawei.fi.rtdexecutor.rtdeventhandlers. DataPreEventHandler;
import com.huawei.fi.rtdexecutor.rtdeventhandlers. DynamicVarsEventHandler;
import com.huawei.fi.rtdexecutor.rtdpipeline.RTDPipeline;
```

## VarsExtensionPlugin

The following code snippets are in the **com.huawei.fi.plugins.varsextension** package.

- **ExampleVarsExtensionPlugin.java**

The custom plug-in needs to inherit the plug-in class of **pf4j** and overwrite the default construction method, as well as **start** and **stop** methods. When the BLU associated with the RTD event source is started or stopped, the **start** or **stop** method is invoked.

In the current version, only one implementation of **EventFieldExtension** and **DynamicVarsExtension** is loaded for **ExampleVarsExtensionPlugin**. Even if the plug-in contains multiple implementations of **EventFieldExtension** and **DynamicVarsExtension**, only one implementation is loaded.

```
public class ExampleVarsExtensionPlugin extends Plugin
{
    private static final Logger LOG = Logger.getLogger(ExampleVarsExtensionPlugin.class);

    public ExampleVarsExtensionPlugin(PluginWrapper wrapper)
    {
```

```

        super(wrapper);
    }

    @Override
    public void start() { LOG.info("ExampleVarsExtensionPlugin.start()"); }

    @Override
    public void stop()
    {
        LOG.info("ExampleVarsExtensionPlugin.stop()");
    }
}

```

- **ExampleEventFieldsExtension.java**

- In this example, packet field extension is implemented, and the BLU does not need to be restarted when **VarsExtensionPlugin** is updated. Inherit the **EventFieldsExtension** class, add the **pf4j** extension point annotation **@Extension**, and rewrite the parent class method. In the **init/destroy** method, initialize or destroy basic resources related to packet extension. **init/destroy** is invoked when the BLU associated with the event source is restarted or the plug-in is updated.

```

@Extension
public class ExampleEventFieldsExtension extends EventFieldsExtension
{
    @Override
    public void init()
    {
        LOG.info("{} init start ...", this.getClass().getName());
        super.init();
        // add your init code at here
        this.timeoutMS = 20L;
        LOG.info("{} init end ...", this.getClass().getName());
    }

    @Override
    public void destroy()
    {
        LOG.info("{} destroy start ...", this.getClass().getName());
        super.destroy();
        // add your destroy code at here
        LOG.info("{} destroy end ...", this.getClass().getName());
    }
}

```

- Implement the **compute** method in **EventFieldsExtension**. The definition and calculation of extended packets are implemented in this method. In the following example code, the **BANK\_CUSTOMER\_ID**, **BANK\_CUSTOMER\_AGE** and **BANK\_CUSTOMER\_SALARY** fields are extended.

```

@Override
public void compute(PipelineContext ctx, EventData eventData) throws EventHandlerException
{
    ExtendEventField<String> bankCustomerID = new
    ExtendEventField.String("BANK_CUSTOMER_ID");
    ExtendEventField<Integer> bankCustomerAge = new
    ExtendEventField.Integer("BANK_CUSTOMER_AGE");
    ExtendEventField<Double> bankCustomerSalary = new
    ExtendEventField.Double("BANK_CUSTOMER_SALARY");

    bankCustomerAge.setValue(32);
    bankCustomerID.setValue("0000001");

    String city = (String) eventData.getData().get("LBS_CITY");
    if("Boston".equals(city)) {
        bankCustomerSalary.setValue(7830.25);
    } else if("New York".equals(city)) {

```

```

        bankCustomerSalary.setValue(9765.40);
    } else {
        bankCustomerSalary.setValue(6300.62);
    }

    putComputeResult(ctx, bankCustomerID.getName(), bankCustomerID);
    putComputeResult(ctx, bankCustomerAge.getName(), bankCustomerAge);
    putComputeResult(ctx, bankCustomerSalary.getName(), bankCustomerSalary);
}

```

- After the plug-in code is developed, use the packet field definition function of event source management on the RTDService web UI to map the plug-in code to event variables for subsequent calculation. In addition, the default value is defined for this function. If the packet extension calculation in the plug-in fails, the default value defined here will be used in subsequent calculation steps.

## DynamicVarsExtension

- In the ExampleDynamicVarsExtension.java example, variables are dynamically extended. When **VarsExtensionPlugin** is updated, the BLU does not need to be restarted. Inherit the **DynamicVarsExtension** class, add the **pf4j** extension point annotation **@Extension**, and rewrite the parent class method. In the **init/destroy** method, initialize or destroy basic resources related to dynamic extended variables. **init/destroy** is invoked when the BLU associated with the event source is restarted or the plug-in is updated.

```

@Extension
@SuppressWarnings({"rawtypes", "unchecked"})
public class ExampleDynamicVarsExtension extends DynamicVarsExtension
{
    /**
     * init your public resource at here,
     * plugins create or update, this method will be invoked
     *
     */
    @Override
    public void init()
    {
        LOG.info("{} init start ...", this.getClass().getName());
        super.init();
        // add your init code at here
        this.timeoutMS = 20L;
        LOG.info("{} init end ...", this.getClass().getName());
    }

    /**
     * recycle your resource at here,
     * plugins destroy or update, this method will be invoked
     *
     */
    @Override
    public void destroy()
    {
        LOG.info("{} destroy start ...", this.getClass().getName());
        super.destroy();
        // add your destroy code at here
        LOG.info("{} destroy end ...", this.getClass().getName());
    }
}

```

- When implementing the **DynamicVarsExtension** subclass, you can override the **verify** method to add some custom checks. This method is invoked when the plug-in is uploaded. If **false** is returned, the update fails.

```

public abstract class DynamicVarsExtension implements ExtensionPoint
{

```

```
public boolean verify()
{
    return true;
}
```

- To dynamically extend variables, implement the **addRequiredInputVars** method to define the event variables, real-time query variables, and batch variables that these variables depend on during calculation. When the update plug-in is uploaded, the system verifies the existence and online status of the variables defined in the method. If the verification fails, the system displays an update error. The definition of the dependent variable is very important. If the dependent variable is not defined or omitted, the calculation of the dynamic extended variable may be abnormal or incorrect, affecting the decision-making result of the request. In the following example, the plug-in depends on event variable **ev\_client\_ip**, real-time query variable **rv\_online\_pay**, and batch variable **bv\_history\_pay**.

```
@Override
public List<String> addRequiredInputVars()
{
    // define input vars which required in your code and built-in RTD platform,
    // include event vars, batch vars, runtime query vars
    List<String> inputVars = new ArrayList<>();
    inputVars.add("ev_client_ip");
    inputVars.add("rv_online_pay");
    inputVars.add("bv_history_pay");

    return inputVars;
}
```

- To dynamically extend variables, implement the **defineDynamicEventVars** method to define the dynamic extended variables contained in the **DynamicVarsExtension** implementation class. The dynamic extended variable must start with **ev\_d\_**, and the type and default value must be specified. In the following example, two dynamic extended variables are defined. The names are **ev\_d\_local\_city** and **ev\_d\_total\_pay**, the types are **String** and **Long**, and the default values are **Boston** and **11**. Currently, dynamic extended variables support five types: Integer, Long, String, Double, and Timestamp.

To dynamically extend variables, you need to implement the **compute** method and calculate the variables based on the variable values in the current pipeline request. The obtained variable values must be defined in **addRequiredInputVars** of the current implementation class.

```
@Override
public Map<String, DynamicEventVar<?>> defineDynamicEventVars()
{
    DynamicEventVar<String> localCity = new DynamicEventVar.String("ev_d_local_city", "Boston");
    DynamicEventVar<Long> totalPay = new DynamicEventVar.Long("ev_d_total_pay", 11L);

    Map<String, DynamicEventVar<?>> dynamicEventVarMap = new HashMap<>();
    dynamicEventVarMap.put(localCity.getName(), localCity);
    dynamicEventVarMap.put(totalPay.getName(), totalPay);

    return dynamicEventVarMap;
}
```

- During the calculation, when obtaining the dependent variable value from the pipeline, check whether the value is null and matches the type. Otherwise, null pointers and type conversion exceptions may occur.

```
@Override
public void compute(PipelineContext ctx, EventData eventData) throws EventHandlerException
{
    DynamicEventVar<?> definedTotalPay = getDefineDynamicEventVars().get("ev_d_total_pay");
    DynamicEventVar<Long> totalPay = computeTotalPay(ctx, eventData, definedTotalPay);
```

```
putComputeResult(ctx, totalPay.getName(), totalPay);

DynamicEventVar<?> definedLocalCity = getDefineDynamicEventVars().get("ev_d_local_city");
DynamicEventVar<String> localCity = computeLocalCity(ctx, eventData, definedLocalCity);
putComputeResult(ctx, localCity.getName(), localCity);
}
```

## End-to-End Timeout Control

Unified timeout control is supported for the pipeline. The timing starts when a request enters the BLU. If the threshold is exceeded, the request is returned quickly.

## Internal Dotting Timeout Control for Plug-ins

The timeout control is based on the timing dotting code. It determines whether the plug-in calculation exceeds the threshold from the time when the **compute** method is executed to the time when the dotting code is executed. The timeout threshold can be initialized in the **init** method or customized for a single timing point.

- In the following DynamicVarsExtensions sample code, `checkTimeoutMS(ctx, timeoutMS, true)` is the specific format of the timing dotting code. The first parameter indicates that the current dotting threshold is the value of the **timeoutMS** attribute. The second parameter indicates the exception handling mode after the timeout. If the value is **true**, the request is interrupted and the subsequent calculation is not performed. If the value is **false**, the subsequent variable calculation logic is skipped. The variables that are not calculated are calculated based on the default values, and the remaining steps are calculated normally.

```
private DynamicEventVar<Long> computeTotalPay(PipelineContext ctx, EventData eventData,
DynamicEventVar totalPayVar) throws EventHandlerException
{
    // get batch vars from eventData to compute dynamicEventVar
    Long historyPay = (Long) eventData.getBatchVars().get("bv_history_pay");
    historyPay = Objects.isNull(historyPay) ? 0L : historyPay;

    // get real time vars from eventData to compute dynamicEventVar
    Long onlinePay = (Long) eventData.getRtqVars().get("rv_online_pay");
    onlinePay = Objects.isNull(onlinePay) ? 0L : onlinePay;

    Long totalPay = historyPay + onlinePay;
    if(totalPay != 0L) {
        totalPayVar.setValue(totalPay);
    } else {
        totalPayVar.setException("required input vars maybe not exists or offline.");
    }

    checkTimeoutMS(ctx, timeoutMS, true);

    return totalPayVar;
}
```

- To ensure that original packet data can be properly imported to the database, **EventFieldExtension** provides only the `checkTimeoutMS(ctx, timeoutMS)` API. The subsequent variable calculation logic is skipped, and the variables that are not calculated are calculated based on the default values in the subsequent steps, and the remaining steps are calculated normally.

## Runtime Exception

If a runtime exception is thrown during variable calculation, the current request is quickly returned and all subsequent calculation steps are skipped.

## Checked Exception

If a checked exception is thrown during variable calculation, the remaining calculation of dynamic extended variables is skipped. However, the subsequent calculation steps defined in the pipeline continue to be processed.

- If an expected exception occurs during the calculation and you want to output the information to the result, you can invoke **DynamicEventVar**. The **setException()** method is used to set exception information. The exception information is displayed in the **EventVarsException** field in the calculation result.
- If the value of a defined dynamic extended variable fails to be set during request calculation, the default value will be used in subsequent calculation and will be displayed in the **EventVarsException** field in the calculation result.
- The following is an example of the **RTDExecutor\_result.log** file.

```
2017-05-08 16:43:47.047 INFO [pipelineTaskConsumer-2] - RTDResult :  
{"seqNo":"1580206d9e0f4e0a82b43911c2203f1d","dsType":"plugins_1","ddUpdate":"","statusCode":2,"s  
tatusMessage":"","data":null,"batchData":  
[],"eventVars":null,"batchVars":null,"rtqVars":null,"modelResults":null,"filterRules":null,"procRules":null,  
"userdata":  
{"event_inbound_time":1494233027021,"total_time":20,"out_mq_total_time":0,"multi_proc_time":0,"Ev  
entFieldException":{},"plugin_dataprep_cost_time":0,"insert_time":11,"FilterRulesException":  
{},"EventVarsException":{"ev_d_hello":"compute fail for some reason, use default  
value."},"BatchVarsException":{},"RtqVarsException":{},"rtqVarExtraData":  
[],"plugin_dynamicvars_cost_time":1,"model_score_time":0,"ProcRulesException":  
{"exception_pr_rule2":"No message specified."}, "procRuleExtraData":  
[],"custom_user_data_key":"CustomUserDataValue","plugin_custom_cost_time":0,"HostIP":null,"Contai  
nerId":null,"PipelineId":944044635,"ThreadId":1261,"TenantId":"1","EventSourceId":"1493972947028","  
TraceFlag":0}, "rtdResults":  
{"score":92,"scoreResult":9,"syncModeStatusCode":0,"syncModeStatusMessage":"OK!"}, "batchId":""}
```

### 1.19.3.2 Decision Engine Custom Program

#### 1.19.3.2.1 Scenario

Users can customize decision engine configurations to control the output of scoring results of multiple rule engines, meeting more complex service requirements.

#### 1.19.3.2.2 Development Guidelines

- Check whether the current event source plug-in is an extended decision-making flow and whether the plug-in code contains a rating decision-making flow. For details, see [Event Source Custom Plug-in Program](#).
- Determine whether to make decisions on the stored procedure rules of the current event source.

### 1.19.3.2.3 Sample Code

#### RtdDroolsFileGeneratorTest

The following code snippets are in the **RtdDroolsFileGeneratorTest** class in the **com/huawei/fi/rtd/drools/generator** package in the **test** directory.

```
package com.huawei.fi.rtd.drools.generator;
import com.huawei.fi.rtd.drools.generator.common.AlgorithmType;
import com.huawei.fi.rtd.drools.generator.common.ZipFileUtil;
import org.apache.commons.lang3.StringUtils;
import org.junit.*;
import org.junit.rules.TemporaryFolder;
import org.kie.api.runtime.KieContainer;
import java.io.*;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
public class RtdDroolsFileGeneratorTest {

    private static final String RTD_DROOLS_K_MODULE_XML = "kmodule.xml";
    private static final String RTD_META_DATA_PROPERTIES = "rtd_meta_data.properties";
    private static final String RTD_DROOLS_DECISION_JSON = "drools_contract.json";
    private static final String RTD_DROOLS_FILE_PREFIX = "rtd_drools_decision_";
    private static final int BUFFER = 10 * 1024;
    private KieContainer kieContainer;
    private RtdDroolsDecision droolsDecision;
    @Rule
    public TemporaryFolder temporaryFolder = new TemporaryFolder();
    @Before
    public void setUp() {

        droolsDecision = new RtdDroolsDecision(AlgorithmType.AVERAGE.name());
        RtdDroolsElementsGroup weightGroupOne = new RtdDroolsElementsGroup("drools_group_01",
AlgorithmType.WEIGHTING.name());
        weightGroupOne.setSalience(99);
        weightGroupOne.addElement(new RtdDroolsElement("pr_rule_01", 0.5));
        weightGroupOne.addElement(new RtdDroolsElement("pr_rule_02", 0.3));
        RtdDroolsElementsGroup weightGroupTwo = new RtdDroolsElementsGroup("drools_group_02",
AlgorithmType.WEIGHTING.name());
        weightGroupTwo.setSalience(99);
        weightGroupTwo.addElement(new RtdDroolsElement("pr_rule_02"));
        weightGroupTwo.addElement(new RtdDroolsElement("pr_rule_03"));
        RtdDroolsElementsGroup averageGroup = new RtdDroolsElementsGroup("drools_group_03",
AlgorithmType.AVERAGE.name());
        averageGroup.setSalience(99);
        averageGroup.addElement(new RtdDroolsElement("pr_rule_02"));
        averageGroup.addElement(new RtdDroolsElement("pr_rule_03"));
        averageGroup.addElement(new RtdDroolsElement("pr_rule_04"));
        RtdDroolsElementsGroup maximumGroup = new RtdDroolsElementsGroup("drools_group_04",
AlgorithmType.MAXIMUM.name());
        maximumGroup.setSalience(99);
        maximumGroup.addElement(new RtdDroolsElement("pr_rule_01"));
        maximumGroup.addElement(new RtdDroolsElement("pr_rule_02"));
        maximumGroup.addElement(new RtdDroolsElement("pr_rule_03"));
        RtdDroolsElementsGroup minimumGroup = new RtdDroolsElementsGroup("drools_group_05",
AlgorithmType.MINIMUM.name());
        minimumGroup.setSalience(99);
        minimumGroup.addElement(new RtdDroolsElement("pr_rule_01"));
        minimumGroup.addElement(new RtdDroolsElement("pr_rule_02"));
        minimumGroup.addElement(new RtdDroolsElement("pr_rule_03"));
        droolsDecision.addElementsGroup(weightGroupOne);
        droolsDecision.addElementsGroup(weightGroupTwo);
        droolsDecision.addElementsGroup(averageGroup);
        droolsDecision.addElementsGroup(maximumGroup);
        droolsDecision.addElementsGroup(minimumGroup);
    }

    @After
}
```

```
public void tearDown() {  
    if (kieContainer != null) {  
        kieContainer.dispose();  
    }  
  
    kieContainer = null;  
}  
  
/**  
 * Run the following commands to generate a decision engine file on the local host:  
 */  
@Test  
public void input_invalid_contract_json_and_out_put_zip_file_then_compare_content_success() {  
  
    RtdDroolsFileGenerator generator = RtdDroolsFileGenerator.INSTANCE;  
    String jsonContract = JSONObject.toJSONString(droolsDecision);  
    File result = null;  
    try {  
        // Specify the directory to be downloaded.  
        File tempDirectory = new File("d:/");  
        result = generator.outputZipFile(jsonContract, tempDirectory.getPath(), "rtd-drools-example.zip");  
    } catch (Exception e) {  
        e.printStackTrace();  
        Assert.fail("Create zip file fail ....");  
    }  
  
    // ZipInputStream zipIn = null;  
    // try {  
    //     zipIn = new ZipInputStream(new BufferedInputStream(new FileInputStream(result)));  
    //     compareZipFileContentOneByOne(zipIn);  
    // } catch (Exception e) {  
    //     Assert.fail("Unzip file fail ....");  
    // } finally {  
    //     ZipFileUtil.closeIOResource(zipIn);  
    // }  
    //  
    // boolean deleteSuccess = result.delete();  
    // Assert.assertTrue(deleteSuccess);  
}  
  
@Test  
public void input_invalid_contract_json_and_out_put_zip_stream_then_compare_content_success() {  
  
    RtdDroolsFileGenerator generator = RtdDroolsFileGenerator.INSTANCE;  
    String jsonContract = JSONObject.toJSONString(droolsDecision);  
    ByteArrayOutputStream outputBaos = generator.outputZipStream(jsonContract);  
    ByteArrayInputStream bais = new ByteArrayInputStream(outputBaos.toByteArray());  
    ZipInputStream zipIn = null;  
    try {  
        zipIn = new ZipInputStream(new BufferedInputStream(bais));  
        compareZipFileContentOneByOne(zipIn);  
    } catch (Exception e) {  
        Assert.fail("Unzip file fail ....");  
    } finally {  
        ZipFileUtil.closeIOResource(zipIn);  
    }  
}  
  
private void compareZipFileContentOneByOne(ZipInputStream zipIn) throws IOException {  
    for (ZipEntry zipEntry; (zipEntry = zipIn.getNextEntry()) != null; ) {  
        ByteArrayOutputStream baos = new ByteArrayOutputStream();  
        byte data[] = new byte[BUFFER];  
        for (int count; (count = zipIn.read(data, 0, BUFFER)) != -1; ) {  
            baos.write(data, 0, count);  
        }  
    }  
}
```

```

        String fileContent = new String(baos.toByteArray());
        ZipFileUtil.closeIOResource(baos);
        if (zipEntry.getName().equals(RTD_DROOLS_K_MODULE_XML)) {
            Assert.assertTrue(fileContent.contains("rtdDecisionStateless"));
        } else if (zipEntry.getName().equals(RTD_META_DATA_PROPERTIES)) {
            Assert.assertTrue(fileContent.contains("drools_group_01"));
        } else if (zipEntry.getName().startsWith(RTD_DROOLS_FILE_PREFIX)) {
            Assert.assertTrue(fileContent.contains("package com.huawei.fi.rtd.drools.common"));
        } else if (zipEntry.getName().equals(RTD_DROOLS_DECISION_JSON)) {
            Assert.assertTrue(StringUtils.isNotBlank(fileContent));
        } else {
            Assert.fail("Invalid file name ....");
        }
    }
}

```

## Sample Description

- Obtain all rule names of the current event source and specify the algorithm to be decided in the `setup()` method. For example, obtain the minimum score of rules `pr_rule_01`, `pr_rule_02` and `pr_rule_03`.

```

RtdDroolsElementsGroup minimumGroup = new RtdDroolsElementsGroup("drools_group_05",
AlgorithmType.MINIMUM.name());
minimumGroup.setSalience(99);
minimumGroup.addElement(new RtdDroolsElement("pr_rule_01"));
minimumGroup.addElement(new RtdDroolsElement("pr_rule_02"));
minimumGroup.addElement(new RtdDroolsElement("pr_rule_03"));

```

- Run the `input_invalid_contract_json_and_out_put_zip_file_then_compare_content_success()` command to download the file to the local host after compiling the custom decision-making code. Specify the download directory and comment out the following code for deleting the compressed package when you download the file.

```

/**
 * Run the following commands to generate a decision engine file on the local host:
 */
@Test
public void input_invalid_contract_json_and_out_put_zip_file_then_compare_content_success() {

    RtdDroolsFileGenerator generator = RtdDroolsFileGenerator.INSTANCE;
    String jsonContract = JSONObject.toJSONString(droolsDecision);
    File result = null;
    try {
        // Specify the directory to be downloaded.
        File tempDirectory = new File("d:/");
        result = generator.outputZipFile(jsonContract, tempDirectory.getPath(), "rtd-drools-
example.zip");
    } catch (Exception e) {
        e.printStackTrace();
        Assert.fail("Create zip file fail ....");
    }

    // ZipInputStream zipIn = null;
    // try {
    //     zipIn = new ZipInputStream(new BufferedInputStream(new FileInputStream(result)));
    //     compareZipFileContentOneByOne(zipIn);
    // } catch (Exception e) {
    //     Assert.fail("Unzip file fail ....");
    // } finally {
    //     ZipFileUtil.closeIOResource(zipIn);
    // }
    // boolean deleteSuccess = result.delete();
}

```

```
//     Assert.assertTrue(deleteSuccess);
}
```

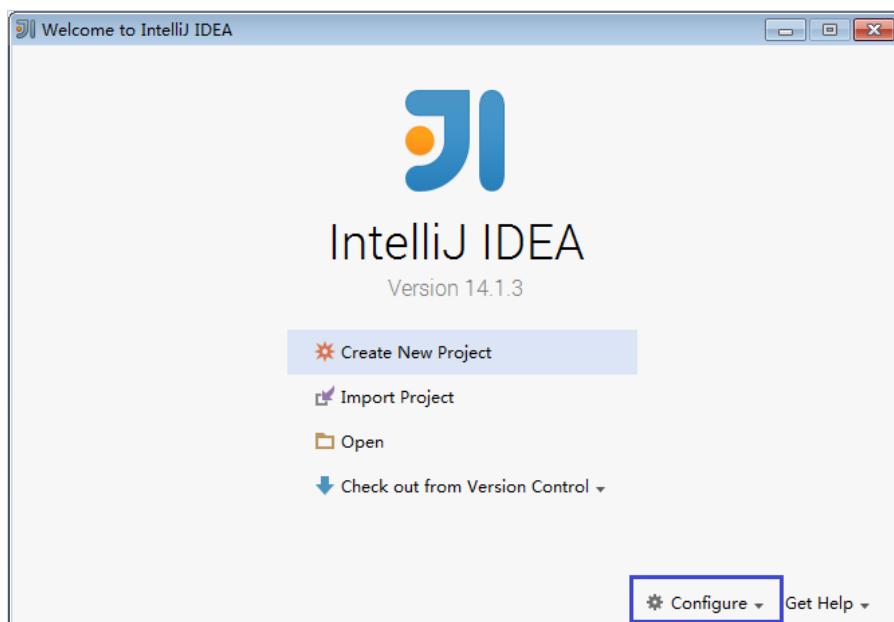
## 1.19.4 Commissioning the Application

### 1.19.4.1 Compiling and Packaging

**Step 1** Before importing the sample project, configure JDK for IntelliJ IDEA.

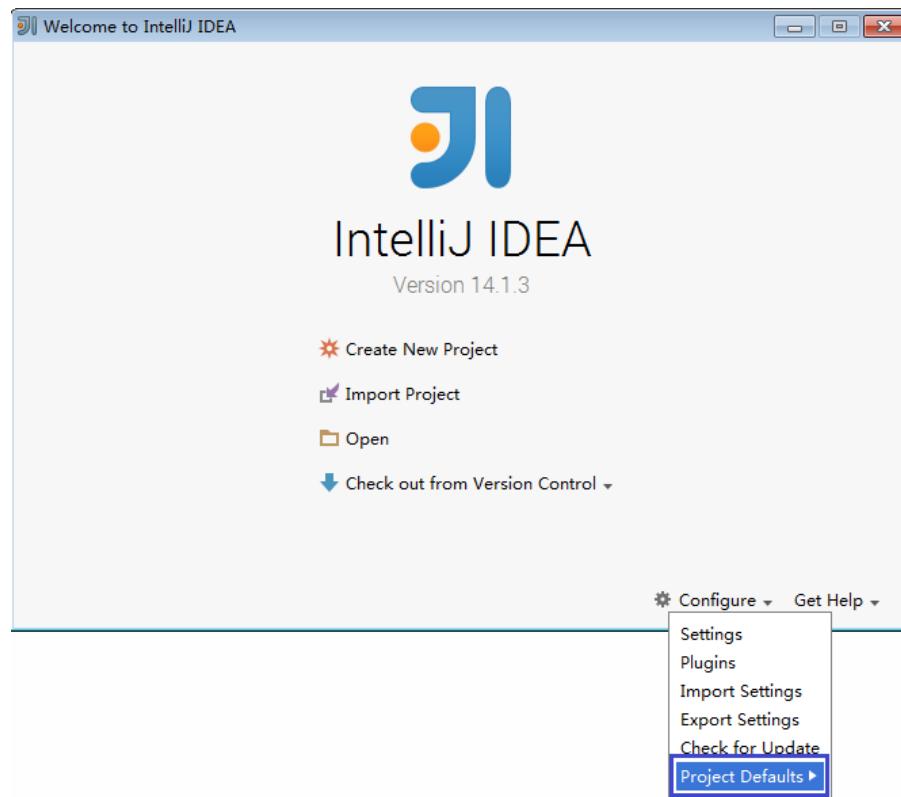
1. Start IntelliJ IDEA and click **Configure**.

**Figure 1-240** Choosing Configure



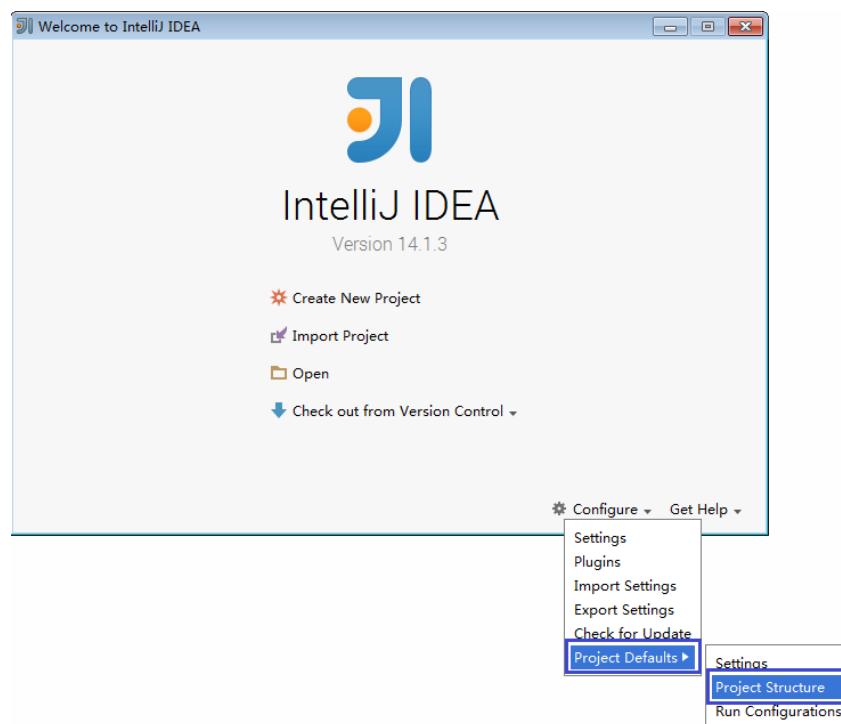
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 1-241 Choosing Project Defaults



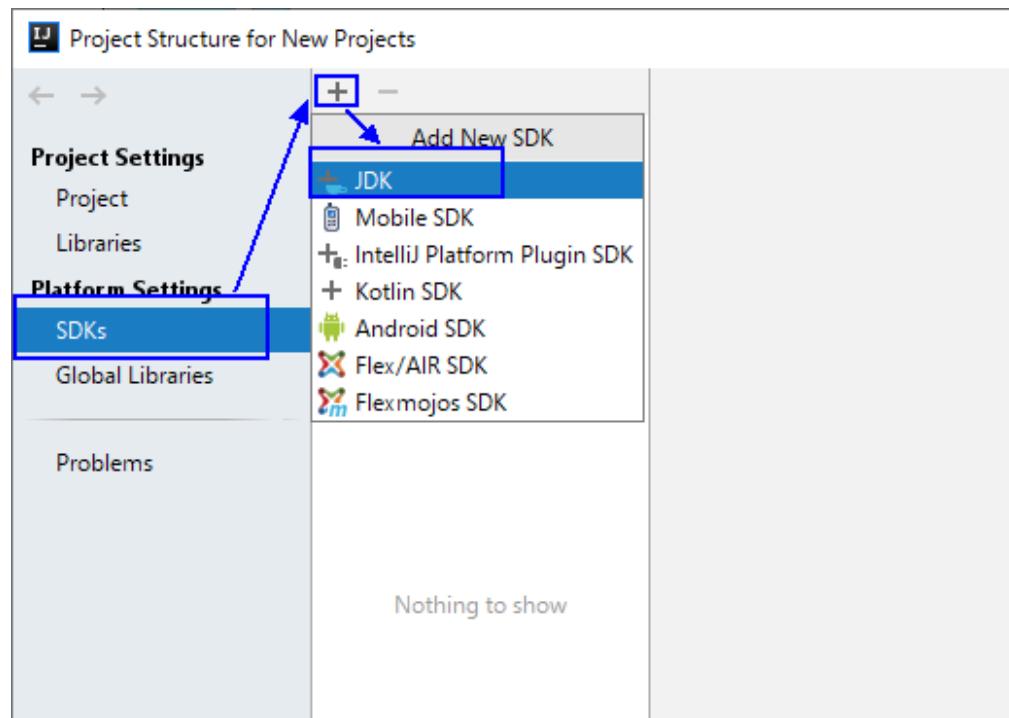
3. Select **Project Structure** from **Project Defaults**.

Figure 1-242 Project Defaults



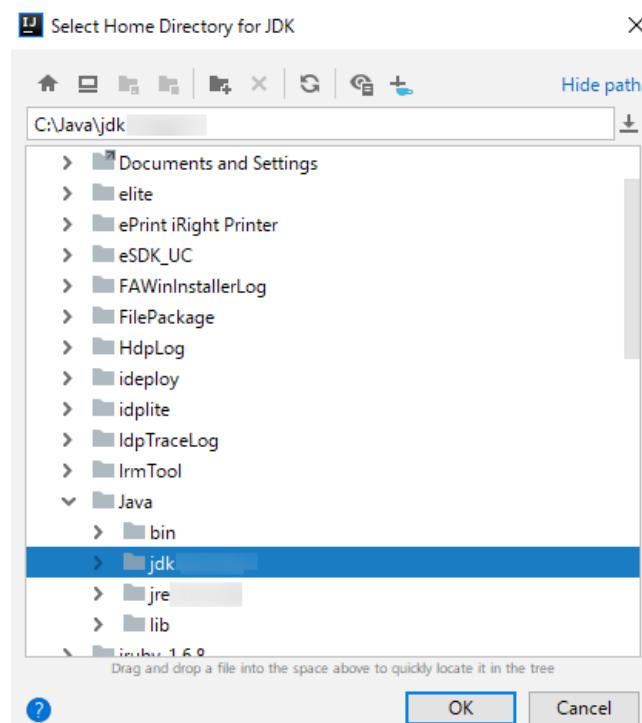
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 1-243 Adding the JDK



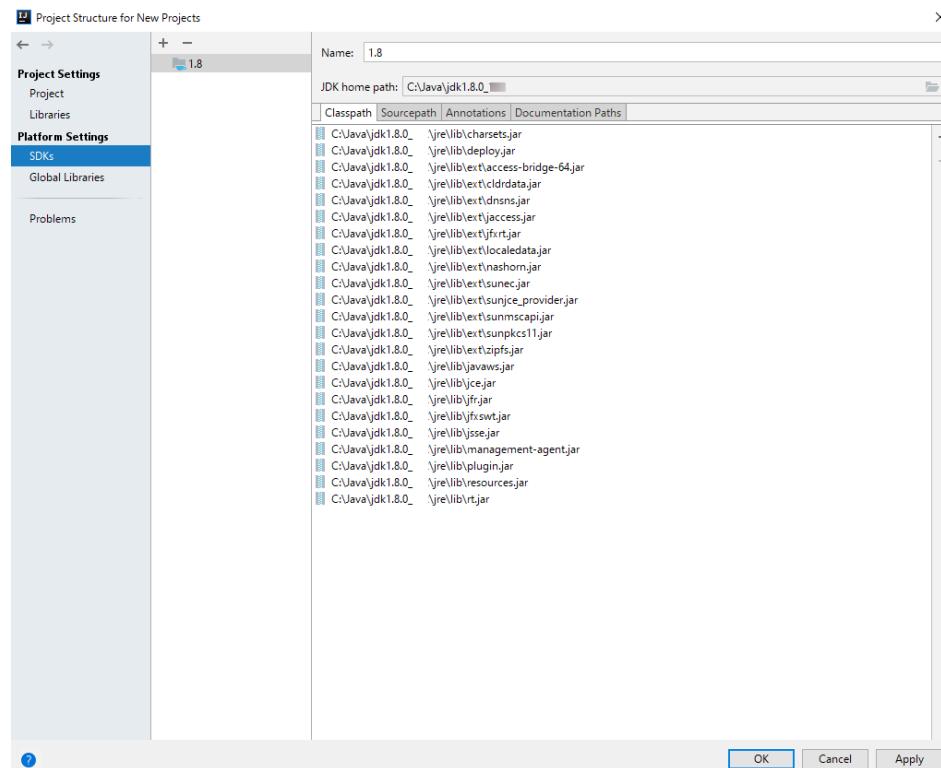
5. On the displayed **Select Home Directory for JDK** page, select the JDK directory and click **OK**.

Figure 1-244 Selecting the JDK directory



6. After selecting the JDK, click **OK** to complete the configuration.

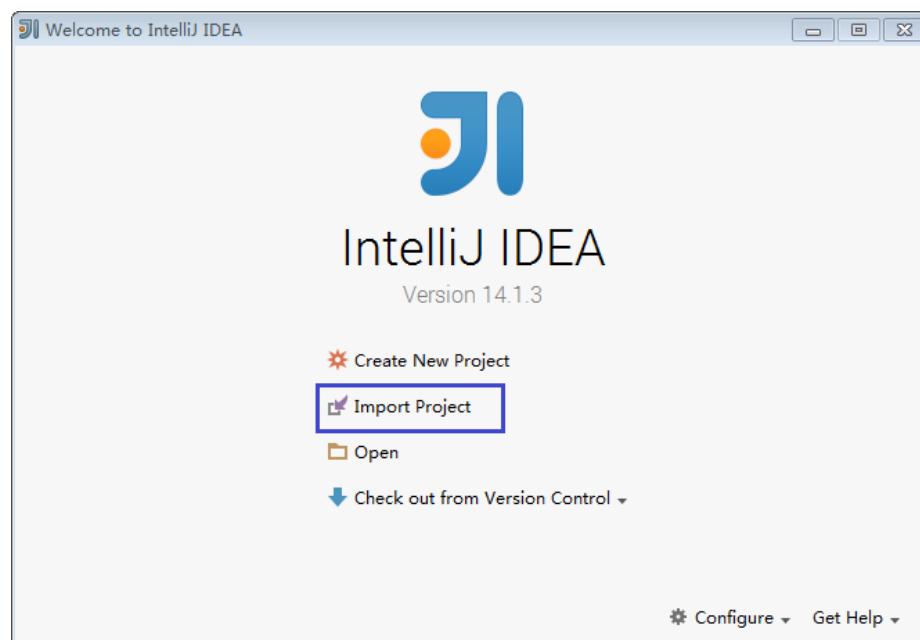
Figure 1-245 Completing the configuration



**Step 2** Import the Java sample projects to the IDEA.

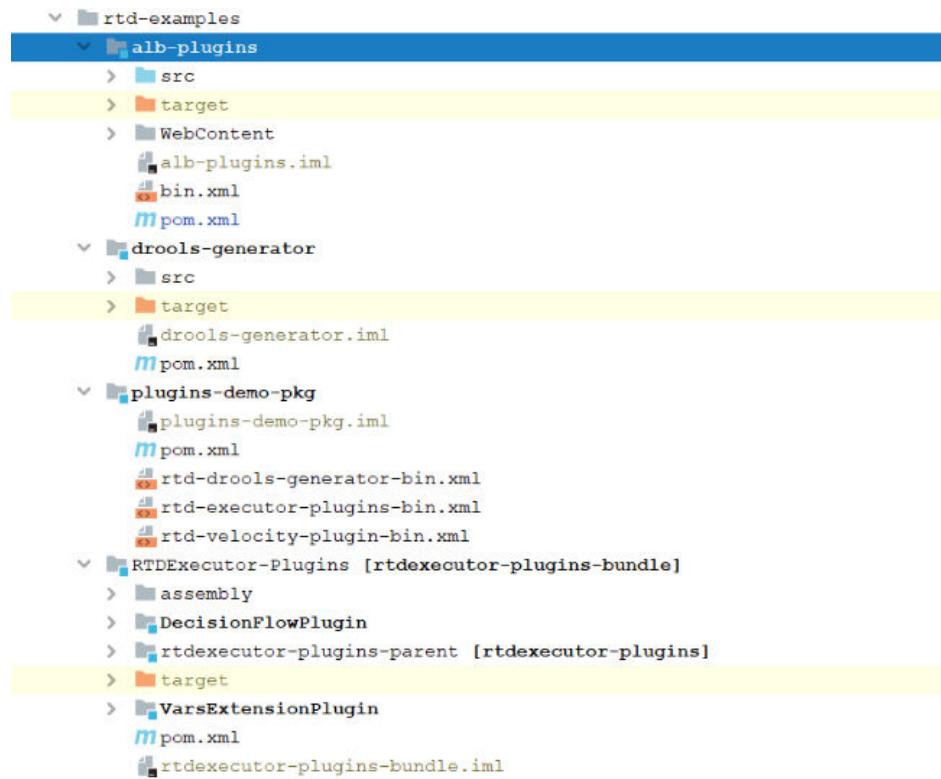
1. Start the IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.  
Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Select **File > Import project...** to import projects.

Figure 1-246 Importing a project (on the Quick Start page)



2. Select the directory for storing the imported projects and click **OK**.

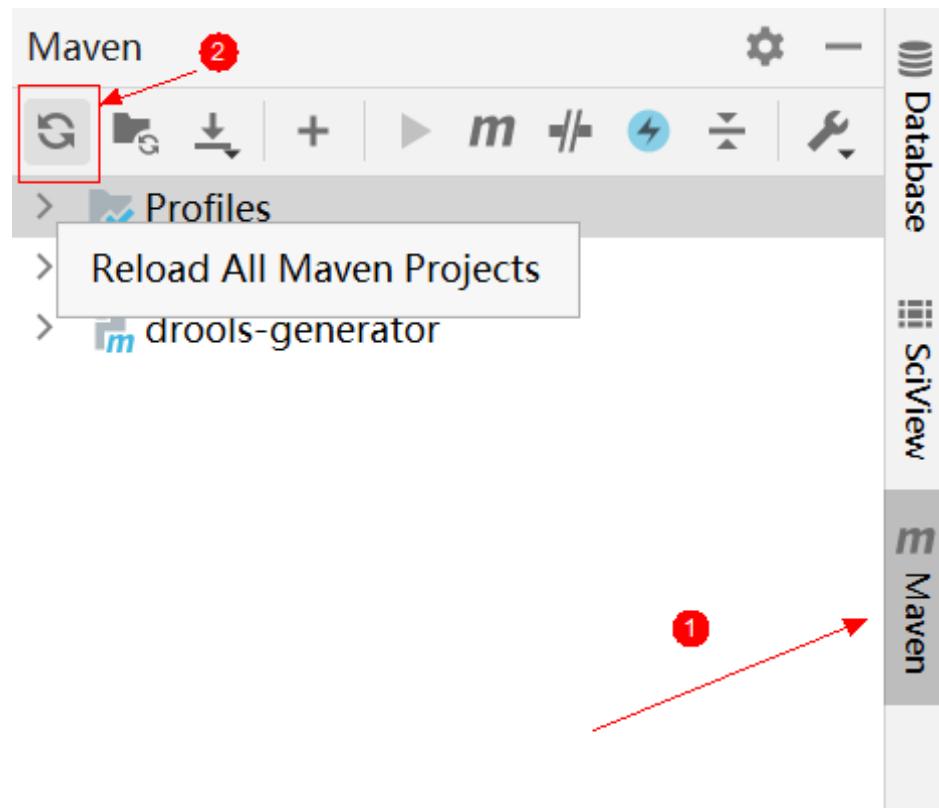
Figure 1-247 Select File or Directory to Import



3. Retain the default dependency library that is automatically recognized by IDEA and the suggested module structure, and click **Next**.

**Step 3** In IntelliJ IDEA, click **Reload All Maven Projects** in the Maven window on the right to import Maven project dependencies.

Figure 1-248 Reload projects

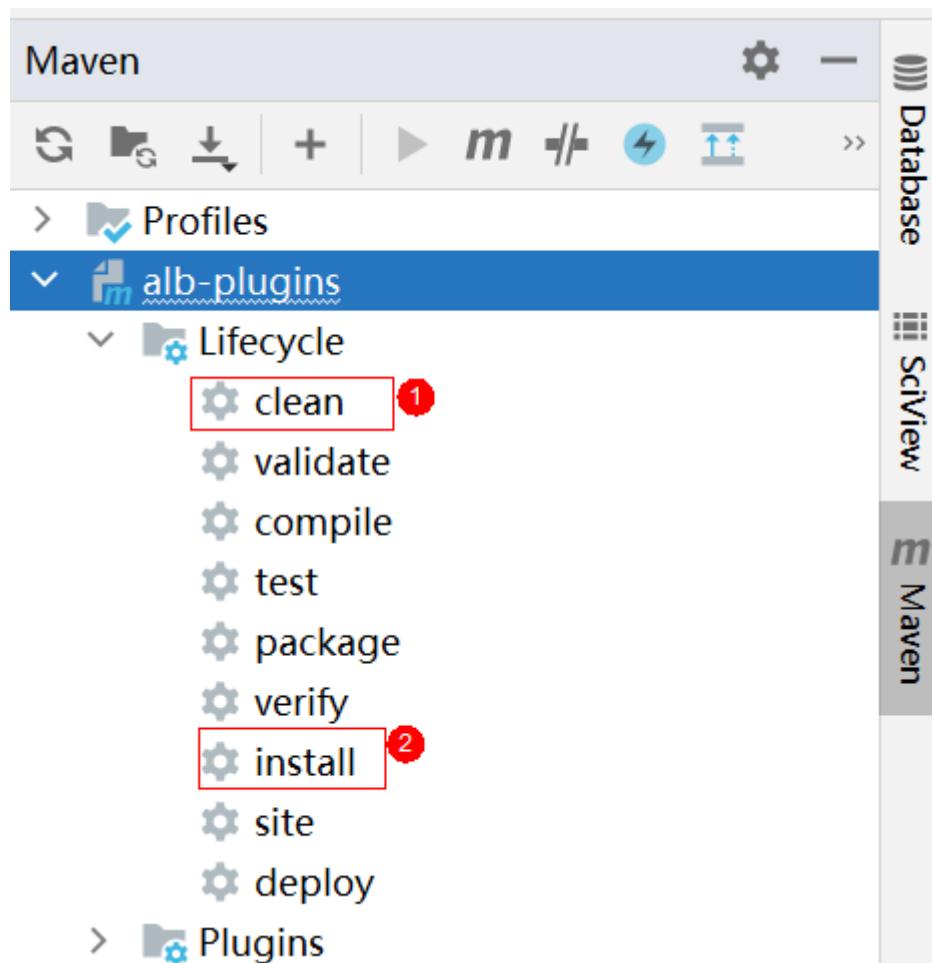


**Step 4** Compile the application.

There are two compilation methods:

- Method 1
  - a. Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.
  - b. Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 1-249 Maven clean and install commands



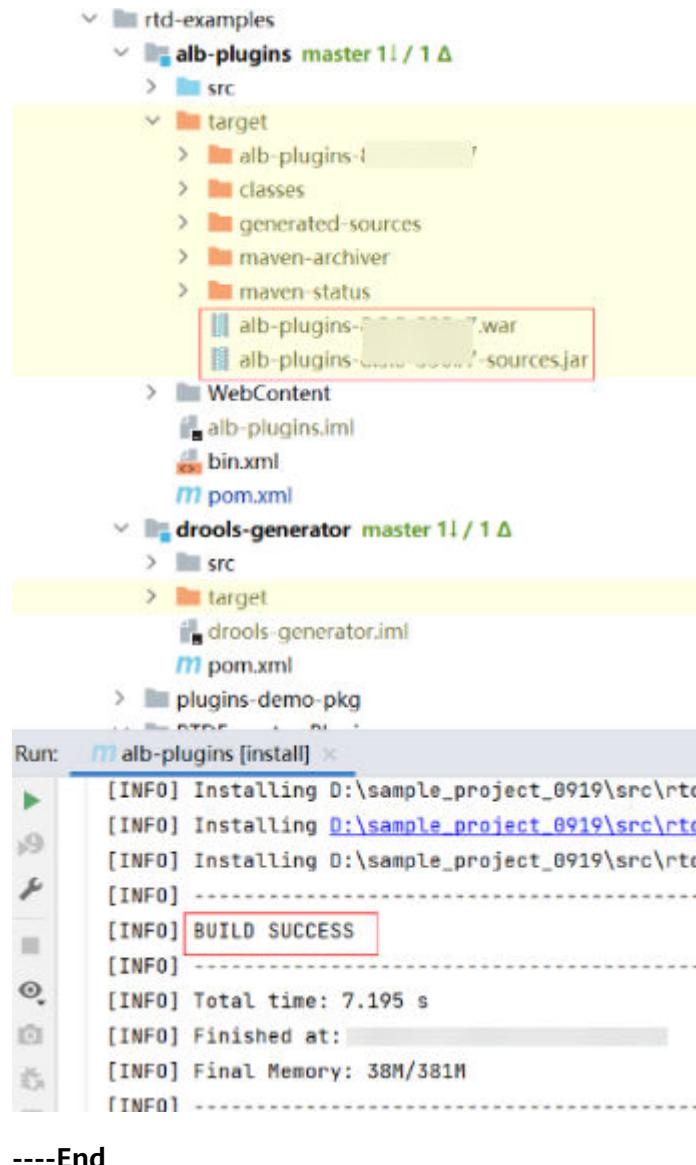
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 1-250 Entering "mvn clean install" to Idea Terminal

The screenshot shows the IntelliJ IDEA terminal window. The command 'mvn clean install' is typed into the terminal, with the 'install' part highlighted in a red box. The terminal tab is selected at the bottom of the interface.

**Step 5** Obtain the JAR or WAR package from the **target** directory which is generated after "BUILD SUCCESS" is displayed.

Figure 1-251 Compilation completed



----End

## 1.20 Solr Development Guide

### 1.20.1 Overview

#### 1.20.1.1 Application Development Overview

##### Solr Introduction

Solr is an independent enterprise search server based on Apache Lucene. It provides Representational State Transfer (REST)-similar Hypertext Transfer Protocol (HTTP)/Extensible Markup Language (XML) and JavaScript object notation (JSON) application programming interfaces (APIs). Main functions of Solr include full-text search, hit highlighting, faceted search, near real-time indexing,

dynamic clustering, database integration, rich document (for example, Word and PDF) handling, and geographic information search.

As an outstanding enterprise search server, Solr supports the following features:

- Advanced full-text search function
- Optimized large-capacity network traffic
- Standard open APIs, including XML, JSON, and HTTP
- Comprehensive HTML management interface
- Java management extensions (JMX) for monitoring servers
- Linear scalability, automatic index replication, and automatic failover and restoration
- Near real-time indexing
- XML configuration enabling flexibility and adaptability
- Extensible plug-in architecture

### 1.20.1.2 Common Concepts

Solr can be deployed in multiple modes, such as single-node mode, master-slave mode, and cloud mode. This document describes the SolrCloud deployment mode. SolrCloud is a distributed search solution in Solr based on Solr and ZooKeeper.

- **Collection**

Collection is a complete logical index in a SolrCloud cluster. A Collection can be divided into multiple Shards that use the same Config Set.

- **Config Set**

Config Set is a group of configuration files required by Solr Cores to provide services. A Config Set includes solrconfig.xml (SolrConfigXml) and schema.xml (SchemaXml).

- **Core**

Core refers to Solr Core. A Solr includes one or multiple Solr Cores. Each Solr Core independently provides indexing and query functions. Each Solr Core corresponds to an index or a Collection Shard. In SolrCloud, Solr configurations are stored in ZooKeeper.

- **Replica**

Replica is a copy of a Shard. Each Replica is in a Solr Core.

- **Shard**

Shard is a logical section of a Collection. Each Shard has one or multiple replicas, among which a leader is elected.

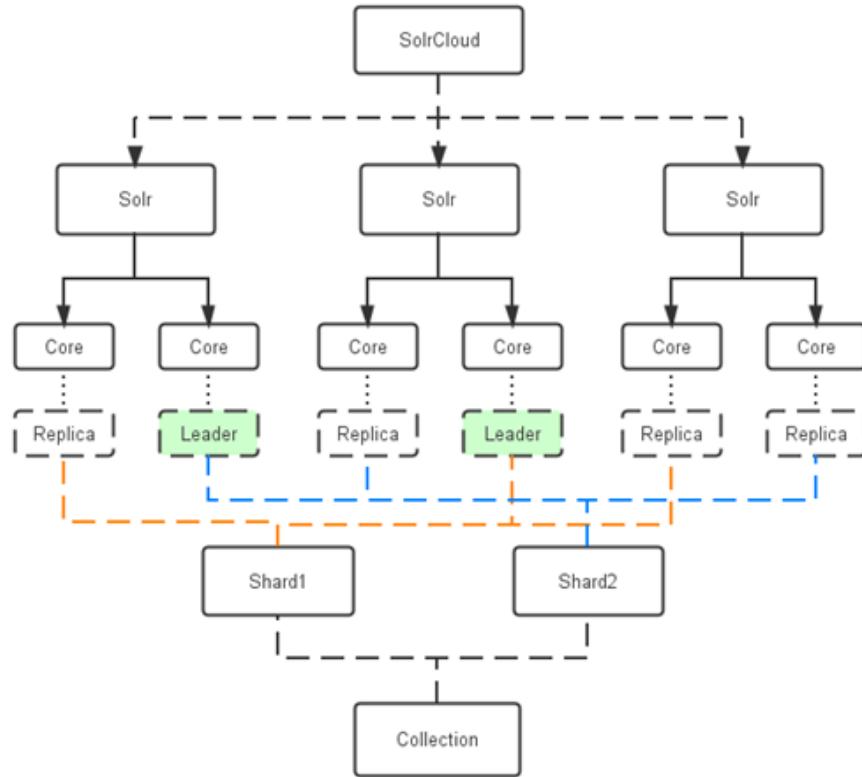
- **Leader**

Leader is a Shard replica elected from multiple replicas. When documents are indexed, SolrCloud transfers them to the leader, and the leader distributes them to replicas of all Shards.

- **Zookeeper**

ZooKeeper is mandatory in SolrCloud. It provides distributed lock and leader election functions.

Figure 1-252 Collection and Solr entities



### 1.20.1.3 Development Process

This document describes HBase application development based on the Java API.

For information about each phase in the development process, see [Figure 1-253](#) and [Table 1-170](#).

Figure 1-253 Solr application development process

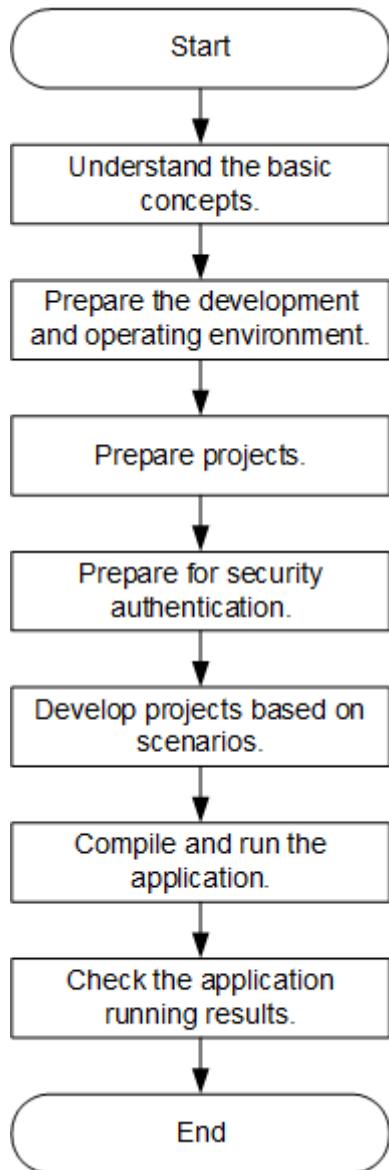


Table 1-170 Solr application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Solr and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of Solr applications. The IntelliJ IDEA tool can be used. The Solr running environment is the Solr client. Install and configure the client according to the guide.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare projects.	Import an example project according to the guide.	<a href="#">Configuring and Importing Sample Projects</a>
Preparing for security authentication.	The security authentication is mandatory if security clusters are used.	<a href="#">Preparing for Security Authentication</a>
Develop projects based on scenarios.	Provide an example project using Java language, covering an entire process for example project from query index to delete index.	<a href="#">Developing an Application</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
View application running results.	Application running results are written into a specified path.	<a href="#">Application Commissioning</a>

## 1.20.2 Environment Preparation

### 1.20.2.1 Preparing for Development and Operating Environment

#### Preparing Development Environment

[Table 1-171](#) describes the environment required for secondary development.

**Table 1-171** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>Development environment: Windows OS. Windows 7 or later is supported.</li><li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>

Item	Description
JUnit plug-in installation	Basic configuration for the development environment.
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
Developer account preparation	See <a href="#">Preparing a Developer Account</a> for configuration.
7-zip	Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#). In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

    - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Solr\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the Solr sample project.

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory.

      - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**
- When configuring IPv6 hosts on the local PC running Windows, add % and InterfaceIndex of the network interface card (NIC) to the end of the IPv6 address. The InterfaceIndex can be obtained by running the **ipconfig** command.

Example:

fec1:0:0:e505:8:99:5:1%10

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Solr\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Solr/config/* root@/IP address of the client node:/opt/hadoopclient/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 1.20.2.2 Configuring and Importing Sample Projects

### Background

Obtain the Solr development example project. Import the project to IntelliJ IDEA for learning.

### Prerequisites

Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the cluster time in the lower-right corner on FusionInsight Manager.

### Procedure

- Step 1** Obtain the sample project folder **solr-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the **user.keytab** and **krb5.conf** files obtained in [Preparing a Developer Account](#) section to the **conf** directory of the sample project, and change the **principal** value in the **solr-example.properties** to the **principal=developuser**.
- Step 3** Change the **zkHost** value in the **solr-example.properties** to the **-DzkHost** value queried on the Dashboard on the **Solr Admin** page.  
Log in to FusionInsight Manager as a user who has the Solr page access permission, choose **Cluster > Services > Solr**, and click the link next to **Solr WebUI**. The Solr Admin page is displayed.
- Step 4** Change the **zkSslEnable** value in the **solr-example.properties** to the value of **ssl.enabled** of ZooKeeper.  
Log in to FusionInsight Manager, choose **Cluster > Services > ZooKeeper > Configurations > All Configurations**, search for and record the value of **ssl.enabled** of the ZooKeeper service.
- Step 5** Change the **createNodeSet** value in the **solr-example.properties** to the IP address and port number of the Solr instance. Use commas (,) to separate multiple values.  
Log in to FusionInsight Manager, choose **Cluster > Services > Solr > Instance**, and view the service IP address of the SolrServer instance. The default port number is **21101**.

The configuration example of the **solr-example.properties** file is as follows:

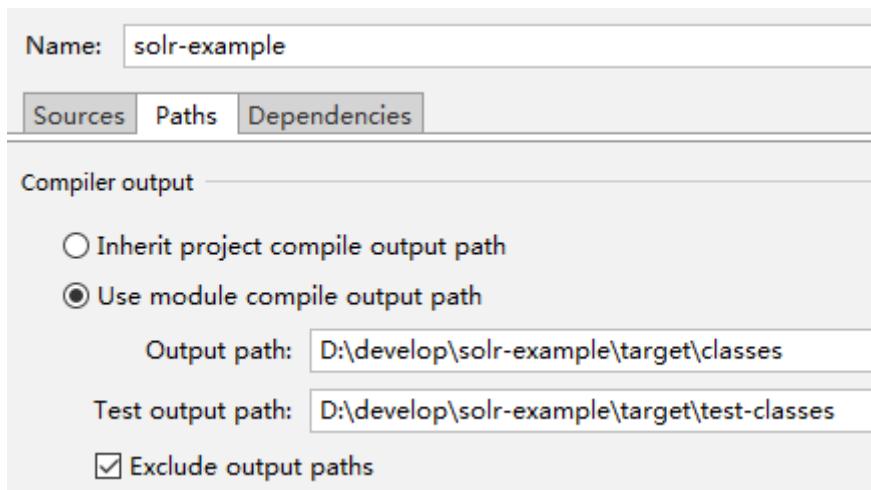
```
zkHost=192.168.1.189:24002,192.168.1.228:24002,192.168.1.150:24002/solr
zkSslEnable=false
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL=zookeeper/hadoop.hadoop.com
principal=developuser
zkClientTimeout=20000
zkConnectTimeout=30000
SOLR_KBS_ENABLED=true
COLLECTION_NAME=col_test
DEFAULT_CONFIG_NAME=confWithSchema
shardNum=3
replicaNum=1
```

```
maxShardsPerNode=1  
autoAddReplicas=false  
assignToSpecifiedNodeSet=false  
createNodeSet=192.168.1.136:21101_solr,192.168.1.228:21101_solr  
isNeedZkClientConfig=true
```

**Step 6** Import the example project to the IntelliJ IDEA development environment.

1. Choose **File > Open... > Open File or Project**.
2. Enter the path of the solr-example sample project.
3. Select the solr-example example project folder, and click **OK**.
4. Before running the **solr-example** sample project, configure **Compiler output** for the project. Otherwise, the error message "the output path is not specified" may be displayed.

Specifically, choose **File > Project Structure > Modules** and configure the compilation output path of the **solr-example** sample project, as shown in the following figure.



**NOTE**

You can customize **Compiler output**. Ensure that the path exists. To change the output path during the project running, create the required path, and then configure the path by following the instruction in the preceding figure. The modification takes effect after the compilation.

----End

### 1.20.2.3 Preparing for Security Authentication

#### Scenario

In a secure cluster environment, components must perform mutual authentication before communicating with each other to ensure communication security.

Solr application development requires ZooKeeper and Kerberos security authentication. The **jaas.conf** file is used for ZooKeeper authentication. Spnego authentication is a security protocol expending Kerberos and using GSS-API authentication mechanism.

Security authentication uses the code authentication mode. Only **jaas.conf** and **krb5.conf** files need to be configured. This example project applies to the Oracle

Java platform and the IBM Java platform. **jaas.conf** is generated and configured using the code, such as **LoginUtil.setJaasFile(principal, path + "user.keytab")** in the following initial configuration example.

- Code authentication

```
if (testSample.SOLR_KBS_ENABLED.equals("true")) {  
    testSample.setSecConfig();  
}
```

- Initial configuration

```
private void setSecConfig() throws SolrException {  
    String path = System.getProperty("user.dir") +  
        File.separator + "conf" + File.separator;  
    path = path.replace("\\", "\\\\"");  
    try {  
        LoginUtil.setJaasFile(principal, path + "user.keytab");  
        LoginUtil.setKrb5Config(path + "krb5.conf");  
        LoginUtil.setZookeeperServerPrincipal(  
            ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);  
  
    } catch (IOException e) {  
        LOG.error("Failed to set security conf", e);  
        throw new SolrException("Failed to set security conf");  
    }  
}
```

## 1.20.3 Developing an Application

### 1.20.3.1 Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the Solr development process and know important interface functions.

#### Scenario

Provided that a user needs an application to manage employees and search employee personal information. Solr can provide the searching service. The search process is as follows:

- Check whether the index to be created exists. If the index exists, delete it.
- Create an index for personal information data of employees.
- Insert the personal information data of employees into the index.
- Query personal information of employees based on the search criteria.
- Delete personal information of employees based on the search criteria.

#### NOTE

The schema information about the column corresponding to the data table of employees needs to be planned in Solr index configurations.

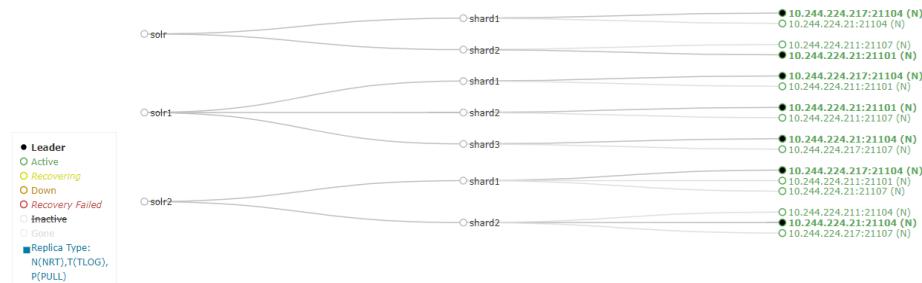
### 1.20.3.2 Development Idea

The core function of Solr is search. You need to create a Collection before searching.

This topic uses the SolrCloud deployment mode as an example. In the following figure, two collections are created, in which three instances of collection 1 are

distributed over three nodes and each shard in collection 2 has two replicas, as shown in [Figure 1-254](#).

**Figure 1-254** Collection relationships



Operations involved in example codes include establishing a connection with the Solr server, querying a Collection, deleting a Collection, creating a Collection, adding one document, adding multiple documents, simple search, and deleting a document. You can learn advanced Solr operations based on basic example codes.

Example codes are described in the following sequence:

1. Establish a connection between the client and the CloudSolr server.
2. Querying Collation.
3. Deleting a Collection.
4. Creating a Collection.
5. Add one or multiple documents.
6. Query documents.
7. Delete documents.
8. Release the CloudSolr connection.

**NOTE**

In this development guide, the used Solr configuration files are all built-in. If users want to customize the configuration set, see [Component Operation Guide > Solr > Common Service Operations > Shell Client Operation Commands](#) to create and upload the configuration set.

### 1.20.3.3 Example Code Description

#### 1.20.3.3.1 Initializing Solr

##### Function

Solr initialization is a prerequisite for using application programming interfaces (APIs) provided by Solr. Solr initialization aims to connect to SolrCloud.

**NOTE**

Call `cloudSolrClient.close ()` to close requested resources after the Solr operation is complete.

## Example Code

The following code snippet belongs to the **getCloudSolrClient** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
/*
 *Initialize CloudSolrClient instance, and collect SolrCloud
 */
private CloudSolrClient getCloudSolrClient(String zkHost) throws SolrException {
    Builder builder = new CloudSolrClient.Builder();
    builder.withZkHost(zkHost);
    CloudSolrClient cloudSolrClient = builder.build();
    cloudSolrClient.setZkClientTimeout(zkClientTimeout);
    cloudSolrClient.setZkConnectTimeout(zkConnectTimeout);
    cloudSolrClient.connect();
    LOG.info("The cloud Server has been connected !!!!");
    ZkStateReader zkStateReader = cloudSolrClient.getZkStateReader();
    ClusterState cloudState = zkStateReader.getClusterState();
    LOG.info("The zookeeper state is : {}", cloudState);
    return cloudSolrClient;
}
```

## Precautions

The three values **zkHost**, **zkClientTimeout**, and **zkConnectTimeout** can be configured in the **solr-example.properties** file under the **conf** directory. Change the **zkHost** value in the **solr-example.properties** to the **-DzkHost** value queried on the Dashboard of the Solr Admin page.

### 1.20.3.3.2 Querying a Collection

#### Function

By calling **process (cloudSolrClient)** in **CollectionAdminRequest.List** and the returned responses, users can obtain the names of all collections.

## Example Code

The following code snippet belongs to the **queryAllCollections** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private List<String> queryAllCollections(CloudSolrClient
    cloudSolrClient) throws SolrException {
    CollectionAdminRequest.List list = new CollectionAdminRequest.List();

    CollectionAdminResponse listRes = null;
    try {
        listRes = list.process(cloudSolrClient);
    } catch (SolrServerException | IOException e) {
        LOG.error("Failed to list collection", e);
        throw new SolrException("Failed to list collection");
    } catch (Exception e) {
        LOG.error("Failed to list collection", e);
        throw new SolrException("unknown exception");
    }

    List<String> collectionNames = (List<String>)
    listRes.getResponse().get("collections");
    LOG.info("All existed collections : {}", collectionNames);
    return collectionNames;
}
```

### 1.20.3.3.3 Deleting a Collection

#### Function

By calling **process (cloudSolrClient)** in **CollectionAdminRequest.Deleted** and the returned responses, users can check whether the deleting collection operation is successfully executed.

#### Example Code

The following code snippet belongs to the **deleteCollection** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void deleteCollection(CloudSolrClient cloudSolrClient)
    throws SolrException {
    CollectionAdminRequest.Delete delete =
        new CollectionAdminRequest.Delete();
    delete.setCollectionName(collectionName);
    CollectionAdminResponse response = null;
    try {
        response = delete.process(cloudSolrClient);
    } catch (SolrServerException | IOException e) {
        LOG.error("Failed to delete collection", e);
        throw new SolrException("Failed to create collection");
    } catch (Exception e) {
        LOG.error("Failed to delete collection", e);
        throw new SolrException("unknown exception");
    }
    if (response.isSuccess()) {
        LOG.info("Success to delete collection[{}]", collectionName);
    } else {
        LOG.error("Failed to delete collection[{}], cause : {}", collectionName, response.getErrorMessages());
        throw new SolrException("Failed to delete collection");
    }
}
```

### 1.20.3.3.4 Creating a Collection

#### Function

By calling process **cloudSolrClient** in the **CollectionAdminRequest.Create** and **responses**, users can check whether the creating collection operation is successfully executed.

#### Example Code

The following code snippet belongs to the **createCollection** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void createCollection(CloudSolrClient cloudSolrClient) throws SolrException {
    CollectionAdminRequest.Create create = CollectionAdminRequest.createCollection(collectionName,
        defaultConfigName, shardNum, replicaNum);
    CollectionAdminResponse response = null;
    try {
        response = create.process(cloudSolrClient);
    } catch (SolrServerException e) {
        LOG.error("Failed to create collection", e);
        throw new SolrException("Failed to create collection");
    } catch (IOException e) {
        LOG.error("Failed to create collection", e);
        throw new SolrException("Failed to create collection");
    }
```

```
        } catch (Exception e) {
            LOG.error("Failed to create collection", e);
            throw new SolrException("unknown exception");
        }
        if (response.isSuccess()) {
            LOG.info("Success to create collection[{}]", collectionName);
        } else {
            LOG.error("Failed to create collection[{}], cause : {}", collectionName, response.getErrorMessages());
            throw new SolrException("Failed to create collection");
        }
    }
```

## Precautions

1. **collectionName** is the name of the index to be created. **defaultConfigName** is the configuration set used by the index. The configuration set is the default Solr confWithSchema. Users can run the client script to download the configuration client information or view the information on the web page of the Solr Admin. **shardNum** indicates the number of fragments. **replicaNum** indicates the number of copies. All of these items can be configured in the **solr-example.properties** file under the **conf** directory and the configuration sets can be changed based on practical application.
2. It is suggested to limit the alias name and collection name **collectionName** at 255 characters while creating a collection, otherwise it might affect the performance of ZooKeeper and Solr. For more information please refer to Solr service configuration **SOLR\_COLLECTION\_CORE\_MAX\_LENGTH**.

### 1.20.3.3.5 Adding a Document

## Function

The index data is added by calling the adding method of cloudSolrClient or by constructing UpdateRequest to call request method of cloudSolrClient.

## Example Code 1

The following code snippet belongs to the **addDocs** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void addDocs(CloudSolrClient cloudSolrClient) throws SolrException {
    Collection<SolrInputDocument> documents = new ArrayList<SolrInputDocument>();
    for (Integer i = 0; i < 5; i++) {
        SolrInputDocument doc = new SolrInputDocument();
        doc.addField("id", i.toString());
        doc.addField("name", "Luna_" + i);
        doc.addField("features", "test" + i);
        doc.addField("price", (float) i * 1.01);
        documents.add(doc);
    }
    try {
        cloudSolrClient.add(documents);
        LOG.info("success to add index");
    } catch (SolrServerException e) {
        LOG.error("Failed to add document to collection", e);
        throw new SolrException("Failed to add document to collection");
    } catch (IOException e) {
        LOG.error("Failed to add document to collection", e);
        throw new SolrException("Failed to add document to collection");
    } catch (Exception e) {
        LOG.error("Failed to add document to collection", e);
        throw new SolrException("unknown exception");
    }
}
```

```
}
```

## Example Code 2

The following code snippet belongs to the **addDocs2** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void addDocs2(CloudSolrClient cloudSolrClient) throws  
SolrException{  
    UpdateRequest request = new UpdateRequest();  
    Collection<SolrInputDocument> documents = new ArrayList<>();  
    for (Integer i = 5; i < 10; i++) {  
        SolrInputDocument doc = new SolrInputDocument();  
        doc.addField("id", i.toString());  
        doc.addField("name", "Tom" + i);  
        doc.addField("features", "test" + i);  
        doc.addField("price", (float) i * 1.01);  
        documents.add(doc);  
    }  
    request.add(documents);  
    try {  
        cloudSolrClient.request(request);  
        cloudSolrClient.commit();  
    } catch (SolrServerException | IOException e) {  
        LOG.error("Failed to add document to collection", e);  
        throw new SolrException("Failed to add document to  
collection");  
    }  
}
```

### 1.20.3.3.6 Querying a Document

#### Function

The index data can be queried by constructing a **SolrQuery** instance and calling the **cloudSolrClient.query** API.

## Example Code

The following code snippet belongs to the **queryIndex** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void queryIndex(CloudSolrClient cloudSolrClient) throws SolrException {  
    SolrQuery query = new SolrQuery();  
    query.setQuery("name:Luna*");  
  
    try {  
        QueryResponse response = cloudSolrClient.query(query);  
        SolrDocumentList docs = response.getResults();  
        LOG.info("Query wasted time : {}ms", response.getQTime());  
  
        LOG.info("Total doc num : {}", docs.getNumFound());  
        for (SolrDocument doc : docs) {  
            LOG.info("doc detail : " + doc.getFieldValueMap());  
        }  
    } catch (SolrServerException e) {  
        LOG.error("Failed to query document", e);  
        throw new SolrException("Failed to query document");  
    } catch (IOException e) {  
        LOG.error("Failed to query document", e);  
        throw new SolrException("Failed to query document");  
    } catch (Exception e) {  
        LOG.error("Failed to query document", e);  
        throw new SolrException("unknown exception");  
    }  
}
```

```
}
```

### 1.20.3.3.7 Deleting a Document

#### Function

The data of a specified index can be deleted by calling **cloudSolrClient.deleteByQuery**.

#### Example Code

The following code snippet belongs to the **removeIndex** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void removeIndex(CloudSolrClient cloudSolrClient) throws  
SolrException {  
    try {  
        cloudSolrClient.deleteByQuery("*:*");  
        cloudSolrClient.commit();  
        LOG.info("Success to delete index");  
    } catch (SolrServerException | IOException e){  
        LOG.error("Failed to remove document", e);  
        throw new SolrException("Failed to remove document");  
    }  
}
```

## 1.20.4 Application Commissioning

### 1.20.4.1 Commissioning an Application in Windows

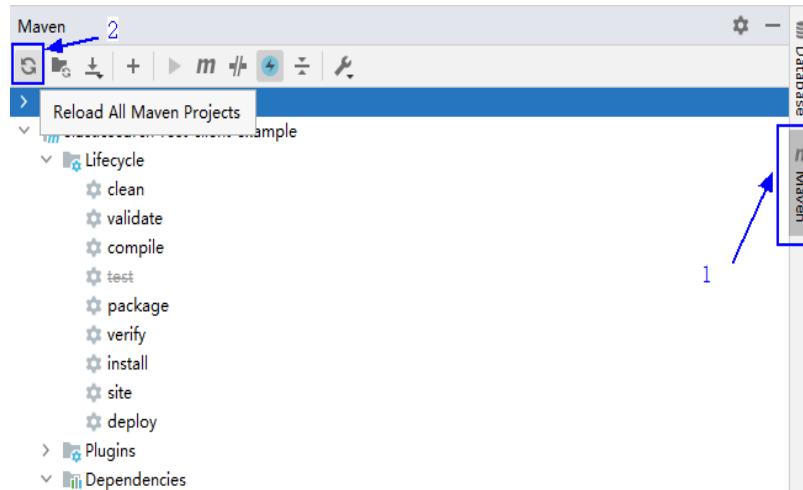
#### 1.20.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

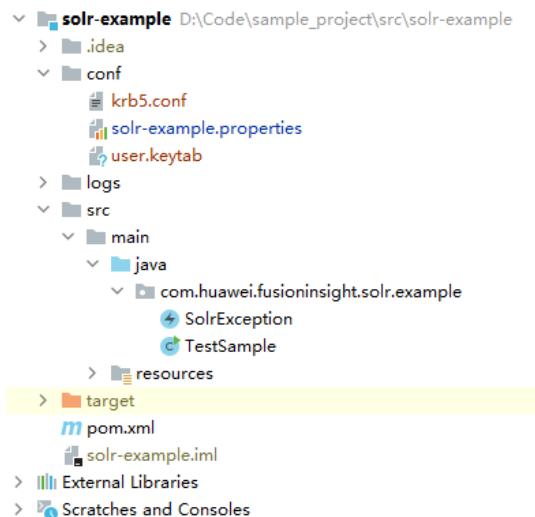
##### Procedure

- Step 1 Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.



## Step 2 Compile and run an application.

1. The following figure shows the file list after the configuration file has been placed and the code has been modified to match the login user.



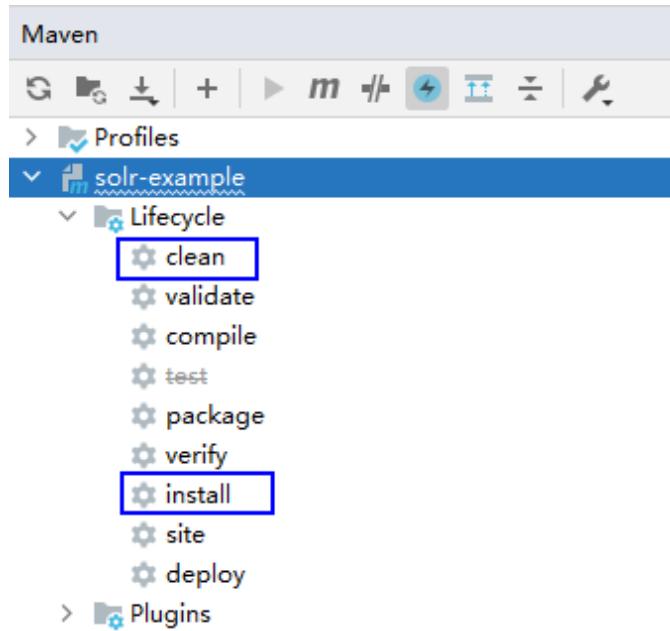
2. Compile an application.

There are two compilation modes.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.



- Method 2:

Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

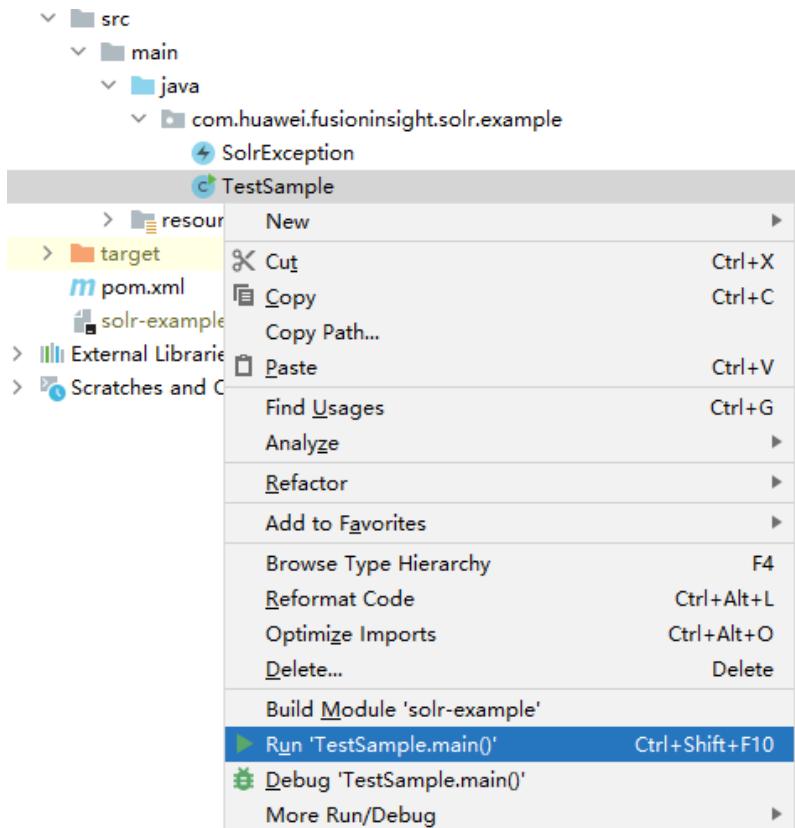
```
Terminal: Local +  
Microsoft Windows  
(c) Microsoft Corporation.  
  
D:\code\sample_project>cd src\solr-example  
  
D:\code\sample_project\src\solr-example>mvn clean compile
```

- After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

```
[INFO] Copying 1 resource  
[INFO]  
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ solr-example ---  
[INFO] Changes detected - recompiling the module!  
[INFO] Compiling 2 source files to D:\code\sample_project\src\solr-example\target\c  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.621 s  
[INFO] Finished at: 2023-09-19T17:18:06+08:00  
[INFO] -----
```

- Run the program.

Right-click the **TestSample** file and choose **Run 'TestSample.main()'** from the shortcut menu to run samples.



----End

#### 1.20.4.1.2 Viewing Commissioning Results

##### Scenario

After an Solr application is run, you can use one of the following methods to view the running result:

- Viewing the IntelliJ IDEA running result
- Viewing Solr logs
- Logging in to the Solr WebUI. For details, see [Component Operation Guide > Solr > Common Service Operations > Operations on Solr Admin UI](#).

##### Procedure

The following information is displayed in the running results:

```
...
2021-07-05 14:27:26,604 | INFO | main | Client is connected to ZooKeeper |
org.apache.solr.common.cloud.ConnectionManager.waitForConnected(ConnectionManager.java:251)
2021-07-05 14:27:26,856 | INFO | main | Updated live nodes from ZooKeeper... (0) -> (3) |
org.apache.solr.common.cloud.ZkStateReader.refreshLiveNodes(ZkStateReader.java:859)
2021-07-05 14:27:27,000 | INFO | main | Cluster at
192.168.1.189:24002,192.168.1.228:24002,192.168.1.150:24002/solr ready |
org.apache.solr.client.solrj.impl.ZkClientClusterStateProvider.getZkStateReader(ZkClientClusterStateProvider.java:177)
2021-07-05 14:27:27,001 | INFO | main | The cloud Server has been connected !!!! |
com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:188)
2021-07-05 14:27:27,001 | INFO | main | The zookeeper state is : znodeVersion: 0
live nodes:[192.168.1.136:21104_solr, 192.168.1.177:21101_solr, 192.168.1.187:21101_solr]
```

```
collections:{} | com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:192)
...
principal is solrtest@HADOOP.COM
Will use keytab
Commit Succeeded
2021-07-05 14:27:27,688 | INFO | main | All existed collections : [] |
com.huawei.fusioninsight.solr.example.TestSample.queryAllCollections(TestSample.java:316)
2021-07-05 14:27:33,482 | INFO | main | Success to create collection[col_test]. |
com.huawei.fusioninsight.solr.example.TestSample.createCollection(TestSample.java:278)
2021-07-05 14:27:33,870 | INFO | main | success to add index |
com.huawei.fusioninsight.solr.example.TestSample.addDocs(TestSample.java:229)
2021-07-05 14:27:36,467 | INFO | main | Query wasted time : 459ms |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:204)
2021-07-05 14:27:36,467 | INFO | main | Total doc num : 5 |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:205)
2021-07-05 14:27:36,467 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=4.0,
price_c=4,XXX, _version_=1704424914868502528} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:36,468 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=6.0,
price_c=6,XXX, _version_=1704424914871648256} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:36,468 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=8.0,
price_c=8,XXX, _version_=1704424914872696832} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:36,478 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0,
price_c=0,XXX, _version_=1704424914868502528} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:36,478 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=2.0,
price_c=2,XXX, _version_=1704424914881085440} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:37,090 | INFO | main | Success to delete index |
com.huawei.fusioninsight.solr.example.TestSample.removeIndex(TestSample.java:218)
2021-07-05 14:27:37,162 | INFO | main | Query wasted time : 42ms |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:204)
2021-07-05 14:27:37,162 | INFO | main | Total doc num : 5 |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:205)
2021-07-05 14:27:37,162 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=4.0,
price_c=4,XXX, _version_=1704424914868502528} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:37,162 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=6.0,
price_c=6,XXX, _version_=1704424914871648256} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:37,163 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=8.0,
price_c=8,XXX, _version_=1704424914872696832} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:37,163 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0,
price_c=0,XXX, _version_=1704424914868502528} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:37,163 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=2.0,
price_c=2,XXX, _version_=1704424914881085440} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:27:37,293 | INFO | main | Connection: 0x4508be79377c4b20 closed |
org.apache.zookeeper.ZooKeeper.close(ZooKeeper.java:1627)
2021-07-05 14:27:37,293 | INFO | main-EventThread | EventThread shut down for connection:
0x4508be79377c4b20 | org.apache.zookeeper.ClientCnxn$EventThread.run(ClientCnxn.java:585)
```

## NOTE

In the Windows environment, the following exception occurs but does not affect services.  
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

## Log description

The log level is **INFO** by default and more detailed information can be viewed by changing the log level (**DEBUG**, **INFO**, **WARN**, **ERROR**, and **FATAL**).

You can modify the **log4j.properties** file to change log levels, for example:

```
# Set root category priority to info and its only appender to console.  
log4j.rootCategory=info,console,R  
#log4j.debug=true  
  
# console is set to be a ConsoleAppender using a PatternLayout.  
log4j.appender.console=org.apache.log4j.ConsoleAppender  
log4j.appender.console.Threshold=DEBUG  
log4j.appender.console.layout=org.apache.log4j.PatternLayout  
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n  
  
# R is set to be a File appender using a PatternLayout.  
log4j.appender.R=org.apache.log4j.RollingFileAppender  
log4j.appender.R.Append=true  
log4j.appender.R.Threshold=debug  
log4j.appender.R.MaxFileSize=1024KB  
log4j.appender.R.MaxBackupIndex=10  
log4j.appender.R.File=logs/solrclient.log  
log4j.appender.R.layout=org.apache.log4j.PatternLayout  
log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n
```

## 1.20.4.2 Commissioning an Application in Linux

### 1.20.4.2.1 Compiling and Running an Application When a Client Is Installed

#### Scenario

In a Linux environment where an Solr client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

#### Prerequisites

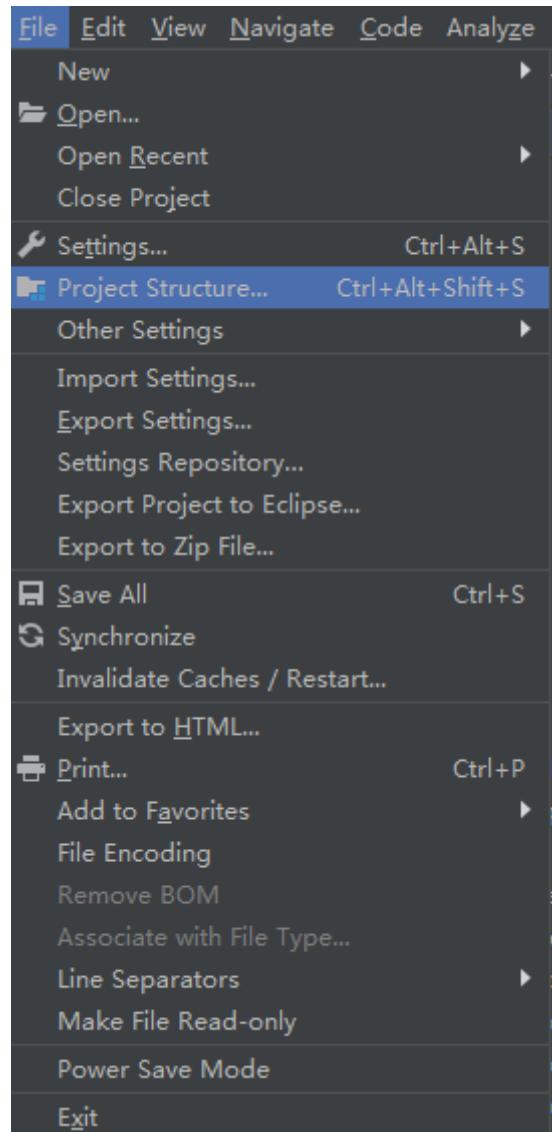
You have installed the Solr client.

#### Procedure

##### Step 1 Export the jar package.

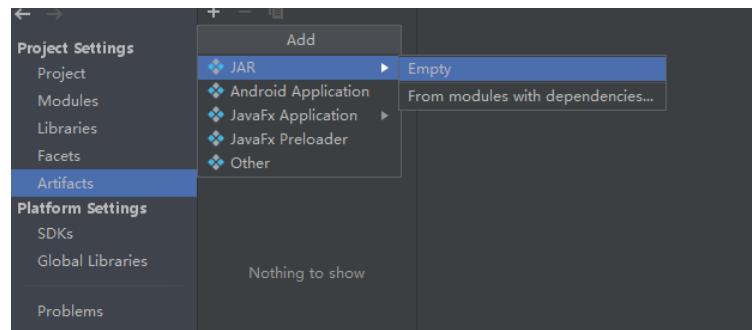
1. Log in to IntelliJ IDEA and choose **File > Project Structure**.

Figure 1-255 Project Structure



2. Select **Artifacts**, click +, and choose **JAR > Empty**.

Figure 1-256 Adding artifacts

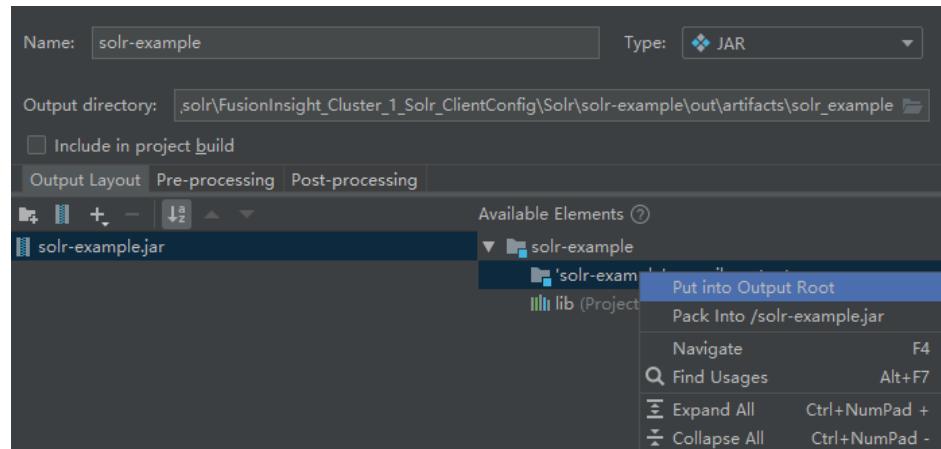


3. Change the value of **Name** to a user-defined name, for example, **solr-example**.



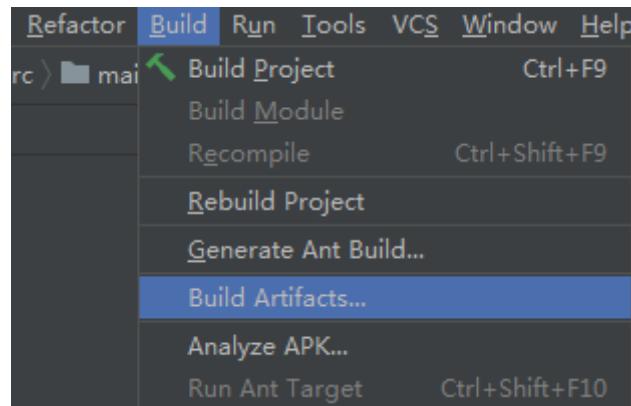
4. Select **Output Layout**, right-click 'solr-example' compile output in **Available Elements** on the right, and choose **Put Into Output Root**.

**Figure 1-257 Adding output**



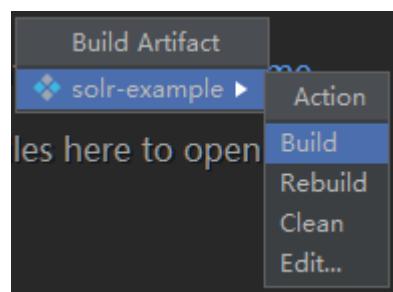
5. Click **Apply** to save the modification, and then click **OK** to exit.
6. Choose **Build > Build Artifacts....**

**Figure 1-258 Preparing for compilation**

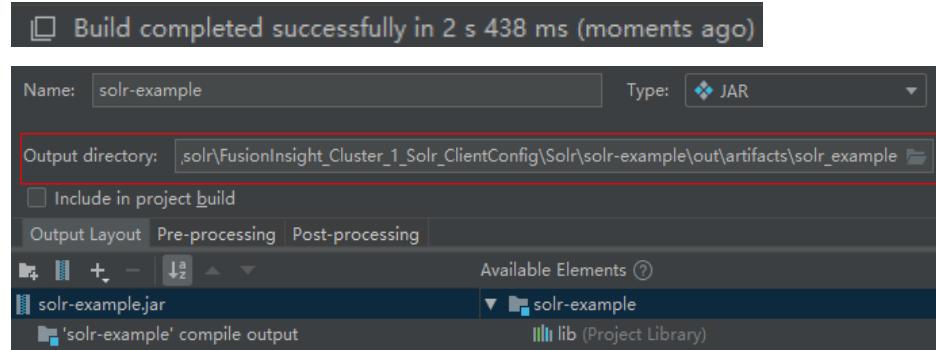


7. In the dialog box that is displayed, choose **solr-example > Build** to compile the artifact.

**Figure 1-259 Compiling an artifact**



8. If information similar to the following is displayed in the event log, the JAR file has been successfully generated. Obtain the JAR file from the **Output directory** configured in **Artifacts**.



#### Step 2 Run the jar package.

1. Assume that the client installation directory is **/opt/hadoopclient**. Copy the exported jar package **solr-example.jar** to **/opt/hadoopclient/Solr/tools/solr-example** in the client installation directory and copy the **conf** directory files of the project to **/opt/hadoopclient/Solr/tools/solr-example/conf**.
2. Run **source /opt/hadoopclient/bigdata\_env**, and then switch to the code function directory **cd /opt/hadoopclient/Solr/tools/solr-example**.
3. Run the java command:  
**java -cp /opt/hadoopclient/Solr/tools/lib/\*:solr-example.jar:/opt/hadoopclient/Solr/tools/solr-example/conf/com.huawei.fusioninsight.solr.example.TestSample**

----End

### 1.20.4.2.2 Compiling and Running an Application When No Client Is Installed

#### Scenario

In a Linux environment where no Solr client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

#### Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the jar package exported from IntelliJ IDEA. Java environment variables have been set.

#### Procedure

- Step 1** Export the jar package. For details, see [Compiling and Running an Application When a Client Is Installed](#).
- Step 2** Access the directory where the **pom.xml** file is stored in the **Terminal** window of the IDEA or other command line tools. Run the following command to generate the **lib** folder in the directory where **pom.xml** is located. The folder contains the JAR package on which the sample project depends.

**mvn dependency:copy-dependencies -DoutputDirectory=lib**

- Step 3** Copy the **lib** directory which the application development environment requires and conf directory to any directory in the client operating environment, for example, **/opt/solr-example**, and then copy the jar package generated in the application environment to **/opt/solr-example/lib/**.
- Step 4** On the client, run the **cd/opt/solr-example** command to switch to the copy directory.
- Step 5** Run the jar commands:

```
java -cp /opt/solr-example/lib/*:/opt/solr-example/conf/  
com.huawei.fusioninsight.solr.example.TestSample
```

----End

#### 1.20.4.2.3 Viewing Commissioning Results

##### Scenario

After an Solr application is run, you can use one of the following methods to view the running result:

- Viewing the command output.
- Viewing Solr logs.
- Logging in to the Solr WebUI. For details, see **Component Operation Guide > Solr > Common Service Operations > Operations on Solr Admin UI**.

##### Procedure

The following information is displayed in the running results:

```
...  
2021-07-05 14:35:14,910 | INFO | main | Client is connected to ZooKeeper |  
org.apache.solr.common.cloud.ConnectionManager.waitForConnected(ConnectionManager.java:251)  
2021-07-05 14:35:14,959 | INFO | main | Updated live nodes from ZooKeeper... (0) -> (3) |  
org.apache.solr.common.cloud.ZkStateReader.refreshLiveNodes(ZkStateReader.java:859)  
2021-07-05 14:35:14,976 | INFO | main | Cluster at  
192.168.1.189:24002,192.168.1.228:24002,192.168.1.150:24002/solr ready |  
org.apache.solr.client.solrj.impl.ZkClientClusterStateProvider.getZkStateReader(ZkClientClusterStateProvider.ja  
va:177)  
2021-07-05 14:35:14,977 | INFO | main | The cloud Server has been connected !!!! |  
com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:188)  
2021-07-05 14:35:14,977 | INFO | main | The zookeeper state is : znodeVersion: 0  
live nodes:[192.168.1.136:21104_solr, 192.168.1.177:21101_solr, 192.168.1.187:21101_solr]  
collections:{col_test=LazyCollectionRef(col_test)} |  
com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:192)  
Debug is true storeKey true useTicketCache false useKeyTab true doNotPrompt false ticketCache is null  
isInitiator true KeyTab is /opt/hadoopclient/Solr/tools/solr-example/conf/user.keytab refreshKrb5Config is  
false principal is solrtest tryFirstPass is false useFirstPass is false storePass is false clearPass is false  
principal is solrtest@HADOOP.COM  
Will use keytab  
Commit Succeeded  
  
2021-07-05 14:35:15,495 | INFO | main | All existed collections : [col_test] |  
com.huawei.fusioninsight.solr.example.TestSample.queryAllCollections(TestSample.java:316)  
2021-07-05 14:35:15,915 | INFO | main | Success to delete collection[col_test]. |  
com.huawei.fusioninsight.solr.example.TestSample.deleteCollection(TestSample.java:295)  
2021-07-05 14:35:18,803 | INFO | main | Success to create collection[col_test]. |  
com.huawei.fusioninsight.solr.example.TestSample.createCollection(TestSample.java:278)  
2021-07-05 14:35:18,930 | INFO | main | success to add index |
```

```
com.huawei.fusioninsight.solr.example.TestSample.addDocs(TestSample.java:229)
2021-07-05 14:35:21,758 | INFO | main | Query wasted time : 750ms |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:204)
2021-07-05 14:35:21,759 | INFO | main | Total doc num : 5 |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:205)
2021-07-05 14:35:21,759 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0,
price_c=0,XXX, _version_=1704425599989186560} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,760 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=2.0,
price_c=2,XXX, _version_=17044255999891283712} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,760 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=4.0,
price_c=4,XXX, _version_=1704425599990235136} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,760 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=6.0,
price_c=6,XXX, _version_=1704425599993380864} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,760 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=8.0,
price_c=8,XXX, _version_=1704425599994429440} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,943 | INFO | main | Success to delete index |
com.huawei.fusioninsight.solr.example.TestSample.removeIndex(TestSample.java:218)
2021-07-05 14:35:21,969 | INFO | main | Query wasted time : 21ms |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:204)
2021-07-05 14:35:21,969 | INFO | main | Total doc num : 5 |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:205)
2021-07-05 14:35:21,969 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0,
price_c=0,XXX, _version_=1704425599989186560} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,970 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=2.0,
price_c=2,XXX, _version_=1704425599991283712} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,970 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=4.0,
price_c=4,XXX, _version_=1704425599990235136} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,970 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=6.0,
price_c=6,XXX, _version_=1704425599993380864} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:21,970 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=8.0,
price_c=8,XXX, _version_=1704425599994429440} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:207)
2021-07-05 14:35:22,080 | INFO | main-EventThread | EventThread shut down for connection:
0x4308be794ece3be7 | org.apache.zookeeper.ClientCnxn$EventThread.run(ClientCnxn.java:585)
```

## 1.20.5 More Information

### 1.20.5.1 External Interfaces

#### 1.20.5.1.1 Shell

You can directly perform operations on Solr by running Shell client scripts.

Methods of running the Shell command (assuming that the client installation directory is: **/opt/hadoopclient**):

```
source /opt/hadoopclient/bigdata_env
```

```
kinit solr service user
```

Run **/opt/hadoopclient/Solr/solr-client/bin/solrctl --help** to obtain the help information for specific Solr parameters.

```
usage: solrctl [options] command [command-arg] [command [command-arg]] ...
```

```
Options:  
--help  
--quiet  
  
Commands:  
confset [--generate path [-schemaless | -confWithHBase]]  
[--create name path -force]  
[--update name path -force]  
[--get name path]  
[--putfile path localpath]  
[--getfile path filename]  
[--clearfile path]  
[--delete name]  
[--list]  
  
collection [--create name -s <numShards>  
[-a Create collection with autoAddReplicas=true]  
[-S <true|false> Create collection true or false with sharedFsReplication]  
[-b <true|false> Create collection true or false with autoShardBalance]  
  
[-c <collection.configName>]  
[-r <replicationFactor>]  
[-m <maxShardsPerNode>]  
[-n <createNodeSet>]  
[-f <router.field>]  
[-p name=value ...]  
[--modify name]  
[-a <true|false> Modify collection true or false with autoAddReplicas]  
[-b <true|false> Modify collection true or false with autoShardBalance]  
[-r <replicationFactor>]  
[-m <maxShardsPerNode>]  
  
[--delete name]  
[--reload name]  
[--splitshard collection shard]  
[--createshard collection shard]  
[--deleteshard collection shard]  
[--createalias aliasname collections]  
[--deletealias aliasname]  
[--deleterepllica [-p name=value]...]  
[--addrepllica [-p name=value]...]  
[--deletedocs name]  
[--list]  
[--replacenode source_node target_node  
[-p Replace node with parallel=true]  
[-t <timeout in ms>]  
[-c Replace node with skipCheck=false]  
]  
  
cluster [--get-solrxml file]  
[--put-solrxml file]  
[--set-property name value]  
[--remove-property name]  
[--list-free-node]  
[--list-local-disk-live-node]
```

### 1.20.5.1.2 Java API

Common Solr Java classes are as follows:

- CloudSolrClient: core class of client applications. It encapsulates HttpClient and communicates with SolrCloud.
- CollectionAdminRequest: various request APIs. It is displayed in the internal form, such as CollectionAdminRequest.Create, CollectionAdminRequest.List, and CollectionAdminRequest.Delete.

- SolrInputDocument: index record.
- SolrQuery: index query class.

For details about the API of each class, see: <https://lucene.apache.org/solr/8.4.0/solr-solrj/index.html?overview-summary.html>

### 1.20.5.1.3 Web UI

#### Scenario

The Web UI displays the Solr cluster status, including summary of a cluster and information about SolrServer, SolrServerAdmin, snapshots, and running processes. You can better understand the Solr cluster status based on information displayed on the Web UI.

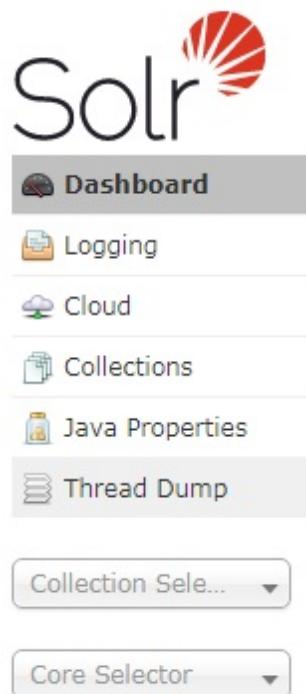
##### NOTE

Contact the administrator to obtain a service account that has the right to access the WebUI and obtain its password.

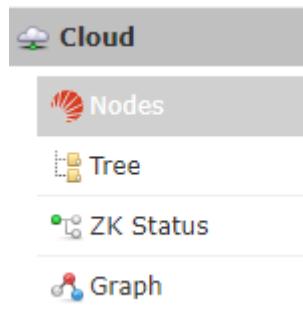
#### Procedure

- Step 1** Accessing FusionInsight Manager of an MRS Cluster. Choose **Cluster > Name of the desired cluster > Services > Solr > SolrServerAdmin** and open the Web UI in Solr.
- Step 2** The Solr web UI page includes several parts several parts, as shown in [Figure 1-260](#).

**Figure 1-260** Solr Web UI control menu



- DashBoard: displays some system parameters, such as JVM startup parameters and JVM memory usage.
- Logging: displays logs at the warn and upper levels on the page, and provides the function of dynamically adjusting the level of logs.
- Cloud: displays each structure of the Collection.



- Tree displays the tree-like structure for the directory files of index in Zookeeper.
- Graph displays the relationship among each Collection, shard, and replica.
- Graph (Radial) displays the relationship among Collection, shard, and replica in radial shape.
- Dump enables a user to download ZooKeeper configuration files of a Collection.
- Collections: Management UI of collections. It provides functions such as Add, Delete, Reload, Create Alias and Delete Alias.
- Java Properties: lists all environment variables used by Solr.
- Thread Dump: thread stack, lists information about all thread stacks running in the background.
- Core Selector: is used to choose a core as well as create data indexes for and search data in the core.

----End

## 1.21 Spark Development Guide

### 1.21.1 Overview

#### 1.21.1.1 Application Development Overview

##### Spark Introduction

Spark is a distributed batch processing system as well as an analysis and mining engine. It provides an iterative memory computation framework and supports the development in multiple programming languages, including Scala, Java, and Python. The application scenarios of Spark include:

- Data processing: Spark can process data quickly and has fault tolerance and scalability.

- Iterative computation: Spark supports iterative computation to meet the requirements of multi-step data processing logic.
- Data mining: Based on massive data, Spark can handle complex data mining and analysis and support multiple data mining and machine learning algorithms.
- Streaming processing: Spark supports stream processing at a seconds-level delay and supports multiple external data sources.
- Query analysis: Spark supports standard SQL query analysis, provides the DSL (DataFrame), and supports multiple external inputs.

This section focuses on the application development guides of Spark, Spark SQL and Spark Streaming.

## Spark Development API Introduction

Spark supports the development in multiple programming languages, including Scala, Java, and Python. Since Spark is developed in Scala and Scala is easy to read, users are advised to develop Spark application in Scala.

Divided by different languages, the APIs of Spark are listed in [Table 1-172](#).

**Table 1-172** Spark APIs

Function	Description
Scala API	Indicates the API in Scala. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Scala</a> . Since Scala is easy to read, users are advised to use Scala APIs in the program development.
Java API	Indicates the API in Java. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Java</a> .
Python API	Indicates the API in Python. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Python</a> .

Divided by different modes, APIs listed in the preceding table are used in the development of Spark Core and Spark Streaming. Spark SQL supports CLI and JDBCServer for accessing. There are two ways to access the JDBCServer: Beeline and the JDBC client code. For details, see [JDBCServer Interface](#).

### NOTE

For spark-sql, spark-shell and spark-submit (which application contains SQL operations), do not use the **proxy user** parameter to submit a task. This is partly because the spark-sql script with the **proxy user** parameter does not support task submission and partly because the sample program mentioned in this document already contains security authentication.

### 1.21.1.2 Basic Concepts

#### Basic Concepts

- **RDD**

Resilient Distributed Dataset (RDD) is a core concept of Spark. It indicates a read-only and partitioned distributed dataset. Partial or all data of this dataset can be cached in the memory and reused between computations.

##### RDD Generation

- An RDD can be generated from the Hadoop file system or other storage systems that are compatible with Hadoop, such as Hadoop Distributed File System (HDFS).
- A new RDD can be transferred from a parent RDD.
- An RDD can be converted from a collection.

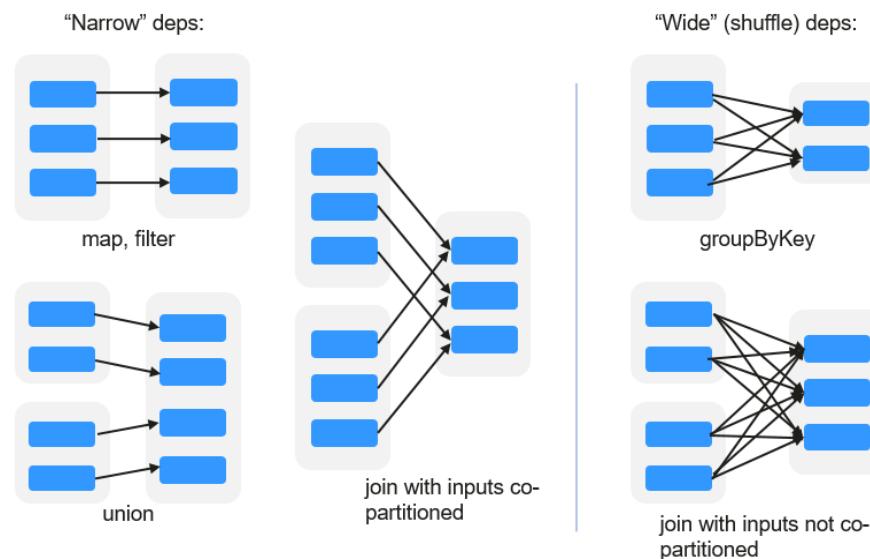
##### RDD Storage

- Users can select different storage levels to store an RDD for reuse. (There are 11 storage levels to store an RDD.)
- The current RDD is stored in the memory by default. When the memory is insufficient, the RDD overflows to the disk.

- **RDD Dependency**

The RDD dependency includes the narrow dependency and wide dependency.

**Figure 1-261** RDD dependency



- **Narrow dependency:** Each partition of the parent RDD is used by at most one partition of the child RDD partition.
- **Wide dependency:** Partitions of the child RDD depend on all partitions of the parent RDD due to shuffle operations.

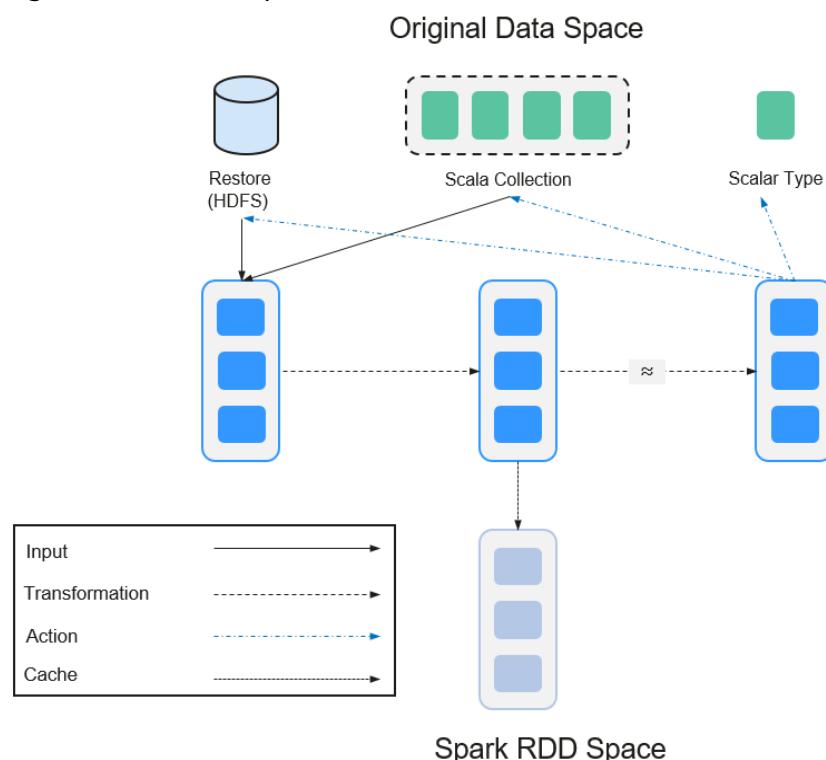
The narrow dependency facilitates the optimization. Logically, each RDD operator is a fork/join process. Fork the RDD to each partition, and then perform the computation. After the computation, join the results, and then

perform the fork/join operation on next operator. It takes a long period of time to directly translate the RDD to physical implementation. There are two reasons: Each RDD (even the intermediate results) must be physicalized to the memory or storage, which takes time and space; the partitions can be joined only when the computation of all partitions is complete (if the computation of a partition is slow, the entire join process is slowed down). If the partitions of the child RDD narrowly depend on the partitions of the parent RDD, the two fork/join processes can be combined to optimize the entire process. If the relationship in the continuous operator sequence is narrow dependency, multiple fork/join processes can be combined to reduce the time for waiting and improve the performance. This is called pipeline optimization in Spark.

- **Transformation and Action (RDD Operations)**

Operations on RDD include transformation (the returned value is an RDD) and action (the returned value is not an RDD). [Figure 1-262](#) shows the process. The transformation is lazy, which indicates that the transformation from one RDD to another RDD is not immediately executed. Spark only records the transformation but does not execute it immediately. The real computation is started only when the action is started. The action returns results or writes the RDD data into the storage system. The action is the driving force for Spark to start the computation.

**Figure 1-262** RDD operation



The data and operation model of RDD are quite different from those of Scala.

```
val file = sc.textFile("hdfs://...")  
val errors = file.filter(_.contains("ERROR"))  
errors.cache()  
errors.count()
```

- a. The **textFile** operator reads log files from HDFS and returns **file** (as an RDD).

- b. The filter operator filters rows with ERROR and assigns them to errors (a new RDD). The filter operator is a transformation.
- c. The cache operator caches errors for future use.
- d. The count operator returns the number of rows of errors. The count operator is an action.

**Transformation includes the following types:**

- The RDD elements are regarded as simple elements:

The input and output have the one-to-one relationship, and the partition structure of the result RDD remains unchanged, for example, map.

The input and output have the one-to-many relationship, and the partition structure of the result RDD remains unchanged, for example, flatMap (one element becomes a sequence containing multiple elements and then flattens to multiple elements).

The input and output have the one-to-one relationship, but the partition structure of the result RDD changes, for example, union (two RDDs integrates to one RDD, and the number of partitions becomes the sum of the number of partitions of two RDDs) and coalesce (partitions are reduced).

Operators of some elements are selected from the input, such as filter, distinct (duplicate elements are deleted), subtract (elements only exist in this RDD are retained), and sample (samples are taken).

- The RDD elements are regarded as Key-Value pairs.

Perform the one-to-one calculation on the single RDD, such as mapValues (the partition mode of the source RDD is retained, which is different from map).

Sort the single RDD, such as sort and partitionBy (partitioning with consistency, which is important to the local optimization).

Restructure and reduce the single RDD based on key, such as groupByKey and reduceByKey.

Join and restructure two RDDs based on key, such as join and cogroup.

 **NOTE**

The later three operations involve sorting and are called shuffle operations.

**Action is classified into the following types:**

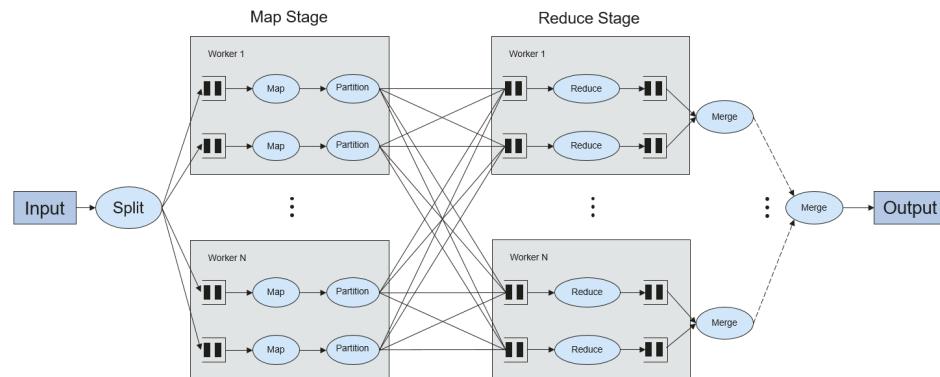
- Generate scalar configuration items, such as count (the number of elements in the returned RDD), reduce, fold/aggregate (the number of scalar configuration items that are returned), and take (the number of elements before the return).
- Generate the Scala collection, such as collect (import all elements in the RDD to the Scala collection) and lookup (look up all values corresponds to the key).
- Write data to the storage, such as saveAsTextFile (which corresponds to the preceding textFile).
- Check points, such as checkpoint. When Lineage is long (which occurs frequently in graphics computation), it takes a long period of time to execute the whole sequence again when a fault occurs. In this case, checkpoint is used as the check point to write the current data to stable storage.

- **Shuffle**

Shuffle is a specific phase in the MapReduce framework, which is located between the Map phase and the Reduce phase. If the output results of Map are to be used by Reduce, each output result must be hashed based on the key and distributed to the corresponding Reducer. This process is called Shuffle. Shuffle involves the read and write of the disk and the transmission of the network, so that the performance of Shuffle directly affects the operation efficiency of the entire program.

The following figure describes the entire process of the MapReduce algorithm.

**Figure 1-263** Algorithm process



Shuffle is a bridge connecting data. The following describes the implementation of shuffle in Spark.

Shuffle divides the Job of a Spark into multiple stages. The former stages contain one or multiple ShuffleMapTasks, and the last stage contains one or multiple ResultTasks.

- **Spark Application Structure**

The Spark application structure includes the initial `SparkContext` and the main program.

- Initial `SparkContext`: constructs the operation environment of the Spark application.

Constructs the `SparkContext` object. For example:

```
new SparkContext(master, appName, [SparkHome], [jars])
```

Parameter description

**master**: indicates the link string. The link modes include local, YARN-cluster, and YARN-client.

**appName**: indicates the application name.

**SparkHome**: indicates the directory where Spark is installed in the cluster.

**jars**: indicates the code and dependency package of the application.

- Main program: processes data.

For submitting applications details, see <https://spark.apache.org/docs/3.3.1/submitting-applications.html>.

- **Spark Shell Command**

The basic Spark shell command supports the submitting of the Spark application. The Spark shell command is

```
./bin/spark-submit \  
  --class <main-class> \  
  --master <master-url> \  
  ... # other options  
  <application-jar> \  
  [application-arguments]
```

Parameter description:

--class: indicates the name of the class of the Spark application.

--master: indicates the master that the Spark application links, such as YARN-client and YARN-cluster.

application-jar: indicates the path of the jar package of the Spark application.

application-arguments: indicates the parameter required to submit the Spark application. (This parameter can be empty.)

- **Spark JobHistory Server**

The Spark web UI is used to monitor the details in each phase of the Spark framework of a running or historical Spark job and provide the log display, which helps users to develop, configure, and optimize the job in more fine-grained units.

## Spark SQL Basic Concepts

### DataSet

A DataSet is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each Dataset also has an untyped view called a DataFrame, which is a Dataset of Row.

DataFrame is a structured distributed data set composed of several columns, which is similar to a table in the relationship database or the data frame in R/Python. DataFrame is a basic concept in Spark SQL, and can be created by using multiple methods, such as structured data set, Hive table, external database, or RDD.

## Spark Streaming Basic Concepts

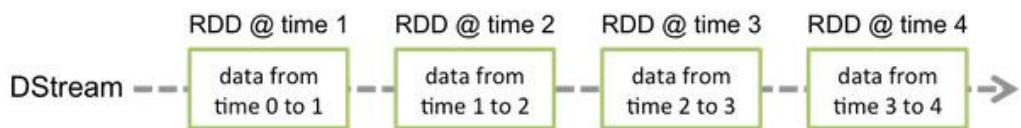
### DStream

DStream (Discretized Stream) is an abstract concept provided by the Spark Streaming.

DStream is a continuous data stream which is obtained from the data source or transformed and generated by the inflow. In essence, a DStream is a series of continuous RDDs. The RDD is a distributed dataset which can be read only and divided into partitions.

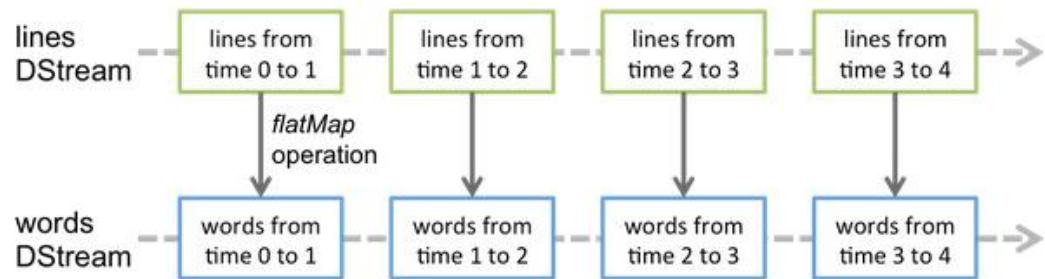
Each RDD in DStream contains the data of a partition as shown in [Figure 1-264](#).

**Figure 1-264** Relationship between DStream and RDD



All operators applied in the DStream are translated to the operations in the lower RDD, as shown in [Figure 1-265](#). The transformation of the lower RDDs is calculated by using the Spark engine. Most operation details are concealed in DStream operators and High-level APIs are provided for developers.

**Figure 1-265** DStream operator transfer



## Structured Streaming Basic Concepts

- **Input Source**

Input data sources. Data sources must support data replay based on the offset. Different data sources have different fault tolerance capabilities.

- **Sink**

Data output. Sink must support idempotence write operations. Different Sinks have different fault tolerance capabilities.

- **outputMode**

Result output mode, which can be:

- Complete Mode: The entire updated result table will be written to the external storage. It is up to the storage connector to decide how to handle writing of the entire table.
- Append Mode: If an interval is triggered, only the new rows appended in the Result Table will be written into an external system. This mode is applicable only to a result set that has already existed and will not be updated.
- Update Mode: If an interval is triggered, only updated data in the result table will be written into an external system, which is the difference between the Complete Mode and Update Mode.

- **Trigger**

Output trigger. Currently, the following trigger types are supported:

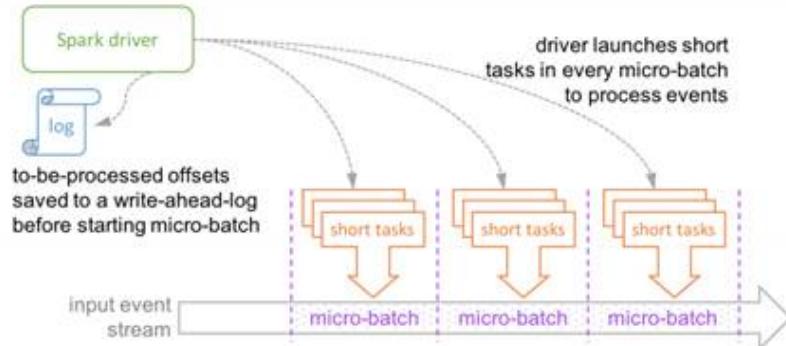
- Default: Micro-batch mode. After a batch is complete, the next batch is automatically executed.
- Specific interval: Processing is performed at a specific interval.
- One-time execution: Query is performed only once.
- Continuous mode: This is an experimental feature. In this mode, the stream processing delay can be decreased to 1 ms.

Structured Streaming supports the micro-batch mode and continuous mode. The micro-batch mode cannot ensure low-delay but has a larger throughput within the same time. The continuous mode is suitable for data processing requiring millisecond-level delay, which is an experimental feature.

 NOTE

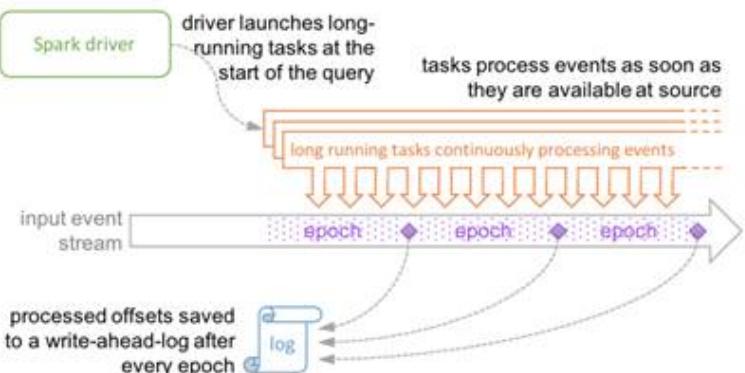
In the current version, if the stream-to-batch joins function is required, **outputMode** must be set to **Append Mode**.

**Figure 1-266** Running process in micro-batch mode



Micro-batch Processing uses periodic tasks to process events

**Figure 1-267** Running process in continuous mode



Continuous Processing uses long-running tasks to continuously process events

### 1.21.1.3 Development Process

#### Development Process of a Spark Application

Spark includes Spark Core, Spark SQL and Spark Streaming, whose development processes are the same.

[Figure 1-268](#) and [Table 1-173](#) describe the stages in the development process.

Figure 1-268 Spark development process

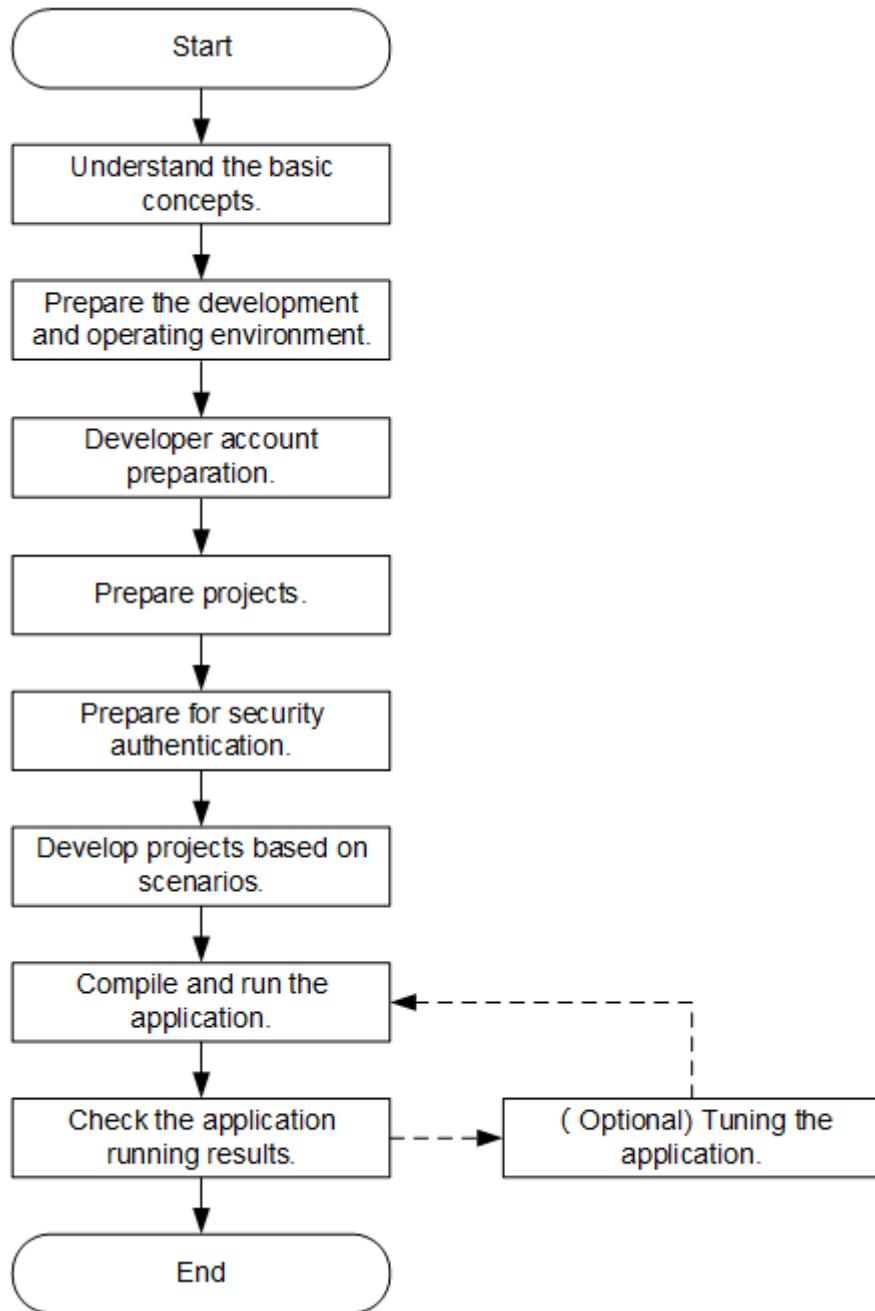


Table 1-173 Description of Spark development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of Spark are required to be understood. Choose the concepts required to be understood based on the actual scenario. The basic concepts include the basic concept of Spark Core, basic concept of Spark SQL and basic concept of Spark Streaming.	<a href="#">Basic Concepts</a>

Stage	Description	Reference
Prepare the development and operating environment.	The Spark application is developed in Scala, Java, and Python. The IDEA tool is recommended to prepare development environments in different languages based on the reference. The running environment of Spark is the Spark client. Install and configure the client based on the reference.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare a developer account.	The development account is used to run the sample project. Only the developer user with permission to access HDFS, YARN, Kafka, and Hive is allowed to run Spark sample projects.	<a href="#">Preparing a Developer Account</a>
Prepare projects.	Spark provides sample projects in various scenarios. Sample projects can be imported for studying. Or you can create a new Spark project based on the reference.	<a href="#">Configuring and Importing Sample Projects</a> <a href="#">Creating a New Project (Optional)</a>
Prepare for safety authentication.	If a safe cluster is used, the safety authentication must be performed.	<a href="#">Preparing for Security Authentication</a>
Develop projects based on scenarios	Sample projects in different languages including Scala, Java, and Python are provided. Sample projects in different scenarios including Streaming, SQL, JDBC client program, and Spark on HBase are also provided.  This helps users to better understand the programming interfaces of all Spark components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Compiling and Running the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Checking the Commissioning Result</a>

Stage	Description	Reference
Tune the application.	<p>Based on the application running results, tune the application to meet the requirements of the service scenario.</p> <p>After application tuning, compile and run the application again.</p>	"Component Operation Guide" > "Using Spark" > "Spark Performance Tuning" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i> .

## 1.21.2 Preparing for the Environment

### 1.21.2.1 Preparing for Development and Operating Environment

- Spark applications can be developed in Scala, Java, and Python. [Table 1-174](#) describes the development and operating environment to be prepared.

**Table 1-174** Development environment

Preparation Item	Description
OS	<ul style="list-style-type: none"><li>Development environment: Windows OS. Windows 7 or later is supported.</li><li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the runtime environment must be able to communicate with the cluster service plane network.</li></ul>

Preparation Item	Description
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>It is a tool used to develop Spark applications. Version 2019.1 or other compatible versions are recommended.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
Scala installation	It is the basic configuration for the Scala development environment. The required version is 2.12.14.

Preparation Item	Description
Scala plug-in installation	It is the basic configuration for the Scala development environment. The required version is 2018.2.11 or other compatible versions.
Editra installation	Editra is an editor in the Python development environment and is used to compile Python programs. You can also use other IDEs for Python programming.
Developer account preparation	See <a href="#">Preparing a Developer Account</a> for configuration.
7-zip	Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.
Python installation	Its version must be 3.7 or later.

## Preparing a Runtime Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. **Accessing FusionInsight Manager of an MRS Cluster** In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file, and then decompress it to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\Spark\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Spark sample project.

The keytab file obtained during the operation of [Preparing a Developer Account](#) is also stored in this directory. **Table 1-175** describes the main configuration files.

**Table 1-175 Configuration files**

File	Function
carbon.properties	CarbonData configuration file
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.
spark-defaults.conf	Configures Spark parameters.
spark-env.sh	Spark environment variable configuration file
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Provides Kerberos server configuration information.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
  - The local **hosts** file in a Windows environment is stored in, for example, C:\WINDOWS\system32\drivers\etc\hosts.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
    - a. Install the client in a directory, for example, /opt/hadoopclient, on the node.

Ensure that the difference between the client time and the cluster time is less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. **Accessing FusionInsight Manager of an MRS Cluster.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the

**FusionInsight\_Cluster\_1\_Services\_ClientConfig/Spark/config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Spark/config/* root@/IP address of the client node:/opt/  
hadoopclient/conf
```

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. [Table 1-176](#) describes the main configuration files.

**Table 1-176** Configuration files

File	Function
carbon.properties	CarbonData configuration file
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.
spark-defaults.conf	Configures Spark parameters.

File	Function
spark-env.sh	Spark environment variable configuration file
user.keytab	Provides HDFS user information for Kerberos security authentication.
krb5.conf	Provides Kerberos server configuration information.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

### 1.21.2.2 Configuring and Importing Sample Projects

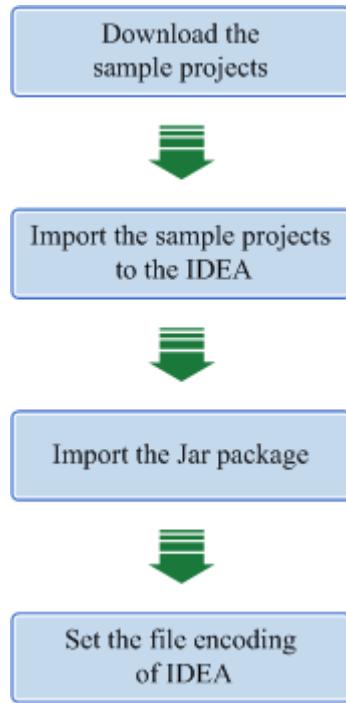
#### Scenario

Spark provides sample projects for multiple scenarios, including Java projects and Scala projects. This helps users to learn Spark projects quickly.

Import methods of Java and Scala projects are the same. Sample projects developed by using the Python do not need to be imported, and you only need to open the Python file (\*.py).

The import of Java sample codes is used as a sample in the following procedure. [Figure 1-269](#) shows the procedure if importing sample projects.

Figure 1-269 Procedure of importing sample projects



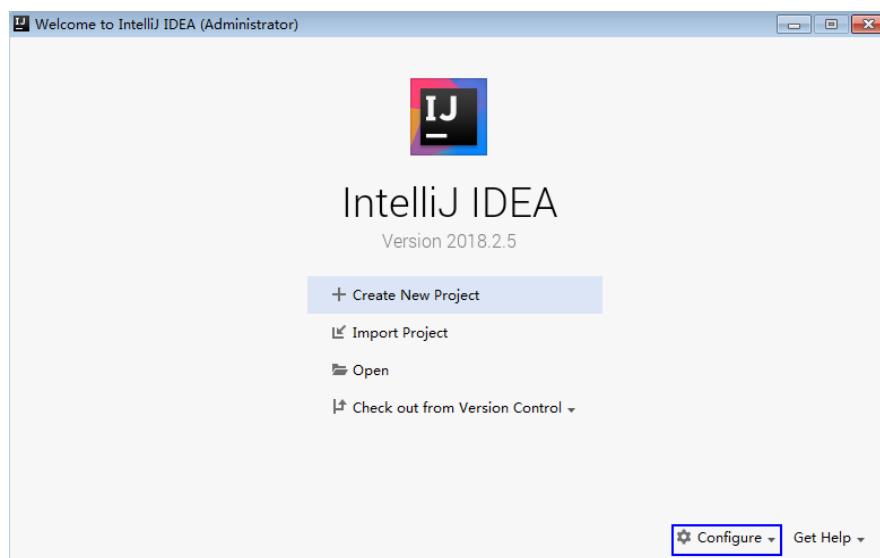
## Procedure

**Step 1** Obtain multiple sample projects such as Scala and Spark Streaming in the **sparksecurity-examples** folder in the **spark-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.

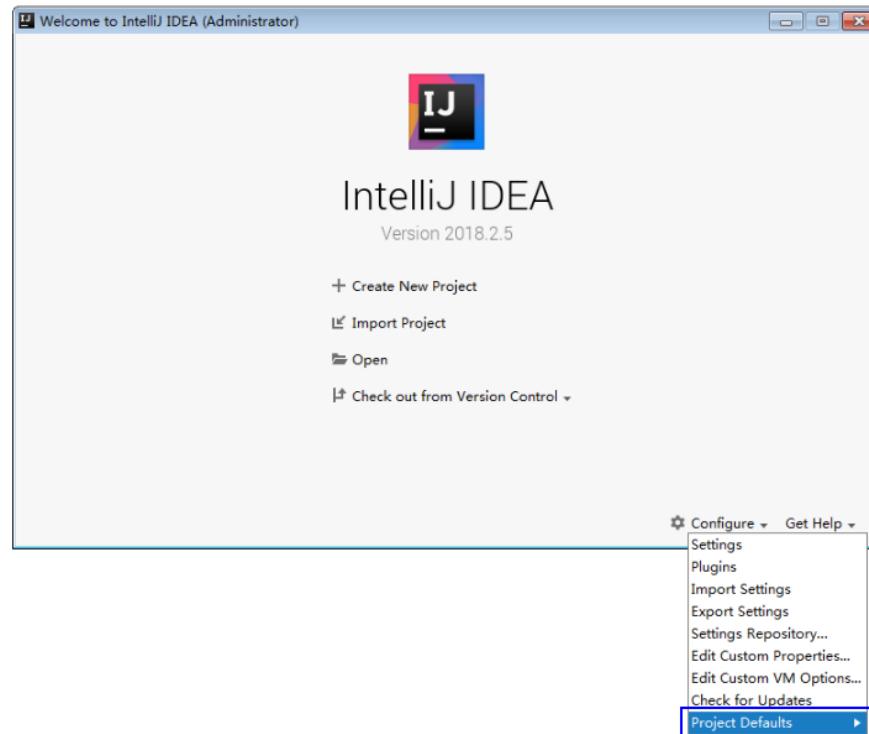
1. Start the IntelliJ IDEA and select **Configure**.

Figure 1-270 Quick Start



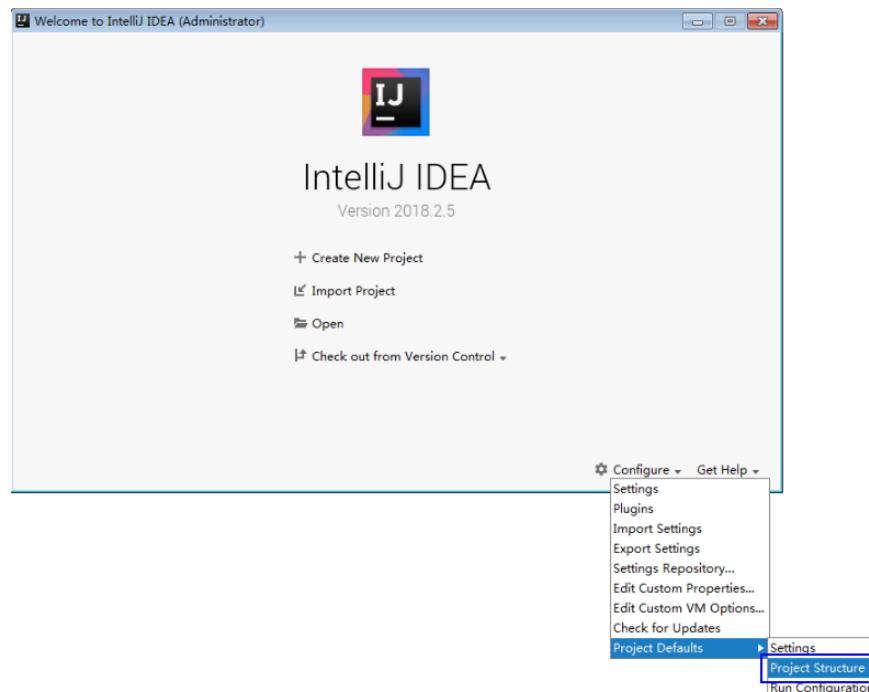
2. Select **Project Defaults** from the **Configure** drop-down list.

Figure 1-271 Configure



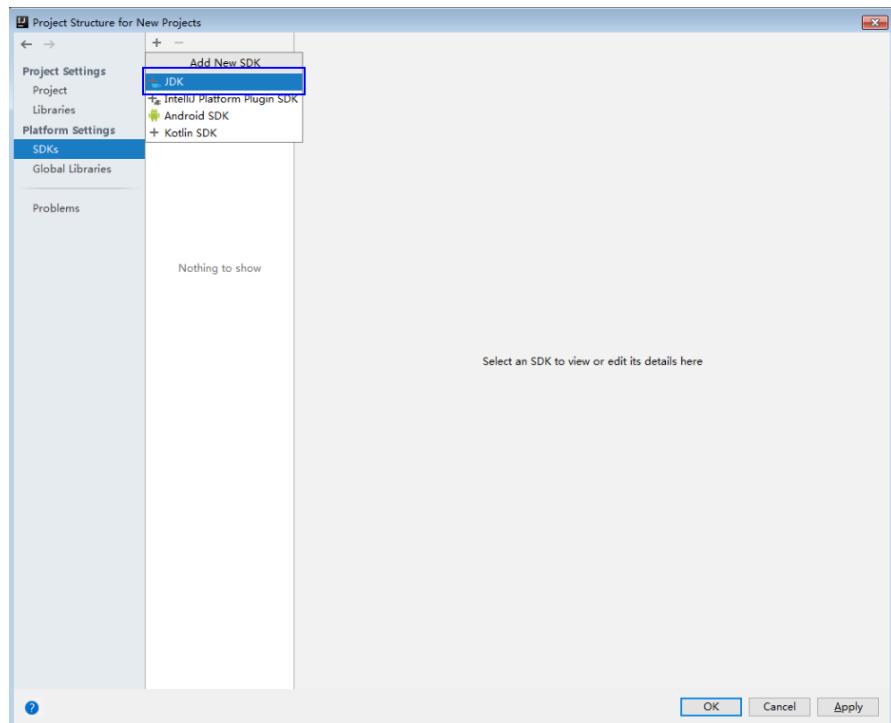
3. Select **Project Structure** from **Project Defaults**.

Figure 1-272 Project Defaults



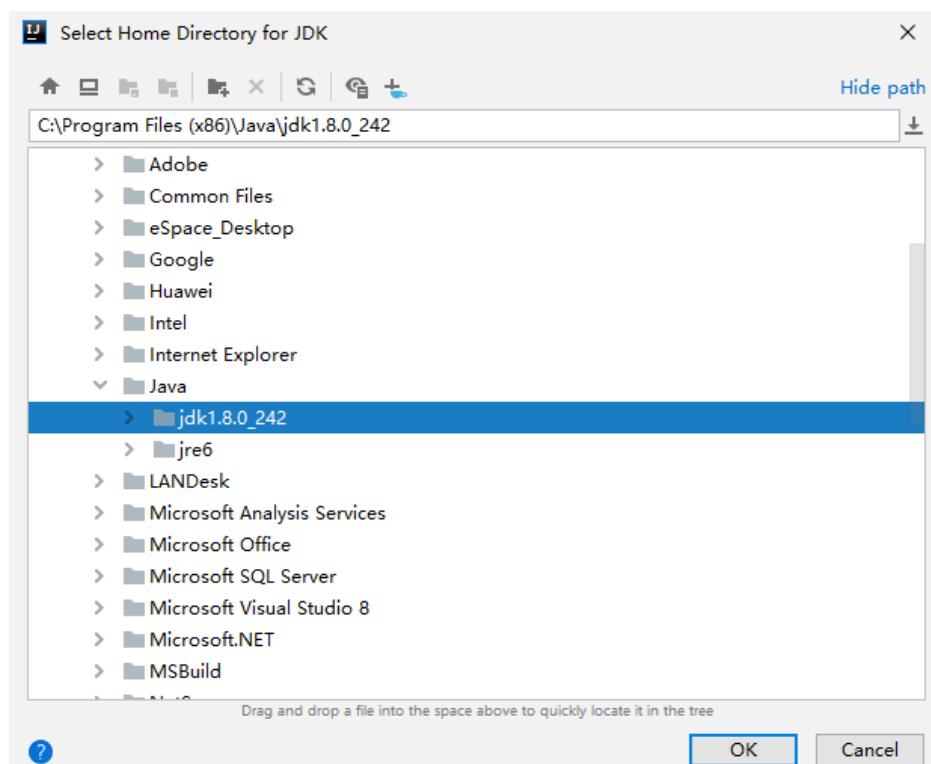
4. On the displayed **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 1-273 Adding the JDK



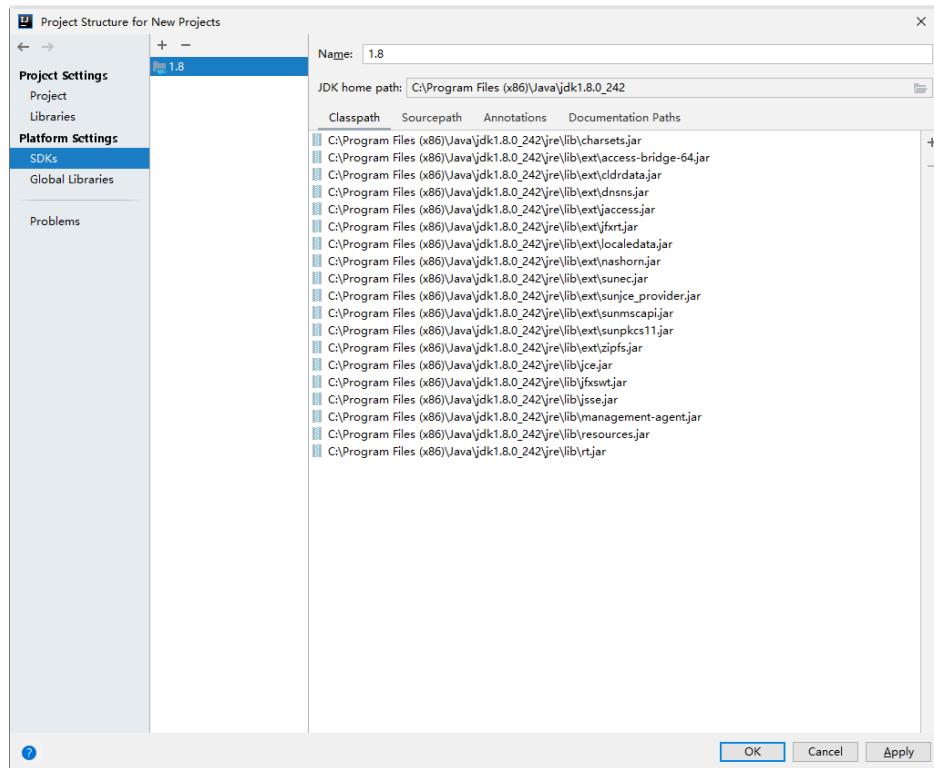
5. In the displayed **Select Home Directory for JDK** window, select a home directory for JDK and click **OK**.

Figure 1-274 Selecting a home directory for the JDK



6. After selecting the JDK, click **OK** to complete the configuration.

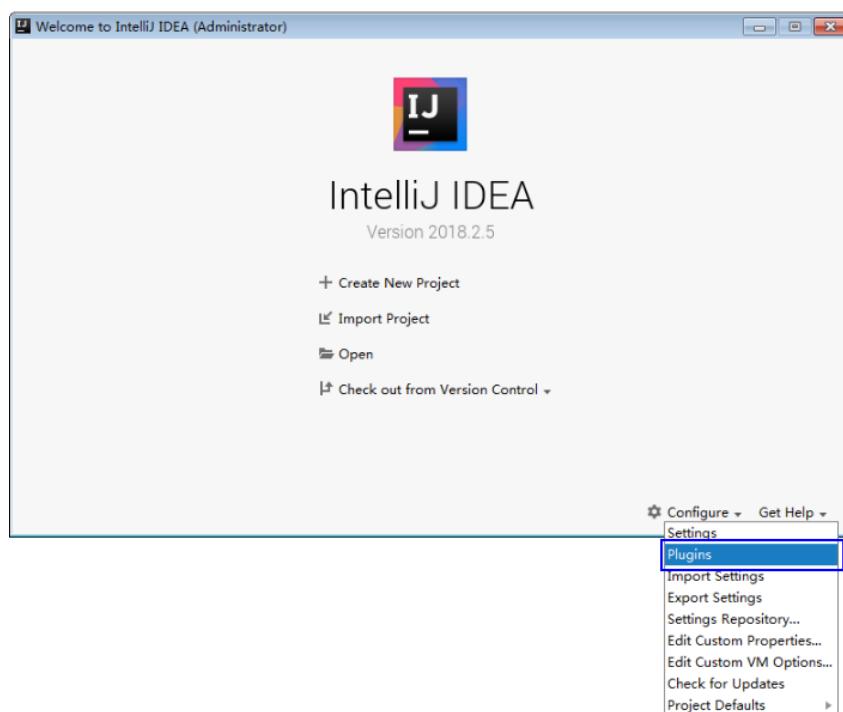
Figure 1-275 Completing the configuration



**Step 3** (Optional) If the Scala development environment is used, install the Scala plug-in in IntelliJ IDEA.

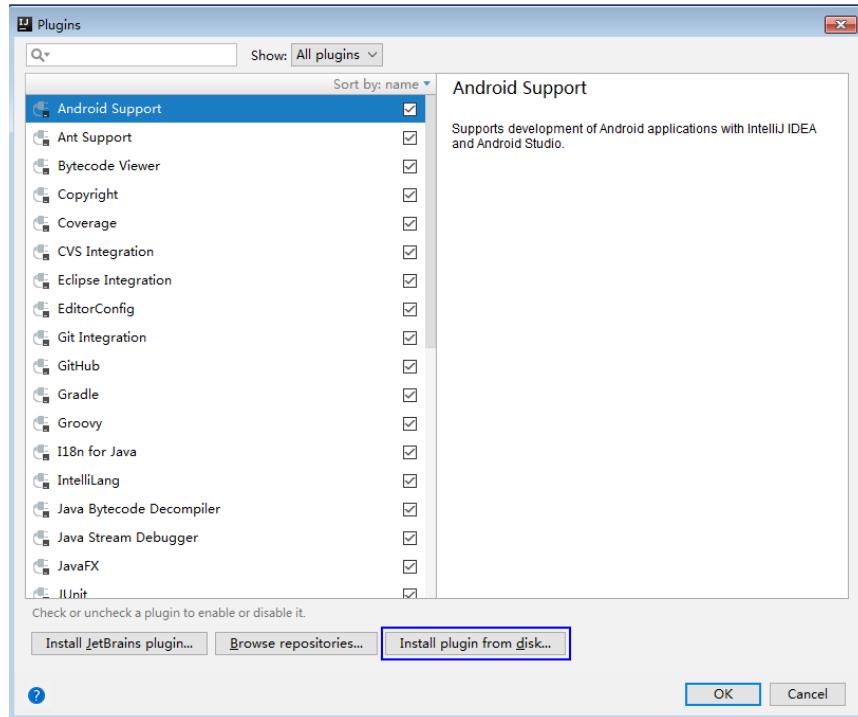
1. Select **Plugins** from the **Configure** drop-down list.

Figure 1-276 Plugins

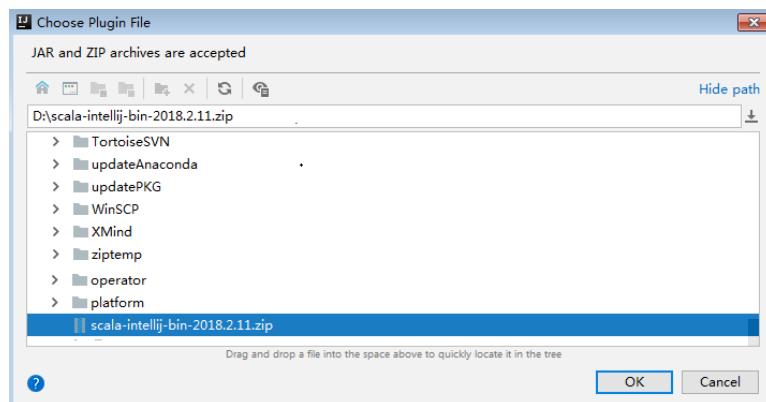


2. On the **Plugins** page, select **Install plugin from disk**.

**Figure 1-277** Install plugin from disk

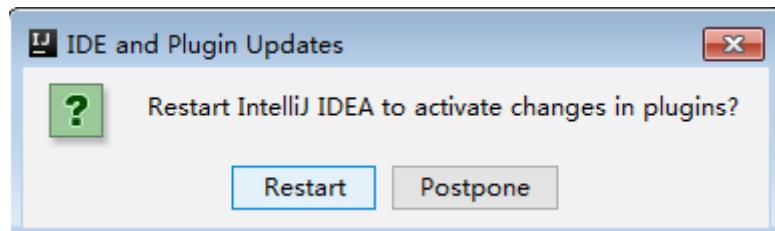


3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.



4. On the **Plugins** page, click **Apply** to install the Scala plugin.
5. On the displayed **Plugins Changed** page, click **Restart** for the configuration to take effect.

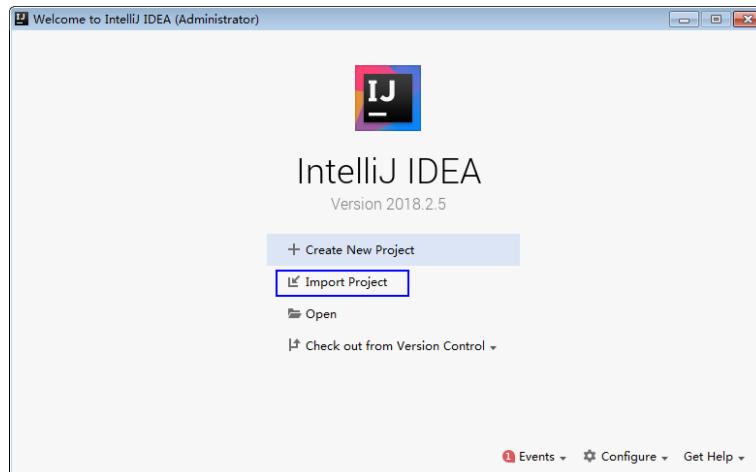
**Figure 1-278** Plugins Changed



**Step 4** Import the Java sample projects to the IDEA.

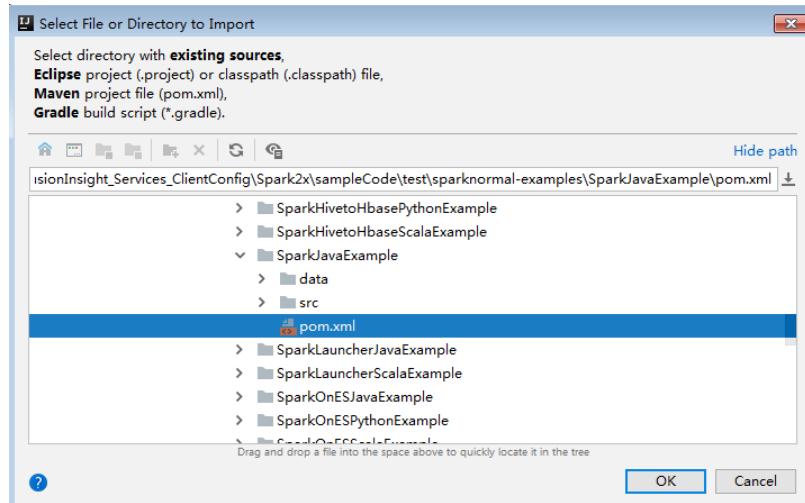
1. Start the IntelliJ IDEA. On the **Quick Start** page, select **Import Project**.  
Or, for the used IDEA tool, add projects directly from the IDEA homepage.  
Select **File > Import project...** to import projects.

**Figure 1-279** Import Project (on the Quick Start page)



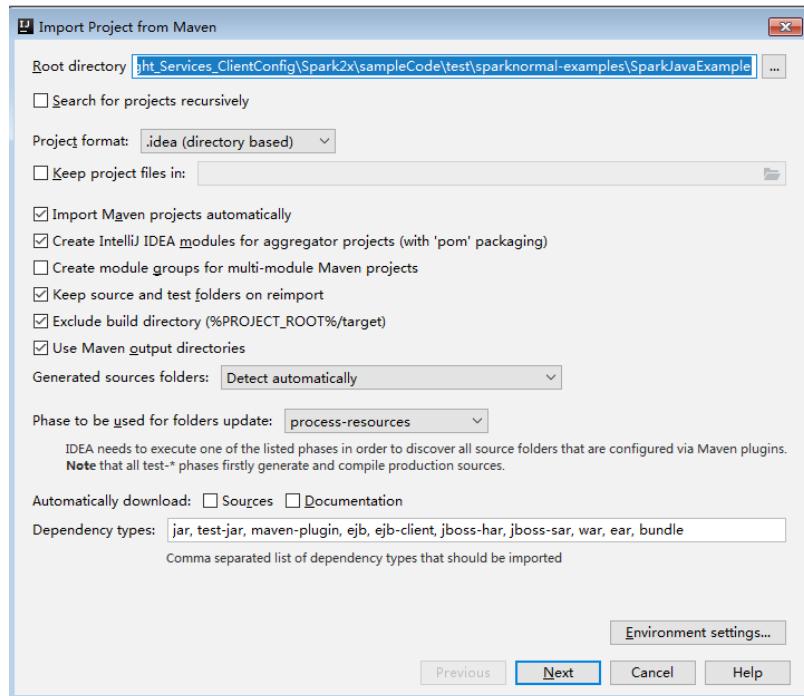
2. Select the directory to store the imported projects and the pom file, and click **OK**.

**Figure 1-280** Select File or Directory to Import



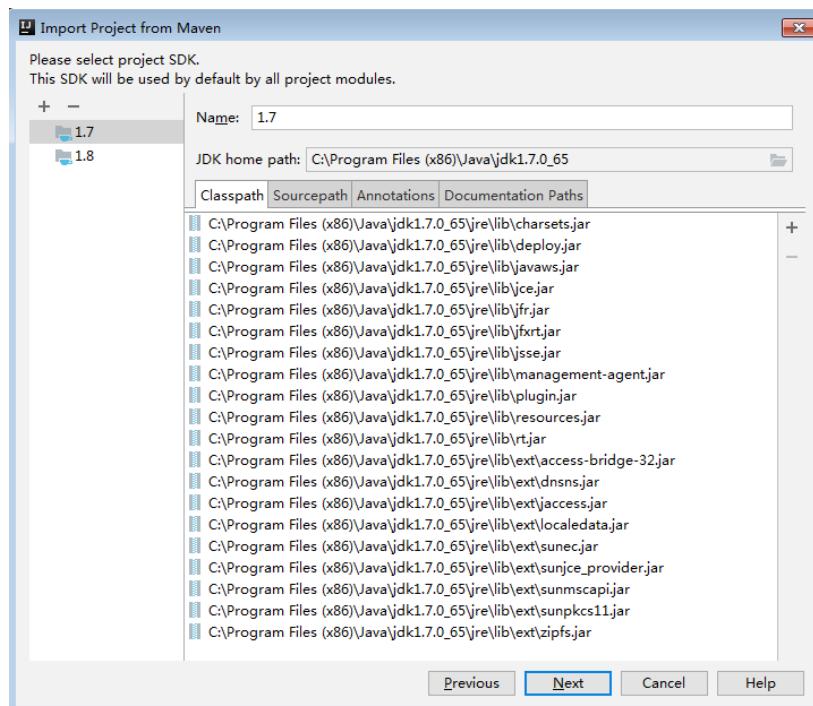
3. Confirm the import directory and project name, and click **Next**.

Figure 1-281 Import Project from Maven



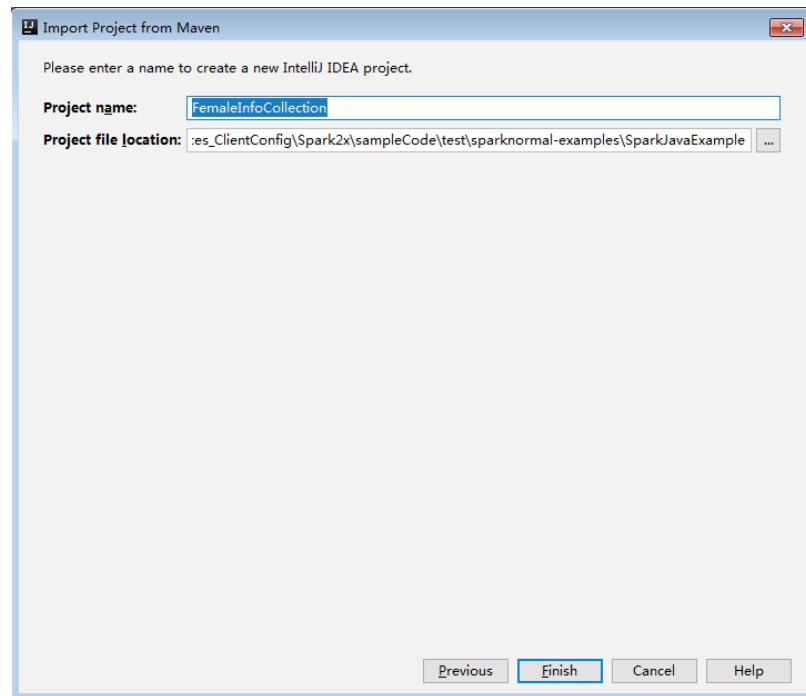
4. Select the projects to import and click **Next**.
5. Confirm the project JDK and click **Next**.

Figure 1-282 Select project SDK



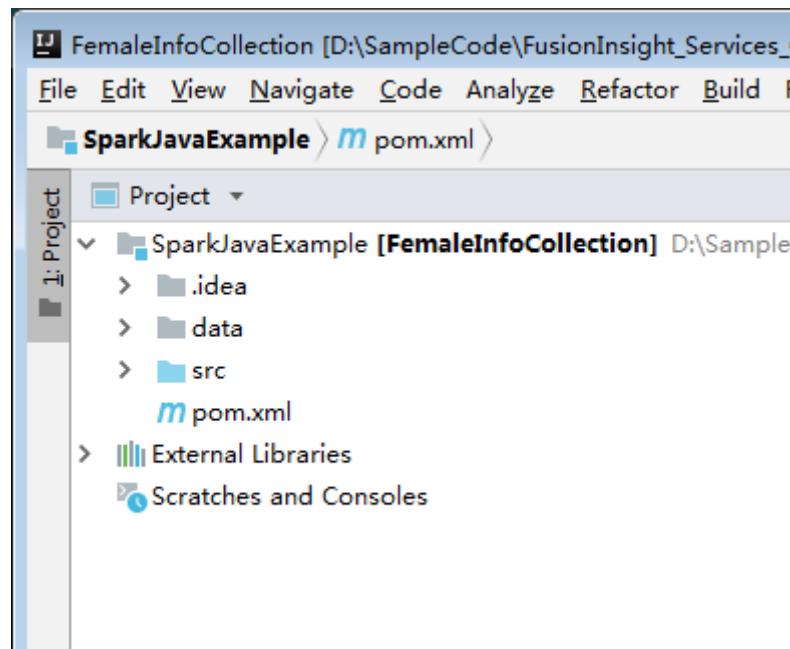
6. Confirm the project name and project file location, and click **Finish** to complete the import.

Figure 1-283 Confirm the project name and file location



- After the import, the imported projects are displayed on the IDEA homepage.

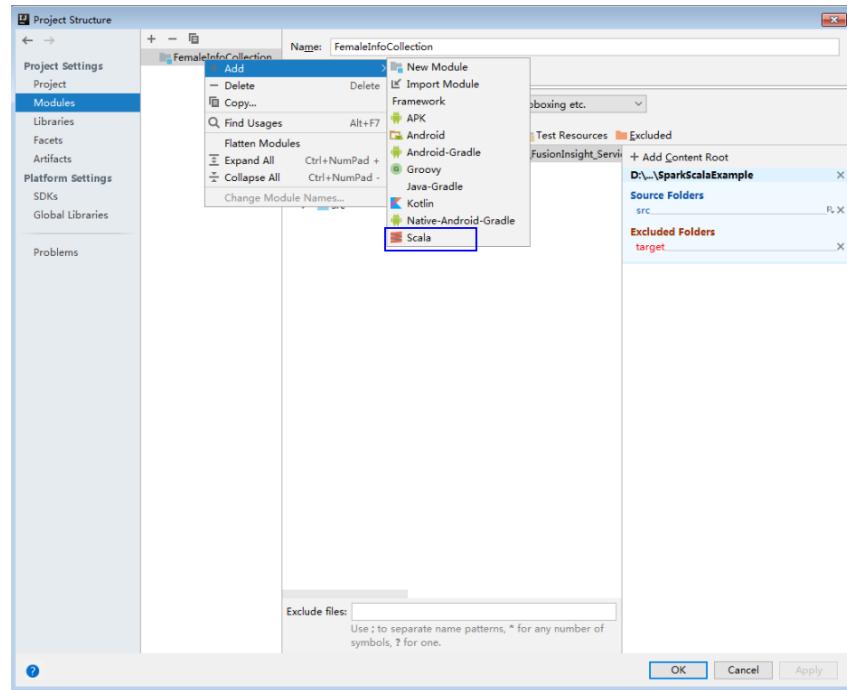
Figure 1-284 Imported projects



**Step 5** (Optional) If a sample application developed in Scala is imported, configure the language for the project.

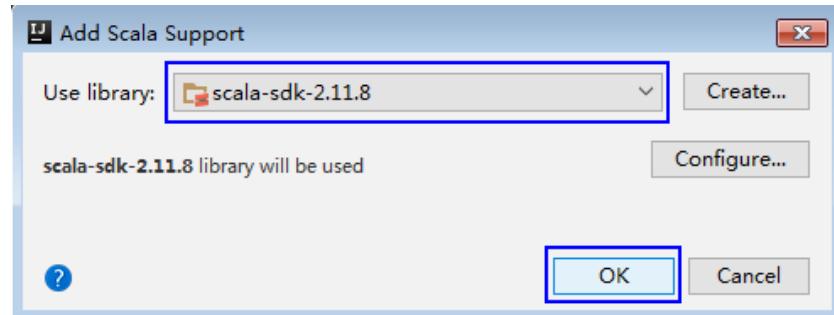
- On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
- Choose **Modules**, right-click the project name, and choose **Add > Scala**.

Figure 1-285 Selecting the Scala language



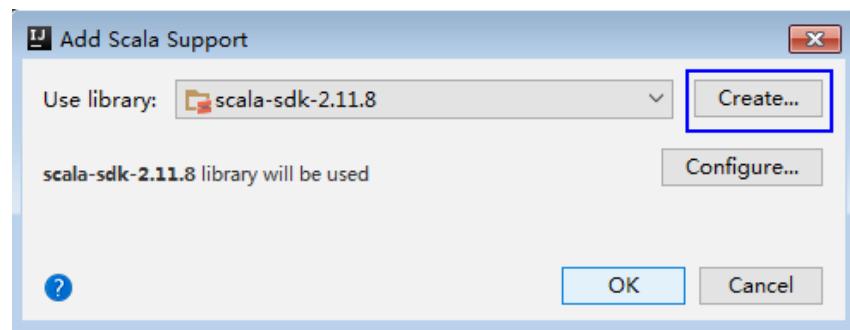
3. Wait until IDEA identifies Scala SDK, select the dependency JAR packages in the **Add Scala Support** dialog box, and then click **OK**

Figure 1-286 Add Scala Support



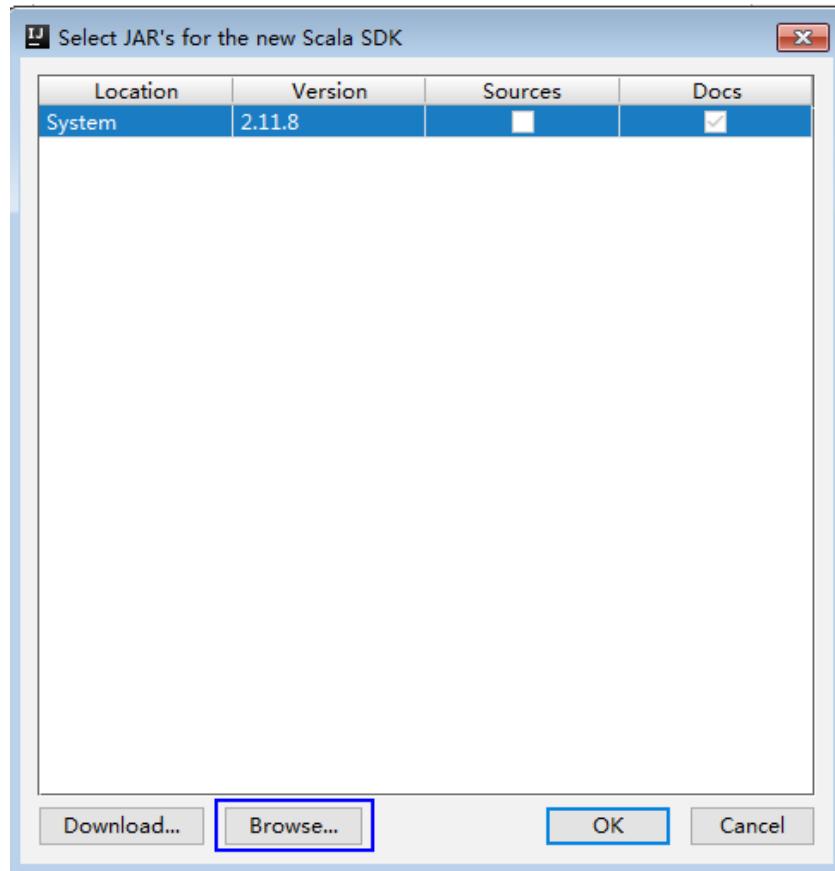
4. If IDEA fails to identify Scala SDK, you are required to create a Scala SDK.
  - a. Click **Create...**

Figure 1-287 Create...



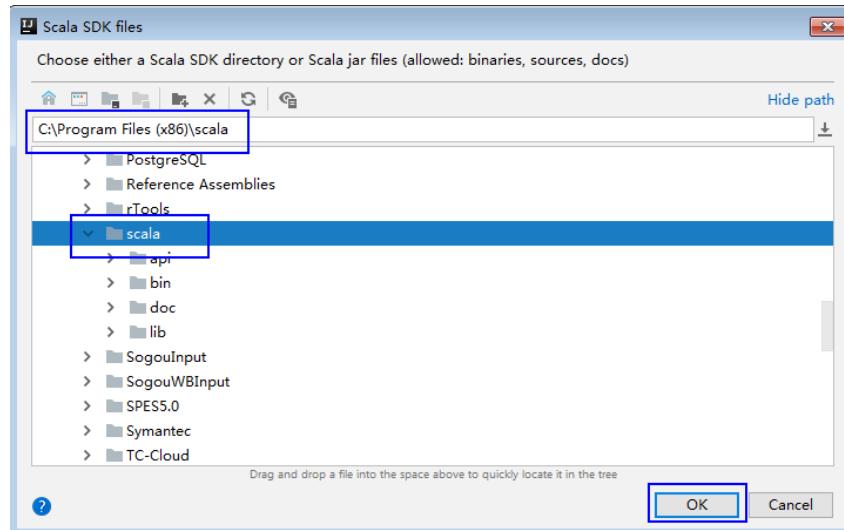
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**

Figure 1-288 Select JAR's for the new Scala SDK



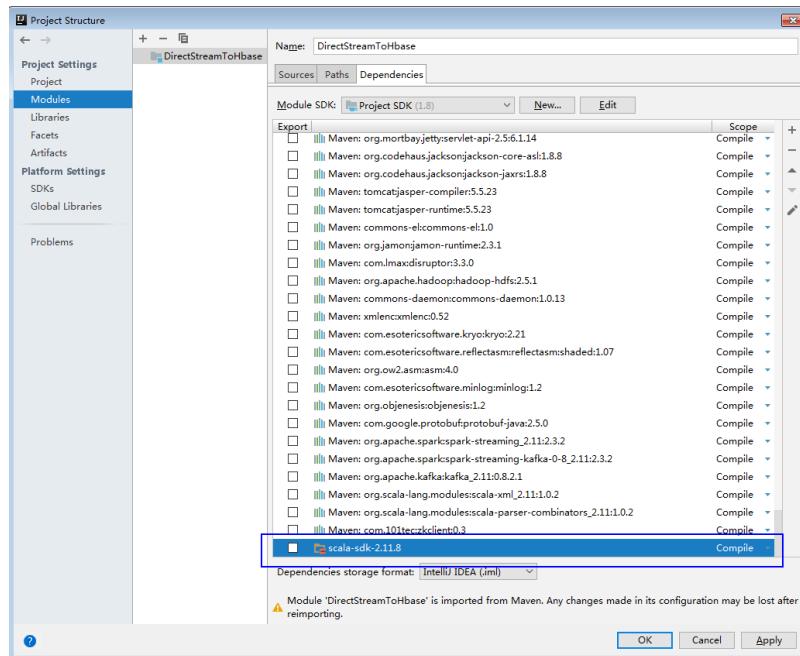
- c. On the **Scala SDK files** page, select the **scala sdk** directory, and then click **OK**.

Figure 1-289 Scala SDK files



5. Click **OK**.

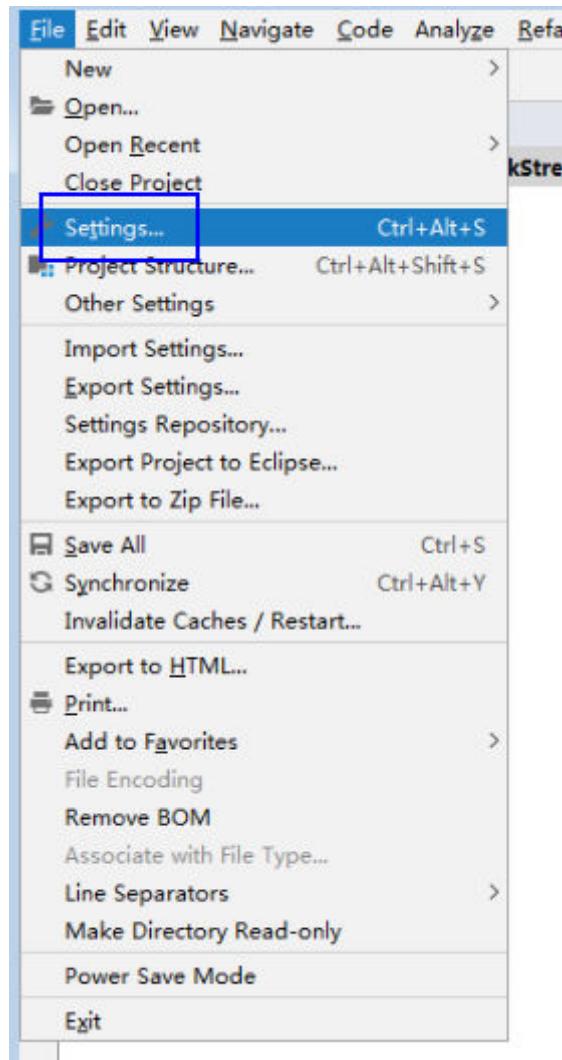
Figure 1-290 Successful configuration



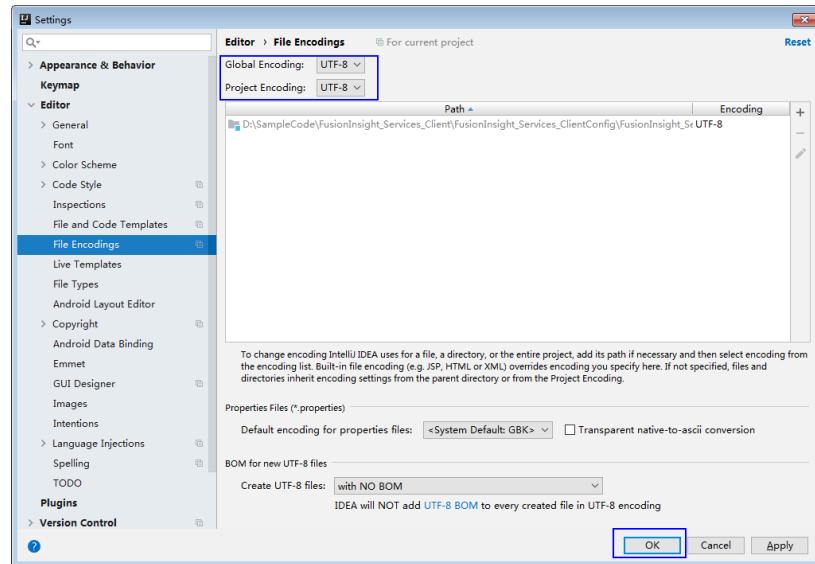
**Step 6** Set the file encoding of IDEA and solve the display of garble characters.

1. On the IDEA homepage, choose **File > Settings....**

Figure 1-291 Choosing Settings



2. Configure the encoding.
  - a. On the **Settings** page, choose **Editor > File Encodings**.
  - b. In the **Global Encoding** and **Project Encoding** drop-down lists, select **UTF-8**, respectively.
  - c. Click **Apply**.
  - d. Click **OK** to complete the encoding configuration.



----End

## Sample Code Path Description

**Table 1-177** Sample code path description

Sample Code Project	Sample Name	Sample Development Language
SparkJavaExample	Spark Core Project	Java
SparkScalaExample	Spark Core Project	Scala
SparkPythonExample	Spark Core Project	Python
SparkSQLJavaExample	Spark SQL Project	Java
SparkSQLScalaExample	Spark SQL Project	Scala
SparkSQLPythonExample	Spark SQL Project	Python
SparkThriftServerJavaExample	Accessing the Spark SQL Through JDBC	Java
SparkThriftServerScalaExample	Accessing the Spark SQL Through JDBC	Scala
SparkOnHbaseJavaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Java
SparkOnHbaseScalaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkOnHbasePythonExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Python
SparkOnHbaseJavaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Java
SparkOnHbaseScalaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Scala
SparkOnHbasePythonExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Python
SparkOnHbaseJavaExample-JavaHBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Java
SparkOnHbaseScalaExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Scala
SparkOnHbasePythonExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Java
SparkOnHbaseScalaExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Scala
SparkOnHbasePythonExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Java
SparkOnHbaseScalaExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Scala
SparkOnHbasePythonExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Java
SparkOnHbaseScalaExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Scala
SparkOnHbasePythonExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Python

Sample Code Project	Sample Name	Sample Development Language
SparkOnHbaseJavaExample- JavaHBaseForEachPartitionExam- ple	Spark on HBase-Using the foreachPartition Interface	Java
SparkOnHbaseScalaExample- HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Scala
SparkOnHbasePythonExample- HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Python
SparkOnHbaseJavaExample- JavaHBaseDistributedScanExam- ple	Spark on HBase- Distributedly Scanning HBase Tables	Java
SparkOnHbaseScalaExample- HBaseDistributedScanExample	Spark on HBase- Distributedly Scanning HBase Tables	Scala
SparkOnHbasePythonExample- HBaseDistributedScanExample	Spark on HBase- Distributedly Scanning HBase Tables	Python
SparkOnHbaseJavaExample- JavaHBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Java
SparkOnHbaseScalaExample- HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Scala
SparkOnHbasePythonExample- HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Python
SparkOnHbaseJavaExample- JavaHBaseStreamingBulkPutEx- ample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Java
SparkOnHbaseScalaExample- HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Scala
SparkOnHbasePythonExample- HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Python
SparkHbasetoHbaseJavaExample	Reading Data from HBase and Write It Back to HBase	Java
SparkHbasetoHbaseScalaExam- ple	Reading Data from HBase and Write It Back to HBase	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkHbasetoHbasePythonExample	Reading Data from HBase and Write It Back to HBase	Python
SparkHivetoHbaseJavaExample	Reading Data from Hive and Write It to HBase	Java
SparkHivetoHbaseScalaExample	Reading Data from Hive and Write It to HBase	Scala
SparkHivetoHbasePythonExample	Reading Data from Hive and Write It to HBase	Python
SparkStreamingKafka010JavaExample	Streaming Connecting to Kafka0-10	Java
SparkStreamingKafka010ScalaExample	Streaming Connecting to Kafka0-10	Scala
SparkStructuredStreamingJavaExample	Structured Streaming Project	Java
SparkStructuredStreamingScalaExample	Structured Streaming Project	Scala
SparkStructuredStreamingPythonExample	Structured Streaming Project	Python
StructuredStreamingADScalaExample	Structured Streaming Stream-Stream Join	Scala
SparkOnEsJavaExample	Spark on Elasticsearch	Java
SparkOnEsScalaExample	Spark on Elasticsearch	Scala
SparkOnEsPythonExample	Spark on Elasticsearch	Python
SparkOnSolrJavaExample	Spark on Solr	Java
SparkOnSolrScalaExample	Spark on Solr	Scala
SparkOnSolrPythonExample	Spark on Solr	Python
StructuredStreamingStateScalaExample	Structured Streaming Status Operation	Scala
SparkOnMultiHbaseScalaExample	Concurrent Access from Spark to HBase in Two Clusters	Scala
SparkHbasetoCarbonJavaExample	Synchronizing HBase Data from Spark to CarbonData	Java
SparkOnHudiJavaExample	Using Spark to Perform Basic Hudi Operations	Java

Sample Code Project	Sample Name	Sample Development Language
SparkOnHudiPythonExample	Using Spark to Perform Basic Hudi Operations	Python
SparkOnHudiScalaExample	Using Spark to Perform Basic Hudi Operations	Scala
SparkOnClickHouseJavaExample	Using Spark to Perform Basic ClickHouse Operations	Java
SparkOnClickHouseScalaExample	Using Spark to Perform Basic ClickHouse Operations	Scala
SparkOnClickHousePythonExample	Using Spark to Perform Basic ClickHouse Operations	Python

### 1.21.2.3 Creating a New Project (Optional)

#### Scenario

Besides importing Spark sample projects, the IDEA can be used to create a new Spark project. A Scala project is used as an example in the following steps.

#### Procedure

**Step 1** Start the IDEA and select **Create New Project**.

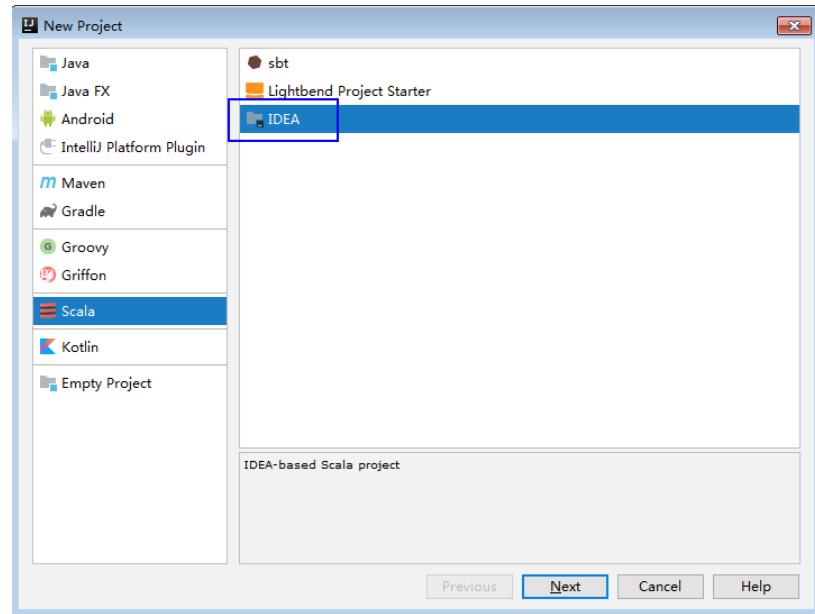
**Figure 1-292** Create a project



**Step 2** On the **New Project** page, select **Scala** as the development environment, select **IDEA**, and click **Next**.

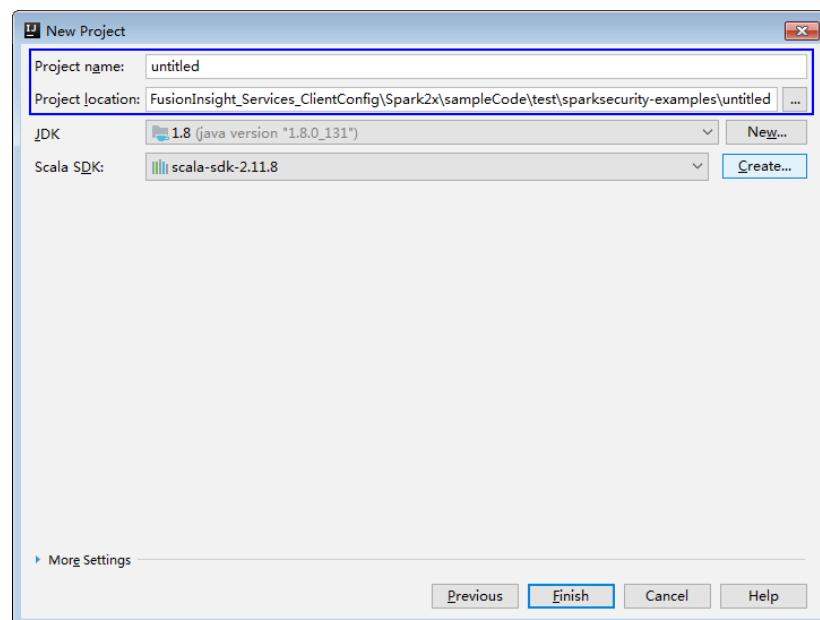
If a new project in Java is required, choose corresponding parameters.

**Figure 1-293** Select the development language



**Step 3** On the project information page, specify **Project name**, **Project location**, **Project JDK**, and **Scala SDK**, and click **Finish** to complete the project creation.

**Figure 1-294** Add the project information



----End

### 1.21.2.4 Preparing for Security Authentication

#### Scenario

In a safe cluster environment, the communication among components cannot be a simple communication. Components must be authorized by each other before the communication to ensure the security of the communication.

When users are developing the Spark application, the Spark is required to interwork with Hadoop and HBase in certain scenarios. Therefore, security authentication codes must be written into the Spark application to ensure that the Spark application can run properly.

Three security authentication modes are available:

- Authentication by running command lines:

Before submitting the Spark application for running or using the CLI to log in to the Spark SQL, run the following command in the Spark client to obtain authentication:

**kinit component service user**

- Authentication by configuring parameters

You can use any of the following methods to specify the security authentication information.

- Configure the **spark.kerberos.keytab** and **spark.kerberos.principal** parameters in the spark-defaults.conf file on the client.
- Add the following parameters to the **bin/spark-submit** command:  
**--conf spark.kerberos.keytab=<keytab file path> --conf spark.kerberos.principal=<Principal account>**
- Add the following parameter to the **bin/spark-submit** command:  
**--keytab <keytab file path> --principal <Principal account>**

- Authentication by adding codes:

Authenticate in the application by obtaining principal and keytab files of the client.

**Table 1-178** lists the authentication method for the sample code in security cluster environment.

**Table 1-178** Authentication methods

Sample Code	Mode	Authentication Method
sparknormal-examples	yarn-client	Command authentication, configuration authentication, or code authentication
	yarn-cluster	By running command lines or configuring parameters.
sparksecurity-examples (containing authentication code)	yarn-client	By adding code.
	yarn-cluster	Not supported.

 NOTE

- In the yarn cluster mode, the security authentication is not supported in the Spark projects. The security authentication needs to be completed before the application is started.
- For the safety authentication codes of the Python sample project that are not provided, configure the safety authentication parameter in the command that runs the application.

## Safety Security Code (Java)

The safety authentication of the sample codes is completed by invoking the LoginUtil class. For the secure login process, see the chapter Security Authentication Interfaces.

In the Spark sample project code, different sample projects use different authentication codes which are basic safety authentication and the basic safety authentication with the ZooKeeper authentication. The example authentication parameters used in the sample project are displayed as [Table 1-179](#). Modify the parameter value as required.

**Table 1-179** Parameter Description

Parameter	Example Parameter Value	Description
userPrincipal	sparkuser	User <b>Principal</b> for authentication. Use the account prepared in <a href="#">Preparing a Developer Account</a> .
userKeytabPath	/opt/FIclient/user.keytab	Users use the authenticated Keytab file, Copy the <b>user.keytab</b> file of the prepared developer account to the directory indicated by the example parameter value.
ZKServerPrincipal	zookeeper/hadoop.<system domain name>	The principal of the server in ZooKeeper. Contact the administrator to obtain the account.

The following code snippet belongs to the main method of the **FemaleInfoCollection** class in the **com.huawei.bigdata.spark.examples** package.

- Basic safety authentication:

Spark Core and Spark SQL programs needs only the basic safety authentication codes because both of them do not need to access the HBase or ZooKeeper. Add the following codes in the program, and configure the safety authentication parameter as required.

```
String userPrincipal = "sparkuser";
String userKeytabPath = "/opt/FIclient/user.keytab";
String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
```

```
Configuration hadoopConf = new Configuration();
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```

- Basic safety authentication with ZooKeeper Authentication:

Because the sample programs "Spark Streaming", "access Spark SQL with JDBC", and "Spark on HBase" require not only the basic safety authentication, but also the Principal of the ZooKeeper server to complete the safety authentication. Add the following codes in the program, and configure the safety authentication parameter as required.

```
String userPrincipal = "sparkuser";
String userKeytabPath = "/opt/FIclient/user.keytab";
String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
String ZKServerPrincipal = "zookeeper/hadoop.<System domain name >";

String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";
String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";

Configuration hadoopConf = new Configuration();
LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userPrincipal, userKeytabPath);
LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY, ZKServerPrincipal);
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```

## Safety Authentication Codes for the Communication of Spark and Zookeeper (Scala)

Currently the safety authentication of the sample codes is completed by invoking the LoginUtil class. For the secure login process, see the chapter about the unified authentication.

In the Spark sample project code, different sample projects use different authentication codes, which are basic safety authentication and basic safety authentication with ZooKeeper authentication. The example authentication parameters used in the sample project are displayed as [Table 1-180](#). Modify the parameter value as required.

**Table 1-180** Parameter Description

Parameter	Example Parameter Value	Description
userPrincipal	sparkuser	User <b>Principal</b> for authentication. Use the account prepared in <a href="#">Preparing a Developer Account</a> .
userKeytabPath	/opt/FIclient/user.keytab	Users use the authenticated Keytab file, Copy the <b>user.keytab</b> file of the prepared developer account to the directory indicated by the example parameter value.
ZKServerPrincipal	zookeeper/hadoop.<System domain name>	The principal of the server in ZooKeeper. Contact the administrator to obtain the account.

- Basic safety authentication:

Spark Core and Spark SQL programs needs only the basic safety authentication codes because both of them do not need to access the HBase or ZooKeeper. Add the following codes in the program, and configure the safety authentication parameter as required.

```
val userPrincipal = "sparkuser"  
val userKeytabPath = "/opt/FIclient/user.keytab"  
val krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf"  
val hadoopConf: Configuration = new Configuration()  
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```

- Basic safety authentication with ZooKeeper Authentication:

Because the sample programs "Spark Streaming", "access Spark SQL with JDBC", and "Spark on HBase" require not only the basic safety authentication, but also the Principal of the ZooKeeper server to complete the safety authentication. Add the following codes in the program, and configure the safety authentication parameter as required.

```
val userPrincipal = "sparkuser"  
val userKeytabPath = "/opt/FIclient/user.keytab"  
val krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf"  
val ZKServerPrincipal = "zookeeper/hadoop.<system domain name>"  
  
val ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME: String = "Client"  
val ZOOKEEPER_SERVER_PRINCIPAL_KEY: String = "zookeeper.server.principal"  
val hadoopConf: Configuration = new Configuration();  
LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, userPrincipal, userKeytabPath)  
LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY, ZKServerPrincipal)  
LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
```

## 1.21.2.5 Configuring the Python3 Sample Project

### Scenario

To run the Python3 interface sample code of the Spark component of MRS, perform the following operations.

### Procedure

- Step 1** Install Python3 of 3.7 or a higher version on the client.

The Python version can be viewed by running the **python3** command on the CLI of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)  
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux  
Type "help", "copyright", "credits" or "license" for more information.
```

- Step 2** Setuptools 47.3.1 must be installed on the client.

To obtain the software, visit the official websites.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```

- Step 3** Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency\_python3.6**, **dependency\_python3.7**, or **dependency\_python3.8** folder based on the Python3 version.
4. Run the **whereis easy\_install** command to find the path of the **easy\_install** program. If there are multiple paths, run the **easy\_install --version** command to select the **easy\_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy\_install**.
5. Run the **easy\_install** command to install the EGG files in the **dependency\_python3.x** folder in sequence. Example:  
`/usr/local/bin/easy_install future-0.18.2-py3.8.egg`  
If the following information is displayed, the EGG file is successfully installed.  
`Finished processing dependencies for future==0.18.2`

----End

## 1.21.3 Developing the Project

### 1.21.3.1 Spark Core Project

#### 1.21.3.1.1 Overview

##### Scenarios

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
```

```
CaiXuyu,female,50  
Liyuan,male,20  
CaiXuyu,female,50  
FangBo,female,50  
LiuYang,female,20  
YuanJing,male,10  
FangBo,female,50  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt**.
2. Create **/tmp/input** on HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:
  - a. On the HDFS client, run the following commands to obtain the safety authentication:  
**cd /opt/hadoopclient**  
**source bigdata\_env**  
**kinit <Service user for authentication>**
  - b. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a hdfs dfs command provides the same function.) to create a directory.
  - c. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the **hadoop fs -put input\_data1.txt /tmp/input** and **hadoop fs -put input\_data2.txt /tmp/input** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

- Before compilation and packaging, change the paths of the user.keytab and krb5.conf files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** and **/opt/female/krb5.conf**.
  - To run the Python sample code, you do not need to use Maven for packaging. You only need to upload the **user.keytab** and **krb5.conf** files to the server where the client is located.
3. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run the Scala and Java sample programs.  
**bin/spark-submit --class com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --deploy-mode client /opt/female/FemaleInfoCollection-1.0.jar <inputPath>**  
*<inputPath>* indicates the input path in HDFS.
- Run the Python sample program.

### NOTE

- The Python sample code does not provide authentication information. Configure **--keytab** and **--principal** to specify authentication information.  
**bin/spark-submit --master yarn --deploy-mode client --keytab /opt/Flclient/user.keytab --principal sparkuser /opt/female/SparkPythonExample/collectFemaleInfo.py <inputPath>**  
*<inputPath>* indicates the input path in HDFS.

### 1.21.3.1.2 Java Sample Code

## Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

## Sample Code

The following code segment is only an example. For details, see the `com.huawei.bigdata.spark.examples.FemaleInfoCollection` class.

```
// Create a configuration class SparkConf, and then create a SparkContext.  
SparkSession spark = SparkSession
```

```
.builder()
.appName("CollectFemaleInfo")
.config("spark.some.config.option", "some-value")
.getOrCreate();

// Read the source file data, and transfer each row of records to an element of the RDD.
JavaRDD<String> data = spark.read()
.textFile(args[0])
.javaRDD();

// Split each column of each record, and generate a Tuple.
JavaRDD<Tuple3<String, String, Integer>> person = data.map(new
Function<String, Tuple3<String, String, Integer>>()
{
    private static final long serialVersionUID = -2381522520231963249L;
    public Tuple3<String, String, Integer> call(String s) throws Exception
    {
        // Split a row of data by commas (,).
        String[] tokens = s.split(",");

        // Integrate the three split elements to a ternary Tuple.
        Tuple3<String, String, Integer> person = new Tuple3<String, String, Integer>(tokens[0], tokens[1],
Integer.parseInt(tokens[2]));
        return person;
    }
});

// Use the filter function to filter the data information about the time that female netizens spend online.
JavaRDD<Tuple3<String, String, Integer>> female = person.filter(new
Function<Tuple3<String, String, Integer>, Boolean>()
{
    private static final long serialVersionUID = -4210609503909770492L;

    public Boolean call(Tuple3<String, String, Integer> person) throws Exception
    {
        // Filter the records of which the gender in the second column is female.
        Boolean isFemale = person._2().equals("female");
        return isFemale;
    }
});

// Aggregate the total time that each female netizen spends online.
JavaPairRDD<String, Integer> females = female.mapToPair(new PairFunction<Tuple3<String, String,
Integer>, String, Integer>()
{
    private static final long serialVersionUID = 8313245377656164868L;

    public Tuple2<String, Integer> call(Tuple3<String, String, Integer> female) throws Exception
    {
        // Extract the two columns representing the name and online time for the sum of online time by
name during further operations.
        Tuple2<String, Integer> femaleAndTime = new Tuple2<String, Integer>(female._1(), female._3());
        return femaleAndTime;
    }
});
JavaPairRDD<String, Integer> femaleTime = females.reduceByKey(new Function2<Integer, Integer,
Integer>()
{
    private static final long serialVersionUID = -3271456048413349559L;

    public Integer call(Integer integer, Integer integer2) throws Exception
    {
        // Sum two online time durations of the same female netizen.
        return (integer + integer2);
    }
});

// Filter the information about female netizens who spend more than 2 hours online.
JavaPairRDD<String, Integer> rightFemales = females.filter(new Function<Tuple2<String, Integer>,
```

```
Boolean>()
{
    private static final long serialVersionUID = -3178168214712105171L;

    public Boolean call(Tuple2<String, Integer> s) throws Exception
    {
        // Extract the total time that female netizens spend online, and determine whether the time is
        more than 2 hours.
        if(s._2() > (2 * 60))
        {
            return true;
        }
        return false;
    }
});

// Print the information about female netizens who meet the requirements.
for(Tuple2<String, Integer> d: rightFemales.collect())
{
    System.out.println(d._1() + "," + d._2());
}
```

### 1.21.3.1.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see the com.huawei.bigdata.spark.examples.FemaleInfoCollection class.

Example: CollectMapper class

```
val spark = SparkSession
    .builder()
    .appName("CollectFemaleInfo")
    .config("spark.some.config.option", "some-value")
    .getOrCreate()

// Read data. This code indicates the data path that the input parameter args(0) specifies.
val text = spark.sparkContext.textFile(args(0))
// Filter the data information about the time that female netizens spend online.
val data = text.filter(_.contains("female"))
// Aggregate the time that each female netizen spends online.
val femaleData:RDD[(String,Int)] = data.map{line =>
    val t= line.split(',')
    (t(0),t(2).toInt)
}.reduceByKey(_ + _)

// Filter the information about female netizens who spend more than 2 hours online, and export the results.
val result = femaleData.filter(line => line._2 > 120)
result.collect().map(x => x._1 + ',' + x._2).foreach(println)
spark.stop()
```

### 1.21.3.1.4 Python Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

## Sample Code

The following code segment is only an example. For details, see [collectFemaleInfo.py](#).

```
def contains(str, substr):
    if substr in str:
        return True
    return False

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: CollectFemaleInfo <file>"
        exit(-1)

    spark = SparkSession \
        .builder \
        .appName("CollectFemaleInfo") \
        .getOrCreate()

    """
    The following programs are used to implement the following functions:
    1. Read data. This code indicates the data path that the input parameter argv[1] specifies. - text
    2. Filter data about the time that female netizens spend online. - filter
    3. Aggregate the total time that each female netizen spends online. - map/map/reduceByKey
    4. Filter information about female netizens who spend more than 2 hours online. - filter
    """

    inputPath = sys.argv[1]
    result = spark.read.text(inputPath).rdd.map(lambda r: r[0])\
        .filter(lambda line: contains(line, "female")) \
        .map(lambda line: line.split(',')) \
        .map(lambda dataArr: (dataArr[0], int(dataArr[2]))) \
        .reduceByKey(lambda v1, v2: v1 + v2) \
        .filter(lambda tupleVal: tupleVal[1] > 120) \
        .collect()
    for (k, v) in result:
        print k + "," + str(v)

    # Stop SparkContext.
    spark.stop()
```

### 1.21.3.2 Spark SQL Project

#### 1.21.3.2.1 Overview

##### Scenario

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
```

```
LiuYang,female,20  
YuanJing,male,10  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20  
YuanJing,male,10  
CaiXuyu,female,50  
FangBo,female,50  
GuoYijun,male,5  
CaiXuyu,female,50  
Liyuan,male,20  
CaiXuyu,female,50  
FangBo,female,50  
LiuYang,female,20  
YuanJing,male,10  
FangBo,female,50  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt**.
2. Create **/tmp/input** on the HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:
  - a. On the HDFS client, run the following commands to obtain the safety authentication:  
**cd /opt/hadoopclient**  
**source bigdata\_env**  
**kinit <component service user>**
  - b. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a hdfs dfs command provides the same function.) to create a directory.
  - c. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the **hadoop fs -put input\_data1.txt /tmp/input** and **hadoop fs -put input\_data2.txt /tmp/input** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Create a table and import the log files into the table.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** and **/opt/female/krb5.conf**.
  - To run the Python sample code, you do not need to use Maven for packaging. You only need to upload the **user.keytab** and **krb5.conf** files to the server where the client is located.
3. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run the Scala and Java sample programs.

### **bin/spark-submit --class**

```
com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --  
deploy-mode client /opt/female/SparkSqlScalaExample-1.0.jar <inputPath>  
<inputPath> indicates the input path in HDFS.
```

- Run the Python sample program

### NOTE

- The Python sample code does not provide authentication information. Configure **--keytab** and **--principal** to specify authentication information.

```
bin/spark-submit --master yarn --deploy-mode client --keytab /opt/  
FIclient/user.keytab --principal sparkuser /opt/female/  
SparkPythonExample/SparkSQLPythonExample.py <inputPath>  
<inputPath> indicates the input path in HDFS.
```

### 1.21.3.2.2 Java Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see com.huawei.bigdata.spark.examples.FemaleInfoCollection.

```
public static void main(String[] args) throws Exception {
    SparkSession spark = SparkSession
        .builder()
        .appName("CollectFemaleInfo")
        .config("spark.some.config.option", "some-value")
        .getOrCreate();

    // Convert RDD to DataFrame through the implicit conversion.
    JavaRDD<FemaleInfo> femaleInfoJavaRDD = spark.read().textFile(args[0]).javaRDD().map(
        new Function<String, FemaleInfo>() {
            @Override
            public FemaleInfo call(String line) throws Exception {
                String[] parts = line.split(",");
                FemaleInfo femaleInfo = new FemaleInfo();
                femaleInfo.setName(parts[0]);
                femaleInfo.setGender(parts[1]);
                femaleInfo.setStayTime(Integer.parseInt(parts[2].trim()));
                return femaleInfo;
            }
        });

    // Register table.
    Dataset<ROW> schemaFemaleInfo = spark.createDataFrame(femaleInfoJavaRDD,FemaleInfo.class);
    schemaFemaleInfo.registerTempTable("FemaleInfoTable");

    // Run SQL query
    Dataset<ROW> femaleTimeInfo = spark.sql("select * from " +
        "(select name,sum(stayTime) as totalStayTime from FemaleInfoTable " +
        "where gender = 'female' group by name )" +
        " tmp where totalStayTime >120");

    // Collect the columns of a row in the result.
    List<String> result = femaleTimeInfo.javaRDD().map(new Function<Row, String>() {
        public String call(Row row) {
            return row.getString(0) + "," + row.getLong(1);
        }
    }).collect();
    System.out.println(result);
    spark.stop();
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.3.1/sql-programming-guide.html#running-sql-queries-programmatically>.

### 1.21.3.2.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

## Sample Code

The following code segment is only an example. For details, see com.huawei.bigdata.spark.examples.FemaleInfoCollection.

```
object FemaleInfoCollection
{
    //Table structure, used for mapping the text data to df
    case class FemaleInfo(name: String, gender: String, stayTime: Int)
    def main(args: Array[String]) {
        //Configure Spark application name
        val spark = SparkSession
            .builder()
            .appName("FemaleInfo")
            .config("spark.some.config.option", "some-value")
            .getOrCreate()
        import spark.implicits._

        //Convert RDD to DataFrame through the implicit conversion, then register table.
        spark.sparkContext.textFile(args(0)).map(_.split(","))
            .map(p => FemaleInfo(p(0), p(1), p(2).trim.toInt))
            .toDF.registerTempTable("FemaleInfoTable")

        //Via SQL statements to screen out the time information of female stay on the Internet , and aggregated
        //the same names.
        val femaleTimeInfo = spark.sql("select name,sum(stayTime) as stayTime from FemaleInfoTable where
        gender = 'female' group by name")
        //Filter information about female netizens who spend more than 2 hours online.
        val c = femaleTimeInfo.filter("stayTime >= 120").collect().foreach(println)
        spark.stop()
    }
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.3.1/sql-programming-guide.html#running-sql-queries-programmatically>.

### 1.21.3.2.4 Python Sample Code

#### Function

Collect information about female netizens who have spent more than 2 hours in online shopping on the weekend.

## Sample Code

The following code segment is only an example. For details, see SparkSQLPythonExample.

```
# -*- coding:utf-8 -*-
import sys
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

def contains(str1, substr1):
    if substr1 in str1:
        return True
    return False

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: SparkSQLPythonExample.py <file>"
```

```
exit(-1)

# Initialize the SparkSession and SQLContext.
sc = SparkSession.builder.appName("CollectFemaleInfo").getOrCreate()
sqlCtx = SQLContext(sc)

#Convert RDD to DataFrame.
inputPath = sys.argv[1]
inputRDD = sc.read.text(inputPath).rdd.map(lambda r: r[0])\
    .map(lambda line: line.split(","))\
    .map(lambda dataArr: (dataArr[0], dataArr[1], int(dataArr[2])))\
    .collect()
df = sqlCtx.createDataFrame(inputRDD)

# Register a table.
df.registerTempTable("FemaleInfoTable")

# Run SQL query statements and display the result.
FemaleTimeInfo = sqlCtx.sql("SELECT * FROM " +
    "(SELECT _1 AS Name,SUM(_3) AS totalStayTime FROM FemaleInfoTable " +
    "WHERE _2 = 'female' GROUP BY _1 )" +
    " WHERE totalStayTime >120").show()

sc.stop()
```

### 1.21.3.3 Accessing the Spark SQL Through JDBC

### **1.21.3.3.1 Overview**

## Scenarios

Users customize JDBCServer clients and use JDBC connections to create, load data to, query, and delete data tables.



Spark allows you to add extension identifiers to JDBC connection strings. These extension identifiers are printed in JDBCServer audit logs to distinguish SQL sources. To add extension identifiers, add the following content to the end of the connection strings in the **`$SPARK_HOME/bin/spark-beeline`** script in the client directory:

**auditAddition=xxx**

```
#!/bin/bash
# Set the current directory as the working directory
# CURRENT_PATH=$PWD
CURRENT_PATH=$(pwd)
echo "It's running the $1 spark-script, it calls $CURRENT_PATH/mainline
and helps to connect to the $2(hadoop namenode)!"
```

*xxx* is the custom identifier. The identifier can contain a maximum of 256 bytes and only letters, digits, underscores (\_), commas (,), and colons (:) are allowed.

After the SQL statement is executed, the configured extension identifier is printed in the HiveServer audit log `/var/log/Bigdata/audit/spark/jdbcserver/jdbcserver-audit.log`. The following is an example.

## Data Preparation

**Step 1** Ensure that the JDBCServer service has been started in multi-active instance HA mode and at least one instance provides connections for client. Create the **/home/data** file on every available instance node of the JDBCServer.

Miranda,32  
Karlie,23  
Candice,27

**Step 2** Ensure that the user whose starts the JDBCServer has the read and write permission on the file.

**Step 3** Ensure that the **hive-site.xml** file exists in **classpath**, and set parameters required for the client connection. For details about parameters required for the JDBCServer, see [JDBCServer Interface](#).

----End

## Development Idea

1. Create a child table in the default database.
2. Add data in **/home/data** to the child table.
3. Query data in the child table.
4. Delete the child table.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Upload the **krb5.conf** and **user.keytab** files to the node where the client is located.
- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** and **/opt/female/krb5.conf**.

- Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the **java -cp** command to run the code (the class name and file name must be the same as those in the actual code. The following is only an example).

- Run the Java sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/
conf:/opt/female/SparkThriftServerJavaExample-1.0.jar
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/
conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

- Run the Scala sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/
conf:/opt/female/SparkThriftServerExample-1.0.jar
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/
conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

#### NOTE

After the SSL feature of ZooKeeper is enabled for the cluster (check the `ssl.enabled` parameter of the ZooKeeper service), add the `-Dzookeeper.client.secure=true -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty` parameter to the command:

```
java -Dzookeeper.client.secure=true -
Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -cp
$SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/conf:/opt/female/
SparkThriftServerJavaExample-1.0.jar
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/conf/hive-
site.xml $SPARK_HOME/conf/spark-defaults.conf
```

### 1.21.3.3.2 Java Sample Code

## Function

The JDBC interface of the user-defined client is used to submit the data analysis task and return the results.

## Sample Code

**Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). For example,

```
ArrayList<String> sqlList = new ArrayList<String>();
sqlList.add("CREATE TABLE CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',';");
sqlList.add("LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD");
sqlList.add("SELECT * FROM child");
sqlList.add("DROP TABLE child");
executeSql(url, sqlList);
```

#### NOTE

The data file in the sample project must be placed into the Home directory of the host where the JDBCServer is located.

**Step 2** Assemble the JDBC URL.

```
String securityConfig = ";saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<system domain
name>@<system domain name>;user.principal=sparkuser;user.keytab=/opt/FIclient/user.keytab;";
Configuration config = new Configuration();
config.addResource(new Path(args[0]));
String zkUrl = config.get("spark.deploy.zookeeper.url");
String auditAddition = "auditAddition=sparktest1111test";
```

```
String zkNamespace = null;
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace");
if (zkNamespace != null) {
```

```
//Remove redundant characters from configuration items
zkNamespace = zkNamespace.substring(1);
}

StringBuilder sb = new StringBuilder("jdbc:hive2://" +
+ zkUrl +
+ ";serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=" +
+ zkNamespace +
+ securityConfig +
+ auditAddition);
String url = sb.toString();
```

#### NOTE

The default validity period of KERBEROS authentication is one day. After the validity period, the authentication needs to be performed again if you want to connect the client and JDBCServer. You can add the user.principal and user.keytab authentication information to **url** to ensure that the authentication is performed each time the connection is established. For example, add **user.principal=sparkuser;user.keytab=/opt/hadoopclient/user.keytab** to **url**.

**Step 3** Load the Hive JDBC driver.

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

**Step 4** Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The **zk.quorum** in the connection string can be replaced by **spark.deploy.zookeeper.url** in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server. The method is as follows:

Before executing the DriverManager.getConnection script to obtain the JDBC connection, add the DriverManager.setLoginTimeout(n) script to configure the timeout. n indicates the timeout length of waiting for the return from the server. The unit is second, the type is Int, and the default value is 0 (indicating never timing out).

```
static void executeSql(String url, ArrayList<String> sqls) throws ClassNotFoundException, SQLException {
    try {
        Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
    } catch (Exception e) {
        e.printStackTrace();
    }
    Connection connection = null;
    PreparedStatement statement = null;

    try {
        connection = DriverManager.getConnection(url);
        for (int i = 0; i < sqls.size(); i++) {
            String sql = sqls.get(i);
            System.out.println("---- Begin executing sql: " + sql + " ----");
            statement = connection.prepareStatement(sql);
            ResultSet result = statement.executeQuery();
            ResultSetMetaData metaData = result.getMetaData();
            Integer colNum = metaData.getColumnCount();
            for (int j = 1; j <= colNum; j++) {
                System.out.println(metaData.getColumnName(j) + "\t");
            }
            System.out.println();

            while (result.next()) {
                for (int j = 1; j <= colNum; j++) {
                    System.out.println(result.getString(j) + "\t");
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
        }
        System.out.println();
    }
    System.out.println("---- Done executing sql: " + sql + " ----");
}

} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (null != statement) {
        statement.close();
    }
    if (null != connection) {
        connection.close();
    }
}
}
```

----End

### 1.21.3.3.3 Scala Sample Code

#### Function

The JDBC interface of the user-defined client is used to submit the data analysis task and return the results.

#### Sample Code

- Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). For example,

```
val sqlList = new ArrayBuffer[String]
sqlList += "CREATE TABLE CHILD (NAME STRING, AGE INT) " +
"ROW FORMAT DELIMITED FIELDS TERMINATED BY ','"
sqlList += "LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD"
sqlList += "SELECT * FROM child"
sqlList += "DROP TABLE child"
```



The data file in the sample project must be placed into the Home directory of the host where the JDBCServer is located.

- Step 2** Assemble the JDBC URL.

```
val securityConfig = ";saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<system domain name>@<system domain name>" + ";"
val config: Configuration = new Configuration()
config.addResource(new Path(args(0)))
val zkUrl = config.get("spark.deploy.zookeeper.url")
val auditAddtion = "auditAddtion=sparktest1111test;"

var zkNamespace: String = null
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace")
//Remove redundant characters from configuration items
if (zkNamespace != null) zkNamespace = zkNamespace.substring(1)
val sb = new StringBuilder("jdbc:hive2://"
+ zkUrl
+ ";serviceDiscoveryMode=zooKeeper;zooKeeperNamespace="
+ zkNamespace
+ securityConfig
+ auditAddtion)
val url = sb.toString()
```

 NOTE

The default validity period of KERBEROS authentication is one day. After the validity period, the authentication needs to be performed again if you want to connect the client and JDBCServer. You can add the user.principal and user.keytab authentication information to **url** to ensure that the authentication is performed each time the connection is established. For example, add **user.principal=sparkuser;user.keytab=/opt/hadoopclient/user.keytab** to **url**.

- Step 3** Load the Hive JDBC driver. Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The **zk.quorum** in the connection string can be replaced by **spark.deploy.zookeeper.url** in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server. The method is as follows:

Before executing the DriverManager.getConnection script to obtain the JDBC connection, add the DriverManager.setLoginTimeout(n) script to configure the timeout. n indicates the timeout length of waiting for the return from the server. The unit is second, the type is Int, and the default value is 0 (indicating never timing out).

```
def executeSql(url: String, sqls: Array[String]): Unit = {
    //Load the Hive JDBC driver.
    Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance()

    var connection: Connection = null
    var statement: PreparedStatement = null
    try {
        connection = DriverManager.getConnection(url)
        for (sql <- sqls) {
            println(s"---- Begin executing sql: $sql ----")
            statement = connection.prepareStatement(sql)

            val result = statement.executeQuery()

            val resultMetaData = result.getMetaData
            val colNum = resultMetaData.getColumnCount
            for (i <- 1 to colNum) {
                print(resultMetaData.getColumnName(i) + "\t")
            }
            println()

            while (result.next()) {
                for (i <- 1 to colNum) {
                    print(result.getString(i) + "\t")
                }
                println()
            }
            println(s"---- Done executing sql: $sql ----")
        }
    } finally {
        if (null != statement) {
            statement.close()
        }
        if (null != connection) {
            connection.close()
        }
    }
}
```

```
}
```

----End

### 1.21.3.4 Spark on HBase

#### 1.21.3.4.1 Performing Operations on Data in Avro Format

##### Scenario

Users can use HBase as data sources in Spark applications. In this example, data is stored in HBase in Avro format. Data is read from the HBase, and the read data is filtered.

##### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'ExampleAvrotable','rowkey','cf1'  
create 'ExampleAvrotableInsert','rowkey','cf1'
```

##### Development Guideline

1. Create an RDD.
2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and performs simple operations on the data.

##### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

##### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

 NOTE

To run the Spark on HBase example program, set `spark.yarn.security.credentials.hbase.enabled` (`false` by default) in the `spark-defaults.conf` file on the Spark client to `true`. Changing the `spark.yarn.security.credentials.hbase.enabled` value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to `false`.) Set the value of the configuration item `spark.inputFormat.cache.enabled` to `false`.

## Submitting Commands

Assume that the JAR package name of the case code is `spark-hbaseContext-test-1.0.jar` that is stored in the `$SPARK_HOME` directory on the client. Run the following commands in the `$SPARK_HOME` directory.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is `SparkOnHbaseJavaExample.jar` and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar AvroSource.py
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource --files /opt/user.keytab,/opt/krb5.conf SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is `SparkOnHbaseJavaExample.jar` and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/user.keytab,/opt/krb5.conf --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar AvroSource.py
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `AvroSource` file in `SparkOnHbaseJavaExample`.

```

public static void main(JavaSparkContext jsc) throws IOException {
    LoginUtil.loginWithUserKeytab();
    SQLContext sqlContext = new SQLContext(jsc);
    Configuration hbaseconf = new HBaseConfiguration().create();
    JavaHBaseContext hBaseContext = new JavaHBaseContext(jsc, hbaseconf);
    List list = new ArrayList<AvroHBaseRecord>();
    for(int i=0; i<=255 ; ++i){
        list.add(AvroHBaseRecord.apply(i));
    }
    try{
        Map<String, String> map = new HashMap<String, String>();
        map.put(HBaseTableCatalog.tableCatalog(), catalog);
        map.put(HBaseTableCatalog.newTable(), "5");
        sqlContext.createDataFrame(list,
        AvroHBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
        Dataset<Row> ds = withCatalog(sqlContext,catalog);
        ds.show();
        ds.printSchema();
        ds.registerTempTable("ExampleAvrotable");
        Dataset<Row> c= sqlContext.sql("select count(1) from ExampleAvrotable");
        c.show();
        Dataset<Row> filtered = ds.select("col0", "col1.favorite_array").where("col0 = 'name1'");
        filtered.show();
        java.util.List<Row> collected = filtered.collectAsList();
        if (collected.get(0).get(1).toString().equals("number1")) {
            throw new UserCustomizedSampleException("value invalid", new Throwable());
        }
        if (collected.get(0).get(1).toString().equals("number2")) {
            throw new UserCustomizedSampleException("value invalid", new Throwable());
        }
        Map avroCatalogInsertMap = new HashMap<String, String>();
        avroCatalogInsertMap.put("avroSchema" , AvroHBaseRecord.schemaString);
        avroCatalogInsertMap.put(HBaseTableCatalog.tableCatalog(), avroCatalogInsert);
        ds.write().options(avroCatalogInsertMap).format("org.apache.hadoop.hbase.spark").save();
        Dataset<Row> newDS = withCatalog(sqlContext,avroCatalogInsert);
        newDS.show();
        newDS.printSchema();
        if (newDS.count() != 256) {
            throw new UserCustomizedSampleException("value invalid", new Throwable());
        }
        ds.filter("col1.name = 'name5' || col1.name <= 'name5'").select("col0","col1.favorite_color",
        "col1.favorite_number").show();
    } finally{
        jsc.stop();
    }
}

```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the [AvroSource file in SparkOnHbaseScalaExample](#).

```

def main(args: Array[String]) {
    LoginUtil.loginWithUserKeytab()
    val sparkConf = new SparkConf().setAppName("AvroSourceExample")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)
    val hbaseConf = HBaseConfiguration.create()
    val hBaseContext = new HBaseContext(sc, hbaseConf)
    import sqlContext.implicits_
    def withCatalog(cat: String): DataFrame = {
        sqlContext
            .read
            .options(Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
        avroCatalog))
            .format("org.apache.hadoop.hbase.spark")
            .load()
    }
    val data = (0 to 255).map { i =>

```

```
    AvroHBaseRecord(i)
}
try {
    sc.parallelize(data).toDF.write.options(
        Map(HBaseTableCatalog.tableCatalog -> catalog, HBaseTableCatalog.newTable -> "5"))
        .format("org.apache.hadoop.hbase.spark")
        .save()

    val df = withCatalog(catalog)
    df.show()
    df.printSchema()
    df.registerTempTable("ExampleAvrotable")
    val c = sqlContext.sql("select count(1) from ExampleAvrotable")
    c.show()

    val filtered = df.select($"col0", $"col1.favorite_array").where($"col0" === "name001")
    filtered.show()
    val collected = filtered.collect()
    if (collected(0).getSeq[String](1)(0) != "number1") {
        throw new UserCustomizedSampleException("value invalid")
    }
    if (collected(0).getSeq[String](1)(1) != "number2") {
        throw new UserCustomizedSampleException("value invalid")
    }

    df.write.options(
        Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
avroCatalogInsert,
        HBaseTableCatalog.newTable -> "5"))
        .format("org.apache.hadoop.hbase.spark")
        .save()
    val newDF = withCatalog(avroCatalogInsert)
    newDF.show()
    newDF.printSchema()
    if (newDF.count() != 256) {
        throw new UserCustomizedSampleException("value invalid")
    }
    df.filter($"col1.name" === "name005" || $"col1.name" <= "name005")
        .select("col0", "col1.favorite_color", "col1.favorite_number")
        .show()
} finally {
    sc.stop()
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `AvroSource` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide Hbase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
    Set spark.yarn.security.credentials.hbase.enabled to true.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("AvroSourceExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.AvroSource')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
```

```
spark._jvm.AvroSource().execute(spark._jsc)
# Stop the SparkSession instance.
spark.stop()
```

### 1.21.3.4.2 Performing Operations on the HBase Data Source

#### Scenario

Users can use HBase as data sources in Spark applications, write DataFrame to HBase, read data from HBase, and filter the read data.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'HBaseSourceExampleTable','rowkey','cf1','cf2','cf3','cf4','cf5','cf6','cf7', 'cf8'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and performs simple operations on the data.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

##### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name of the case code is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. Run the following commands in the **\$SPARK\_HOME** directory.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource --files /opt/user.keytab,/opt/krb5.conf SparkOnHbaseJavaExample.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --files /opt/user.keytab,/opt/krb5.conf --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseSource** file in **SparkOnHbaseJavaExample**.

```
public static void main(String args[]) throws IOException{
    LoginUtil.loginWithUserKeytab();
    SparkConf sparkConf = new SparkConf().setAppName("HBaseSourceExample");
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    SQLContext sqlContext = new SQLContext(jsc);

    Configuration conf = HBaseConfiguration.create();
    JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc,conf);
    try{
        List<HBaseRecord> list = new ArrayList<HBaseRecord>();
        for(int i=0 ; i<256; i++){
            list.add(new HBaseRecord());
        }
        hbaseContext.write(list);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
        list.add(new HBaseRecord(i));
    }
    Map map = new HashMap<String, String>();
    map.put(HBaseTableCatalog.tableCatalog(), cat);
    map.put(HBaseTableCatalog.newTable(), "5");
    System.out.println("Before insert data into hbase table");
    sqlContext.createDataFrame(list,
HBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
    Dataset<Row> ds = withCatalog(sqlContext, cat);
    System.out.println("After insert data into hbase table");
    ds.printSchema();
    ds.show();
    ds.filter("key <= 'row5'").select("key", "col1").show();
    ds.registerTempTable("table1");
    Dataset<Row> tempDS = sqlContext.sql("select count(col1) from table1 where key < 'row5'");
    tempDS.show();
} finally {
    jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    LoginUtil.loginWithUserKeytab()
    val sparkConf = new SparkConf().setAppName("HBaseSourceExample")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)
    val conf = HBaseConfiguration.create()
    val hbaseContext = new HBaseContext(sc, conf)
    import sqlContext.implicits._

    def withCatalog(cat: String): DataFrame = {
        sqlContext
            .read
            .options(Map(HBaseTableCatalog.tableCatalog -> cat))
            .format("org.apache.hadoop.hbase.spark")
            .load()
    }

    val data = (0 to 255).map { i =>
        HBaseRecord(i)
    }

    try {
        sc.parallelize(data).toDF.write.options(
            Map(HBaseTableCatalog.tableCatalog -> cat, HBaseTableCatalog.newTable -> "5"))
            .format("org.apache.hadoop.hbase.spark")
            .save()

        val df = withCatalog(cat)
        df.show()
        df.filter($"col0" <= "row005")
            .select($"col0", $"col1").show()
        df.registerTempTable("table1")
        val c = sqlContext.sql("select count(col1) from table1 where col0 < 'row050'")
        c.show()
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""

```

[Note]

(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to implement the required operations.

(2) If yarn-client is used, ensure that the **spark.yarn.security.credentials.hbase.enabled** parameter in the **spark-defaults.conf** file under **Spark/spark/conf/** is set to **true** on the Spark client.

Set **spark.yarn.security.credentials.hbase.enabled** to **true**.

.....

```
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("HBaseSourceExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.HBaseSource')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.HBaseSource().execute(spark._jsc)
# Stop the SparkSession instance.
spark.stop()
```

#### 1.21.3.4.3 Using the BulkPut Interface

##### Scenario

Users can use the HBaseContext method to use HBase in Spark applications and write the constructed RDD into HBase.

##### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'bulktable','cf1'
```

##### Development Guideline

1. Create an RDD.
2. Perform operations on HBase in HBaseContext mode and write the generated RDD into the HBase table.

##### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

##### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hBaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory. Java is displayed before the class name of the Java API. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa  
mple SparkOnHbaseJavaExample.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseBulkPutExample.py bulktable cf1
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa  
mple --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseBulkPutExample.py bulktable cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **JavaHBaseBulkPutExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws Exception{  
    if (args.length < 2) {
```

```
System.out.println("JavaHBaseBulkPutExample " +
    "{tableName} {columnFamily}");
return;
}
LoginUtil.loginWithUserKeytab();
String tableName = args[0];
String columnFamily = args[1];
SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkPutExample " + tableName);
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
try {
    List<String> list = new ArrayList<String>(5);
    list.add("1," + columnFamily + ",1,1");
    list.add("2," + columnFamily + ",1,2");
    list.add("3," + columnFamily + ",1,3");
    list.add("4," + columnFamily + ",1,4");
    list.add("5," + columnFamily + ",1,5");
    list.add("6," + columnFamily + ",1,6");
    list.add("7," + columnFamily + ",1,7");
    list.add("8," + columnFamily + ",1,8");
    list.add("9," + columnFamily + ",1,9");
    list.add("10," + columnFamily + ",1,10");
    JavaRDD<String> rdd = jsc.parallelize(list);
    Configuration conf = HBaseConfiguration.create();
    JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
    hbaseContext.bulkPut(rdd,
        TableName.valueOf(tableName),
        new PutFunction());
    System.out.println("Bulk put into Hbase successfully!");
} finally {
    jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkPutExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    if (args.length < 2) {
        System.out.println("HBaseBulkPutTimestampExample {tableName} {columnFamily} are missing an argument")
        return
    }
    LoginUtil.loginWithUserKeytab()
    val tableName = args(0)
    val columnFamily = args(1)
    val sparkConf = new SparkConf().setAppName("HBaseBulkPutTimestampExample " +
        tableName + " " + columnFamily)
    val sc = new SparkContext(sparkConf)
    try {
        val rdd = sc.parallelize(Array(
            (Bytes.toBytes("1"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1")))),
            (Bytes.toBytes("2"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2")))),
            (Bytes.toBytes("3"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3")))),
            (Bytes.toBytes("4"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4")))),
            (Bytes.toBytes("5"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5")))),
            (Bytes.toBytes("6"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("6")))),
            (Bytes.toBytes("7"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("7")))),
            (Bytes.toBytes("8"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("8")))),
            (Bytes.toBytes("9"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("9"))))
        ))
    }
}
```

```
        Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("9")))),  
        (Bytes.toBytes("10"),  
         Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("10")))))  
    val conf = HBaseConfiguration.create()  
    val timeStamp = System.currentTimeMillis()  
    val hbaseContext = new HBaseContext(sc, conf)  
    hbaseContext.bulkPut([(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])](rdd,  
        TableName.valueOf(tableName),  
        (putRecord) => {  
            val put = new Put(putRecord._1)  
            putRecord._2.foreach((putValue) => put.addColumn(putValue._1, putValue._2,  
                timeStamp, putValue._3))  
            put  
        })  
    } finally {  
        sc.stop()  
    }  
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkPutExample` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-  
*****  
[Note]  
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to implement required operations.  
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.  
    Set spark.yarn.security.credentials.hbase.enabled to true.  
*****  
from py4j.java_gateway import java_import  
from pyspark.sql import SparkSession  
# Create a SparkSession instance.  
spark = SparkSession\  
    .builder\  
    .appName("JavaHBaseBulkPutExample")\  
    .getOrCreate()  
# Import the required class to sc._jvm.  
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExample')  
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.  
spark._jvm.JavaHBaseBulkPutExample().execute(spark._jsc, sys.argv)  
# Stop the SparkSession instance.  
spark.stop()
```

### 1.21.3.4.4 Using the BulkGet Interface

#### Scenario

Users can use the `HBaseContext` method to use HBase in Spark applications, construct the rowkey of the data to be obtained into RDDs, and obtain the data corresponding to the rowkey in the HBase tables through the BulkGet interface of `HBaseContext`.

#### Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#).

## Development Guideline

- Creates RDDs containing the rowkey to be obtained.
- Perform operations on HBase in HBaseContext mode and obtain data corresponding to rowkey in HBase tables through the BulkGet interface of HBaseContext.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExa  
mple SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseBulkGetExample.py bulktable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExa  
mple --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseBulkGetExample.py bulktable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseBulkGetExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws IOException{  
    if (args.length < 1) {  
        System.out.println("JavaHBaseBulkGetExample {tableName}");  
        return;  
    }  
    LoginUtil.loginWithUserKeytab();  
    String tableName = args[0];  
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);  
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
    try {  
        List<byte[]> list = new ArrayList<byte[]>(5);  
        list.add(Bytes.toBytes("1"));  
        list.add(Bytes.toBytes("2"));  
        list.add(Bytes.toBytes("3"));  
        list.add(Bytes.toBytes("4"));  
        list.add(Bytes.toBytes("5"));  
        JavaRDD<byte[]> rdd = jsc.parallelize(list);  
        Configuration conf = HBaseConfiguration.create();  
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);  
        List resultList = hbaseContext.bulkGet(TableName.valueOf(tableName), 2, rdd, new GetFunction(),  
            new ResultFunction()).collect();  
        for(int i=0 ;<resultList.size();i++){  
            System.out.println(resultList.get(i));  
        }  
    } finally {  
        jsc.stop();  
    }  
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseBulkGetExample** file in **SparkOnHbaseScalaExample**.

```
def main(args: Array[String]) {  
    if (args.length < 1) {  
        println("HBaseBulkGetExample {tableName} missing an argument")  
        return  
    }  
    LoginUtil.loginWithUserKeytab()  
    val tableName = args(0)
```

```
val sparkConf = new SparkConf().setAppName("HBaseBulkGetExample " + tableName)
val sc = new SparkContext(sparkConf)
try {
    //[(Array[Byte])]
    val rdd = sc.parallelize(Array(
        Bytes.toBytes("1"),
        Bytes.toBytes("2"),
        Bytes.toBytes("3"),
        Bytes.toBytes("4"),
        Bytes.toBytes("5"),
        Bytes.toBytes("6"),
        Bytes.toBytes("7")))
    val conf = HBaseConfiguration.create()
    val hbaseContext = new HBaseContext(sc, conf)
    val getRdd = hbaseContext.bulkGet[Array[Byte], String](
        TableName.valueOf(tableName),
        2,
        rdd,
        record => {
            System.out.println("making Get")
            new Get(record)
        },
        (result: Result) => {
            val it = result.listCells().iterator()
            val b = new StringBuilder
            b.append(Bytes.toString(result.getRow) + ":")
            while (it.hasNext) {
                val cell = it.next()
                val q = Bytes.toString(CellUtil.cloneQualifier(cell))
                if (q.equals("counter")) {
                    b.append("(" + q + "," + Bytes.toLong(CellUtil.cloneValue(cell)) + ")")
                } else {
                    b.append("(" + q + "," + Bytes.toString(CellUtil.cloneValue(cell)) + ")")
                }
            }
            b.toString()
        })
        getRdd.collect().foreach(v => println(v))
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkGetExample` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
    Set spark.yarn.security.credentials.hbase.enabled to true.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseBulkGetExample")\
    .getOrCreate()
# Import required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkGetExample().execute(spark._jsc, sys.argv)
```

```
# Stop the SparkSession instance.  
spark.stop()
```

#### 1.21.3.4.5 Using the BulkDelete Interface

##### Scenario

Users can use the HBaseContext method to use HBase in Spark applications, construct rowkey of the data to be deleted into RDDs, and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of HBaseContext.

##### Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#).

##### Development Guideline

1. Create RDDs containing the rowkey to be deleted.
2. Perform operations on the HBase in HBaseContext mode and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of HBaseContext.

##### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

##### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

###### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

##### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are

executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE  
xample SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseBulkDeleteExample.py bulktable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE  
xample --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseBulkDeleteExample.py bulktable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseBulkDeleteExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws IOException {  
    if (args.length < 1) {  
        System.out.println("JavaHBaseBulkDeleteExample {tableName}");  
        return;  
    }  
    LoginUtil.loginWithUserKeytab();  
    String tableName = args[0];  
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkDeleteExample " + tableName);  
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
    try {  
        List<byte[]> list = new ArrayList<byte[]>(5);  
        list.add(Bytes.toBytes("1"));  
        list.add(Bytes.toBytes("2"));  
        list.add(Bytes.toBytes("3"));  
        list.add(Bytes.toBytes("4"));  
        list.add(Bytes.toBytes("5"));  
        JavaRDD<byte[]> rdd = jsc.parallelize(list);  
        Configuration conf = HBaseConfiguration.create();  
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);  
        hbaseContext.bulkDelete(rdd,  
                               TableName.valueOf(tableName), new DeleteFunction(), 4);  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        jsc.stop();  
    }  
}
```

```
        System.out.println("Bulk Delete successfully!");
    } finally {
        jsc.stop();
    }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkDeleteExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
    if (args.length < 1) {
        println("HBaseBulkDeleteExample {tableName} missing an argument")
        return
    }
    LoginUtil.loginWithUserKeytab()
    val tableName = args(0)
    val sparkConf = new SparkConf().setAppName("HBaseBulkDeleteExample " + tableName)
    val sc = new SparkContext(sparkConf)
    try {
        //[[Array[Byte]]]
        val rdd = sc.parallelize(Array(
            Bytes.toBytes("1"),
            Bytes.toBytes("2"),
            Bytes.toBytes("3"),
            Bytes.toBytes("4"),
            Bytes.toBytes("5")
        ))
        val conf = HBaseConfiguration.create()
        val hbaseContext = new HBaseContext(sc, conf)
        hbaseContext.bulkDelete[Array[Byte]](rdd,
            TableName.valueOf(tableName),
            putRecord => new Delete(putRecord),
            4)
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkDeleteExample` file in `SparkOnHbasePythonExample`.

```
def main(args: Array[String]) {
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. This example uses Python to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
    Set spark.yarn.security.credentials.hbase.enabled to true.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseBulkDeleteExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteExample')
# Create a class instance and invoke the method, Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkDeleteExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 1.21.3.4.6 Using the BulkLoad Interface

#### Scenario

Users can use HBaseContext to use HBase in Spark applications, construct rowkey of the data to be inserted into RDDs, write RDDs into HFiles through the BulkLoad interface of HBaseContext. The following command is used to import the generated HFiles to the HBase table and will not be described in this section.

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles {hfilePath}  
{tableName}
```

#### Data Planning

1. Run the **hbase shell** command on the client to go to the HBase command line.
2. Run the following command to create HBase tables:  
`create 'bulkload-table-test','f1','f2'`

#### Development Guideline

1. Construct the data to be imported into RDDs
2. Perform operations on HBase in HBaseContext mode and write RDDs into HFiles through the BulkLoad interface of HBaseContext.

### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on FusionInsight Manager. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

#### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExa  
mple SparkOnHbaseJavaExample.jar /tmp/hfile bulkload-table-test
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseBulkLoadExample.py /tmp/hfile  
bulkload-table-test
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExa  
mple --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar /tmp/hfile bulkload-table-test
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseBulkLoadExample.py /tmp/hfile bulkload-table-test
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseBulkLoadExample file in SparkOnHbaseJavaExample.

```
public static void main(String[] args) throws IOException{  
    if (args.length < 2) {  
        System.out.println("JavaHBaseBulkLoadExample {outputPath} {tableName}");  
        return;  
    }  
    LoginUtil.loginWithUserKeytab();  
    String outputPath = args[0];  
    String tableName = args[1];  
    String columnFamily1 = "f1";  
    String columnFamily2 = "f2";  
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName);  
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
    try {  
        List<String> list= new ArrayList<String>();  
        // row1  
        list.add("1," + columnFamily1 + ",b,1");  
        // row3  
        list.add("3," + columnFamily1 + ",a,2");  
        list.add("3," + columnFamily1 + ",b,1");  
        list.add("3," + columnFamily2 + ",a,1");  
        /* row2 */  
        list.add("2," + columnFamily2 + ",a,3");  
        list.add("2," + columnFamily2 + ",b,3");  
        JavaRDD<String> rdd = jsc.parallelize(list);  
        Configuration conf = HBaseConfiguration.create();  
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);  
        hbaseContext.bulkLoad(rdd, TableName.valueOf(tableName),new BulkLoadFunction(), outputPath,  
        new HashMap<byte[], FamilyHFileWriteOptions>(), false, HConstants.DEFAULT_MAX_FILE_SIZE);  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        jsc.stop();  
    }  
}
```

```
    } finally {
        jsc.stop();
    }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkLoadExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    if(args.length < 2) {
        println("HBaseBulkLoadExample {outputPath} {tableName}")
        return
    }
    LoginUtil.loginWithUserKeytab()
    val Array(outputPath, tableName) = args
    val columnFamily1 = "f1"
    val columnFamily2 = "f2"
    val sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName)
    val sc = new SparkContext(sparkConf)
    try {
        val arr = Array("1," + columnFamily1 + ",b,1",
                      "2," + columnFamily1 + ",a,2",
                      "3," + columnFamily1 + ",b,1",
                      "3," + columnFamily2 + ",a,1",
                      "4," + columnFamily2 + ",a,3",
                      "5," + columnFamily2 + ",b,3")

        val rdd = sc.parallelize(arr)
        val config = HBaseConfiguration.create
        val hbaseContext = new HBaseContext(sc, config)
        hbaseContext.bulkLoad[String](rdd,
                                      TableName.valueOf(tableName),
                                      (putRecord) => {
            if(putRecord.length > 0) {
                val strArray = putRecord.split(",")
                val kfq = new KeyFamilyQualifier(Bytes.toBytes(strArray(0)), Bytes.toBytes(strArray(1)),
                                                Bytes.toBytes(strArray(2)))
                val ite = (kfq, Bytes.toBytes(strArray(3)))
                val itea = List(ite).iterator
                itea
            } else {
                null
            },
            outputPath)
        } finally {
            sc.stop()
        }
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkLoadPythonExample file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
Set spark.yarn.security.credentials.hbase.enabled to true.
```

```
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseBulkLoadExample")\
    .getOrCreate()
# Import required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.HBaseBulkLoadPythonExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.HBaseBulkLoadPythonExample().hbaseBulkLoad(spark._jsc, sys.argv[1], sys.argv[2])
# Stop the SparkSession instance.
spark.stop()
```

#### 1.21.3.4.7 Using the foreachPartition Interface

##### Scenario

Users can use HBaseContext to perform operations on HBase in the Spark application, construct rowkey of the data to be inserted into RDDs, and write RDDs to HBase tables through the mapPartition interface of HBaseContext.

##### Data Planning

1. Run the **hbase shell** command on the client to go to the HBase command line.
2. Run the following command to create an HBase table:  
**create 'table2','cf1'**

##### Development Guideline

1. Construct the data to be imported into RDDs
2. Perform operations on HBase in HBaseContext mode and concurrently write data to HBase through the foreachPartition interface of HBaseContext.

##### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

##### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set `spark.yarn.security.credentials.hbase.enabled` (`false` by default) in the `spark-defaults.conf` file on the Spark client to `true`. Changing the `spark.yarn.security.credentials.hbase.enabled` value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to `false`.) Set configuration item `spark.inputFormat.cache.enabled` to `false`.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the `$SPARK_HOME` directory on the client. The following commands are executed in the `$SPARK_HOME` directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachParti-  
tionExample SparkOnHbaseJavaExample.jar table2 cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseForEachPartitionExample.py table2  
cf1
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachParti-  
tionExample --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar table2 cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseForEachPartitionExample.py table2 cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseForEachPartitionExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String[] args) throws IOException {  
    if (args.length < 1) {  
        System.out.println("JavaHBaseForEachPartitionExample {tableName} {columnFamily}");  
        return;  
    }
```

```

    }
    LoginUtil.loginWithUserKeytab();
    final String tableName = args[0];
    final String columnFamily = args[1];
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try {
        List<byte[]> list = new ArrayList<byte[]>(5);
        list.add(Bytes.toBytes("1"));
        list.add(Bytes.toBytes("2"));
        list.add(Bytes.toBytes("3"));
        list.add(Bytes.toBytes("4"));
        list.add(Bytes.toBytes("5"));
        JavaRDD<byte[]> rdd = jsc.parallelize(list);
        Configuration conf = HBaseConfiguration.create();
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
        hbaseContext.foreachPartition(rdd,
            new VoidFunction<Tuple2<Iterator<byte[]>, Connection>>() {
                public void call(Tuple2<Iterator<byte[]>, Connection> t)
                    throws Exception {
                    Connection con = t._2();
                    Iterator<byte[]> it = t._1();
                    BufferedMutator buf = con.getBufferedMutator(tableName.valueOf(tableName));
                    while (it.hasNext()) {
                        byte[] b = it.next();
                        Put put = new Put(b);
                        put.add(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"), b);
                        buf.mutate(put);
                    }
                    buf.flush();
                    buf.close();
                }
            });
        } finally {
            jsc.stop();
        }
    }
}

```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseForEachPartitionExample` file in `SparkOnHbaseScalaExample`.

```

def main(args: Array[String]) {
    if (args.length < 2) {
        println("HBaseForEachPartitionExample {tableName} {columnFamily} are missing an arguments")
        return
    }
    LoginUtil.loginWithUserKeytab()
    val tableName = args(0)
    val columnFamily = args(1)
    val sparkConf = new SparkConf().setAppName("HBaseForEachPartitionExample " +
        tableName + " " + columnFamily)
    val sc = new SparkContext(sparkConf)
    try {
        //[(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])]
        val rdd = sc.parallelize(Array(
            (Bytes.toBytes("1"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1")))),
            (Bytes.toBytes("2"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2")))),
            (Bytes.toBytes("3"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3")))),
            (Bytes.toBytes("4"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4")))),
            (Bytes.toBytes("5"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5"))))
        ))
    }
}

```

```
val conf = HBaseConfiguration.create()
val hbaseContext = new HBaseContext(sc, conf)
rdd.hbaseForEachPartition(hbaseContext,
  (it, connection) => {
  val m = connection.getBufferedMutator(TableName.valueOf(tableName))
  it.foreach(r => {
    val put = new Put(r._1)
    r._2.foreach((putValue) =>
      put.addColumn(putValue._1, putValue._2, putValue._3))
    m.mutate(put)
  })
  m.flush()
  m.close()
})
} finally {
  sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseForEachPartitionExample` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
  Set spark.yarn.security.credentials.hbase.enabled to true.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
  .builder\
  .appName("JavaHBaseForEachPartitionExample")\
  .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachPartitionExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseForEachPartitionExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 1.21.3.4.8 Distributedly Scanning HBase Tables

#### Scenario

Users can use `HBaseContext` to perform operations on HBase in Spark applications and use HBase RDDs to scan HBase tables based on specific rules.

#### Data Planning

Use HBase tables created in [Performing Operations on Data in Avro Format](#)

#### Development Guideline

1. Set the scanning rule. For example: `setCaching`.
2. Use specific rules to scan the HBase table.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hBaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS  
canExample SparkOnHbaseJavaExample.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseDistributedScanExample.py  
ExampleAvrotable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS
```

```
canExample --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseDistributedScanExample.py ExampleAvrotable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseDistributedScanExample file in SparkOnHbaseJavaExample.

```
public static void main(String[] args) throws IOException{  
    if (args.length < 1) {  
        System.out.println("JavaHBaseDistributedScan {tableName}");  
        return;  
    }  
    LoginUtil.loginWithUserKeytab();  
    String tableName = args[0];  
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseDistributedScan " + tableName);  
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
    try {  
        Configuration conf = HBaseConfiguration.create();  
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);  
        Scan scan = new Scan();  
        scan.setCaching(100);  
        JavaRDD<Tuple2<ImmutableBytesWritable, Result>> javaRdd =  
            hbaseContext.hbaseRDD(tableName.valueOf(tableName), scan);  
        List<String> results = javaRdd.map(new ScanConvertFunction()).collect();  
        System.out.println("Result Size: " + results.size());  
    } finally {  
        jsc.stop();  
    }  
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseDistributedScanExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {  
    if (args.length < 1) {  
        println("HBaseDistributedScanExample {tableName} missing an argument")  
        return  
    }  
    LoginUtil.loginWithUserKeytab()  
    val tableName = args(0)  
    val sparkConf = new SparkConf().setAppName("HBaseDistributedScanExample " + tableName )  
    val sc = new SparkContext(sparkConf)  
    try {  
        val conf = HBaseConfiguration.create()  
        val hbaseContext = new HBaseContext(sc, conf)  
        val scan = new Scan()  
        scan.setCaching(100)  
        val getRdd = hbaseContext.hbaseRDD(tableName.valueOf(tableName), scan)  
        getRdd.foreach(v => println(Bytes.toString(v._1.get())))  
        println("Length: " + getRdd.map(r => r._1.copyBytes()).collect().length);  
    } finally {  
        sc.stop()  
    }  
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseDistributedScanExample file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
    Set spark.yarn.security.credentials.hbase.enabled to true.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseDistributedScan")\
    .getOrCreate()
# Import the required class into sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedScanExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseDistributedScan().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 1.21.3.4.9 Using the mapPartition Interface

#### Scenario

Users can use the HBaseContext method to perform operations on HBase in Spark applications and use the mapPartition interface to traverse HBase tables in parallel.

#### Data Planning

Use HBase tables created in [Using the foreachPartition Interface](#).

#### Development Guideline

1. Construct RDDs corresponding to rowkey in HBase tables to be traversed.
2. Use the mapPartition interface to traverse the data corresponding to rowkey and perform simple operations.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.
- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hBaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitio  
nExample SparkOnHbaseJavaExample.jar table2
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseMapPartitionExample.py table2
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitio  
nExample --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar table2
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseMapPartitionExample.py table2
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseMapPartitionExample file in SparkOnHbaseJavaExample.

```
public static void main(String args[]) throws IOException {
    if(args.length <1){
        System.out.println("JavaHBaseMapPartitionExample {tableName} is missing an argument");
        return;
    }
    LoginUtil.loginWithUserKeytab();
    final String tableName = args[0];
    SparkConf sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try{
        List<byte []> list = new ArrayList();
        list.add(Bytes.toBytes("1"));
        list.add(Bytes.toBytes("2"));
        list.add(Bytes.toBytes("3"));
        list.add(Bytes.toBytes("4"));
        list.add(Bytes.toBytes("5"));

        JavaRDD<byte []> rdd = jsc.parallelize(list);
        Configuration hbaseconf = HBaseConfiguration.create();
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, hbaseconf);
        JavaRDD<byte []> getrdd = hbaseContext.mapPartitions(rdd, new
FlatMapFunction<Tuple2<Iterator<byte []>, Connection>, Object>() {
            public Iterator<Object> call(Tuple2<Iterator<byte []>, Connection> t)
                throws Exception {
                Table table = t._2.getTable(tableName);
                //go through rdd
                List<String> list = new ArrayList<String>();
                while(t._1.hasNext()){
                    byte[] bytes = t._1.next();
                    Result result = table.get(new Get(bytes));
                    Iterator<Cell> it = result.listCells().iterator();
                    StringBuilder sb = new StringBuilder();
                    sb.append(Bytes.toString(result.getRow()) + ":");
                    while(it.hasNext()){
                        Cell cell = it.next();
                        String column = Bytes.toString(cell.getQualifierArray());
                        if(column.equals("counter")){
                            sb.append("(" + column + "," + Bytes.toLong(cell.getValueArray()) + ")");
                        } else {
                            sb.append("(" + column + "," + Bytes.toString(cell.getValueArray()) + ")");
                        }
                    }
                    list.add(sb.toString());
                }
                return list.iterator();
            }
        });
        List<byte []> resultList = getrdd.collect();
        if(null == resultList || 0 == resultList.size()){
            System.out.println("Nothing matches!");
        }else{
            for(int i =0; i< resultList.size(); i++){
                System.out.println(resultList.get(i));
            }
        }
    } finally {
        jsc.stop();
    }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseMapPartitionExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    if (args.length < 1) {
        println("HBaseMapPartitionExample {tableName} is missing an argument")
        return
    }
    LoginUtil.loginWithUserKeytab()
    val tableName = args(0)
    val sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName)
    val sc = new SparkContext(sparkConf)
    try {
        //[(Array[Byte])]
        val rdd = sc.parallelize(Array(
            Bytes.toBytes("1"),
            Bytes.toBytes("2"),
            Bytes.toBytes("3"),
            Bytes.toBytes("4"),
            Bytes.toBytes("5")))
        val conf = HBaseConfiguration.create()
        val hbaseContext = new HBaseContext(sc, conf)
        val b = new StringBuilder
        val getRdd = rdd.hbaseMapPartitions[String](hbaseContext, (it, connection) => {
            val table = connection.getTable(tableName.valueOf(tableName))
            it.map{r =>
                //batching would be faster. This is just an example
                val result = table.get(new Get(r))
                val it = result.listCells().iterator()
                b.append(Bytes.toString(result.getRow) + ":")
                while (it.hasNext) {
                    val cell = it.next()
                    val q = Bytes.toString(cell.getQualifierArray)
                    if (q.equals("counter")) {
                        b.append("(" + q + "," + Bytes.toLong(cell.getValueArray) + ")")
                    } else {
                        b.append("(" + q + "," + Bytes.toString(cell.getValueArray) + ")")
                    }
                }
                b.toString()
            }
        })
        getRdd.collect().foreach(v => println(v))
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseMapPartitionExample file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
    Set spark.yarn.security.credentials.hbase.enabled to true.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
```

```
.appName("JavaHBaseMapPartitionExample")\
.getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseMapPartitionExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 1.21.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming

#### Scenario

Users can use HBaseContext to perform operations on HBase in Spark applications and write streaming data to HBase tables using the streamBulkPut interface.

#### Data Planning

1. Create a session connected to the client and run the **hbase shell** command in the session to go to the HBase command line.
2. Run the following command in the HBase command line to create an HBase table:  
**create 'streamingTable','cf1'**
3. In another session of the client, run the Linux command to construct a port for receiving data. (The command may be different for servers running different operating systems. For the SUSE operating system, the following command is used: **netcat -lk 9999**.)  
**nc -lk 9999**

After the command for submitting a task is executed, enter the data to be submitted in this command and receive the data through the HBase table.

 NOTE

To construct a port for receiving data, you need to install netcat on the server where the client is located.

#### Development Guideline

1. Use SparkStreaming to continuously read data from a specific port.
2. Write the read DStream to HBase tables through the streamBulkPut interface.

#### Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed (The file upload path must be the same as the path of the generated JAR file).

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hBaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) **\${ip}** must be the IP address of the host where the **nc -lk 9999** command is executed.

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkP  
utExample SparkOnHbaseJavaExample.jar ${ip} 9999 streamingTable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample.jar HBaseStreamingBulkPutExample.py ${ip}  
9999 streamingTable cf1
```

- yarn-cluster mode:

Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) **\${ip}** must be the IP address of the host where the **nc -lk 9999** command is executed.

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkP  
utExample --files /opt/user.keytab,/opt/krb5.conf  
SparkOnHbaseJavaExample.jar ${ip} 9999 streamingTable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.) Assume that the package name of the corresponding Java code is **SparkOnHbaseJavaExample.jar** and the package is saved to the current directory.

```
bin/spark-submit --master yarn --deploy-mode cluster --files /opt/  
user.keytab,/opt/krb5.conf --jars SparkOnHbaseJavaExample.jar  
HBaseStreamingBulkPutExample.py ${ip} 9999 streamingTable cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseStreamingBulkPutExample file in SparkOnHbaseJavaExample.

### NOTE

The **awaitTerminationOrTimeout()** method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
public static void main(String[] args) throws IOException {
    if (args.length < 4) {
        System.out.println("JavaHBaseBulkPutExample " +
            "{host} {port} {tableName}");
        return;
    }
    LoginUtil.loginWithUserKeytab();
    String host = args[0];
    String port = args[1];
    String tableName = args[2];
    String columnFamily = args[3];
    SparkConf sparkConf =
        new SparkConf().setAppName("JavaHBaseStreamingBulkPutExample " +
            tableName + ":" + port + ":" + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try {
        JavaStreamingContext jssc =
            new JavaStreamingContext(jsc, new Duration(1000));
        JavaReceiverInputDStream<String> javaDstream =
            jssc.socketTextStream(host, Integer.parseInt(port));
        Configuration conf = HBaseConfiguration.create();
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
        hbaseContext.streamBulkPut(javaDstream,
            TableName.valueOf(tableName),
            new PutFunction(columnFamily));
        jssc.start();
        jssc.awaitTerminationOrTimeout(60000);
        jssc.stop(false,true);
    }catch(InterruptedException e){
        e.printStackTrace();
    } finally {
        jsc.stop();
    }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseStreamingBulkPutExample file in SparkOnHbaseScalaExample.

### NOTE

The **awaitTerminationOrTimeout()** method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
def main(args: Array[String]): Unit = {
    loginUtil.loginWithUserKeytab()
    val host = args(0)
    val port = args(1)
    val tableName = args(2)
    val columnFamily = args(3)
    val conf = new SparkConf()
    conf.setAppName("HBase Streaming Bulk Put Example")
    val sc = new SparkContext(conf)
    try {
```

```
val config = HBaseConfiguration.create()
val hbaseContext = new HBaseContext(sc, config)
val ssc = new StreamingContext(sc, Seconds(1))
val lines = ssc.socketTextStream(host, port.toInt)
hbaseContext.streamBulkPut[String](lines,
    TableName.valueOf(tableName),
    (putRecord) => {
    if (putRecord.length() > 0) {
        val put = new Put(Bytes.toBytes(putRecord))
        put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("foo"), Bytes.toBytes("bar"))
        put
    } else {
        null
    }
})
ssc.start()
ssc.awaitTerminationOrTimeout(60000)
ssc.stop(stopSparkContext = false)
} finally {
    sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseStreamingBulkPutExample` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-
"""
[Note]
(1) PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
(2) If yarn-client is used, ensure that the spark.yarn.security.credentials.hbase.enabled parameter in the
spark-defaults.conf file under Spark/spark/conf/ is set to true on the Spark client.
    Set spark.yarn.security.credentials.hbase.enabled to true.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseStreamingBulkPutExample")\
    .getOrCreate()
# Import required class to sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample')
# Create class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseStreamingBulkPutExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 1.21.3.5 Reading Data from HBase and Write It Back to HBase

#### 1.21.3.5.1 Overview

##### Scenarios

Assume `table1` of HBase stores the user consumption amount of the current day and `table2` stores the history consumption data.

In `table1`, `key=1` and `cf:cid=100` indicate that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

## Data Preparation

Use the Spark-Beeline tool to create table1 and table2 (Spark table and HBase table, respectively), and insert data by HBase.

**Step 1** Ensure that JDBCServer is started. On the Spark client, perform the following operations using the Spark-Beeline command tool:

**Step 2** Use the Spark-Beeline tool to create Spark table1:

```
create table table1
(
    key string,
    cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
    hbaseTableName "table1",
    keyCols "key",
    colsMapping "cid=cf.cid");
```

**Step 3** Run the following command on HBase to insert data to table1:

```
put 'table1', '1', 'cf:cid', '100'
```

**Step 4** Use the Spark-Beeline tool to create Spark table2:

```
create table table2
(
    key string,
    cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
    hbaseTableName "table2",
```

```
keyCols "key",
colsMapping "cid=cf.cid");
```

**Step 5** Run the following command on HBase to insert data to table 2:

```
put 'table2', '1', 'cf:cid', '1000'
```

**Step 6** Run the following command to update **table1**:

```
flush 'table1'
```

```
----End
```

## Development Idea

1. Query the data in table1.
2. Query the data in table2 using the key value of table1.
3. Add up the queried data.
4. Write the results of the preceding step to table2.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** and **/opt/female/krb5.conf**.
  - Before running the sample program, set the **spark.yarn.security.credentials.hbase.enabled** configuration item to **true** in the **spark-defaults.conf** configuration file of Spark client. (The default value is **false**. Changing the value to **true** does not affect existing services.) If you want to uninstall the HBase service, change the value back to **false** first.
3. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code.

```
bin/spark-submit --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.SparkHbasetoHbase --master yarn --deploy-mode client /opt/female/SparkHbasetoHbase-1.0.jar
```

- Run the Python sample program.

 NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR package and place it in the same driver class directory. When running the Python program, configure **--jars** to load the JAR package to the directory where the Python file resides.
- The Python sample code does not provide authentication information. Configure **--keytab** and **--principal** to specify authentication information

```
bin/spark-submit --master yarn --deploy-mode client --keytab /opt/FIclient/user.keytab --principal sparkuser --conf spark.yarn.user.classpath.first=true --jars /opt/female/SparkHbasetoHbasePythonExample/SparkHbasetoHbase-1.0.jar,/opt/female/protobuf-java-2.5.0.jar /opt/female/SparkHbasetoHbasePythonExample/SparkHbasetoHbasePythonExample.py
```

### 1.21.3.5.2 Java Example Code

#### Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

#### Example Code

For details about the code, see the class  
com.huawei.bigdata.spark.examples.SparkHbasetoHbase.

Example code:

```
/**  
 * calculate data from hbase1/hbase2,then update to hbase2  
 */  
public class SparkHbasetoHbase {  
  
    public static void main(String[] args) throws Exception {  
  
        SparkConf conf = new SparkConf().setAppName("SparkHbasetoHbase");  
        conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");  
        conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrar");  
        JavaSparkContext jsc = new JavaSparkContext(conf);  
        // Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the  
        //classpath.  
        Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());  
  
        // Declare the information of the table to be queried.  
        Scan scan = new org.apache.hadoop.hbase.client.Scan();  
        scan.addFamily(Bytes.toBytes("cf")); //column family  
        org.apache.hadoop.hbase.protobuf.generated.ClientProtos.Scan proto = ProtobufUtil.toScan(scan);  
        String scanToString = Base64.encodeBytes(proto.toByteArray());  
        hbConf.set(TableInputFormat.INPUT_TABLE, "table1"); //table name  
        hbConf.set(TableInputFormat.SCAN, scanToString);  
  
        // Obtain the data in the table through the Spark interface.
```

```
JavaPairRDD rdd = jsc.newAPIHadoopRDD(hbConf, TableInputFormat.class,
ImmutableBytesWritable.class, Result.class);

// Traverse every Partition in the HBase table1 and update the HBase table2
//If less data, you can use rdd.foreach()

rdd.foreachPartition(
    new VoidFunction<Iterator<Tuple2<ImmutableBytesWritable, Result>>() {
        public void call(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws Exception {
            hBaseWriter(iterator);
        }
    }
);

jsc.stop();
}

/**
 * write to table2 in exetutor
 *
 * @param iterator partition data from table1
 */
private static void hBaseWriter(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws
IOException {
    //read hbase
    String tableName = "table2";
    String columnFamily = "cf";
    String qualifier = "cid";
    Configuration conf = HBaseConfiguration.create();

    Connection connection = null;
    Table table = null;

    try {
        connection =ConnectionFactory.createConnection(conf);
        table = connection.getTable(Table.Name.valueOf(tableName));

        List<Get> rowList = new ArrayList<Get>();
        List<Tuple2<ImmutableBytesWritable, Result>> table1List = new
ArrayList<Tuple2<ImmutableBytesWritable, Result>>();
        while (iterator.hasNext()) {
            Tuple2<ImmutableBytesWritable, Result> item = iterator.next();
            Get get = new Get(item._2().getRow());
            table1List.add(item);
            rowList.add(get);
        }

        //get data from hbase table2
        Result[] resultDataBuffer = table.get(rowList);

        //set data for hbase
        List<Put> putList = new ArrayList<Put>();
        for (int i = 0; i < resultDataBuffer.length; i++) {
            Result resultData = resultDataBuffer[i];//hbase2 row
            if (!resultData.isEmpty()) {
                //query hbase1Value
                String hbase1Value = "";
                Iterator<Cell> it = table1List.get(i)._2().listCells().iterator();
                while (it.hasNext()) {
                    Cell c = it.next();
                    // query table1 value by colomn family and colomn qualifier
                    if (columnFamily.equals(Bytes.toString(CellUtil.cloneFamily(c)))
                        && qualifier.equals(Bytes.toString(CellUtil.cloneQualifier(c)))) {
                        hbase1Value = Bytes.toString(CellUtil.cloneValue(c));
                    }
                }
            }

            String hbase2Value = Bytes.toString(resultData.getValue(columnFamily.getBytes(),
                qualifier.getBytes()));
            Put put = new Put(item.getRow());
            put.addColumn(columnFamily, qualifier, Bytes.toBytes(hbase2Value));
            putList.add(put);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (connection != null) {
            try {
                connection.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
Put put = new Put(table1List.get(i)._2().getRow());

//calculate result value
int resultValue = Integer.parseInt(hbase1Value) + Integer.parseInt(hbase2Value);
//set data to put
put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifier),
Bytes.toBytes(String.valueOf(resultValue)));
putList.add(put);
}

if (putList.size() > 0) {
    table.put(putList);
}
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (table != null) {
        try {
            table.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    if (connection != null) {
        try {
            // Close the HBase connection
            connection.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

}
```

### 1.21.3.5.3 Scala Example Code

#### Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

#### Example Code

For details about code, see  
[com.huawei.bigdata.spark.examples.SparkHbasetoHbase](#).

Example code:

```
/*
 * calculate data from hbase1/hbase2,then update to hbase2
 */
object SparkHbasetoHbase {

    case class FemaleInfo(name: String, gender: String, stayTime: Int)

    def main(args: Array[String]) {

        val conf = new SparkConf().setAppName("SparkHbasetoHbase")
        conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
        conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrar")
        val sc = new SparkContext(conf)
        // Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the
        //classpath.
```

```
val hbConf = HBaseConfiguration.create(sc.hadoopConfiguration)

// Declare the information of the table to be queried.
val scan = new Scan()
scan.addFamily(Bytes.toBytes("cf")) //column family
val proto = ProtobufUtil.toScan(scan)
val scanToString = Base64.encodeBytes(proto.toByteArray)
hbConf.set(TableInputFormat.INPUT_TABLE, "table1") //table name
hbConf.set(TableInputFormat.SCAN, scanToString)

// Obtain the data in the table through the Spark interface.
val rdd = sc.newAPIHadoopRDD(hbConf, classOf[TableInputFormat], classOf[ImmutableBytesWritable],
classOf[Result])

// Traverse every Partition in the HBase table1 and update the HBase table2
//If less data, you can use rdd.foreach()
rdd.foreachPartition(x => hBaseWriter(x))

sc.stop()
}

/**
 * write to table2 in exetutor
 *
 * @param iterator partition data from table1
 */
def hBaseWriter(iterator: Iterator[(ImmutableBytesWritable, Result)]): Unit = {
//read hbase
val tableName = "table2"
val columnFamily = "cf"
val qualifier = "cid"
val conf = HBaseConfiguration.create()

var table: Table = null
var connection: Connection = null

try {
connection =ConnectionFactory.createConnection(conf)
table = connection.getTable(TableNames.valueOf(tableName))

val iteratorArray = iterator.toArray
val rowList = new util.ArrayList[Get]()
for (row <- iteratorArray) {
val get = new Get(row._2.getRow)
rowList.add(get)
}

//get data from hbase table2
val resultDataBuffer = table.get(rowList)

//set data for hbase
val putList = new util.ArrayList[Put]()
for (i < 0 until iteratorArray.size) {
val resultData = resultDataBuffer(i) //hbase2 row
if (!resultData.isEmpty) {
//query hbase1Value
var hbase1Value = ""
val it = iteratorArray(i)._2.listCells().iterator()
while (it.hasNext) {
val c = it.next()
// query table1 value by column family and column qualifier
if (columnFamily.equals(Bytes.toString(CellUtil.cloneFamily(c)))
&& qualifier.equals(Bytes.toString(CellUtil.cloneQualifier(c)))) {
hbase1Value = Bytes.toString(CellUtil.cloneValue(c))
}
}

val hbase2Value = Bytes.toString(resultData.getValue(columnFamily.getBytes, qualifier.getBytes))
val put = new Put(iteratorArray(i)._2.getRow)
putList.add(put)
}
}
}
```

```
//calculate result value
val resultValue = hbase1Value.toInt + hbase2Value.toInt
//set data to put
put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifier),
Bytes.toBytes(resultValue.toString))
putList.add(put)
}
}

if (putList.size() > 0) {
    table.put(putList)
}
} catch {
    case e: IOException =>
        e.printStackTrace();
} finally {
    if (table != null) {
        try {
            table.close()
        } catch {
            case e: IOException =>
                e.printStackTrace();
        }
    }
    if (connection != null) {
        try {
            //Close the HBase connection.
            connection.close()
        } catch {
            case e: IOException =>
                e.printStackTrace()
        }
    }
}
}

/**
 * Define serializer class.
 */
class MyRegistrar extends KryoRegistrar {
    override def registerClasses(kryo: Kryo) {
        kryo.register(classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable])
        kryo.register(classOf[org.apache.hadoop.hbase.client.Result])
        kryo.register(classOf[Array[(Any, Any)]])
        kryo.register(classOf[Array[org.apache.hadoop.hbase.Cell]])
        kryo.register(classOf[org.apache.hadoop.hbase.NoTagsKeyValue])
        kryo.register(classOf[org.apache.hadoop.hbase.protobuf.generated.ClientProtos.RegionLoadStats])
    }
}
```

#### 1.21.3.5.4 Python Example Code

### Function

Use Spark to call an HBase API to operate HBase table1 and write the data analysis result of table1 to HBase table2.

### Example Code

PySpark does not provide HBase related APIs. In this example, Python with the Java programming language invoked is used.

The following code snippets are used as an example. For complete codes, see [SparkHbaseToHbasePythonExample](#).

```
# -*- coding:utf-8 -*-

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

# Create the SparkContext and set kryo serialization.
spark = SparkSession\
    .builder\
    .appName("SparkHbasetoHbase") \
    .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
    .config("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator") \
    .getOrCreate()

# Import the class that will run into sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHbasetoHbase')

# Create a class instance and invoke the method.
spark._jvm.SparkHbasetoHbase().hbasetohbase(spark._jsc)

# Stop the SparkSession
spark.stop()
```

### 1.21.3.6 Reading Data from Hive and Write It to HBase

#### 1.21.3.6.1 Overview

##### Scenarios

Assume that table **person** of Hive stores the user consumption amount of the current day and table2 of HBase stores the history consumption data.

In table person, name=1 and account=100 indicates that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

#### Data Preparation

Before develop the application, create the Hive table **person** and insert data to it. Create HBase table2.

##### Step 1 Place the source log file to HDFS.

1. Create a blank file **log1.txt** in the local and write the following content to the file:  
1,100
2. Create a directory **/tmp/input** in HDFS and copy the **log1.txt** file to the directory.
  - a. On the HDFS client, run the following commands to obtain the security authentication:

- ```
cd /opt/hadoopclient
kinit <Service user for authentication>
b. On the Linux HDFS client, run the hadoop fs -mkdir /tmp/input
   command (a hdfs dfs command provides the same function.) to create a
   directory.
c. On the Linux HDFS client, run the hadoop fs -put log1.txt /tmp/input
   command to upload the data file.
```

**Step 2** Store the imported data to the Hive table.

Ensure that JDBCServer is started. Use the Beeline tool to create a Hive table and insert data to it.

1. Run the following commands to create the Hive table person:

```
create table person
(
  name STRING,
  account INT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ESCAPED BY '\\'
STORED AS TEXTFILE;
```

2. Run the following command to insert data to the table:

```
load data inpath '/tmp/input/log1.txt' into table person;
```

**Step 3** Create a HBase table:

Ensure that JDBCServer is started. Use the Spark-Beeline tool to create a HBase table and insert data to it.

1. Run the following commands to create the HBase table table2:

```
create table table2
(
  key string,
  cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
  hbaseTableName "table2",
  keyCols "key",
  colsMapping "cid=cf.cid"
);
```

2. Run the following command on HBase to insert data to the table:

```
put 'table2', '1', 'cf:cid', '1000'
```

----End

## Development Idea

1. Query the data in Hive table person.
2. Query the data in table2 using the key value of table person.
3. Add the queried data.
4. Write the results of the preceding step to table2.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
2. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** and **/opt/female/krb5.conf**.
  - Before running the sample program, set the **spark.yarn.security.credentials.hbase.enabled** configuration item to **true** in the **spark-defaults.conf** configuration file of Spark client. (The default value is **false**. Changing the value to **true** does not affect existing services.) If you want to uninstall the HBase service, change the value back to **false** first.
3. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code.  
**bin/spark-submit --class com.huawei.bigdata.spark.examples.SparkHivetoHbase --master yarn --deploy-mode client /opt/female/SparkHivetoHbase-1.0.jar**
- Run the Python sample program.

#### NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR file and place it in the same directory. When running the Python program, use **--jars** to load the JAR file to **classpath**.
- The Python sample code does not provide authentication information. Configure **--keytab** and **--principal** to specify authentication information.

```
bin/spark-submit --master yarn --deploy-mode client --keytab /opt/
Flclient/user.keytab --principal sparkuser --jars /opt/female/
SparkHivetoHbasePythonExample/SparkHivetoHbase-1.0.jar /opt/female/
SparkHivetoHbasePythonExample/SparkHivetoHbasePythonExample.py
```

### 1.21.3.6.2 Java Sample Code

#### Function

In a Spark application, users can use Spark to call a Hive API to operate a Hive table, and write the data analysis result of the Hive table to an HBase table.

#### Sample Code

The following code snippets are used as an example. For complete codes, see [com.huawei.bigdata.spark.examples.SparkHivetoHbase](#).

```
/*
 * Read data from the Hive table, and obtain the corresponding record from the HBase table based on the
 * key value. Sum the obtained two data records and update the sum result to the HBase table.
 */
public class SparkHivetoHbase {

    public static void main(String[] args) throws Exception {

        String userPrincipal = "sparkuser";
        String userKeytabPath = "/opt/Flclient/user.keytab";
        String krb5ConfPath = "/opt/Flclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
        Configuration hadoopConf = new Configuration();
        LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
        // Use the Spark API to obtain table data.
        SparkConf conf = new SparkConf().setAppName("SparkHivetoHbase");
        JavaSparkContext jsc = new JavaSparkContext(conf);
        HiveContext sqlContext = new org.apache.spark.sql.hive.HiveContext(jsc);

        Dataset<Row> dataFrame = sqlContext.sql("select name, account from person");

        // Traverse every partition in the Hive table and update data to the HBase table.
        // If the number of data records is small, you can use the foreach() method.
        dataFrame.toJavaRDD().foreachPartition(
            new VoidFunction<Iterator<Row>>() {
                public void call(Iterator<Row> iterator) throws Exception {
                    hBaseWriter(iterator);
                }
            });
        spark.stop();
    }

    /**
     * Update records in the HBase table on the executor.
     *
     * @param iterator Partition data in the Hive table.
     */
}
```

```
private static void hBaseWriter(Iterator<Row> iterator) throws IOException {
    // Read the HBase table.
    String tableName = "table2";
    String columnFamily = "cf";
    Configuration conf = HBaseConfiguration.create();
    Connection connection = null;
    Table table = null;
    try {
        connection = ConnectionFactory.createConnection(conf);
        table = connection.getTable(Table.Name.valueOf(tableName));
        List<Row> table1List = new ArrayList<Row>();
        List<Get> rowList = new ArrayList<Get>();
        while (iterator.hasNext()) {
            Row item = iterator.next();
            Get get = new Get(item.getString(0).getBytes());
            table1List.add(item);
            rowList.add(get);
        }
        // Obtain the records in the HBase table.
        Result[] resultDataBuffer = table.get(rowList);
        // Modify records in the HBase table.
        List<Put> putList = new ArrayList<Put>();
        for (int i = 0; i < resultDataBuffer.length; i++) {
            // Hive table value
            Result resultData = resultDataBuffer[i];
            if (!resultData.isEmpty()) {
                // get hiveValue
                int hiveValue = table1List.get(i).getInt(1);
                // Obtain the HBase table value based on the column family and column.
                String hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes(), "cid".getBytes()));
                Put put = new Put(table1List.get(i).getString(0).getBytes());
                // Calculate the result.
                int resultValue = hiveValue + Integer.valueOf(hbaseValue);
                // Set the result to the Put object.
                put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"),
                    Bytes.toBytes(String.valueOf(resultValue)));
                putList.add(put);
            }
        }
        if (putList.size() > 0) {
            table.put(putList);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (table != null) {
            try {
                table.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                // Close the HBase connection.
                connection.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

### 1.21.3.6.3 Scala Example Code

#### Function

In Spark application, call Hive API using Spark to operate Hive table, analyze the data, and then write the analyzed data to the HBase table.

#### Example Code

For details about code, see  
[com.huawei.bigdata.spark.examples.SparkHbasetoHbase](#).

Example code

```
/*
 * calculate data from hive/hbase,then update to hbase
 */
object SparkHivetoHbase {

  case class FemaleInfo(name: String, gender: String, stayTime: Int)

  def main(args: Array[String]) {

    String userPrincipal = "sparkuser";
    String userKeytabPath = "/opt/FIclient/user.keytab";
    String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";
    Configuration hadoopConf = new Configuration();
    LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);
    // Obtain the data in the table through the Spark interface.

    val spark = SparkSession
      .builder()
      .appName("SparkHiveHbase")
      .config("spark.sql.warehouse.dir", "spaek-warehouse")
      .enableHiveSupport()
      .getOrCreate()

    import spark.implicits._
    val dataFrame = spark.sql("select name, account from person")

    // Traverse every Partition in the hive table and update the hbase table
    // If less data, you can use rdd.foreach()
    dataFrame.rdd.foreachPartition(x => hBaseWriter(x))

    spark.stop()
  }

  /**
   * write to hbase table in exetutor
   *
   * @param iterator partition data from hive table
   */
  def hBaseWriter(iterator: Iterator[Row]): Unit = {
    // read hbase
    val tableName = "table2"
    val columnFamily = "cf"
    val conf = HBaseConfiguration.create()

    var table: Table = null
    var connection: Connection = null

    try {
      connection = ConnectionFactory.createConnection(conf)
      table = connection.getTable(TableName.valueOf(tableName))

      val iteratorArray = iterator.toArray
      for (row<- iteratorArray) {
        val family = row.get(0).asInstanceOf[Cell].getFamily()
        val value = row.get(0).asInstanceOf[Cell].getQualifier()
        val timestamp = row.get(0).asInstanceOf[Cell].getTimestamp()
        val cell = new Cell(family, value, timestamp)
        table.put(row.get(1), cell)
      }
    } catch {
      case e: Exception =>
        e.printStackTrace()
    } finally {
      if (connection != null) {
        connection.close()
      }
    }
  }
}
```

```
val rowList = new util.ArrayList[Get]()
for (row <- iteratorArray) {
    val get = new Get(row.getString(0).getBytes)
    rowList.add(get)
}

// get data from hbase table
val resultDataBuffer = table.get(rowList)

// set data for hbase
val putList = new util.ArrayList[Put]()
for (i <- 0 until iteratorArray.size) {
    // hbase row
    val resultData = resultDataBuffer(i)
    if (!resultData.isEmpty) {
        // get hiveValue
        var hiveValue = iteratorArray(i).getInt(1)

        // get hbaseValue by column Family and column qualifier
        val hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes, "cid".getBytes))
        val put = new Put(iteratorArray(i).getString(0).getBytes)

        // calculate result value
        val resultValue = hiveValue + hbaseValue.toInt

        // set data to put
        put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"),
        Bytes.toBytes(resultValue.toString))
        putList.add(put)
    }
}

if (putList.size() > 0) {
    table.put(putList)
}
} catch {
    case e: IOException =>
    e.printStackTrace();
} finally {
    if (table != null) {
        try {
            table.close()
        } catch {
            case e: IOException =>
            e.printStackTrace();
        }
    }
    if (connection != null) {
        try {
            //Close the HBase connection.
            connection.close()
        } catch {
            case e: IOException =>
            e.printStackTrace()
        }
    }
}
}
```

#### 1.21.3.6.4 Python Example Code

### Function

In a Spark application, use Spark to call a Hive API to operate a Hive table, and write the data analysis result of the Hive table to an HBase table.

## Example Code

PySpark does not provide HBase related APIs. In this example, Python with the Java programming language invoked is used.

The following code snippets are used as an example. For complete codes, see [SparkHivetoHbasePythonExample](#).

```
# -*- coding:utf-8 -*-

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

# Create the SparkSession
spark = SparkSession\
    .builder\
    .appName("SparkHivetoHbase") \
    .getOrCreate()

# Import the class that will run into sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHivetoHbase')

# Create a class instance and invoke the method.
spark._jvm.SparkHivetoHbase().hivetohbase(spark._jsc)

# Stop the SparkSession
spark.stop()
```

### 1.21.3.7 Streaming Connecting to Kafka0-10

#### 1.21.3.7.1 Overview

##### Scenarios

Assume that the Kafka component receives one word record every 1 second.

The developed Spark application needs to achieve the following function:

Calculate the sum of records for each word in real time.

**log1.txt** example file:

```
LiuYang
YuanJing
GuoYijun
CaiXuyu
Liyuan
FangBo
LiuYang
YuanJing
GuoYijun
CaiXuyu
FangBo
```

##### Data Planning

Spark Streaming sample project data is stored in the Kafka component. A user with Kafka permission sends data to Kafka component.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.
2. Create the **input\_data1.txt** file in the local and copy the content of the **log1.txt** file to the **input\_data1.txt** file.

On the client installation node, create the **/home/data** directory and upload the **input\_data1.txt** file to the **/home/data** directory.

3. Change the value of **allow.everyone.if.no.acl.found**, the Broker configuration value of Kafka, to **true**.
4. Create a topic.

{*IP address of the Kafka cluster*} indicates the service IP address of the Broker service.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --bootstrap-server <IP address of the Kafka cluster:21007> --command-config $KAFKA_HOME/config/client.properties --replication-factor 1 --partitions 3 --topic {Topic}
```

5. Start the Producer of Kafka and send data to Kafka.

```
java -cp {ClassPath}
com.huawei.bigdata.spark.examples.StreamingExampleProducer
{BrokerList} {Topic}
```

In this command, **ClassPath** must contain the absolute path of the Kafka JAR package on the Spark client in addition to the path of the sample JAR package, for example: /opt/hadoopclient/Spark/spark/jars/\*:/opt/hadoopclient/Spark/spark/jars/streamingClient010/\*:{ClassPath}.

## Development Approach

1. Receive data from Kafka and generate DStream.
2. Collect the statistics of word records by category.
3. Calculate and print the result.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server where the files are located. Example: /opt/female/user.keytab and /opt/female/krb5.conf.

- Upload the JAR package to any directory (for example, /opt) on the server where the Spark client is located.
- Prepare dependency packages and upload the following JAR packages to the **\$SPARK\_HOME/jars/streamingClient010** directory on the server where the Spark client is located.

- spark-streaming-kafkaWriter-xxx.cbu.mrs.jar
- kafka-clients-xxx.jar
- spark-sql-kafka-xxx.cbu.mrs.xxx.jar
- spark-streaming-kafka-xxx.cbu.mrs.xxx.jar
- spark-token-provider-kafka-xxx.cbu.mrs.xxx.jar

 NOTE

- For the dependency package whose version number contains "cbu.mrs", download from the Huawei open-source image site.
- For the dependency package whose version number does not contain "cbu.mrs", obtain them from the Maven central repository.

## Running Tasks

When running the sample program, you need to specify **<checkpointDir>**, **<brokers>**, **<topic>**, and **<batchTime>**. **<checkPointDir>** indicates the path for storing the program result backup in HDFS. **<brokers>** indicates the Kafka address for obtaining metadata. **<topic>** indicates the topic name read from Kafka. **<batchTime>** indicates the interval for Streaming processing in batches.

 NOTE

The location of Spark Streaming Kafka dependency package on the client is different from the location of other dependency packages. For example, the path to the Spark Streaming Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**, whereas the path to other dependency packages is **\$SPARK\_HOME/jars**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \$ (files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=,; echo "\${files[\*]}")**

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to **\$SPARK\_HOME/conf/jaas.conf**:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=false  
    useTicketCache=true  
    debug=false;  
};
```

2. Add configuration items to **\$SPARK\_HOME/conf/jaas-zk.conf**:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab=".user.keytab"  
    principal="sparkuser@<system domain name>"  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

3. Use **--files** and relative path to submit the **keytab** file to ensure that the **keytab** file is loaded to the container of the executor.

4. For the port number in **<brokers>**, use **SASL\_PLAINTEXT** for the **Kafka 0-10 Write To Print** example, use **PLAINTEXT** for the **Write To Kafka 0-10** example.

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Sample code (**Spark Streaming** read **Kafka 0-10 Write To Print**)  
**bin/spark-submit --master yarn --deploy-mode client --files ./jaas.conf,./user.keytab --jars \$(files=(\${SPARK\_HOME}/jars/streamingClient010/\*.jar);IFS=;; echo "\${files[\*]}") --class com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /opt/SparkStreamingKafka010JavaExample-1.0.jar <checkpointDir> <brokers> <topic> <batchTime>**  
The configuration example is as follows:  
--files ./jaas.conf,./user.keytab //Use --files to specify the jaas.conf and keytab files.
- Sample code (**Spark Streaming Write To Kafka 0-10**)  
**bin/spark-submit --master yarn --deploy-mode client --jars \${files=(\${SPARK\_HOME}/jars/streamingClient010/\*.jar);IFS=;; echo "\${files[\*]}") --class com.huawei.bigdata.spark.examples.JavaDstreamKafkaWriter /opt/SparkStreamingKafka010JavaExample-1.0.jar <groupId> <brokers> <topics>**

### 1.21.3.7.2 Java Example Code

#### Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

#### Example Code (**Streaming Read Data from Kafka 0-10**)

The following code is an example. For details, see `com.huawei.bigdata.spark.examples.SecurityKafkaWordCount`.

```
**  
* Consumes messages from one or more topics in Kafka.  
* <checkPointDir> is the Spark Streaming checkpoint directory.  
* <brokers> is for bootstrapping and the producer will only use it for getting metadata  
* <topics> is a list of one or more kafka topics to consume from  
* <batchTime> is the Spark Streaming batch duration in seconds.  
*/  
public class SecurityKafkaWordCount  
{  
    public static void main(String[] args) throws Exception {  
        JavaStreamingContext ssc = createContext(args);  
  
        //The Streaming system starts.  
        ssc.start();  
        try {  
            ssc.awaitTermination();  
        } catch (InterruptedException e) {  
        }  
    }  
  
    private static JavaStreamingContext createContext(String[] args) throws Exception {  
        String checkPointDir = args[0];  
        String brokers = args[1];  
        String topics = args[2];  
        String batchTime = args[3];  
  
        // Create a Streaming startup environment.  
        SparkConf sparkConf = new SparkConf().setAppName("KafkaWordCount");  
        JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, new  
Duration(Long.parseLong(batchTime) * 1000));  
    }  
}
```

```
//Configure the CheckPoint directory for the Streaming.  
//This parameter is mandatory because of existence of the window concept.  
ssc.checkpoint(checkPointDir);  
  
// Get the list of topic used by kafka  
String[] topicArr = topics.split(",");  
Set<String> topicSet = new HashSet<String>(Arrays.asList(topicArr));  
Map<String, Object> kafkaParams = new HashMap();  
kafkaParams.put("bootstrap.servers", brokers);  
kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");  
kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");  
kafkaParams.put("group.id", "DemoConsumer");  
kafkaParams.put("security.protocol", "SASL_PLAINTEXT");  
kafkaParams.put("sasl.kerberos.service.name", "kafka");  
kafkaParams.put("kerberos.domain.name", "hadoop.<system domain name>");  
  
LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();  
ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicSet, kafkaParams);  
  
// Create direct kafka stream with brokers and topics  
// Receive data from the Kafka and generate the corresponding DStream  
JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(ssc,  
locationStrategy, consumerStrategy);  
  
// Obtain field properties in each row.  
JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {  
    @Override  
    public String call(ConsumerRecord<String, String> tuple2) throws Exception {  
        return tuple2.value();  
    }  
});  
  
// Aggregate the total time that calculate word count  
JavaPairDStream<String, Integer> wordCounts = lines.mapToPair(  
    new PairFunction<String, String, Integer>() {  
        @Override  
        public Tuple2<String, Integer> call(String s) {  
            return new Tuple2<String, Integer>(s, 1);  
        }  
    }).reduceByKey(new Function2<Integer, Integer, Integer>() {  
    @Override  
    public Integer call(Integer i1, Integer i2) {  
        return i1 + i2;  
    }  
}).updateStateByKey(  
    new Function2<List<Integer>, Optional<Integer>, Optional<Integer>>() {  
        @Override  
        public Optional<Integer> call(List<Integer> values, Optional<Integer> state) {  
            int out = 0;  
            if (state.isPresent()) {  
                out += state.get();  
            }  
            for (Integer v : values) {  
                out += v;  
            }  
            return Optional.of(out);  
        }  
});  
  
// print the results  
wordCounts.print();  
return ssc;  
}  
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see com.huawei.bigdata.spark.examples.DstreamKafkaWriter.

### NOTE

It is advisable to use the new API createDirectStream instead of the old API createStream for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
/*
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
public class JavaDstreamKafkaWriter {

    public static void main(String[] args) throws InterruptedException {
        if (args.length != 3) {
            System.err.println("Usage: JavaDstreamKafkaWriter <groupId> <brokers> <topic>");
            System.exit(1);
        }

        final String groupId = args[0];
        final String brokers = args[1];
        final String topic = args[2];

        SparkConf sparkConf = new SparkConf().setAppName("KafkaWriter");

        // Populate Kafka properties
        Map<String, Object> kafkaParams = new HashMap<String, Object>();
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("value.serializer", "org.apache.kafka.common.serialization.ByteArraySerializer");
        kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        kafkaParams.put("bootstrap.servers", brokers);
        kafkaParams.put("group.id", groupId);
        kafkaParams.put("auto.offset.reset", "smallest");

        // Create Spark Java streaming context
        JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));

        // Populate data to write to kafka
        List<String> sentData = new ArrayList();
        sentData.add("kafka_writer_test_msg_01");
        sentData.add("kafka_writer_test_msg_02");
        sentData.add("kafka_writer_test_msg_03");

        // Create Java RDD queue
        Queue<JavaRDD<String>> sent = new LinkedList();
        sent.add(ssc.sparkContext().parallelize(sentData));

        // Create java Dstream with the data to be written
        JavaDStream wStream = ssc.queueStream(sent);

        // Write to kafka
        JavaDStreamKafkaWriterFactory.fromJavaDStream(wStream).writeToKafka(
            JavaConverters.mapAsScalaMapConverter(kafkaParams).asScala(),
            new Function<String, ProducerRecord<String, byte[]>>() {
                public ProducerRecord<String, byte[]> call(String s) throws Exception {
                    return new ProducerRecord(topic, s.toString().getBytes());
                }
            });
    }

    ssc.start();
    ssc.awaitTermination();
}
```

```
}
```

### 1.21.3.7.3 Scala Example Code

#### Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

#### Example Code (Streaming Read Data from Kafka 0-10)

The following code is an example. For details, see [com.huawei.bigdata.spark.examples.SecurityKafkaWordCount](#).

```
/*
 * Consumes messages from one or more topics in Kafka.
 * <checkPointDir> is the Spark Streaming checkpoint directory.
 * <brokers> is for bootstrapping and the producer will only use it for getting metadata
 * <topics> is a list of one or more kafka topics to consume from
 * <batchTime> is the Spark Streaming batch duration in seconds.
 */
object SecurityKafkaWordCount {

    def main(args: Array[String]) {
        val ssc = createContext(args)

        //The Streaming system starts.
        ssc.start()
        ssc.awaitTermination()
    }

    def createContext(args : Array[String]) : StreamingContext = {

        val Array(checkPointDir, brokers, topics, batchSize) = args

        // Create a Streaming startup environment.
        val sparkConf = new SparkConf().setAppName("KafkaWordCount")
        val ssc = new StreamingContext(sparkConf, Seconds(batchSize.toLong))

        //Configure the CheckPoint directory for the Streaming.
        //This parameter is mandatory because of existence of the window concept.
        ssc.checkpoint(checkPointDir)

        // Get the list of topic used by kafka
        val topicArr = topics.split(",")
        val topicSet = topicArr.toSet
        val kafkaParams = Map[String, String](
            "bootstrap.servers" -> brokers,
            "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
            "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
            "group.id" -> "DemoConsumer",
            "security.protocol" -> "SASL_PLAINTEXT",
            "sasl.kerberos.service.name" -> "kafka",
            "kerberos.domain.name" -> "hadoop.<system domain name>"
        );

        val locationStrategy = LocationStrategies.PreferConsistent
        val consumerStrategy = ConsumerStrategies.Subscribe[String, String](topicSet, kafkaParams)

        // Create direct kafka stream with brokers and topics
        // Receive data from the Kafka and generate the corresponding DStream
        val stream = KafkaUtils.createDirectStream[String, String](ssc, locationStrategy, consumerStrategy)

        // Obtain field properties in each row.
    }
}
```

```
val tf = stream.transform ( rdd =>
    rdd.map(r => (r.value, 1L))
)

// Aggregate the total time that calculate word count
val wordCounts = tf.reduceByKey(_ + _)
val totalCounts = wordCounts.updateStateByKey(updataFunc)
totalCounts.print()

ssc
}

def updataFunc(values : Seq[Long], state : Option[Long]) : Option[Long] =
  Some(values.sum + state.getOrElse(0L))
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see [com.huawei.bigdata.spark.examples.DstreamKafkaWriter](#).

### NOTE

After updates to Spark, it is advisable to use the new API `createDirectStream` instead of the old API `createStream` for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
package com.huawei.bigdata.spark.examples

import scala.collection.mutable
import scala.language.postfixOps

import com.huawei.spark.streaming.kafka010.KafkaWriter._
import org.apache.kafka.clients.producer.ProducerRecord
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.{Milliseconds, StreamingContext}

/**
 * Example code to demonstrate the usage of dstream.writeToKafka API
 *
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
object DstreamKafkaWriter {
  def main(args: Array[String]) {

    if (args.length != 3) {
      System.err.println("Usage: DstreamKafkaWriter <groupId> <brokers> <topic>")
      System.exit(1)
    }

    val Array(groupId, brokers, topic) = args
    val sparkConf = new SparkConf().setAppName("KafkaWriter")

    // Populate Kafka properties
    val kafkaParams = Map[String, String](
      "bootstrap.servers" -> brokers,
      "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
      "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
      "value.serializer" -> "org.apache.kafka.common.serialization.ByteArraySerializer",
      "key.serializer" -> "org.apache.kafka.common.serialization.StringSerializer",
      "group.id" -> groupId,
      "auto.offset.reset" -> "latest"
    )
  }
}
```

```
// Create Spark streaming context
val ssc = new StreamingContext(sparkConf, Milliseconds(500));

// Populate data to write to kafka
val sentData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02",
    "kafka_writer_test_msg_03")

// Create RDD queue
val sent = new mutable.Queue[RDD[String]]()
sent.enqueue(ssc.sparkContext.makeRDD(sentData))

// Create Dstream with the data to be written
val wStream = ssc.queueStream(sent)

// Write to kafka
wStream.writeToKafka(kafkaParams,
    (x: String) => new ProducerRecord[String, Array[Byte]](topic, x.getBytes))

ssc.start()
ssc.awaitTermination()
}
```

### 1.21.3.8 Structured Streaming Project

#### 1.21.3.8.1 Overview

##### Scenarios

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

##### Data Planning

Sample project data of StructuredStreaming is stored in Kafka. A user with Kafka permission sends data to Kafka.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.
2. Change the value of **allow.everyone.if.no.acl.found**, the Broker configuration value of Kafka, to **true**.
3. Create a topic.

{*IP address of the Kafka cluster*} indicates the service IP address of the Broker service.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --bootstrap-server <IP address of the Kafka cluster:21007> --command-config $KAFKA_HOME/config/client.properties --replication-factor 1 --partitions 1 --topic {Topic}
```

4. Start the Producer of Kafka and send data to Kafka.  
{*ClassPath*} indicates the storage path of engineer JAR packages and is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:{ClassPath} com.huawei.bigdata.spark.examples.KafkaWordCountProducer {BrokerList} {Topic} {messagesPerSec} {wordsPerMessage}
```

## Development Approach

1. Receive data from Kafka and generate DataStreamReader.
2. Collect the statistics of word records.
3. Calculate and print the result.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

Before compilation and packaging, change the paths of the user.keytab and krb5.conf files in the sample code to the actual paths on the client server where the files are located. Example: **/opt/female/user.keytab** and **/opt/female/krb5.conf**.

- Upload the JAR package to any directory (for example, **/opt**) on the server where the Spark client is located.

## Running Tasks

- When running the sample application, you need to specify **<brokers>**, **<subscribe-type>**, **<topic>**, **<protocol>**, **<service>**, **<domain>** and **<checkpointDir>**. **<brokers>** indicates the Kafka address (port **21007** is required) for obtaining metadata, **<subscribe-type>** indicates the Kafka subscription type (for example, **subscribe**), and **<topic>** indicates the name of the topic read from Kafka. **<protocol>** indicates the secure access protocol (for example, **SASL\_PLAINTEXT**). **<service>** indicates the Kerberos service name (for example, **kafka**). **<domain>** indicates the Kerberos domain name (for example, **hadoop.<system domain name>**), **<checkpointDir>** indicates the path for storing checkpoint files.

 NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**. Whereas the path of the Spark Structured Streaming Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar);IFS=;; echo "\${files[\*]}")**

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to **\$SPARK\_HOME/conf/jaas.conf**:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=false  
    useTicketCache=true  
    debug=false;  
};
```

2. Add configuration items to **\$SPARK\_HOME/conf/jaas-zk.conf**:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab=".user.keytab"  
    principal="sparkuser@<system domain name>"  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

3. Use **--files** and relative path to submit the **keytab** file to ensure that the **keytab** file is loaded to the container of the executor.

 CAUTION

When submitting a structured stream task, you need to run the **--jars** command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the kafka-clients jar file from the **\$SPARK\_HOME/jars/streamingClient010** directory to the **\$SPARK\_HOME/jars** directory. Otherwise, the "class not found" error is reported.

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code:

```
bin/spark-submit --master yarn --deploy-mode client --conf  
spark.driver.userClassPathFirst=true --conf  
spark.executor.userClassPathFirst=true --files <local Path>/jaas.conf,<local  
path>/user.keytab --jars $(files=($SPARK_HOME/jars/streamingClient010/  
*.jar);IFS=;; echo "${files[*]}") --class  
com.huawei.bigdata.spark.examples.SecurityKafkaWordCount /opt/  
SparkStructuredStreamingScalaExample-1.0.jar <brokers> <subscribe-type>  
<topic> <protocol> <service> <domain> <checkpointDir>
```

The configuration example is as follows:

```
-files <local Path>/jaas.conf,<local Path>/user.keytab //Use --files to specify the jaas.conf and keytab  
files.
```

- Run the Python sample code:

 NOTE

When running the Python sample code, you need to add the JAR package of the Java project to the **streamingClient010/** directory.

```
bin/spark-submit --master yarn --deploy-mode client --deploy-mode
client --conf spark.driver.userClassPathFirst=true --conf
spark.executor.userClassPathFirst=true --files /opt/FIclient/user.keytab --
jars ${files=(${SPARK_HOME/jars/streamingClient010/*.jar});IFS=,;echo "${files[*]}"} /opt/female/SparkStructuredStreamingPythonExample/
SecurityKafkaWordCount.py <brokers> <subscribe-type> <topic> <protocol>
<service> <domain> <checkpointDir>
```

### 1.21.3.8.2 Java Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see [com.huawei.bigdata.spark.examples.SecurityKafkaWordCount](#).

 NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver. The default value is **append**. To change the value, see the **outputMode** description of [Spark > Scala in MapReduce Service \(MRS\) 3.1-LTS API Reference \(for Huawei Cloud Stack 8.3.1\)](#).

```
public class SecurityKafkaWordCount
{
    public static void main(String[] args) throws Exception {
        if (args.length < 6) {
            System.err.println("Usage: SecurityKafkaWordCount <bootstrap-servers> " +
                "<subscribe-type> <topics> <protocol> <service> <domain>");
            System.exit(1);
        }

        String bootstrapServers = args[0];
        String subscribeType = args[1];
        String topics = args[2];
        String protocol = args[3];
        String service = args[4];
        String domain = args[5];

        SparkSession spark = SparkSession
            .builder()
            .appName("SecurityKafkaWordCount")
            .getOrCreate();

        // Create DataSet representing the stream of input lines from kafka
        Dataset<String> lines = spark
            .readStream()
            .format("kafka")
            .option("kafka.bootstrap.servers", bootstrapServers)
            .option(subscribeType, topics)
            .option("kafka.security.protocol", protocol)
```

```
.option("kafka.sasl.kerberos.service.name", service)
.option("kafka.kerberos.domain.name", domain)
.load()
.selectExpr("CAST(value AS STRING)")
.as(Encoders.STRING());

// Generate running word count
Dataset<Row> wordCounts = lines.flatMap(new FlatMapFunction<String, String>() {
    @Override
    public Iterator<String> call(String x) {
        return Arrays.asList(x.split(" ")).iterator();
    }
}, Encoders.STRING()).groupByKey("value").count();

// Start running the query that prints the running counts to the console
StreamingQuery query = wordCounts.writeStream()
    .outputMode("complete")
    .format("console")
    .start();

query.awaitTermination();
}
```

### 1.21.3.8.3 Scala Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see [com.huawei.bigdata.spark.examples.SecurityKafkaWordCount](#).

##### NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver. The default value is **append**. To change the value, see the **outputMode** description of [Spark > Scala in MapReduce Service \(MRS\) 3.3.1-LTS API Reference \(for Huawei Cloud Stack 8.3.1\)](#).

```
object SecurityKafkaWordCount {
    def main(args: Array[String]): Unit = {
        if (args.length < 6) {
            System.err.println("Usage: SecurityKafkaWordCount <bootstrap-servers> " +
                "<subscribe-type> <topics> <protocol> <service> <domain>")
            System.exit(1)
        }

        val Array(bootstrapServers, subscribeType, topics, protocol, service, domain) = args

        val spark = SparkSession
            .builder
            .appName("SecurityKafkaWordCount")
            .getOrCreate()

        import spark.implicits._

        // Create DataSet representing the stream of input lines from kafka
        val lines = spark
            .readStream
            .format("kafka")
            .option("kafka.bootstrap.servers", bootstrapServers)
            .option(subscribeType, topics)
```

```
.option("kafka.security.protocol", protocol)
.option("kafka.sasl.kerberos.service.name", service)
.option("kafka.kerberos.domain.name", domain)
.load()
.selectExpr("CAST(value AS STRING)")
.as[String]

// Generate running word count
val wordCounts = lines.flatMap(_.split(" ")).groupBy("value").count()

// Start running the query that prints the running counts to the console
val query = wordCounts.writeStream
    .outputMode("complete")
    .format("console")
    .start()

query.awaitTermination()
}
```

### 1.21.3.8.4 Python Sample Code

## Function

In Spark applications, use StructuredStreaming to invoke Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word.

## Sample Code

The following code segment is only an example. For details, see [SecurityKafkaWordCount](#)



When there is new available data in Streaming DataFrame/Dataset, **outputMode** indicates data written to the Streaming receiver. The default value is **append**. To change the value, see the **outputMode** description of **Spark > Scala** in *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split

if __name__ == "__main__":
    if len(sys.argv) < 6:
        print("Usage: <bootstrapServers> <subscribeType> <topics> <protocol> <service> <domain>")
        exit(-1)

    bootstrapServers = sys.argv[1]
    subscribeType = sys.argv[2]
    topics = sys.argv[3]
    protocol = sys.argv[4]
    service = sys.argv[5]
    domain = sys.argv[6]

    # Initialize the SparkSession.
    spark = SparkSession.builder.appName("SecurityKafkaWordCount").getOrCreate()

    # Create the DataFrame of input lines stream from Kafka.
    # In security mode, set spark/conf/jaas.conf and jaas-zk.conf to KafkaClient.
    lines = spark.readStream.format("kafka")\
        .option("kafka.bootstrap.servers", bootstrapServers)\
```

```
.option(subscribeType, topics)\\
.option("kafka.security.protocol", protocol)\\
.option("kafka.sasl.kerberos.service.name", service)\\
.option("kafka.kerberos.domain.name", domain)\\
.load()\\
.selectExpr("CAST(value AS STRING)")

# Split lines into words.
words = lines.select(explode(split(lines.value, " ")).alias("word"))
# Generate the running word count.
wordCounts = words.groupBy("word").count()

# Start to query whether the running counts are printed to the console.
query = wordCounts.writeStream\\
.outputMode("complete")\\
.format("console")\\
.start()

query.awaitTermination()
```

### 1.21.3.9 Structured Streaming Stream-Stream Join

#### 1.21.3.9.1 Overview

##### Scenarios

An advertisement service involves advertisement request events, advertisement display events, and advertisement click events. The advertiser needs to collect statistics on valid advertisement displays and advertisement click data.

Known conditions are as follows:

1. Each time a user requests an advertisement, an advertisement request event is generated and saved to the adRequest topic of Kafka.
2. After an advertisement is requested, the advertisement may be displayed for multiple times. Each time the advertisement is displayed, an advertisement display event is generated and saved to the adShow topic of Kafka.
3. Each advertisement may be clicked for multiple times. Each time it is clicked, an advertisement click event is generated and saved to the adClick topic of Kafka.
4. For advertisement display:
  - a. If the duration from the time when a request is generated to the time when the advertisement is displayed exceeds A minutes, the display is invalid.
  - b. Advertisement display events generated during A minutes are valid events.
5. For advertisement click:
  - a. If the duration from the display event to the click event exceeds B minutes, the click is invalid.
  - b. If there are multiple click events within B minutes, only the first click event is valid.

The simple data structure in this scenario is as follows:

- Advertisement request event  
Data structure: adID<sup>reqTime</sup>
- Advertisement display event  
Data structure: adID<sup>showID</sup><sup>showTime</sup>
- Advertisement click event  
Data structure: adID<sup>showID</sup><sup>clickTime</sup>

Data association relationships are as follows:

- The advertisement request event is associated with the advertisement display event through the adID.
- The advertisement display event is associated with the advertisement click event through the adID and showID.

Data requirements:

- The delay from the time when data is generated to the time when the data reaches the stream processing engine does not exceed two hours.
- The time for advertisement request events, advertisement display events, and advertisement click events reach the stream processing engine may not be in sequence or be aligned with each other.

## Data Planning

1. Enable users with the Kafka permission to generate simulation data in Kafka.

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/
streamingClient010/*:{ClassPath}
com.huawei.bigdata.spark.examples.KafkaADEventProducer {BrokerList}
{timeOfProduceReqEvent} {eventTimeBeforeCurrentTime} {reqTopic}
{reqEventCount} {showTopic} {showEventMaxDelay} {clickTopic}
{clickEventMaxDelay}
```

### NOTE

- Ensure that the clusters are installed, including HDFS, Yarn, Spark, and Kafka.
- Set the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**.
- Start Kafka Producer and enable it to send data to Kafka.
- **{ClassPath}** indicates the path for storing the JAR package of the project. The path is specified by users. For details, see the step of exporting JAR packages in [Compiling and Running the Application](#).

Command example:

```
java -cp /opt/hadoopclient/Spark/spark/conf:/opt/
StructuredStreamingADScalaExample-1.0.jar:/opt/hadoopclient/Spark/
spark/jars/*:/opt/hadoopclient/Spark/spark/jars/streamingClient010/*
com.huawei.bigdata.spark.examples.KafkaADEventProducer
10.132.190.170:21005,10.132.190.165:21005 2h 1h req 10000000 show 5m
click 5m
```

This command creates three topics on Kafka: req, show, and click. Ten million data records of request events are generated in two hours. The time range of the request events is from one hour ahead of the current time to the current time, and up to five display events are randomly generated for each request event. The time range of the display events is from the request event time to

five minutes after the request event time. Up to five click events are randomly generated for each display event. The time range of the click events is from the display event time to five minutes after the display event time.

## Development Guideline

1. Use structured streaming to receive data from Kafka and generate request flows, display flows, and click flows.
2. Perform join query of data in request flows, display flows, and click flows.
3. Write the statistics result to Kafka.
4. Monitor the flow processing task status in the application.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed.
- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

### NOTE

Before compilation and packaging, change the paths of the user.keytab and krb5.conf files in the sample code to the actual paths on the client server where the files are located. Example: /opt/female/user.keytab or /opt/female/krb5.conf.

- Upload the JAR package to any directory (for example, /opt) on the server where the Spark client is located.

## Running Tasks

When running the sample program, you need to specify **<kafkaBootstrapServers>**, **<maxEventDelay>**, **<reqTopic>**, **<showTopic>**, **<maxShowDelay>**, **<clickTopic>**, **<maxClickDelay>**, **<triggerInterval>**, **<checkpointDir>**, **<kafkaProtocol>**, **<kafkaService>**, and **<kafkaDomain>**. **<kafkaBootstrapServers>** indicates the Kafka address for obtaining metadata (port 21007 is required), and **<maxEventDelay>** indicates the maximum delay from data generation to stream processing. **<reqTopic>** indicates the topic name of the request event. **<showTopic>** indicates the topic name of the display event. **<maxShowDelay>** indicates the maximum delay for displaying the event, and **<clickTopic>** indicates the topic name of the click event. **<maxClickDelay>** indicates the maximum delay time of a valid click event. **<triggerInterval>** indicates the interval for triggering a stream processing task. **<checkpointDir>** indicates the path for storing the checkpoint file, and **<kafkaProtocol>** indicates the secure access protocol (for example, **SASL\_PLAINTEXT**). **<kafkaService>** indicates the Kerberos service name (for example, **kafka**), and **<kafkaDomain>** indicates the Kerberos domain name (for example, **hadoop.<system domain name>**).

 NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**. Whereas the path of the Spark Structured Streaming Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars ${files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,:; echo "${files[*]}"}`

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to **\$SPARK\_HOME/conf/jaas.conf**:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=false  
    useTicketCache=true  
    debug=false;  
};
```

2. Add configuration items to **\$SPARK\_HOME/conf/jaas-zk.conf**

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab=".:/user.keytab"  
    principal="sparkuser@<System domain name>"  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

3. Use **--files** and relative path to submit the **keytab** file to ensure that the **keytab** file is loaded to the container of the executor.

 CAUTION

When submitting a structured stream task, you need to run the **--jars** command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clientsjar` file from the **\$SPARK\_HOME/jars/streamingClient010** directory to the **\$SPARK\_HOME/jars** directory. Otherwise, the "class not found" error is reported.

Go to the Spark client directory and run the following command to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

```
bin/spark-submit --master yarn --deploy-mode client --files <local Path>/  
jaas.conf,<local path>/user.keytab --jars ${files=($SPARK_HOME/jars/  
streamingClient010/*.jar); IFS=,:; echo "${files[*]}"} --conf  
"spark.sql.streaming.statefulOperator.checkCorrectness.enabled=false" --class  
com.huawei.bigdata.spark.examples.KafkaADCount /opt/  
StructuredStreamingADScalaExample-1.0.jar <kafkaBootstrapServers>  
<maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> <clickTopic>  
<maxClickDelay> <triggerInterval> <checkpointDir> <kafkaProtocol>  
<kafkaService> <kafkaDomain>
```

Go to the Spark client directory and run the following command to invoke the **bin/spark-submit** script to run the code:

```
--files ./jaas.conf,./user.keytab //Use --files to specify the jaas.conf and keytab files.
```

### 1.21.3.9.2 Scala Example Code

#### Function

Structured Streaming is used to read advertisement request data, display data, and click data from Kafka, obtain effective display statistics and click statistics in real time, and write the statistics to Kafka.

#### Example code

The following code snippets are used as an example. For complete codes, see com.huawei.bigdata.spark.examples.KafkaADCount.

```
/*
 * Run the Structured Streaming task to collect statistics on valid advertisement display and click data. The
result is written into Kafka.
 */

object KafkaADCount {
    def main(args: Array[String]): Unit = {
        if (args.length < 12) {
            System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +
                "<maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> " +
                "<clickTopic> <maxClickDelay> <triggerInterver> " +
                "<checkpointLocation> <protocol> <service> <domain>")
            System.exit(1)
        }

        val Array(bootstrapServers, maxEventDelay, reqTopic, showTopic,
            maxShowDelay, clickTopic, maxClickDelay, triggerInterver, checkpointLocation,
            protocol, service, domain) = args

        val maxEventDelayMills = JavaUtils.timeStringAs(maxEventDelay, TimeUnit.MILLISECONDS)
        val maxShowDelayMills = JavaUtils.timeStringAs(maxShowDelay, TimeUnit.MILLISECONDS)
        val maxClickDelayMills = JavaUtils.timeStringAs(maxClickDelay, TimeUnit.MILLISECONDS)
        val triggerMills = JavaUtils.timeStringAs(triggerInterver, TimeUnit.MILLISECONDS)

        val spark = SparkSession
            .builder
            .appName("KafkaADCount")
            .getOrCreate()

        spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

        import spark.implicits._

        // Create DataSet representing the stream of input lines from kafka
        val reqDf = spark
            .readStream
            .format("kafka")
            .option("kafka.bootstrap.servers", bootstrapServers)
            .option("kafka.security.protocol", protocol)
            .option("kafka.sasl.kerberos.service.name", service)
            .option("kafka.kerberos.domain.name", domain)
            .option("subscribe", reqTopic)
            .load()
            .selectExpr("CAST(value AS STRING)")
            .as[String]
            .map{
                _split('^') match {
                    case Array(reqAdID, reqTime) => ReqEvent(reqAdID,
                        Timestamp.valueOf(reqTime))
                }
            }
            .as[ReqEvent]
            .withWatermark("reqTime", maxEventDelayMills +
```

```
maxShowDelayMills + " millisecond")

val showDf = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option("subscribe", showTopic)
    .load()
    .selectExpr("CAST(value AS STRING)")
    .as[String]
    .map{
        _split('^') match {
            case Array(showAdID, showID, showTime) => ShowEvent(showAdID,
                showID, Timestamp.valueOf(showTime))
        }
    }
    .as[ShowEvent]
    .withWatermark("showTime", maxEventDelayMills +
        maxShowDelayMills + maxClickDelayMills + " millisecond")

val clickDf = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option("subscribe", clickTopic)
    .load()
    .selectExpr("CAST(value AS STRING)")
    .as[String]
    .map{
        _split('^') match {
            case Array(clickAdID, clickShowID, clickTime) => ClickEvent(clickAdID,
                clickShowID, Timestamp.valueOf(clickTime))
        }
    }
    .as[ClickEvent]
    .withWatermark("clickTime", maxEventDelayMills + " millisecond")

val showStaticsQuery = reqDf.join(showDf,
    expr(s"""
        reqAdID = showAdID
        AND showTime >= reqTime + interval ${maxShowDelayMills} millisecond
    """))
    .selectExpr("concat_ws('^', showAdID, showID, showTime) as value")
    .writeStream
    .queryName("showEventStatics")
    .outputMode("append")
    .trigger(Trigger.ProcessingTime(triggerMills.millis))
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option("topic", "showEventStatics")
    .start()

val clickStaticsQuery = showDf.join(clickDf,
    expr(s"""
        showAdID = clickAdID AND
        showID = clickShowID AND
        clickTime >= showTime + interval ${maxClickDelayMills} millisecond
    """), joinType = "rightouter")
    .dropDuplicates("showAdID")
    .selectExpr("concat_ws('^', clickAdID, clickShowID, clickTime) as value")
```

```
.writeStream
.queryName("clickEventStatics")
.outputMode("append")
.trigger(Trigger.ProcessingTime(triggerMills.millis))
.format("kafka")
.option("kafka.bootstrap.servers", bootstrapServers)
.option("kafka.security.protocol", protocol)
.option("kafka.sasl.kerberos.service.name", service)
.option("kafka.kerberos.domain.name", domain)
.option("topic", "clickEventStatics")
.start()

new Thread(new Runnable {
    override def run(): Unit = {
        while(true) {
            println("-----get showStatic progress-----")
            //println(showStaticsQuery.lastProgress)
            println(showStaticsQuery.status)
            println("-----get clickStatic progress-----")
            //println(clickStaticsQuery.lastProgress)
            println(clickStaticsQuery.status)
            Thread.sleep(10000)
        }
    }
}).start

spark.streams.awaitAnyTermination()
}
```

### 1.21.3.10 Spark on Elasticsearch

#### 1.21.3.10.1 Overview

##### Scenarios

In Spark, you can create, delete, and query indexes using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.

##### Data Planning

Perform the following operations to save the data files in HDFS:

1. Obtain a test data file and obtain the **people.json** file in the **data** directory of the sample code.
2. Create the **spark-on-es** folder in HDFS and upload the **people.json** file to the directory **/user/spark-on-es** of HDFS. The command is as follows:
  - a. On the HDFS client, run the following commands to obtain security authentication:  
**cd /opt/hadoopclient**  
**kinit <Service user for authentication>**
  - b. On a Linux-based HDFS client, run the **hadoop fs -mkdir /user/spark-on-es** command (the **hdfs dfs** command provides the same function).
  - c. On a Linux-based HDFS client, run the **hadoop fs -put people.json /user/spark-on-es** command.

## Development Idea

1. Initialize basic configuration parameters of Elasticsearch.
2. Create an Elasticsearch client.
3. Create indexes of Elasticsearch.
4. Initialize the SparkContext.
5. Use the **BulkRequest** command to add data in batches in Elasticsearch.
6. Spark reads the test data (the **people.json** file obtained in **Data Planning**) from HDFS and adds the data to the Elasticsearch indexes created in 3 using the elasticsearch-spark API.
7. Query data in the Elasticsearch indexes. Two query methods displayed in the sample are as follows:
  - Query data using the elasticsearch-spark API.
  - Query data using the Elasticsearch Scroll API.
8. Operate the queried data by using the Spark Application. The sample demonstrates how to group and sort the queried data.

## Configuration Files

The content of the **esParams.properties** configuration file is as follows:

```
# IP address and port number of the node in the Elasticsearch cluster. Enter a value in the format of "IP
address:port number". Separate multiple values with commas (,).
esServerHost=10.10.10.11:24100,10.10.10.12:24100,10.10.10.13:24100
# Whether to enable the sniffer function.
snifferEnable=true
connectTimeout=5000
socketTimeout=60000
connectionRequestTimeout=3000
# Maximum number of connections on a single route.
maxConnPerRoute=100
maxConnTotal=1000
maxRetryTimeoutMillis=60000

# Whether the cluster is in security mode. If yes, set the parameter to true; if no, set it to false.
isSecureMode=true
# Whether to enable the SSL mode of Elasticsearch. If the cluster is in security mode, set the parameter to
true; otherwise, set it to false.
sslEnabled=true
# The jaas.conf file path. If the parameter is left blank, the path is generated automatically.
customJaasPath=
# Authentication usernames of the Spark and Elasticsearch cluster.
principal=sparkuser
# If sslEnabled is set to true, userPassword must be set, or the Elasticsearch security cluster will fail to be
connected.
userPassword=xxxxxx

# Field to be returned. Separate multiple values with commas (,).
# For example: https://www.elastic.co/guide/en/elasticsearch/hadoop/current/configuration.html
esFilterField=name,age,createdTime
# The maximum of Elasticsearch batch query is 10,000 by default.
# It is recommended that the value be less than 50,000. Otherwise, out of memory (OOM) may occur.
esScrollSize=10000
# This parameter is used to set the number of Spark data partitions. It is supported by Elasticsearch 5.0 and
later.
# This parameter, together with scroll.size, can improve the throughput and task concurrency.
# If it is not set, the number of Spark partitions is equal to that of index shards in Elasticsearch.
esInputMaxDocsPerPartition=500000

# The name of the index to be operated in Elasticsearch, the number of shards, and the number of replicas.
```

```
esIndex=people
esShardNum=5
esReplicaNum=1

# Information required for group statistics and range query in Elasticsearch.
esGroupField=age
esQueryField=createdTime
# Start value of the range query, which is included in the range.
esQueryRangeBegin=2010-01-01T00:00:00Z
# End value of the range query, which is not included in the range.
esQueryRangeEnd=2015-12-31T23:59:59
# JSON character string of Elasticsearch DSL, which cannot contain line breaks.
# Only the query parameter is supported. Other parameters such as _source and size are not supported.
esQueryJsonString={"query": {"range": { "%s": {"gte": "%s", "lt": "%s"} } } }
```

## Preparation Operations Before Running

In security mode, the **user.keytab** file, the **krb5.conf** file, and the **eParams.properties** file are needed in the SparkOnEs sample code. For details about the **eParams.properties** file, see the example in the **conf** directory of the sample code. The **user.keytab** and **krb5.conf** files are authentication files in security mode. The authentication credential of the **principal** user needs to be downloaded on FusionInsight Manager. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

For the Python sample, you need to modify the Python code and set **es.nodes** in **SparkOnEsPythonExample.py** to the IP address of the node where EsNode resides in the cluster.

## Packaging the Project

1. Modify the project configuration to make it the same as that of the cluster to be tested (You can find the Python sample configuration in the **SparkOnEsPythonExample.py** file.).
2. Use the Maven tool provided by IDEA to package the project and generate **target\SparkOnEs-1.0.jar**. For details, see [Compiling and Running the Application](#).
3. Upload the generated JAR package to any directory (for example, **/opt/spark-on-es/**) on the server where the Spark client is located.
4. Upload the **esParams.properties**, **user.keytab**, and **krb5.conf** files to the directory in **3**.
5. Prepare the dependency package. Upload the following JAR package to the **/opt/spark-on-es/libs/** directory on the server where the Spark client is located.

The following JAR packages are required for running the sample code:

- **elasticsearch-xxx.cbu.mrs.xxx.jar**
- **elasticsearch-core-xxx.cbu.mrs.xxx.jar**
- **elasticsearch-rest-client-xxx.cbu.mrs.xxx.jar**
- **elasticsearch-rest-high-level-client-xxx.cbu.mrs.xxx.jar**
- **elasticsearch-spark-20\_2.12-7.12.0.jar**
- **elasticsearch-x-content-xxx.cbu.mrs.xxx.jar**
- **lang-mustache-client-xxx.cbu.mrs.xxx.jar**

- log4j-api-2.12.0.jar
- log4j-core-2.12.0.jar
- lucene-core-8.7.0.jar
- rank-eval-client-xxx.cbu.mrs.xxx.jar
- commons-httpclient-\*jar

## Running a Task

After kinit authentication, run the following commands (the class name and file name must be the same as those in the actual code. The following is only an example):

1. In client mode:

```
cd /opt/spark-on-es
spark-submit --class com.huawei.bigdata.spark.examples.SparkOnEs --
master yarn --deploy-mode client --conf
spark.driver.userClassPathFirst=true --jars $(files=(/opt/spark-on-es/libs/
*.jar); IFS=,; echo "${files[*]}") ./SparkOnEs-1.0.jar
```

In the preceding command, `/opt/spark-on-es/libs/` indicates the path of the external dependency JAR package.

2. In cluster mode:

```
cd /opt/spark-on-es
spark-submit --class com.huawei.bigdata.spark.examples.SparkOnEs --
master yarn --deploy-mode cluster --conf
spark.driver.userClassPathFirst=true --jars $(files=(/opt/spark-on-es/libs/
*.jar); IFS=,; echo "${files[*]}") --files ./user.keytab./krb5.conf./
esParams.properties ./SparkOnEs-1.0.jar
```

In the preceding command, the `--files` parameter specifies the configuration files required for program running. Use commas (,) to separate multiple files.

## Querying Results

1. Query data in Elasticsearch

```
# After kinit authentication, check the index in Elasticsearch.
curl -XGET --tlsv1.2 --negotiate -k -u : 'https://10.10.10.11:24100/_cat/indices?v'

# Run the following commands to query the range of data in the people index.
curl -XPOST --tlsv1.2 --negotiate -k -u : 'https://10.10.10.11:24100/people/_search?pretty' -H 'Content-
Type:application/json' -d '
{
    "query": {
        "range": {
            "createdTime": {"gte": "2010-01-01T00:00:00Z", "lt": "2015-12-31T23:59:59Z"}
        }
    }
}'
```

2. Query files in HDFS.

```
# After kinit authentication, view the grouping result file saved in HDFS.
hdfs dfs -ls /user/spark-on-es/group-result

# View the content of a file. For example, view the content of file part-00001.
hdfs dfs -cat /user/spark-on-es/group-result/part-00001
```

### 1.21.3.10.2 Java Sample Code

#### Function Description

In Spark, you can read, write, and collect statistics on the data in Elasticsearch using Elasticsearch native APIs and the APIs provided by **elasticsearch-spark**.

#### Code Example

The following code snippets are used as an example. For complete codes, see the following `SparkOnEsJavaExample` project:

```
public static void main(String[] args) {
    LOG.info("***** Start to run the Spark on ES test.");

    try {
        // init the global properties
        initProperties();

        // login with keytab
        String userKeytabPath = System.getProperty("user.dir") + File.separator + "user.keytab";
        String krb5ConfPath = System.getProperty("user.dir") + File.separator + "krb5.conf";
        // generate jaas file, which is in the same dir with keytab file
        LoginUtil.setJaasFile(principal, userKeytabPath);
        Configuration hadoopConf = new Configuration();
        LoginUtil.login(principal, userKeytabPath, krb5ConfPath, hadoopConf);

        String path = System.getProperty("user.dir") + File.separator;
        HwRestClient hwRestClient = new HwRestClient(path);
        restClient = hwRestClient.getRestClient();
        highLevelClient = new RestHighLevelClient((hwRestClient.getRestClientBuilder()));

        // Check whether the index to be added has already exist, remove the exist one
        if (exist(esIndex)) {
            deleteIndex(esIndex);
        }
        createIndex(esIndex);

        SparkOnEs instance = new SparkOnEs();
        // Put data by ES bulk request
        instance.putDataByBulk(highLevelClient);

        // Create a configuration class SparkConf,
        // meanwhile set the Secure configuration that the Elasticsearch Cluster needed,
        // finally create a SparkContext.
        SparkConf conf =
            new SparkConf()
                .setAppName("SparkOnEs")
                .set("es.nodes", esServerHost)
                // when you specified in es.nodes, then es.port is not necessary
                // .set("es.port", "24100")
                .set("es.nodes.discovery", "true")
                .set("es.index.auto.create", "true")
                .set("es.internal.spark.sql.pushdown", "true")
                .set("es.net.ssl", sslEnabled)
                .set("es.net.http.auth.user", principal)
                .set("es.net.http.auth.pass", userPassword)
                .set("es.read.source.filter", esFilterField)
                .set("es.scroll.size", esScrollSize)
                .set("es.input.max.docs.per.partition", esInputMaxDocsPerPartition);

        JavaSparkContext jsc = new JavaSparkContext(conf);

        instance.putHdfsData(jsc); // Put data from HDFS into ES
        instance.queryDataByDataset(conf); // Query data from ES
        instance.queryDataByRestClient(highLevelClient, jsc);
        instance.groupBySpark(jsc); // Group data from ES
```

```
        jsc.stop();
    } catch (IOException e) {
        LOG.error("***** There are exceptions in main.", e);
    } finally {
        closeResources();
    }
}
```

### 1.21.3.10.3 Scala Sample Code

## Function

In Spark, you can read, write, and collect statistics on the data in Elasticsearch using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.

## Sample Code

The following code snippets are used as an example. For complete codes, see the following `SparkOnEsScalaExample` project:

```
object SparkOnEs {
  def main(args: Array[String]): Unit = {
    val sparkOnEs = new SparkOnEs
    sparkOnEs.LOG.info("***** Start to run the Spark on ES test.")

    try {
      // init the global properties
      sparkOnEs.initProperties()

      // login with keytab
      val userKeytabPath = System.getProperty("user.dir") + File.separator + "user.keytab"
      val krb5ConfPath = System.getProperty("user.dir") + File.separator + "krb5.conf"
      // generate jaas file, which is in the same dir with keytab file
      LoginUtil.setJaasFile(sparkOnEs.principal, userKeytabPath)
      val hadoopConf = new Configuration
      LoginUtil.login(sparkOnEs.principal, userKeytabPath, krb5ConfPath, hadoopConf)

      val path = System.getProperty("user.dir") + File.separator
      val hwRestClient: HwRestClient = new HwRestClient(path)
      sparkOnEs.restClient = hwRestClient.getRestClient
      sparkOnEs.highLevelClient = new RestHighLevelClient(hwRestClient.getRestClientBuilder)

      // Check whether the index to be added has already exist, remove the exist one
      if (sparkOnEs.exist(sparkOnEs.esIndex)) sparkOnEs.deleteIndex(sparkOnEs.esIndex)
      sparkOnEs.createIndex(sparkOnEs.esIndex)

      // Put data by ES bulk request
      sparkOnEs.putDataByBulk(sparkOnEs.highLevelClient)

      // Create a configuration class SparkConf,
      // meanwhile set the Secure configuration that the Elasticsearch Cluster needed,
      // finally create a SparkContext.
      val conf: SparkConf = new SparkConf().setAppName("SparkOnEs")
        .set("es.nodes", sparkOnEs.esServerHost)
        // when you specified in es.nodes, then es.port is not necessary
        // .set("es.port", "24100")
        .set("es.nodes.discovery", "true")
        .set("es.index.auto.create", "true")
        .set("es.internal.spark.sql.pushdown", "true")
        .set("es.net.ssl", sparkOnEs.sslEnabled)
        .set("es.net.http.auth.user", sparkOnEs.principal)
        .set("es.net.http.auth.pass", sparkOnEs.userPassword)
        .set("es.read.source.filter", sparkOnEs.esFilterField)
        .set("es.scroll.size", sparkOnEs.esScrollSize)
        .set("es.input.max.docs.per.partition", sparkOnEs.esInputMaxDocsPerPartition)
```

```
val jsc: JavaSparkContext = new JavaSparkContext(conf)

sparkOnEs.putHdfsData(jsc) // Put data from HDFS into ES
sparkOnEs.queryDataByDataset(conf) // Query data from ES
sparkOnEs.queryDataByRestClient(sparkOnEs.highLevelClient, jsc)
sparkOnEs.groupBySpark(jsc) // Group data from ES

jsc.close()
} catch {
  case e: IOException =>
    sparkOnEs.LOG.error("***** There are exceptions in main.", e)
} finally {
  sparkOnEs.closeResources()
}
}
```

### 1.21.3.10.4 Python Sample Code

## Function

In Spark, you can read, write, and collect statistics on the data in Elasticsearch using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.

## Sample Code

The following code snippets are used as an example. For complete codes, see the following `SparkOnEsPythonExample` project.

```
# -*- coding:utf-8 -*-
"""
[Note] Since PySpark doesn't support the Elasticsearch, this example calls Java code through Python.
"""

from py4j.java_gateway import java_import
from pyspark import SparkConf, SparkContext

# create the SparkConf
conf = SparkConf().setAppName("SparkOnEs")

# the configurations for Elasticsearch cluster
conf.set("es.nodes", "10.10.10.11:24100,10.10.10.12:24100,10.10.10.13:24100")
# when you specified in es.nodes, then es.port is not necessary
# conf.set("es.port", "24100")
conf.set("es.nodes.discovery", "true")
conf.set("es.index.auto.create", "true")
conf.set("es.internal.spark.sql.pushdown", "true")
conf.set("es.net.ssl", "true")
conf.set("es.net.http.auth.user", "sparkuser")
conf.set("es.net.http.auth.pass", "xxxxxx")
conf.set("es.read.source.filter", "name,age,createdTime")
conf.set("es.scroll.size", "10000")
conf.set("es.input.max.docs.per.partition", "500000")

# create the SparkContext
sc = SparkContext(conf = conf)

# import the program's class to sc._jvm
java_import(sc._jvm, "com.huawei.bigdata.spark.examples.SparkOnEs")

# create an instance of the above class, then invoke its method to run the program
sc._jvm.SparkOnEs().main(sc._jsc)

# stop the SparkContext
sc.stop()
```

## Running a Task

PySpark does not provide APIs for operating Elasticsearch. In this sample code, Python is used to invoke Java for task running. When running the task, you need to add the JAR package of the Java project to the **libs/** directory and upload the JAR package and other dependency packages to the class path.

After kinit authentication, run the following commands:

1. In client mode:

```
cd /opt/spark-on-es  
spark-submit --master yarn --deploy-mode client --conf  
spark.driver.userClassPathFirst=true --jars $(files=(/opt/spark-on-es/libs/  
*.jar); IFS=,; echo "${files[*]}") SparkOnEsPythonExample.py
```

In the preceding command, `/opt/spark-on-es/libs/` indicates the path of the external dependency JAR package, **including the sample JAR package**.

2. In cluster mode:

```
cd /opt/spark-on-es  
spark-submit --master yarn --deploy-mode cluster --conf  
spark.driver.userClassPathFirst=true --jars $(files=(/opt/spark-on-es/libs/  
*.jar); IFS=,; echo "${files[*]}") --files ./user.keytab,./krb5.conf,./  
esParams.properties SparkOnEsPythonExample.py
```

In the preceding command, the `--files` parameter specifies the configuration files required for program running. Use commas (,) to separate multiple files.

## 1.21.3.11 Spark on Solr

### 1.21.3.11.1 Overview

#### Scenarios

Users can use Spark to invoke Solr APIs to operate data in Solr. In the Spark application, users can use the Solr APIs to create, query, and delete indexes.

#### Development Idea

1. Initialize the Solr configuration and obtain a Solr client.
2. Create Solr indexes.
3. Add the data to the indexes.
4. Read the data in the indexes and use Spark to perform simple operations.
5. Delete the created indexes.
6. Read the data in the indexes again.

#### Dependency

- `solr-solrj-xxx.jar`
- `junit-4.11.jar`
- `spark-core_2.12-xxx.jar`

- hadoop-common-xxx.jar
- hadoop-mapreduce-client-core-xxx.jar
- slf4j-log4jxxx.jar
- commons-logging-xxx.jar
- commons-langxxx.jar
- jackson-databind-xxx.jar
- jetty-client-xxx.jar
- httpmime-xxx.jar

## Configuration Files

`solr-example.properties`

```
#zookeeper address
zkHost=10.111.111.111:24002/solr
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL=zookeeper/hadoop.<system domain name>
#users to be authenticated can be specified explicitly in the code (the user is explicitly specified as super in
the sample code)
principal=super
zkClientTimeout=20000
zkConnectTimeout=30000
#Whether it is security mode, if it is security mode, set it to true; if it is non-security mode, set to false
SOLR_KBS_ENABLED=true
COLLECTION_NAME=col_test
DEFAULT_CONFIG_NAME=confWithSchema
#Set this parameter based on the number of instance nodes. You are advised to set this parameter to the
number of instance nodes.
shardNum=3
replicaNum=1
maxShardsPerNode=1
autoAddReplicas=false
assignToSpecifiedNodeSet=false
#Solr service SolrServer instance IP
createNodeSet=10.111.111.111:21101_solr,10.222.222.222:21101_solr
```

## Preparation Operations Before Running

In security mode, the **user.keytab** file, the **krb5.conf** file, and the **solr-example.properties** file need to be used in the SparkOnSolr sample code. For details about the **solr-example.properties** file, see the example in the **conf** directory of the sample code. The **user.keytab** and **krb5.conf** files are authentication files in security mode. Download the authentication credentials of the corresponding users on the FusionInsight\_Manager page. The user in the sample code is **super**, change the value to the prepared development user name.

### Yarn-client mode

1. Copy the **user.keytab** file, the **krb5.conf** file, and the **solr-example.properties** file to the **\$SPARK\_HOME/bin** running directory on the client.
2. Extra JAR packages are required for running Solr. Therefore, you need to add the JAR package that is required by the Spark application and not included by the cluster to the class path of Driver. Modify **spark.driver.extraClassPath** in the **spark-defaults.conf** configuration file on the client. The configuration method is as follows:

**spark.driver.extraClassPath = /opt/hadoopclient/Spark/customJars/\***

#### NOTE

The JAR packages to be added to the sample code include **httpmime-xxx.jar** and **jetty-client-xxx.jar** (xxx indicates the version number of the JAR package, which can be obtained from the **\$SOLR\_HOME/lib/** directory). Add or delete as required.

3. Run the following command (the class name and file name must be the same as those in the actual code. The following is only an example):

```
./spark-submit --class com.huawei.bigdata.spark.examples.SparkOnSolr --  
jars /opt/hadoopclient/Spark/customJars/jetty-client-xxx.jar,/opt/  
hadoopclient/Spark/customJars/httpmime-xxx.jar --master yarn --deploy-  
mode client ./SparkOnSolr.jar
```

#### Yarn-cluster mode

1. Copy the **user.keytab** file, the **krb5.conf** file, and the **es-example.properties** file to any directory on the node on which the client is located.
2. Extra JAR packages are required for running Solr. Therefore, you need to add the JAR package required by the Spark application to the cluster and modify **spark.yarn.dist.innerfiles** and **spark.yarn.cluster.driver.extraClassPath** in the **spark-defaults.conf** file on the client. The configuration method is as follows:
  - Distribute the extra JAR packages to each node of the cluster by using the configuration item.  
**spark.yarn.dist.files = /opt/hadoopclient/Spark/customJars/httpmime-xxx.jar,/opt/hadoopclient/Spark/customJars/jetty-client-xxx.jar**
  - Add the specified JAR package to the class path of Driver in Yarn-cluster mode by using the configuration item.  
**spark.yarn.cluster.driver.extraClassPath = \$PWD/httpmime-xxx.jar:\$PWD/jetty-client-xxx.jar**
3. When running the Spark program, add -file and paths of the **user.keytab** file, the **krb5.conf** file, and the **es-example.properties** file to the command. For example:  

```
./spark-submit --class com.huawei.bigdata.spark.examples.SparkOnSolr --  
files ./user.keytab,/krb5.conf,/solr-example.properties --master yarn --  
deploy-mode cluster ./SparkOnSolr.jar
```

### 1.21.3.11.2 Java Sample Code

#### Function

In the Spark application, Solr client APIs are used to create indexes, insert data, and search for data.

#### Sample Code

The following code snippets are used as an example. For complete codes, see the following SparkOnSolrJavaExample project:

```
/**  
 * 1-Initialize Solr configuration and obtain the Solr client;  
 * 2-Create Solr indexes;  
 * 3-Add data to the indexes;  
 * 4-Read the data in the indexes and use Spark to perform simple operations;  
 * 5>Delete the created indexes;
```

```
* 6-Read the data in the indexes again.  
*/  
  
public class SparkOnSolr{  
    public static void main(String[] args) throws Exception {  
        SparkOnSolr testSample = new SparkOnSolr();  
        testSample.initProperties();  
        SparkConf conf = new SparkConf().setAppName("SparkOnSolr");  
        JavaSparkContext jsc = new JavaSparkContext(conf);  
        CloudSolrClient cloudSolrClient = null;  
        try {  
            cloudSolrClient = testSample.getCloudSolrClient(zkHost);  
            List<String> collectionNames = testSample.queryAllCollections(cloudSolrClient);  
            //delete the collection if exists  
            if (collectionNames.contains(collectionName)) {  
                testSample.deleteCollection(cloudSolrClient);  
            }  
            testSample.createCollection(cloudSolrClient);  
            cloudSolrClient.setDefaultCollection(collectionName);  
            testSample.addDocs(cloudSolrClient);  
            testSample.addDocs2(cloudSolrClient);  
            //Some time is needed to build the index  
            try {  
                Thread.sleep(2000);  
            } catch (InterruptedException e) {}  
            List<Map<String, Object>> queryResult1 = testSample.queryIndex(cloudSolrClient);  
            List<List<String>> resultList = new ArrayList<>();  
            for(int i=0; i < queryResult1.size(); i++){  
                resultList.add(testSample.transferMapToList(queryResult1.get(i)));  
            }  
            JavaRDD<List<String>> resultRDD = jsc.parallelize(resultList);  
            System.out.println("Before delete, the total count is:" + resultRDD.count() + "\n");  
            testSample.removeIndex(cloudSolrClient);  
            //Some time is needed to delete the index  
            try {  
                Thread.sleep(2000);  
            } catch (InterruptedException e) {}  
            List<Map<String, Object>> queryResult2 = testSample.queryIndex(cloudSolrClient);  
            JavaRDD<Map<String, Object>> resultRDD2 = jsc.parallelize(queryResult2);  
            System.out.println("After delete, the total count is:" + resultRDD2.count() + "\n");  
        } catch (SolrException e) {  
            throw new SolrException(e.getMessage());  
        } finally {  
            if (cloudSolrClient != null) {  
                try {  
                    cloudSolrClient.close();  
                } catch (IOException e) {  
                    LOG.warn("Failed to close cloudSolrClient", e);  
                }  
            }  
            jsc.stop();  
        }  
    }  
}
```

### 1.21.3.11.3 Scala Sample Code

#### Function

In the Spark application, Solr client APIs are used to create indexes, insert data, and search for data. Finally, Spark performs simple operations on the queried data.

#### Sample Code

The following code snippets are used as an example. For complete codes, see the following [SparkOnSolrScalaExample](#) project:

```
/*
 * 1-Initialize Solr configuration and obtain the Solr client;
 * 2>Create Solr indexes;
 * 3-Add data to the indexes;
 * 4-Read the data in the indexes and use Spark to perform simple operations;
 * 5>Delete the created indexes;
 * 6-Read the data in the indexes again.
 */
object SparkOnSolr{
  def main(args: Array[String]): Unit ={
    val testSample = new SparkOnSolr
    testSample.initProperties()
    val conf = new SparkConf().setAppName("SparkOnSolr")
    val sc = new SparkContext(conf)
    var cloudSolrClient: CloudSolrClient = null
    try {
      cloudSolrClient = testSample.getCloudSolrClient(testSample.zkHost)
      val collectionNames = testSample.queryAllCollections(cloudSolrClient)
      //delete the collection if exists
      if (collectionNames.contains(testSample.collectionName)) testSample.deleteCollection(cloudSolrClient)
      testSample.createCollection(cloudSolrClient)
      cloudSolrClient.setDefaultCollection(testSample.collectionName)
      testSample.addDocs(cloudSolrClient)
      testSample.addDocs2(cloudSolrClient)
      //Some time is needed to build the index
      try
        Thread.sleep(2000)
      catch {
        case e: InterruptedException =>
      }
      val queryResult1 = testSample.queryIndex(cloudSolrClient)
      import collection.JavaConversions._
      val resultRDD = sc.parallelize(queryResult1)
      System.out.println("Before delete, the total count is:" + resultRDD.count + "\n")
      testSample.removeIndex(cloudSolrClient)
      //Some time is needed to delete the index
      try
        Thread.sleep(2000)
      catch {
        case e: InterruptedException =>
      }
      val queryResult2 = testSample.queryIndex(cloudSolrClient)
      val resultRDD2 = sc.parallelize(queryResult2)
      System.out.println("After delete, the total count is:" + resultRDD2.count + "\n")
    } catch {
      case e: SolrException =>
        throw new SolrException(e.getMessage, null)
    } finally if (cloudSolrClient != null) try
      cloudSolrClient.close()
    catch {
      case e: IOException =>
        testSample.LOG.warn("Failed to close cloudSolrClient", e)
    }
    sc.stop()
  }
}
```

#### 1.21.3.11.4 Python Sample Code

### Function

In the Spark application, Solr client APIs are used to create indexes, insert data, and search for data. Finally, Spark performs simple operations on the queried data.

## Sample Code

PySpark does not provide Solr APIs. This example is implemented by using Python to invoke Java. When the application is running, you need to add some JAR packages of Java to the class path.

1. In Yarn-client mode, use the **--jars** parameter to load the Java **SparkOnSolr.jar** package to be invoked.

```
./spark-submit --jars SparkOnSolr.jar --master yarn-client  
SparkOnSolrPythonExample.py
```

2. In Yarn-cluster mode, modify the configuration item in the **spark-defaults.conf** file to add the **SparkOnSolr.jar** file to the class path. The configuration method is as follows:

- Distribute the extra JAR packages to each node of the cluster by using the configuration item.

```
spark.yarn.dist.innerfiles = /opt/hadoopclient/Spark/customJars/  
SparkOnSolr.jar,/opt/hadoopclient/Spark/customJars/  
httpmime-4.4.1.jar
```

- Add the specified JAR package to the class path of Driver in Yarn-cluster mode by using the configuration item.

```
spark.yarn.cluster.driver.extraClassPath = $PWD/  
SparkOnSolr.jar:$PWD/httpmime-4.4.1.jar
```

- Run the following command in Yarn-cluster mode:

```
./spark-submit --files ./user.keytab,./krb5.conf,./solr-  
example.properties --master yarn-cluster --jars SparkOnSolr.jar  
SparkOnSolrPythonExample.py
```

The following code snippets are used as an example. For complete codes, see the following **SparkOnSolrPythonExample** project:

```
# -*- coding:utf-8 -*-
from py4j.java_gateway import java_import
from pyspark import SparkContext

# Create SparkContext
spark = SparkContext(appName='SparkOnSolr')
# Import the class to be run to sc._jvm
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkOnSolr')

# Create class instances and invoke the method
spark._jvm.SparkOnSolr().sparkonsolr(spark._jsc)

# Stop SparkContext
spark.stop()
```

### 1.21.3.12 Structured Streaming Status Operation

#### 1.21.3.12.1 Overview

##### Scenarios

Assume that you need to collect statistics on the number of events in each session and the start and end timestamp of the sessions.

You need to export the sessions that are in the updated state in this batch.

## Data Planning

1. Generate simulated data in Kafka (the Kafka permission is required).
2. Ensure that the cluster has been installed, including the HDFS, Yarn, Spark, and Kafka services.
3. Set the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**.
4. Create a topic.

{*IP address of the Kafka cluster*} indicates the service IP address of the Broker service.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --bootstrap-server <IP address of the Kafka cluster:21007> --command-config $KAFKA_HOME/config/client.properties --replication-factor 1 --partitions 1 --topic {Topic}
```

5. Start the Producer of Kafka and send data to Kafka.

{*ClassPath*} indicates the storage path of the project JAR package that is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:{ClassPath} com.huawei.bigdata.spark.examples.KafkaProducer {brokerlist} {topic} {number of events produce every 0.02s}
```

Example:

```
java -cp /opt/hadoopclient/Spark/spark/conf:/opt/StructuredStreamingState-1.0.jar:/opt/hadoopclient/Spark/spark/jars/*:/opt/hadoopclient/Spark/spark/jars/streamingClient010/* com.huawei.bigdata.spark.examples.KafkaProducer xxx.xxx.xxx:21005,xxx.xxx.xxx:21005,xxx.xxx.xxx:21005 mytopic 10
```

## Development Guideline

1. Receive data from Kafka and generate the corresponding DataStreamReader.
2. Collect statistics by category.
3. Calculate the result and print it.

## Configuration Operations Before Running

In security mode, the Spark Core sample code needs to read two files (**user.keytab** and **krb5.conf**). The **user.keytab** and **krb5.conf** files are authentication files in the security mode. Download the authentication credentials of the user **principal** on the FusionInsight Manager page. The user in the sample code is **sparkuser**, change the value to the prepared development user name.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `/opt`) on the server where the Spark client is located.

- Upload the **user.keytab** and **krb5.conf** files to the server where the client is installed. (The file upload path must be the same as the path of the generated JAR file).

## Running Tasks

When running the example program, you need to specify **<brokers>**, **<subscribe-type>**, **<kafkaProtocol>**, **<kafkaService>**, **<kafkaDomain>**, **<topic>**, and **<checkpointLocation>**, where **<brokers>** indicates the Kafka address for obtaining metadata (port 21007 is required). **<subscribe-type>** indicates the Kafka consumption mode, and **<kafkaProtocol>** indicates the secure access protocol (such as **SASL\_PLAINTEXT**). **<kafkaService>** indicates the Kerberos service name (for example, **kafka**). **<kafkaDomain>** indicates the Kerberos domain name (for example, **hadoop.<system domain name>**). **<topic>** indicates the Kafka topic to be consumed, and **<checkpointLocation>** indicates the path for storing the checkpoint of the Spark task.

### NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**. Whereas the path of the Spark Streaming Structured Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=;; echo "\${files[\*]}")**

Because the cluster is security mode, you need to add configuration items and modify the command parameters.

1. Add configuration items to **\$SPARK\_HOME/conf/jaas.conf**:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=false  
    useTicketCache=true  
    debug=false;  
};
```

2. Add configuration items to **\$SPARK\_HOME/conf/jaas-zk.conf**:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab=".user.keytab"  
    principal="sparkuser@<system domain name>"  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

3. Use **--files** and relative path to submit the **keytab** file to ensure that the **keytab** file is loaded to the container of the executor.

Go to the Spark client directory and run the following command to invoke the **bin/spark-submit** script to run the code (the class name and file name must be the same as those in the actual code. The following is only an example):

- **bin/spark-submit --master yarn --deploy-mode client --files <local Path>/jaas.conf,<local path>/user.keytab --jars \$(files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar); IFS=;; echo "\${files[\*]}") --class com.huawei.bigdata.spark.examples.kafkaSessionization /opt/StructuredStreamingState-1.0.jar <brokers> <subscribe-type> <kafkaProtocol> <kafkaService> <kafkaDomain> <topic> <checkpointLocation>**

The configuration example is as follows:

```
--files ./jaas.conf,./user.keytab //Use --files to specify the jaas.conf and keytab files.
```

 CAUTION

When submitting a structured stream task, you need to run the **--jars** command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the kafka-clientsjar file from the **\$SPARK\_HOME/jars/streamingClient010** directory to the **\$SPARK\_HOME/jars** directory. Otherwise, the "class not found" error is reported.

### 1.21.3.12.2 Scala Sample Code

#### Function

In the Spark structure flow application, the number of events in each session and the start and end timestamp of the sessions are collected in different batches. At the same time, the system exports the sessions that are in the updated state in this batch.

#### Code Example

The following code snippets are used as an example. For complete codes, see [com.huawei.bigdata.spark.examples.kafkaSessionization](#).

 NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver. The default value is **append**. If you want to change the export mode, see the **outputMode** description of **Spark > Scala** in MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1).

```
object kafkaSessionization {
  def main(args: Array[String]): Unit = {
    if (args.length < 7) {
      System.err.println("Usage: kafkaSessionization <bootstrap-servers> " +
        "<subscribe-type> <protocol> <service> <domain> <topics> <checkpointLocation>")
      System.exit(1)
    }

    val Array(bootstrapServers, subscribeType, protocol, service, domain, topics, checkpointLocation) = args

    val spark = SparkSession
      .builder
      .appName("kafkaSessionization")
      .getOrCreate()

    spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

    spark.streams.addListener(new StreamingQueryListener {

      @volatile private var startTime: Long = 0L
      @volatile private var endTime: Long = 0L
      @volatile private var numRecs: Long = 0L

      override def onQueryStarted(event: StreamingQueryListener.QueryStartedEvent): Unit = {
        println("Query started: " + event.id)
        startTime = System.currentTimeMillis
      }
    })
  }
}
```

```
override def onQueryProgress(event: StreamingQueryListener.QueryProgressEvent): Unit = {
    println("Query made progress: " + event.progress)
    numRecs += event.progress.numInputRows
}

override def onQueryTerminated(event: StreamingQueryListener.QueryTerminatedEvent): Unit = {
    println("Query terminated: " + event.id)
    endTime = System.currentTimeMillis
}
}

import spark.implicits._

val df = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option(subscribeType, topics)
    .load()
    .selectExpr("CAST(value AS STRING)")
    .as[String]
    .map { x =>
        val splitStr = x.split(",")
        (splitStr(0), Timestamp.valueOf(splitStr(1)))
    }.as[(String, Timestamp)].flatMap { case(line, timestamp) =>
    line.split(" ").map(word => Event(sessionId = word, timestamp))}

// Sessionize the events. Track number of events, start and end timestamps of session, and
// and report session updates.
val sessionUpdates = df
    .groupByKey(event => event.sessionId)
    .mapGroupsWithState[SessionInfo, SessionUpdate](GroupStateTimeout.ProcessingTimeTimeout) {

    case (sessionId: String, events: Iterator[Event], state: GroupState[SessionInfo]) =>

        // If timed out, then remove session and send final update
        if (state.hasTimedOut) {
            val finalUpdate =
                SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = true)
            state.remove()
            finalUpdate
        } else {
            // Update start and end timestamps in session
            val timestamps = events.map(_.timestamp.getTime).toSeq
            val updatedSession = if (state.exists) {
                val oldSession = state.get
                SessionInfo(
                    oldSession.numEvents + timestamps.size,
                    oldSession.startTimestampMs,
                    math.max(oldSession.endTimestampMs, timestamps.max))
            } else {
                SessionInfo(timestamps.size, timestamps.min, timestamps.max)
            }
            state.update(updatedSession)

            // Set timeout such that the session will be expired if no data received for 10 seconds
            state.setTimeoutDuration("10 seconds")
            SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = false)
        }
    }

// Start running the query that prints the session updates to the console
val query = sessionUpdates
    .writeStream
    .outputMode("update")
```

```
.format("console")
.start()

query.awaitTermination()
}
```

### 1.21.3.13 Concurrent Access from Spark to HBase in Two Clusters

#### 1.21.3.13.1 Overview

##### Scenarios

Spark can access HBase in two clusters concurrently on condition that mutual trust has been configured between these two clusters.

##### Data Planning

1. Configure the IP addresses and host names of all ZooKeeper and HBase nodes in **cluster2** to the **/etc/hosts** file on the client node of **cluster1**.
2. In **cluster1** and **cluster2**, find the **hbase-site.xml** file in the **conf** directory of the Spark client, and save it to the **/opt/example/A** and **/opt/example/B** directories.
3. Run the following **spark-submit** command:

###### NOTE

Before running the sample program, set the **spark.yarn.security.credentials.hbase.enabled** configuration item to **true** in the **spark-defaults.conf** configuration file of Spark client. (The default value is **false**. Changing the value to **true** does not affect existing services.) If you want to uninstall the HBase service, change the value back to **false** first.

```
spark-submit --master yarn --deploy-mode client --files /opt/example/B/hbase-site.xml --keytab /opt/
Flclient/user.keytab --principal sparkuser --class com.huawei.spark.examples.SparkOnMultiHbase /opt/
example/SparkOnMultiHbase-1.0.jar
```

##### Development Approach

1. When accessing HBase, you need to use the configuration file of the corresponding cluster to create a **Configuration** object for creating a **Connection** object.
2. Use the **Connection** object you create to perform operations on the HBase table, such as creating a table and inserting, viewing, and printing data.

#### 1.21.3.13.2 Scala Sample Code

The following code snippets are used as an example. For complete codes, see **com.huawei.spark.examples.SparkOnMultiHbase**

```
def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("SparkOnMultiHbaseExample")
    conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
    conf.set("spark.kryo.registrator", "com.huawei.spark.examples.MyRegistrar")
    val sc = new SparkContext(conf)
    val tableName = "SparkOnMultiHbase"
    val clusterFlagList = List("B", "A")
    clusterFlagList.foreach{ item =>
        val hbaseConf = getConf(item)
    }
}
```

```
    println(hbaseConf.get("hbase.zookeeper.quorum"))
    val hbaseUtil = new HbaseUtil(sc,hbaseConf)
    hbaseUtil.writeToHbase(tableName)
    hbaseUtil.readFromHbase(tableName)
}
sc.stop()
}
private def getConf(item:String):Configuration={
    val conf: Configuration = HBaseConfiguration.create()
    val url = "/opt" + File.separator + "example" + File.separator + item + File.separator + "hbase-site.xml"
    conf.addResource(new File(url).toURI.toURL)
    conf
}
```

### 1.21.3.14 Synchronizing HBase Data from Spark to CarbonData

#### 1.21.3.14.1 Overview

##### Scenarios

Data is written to HBase in real time for point query services and is synchronized to CarbonData tables in batches at a specified interval for analytical query services.

##### Configuration Operations Before Running

In security mode, the sample code needs to read the **user.keytab** and **krb5.conf** files. The **user.keytab** and **krb5.conf** files are the authentication files in security mode. You need to download the authentication credential of the principal user on FusionInsight Manager. The user used in the sample code is **sparkuser**, which needs to be changed to the prepared development user.

##### Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is located.
2. Use the Maven tool provided by IDEA to package the project and generate the JAR file. For details, see [Compiling and Running the Application](#).

###### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server, for example, **/opt/user.keytab** and **/opt/krb5.conf**.
  - Before running the sample program, set the **spark.yarn.security.credentials.hbase.enabled** configuration item to **true** in the **spark-defaults.conf** configuration file of Spark client. (The default value is **false**. Changing the value to **true** does not affect existing services.) If you want to uninstall the HBase service, change the value back to **false** first.
3. Upload the generated JAR package to any directory (for example, **/opt/**) on the server where the Spark client is located.

##### Data Preparation

1. Create an HBase table and construct data with **key**, **modify\_time**, and **valid** columns. **key** of each data record is unique in the table. **modify\_time**

indicates the modification time, and **valid** indicates whether the data is valid. In this example, **1** indicates that the data is valid, and **0** indicates that the data is invalid.

For example, go to HBase Shell and run the following commands:

```
create 'hbase_table','key','info'  
put 'hbase_table','1','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','1','info:valid','1'  
put 'hbase_table','2','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','2','info:valid','1'  
put 'hbase_table','3','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','3','info:valid','0'  
put 'hbase_table','4','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','4','info:valid','1'
```

#### NOTE

The values of **modify\_time** in the preceding information can be set to the time earlier than the current time.

```
put 'hbase_table','5','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','5','info:valid','1'  
put 'hbase_table','6','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','6','info:valid','1'  
put 'hbase_table','7','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','7','info:valid','0'  
put 'hbase_table','8','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','8','info:valid','1'  
put 'hbase_table','4','info:valid','0'  
put 'hbase_table','4','info:modify_time','2021-03-03 15:20:39
```

#### NOTE

The values of **modify\_time** in the preceding information can be set to the time within 30 minutes after the sample program is started. (30 minutes is the default synchronization interval of the sample program and can be modified.)

```
put 'hbase_table','9','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','9','info:valid','1'  
put 'hbase_table','10','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','10','info:valid','1'  
put 'hbase_table','11','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','11','info:valid','0'  
put 'hbase_table','12','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','12','info:valid','1'
```

#### NOTE

The values of **modify\_time** in the preceding information can be set to the time from 30 minutes to 60 minutes after the sample program is started, that is, the second synchronization period.

2. Run the following commands to create a Hive foreign table for HBase in SparkSQL:

```
create table external_hbase_table(key string ,modify_time STRING, valid STRING)
```

```
using org.apache.spark.sql.hbase.HBaseSource
```

```
options(hbaseTableName "hbase_table", keyCols "key", colsMapping "modify_time=info.modify_time,valid=info.valid");
```

3. Run the following command to create a CarbonData table in SparkSQL:

```
create table carbon01(key string,modify_time STRING, valid STRING) stored as carbondata;
```

4. Initialize and load all data in the current HBase table to the CarbonData table.

```
insert into table carbon01 select * from external_hbase_table where valid='1';
```

5. Run the following **spark-submit** command:

```
spark-submit --master yarn --deploy-mode client --keytab /opt/FIclient/user.keytab --principal sparkuser --class com.huawei.bigdata.spark.examples.HBaseExternalHivetoCarbon /opt/example/HBaseExternalHivetoCarbon-1.0.jar
```

#### 1.21.3.14.2 Java Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.spark.examples.HBaseExternalHivetoCarbon](#).

```
public static void main(String[] args) throws Exception {
    spark = SparkSession.builder().appName("HBaseExternalHiveToCarbon").getOrCreate();

    Timer timer = new Timer();
    timer.schedule(new TimerTask() {
        public void run() {
            timeEnd = timeStart + TIMEWINDOW;

            queryTimeStart = transferDateToStr(timeStart);
            queryTimeEnd = transferDateToStr(timeEnd);

            //run delete logic
            cmdsb = new StringBuilder();
            cmdsb.append("delete from ")
                .append(carbonTableName)
                .append(" where key in (select key from ")
                .append(externalHiveTableName)
                .append(" where modify_time>'")
                .append(queryTimeStart)
                .append("' and modify_time<'")
                .append(queryTimeEnd)
                .append("' and valid='0')");
            spark.sql(cmdsb.toString());

            //run insert logic
            cmdsb = new StringBuilder();
            cmdsb.append("insert into ")
                .append(carbonTableName)
                .append(" select * from ")
                .append(externalHiveTableName)
                .append(" where modify_time>'")
                .append(queryTimeStart)
                .append("' and modify_time<'")
                .append(queryTimeEnd)
                .append("' and valid='1'");
            spark.sql(cmdsb.toString());

            timeStart = timeEnd;
        }
    }, 0, TIMEWINDOW);
}
```

```
    }, TIMEWINDOW, TIMEWINDOW);
}
```

### 1.21.3.15 Using Spark to Perform Basic Hudi Operations

#### 1.21.3.15.1 Overview

##### Scenarios

This section describes how to use Spark to perform operations such as data insertion, query, update, incremental query, query at a specific time point, and data deletion on Hudi.

For details, see the sample code.

#### Packaging the Project

1. Upload the **user.keytab** and **krb5.conf** files to the server where the client is located.
2. Use the Maven tool provided by IDEA to package the project and generate the JAR file. For details, see [Compiling and Running the Application](#).

##### NOTE

- Before compilation and packaging, change the paths of the **user.keytab** and **krb5.conf** files in the sample code to the actual paths on the client server.
  - The Python sample code does not need to be packaged using Maven.
3. Upload the generated JAR file to any directory (for example, **/opt/example/**) on the server where the Spark client is located.

#### Running tasks

1. Log in to the Spark client node and run the following commands:  
**source Client installation directory/bigdata\_env**  
**source Client installation directory/Hudi/component\_env**  
**kinit Hudi development user**
2. Use the **spark-submit** command to perform the write, update, query, and delete operations in sequence after compiling and building the sample code.
  - Run the Java sample program.  
**spark-submit --keytab <user\_keytab\_path> --principal=<principal\_name> --class com.huawei.bigdata.hudi.examples.HoodieWriteClientExample /opt/example/hudi-java-security-examples-1.0.jar hdfs://hacluster/tmp/example/hoodie\_java hoodie\_java**  
**<user\_keytab\_path>** indicates the authentication file path,  
**<principal\_name>** indicates the authentication user name, **/opt/example/hudi-java-examples-1.0.jar** indicates the JAR file path, **hdfs://hacluster/tmp/example/hoodie\_java** indicates the storage path of the Hudi table, and **hoodie\_java** indicates the name of the Hudi table.

- Run the Scala sample program.

```
spark-submit --keytab <user_keytab_path> --
principal=<principal_name> --class
com.huawei.bigdata.hudi.examples.HoodieDataSourceExample /opt/
example/hudi-scala-security-examples-1.0.jar hdfs://hacluster/tmp/
example/hoodie_scala hoodie_scala
```

`/opt/example/hudi-scala-examples-1.0.jar` indicates the JAR file path, `<user_keytab_path>` indicates the authentication file path, `<principal_name>` indicates the authentication user name, `hdfs://hacluster/tmp/example/hoodie_scala` indicates the storage path of the Hudi table, and `hoodie_Scala` indicates the name of the Hudi table.

- Run the Python sample program.

```
spark-submit /opt/example/HudiPythonExample.py hdfs://
hacluster/tmp/huditest/example/python hudi_trips_cow
```

`hdfs://hacluster/tmp/huditest/example/python` indicates the storage path of the Hudi table, and `hudi_trips_cow` indicates the name of the Hudi table.

### 1.21.3.15.2 Java Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.bigdata.hudi.examples.HoodieWriteClientExample](#).

Create a client object to operate Hudi:

```
String tablePath = args[0];
String tableName = args[1];
SparkConf sparkConf = HoodieExampleSparkUtils.defaultSparkConf("hoodie-client-example");
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
// Generator of some records to be loaded in.
HoodieExampleDataGenerator<HoodieAvroPayload> dataGen = new HoodieExampleDataGenerator<>();
// initialize the table, if not done already
Path path = new Path(tablePath);
FileSystem fs = FSUtils.getFs(tablePath, jsc.hadoopConfiguration());
if (!fs.exists(path)) {
    HoodieTableMetaClient.initTableType(jsc.hadoopConfiguration(), tablePath,
    HoodieTableType.valueOf(tableName),
    tableName, HoodieAvroPayload.class.getName());
}

// Create the write client to write some records in
HoodieWriteConfig cfg = HoodieWriteConfig.newBuilder().withPath(tablePath)
    .withSchema(HoodieExampleDataGenerator.TRIP_EXAMPLE_SCHEMA).withParallelism(2, 2)
    .withDeleteParallelism(2).forTable(tableName)
    .withIndexConfig(HoodieIndexConfig.newBuilder().withIndexType(HoodieIndex.IndexType.BLOOM).build()
)
    .withCompactionConfig(HoodieCompactionConfig.newBuilder().archiveCommitsWith(20,
30).build()).build();
SparkRDDWriteClient<HoodieAvroPayload> client = new SparkRDDWriteClient<>(new
HoodieSparkEngineContext(jsc), cfg);
```

Insert data:

```
String newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
List<HoodieRecord<HoodieAvroPayload>> records = dataGen.generateInserts(newCommitTime, 10);
List<HoodieRecord<HoodieAvroPayload>> recordsSoFar = new ArrayList<>(records);
JavaRDD<HoodieRecord<HoodieAvroPayload>> writeRecords = jsc.parallelize(records, 1);
client.upsert(writeRecords, newCommitTime);
```

Update data:

```

newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
List<HoodieRecord<HoodieAvroPayload>> toBeUpdated = dataGen.generateUpdates(newCommitTime, 2);
records.addAll(toBeUpdated);
recordsSoFar.addAll(toBeUpdated);
writeRecords = jsc.parallelize(records, 1);
client.upsert(writeRecords, newCommitTime);

```

Delete data:

```

newCommitTime = client.startCommit();
LOG.info("Starting commit " + newCommitTime);
// just delete half of the records
int numToDelete = recordsSoFar.size() / 2;
List<HoodieKey> toBeDeleted =
recordsSoFar.stream().map(HoodieRecord::getKey).limit(numToDelete).collect(Collectors.toList());
JavaRDD<HoodieKey> deleteRecords = jsc.parallelize(toBeDeleted, 1);
client.delete(deleteRecords, newCommitTime);

```

Compress data.

```

if (HoodieTableType.valueOf(tableType) == HoodieTableType.MERGE_ON_READ) {
    Option<String> instant = client.scheduleCompaction(Option.empty());
    JavaRDD<WriteStatus> writeStatuses = client.compact(instant.get());
    client.commitCompaction(instant.get(), writeStatuses, Option.empty());
}

```

### 1.21.3.15.3 Scala Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.bigdata.hudi.examples.HoodieDataSourceExample](#).

Insert data:

```

def insertData(spark: SparkSession, tablePath: String, tableName: String, dataGen: HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
val commitTime: String = System.currentTimeMillis().toString
val inserts = dataGen.convertToStringList(dataGen.generateInserts(commitTime, 20))
spark.sparkContext.parallelize(inserts, 2)
val df = spark.read.json(spark.sparkContext.parallelize(inserts, 1)).write.format("org.apache.hudi").
    options(getQuickstartWriteConfigs).
    option(PRECOMBINE_FIELD_OPT_KEY, "ts").
    option(RECORDKEY_FIELD_OPT_KEY, "uuid").
    option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
    option(TABLE_NAME, tableName).
    mode(Overwrite).
    save(tablePath)}

```

Query data.

```

def queryData(spark: SparkSession, tablePath: String, tableName: String, dataGen: HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
val roViewDF = spark.
    read.
    format("org.apache.hudi").
    load(tablePath + "/*/*/*")
roViewDF.createOrReplaceTempView("hudi_ro_table")
spark.sql("select fare, begin_lon, begin_lat, ts from hudi_ro_table where fare > 20.0").show()
// +-----+-----+-----+-----+
// |      fare|    begin_lon|    begin_lat| ts|
// +-----+-----+-----+-----+
// |98.88075495133515|0.39556048623031603|0.17851135255091155|0.0|
// ...
spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from hudi_ro_table").show()
// +-----+-----+-----+-----+-----+
// |_hoodie_commit_time| _hoodie_record_key|_hoodie_partition_path|      rider|
// |driver|      fare|
// +-----+-----+-----+-----+

```

```
+-----+
// | 20191231181501|31cafb9f-0196-4b1...| 2020/01/02|rider-1577787297889|
driver-1577787297889| 98.88075495133515|
// ...
}
```

### Update data:

```
def updateData(spark: SparkSession, tablePath: String, tableName: String, dataGen: HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
  val commitTime: String = System.currentTimeMillis().toString
  val updates = dataGen.convertToStringList(dataGen.generateUpdates(commitTime, 10))
  val df = spark.read.json(spark.sparkContext.parallelize(updates, 1))
  df.write.format("org.apache.hudi").
    options(getQuickstartWriteConfigs).
    option(PRECOMBINE_FIELD_OPT_KEY, "ts").
    option(RECORDKEY_FIELD_OPT_KEY, "uuid").
    option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
    option(TABLE_NAME, tableName).
    mode(Append).
    save(tablePath)}
```

### Incremental query:

```
def incrementalQuery(spark: SparkSession, tablePath: String, tableName: String) {
  import spark.implicits._
  val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by commitTime").map(k => k.getString(0)).take(50)
  val beginTime = commits(commits.length - 2)

  val incViewDF = spark.
    read.
    format("org.apache.hudi").
    option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
    option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
    load(tablePath)
  incViewDF.createOrReplaceTempView("hudi_incr_table")
  spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare > 20.0").show()
```

### Query at a specific time point:

```
def pointInTimeQuery(spark: SparkSession, tablePath: String, tableName: String) {
  import spark.implicits._
  val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by commitTime").map(k => k.getString(0)).take(50)
  val beginTime = "000"
  // Represents all commits > this time.
  val endTime = commits(commits.length - 2)
  // commit time we are interested in
  //incrementally query data
  val incViewDF = spark.read.format("org.apache.hudi").
    option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
    option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
    option(END_INSTANTTIME_OPT_KEY, endTime).
    load(tablePath)
  incViewDF.createOrReplaceTempView("hudi_incr_table")
  spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare > 20.0").show()}
```

## 1.21.3.15.4 Python Example Code

The following code snippets are used as an example. For complete code, see [HudiPythonExample.py](#).

### Insert data:

```
#insert
inserts = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateInserts(10))
```

```
df = spark.read.json(spark.sparkContext.parallelize(inserts, 2))
hudi_options = {
    'hoodie.table.name': tableName,
    'hoodie.datasource.write.recordkey.field': 'uuid',
    'hoodie.datasource.write.partitionpath.field': 'partitionpath',
    'hoodie.datasource.write.table.name': tableName,
    'hoodie.datasource.write.operation': 'insert',
    'hoodie.datasource.write.precombine.field': 'ts',
    'hoodie.upsert.shuffle.parallelism': 2,
    'hoodie.insert.shuffle.parallelism': 2
}
df.write.format("hudi"). \
    options(**hudi_options). \
    mode("overwrite"). \
    save(basePath)
```

### Query data.

```
tripsSnapshotDF = spark. \
    read. \
    format("hudi"). \
    load(basePath + "/*/*/*/*")
tripsSnapshotDF.createOrReplaceTempView("hudi_trips_snapshot")
spark.sql("select fare, begin_lon, begin_lat, ts from hudi_trips_snapshot where fare > 20.0").show()
spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from hudi_trips_snapshot").show()
```

### Update data:

```
updates = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateUpdates(10))
df = spark.read.json(spark.sparkContext.parallelize(updates, 2))
df.write.format("hudi"). \
    options(**hudi_options). \
    mode("append"). \
    save(basePath)
```

### Incremental query:

```
spark. \
    read. \
    format("hudi"). \
    load(basePath + "/*/*/*/*"). \
    createOrReplaceTempView("hudi_trips_snapshot")
incremental_read_options = {
    'hoodie.datasource.query.type': 'incremental',
    'hoodie.datasource.read.begin.instanttime': beginTime,
}
tripsIncrementalDF = spark.read.format("hudi"). \
    options(**incremental_read_options). \
    load(basePath)
tripsIncrementalDF.createOrReplaceTempView("hudi_trips_incremental")
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_incremental where fare > 20.0").show()
```

### Query at a specific time point:

```
# Represents all commits > this time.
beginTime = "000"
endTime = commits[len(commits) - 2]
point_in_time_read_options = {
    'hoodie.datasource.query.type': 'incremental',
    'hoodie.datasource.read.end.instanttime': endTime,
    'hoodie.datasource.read.begin.instanttime': beginTime
}
tripsPointInTimeDF = spark.read.format("hudi"). \
    options(**point_in_time_read_options). \
    load(basePath)
tripsPointInTimeDF.createOrReplaceTempView("hudi_trips_point_in_time")
```

```
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_point_in_time where fare > 20.0").show()
```

Delete data:

```
# Obtain the total number of records.  
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()  
# Obtain two records to be deleted.  
ds = spark.sql("select uuid, partitionpath from hudi_trips_snapshot").limit(2)  
# Delete the records.  
hudi_delete_options = {  
    'hoodie.table.name': tableName,  
    'hoodie.datasource.write.recordkey.field': 'uuid',  
    'hoodie.datasource.write.partitionpath.field': 'partitionpath',  
    'hoodie.datasource.write.table.name': tableName,  
    'hoodie.datasource.write.operation': 'delete',  
    'hoodie.datasource.write.precombine.field': 'ts',  
    'hoodie.upsert.shuffle.parallelism': 2,  
    'hoodie.insert.shuffle.parallelism': 2  
}  
from pyspark.sql.functions import lit  
deletes = list(map(lambda row: (row[0], row[1]), ds.collect()))  
df = spark.sparkContext.parallelize(deletes).toDF(['uuid', 'partitionpath']).withColumn('ts', lit(0.0))  
df.write.format("hudi"). \  
    options(**hudi_delete_options). \  
    mode("append"). \  
    save(basePath)  
# Perform the query in the same way.  
roAfterDeleteViewDF = spark. \  
    read. \  
    format("hudi"). \  
    load(basePath + "/*/*/*")  
roAfterDeleteViewDF.registerTempTable("hudi_trips_snapshot")  
# Return (total - 2) records.  
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()  
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").show()
```

### 1.21.3.16 Compiling User-defined Configuration Items for Hudi

#### 1.21.3.16.1 HoodieDeltaStreamer

Compile a user-defined conversion class for **Transformer**.

Compile a user-defined schema for **SchemaProvider**.

Add the following parameters when running **HoodieDeltaStreamer**:

```
--schemaprovider-class Defined schema class --transformer-class Defined transform class
```

Examples of **Transformer** and **SchemaProvider**:

```
public class TransformerExample implements Transformer, Serializable {  
    @Override  
    public Dataset<Row> apply(JavaSparkContext jsc, SparkSession sparkSession, Dataset<Row> rowDataset,  
        TypedProperties properties) {  
        JavaRDD<Row> rowJavaRdd = rowDataset.toJavaRDD();  
        List<Row> rowList = new ArrayList<>();  
        for(Row row: rowJavaRdd.collect()){  
            rowList.add(buildRow(row));  
        }  
        JavaRDD<Row> stringJavaRdd = jsc.parallelize(rowList);  
        List<StructField> fields = new ArrayList<>();  
        buildFields(fields);  
        StructType schema = DataTypes.createStructType(fields);  
        Dataset<Row> dataFrame = sparkSession.createDataFrame(stringJavaRdd, schema);  
        return dataFrame;  
    }  
}
```

```

    }

    private void buildFields(List<StructField> fields) {
        fields.add(DataTypes.createStructField("age", DataTypes.StringType, true));
        fields.add(DataTypes.createStructField("id", DataTypes.StringType, true));
        fields.add(DataTypes.createStructField("name", DataTypes.StringType, true));
        fields.add(DataTypes.createStructField("job", DataTypes.StringType, true));
    }

    private Row buildRow(Row row){
        String age = row.getString(0);
        String id = row.getString(1);
        String job = row.getString(2);
        String name = row.getString(3);
        Row returnRow = RowFactory.create(age, id, job, name);
        return returnRow;
    }
}

public class DataSchemaProviderExample extends SchemaProvider {

    public DataSchemaProviderExample(TypedProperties props, JavaSparkContext jssc) {
        super(props, jssc);
    }

    @Override
    public Schema getSourceSchema() {
        Schema avroSchema = new Schema.Parser().parse(
            "{\"type\":\"record\",\"name\":\"hoodie_source\",\"fields\":[{\"name\":\"age\",\"type\":\"string\"},\"name\":\"id\",\"type\":\"string\"},{\"name\":\"job\",\"type\":\"string\"},{\"name\":\"name\",\"type\":\"string\"}]}");
        return avroSchema;
    }

    @Override
    public Schema getTargetSchema() {
        Schema avroSchema = new Schema.Parser().parse(
            "{\"type\":\"record\",\"name\":\"mytest_record\",\"namespace\":\"hoodie.mytest\",\"fields\":[{\"name\":\"age\",\"type\":\"string\"},{\"name\":\"id\",\"type\":\"string\"},{\"name\":\"job\",\"type\":\"string\"},{\"name\":\"name\",\"type\":\"string\"}]}");
        return avroSchema;
    }
}

```

### 1.21.3.16.2 User-defined Partitioner

Compile a user-defined partitioner class that inherits **BulkInsertPartitioner** and add the following configuration when writing data to Hudi:

```
.option(BULKINSERT_USER_DEFINED_PARTITIONER_CLASS, <User-defined partitioner class package name + class name>)
```

Example of the user-defined partitioner:

```

public class HoodieSortExample<T extends HoodieRecordPayload>
    implements BulkInsertPartitioner<JavaRDD<HoodieRecord<T>>> {
    @Override
    public JavaRDD<HoodieRecord<T>> repartitionRecords(JavaRDD<HoodieRecord<T>> records, int
    outputSparkPartitions) {
        JavaPairRDD<String,
        HoodieRecord<T>> stringHoodieRecordJavaPairRDD = records.coalesce(outputSparkPartitions)
            .mapToPair(record -> new Tuple2<>(new StringBuilder().append(record.getPartitionPath())
                .append("+")
                .append(record.getRecordKey())
                .toString(), record));
        JavaRDD<HoodieRecord<T>> hoodieRecordJavaRDD =
        stringHoodieRecordJavaPairRDD.mapPartitions(partition -> {
            List<Tuple2<String, HoodieRecord<T>>> recordList = new ArrayList<>();

```

```
        for (; partition.hasNext();) {
            recordList.add(partition.next());
        }
        Collections.sort(recordList, (o1, o2) -> {
            if (o1._1().split("[+])[0] == o2._1().split("[+])[0]) {
                return Integer.parseInt(o1._1().split("[+])[1]) - Integer.parseInt(o2._1().split("[+])[1]);
            } else {
                return o1._1().split("[+])[0].compareTo(o2._1().split("[+])[0]);
            }
        });
        return recordList.stream().map(e -> e._2).iterator();
    });

    @Override
    public boolean arePartitionRecordsSorted() {
        return true;
    }
}
```

### 1.21.3.17 Using Spark to Write Data to ClickHouse

#### 1.21.3.17.1 Overview

##### Scenarios

In Spark applications, users can use the native APIs of ClickHouse JDBC and the Spark JDBC driver to create, query, and insert ClickHouse databases and tables.

##### Development Guidelines

Use the ClickHouse JDBC driver to create databases and tables, and insert data. Then, call the Spark JDBC APIs to read data from the ClickHouse table, convert the data, and write the converted data to the ClickHouse table.

The process is as follows:

- Create a ClickHouse database and table, and insert data into the table.
- Use the Spark JDBC API to read data from the ClickHouse table.
- Register a temporary table, process the field ID in the table, and return a new data set.
- Append the new data set to the ClickHouse table.

##### Preparations

- Prepare user information, including the username and password. The user must have the ClickHouse administrator rights.
- Prepare the names of ClickHouse database and table to be created. The database and table names must be different from the existing ones. In addition, you need to create a ClickHouse logical cluster in advance. For details, see "Creating a User-defined ClickHouse Logical Cluster" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1). For example, creating logical cluster **default\_cluster**.

## Packaging the Project

1. Use the Maven tool provided by IDEA to package the project and generate the JAR file.
2. Upload the generated JAR file to the Spark client directory, for example, **/opt/hadoopclient/Spark/spark**.
3. ClickHouse JDBC driver package **clickhouse-jdbc\*.jar** is automatically downloaded to the local Maven library during JAR file packaging and compilation. Upload the package to the Spark client directory on the server, for example, **/opt/hadoopclient/Spark/spark**. The JAR package version in the following steps is only an example.

## Running tasks

Go to the Spark client directory, source environment variables, and perform user authentication. Assume that the following commands are executed in the **spark** directory of the Spark client. Call the **bin/spark-submit** script to run the code. The running commands are as follows (the class name and file name must be consistent with those in the actual code. The following is only an example):

- Run the Java sample program.  
**bin/spark-submit --master yarn --deploy-mode client/cluster --jars clickhouse-jdbc-0.3.1-h0.cbu.mrs.313.r1-SNAPSHOT.jar --class com.huawei.bigdata.spark.examples.SparkOnClickHouseExample SparkOnClickHouseJavaExample-1.0.jar <jdbcurl> <clickHouseDBName> <clickHouseTableName> <userName> <password>**
- Run the Scala sample program.  
**bin/spark-submit --master yarn --deploy-mode client/cluster --jars clickhouse-jdbc-0.3.1-h0.cbu.mrs.313.r1-SNAPSHOT.jar --class com.huawei.bigdata.spark.examples.SparkOnClickHouseExample SparkOnClickHouseScalaExample-1.0.jar <jdbcurl> <clickHouseDBName> <clickHouseTableName> <userName> <password>**
- Run the Python sample program.  
**bin/spark-submit --master yarn --deploy-mode client/cluster --jars SparkOnClickHousePythonExample-1.0.jar,clickhouse-jdbc-0.3.1-h0.cbu.mrs.313.r1-SNAPSHOT.jar SparkOnClickHousePythonExample.py <jdbcurl> <clickHouseDBName> <clickHouseTableName> <userName> <password>**

### NOTE

- **jdbcurl** is the ClickHouse JDBC connection string, for example, **jdbc:clickhouse://{ip}:{port}**. To query the port number, perform the following steps:  
Log in to FusionInsight Manager, choose **Cluster > Services > ClickHouse**, click **Logic Cluster**, and obtain the port number from the **HTTP Balancer Port** column in the cluster list.
- **clickHouseDBName** is the name of the ClickHouse database to be created, for example, **ckdb001**.
- **clickHouseTableName** is the name of the ClickHouse table to be created, for example, **ckdb01**.
- **userName** is the username, and **password** is the corresponding password.

### 1.21.3.17.2 Java Sample Code

The following code snippets are used as an example. For complete code, see:  
<com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.java>.

## Prerequisites

Create a user, user group (**hadoop** and **hive**), and primary group (**hadoop**) with the ClickHouse permission. For example, create user **sparkuser**.

## Procedure

**Step 1** Prepare the spark-clickhouse sample code. For details, see [Overview](#).

**Step 2** Import the sample project to IDEA.

**Step 3** Create a table and insert data using the ClickHouse JDBC API.

```
private static void clickHouseExecute(ClickHouseStatement ckStatement, String clickHouseDB, String clickHouseTable) throws SQLException {
    String createDB = "CREATE DATABASE " + clickHouseDB + " ON CLUSTER default_cluster";
    String createTable = "CREATE TABLE " + clickHouseDB + "." + clickHouseTable + " ON CLUSTER
default_cluster" +
        "('EventDate` DateTime,`id` UInt64,`name` String,`age` UInt8,`address` String)" +
        "ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}'" + clickHouseDB + "/" + clickHouseTable
+ "", '{replica}')"
        + "PARTITION BY toYYYYMM(EventDate) ORDER BY id";
    String insertData = "insert into " + clickHouseDB + "." + clickHouseTable + " (id, name, age, address)
values (1, 'zhangsan', 17, 'xian'), (2, 'lisi', 36, 'beijing')";
    ckStatement.execute(createDB);
    ckStatement.execute(createTable);
    ckStatement.execute(insertData);
}
```

**Step 4** Package the code and upload it to the Spark client directory, for example, **/opt/hadoopclient/Spark/spark**.

**Step 5** Go to the client directory and run the following commands:

```
cd Cluster client installation directory
source bigdata_env
source Spark/component_env
kinit sparkuser
```

**Step 6** Run the task.

```
dbName: ckdb001
dbName: ckdb003
dbName: ckdb005
dbName: ckdb01
dbName: ckdb02
dbName: ckdb03
dbName: default
dbName: system
dbName: test
[Elapsed]770
2022-03-19 09:08:14,970 | WARN  | main | The enable mv value "null" is : | org.apache.carbondata.core.util.CarbonProperties.validateEnableMV(Carb
2022-03-19 09:08:15,002 | WARN  | main | The value "LOCALLOCK" configure r current file system. Use the default value HDFSLOCK instead. | org.apa
s.validateAndConfigureLockType(CarbonProperties.java:444)
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 1|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 2| lisi| 36| [REDACTED]
+-----+-----+-----+
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 21|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 22| lisi| 36| [REDACTED]
```

- Step 7** View the running result on the ClickHouse client. Assume that the created ClickHouse database is **ckdb0001** and the table is **cktbo1**.

```
clickhouse client --host IP address of the ClickHouse instance --port 21427 --secure
```

```
:) select * from ckdb0001.cktbo1;

SELECT *
FROM ckdb0001.cktbo1

Query id: 0fd090e8-c81c-4752-9a4d-6 [REDACTED] 3e

+-----+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 21|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 22| lisi| 36| [REDACTED]
+-----+-----+-----+
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 1|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 2| lisi| 36| [REDACTED]

4 rows in set. Elapsed: 0.004 sec.
```

----End

### 1.21.3.17.3 Scala Sample Code

The following code snippets are used as an example. For complete code, see:  
<com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.Scala>.

## Prerequisites

Create a user, user group (**hadoop** and **hive**), and primary group (**hadoop**) with the ClickHouse permission. For example, create user **sparkuser**.

## Procedure

**Step 1** Prepare the spark-clickhouse sample code. For details, see [Overview](#).

**Step 2** Import the sample project to IDEA.

**Step 3** Create a table and insert data using the ClickHouse JDBC API.

```
def clickHouseExecute(ckStatement: ClickHouseStatement, clickHouseDB: String, clickHouseTable: String) {  
    val createDB = s"CREATE DATABASE ${clickHouseDB} ON CLUSTER default_cluster"  
    val createTable = s"CREATE TABLE ${clickHouseDB}.${clickHouseTable} ON CLUSTER default_cluster "  
    +"(`EventDate` DateTime, `id` UInt64, `name` String, `age` UInt8, `address` String)" +s"ENGINE =  
    ReplicatedMergeTree('/clickhouse/tables/{shard}/${clickHouseDB}/${clickHouseTable}', '{replica}')"  
    +"PARTITION BY toYYYYMM(EventDate) ORDER BY id"  
    val insertData = s"insert into ${clickHouseDB}.${clickHouseTable} (id, name, age, address) values (1,  
    'zhangsan', 17, 'xian'), (2, 'lisi', 36, 'beijing')"  
    executeSqlText(ckStatement, createDB)  
    executeSqlText(ckStatement, createTable)  
    executeSqlText(ckStatement, insertData)  
}
```

**Step 4** Package the code and upload it to the Spark client directory, for example, **/opt/hadoopclient/Spark/spark**.

**Step 5** Go to the client directory and run the following commands:

**cd** *Cluster client installation directory*

**source** **bigdata\_env**

**source** **Spark/component\_env**

**kinit** **sparkuser**

**Step 6** Run the task.

```
dbName: ckdb001  
dbName: ckdb003  
dbName: ckdb005  
dbName: ckdb01  
dbName: ckdb02  
dbName: ckdb03  
dbName: default  
dbName: system  
dbName: test  
[Elapsed]770  
2022-03-19 09:08:14,970 | WARN  | main | The enable mv value "null" is i  
| org.apache.carbondata.core.util.CarbonProperties.validateEnableMV(Carb  
2022-03-19 09:08:15,002 | WARN  | main | The value "LOCALLOCK" configures  
r current file system. Use the default value HDFSLOCK instead. | org.apa  
s.validateAndConfigureLockType(CarbonProperties.java:444)  
+-----+-----+-----+-----+  
|     EventDate| id|   name|age|address|  
+-----+-----+-----+-----+  
|1970-01-01 08:00:00| 1|zhangsan| 17|  
|1970-01-01 08:00:00| 2| lisi| 36|  
+-----+-----+-----+-----+  
  
+-----+-----+-----+-----+  
|     EventDate| id|   name|age|address|  
+-----+-----+-----+-----+  
|1970-01-01 08:00:00| 21|zhangsan| 17|  
|1970-01-01 08:00:00| 22| lisi| 36|  
+-----+-----+-----+-----+
```

**Step 7** View the running result on the ClickHouse client. Assume that the created ClickHouse database is **ckdb0001** and the table is **cktb01**.

**clickhouse client --host** *IP address of the ClickHouse instance* **--port** 21427 **--secure**

```
:) select * from ckdb0001.ctb01;
SELECT *
FROM ckdb0001.ctb01
Query id: 0fd090e8-c81c-4752-9a4d-3e
+-----+-----+-----+-----+-----+
| EventDate | id | name | age | address |
+-----+-----+-----+-----+-----+
| 1970-01-01 08:00:00 | 21 | zhangsan | 17 | [REDACTED] |
| 1970-01-01 08:00:00 | 22 | lisi | 36 | [REDACTED] |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| EventDate | id | name | age | address |
+-----+-----+-----+-----+-----+
| 1970-01-01 08:00:00 | 1 | zhangsan | 17 | [REDACTED] |
| 1970-01-01 08:00:00 | 2 | lisi | 36 | [REDACTED] |
+-----+-----+-----+-----+-----+
4 rows in set. Elapsed: 0.004 sec.
```

----End

#### 1.21.3.17.4 Python Sample Code

The following code snippets are used as an example. For complete code, see:  
[com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.java](https://github.com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.java).

### Prerequisites

Create a user, user group (**hadoop** and **hive**), and primary group (**hadoop**) with the ClickHouse permission. For example, create user **sparkuser**.

### Procedure

**Step 1** Prepare the spark-clickhouse sample code. For details, see [Overview](#).

**Step 2** Import the sample project to IDEA.

**Step 3** Create a table and insert data using the ClickHouse JDBC API.

```
private static void clickHouseExecute(ClickHouseStatement ckStatement, String clickHouseDB, String clickHouseTable) throws SQLException {
    String createDB = "CREATE DATABASE " + clickHouseDB + " ON CLUSTER default_cluster";
    String createTable = "CREATE TABLE " + clickHouseDB + "." + clickHouseTable + " ON CLUSTER default_cluster" +
        "(EventDate` DateTime, `id` UInt64, `name` String, `age` UInt8, `address` String)" +
        "ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}' /" + clickHouseDB + "/" + clickHouseTable +
        "+ "", '{replica}')"
    String insertData = "insert into " + clickHouseDB + "." + clickHouseTable + " (id, name, age, address)" +
        "values (1, 'zhangsan', 17, 'xian'), (2, 'lisi', 36, 'beijing')";
    ckStatement.execute(createDB);
    ckStatement.execute(createTable);
    ckStatement.execute(insertData);
}
```

**Step 4** Package the code and upload it to the Spark client directory, for example, `/opt/hadoopclient/Spark/spark`.

**Step 5** Go to the client directory and run the following commands:

```
cd Cluster client installation directory  
source bigdata_env  
source Spark/component_env  
kinit sparkuser
```

**Step 6** Run the task.

```
dbName: ckdb001  
dbName: ckdb003  
dbName: ckdb005  
dbName: ckdb01  
dbName: ckdb02  
dbName: ckdb03  
dbName: default  
dbName: system  
dbName: test  
[Elapsed]770  
2022-03-19 09:08:14,970 | WARN  | main | The enable_mv value "null" is i  
| org.apache.carbondata.core.util.CarbonProperties.validateEnableMV(Carb  
2022-03-19 09:08:15,002 | WARN  | main | The value "LOCALLOCK" configures  
r current file system. Use the default value HDFSLOCK instead. | org.apa  
s.validateAndConfigureLockType(CarbonProperties.java:444)  
+-----+-----+-----+-----+  
| EventDate| id| name|age|address|  
+-----+-----+-----+-----+  
|1970-01-01 08:00:00| 1|zhangsan| 17|  
|1970-01-01 08:00:00| 2| lisi| 36||  
+-----+-----+-----+-----+  
  
+-----+-----+-----+-----+  
| EventDate| id| name|age|address|  
+-----+-----+-----+-----+  
|1970-01-01 08:00:00| 21|zhangsan| 17|  
|1970-01-01 08:00:00| 22| lisi| 36||  
+-----+-----+-----+-----+
```

**Step 7** View the running result on the ClickHouse client. Assume that the created ClickHouse database is `ckdb0001` and the table is `cktb01`.

```
clickhouse client --host IP address of the ClickHouse instance --port 21427 --  
secure
```

```
:) select * from ckdb0001.cktb01;

SELECT *
FROM ckdb0001.cktb01

Query id: 0fd090e8-c81c-4752-9a4d-63e

+-----+-----+-----+-----+-----+
| EventDate | id | name | age | address |
+-----+-----+-----+-----+-----+
| 1970-01-01 08:00:00 | 21 | zhangsan | 17 |          |
| 1970-01-01 08:00:00 | 22 | lisi | 36 |          |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| EventDate | id | name | age | address |
+-----+-----+-----+-----+-----+
| 1970-01-01 08:00:00 | 1 | zhangsan | 17 |          |
| 1970-01-01 08:00:00 | 2 | lisi | 36 |          |
+-----+-----+-----+-----+-----+

4 rows in set. Elapsed: 0.004 sec.
```

----End

### 1.21.3.17.5 SQL Example

Spark allows you to use SQL statements to create a Spark table and associate it with the ClickHouse table. When running the **Spark SQL CLI** command, you need to add **--jars clickhouse-jdbc-xxxjar**. For details about the actual JAR package name, see [Using Spark to Write Data to ClickHouse](#). The syntax for creating a table is as follows:

```
create table spark_cktb01 using org.apache.spark.sql.jdbc options(url
"url", ssl "true", isCheckConnection "true", sslMode "none", driver
"ru.yandex.clickhouse.ClickHouseDriver", dbtable
"clickHouseDBName.clickHouseTableName", user "userName", password
"password");
```

#### NOTE

- **jdbcurl** is the ClickHouse JDBC connection string, for example, **jdbc:clickhouse://{IP address}:{Port number}**. The port number is **21426**.
- **clickHouseDBName** is the name of the ClickHouse database to be created, for example, **ckdb001**.
- **clickHouseTableName** is the name of the ClickHouse table to be created, for example, **cktb01**.
- **userName** is the username.
- **password** is the password of the user.

After the ClickHouse table is created, you can read, write, and query the **spark\_cktb01** table to perform the same operations on the ClickHouse table.

## 1.21.4 Commissioning the Application

### 1.21.4.1 Commissioning Applications on Windows

### 1.21.4.1.1 Compiling and Running Applications

#### Scenario

You can run applications in the Windows environment after application code development is complete. The procedures for running applications developed using Scala or Java are the same on IDEA.

##### NOTE

- In the Windows environment, only the sample code for accessing Spark SQL using JDBC is provided.
- Ensure that the Maven image repository of the SDK in the Huawei image site has been configured for Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

#### Procedure

##### Step 1 Obtain the sample code.

Download the Maven project source code and configuration file of the sample project. For details, see [Obtaining Sample Projects](#).

Import the sample code to IDEA.

##### Step 2 Obtain configuration files.

1. Obtain the files from the cluster client. Download the **hive-site.xml** and **spark-defaults.conf** files from **\$SPARK\_HOME/conf** to a local directory.
2. On the FusionInsight Manager page of the cluster, download the user authentication file to a local directory.

##### Step 3 Upload data to HDFS.

1. Create a data text file on Linux and save the following data to the data file:

```
Miranda,32
Karlie,23
Candice,27
```
2. On the HDFS client, run the following commands for authentication:  
**cd {Client installation directory}**  
**kinit <Service user for authentication>**
3. On the HDFS client running the Linux OS, run the **hadoop fs -mkdir /data** command (or the **hdfs dfs** command) to create a directory.
4. On the HDFS client running the Linux OS, run the **hadoop fs -put data /data** command to upload the data file.

##### Step 4 Configure related parameters in the sample code.

1. Configure the authentication information.  
Set **userPrincipal** to the username.  
Set **userKeytabPath** to the path of the downloaded **keytab** file.  
Set **Krb5ConfPath** to the path of the downloaded **krb5.conf** file.

```

public class ThriftServerQueriesTest {
    public static void main(String[] args) throws SQLException, ClassNotFoundException, IOException {
        String userPrincipal = "admin@test";
        String userKeytabPath = "D:\\conf\\keytab\\user.keytab";
        String krb5ConfPath = "D:\\conf\\keytab\\krb5.conf";
        String principalName = KerberosUtil.DEFAULT_REALM;
        String ZKServerPrincipal = "zookeeper/hadoop." + principalName;

        String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";
        String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";
    }
}

```

Set the domain name to **DEFAULT\_REALM**. In the **KerberosUtil** class, change **DEFAULT\_REALM** to the domain name of the cluster.

```

public class KerberosUtil {
    private static Logger logger = Logger.getLogger(KerberosUtil.class);

    public static final String JAVA_VENDOR = "java.vendor";
    public static final String IBM_FLAG = "IBM";
    public static final String CONFIG_CLASS_FOR_IBM = "com.ibm.security.krb5.internal.Config";
    public static final String CONFIG_CLASS_FOR_SUN = "sun.security.krb5.Config";
    public static final String METHOD_GET_INSTANCE = "getInstance";
    public static final String METHOD_GET_DEFAULT_REALM = "getDefaultValue";
    public static final String DEFAULT_REALM = "HADOOP.COM";
}

```

2. Change **user.principal** and **user.keytab** in the string concatenated by **securityConfig** to the corresponding username and path. Note that the path of the **keytab** file must use slashes (/).

```

String securityConfig = "saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop." + principalName + "@"
    + principalName + ";user.principal=admin@test;user.keytab=D:/conf/keytab/user.keytab;";

```

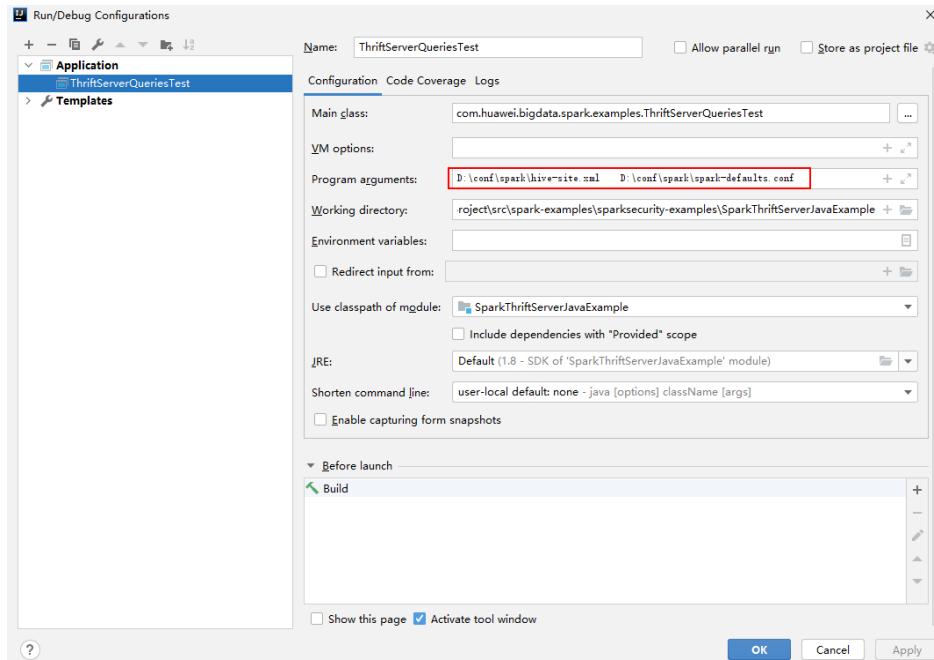
3. Change the SQL statement for loading data to **LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD**.

```

ArrayList<String> sqlList = new ArrayList<>();
sqlList.add("CREATE TABLE IF NOT EXISTS child (name STRING, age INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY"
    + " ' '");
sqlList.add("LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE child");
sqlList.add("SELECT * FROM child");
sqlList.add("DROP TABLE child");
executeSql(url, sqlList);

```

**Step 5** Add running parameters to the **hive-site.xml** and **spark-defaults.conf** files when the application is running.



**Step 6** Run the application.

----End

#### 1.21.4.1.2 View Debugging Results

```
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/apache/logging/log4j/log4j-slf4j-impl/2.6.2/log4j-slf4j-  
impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/slf4j/slf4j-log4j12/1.7.30/slf4j-log4j12-1.7.30.jar!/org/  
slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]  
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to  
the console.  
---- Begin executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT  
DELIMITED FIELDS TERMINATED BY ',' ----  
Result  
---- Done executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT  
DELIMITED FIELDS TERMINATED BY ',' ----  
---- Begin executing sql: LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD ----  
Result  
---- Done executing sql: LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD ----  
---- Begin executing sql: SELECT * FROM child ----  
NAME AGE  
Miranda 32  
Kartie 23  
Candice 27  
---- Done executing sql: SELECT * FROM child ----  
---- Begin executing sql: DROP TABLE child ----  
Result  
---- Done executing sql: DROP TABLE child ----  
Process finished with exit code 0
```

#### 1.21.4.2 Commissioning an Application in Linux

### 1.21.4.2.1 Compiling and Running the Application

#### Scenario

After the program codes are developed, you can upload the codes to the Linux client for running. The running procedures of applications developed in Scala or Java are the same.

##### NOTE

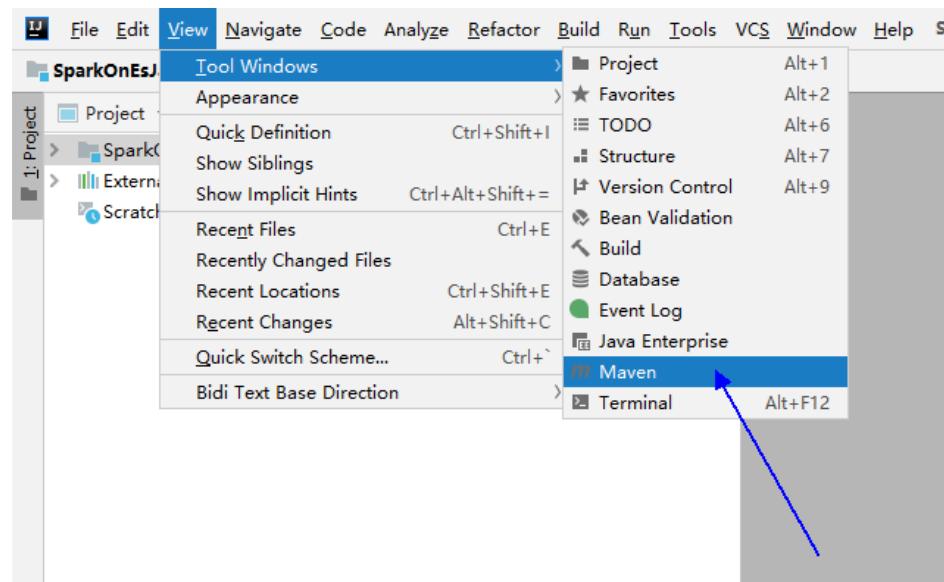
- The Spark application developed in Python does not need to build Artifacts as a jar. You just need to copy the sample projects to the compiler.
- It is needed to ensure that the version of Python installed on the worker and driver is consistent, otherwise the following error will be reported: "Python in worker has different version %s than that in driver %s."
- Ensure that Maven image repository of the SDK in the Huawei image site has been configured in Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

#### Procedure

##### Step 1 In the IntelliJ IDEA, open the Maven tool window.

On the main page of the IDEA, choose **View > Tool Windows > Maven** to open the Maven tool window.

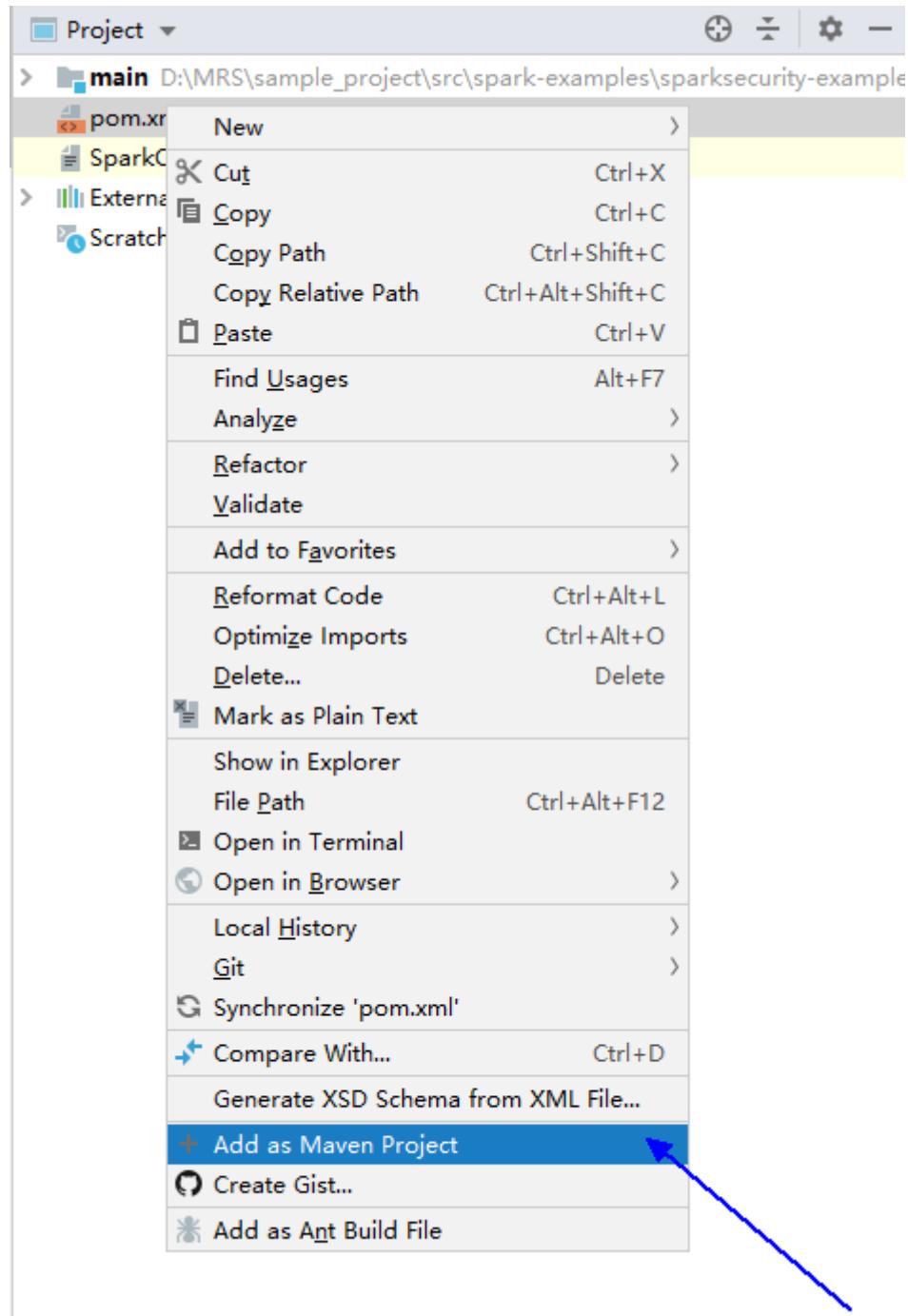
**Figure 1-295** Opening the Maven tool window



If the project is not imported using Maven, perform the following operations:

Right-click the **pom** file in the sample code project and choose **Add as Maven Project** from the shortcut menu to add a Maven project.

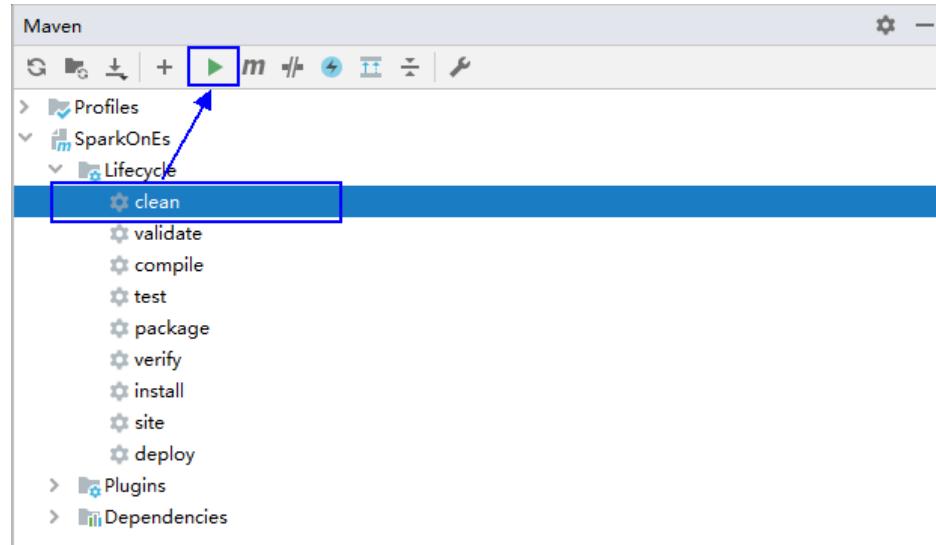
Figure 1-296 Adding a Maven project



**Step 2** Use Maven to generate a JAR file.

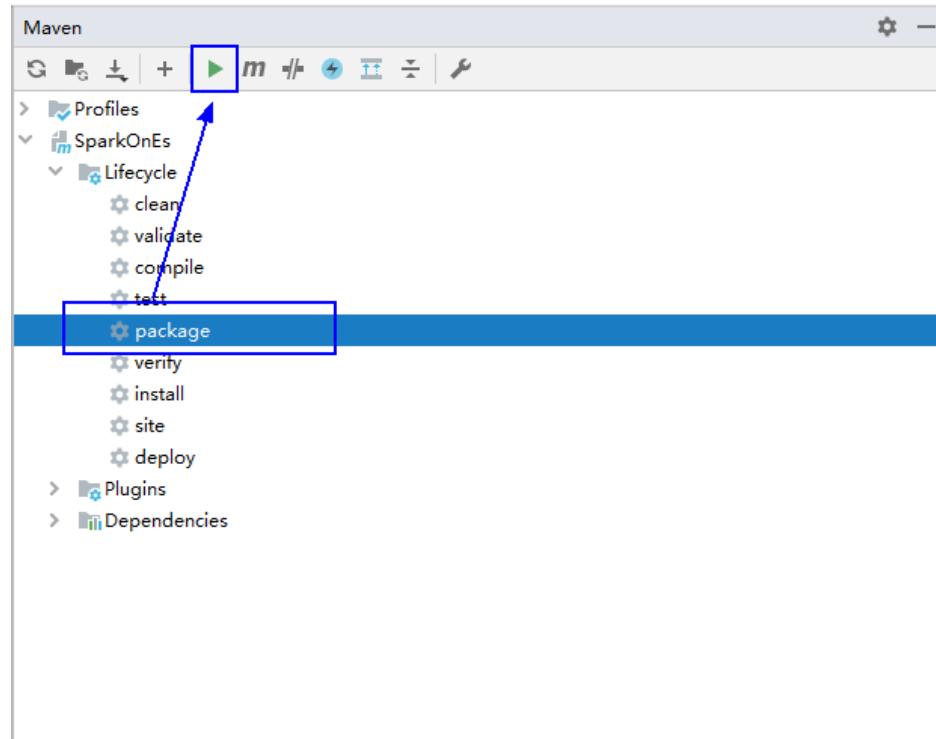
1. In the Maven tool window, select **clean** from **Lifecycle** to execute the Maven building process.

**Figure 1-297** Selecting **clean** from **Lifecycle** and execute the Maven building process



2. In the Maven tool window, select **package** from **Lifecycle** and execute the Maven building process.

**Figure 1-298** Selecting **package** from **Lifecycle** and executing the Maven build process



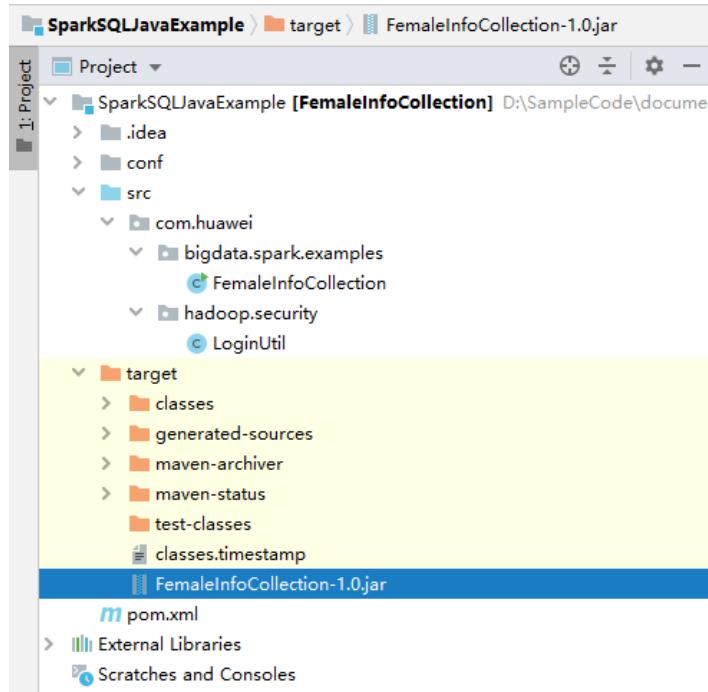
If the following information is displayed in **Run:**, the packaging is successful.

**Figure 1-299** Packaging success message

```
[INFO] [INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FemaleInfoCollection ---
[INFO] [INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ FemaleInfoCollection ---
[INFO] Building jar: D:\SampleCode\document\VT\code\sparksecurity-examples\SparkSQLJavaExample\target\FemaleInfoCollection-1.0.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 19.427 s
[INFO] Finished at: 2020-09-21T11:17:31+08:00
[INFO] -----
```

3. You can obtain the JAR package from the target folder in the project directory.

**Figure 1-300** Obtaining the JAR Package



**Step 3** Copy the JAR file generated in **Step 2** (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. Run the Spark application. For details about the example application, see [Developing the Project](#).

**⚠ CAUTION**

Do not restart the HDFS service or all DataNode instances during Spark job running. Otherwise, the job may fail and some JobHistory data may be lost.

----End

#### 1.21.4.2.2 Checking the Commissioning Result

##### Scenario

After a Spark application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the Spark web UI.
- Viewing Spark logs.

## Procedure

- **Check the operating result data of the Spark application.**  
The data storage directory and format are specified by users in the Spark application. You can obtain the data in the specified file.
- **Check the status of the Spark application.**  
The Spark contains the following two web UIs:
  - The Spark UI displays the status of applications being executed.  
The Spark UI contains the **Spark Jobs**, **Spark Stages**, **Storage**, **Environment**, and **Executors** parts. Besides these parts, **Streaming** is displayed for the Streaming application.  
Access to the interface: On the web UI of the YARN, find the corresponding Spark application, and click **ApplicationMaster** in the last column of the application information. The Spark UI page is displayed.
  - The History Server UI displays the status of all Spark applications.  
The History Server UI displays information such as the application ID, application name, start time, end time, execution time, and user to whom the application belongs. After the application ID is clicked, the Spark UI of the application is displayed.
- **View Spark logs to learn application running conditions.**  
The logs of Spark offer immediate visibility into application running conditions. You can adjust application programs based on the logs. Log related information can be referenced to "Component Operation Guide" > "Using Spark" > "Spark Logs" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

## 1.21.5 More Information

### 1.21.5.1 Common APIs

#### 1.21.5.1.1 Java

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see *MapReduce Service (MRS) 3.3.1-LTS Manager Rest API Reference (for Huawei Cloud Stack 8.3.1)*.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- **JavaSparkContext**: external interface of Spark, which is used to provide the functions of Spark for Java applications that invoke this class, for example,

connecting Spark clusters and generating RDDs, accumulations, and broadcasts. It functions as a container.

- **SparkConf:** Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- **JavaRDD:** class used to define the JavaRDD in the Java application, which functions like the RDD(Resilient Distributed Dataset) class of Scala.
- **JavaPairRDD:** indicates the JavaRDD in the key-value format. This class provides methods such as groupByKey and reduceByKey.
- **Broadcast:** broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- **StorageLevel:** data storage levels, including memory (MEMORY\_ONLY), disk (DISK\_ONLY), and memory+disk (MEMORY\_AND\_DISK).

The JavaRDD supports two types of operations, transformation and action. [Table 1-181](#) and [Table 1-182](#) show the common methods.

**Table 1-181** Transformation

| Method                                                                             | Description                                                                                                  |
|------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| <R> JavaRDD<R><br>map(Function<T,R> f)                                             | Returns a new RDD by applying a function to all elements of this RDD.                                        |
| JavaRDD<T><br>filter(Function<T,Boolean> f)                                        | Invokes Function on all elements of the RDD and returns the element that is <b>true</b> .                    |
| <U> JavaRDD<U><br>flatMap(FlatMapFunction<T,U> f)                                  | Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results. |
| JavaRDD<T><br>sample(boolean<br>withReplacement,<br>double fraction, long<br>seed) | Returns a sampled subset of this RDD.                                                                        |
| JavaRDD<T><br>distinct(int<br>numPartitions)                                       | Returns a new RDD containing the distinct elements in this RDD.                                              |
| JavaPairRDD<K,Iterable<V>><br>groupByKey(int<br>numPartitions)                     | Returns (K,Seq[V]) and combines the values of the same key to a set.                                         |
| JavaPairRDD<K,V><br>reduceByKey(Function<br>2<V,V> func, int<br>numPartitions)     | Invokes Function on the values of the same key.                                                              |
| JavaPairRDD<K,V><br>sortByKey(boolean<br>ascending, int<br>numPartitions)          | Sorts by key in ascending or descending order.<br>Ascending is of the boolean type.                          |

| Method                                                                                                      | Description                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JavaPairRDD<K,scala.Tuple2<V,W>><br>join(JavaPairRDD<K,W> other)                                            | Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numTasks</b> indicates the number of concurrent tasks.                                 |
| JavaPairRDD<K,scala.Tuple2<Iterable<V>,Iterable<W>>><br>cogroup(JavaPairRDD <K,W> other, int numPartitions) | Returns the dataset of <K,scala.Tuple2<Iterable<V>,Iterable<W>>> when the (K,V) and (K,W) datasets exist. <b>numTasks</b> indicates the number of concurrent tasks. |
| JavaPairRDD<T,U><br>cartesian(JavaRDDLike<U,?> other)                                                       | Returns the Cartesian product of the RDD and other RDDs.                                                                                                            |

**Table 1-182 Action**

| Method                                                                                                               | Description                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| T<br>reduce(Function2<T,T,<br>T> f)                                                                                  | Invokes Function2 on elements of the RDD.                                                                                                                 |
| java.util.List<T><br>collect()                                                                                       | Returns an array that contains all of the elements in this RDD.                                                                                           |
| long count()                                                                                                         | Returns the number of elements in the dataset.                                                                                                            |
| T first()                                                                                                            | Returns the first element in the dataset.                                                                                                                 |
| java.util.List<T><br>take(int num)                                                                                   | Returns the first <i>n</i> elements in this RDD.                                                                                                          |
| java.util.List<T><br>takeSample(boolean<br>withReplacement, int<br>num, long seed)                                   | Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.                         |
| void<br>saveAsTextFile(String<br>path, Class<? extends<br>org.apache.hadoop.io.<br>compress.CompressionCodec> codec) | Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file. |
| java.util.Map<K,Object<br>> countByKey()                                                                             | Counts the appearance times of each key.                                                                                                                  |
| void<br>foreach(VoidFunction<br><T> f)                                                                               | Runs a function <i>func</i> on each element of the RDD.                                                                                                   |

| Method                                  | Description                                           |
|-----------------------------------------|-------------------------------------------------------|
| java.util.Map<T,Long><br>countByValue() | Counts the times that each element of the RDD occurs. |

**Table 1-183** New APIs of Spark core

| API                                                                          | Description                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public<br>java.util.concurrent.atomic.AtomicBool<br>ean isSparkContextDown() | Determines whether sparkContext is shut down completely. The initial value is <b>false</b> .<br><br>The value <b>true</b> indicates that sparkContext is shut down completely.<br>The value <b>false</b> indicates that sparkContext is not shut down.<br><br>For example,<br><b>jsc.sc().isSparkContextDown().get() == true</b> indicates that sparkContext is shut down completely. |

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- JavaStreamingContext: main entrance of the Spark Streaming, which is used to provide methods for creating DStream. Intervals by batch need to be set in the input parameter.
- JavaDStream: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- JavaPairDStream: interface of KV DStream, which is used to provide the reduceByKey and join operations.
- JavaReceiverInputDStream<T>: specifies any inflow accepting data from the network.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 1-184** Spark Streaming methods

| Method                                                                                              | Description                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JavaReceiverInputD-<br>Stream<java.lang.String><br>socketStream(java.lang.String hostname,int port) | Creates an inflow to receive data from the corresponding hostname and port through the TCP socket. Received data is resolved to the UTF8 format. The default storage level is the Memory+Disk. |

| Method                                                                                                                                            | Description                                                                                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JavaDStream<java.lang.String><br>textFileStream(java.lang.String directory)                                                                       | Creates an inflow to detect new files compatible with the Hadoop file system, and read it as a text file. The directory of the input parameter is an HDFS directory. |
| void start()                                                                                                                                      | Starts the Spark Streaming computing.                                                                                                                                |
| void awaitTermination()                                                                                                                           | Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .                                                                                    |
| void stop()                                                                                                                                       | Stops the Spark Streaming computing.                                                                                                                                 |
| <T> JavaDStream<T><br>transform(java.util.List<JavaDStream<?>><br>dstreams,Function2<java.util.List<JavaRDD<?>,Time,JavaRDD<T>><br>transformFunc) | Performs the Function operation on each RDD to obtain a new DStream. In this function, the sequence of the JavaRDDs must be the same as the corresponding DStreams.  |
| <T> JavaDStream<T><br>union(JavaDStream<T><br>first,java.util.List<JavaDStream<T>> rest)                                                          | Creates a unified DStream from multiple DStreams with the same type and sliding window.                                                                              |

**Table 1-185** Spark Streaming enhancement interface

| Method                                        | Description                                     |
|-----------------------------------------------|-------------------------------------------------|
| JAVADStreamKafkaWriter.writeToKafka()         | Writes data from DStream into Kafka in batch.   |
| JAVADStreamKafkaWriter.writeToKafkaBySingle() | Writes data from DStream into Kafka one by one. |

## Spark SQL Common Interfaces

Spark SQL mainly uses the following classes:

- **SQLContext**: main entrance of the Spark SQL function and DataFrame.
- **DataFrame**: a distributed dataset organized by naming columns.
- **DataFrameReader**: interface for loading the DataFrame from external storage systems.
- **DataFrameStatFunctions**: implementation the statistic function of the DataFrame.
- **UserDefinedFunction**: function defined by users.

Common Actions methods are described in the following table.

**Table 1-186** Spark SQL methods

| Method                                       | Description                                                                                                                   |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| Row[] collect()                              | Returns an array containing all DataFrame columns.                                                                            |
| long count()                                 | Returns the number of DataFrame rows.                                                                                         |
| DataFrame describe(java.lang.String... cols) | Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value. |
| Row first()                                  | Returns the first row.                                                                                                        |
| Row[] head(int n)                            | Returns the first <i>n</i> rows.                                                                                              |
| void show()                                  | Displays the first 20 rows in table.                                                                                          |
| Row[] take(int n)                            | Returns the first <i>n</i> rows in the DataFrame.                                                                             |

**Table 1-187** Basic DataFrame Functions

| Method                                                                | Description                                                                            |
|-----------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| void explain(boolean extended)                                        | Prints the logical plan and physical plan of the SQL.                                  |
| void printSchema()                                                    | Prints the schema information to the console.                                          |
| registerTempTable                                                     | Registers the DataFrame as a temporary table, whose period is bound to the SQLContext. |
| DataFrame toDF(java.lang.String.. colNames)                           | Returns a DataFrame whose columns are renamed.                                         |
| DataFrame sort(java.lang.String sortCol,java.lang.String... sortCols) | Sorts columns in ascending or descending orders based on different columns.            |
| GroupedData rollup(Column... cols)                                    | Performs multiple-dimension crankback on the specified columns in the DataFrame.       |

### 1.21.5.1.2 Scala

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see *MapReduce Service (MRS) 3.3.1-LTS Manager Rest API Reference (for Huawei Cloud Stack 8.3.1)*.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- `SparkContext`: external interface of Spark, which is used to provide the functions of Spark for Scala applications that invoke this class, for example, connecting Spark clusters and generating RDDs.
- `SparkConf`: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- `Resilient Distributed Dataset (RDD)`: defines the RDD class in the Spark application. The class provides the data collection operation methods, such as `map` and `filter`.
- `PairRDDFunctions`: provides computation operations for the RDD data of the key-value pair, such as `groupByKey`.
- `Broadcast`: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- `StorageLevel`: data storage levels, including memory (MEMORY\_ONLY), disk (DISK\_ONLY), and memory+disk (MEMORY\_AND\_DISK).

RDD supports two types of operations, transformation and action. [Table 1-188](#) and [Table 1-189](#) describe the common methods.

**Table 1-188** Transformation

| Method                                                                                                        | Description                                                                                                     |
|---------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <code>map[U](f: (T) =&gt; U): RDD[U]</code>                                                                   | Returns a new RDD by applying a function to all elements of this RDD.                                           |
| <code>filter(f: (T) =&gt; Boolean): RDD[T]</code>                                                             | Invokes the f method for all RDD elements to generate a satisfied data set that is returned in the form of RDD. |
| <code>flatMap[U](f: (T) =&gt; TraversableOnce[U]) (implicit arg0: ClassTag[U]): RDD[U]</code>                 | Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.    |
| <code>sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong()): RDD[T]</code> | Returns a sampled subset of this RDD.                                                                           |
| <code>union(other: RDD[T]): RDD[T]</code>                                                                     | Returns a new RDD, contains source RDD and the group of RDD's elements.                                         |
| <code>distinct([numPartitions: Int]): RDD[T]</code>                                                           | Returns a new RDD containing the distinct elements in this RDD.                                                 |
| <code>groupByKey(): RDD[(K, Iterable[V])]</code>                                                              | Returns (K,Iterable[V]) and combines the values of the same key to a set.                                       |
| <code>reduceByKey(func: (V, V) =&gt; V[, numPartitions: Int]): RDD[(K, V)]</code>                             | Invokes func on the values of the same key.                                                                     |

| Method                                                                                         | Description                                                                                                                                                                           |
|------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)] | Sorts by key in ascending or descending order. Ascending is of the boolean type.                                                                                                      |
| join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]                            | Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.                                              |
| cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]       | Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks. |
| cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]                           | Returns the Cartesian product of the RDD and other RDDs.                                                                                                                              |

**Table 1-189** Action

| API                                                                                            | Description                                                                                                                                               |
|------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| reduce(f: (T, T) => T):                                                                        | Invokes f on elements of the RDD.                                                                                                                         |
| collect(): Array[T]                                                                            | Returns an array that contains all of the elements in this RDD.                                                                                           |
| count(): Long                                                                                  | Returns the number of elements in the dataset.                                                                                                            |
| first(): T                                                                                     | Returns the first element in the dataset.                                                                                                                 |
| take(num: Int): Array[T]                                                                       | Returns the first <i>n</i> elements in this RDD.                                                                                                          |
| takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong()): Array[T] | Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.                         |
| saveAsTextFile(path: String): Unit                                                             | Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file. |

| API                                                                                        | Description                                                                                                                                 |
|--------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| saveAsSequenceFile(path: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit | This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system. |
| countByKey(): Map[K, Long]                                                                 | Counts the appearance times of each key.                                                                                                    |
| foreach(func: (T) => Unit): Unit                                                           | Applies a function f to all elements of this RDD.                                                                                           |
| countByValue()(implicit ord: Ordering[T] = null): Map[T, Long]                             | Counts the times that each element of the RDD occurs.                                                                                       |

**Table 1-190** New APIs of Spark core

| API                              | Description                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| isSparkContextDown:AtomicBoolean | Determines whether sparkContext is shut down completely. The initial value is <b>false</b> .<br>The value <b>true</b> indicates that sparkContext is shut down completely.<br>The value <b>false</b> indicates that sparkContext is not shut down.<br>For example,<br><b>sc.isSparkContextDown.get() == true</b><br>indicates that sparkContext is shut down completely. |

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- `StreamingContext`: main entrance of the Spark Streaming, which is used to provide methods for creating DStream. Intervals by batch need to be set in the input parameter.
- `dstream.DStream`: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- `dstream.PairDStreamFunctions`: DStream of key-value, common operations are `groupByKey` and `reduceByKey`.

The cooperated Java API of Spark Streaming are `JavaStreamingContext`, `JavaDStream`, `JavaPairDStream`.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 1-191** Spark Streaming methods

| Method                                                                                                                                       | Description                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| socketTextStream(hostname: String, port: Int, storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2): ReceiverInputDStream[String] | Creates an input stream from the TCP source host:port.                                                                                                                                             |
| start():Unit                                                                                                                                 | Starts the Spark Streaming computing.                                                                                                                                                              |
| awaitTermination(timeout: long):Unit                                                                                                         | Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .                                                                                                                  |
| stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit                                                                               | Stops the Spark Streaming computing.                                                                                                                                                               |
| transform[T](dstreams: Seq[DStream[_]], transformFunc: (Seq[RDD[_]], Time) ? RDD[T])(implicit arg0: ClassTag[T]): DStream[T]                 | Performs the function operation on each RDD to obtain a new DStream.                                                                                                                               |
| UpdateStateByKey(func)                                                                                                                       | Updates the status of DStream. To use this method, you need to define the state and state update functions.                                                                                        |
| window(windowLength, slideInterval)                                                                                                          | Generates a new DStream by batch calculating according to the window of the source DStream.                                                                                                        |
| countByWindow(windowLength, slideInterval)                                                                                                   | Returns the number of sliding window elements in the stream.                                                                                                                                       |
| reduceByWindow(func, windowLength, slideInterval)                                                                                            | When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window. |
| join(otherStream, [numTasks])                                                                                                                | Performs a join operation between different Spark Streamings.                                                                                                                                      |
| DStreamKafkaWriter.writeToKafka()                                                                                                            | Writes data from DStream into Kafka in batch.                                                                                                                                                      |

| Method                                    | Description                                     |
|-------------------------------------------|-------------------------------------------------|
| DStreamKafkaWriter.writeToKafkaBySingle() | Writes data from DStream into Kafka one by one. |

**Table 1-192** Spark Streaming enhancement interface

| Method                                    | Description                                     |
|-------------------------------------------|-------------------------------------------------|
| DStreamKafkaWriter.writeToKafka()         | Writes data from DStream into Kafka in batch.   |
| DStreamKafkaWriter.writeToKafkaBySingle() | Writes data from DStream into Kafka one by one. |

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of the Spark SQL function and DataFrame.
- DataFrame: a distributed dataset organized by naming columns.
- HiveContext: An instance of the Spark SQL execution engine that integrates with data stored in Hive.

**Table 1-193** Common Actions methods

| Method                                      | Description                                                                                                                   |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| collect(): Array[Row]                       | Returns an array containing all DataFrame columns.                                                                            |
| count(): Long                               | Returns the number of DataFrame rows.                                                                                         |
| describe(cols: String*): DataFrame          | Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value. |
| first(): Row                                | Returns the first row.                                                                                                        |
| Head(n:Int): Row                            | Returns the first <i>n</i> rows.                                                                                              |
| show numRows: Int, truncate: Boolean): Unit | Displays DataFrame in a table.                                                                                                |
| take(n:Int): Array[Row]                     | Returns the first <i>n</i> rows in the DataFrame.                                                                             |

**Table 1-194** Basic DataFrame Functions

| Method                                     | Description                                                                            |
|--------------------------------------------|----------------------------------------------------------------------------------------|
| explain(): Unit                            | Prints the logical plan and physical plan of the SQL.                                  |
| printSchema(): Unit                        | Prints the schema information to the console.                                          |
| registerTempTable(tableName: String): Unit | Registers the DataFrame as a temporary table, whose period is bound to the SQLContext. |
| toDF(colNames: String*): DataFrame         | Returns a DataFrame whose columns are renamed.                                         |

### 1.21.5.1.3 Python

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see *MapReduce Service (MRS) 3.3.1-LTS Manager Rest API Reference (for Huawei Cloud Stack 8.3.1)*

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- pyspark.SparkContext: external API of Spark. It provides the functions of Spark for Python applications that invoke this class, for example, connecting Spark clusters, creating RDDs, and broadcasting variables.
- pyspark.SparkConf: Spark application configuration class. It is used to set an application name, execution mode, and executor memory.
- pyspark.RDD: class used to define the RDD in the Spark application. The class provides the data collection operation methods, such as map and filter.
- pyspark.Broadcast: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- pyspark.StorageLevel: data storage levels, including memory (MEMORY\_ONLY), disk (DISK\_ONLY), and memory+disk (MEMORY\_AND\_DISK).
- pyspark.sql.SQLContext: Main entry point for SparkSQL functionality. It can be used to create DataFrame, register DataFrame as a table, and execute SQL on a table.
- pyspark.sql.DataFrame: A distributed collection of data grouped into named columns. A DataFrame is equivalent to a relational table in Spark SQL, and can be created using various functions in SQLContext.
- pyspark.sql.DataFrameNaFunctions: Functionality for working with missing data in DataFrame.
- pyspark.sql.DataFrameStatFunctions: Functionality for statistic functions with DataFrame.

The RDD supports two types of operations, transformation and action. [Table 1-195](#) and [Table 1-196](#) show the common methods.

**Table 1-195** Transformation

| Method                                                                             | Description                                                                                                                                                                           |
|------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| map(f,<br>preservesPartition-<br>ing=False)                                        | Returns a new RDD by applying a function to all elements of this RDD.                                                                                                                 |
| filter(f)                                                                          | Invokes the Func method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.                                                                    |
| flatMap(f,<br>preservesPartition-<br>ing=False)                                    | Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.                                                                          |
| sample(withReplace-<br>ment, fraction,<br>seed=None)                               | Returns a sampled subset of this RDD.                                                                                                                                                 |
| union(rdds)                                                                        | Returns a new RDD, contains source RDD and the group of RDD's elements.                                                                                                               |
| distinct([numPartitions: Int]): RDD[T]                                             | Returns a new RDD containing the distinct elements in this RDD.                                                                                                                       |
| groupByKey():<br>RDD[(K, Iterable[V])]                                             | Returns (K,Iterable[V]) and combines the values of the same key to a set.                                                                                                             |
| reduceByKey(func,<br>numPartitions=None)                                           | Invokes Func on the values of the same key.                                                                                                                                           |
| sortByKey(ascending= True,<br>numPartitions=None,<br>keyfunc=function<br><lambda>) | Sorts by key in ascending or descending order.<br>Ascending is of the boolean type.                                                                                                   |
| join(other,<br>numPartitions)                                                      | Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.                                              |
| cogroup(other,<br>numPartitions)                                                   | Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks. |
| cartesian(other)                                                                   | Returns the Cartesian product of the RDD and other RDDs.                                                                                                                              |

**Table 1-196** Action

| API       | Description                          |
|-----------|--------------------------------------|
| reduce(f) | Invokes Func on elements of the RDD. |

| API                                                  | Description                                                                                                                                               |
|------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| collect()                                            | Returns an array that contains all of the elements in this RDD.                                                                                           |
| count()                                              | Returns the number of elements in the dataset.                                                                                                            |
| first()                                              | Returns the first element in the dataset.                                                                                                                 |
| take(num)                                            | Returns the first num elements of the RDD.                                                                                                                |
| takeSample(withReplacement, num, seed)               | Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.                         |
| saveAsTextFile(path, compressionCodecClass)          | Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file. |
| saveAsSequenceFile(path, compressionCodecClass=None) | This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system.               |
| countByKey()                                         | Counts the appearance times of each key.                                                                                                                  |
| foreach(func)                                        | Applies a function f to all elements of this RDD.                                                                                                         |
| countByValue()                                       | Counts the times that each value of the RDD occurs.                                                                                                       |

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- `pyspark.streaming.StreamingContext`: main entrance of Spark Streaming. It provides methods for creating the DStream. A batch interval needs to be set in the input parameter.
- `pyspark.streaming.DStream`: A Discretized Stream (DStream), the basic abstraction in Spark Streaming, is a continuous sequence of RDDs (of the same type) representing a continuous stream of data.
- `dstream.PairDStreamFunctions`: DStream of key-value, common operations are `groupByKey` and `reduceByKey`.

The cooperated Java API of Spark Streaming are `JavaStreamingContext`, `JavaDStream`, `JavaPairDStream`.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 1-197** Spark Streaming common interfaces

| Method                                            | Description                                                                                                                                                                                                                      |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| socketTextStream(hostName, port, storageLevel)    | Creates an input stream from the TCP source host:port.                                                                                                                                                                           |
| start()                                           | Starts the Spark Streaming computing.                                                                                                                                                                                            |
| awaitTermination(timeout)                         | Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .                                                                                                                                                |
| stop(stopSparkContext, stopGracefully)            | Stops Spark Streaming computing. <b>stopSparkContext</b> is used to determine whether SparkContext needs to be terminated. <b>StopGracefully</b> is used to determine whether to wait for all the received data to be processed. |
| updateStateByKey(func)                            | Updates the status of DStream. To use this method, you need to define the state and state update functions.                                                                                                                      |
| window(windowLength, slideInterval)               | Generates a new DStream by batch calculating according to the window of the source DStream.                                                                                                                                      |
| countByWindow(windowLength, slideInterval)        | Returns the number of sliding window elements in the stream.                                                                                                                                                                     |
| reduceByWindow(func, windowLength, slideInterval) | When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.                               |
| join(other, numPartitions)                        | Performs a join operation between different Spark Streamings.                                                                                                                                                                    |

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- `pyspark.sql.SQLContext`: main entrance of the Spark SQL function and `DataFrame`.
- `pyspark.sql.DataFrame`: a distributed dataset organized by naming columns.
- `pyspark.sql.HiveContext`: main entrance for obtaining data stored in Hive.
- `pyspark.sql.DataFrameStatFunctions`: Functionality for statistic functions with `DataFrame`.
- `pyspark.sql.functions`: A collection of builtin functions.
- `pyspark.sql.Window`: window function provided by SQL

**Table 1-198** Spark SQL common Actions

| Method     | Description                                                                                                                   |
|------------|-------------------------------------------------------------------------------------------------------------------------------|
| collect()  | Returns an array containing all DataFrame columns.                                                                            |
| count()    | Returns the number of DataFrame rows.                                                                                         |
| describe() | Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value. |
| first()    | Returns the first row.                                                                                                        |
| head(n)    | Returns the first <i>n</i> rows.                                                                                              |
| show()     | Displays DataFrame in a table.                                                                                                |
| take(num)  | Returns the first num rows in the DataFrame.                                                                                  |

**Table 1-199** Basic DataFrame Functions

| Method                  | Description                                                                            |
|-------------------------|----------------------------------------------------------------------------------------|
| explain()               | Prints the logical plan and physical plan of the SQL.                                  |
| printSchema()           | Prints the schema information to the console.                                          |
| registerTempTable(name) | Registers the DataFrame as a temporary table, whose period is bound to the SQLContext. |
| toDF()                  | Returns a DataFrame whose columns are renamed.                                         |

#### 1.21.5.1.4 REST API

##### Function Description

The Spark REST API presents some web UI metrics in the JSON format, providing users with a simpler method to create new visualization and monitoring tools. The REST API can be used to query information about running and historical applications. The open-source Spark REST API allows users to query information about Jobs, Stages, Storage, Environment, and Executors. In the FusionInsight version, the REST API used to query SQL, JDBC server, and Streaming information is added. For more information about the open-source REST API, see <https://spark.apache.org/docs/3.3.1/monitoring.html#rest-api>.

##### Preparing Running Environment

Install the FusionInsight client. Install a FusionInsight client on the node. For example, install the client in the **/opt/hadoopclient** directory.

## REST API

You can use the following commands to dodge the REST API filter and directly obtain the application information:

### NOTICE

- In security mode, the JobHistory supports only the HTTPS protocol. Therefore, use the HTTPS protocol in the URL of the following command.
- In security mode, you need to set **spark.ui.customErrorPage** to **false** and restart Spark. Change the value of this parameter for the JobHistory, JDBCServer, and SparkResource instances.

### NOTE

Perform the following steps to upgrade the curl version on the node:

1. Download the curl installation package from the following website: <http://curl.haxx.se/download/>
2. Run the following command to decompress the installation package:  
**tar -xzvf curl-x.x.x.tar.gz**
3. Run the following commands to overwrite the old curl version with the new one:  
**cd curl-x.x.x**  
**./configure**  
**make**  
**make install**
4. Run the following command to update the dynamic link library of curl:  
**ldconfig**
5. After the installation is successful, log in to the node again and run the following command to check whether the curl version is successfully updated:  
**curl --version**

- Obtaining information about all applications on the JobHistory node:

- Command:

```
curl -k -i --negotiate -u: "https://192.168.227.16:22500/api/v1/applications"
```

**192.168.227.16** indicates the service IP address of the JobHistory node and **22500** indicates the port number of the JobHistory node.

- Command output:

```
[ {  
    "id" : "application_1517290848707_0008",  
    "name" : "Spark Pi",  
    "attempts" : [ {  
        "startTime" : "2018-01-30T15:05:37.433CST",  
        "endTime" : "2018-01-30T15:06:04.625CST",  
        "lastUpdated" : "2018-01-30T15:06:04.848CST",  
        "duration" : 27192,  
        "sparkUser" : "sparkuser",  
        "completed" : true,  
        "startTimeEpoch" : 1517295937433,  
        "endTimeEpoch" : 1517295964625,  
        "lastUpdatedEpoch" : 1517295964848  
    } ]  
, {  
    "  
id" : "application_1517290848707_0145",  
}
```

```
"name" : "Spark shell",
"attempts" : [ {
  "startTime" : "2018-01-31T15:20:31.286CST",
  "endTime" : "1970-01-01T07:59:59.999CST",
  "lastUpdated" : "2018-01-31T15:20:47.086CST",
  "duration" : 0,
  "sparkUser" : "admintest",
  "completed" : false,
  "startTimeEpoch" : 1517383231286,
  "endTimeEpoch" : -1,
  "lastUpdatedEpoch" : 1517383247086
} ]
}]
```

- Analysis:

With this command, you can query information about all Spark applications in the current cluster, including running applications and the completed applications. **Table 1-200** provides information about each application.

**Table 1-200** Parameter description

| Parameter | Description                                                                                                                                                                          |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id        | Application ID.                                                                                                                                                                      |
| name      | Application name.                                                                                                                                                                    |
| attempts  | Attempts executed by the application, including the attempt start time, attempt end time, user who initiates the attempts, and status indicating whether the attempts are completed. |

- Obtaining information about a specific application on the JobHistory node:

- Command:

```
curl -k -i --negotiate -u: "https://192.168.227.16:22500/api/v1/applications/
application_1517290848707_0008"
```

**192.168.227.16** indicates the service IP address of the JobHistory node, **22500** indicates the port number of the JobHistory node, and **application\_1517290848707\_0008** indicates the application ID.

- Command output:

```
{
  "id" : "application_1517290848707_0008",
  "name" : "Spark Pi",
  "attempts" : [ {
    "startTime" : "2018-01-30T15:05:37.433CST",
    "endTime" : "2018-01-30T15:06:04.625CST",
    "lastUpdated" : "2018-01-30T15:06:04.848CST",
    "duration" : 27192,
    "sparkUser" : "sparkuser",
    "completed" : true,
    "startTimeEpoch" : 1517295937433,
    "endTimeEpoch" : 1517295964625,
    "lastUpdatedEpoch" : 1517295964848
  } ]
}
```

- Analysis:

With this command, you can query information about a Spark application. [Table 1-200](#) provides information about the application.

- Obtaining information about the executor of a running application:

- Command of alive executors list:

```
curl -k -i --negotiate -u: "https://192.168.169.84:26001/proxy/  
application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/  
executors"
```

- Command of all executors (alive and dead) list:

```
curl -k -i --negotiate -u: "https://192.168.169.84:26001/proxy/  
application_1478570725074_0046/api/v1/applications/application_1478570725074_0046/  
allexecutors"
```

**192.168.169.84** indicates the service IP address of the master node of the ResourceManager, **26001** indicates the port number of the ResourceManager, and **application\_1478570725074\_0046** indicates the application ID in YARN.

- Command output:

```
[  
  {  
    "id" : "driver",  
    "hostPort" : "192.168.169.84:23886",  
    "isActive" : true,  
    "rddBlocks" : 0,  
    "memoryUsed" : 0,  
    "diskUsed" : 0,  
    "activeTasks" : 0,  
    "failedTasks" : 0,  
    "completedTasks" : 0,  
    "totalTasks" : 0,  
    "totalDuration" : 0,  
    "totalInputBytes" : 0,  
    "totalShuffleRead" : 0,  
    "totalShuffleWrite" : 0,  
    "maxMemory" : 278019440,  
    "executorLogs" : {}  
  }, {  
    "id" : "1",  
    "hostPort" : "192.168.169.84:23902",  
    "isActive" : true,  
    "rddBlocks" : 0,  
    "memoryUsed" : 0,  
    "diskUsed" : 0,  
    "totalCores" : 1,  
    "maxTasks" : 1,  
    "activeTasks" : 0,  
    "failedTasks" : 0,  
    "completedTasks" : 0,  
    "totalTasks" : 0,  
    "totalDuration" : 0,  
    "totalGCTime" : 139,  
    "totalInputBytes" : 0,  
    "totalShuffleRead" : 0,  
    "totalShuffleWrite" : 0,  
    "maxMemory" : 555755765,  
    "executorLogs" : {  
      "stdout" : "https://XTJ-224:26010/node/containerlogs/  
container_1478570725074_0049_01_000002/admin/stdout?start=-4096",  
      "stderr" : "https://XTJ-224:26010/node/containerlogs/  
container_1478570725074_0049_01_000002/admin/stderr?start=-4096"  
    }  
  }]  
]
```

- Analysis:

With this command, you can query information about all executors including drivers of the current application. [Table 1-201](#) provides basic information about each executor.

**Table 1-201** Parameter description

| Parameter    | Description                                                                                                   |
|--------------|---------------------------------------------------------------------------------------------------------------|
| id           | Executor ID.                                                                                                  |
| hostPort     | IP address and port number of the node where the Executor resides.<br>Format: <i>IP address:port number</i> . |
| executorLogs | Path to Executor logs.                                                                                        |

## Enhanced REST API

- SQL related: obtaining all the SQL statements and those with the longest execution time.

- SparkUI command:

```
curl -k -i --negotiate -u: "https://192.168.195.232:26001/proxy/  
application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/SQL"
```

**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **26001** indicates the port number of the ResourceManager, and **application\_1476947670799\_0053** indicates the application ID in YARN.

### NOTE

You can add parameters to the URL after the command to search for the corresponding SQL statements.

For example, run the following command to view 100 SQL statements:

```
curl -k -i --negotiate -u: "https://192.168.195.232:26001/proxy/  
application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/  
SQL?limit=100"
```

Run the following command to view running parameters:

```
curl -k -i --negotiate -u: "https://192.168.195.232:26001/proxy/  
application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/  
SQL?completed=false"
```

- JobHistory command:

```
curl -k -i --negotiate -u: "https://192.168.227.16:22500/api/v1/applications/  
application_1478570725074_0004/SQL"
```

**192.168.227.16** indicates the service IP address of the JobHistory node, **22500** indicates the port number of the JobHistory node, and **application\_1478570725074\_0004** indicates the application ID.

- Command output:

The command output of the SparkUI and JobHistory commands is as follows:

```
{  
  "longestDurationOfCompletedSQL" : [ {  
    "id" : 0,  
    "status" : "COMPLETED",  
    "description" : "getCallSite at SQLExecution.scala:48",  
    "submissionTime" : "2016/11/08 15:39:00",  
    "duration" : "2 s",  
    "runningJobs" : [],  
    "successtedJobs" : [ 0 ],  
    "failedJobs" : []  
  } ],
```

```
"sqls" : [ {
    "id" : 0,
    "status" : "COMPLETED",
    "description" : "getCallSite at SQLExecution.scala:48",
    "submissionTime" : "2016/11/08 15:39:00",
    "duration" : "2 s",
    "runningJobs" : [ ],
    "successedJobs" : [ 0 ],
    "failedJobs" : [ ]
  }
}
```

- Analysis:

After running this command, you can obtain all the SQL statements executed by the current application (the **sqls** part of the command output) and the SQL statements with the longest execution time (the **longestDurationOfCompletedSQL** part of the command output). The information about each SQL statement is listed in [Table 1-202](#).

**Table 1-202** Parameter description

| Parameter     | Description                                                                          |
|---------------|--------------------------------------------------------------------------------------|
| id            | SQL statement ID.                                                                    |
| status        | Execution status of the SQL statement, which can be: running, completed, and failed. |
| runningJobs   | Jobs that are being executed generated by the SQL statement.                         |
| successedJobs | Jobs that are successfully executed generated by the SQL statement.                  |
| failedJobs    | Job that fails to be executed generated by the SQL statement.                        |

- JDBC server related: obtaining the number of sessions, number of being-executed SQL statements, information about all sessions, and information about SQL statements.

- Command:

```
curl -k -i --negotiate -u: "https://192.168.195.232:26001/proxy/
application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/
sqlserver"
```

**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **26001** indicates the port number of the ResourceManager, and **application\_1476947670799\_0053** indicates the application ID in YARN.

- Command output:

```
{
  "sessionNum" : 1,
  "runningSqlNum" : 0,
  "sessions" : [ {
    "user" : "spark",
    "ip" : "192.168.169.84",
    "sessionId" : "9dfec575-48b4-4187-876a-71711d3d7a97",
    "startTime" : "2016/10/29 15:21:10",
    "finishTime" : ""
  }
}
```

```
        "duration" : "1 minute 50 seconds",
        "totalExecute" : 1
    },
    "sqls" : [ {
        "user" : "spark",
        "jobId" : [ ],
        "groupId" : "e49ff81a-230f-4892-a209-a48abea2d969",
        "startTime" : "2016/10/29 15:21:13",
        "finishTime" : "2016/10/29 15:21:14",
        "duration" : "555 ms",
        "statement" : "show tables",
        "state" : "FINISHED",
        "detail" : "== Parsed Logical Plan ==\nShowTablesCommand None\n== Analyzed Logical\nPlan ==\ntableName: string, isTemporary: boolean\nShowTablesCommand None\n== Cached\nLogical Plan ==\nShowTablesCommand None\n== Optimized Logical Plan ==\nShowTablesCommand None\n== Physical Plan ==\nExecutedCommand\nShowTablesCommand None\n==\nCode Generation: true"
    }
}
```

- Analysis:

After running this command, you can query the number of sessions in the current JDBC application, number of being-executed SQL statements, and information about all sessions and SQL statements. The information about each session is listed in [Table 1-203](#), and the information about each SQL statement is listed in [Table 1-204](#):

**Table 1-203** Session parameter description

| Parameter    | Description                                       |
|--------------|---------------------------------------------------|
| user         | User to whom the session connects.                |
| ip           | IP address of the node where the session resides. |
| sessionId    | Session ID.                                       |
| startTime    | Time when the session starts the connection.      |
| finishTime   | Time when the session ends the connection.        |
| duration     | Connection duration of the session.               |
| totalExecute | Number of SQL statements executed by the session. |

**Table 1-204** SQL parameter description

| Parameter | Description                          |
|-----------|--------------------------------------|
| user      | User who executes the SQL statement. |

| Parameter  | Description                                      |
|------------|--------------------------------------------------|
| jobId      | IDs of jobs contained in the SQL statement.      |
| groupId    | ID of the group where the SQL statement resides. |
| startTime  | Start time.                                      |
| finishTime | End time.                                        |
| duration   | SQL statement execution duration.                |
| statement  | SQL statement.                                   |
| detail     | Logical/Physical plan.                           |

- JDBC API enhancement cancels the SQL statement that is being executed by using the execution ID obtained from the beeline.
  - Commands for the operation:  
`curl -k -i --negotiate -X PUT -u: "https://192.168.195.232:26001/proxy/application_1477722033672_0008/api/v1/applications/application_1477722033672_0008/cancel/execution?executionId=8"`
  - Command output:  
 Cancel the job whose execution ID is 8.
  - Remarks:  
 Run the SQL statement in spark-beeline. If the SQL statement generates a Spark task, the execution ID of the SQL statement will be printed in beeline. To cancel the execution of the SQL statement, run the preceding command.
- Streaming related: obtaining the average input frequency, average scheduling delay, average execution duration, and average value of the overall delay.
  - Command:  
`curl -k -i --negotiate -u: "https://192.168.195.232:26001/proxy/application_1477722033672_0008/api/v1/applications/application_1477722033672_0008/streaming/statistics"`

**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **26001** indicates the port number of the ResourceManager, and **application\_1477722033672\_0008** indicates the application ID in YARN.
  - Command:  
`{
 "startTime" : "2018-12-25T08:58:10.836GMT",
 "batchDuration" : 1000,
 "numReceivers" : 1,
 "numActiveReceivers" : 1,
 "numInactiveReceivers" : 0,
 "numTotalCompletedBatches" : 373,
 "numRetainedCompletedBatches" : 373,
 "numActiveBatches" : 0,
 "numProcessedRecords" : 1,
 "numReceivedRecords" : 1,
 "avgInputRate" : 0.002680965147453083,
 "avgSchedulingDelay" : 14,
 "avgProcessingTime" : 47,`

```
    "avgTotalDelay" : 62
}
```

- Analysis:

After running this command, you can query the average input frequency (unit: events/sec), average scheduling delay (unit: ms), average execution time (unit: ms), and average value of the total delay (unit: ms) of the current Streaming application.

### 1.21.5.2 Common CLIs

For details about how to use the Spark CLIs, visit the official website <http://spark.apache.org/docs/3.3.1/quick-start.html>.

#### Common CLI

Common Spark CLIs are described as follows:

- **spark-shell**

It provides an easy way to learn APIs, which is similar to the tool for interactive data analysis. It supports two languages including Scala and Python. In the Spark directory, run the `./bin/spark-shell` command to access the interactive interface of Scala, obtain data from HDFS, and then perform the RDD.

For example: a row of codes can count all words in a file.

```
scala> sc.textFile("hdfs://10.96.1.57:9000//wordcount_data.txt").flatMap(l => l.split(" ")).map(w => (w,1)).reduceByKey(_ + _).collect()
```

You can directly specify the Keytab and Principal in the command line to obtain authentication, and regularly update the keytab and authorized tokens to avoid the authentication expiry. The following command is used as an example.

```
spark-shell --principal spark2x/hadoop.<System domain name>@<System domain name> --keytab ${BIGDATA_HOME}/FusionInsight_Spark_8.3.1/install/FusionInsight-Spark-3.3.1/keytab/spark/SparkResource/spark2x.keytab --master yarn
```

- **spark-submit**

It is used to submit the Spark application to the Spark cluster for running and return the running results. The class, master, jar and input parameter need to be specified.

For example: Run the GroupByTest example in the jar. There are four input parameters and the specified running mode of the cluster is local single platform.

```
./bin/spark-submit --class org.apache.spark.examples.GroupByTest --master local[1] examples/jars/spark-examples_xxx.cbu.mrs.xxx.jar 6 10 10 3
```

You can directly specify the Keytab and Principal in the command line to obtain authentication, and regularly update the keytab and authorized tokens to avoid the authentication expiry. The following command is used as an example.

```
spark-submit --class org.apache.spark.examples.GroupByTest --master yarn --principal spark2x/hadoop.<System domain name>@<System domain
```

```
name> --keytab ${BIGDATA_HOME}/FusionInsight_Spark_8.3.1/install/
FusionInsight-Spark-3.3.1/keytab/spark/SparkResource/spark2x.keytab
examples/jars/spark-examples_xxx.cbu.mrs.xxx.jar 6 10 10 3
```

- **spark-sql**

It is used to perform the Hive metadata service and query command lines in the local mode. If its logical plan needs to be queried, add the explain extended before the SQL statement.

For example:

**Select key from src group by key**

You can directly specify the Keytab and Principal in the command line to obtain authentication, and regularly update the keytab and authorized tokens to avoid the authentication expiry. The following command is used as an example.

```
spark-sql --principal spark2x/hadoop.<System domain name>@<System
domain name> --keytab ${BIGDATA_HOME}/FusionInsight_Spark_8.3.1/
install/FusionInsight-Spark-3.3.1/keytab/spark/SparkResource/
spark2x.keytab --master yarn
```

- **run-example**

It is used to run or debug the default example in the Spark open-source community.

For example: Run the SparkPi.

```
./run-example SparkPi 100
```

### 1.21.5.3 JDBCServer Interface

#### Overview

The JDBCServer is another implement of HiveServer2 in the Hive. The Spark SQL is used to process the SQL statement at its bottom. Therefore, the JDBCServer has better performance than the Hive.

The JDBCServer is a JDBC interface. Users can log in to the JDBCServer and access the Spark SQL data through the JDBC. When the JDBCServer is started, a Spark SQL application is started, and the clients connected through the JDBC share the resources in this application. That is, various users can share data. When the JDBCServer is started, a listener is also started to wait for the connection of the JDBC client and submit the query after the connection. Therefore, during the configuration of the JDBCServer, at least the host name and port of the JDBCServer must be configured. If Hive data is required, the uris of the hive metastore needs to be provided.

JDBCServer starts a JDBC service on port 22550 of the installation node by default. (If you want to change the port, configure the **hive.server2.thrift.port** parameter.) You can connect to JDBCServer using Beeline or running the JDBC client code to run SQL statements.

For other information about the JDBCServer, visit the Spark official website: <http://spark.apache.org/docs/3.3.1/sql-programming-guide.html#distributed-sql-engine>.

## Beeline

For connection methods of the Beeline provided by the open-source community, visit <https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients>.

To solve the connection problem in two scenarios of the Beeline, the authentication information is added in the Beeline connection. The **user.keytab** and **user.principal** parameters are added in the URL of the JDBC. When the key tab expires, the login information of the client can be read automatically and the connection succeeds again.

Users do not want to perform the key tab authentication by running the **kinit** command because the key tab expires every 24 hours. The Keytab file and principal information can be obtained from the administrator. The following command is used as a connection example of Beeline.

```
sh CLIENT_HOME/spark/bin/beeline -u "jdbc:hive2://<zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spark thriftserver;user.principal=spark2x/hadoop.<System domain name>@<System domain name>;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<System domain name>@<System domain name>;"
```

### NOTE

- <zkNode1\_IP>:<zkNode1\_Port>,<zkNode2\_IP>:<zkNode2\_Port>,<zkNode3\_IP>:<zkNode3\_Port> indicates the URL of ZooKeeper. Multiple URLs are separated by comma. For example: 192.168.81.37:24002,192.168.195.232:24002,192.168.169.84:24002.
- **sparkthriftserver** indicates the directory in Zookeeper where a random JDBCServer instance is selected for the connection to the client.

## JDBC Client Codes

Log in to the JDBCServer by using the JDBC client codes and access the Spark SQL data. For details, see [Accessing the Spark SQL Through JDBC](#).

## Enhanced Features

Compared with the open-source community, Huawei provides two enhanced features: the JDBCServer HA solution and timeout of configuring the JDBCServer.

- The JDBCServer HA solution is described as follows:  
When multiple active nodes of JDBCServer provides services at the same time, a new client will be connected to another active node if a fault occurs on one node, ensuring continuous services for clusters. The operations of Beeline and JDBC client code connection are the same.
- Configure the timeout of the connection between the client and JDBCServer.
  - Beeline  
In network congestion, this feature can avoid the suspending of Beeline due to timeless wait of the return from the server. The method is described as follows:  
When the Beeline is started, add **--socketTimeOut=n**. The n indicates the timeout waiting for the service return. The unit is second and the default

- value is 0 (indicating never timing out). Set the maximum timeout waiting time as required.
- JDBC Client Codes  
In the scenario of network congestion, this feature can avoid the suspending of the client due to limitless wait of the return of server. The method to use is shown as follows:  
Before the obtaining of the JDBC by using the DriverManager.getConnection method, add the DriverManager.setLoginTimeout(n) method to configure the timeout length. n indicates the timeout length of waiting for the service return. The unit is second and the type is Int. The default value is 0 (indicating never timing out). Set the maximum timeout waiting time as required.

#### 1.21.5.4 Structured Streaming Functions and Reliability

##### Functions Supported by Structured Streaming

1. ETL operations on streaming data are supported.
2. Schema inference and partitioning of streaming DataFrames or Datasets are supported.
3. Operations on the streaming DataFrames or Datasets are supported, including SQL-like operations without types (such as select, where, and groupBy) and RDD operations with types (such as map, filter, and flatMap).
4. Aggregation calculation based on Event Time and processing of delay data are supported.
5. Deduplication of streaming data is supported.
6. Status computing is supported.
7. Stream processing tasks can be monitored.
8. Batch-stream join and stream join are supported.

The following table lists the supported join operations.

| Left Table | Right Table | Supported Join Type | Description                                                                                      |
|------------|-------------|---------------------|--------------------------------------------------------------------------------------------------|
| Static     | Static      | All types           | In stream processing, join operations with no streaming data involved are supported.             |
| Stream     | Static      | Inner               | Supported, but stateless.                                                                        |
|            |             | Left Outer          | Supported, but stateless.                                                                        |
|            |             | Right Outer         | Not supported.                                                                                   |
|            |             | Full Outer          | Not supported.                                                                                   |
| Stream     | Stream      | Inner               | Supported. The watermark or time range can be used to clear status of the left and right tables. |

| Left Table | Right Table | Supported Join Type | Description                                                                                                                                          |
|------------|-------------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
|            |             | Left Outer          | Conditionally supported. The watermark can be used to clear status of the left table, and watermark and time range must be used for the right table. |
|            |             | Right Outer         | Conditionally supported. The watermark can be used to clear status of the right table, and watermark and time range must be used for the left table. |
|            |             | Full Outer          | Not supported.                                                                                                                                       |

## Functions Not Supported by Structured Streaming

1. Multi-stream aggregation is not supported.
2. The following operations of obtaining multiple rows are not supported: limit, first, and take.
3. Distinct is not supported.
4. Sorting is supported only when output mode is complete.
5. The external connection between streams and static data sets is conditionally supported.
6. Immediate query and result return on some data sets are not supported.
  - count(): Instead of returning a single count from streaming data sets, it uses ds.groupBy().count() to return a streaming data set containing the running count.
  - foreach(): The ds.writeStream.foreach(...) is used to replace it.
  - show(): The output console sink is used to replace it.

## Structured Streaming Reliability

Based on the checkpoint and WAL mechanisms, Structured Streaming provides end-to-end exactly-once error tolerance semantics for the sources that can be replayed and the idempotent sinks that support repeated processing.

1. You can enable the checkpoint function by setting option ("checkpointLocation", "checkpoint path") in the program.

When data is restored from checkpoint, the application or configuration may change. Some changes may result in the failure in restoring data from the checkpoint. The restrictions are as follows:

- a. The number or type of sources cannot be changed.
- b. The source type and query statement determine whether the source parameter can change. For example:
  - The parameters related to rate control can be added, deleted, or modified. For example:

`spark.readStream.format("kafka").option("subscribe", "topic")` is changed to `spark.readStream.format("kafka").option("subscribe", "topic").option("maxOffsetsPerTrigger", ...)`.

- Unexpected problems may occur when the topic/file of the consumption is modified. For example:  
`spark.readStream.format("kafka").option("subscribe", "topic")` is changed to `spark.readStream.format("kafka").option("subscribe", "newTopic")`.

- c. The type of sink changes. Specific sinks can be combined. The specific scenarios need to be verified. For example:

- File sink can be changed to kafka sink. Kafka processes only new data.
- Kafka sink cannot be changed to file sink.
- Kafka sink can be changed to foreach sink, and vice versa.

- d. The sink type and query statement determine whether the sink parameter can change. For example:

- The output path of file sink cannot be changed.
- The output topic of Kafka sink can be changed.
- The custom operator code in foreach sink can be changed, but the change result depends on the user code.

- e. The projection, filter, and map-like operations can be changed in some scenarios. For example:

- Filters can be added and deleted. For example: `sdf.selectExpr("a")` is changed to `sdf.where(...).selectExpr("a").filter(...)`.
- When Output schema is the same, projections can be changed. For example: `sdf.selectExpr("stringColumn AS json").writeStream` is changed to `sdf.select(to_json(...).as("json")).writeStream`.
- If Output schema is different, projections can be changed in some conditions. For example: when `sdf.selectExpr("a").writeStream` is changed to `sdf.selectExpr("b").writeStream`, no error occurs only when the sink supports schema conversion from a to b.

- f. In some scenarios, the status restoration will fail after status is changed.

- Streaming aggregation: For example, in the `sdf.groupBy("a").agg(...)` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
- Streaming deduplication: For example, in the `sdf.dropDuplicates("a")` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
- Stream-stream join: For example, in the `sdf1.join(sdf2, ...)` operation, the schema of the association key cannot be changed, and the join type cannot be changed. Change of other join conditions may lead to uncertainty results.

- Any status computing: For example, in the `sdf.groupByKey(...).mapGroupsWithState(...)` or `sdf.groupByKey(...).flatMapGroupsWithState(...)` operation, the schema or timeout type of the user-defined status cannot be changed. Users can customize the state-mapping function change, but the change result depends on the user code. If schema changes are required, users can encode or decode the status data into binary data to support schema migration.

## 2. Fault tolerance list of Source

| Sources       | Supported Options                                                                                                                                                                                                                                                                                                                                                                                                      | Fault Tolerance Supported | Description                                                                                                                                                                                                                     |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| File source   | <b>path:</b> file path, which is mandatory.<br><b>maxFilesPerTrigger:</b> maximum number of files in each trigger. The default value is infinity.<br><b>latestFirst:</b> whether to process limited number of new files. The default value is <b>false</b> .<br><b>fileNameOnly:</b> whether the file name is used as the new file for verification instead of using the complete path. (Default value: <b>false</b> ) | Supported                 | Paths with wildcard are supported, but multiple paths separated by commas (,) are not supported. Files must be placed in a given directory in atomic mode, which can be implemented through file movement in most file systems. |
| Socket Source | <b>host:</b> IP address of the connected node, which is mandatory.<br><b>port:</b> connected port, which is mandatory.                                                                                                                                                                                                                                                                                                 | Not supported.            | -                                                                                                                                                                                                                               |
| Rate Source   | <b>rowsPerSecond:</b> number of rows generated per second. The default value is 1.<br><b>rampUpTime:</b> rising time before the speed specified by <b>rowsPerSecond</b> is reached.<br><b>numPartitions:</b> concurrency degree for generating data rows.                                                                                                                                                              | Supported                 | -                                                                                                                                                                                                                               |
| Kafka Source  | For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html</a> .                                                                                                                                                                                                                       | Supported                 | -                                                                                                                                                                                                                               |

### 3. Fault tolerance list of Sink

| Sinks             | Supported Output Mode    | Supported Options                                                                                                                                                                                      | Fault Tolerance            | Description                                                                                                                                                                                                                                                      |
|-------------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| File Sink         | Append                   | <b>Path:</b> The specified file format must be specified.<br>For details, see APIs in DataFrameWriter .                                                                                                | exactly-once               | Data can be written to partition tables. Time-based partition is better.                                                                                                                                                                                         |
| Kafka Sink        | Append, Update, Complete | For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html</a> .       | at-least-once              | For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html</a> .                                                                 |
| Foreach Sink      | Append, Update, Complete | None                                                                                                                                                                                                   | Depends on Foreach Writer. | For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-programming-guide.html#using-foreach">https://spark.apache.org/docs/3.3.1/structured-streaming-programming-guide.html#using-foreach</a> .                                     |
| ForeachBatch Sink | Append, Update, Complete | None                                                                                                                                                                                                   | Depends on operator.       | For details, see <a href="https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch">https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch</a> . |
| Console Sink      | Append, Update, Complete | <b>numRows:</b> number of rows printed in each round. The default value is <b>20</b> .<br><b>truncate:</b> whether to clear the output when the output is too long. The default value is <b>true</b> . | Not supported              | -                                                                                                                                                                                                                                                                |

| Sinks       | Supported Output Mode | Supported Options | Fault Tolerance                                                                            | Description |
|-------------|-----------------------|-------------------|--------------------------------------------------------------------------------------------|-------------|
| Memory Sink | Append, Complete      | None              | Not supported. In complete mode, the entire table is rebuilt after the query is restarted. | -           |

## 1.21.5.5 FAQ

### 1.21.5.5.1 How to Add a User-Defined Library

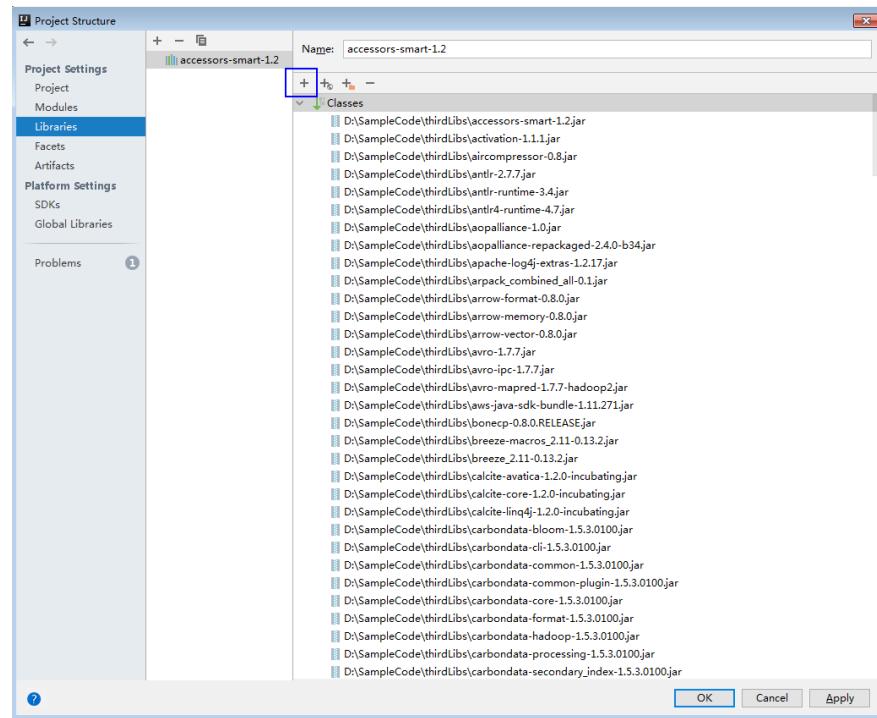
#### Question

In the development of the Spark application, users may add a user-defined dependent library which is different from the sample project. This section describes how to add a dependent library with user-defined codes into the project.

#### Answer

- Step 1** On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
- Step 2** Select the **Libraries** tab, click the icon "+" on the following page, and add the local dependent library.

Figure 1-301 Add the Dependent Library

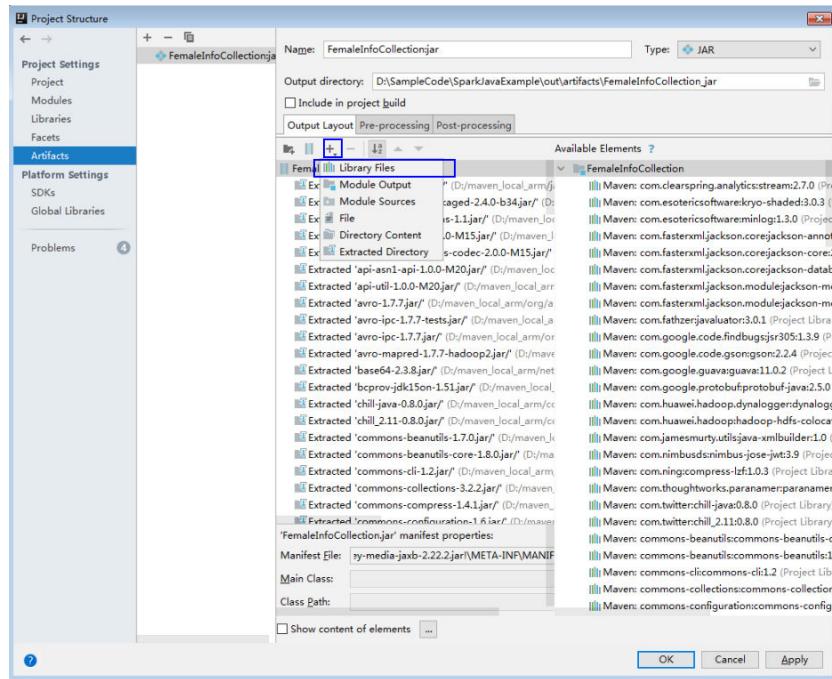


**Step 3** Click **Apply** to load the dependent library and click **OK** to complete the configuration.

**Step 4** Add the dependent library when building Artifacts. This is because the user-defined dependent library does not exist in the operating environment. By adding the library, the created jar contains the user-defined dependent library, which ensures that the Spark application runs properly.

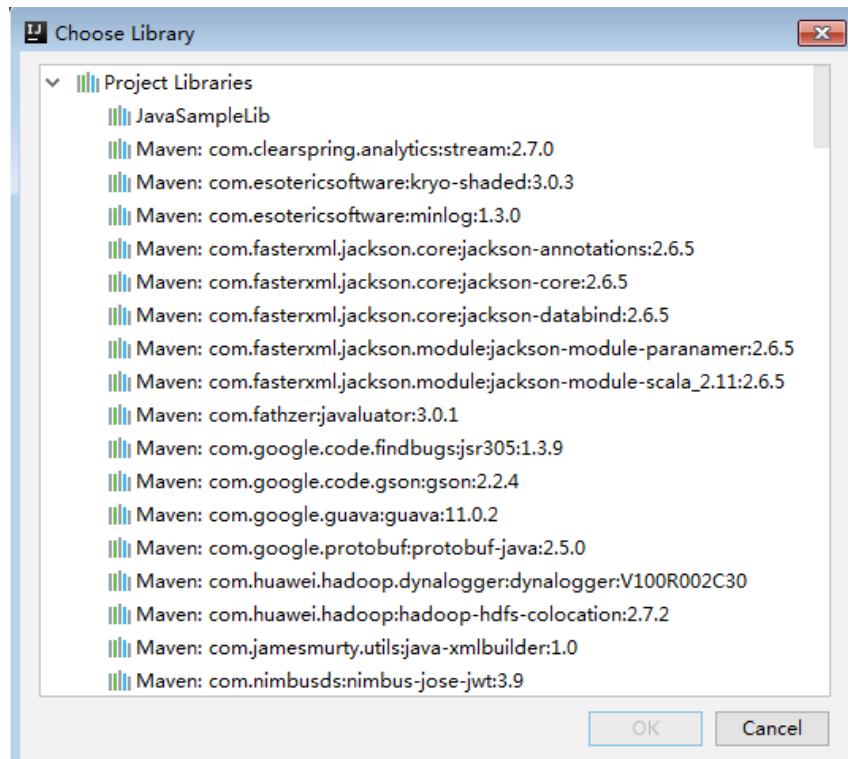
1. On the **Project Structure** page, select the **Artifacts** tab.
2. Click the icon "+" and select **Library Files** in the right window to add the dependent library.

**Figure 1-302 Add Library Files**



3. Choose the dependent library to be added and click **OK**.

**Figure 1-303 Choose Libraries**



4. Click **Apply** to load the dependent library and click **OK** to complete the configuration.

----End

### 1.21.5.5.2 How to Automatically Load Jars Packages?

#### Question

Before importing a project by using the IDEA tool, if Maven has been configured in the IDEA tool, the IDEA tool will automatically load the Jars packages in the Maven configuration. When the automatically loaded Jars packages do not match with the application, the project fails to be built. How to Automatically Load Jars Packages?

#### Answer

After a project is imported, to manually delete the automatically loaded Jars packages, perform the following steps:

1. On the IDEA tool, choose **File > Project Structures....**
2. Select **Libraries** and select the Jars packages that are automatically imported. Right-click the mouse and select **Delete**.

### 1.21.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStresmingKafka Project Is Running?

#### Question

While the KafkaWordCount task (`org.apache.spark.examples.streaming.KafkaWordCount`) is being submitted by running the `spark-submit` script, the log file shows that the Kafka-related class does not exist. The KafkaWordCount sample is provided by the Spark open-source community.

#### Answer

When Spark is deployed, the following JAR files are saved in the `$SPARK_HOME/jars/streamingClient010` directory on the client and the `$BIGDATA_HOME/FusionInsight_Spark_8.3.1/install/FusionInsight-Spark-3.3.1/spark/jars/streamingClient010` directory on the server.

- `kafka-clients-xxx.jar`
- `kafka_2.12-xxx.jar`
- `spark-streaming-kafka-xxx.cbu.mrs.xxx.jar`
- `spark-token-provider-kafka-xxx.cbu.mrs.xxx.jar`

Because `$SPARK_HOME/jars/streamingClient010/*` is not added in to classpath by default, you need to configure manually.

When the application is submitted and run, add following parameters in the command. See [Compiling and Running the Application](#).

`--jars $SPARK_CLIENT_HOME/jars/streamingClient010/kafka-client-2.4.0.jar,$SPARK_CLIENT_HOME/jars/streamingClient010/kafka_2.12-.jar,$SPARK_CLIENT_HOME/jars/streamingClient010/spark-streaming-kafka-xxx.cbu.mrs.xxx.jar`

You can run the preceding command to submit the self-developed applications and sample projects.

To submit the sample projects such as KafkaWordCount provided by Spark open source community, you need to add other parameters in addition to **--jars**. Otherwise, the `ClassNotFoundException` error will occur. The configurations in `yarn-client` and `yarn-cluster` modes are as follows:

- `yarn-client` mode

In the configuration file **spark-defaults.conf** on the client, add the path of the client dependency package, for example `$SPARK_HOME/jars/streamingClient010/*`, (in addition to **--jars**) to the **spark.driver.extraClassPath** parameter.

- `yarn-cluster` mode

Perform any one of the following configurations in addition to **--jars**:

- In the configuration file **spark-defaults.conf** on the client, add the path of the server dependency package, for example,  `${BIGDATA_HOME}/FusionInsight_Spark_8.3.1/install/FusionInsight-Spark-3.3.1/spark/jars/streamingClient010/*`, to the **spark.yarn.cluster.driver.extraClassPath** parameter.
- Delete the `original-spark-examples_2.12-3.3.1-xxx.jar` packages from all the server nodes.
- In the **spark-defaults.conf** configuration file on the client, modify (or add and modify) the parameter **spark.driver.userClassPathFirst** to **true**.

#### 1.21.5.5.4 Privilege Control Mechanism of SparkSQL UDF Feature

##### Question

What are the privilege control mechanism for UDF in SparkSQL?

##### Answer

When the SQL statement unable to meet user scenarios, the user can use the UDF feature to do the operations on HDFS or HBase data.

To ensure data security, SparkSQL UDF can only be used by admin users, meanwhile the users must make sure those self-defined functions do not contain malicious codes.

#### 1.21.5.5 Why Does Kafka Fail to Receive the Data Written Back by SLog in to the node where the client is installed as the client installation user.park Streaming?

##### Question

While a running Spark Streaming task is writing data back to Kafka, Kafka cannot receive the written data and Kafka logs contain the following error information:

```
2016-03-02 17:46:19,017 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to /  
10.91.8.208 due to invalid request: Request of length  
122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor  
(Logging.scala:68)  
2016-03-02 17:46:19,155 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /  
10.91.8.208. | kafka.network.Processor (Logging.scala:68)
```

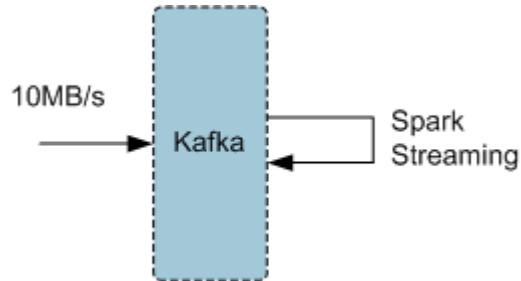
```
2016-03-02 17:46:19,270 | INFO | [kafka-network-thread-21005-0] | Closing socket connection to /  
10.91.8.208 due to invalid request:  
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |  
kafka.network.Processor (Logging.scala:68)  
2016-03-02 17:46:19,513 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to /  
10.91.8.208 due to invalid request:  
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |  
kafka.network.Processor (Logging.scala:68)  
2016-03-02 17:46:19,763 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to /  
10.91.8.208 due to invalid request:  
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. |  
kafka.network.Processor (Logging.scala:68)  
53393 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 50
```

## Answer

As shown in the figure below, the logic defined in Spark Streaming applications is as follows: reading data from Kafka -> executing processing -> writing result data back to Kafka.

Imagine that data is written into Kafka at a data rate of 10 MB/s, the interval (defined in Spark Streaming) between write-back operations is 60s, and a total of 600-MB data needs to be written back into Kafka. If Kafka defines that a maximum of 500-MB data can be received at a time, then the size of written-back data exceeds the threshold, triggering the error information.

**Figure 1-304 Application scenario**



Solution:

- Method 1: On Spark Streaming, reduce the interval between write-back operations to avoid the size of written-back data exceeding the threshold defined by Kafka. The recommended interval is 5-10 seconds.
- Method 2: Increase the threshold defined in Kafka. It is advisable to increase the threshold by adjusting the **socket.request.max.bytes** parameter of Kafka service on FusionInsight Manager.

### 1.21.5.5.6 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?

## Question

A Spark Core application is attempting to collect intensive data and store it into the Driver. When the Driver runs out of memory, the Spark Core application is suspended. Why does the Spark Core application not exit?

The following is the log information displayed at the time of out-of-memory (OOM) error.

```
16/04/19 15:56:22 ERROR Utils: Uncaught exception in thread task-result-getter-2
java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newArray(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$anonfun$run$1.apply$mcV
$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
Exception in thread "task-result-getter-2" java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newArray(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$anonfun$run$1.apply$mcV
$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
```

## Answer

If memory of the Driver is insufficient to store the intensive data that has been collected, the OOM error is reported and the Driver performs garbage collection repeatedly to reclaim the memory occupied by garbage. The Spark Core application remains suspended while garbage collection is under way.

If you expect the Spark Core application to exit forcibly in the event of OOM error, add the following information to the configuration option

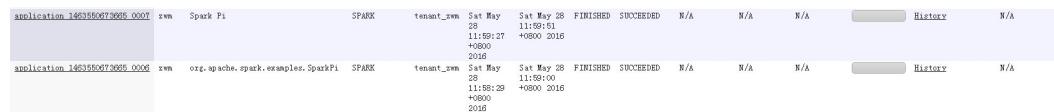
**spark.driver.extraJavaOptions** in the Spark client configuration file **\$SPARK\_HOME/conf/spark-defaults.conf** when you start the Spark Core application for the first time:  
`-XX:OnOutOfMemoryError='kill -9 %p'`

### 1.21.5.5.7 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?

#### Question

The name of the Spark application submitted in yarn-cluster mode does not take effect, whereas the Spark application name submitted in yarn-client mode takes effect. As shown in **Figure 1-305**, the first application is submitted in yarn-client mode and the application name **Spark Pi** takes effect. However, the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode.

**Figure 1-305** Submitting the application



#### Answer

The reason is that the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode. In yarn-client mode, the setAppName is read before the application is registered in yarn. However, in yarn-cluster mode, the setAppName is read after the application registers with yarn, so the name of the second application does not take effect.

#### Troubleshooting solution

When submitting tasks using the spark-submit script, set **--name** the same as the application name in sparkconf.setAppName (appname).

For example, if the application name is **Spark Pi**, run the following command to add the application name after **--name** when submitting the application in yarn-cluster mode:

```
./spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name SparkPi jars/original-spark-examples*.jar 10
```

### 1.21.5.5.8 How to Perform Remote Debugging Using IDEA?

#### Question

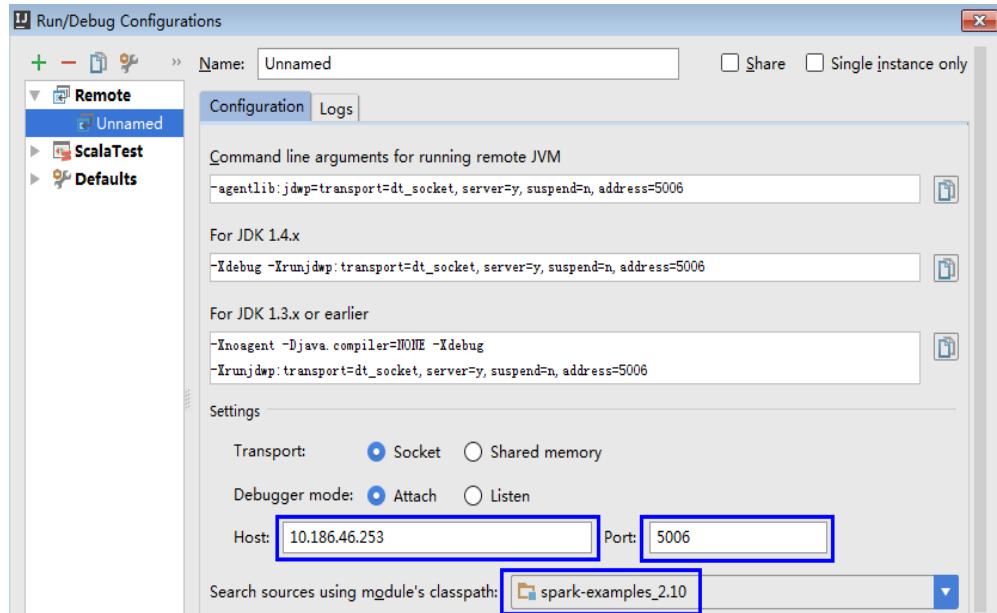
How to perform remote debugging using IDEA during Spark secondary development?

#### Answer

The SparkPi application is used as an example here to illustrate how to perform remote debugging using IDEA.

1. Open the project and choose **Run > Edit Configurations**.
2. In the displayed window, click  at the upper left corner, and then choose **Remote** on the drop-down menu, as shown in [Figure 1-306](#).

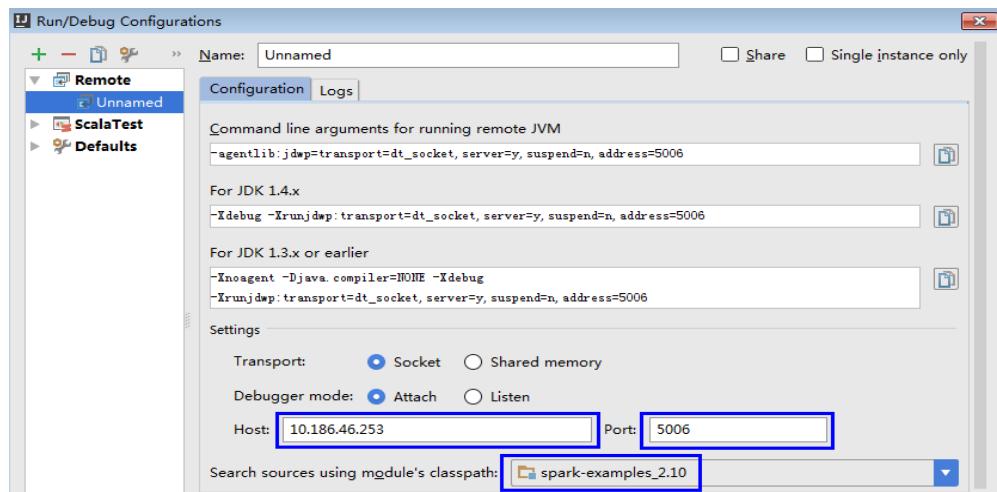
**Figure 1-306** Choosing **Remote**



3. Configure the **Host**, **Port**, and **Search source using module's classpath**, as shown in [Figure 1-307](#).

**Host** indicates the IP address of the Spark client and **Port** indicates the debugging port. Ensure that the port is available on the VM.

**Figure 1-307** Configuring parameters



 NOTE

If the value of **Port** is changed, the debugging command of **For JDK1.4.x** must be changed accordingly. For example, if the value of **Port** is changed to **5006**, the debugging command must be changed to **-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=5006**, which will be used during the startup of Spark.

- Run the following command to remotely start SparkPi on the Spark client:

```
./spark-submit --master yarn-client --driver-java-options "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006" --class org.apache.spark.examples.SparkPi /opt/Fl-Client/Spark/spark/examples/jars/spark-examples_2.12-3.3.1-xxx.jar
```

Change the **--class** and JAR package in the preceding command to the **--class** and JAR package of the actual application. Change the **-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=5006** to the **For JDK1.4.x** debugging command obtained in **3**.

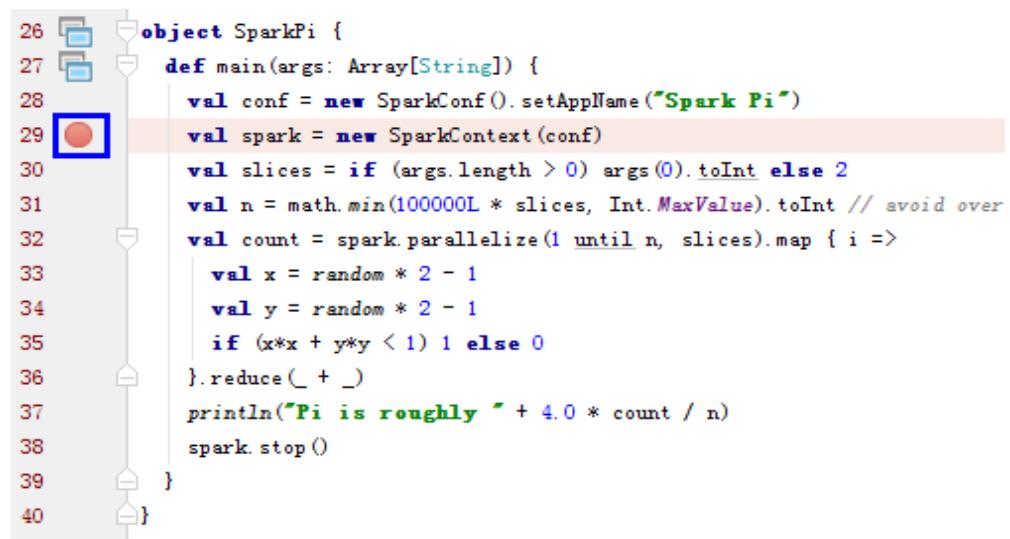
**Figure 1-308** Command for running Spark

```
[root@host2 bin]# ./spark-submit --master yarn-client --driver-java-options "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006" --class org.apache.spark.examples.SparkPi /opt/client/Spark2x/spark/examples/jars/spark-examples_2.11-2.1.0.jar
Listening for transport dt_socket at address: 5006
```

- Set the debugging breakpoint.

Click the blank area on the left of the code editing window to select the breakpoint of code. **Figure 1-309** illustrates how to select the breakpoint of the code in row 29 of **SparkPi.scala**.

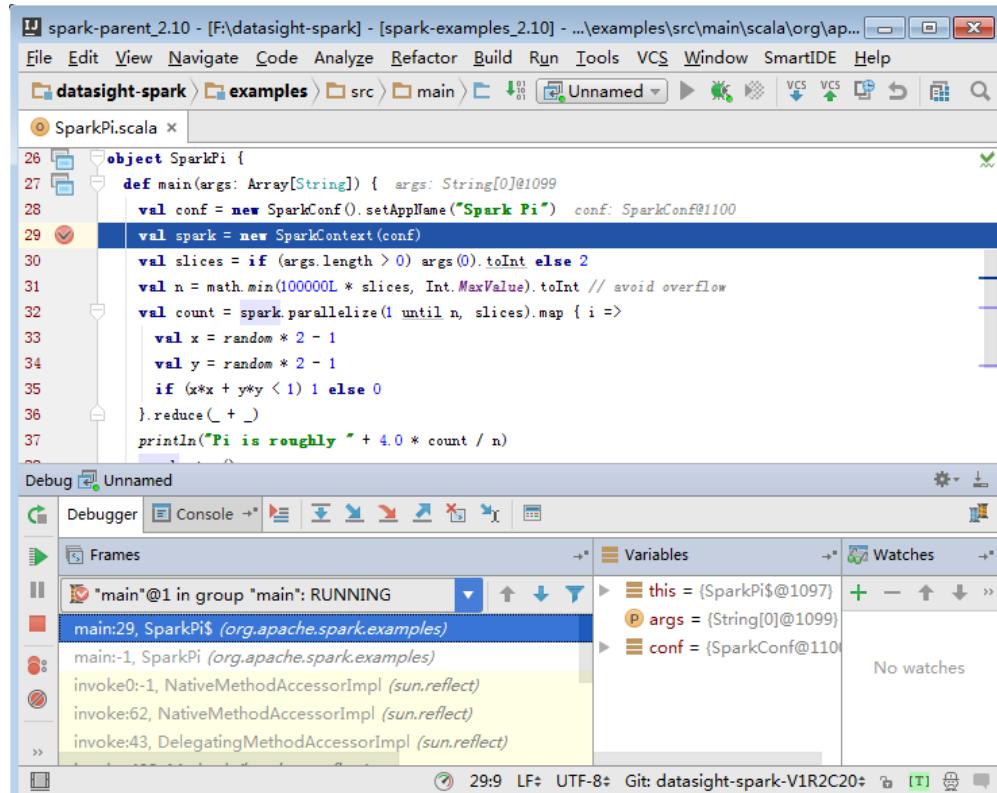
**Figure 1-309** Setting the breakpoint



- Start the debugging.

On the menu bar of IDEA, choose **Run > Debug 'Unnamed'** to open a debugging window. Start the debugging of **SparkPi**, for example, performing step-by-step debugging, checking call stack information, and tracking variable values, as shown in **Figure 1-310**.

Figure 1-310 Debugging



### 1.21.5.5.9 How to Submit the Spark Application Using Java Commands?

#### Question

How to submit the Spark application using Java commands in addition to spark-submit commands?

#### Answer

Use the org.apache.spark.launcher.SparkLauncher class and run Java command to submit the Spark application. The procedure is as follows:

**Step 1** Define the org.apache.spark.launcher.SparkLauncher class. The SparkLauncherJavaExample and SparkLauncherScalaExample are provided by default as sample codes. You can modify the input parameters of sample codes as required.

- If you use Java as the development language, you can compile the SparkLauncher class by referring to the following code:

```
public static void main(String[] args) throws Exception {
    System.out.println("com.huawei.bigdata.spark.examples.SparkLauncherExample <mode>
<jarPath> <app_main_class> <appArgs>");
    SparkLauncher launcher = new SparkLauncher();
    launcher.setMaster(args[0])
        .setAppResource(args[1]) // Specify user app jar path
        .setMainClass(args[2]);
    if (args.length > 3) {
        String[] list = new String[args.length - 3];
        for (int i = 3; i < args.length; i++) {
            list[i-3] = args[i];
        }
    }
}
```

```
// Set app args
launcher.addAppArgs(list);
}

// Launch the app
Process process = launcher.launch();
// Get Spark driver log
new Thread(new ISRRunnable(process.getErrorStream())).start();
int exitCode = process.waitFor();
System.out.println("Finished! Exit code is " + exitCode);
}
```

- If you use Scala as the development language, you can compile the `SparkLauncher` class by referring to the following code:

```
def main(args: Array[String]) {
    println(s"com.huawei.bigdata.spark.examples.SparkLauncherExample <mode> <jarPath>
<app_main_class> <appArgs>")
    val launcher = new SparkLauncher()
    launcher.setMaster(args(0))
    .setAppResource(args(1)) // Specify user app jar path
    .setMainClass(args(2))
    if (args.drop(3).length > 0) {
        // Set app args
        launcher.addAppArgs(args.drop(3): _*)
    }

    // Launch the app
    val process = launcher.launch()
    // Get Spark driver log
    new Thread(new ISRRunnable(process.getErrorStream())).start()
    val exitCode = process.waitFor()
    println(s"Finished! Exit code is $exitCode")
}
```

**Step 2** Develop the Spark application based on the service logic and configure constant values such as the main class of the user-compiled Spark application. For details about different scenarios, see [Developing the Project](#).

- If you use the security mode, you are advised to prepare the security authentication code, service application code, and related configurations according to the security requirements.

#### NOTE

In yarn-cluster mode, security authentication cannot be added to the Spark project. Therefore, users need to add security authentication code or run commands to perform security authentication. There is security authentication code in the sample code. In yarn-cluster mode, modify the corresponding security code before running the operation.

- In normal mode, prepare the service application code and related configurations.

**Step 3** Call the `org.apache.spark.launcher.SparkLauncher.launch()` function to submit user applications.

1. Generate jar packages from the `SparkLauncher` application and user applications, and upload the jar packages to the Spark node of the application. For details about how to generate jar packages, see [Compiling and Running the Application](#).
  - The compilation dependency package of `SparkLauncher` is **spark-launcher\_xxx.cbu.mrs.xxx.jar**. Please obtain it from the **jars** directory of **FusionInsight\_Spark\_8.3.1.tar.gz** in **Software**.

- The compilation dependency packages of user applications vary with the code. You need to load the dependency package based on the compiled code.
- 2. Upload the dependency jar package of the application to a directory, for example, **\$SPARK\_HOME/jars** (the node where the application will run).  
Upload the dependency packages of the SparkLauncher class and the application to the **jars** directory on the client. The dependency package of the sample code has existed in the **jars** directory on the client.

 **NOTE**

If you want to use the Spark Launcher class, the node where the application runs must have the Spark client installed. The running of the Spark Launcher class is dependent on the configured environment variables, running dependency package, and configuration files.

- 3. In the node where the Spark application is running, run the following command to submit the application. Then you can check the running situation through Spark web UI and check the result by obtaining specified files. See [Checking the Commissioning Result](#) for details.

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/  
*:SparkLauncherExample.jar  
com.huawei.bigdata.spark.examples.SparkLauncherExample yarn-  
client /opt/female/FemaleInfoCollection.jar  
com.huawei.bigdata.spark.examples.FemaleInfoCollection <inputPath>
```

----End

### 1.21.5.5.10 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used

#### Question

A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used.

#### Answer

##### Possible Causes

The authentication fails because the duration for creating a JDBC connection on IBM JDK exceeds the timeout duration for user authentication (one day by default).

 **NOTE**

The authentication mechanism of IBM JDK differs from that of Oracle JDK. IBM JDK checks time but does not detect external time update. Therefore, time is not updated even though **relogin** is called.

##### Solution

When one JDBC connection fails, disable this connection, and create a new connection to continue performing previous operations.

### 1.21.5.5.11 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming

#### Question

If ApplicationManager is terminated during data processing in the cluster mode of Structured Streaming, the following information is displayed when the application is executed, indicating an error:

```
2017-05-09 20:46:02,393 | INFO | main |
  client token: Token { kind: YARN_CLIENT_TOKEN, service: }
  diagnostics: User class threw exception: org.apache.spark.sql.AnalysisException: This query does not
  support recovering from checkpoint location. Delete hdfs://hacluster/structuredtest/checkpoint/offsets to
  start over;
  ApplicationMaster host: 10.96.101.170
  ApplicationMaster RPC port: 0
  queue: default
  start time: 1494333891969
  final status: FAILED
  tracking URL: https://9-96-101-191:26001/proxy/application_1493689105146_0052/
  user: spark2x | org.apache.spark.internal.Logging$class.logInfo(Logging.scala:54)
Exception in thread "main" org.apache.spark.SparkException: Application application_1493689105146_0052
finished with failed status
```

#### Answer

**Possible causes:** The error occurs because **recoverFromCheckpointLocation** is determined as **false** but the checkpoint directory is configured.

The value of the **recoverFromCheckpointLocation** parameter is the result of the **outputMode == OutputMode.Complete()** statement in the code. (The default **outputMode** is **append**.)

**Solution:** When compiling an application, you can change the data output mode based on the actual conditions. For operations to change the output mode by calling **outputMode**, see the **outputMode** description of **Spark > Scala** in *MapReduce Service (MRS) 3.3.1-LTS Manager Rest API Reference (for Huawei Cloud Stack 8.3.1)*.

When the output mode is changed to **complete**, the value of **recoverFromCheckpointLocation** is determined as **true**. No error would be indicated if the checkpoint directory is configured at the time.

### 1.21.5.5.12 Restrictions on Restoring the Spark Application from the checkpoint

#### Question

The Spark application can be restored from the checkpoint and continues to execute the task from the breakpoint of the last task, ensuring that data is not lost. However, in some cases, the Spark application fails to be restored from the checkpoint.

#### Answer

The checkpoint contains the object serialization information, task execution status information, and configuration information of the Spark application. Therefore, the Spark application cannot be restored from the checkpoint if the following problems exist:

1. The service code is changed and the serialVersionUID is not specified in the changed class.
2. The internal Spark class is changed and the serialVersionUID is not specified in the changed class.

Besides, some configuration items are stored in the checkpoint. Therefore, if some configuration items of the service are modified, the configuration items may remain unchanged when the service is restored from the checkpoint. Currently, only the following configurations are reloaded when the service is restored from the checkpoint.

```
"spark.yarn.app.id",
"spark.yarn.app.attemptId",
"spark.driver.host",
"spark.driver.bindAddress",
"spark.driver.port",
"spark.master",
"spark.yarn.jars",
"spark.yarn.keytab",
"spark.yarn.principal",
"spark.yarn.credentials.file",
"spark.yarn.credentials.renewalTime",
"spark.yarn.credentials.updateTime",
"spark.ui.filters",
"spark.mesos.driver.frameworkId",
"spark.yarn.jars"
```

## Solution

Manually delete the checkpoint directory and restart the service program.



Deleting a file is a high-risk operation. Ensure that the files are no longer needed before performing this operation.

### 1.21.5.5.13 Support for Third-party JAR Packages on x86 and TaiShan Platforms

#### Question

How to enable Spark to support the third-party JAR packages (for example, custom UDF packages) if these packages have two versions (x86 and TaiShan)?

#### Answer

Use the hybrid solution.

- Step 1** Go to the installation directory of the Spark SparkResource on the server. The cluster may be installed on multiple nodes. You can go to any SparkResource node to go to the installation directory of the SparkResource.
- Step 2** Prepare the .jar package, for example, xx.jar packages for the x86 and TaiShan platforms. Copy the xx.jar packages of x86 and TaiShan to the x86 and TaiShan folders respectively.
- Step 3** Run the following commands in the current directory to compress the JAR packages:

```
zip -qDj spark-archive-x86.zip x86/*
```

```
zip -qDj spark-archive-arm.zip arm/*
```

```
total 20210
lwx----- 2 omm wheel      63 Dec 14 17:57 arm
lwx----- 2 omm wheel      4096 Dec 14 17:57 bin
lwxr-x--- 2 omm ficommon  4096 Dec 14 17:57 carbonlib
rw----- 1 omm wheel      1403 Dec 15 12:51 child.crt
rw----- 1 omm wheel      1097 Dec 15 12:51 child.csr
rw----- 1 omm wheel      6296 Dec 15 12:51 child.keystore
lwx----- 2 omm wheel      326 Dec 14 17:57 conf
lwx----- 3 omm wheel      18 Dec 14 17:54 examples
lwxr-x--- 4 omm ficommon  12288 Dec 14 17:57 jars
rw----- 1 omm wheel      18045 Dec 14 17:54 LICENSE
rw----- 1 omm wheel      26366 Dec 14 17:54 NOTICE
lwx----- 5 omm wheel      129 Dec 14 17:57 python
lwx----- 3 omm wheel      17 Dec 14 17:54 R
rw----- 1 omm wheel      501 Dec 14 17:54 README
rw----- 1 omm wheel      20 Dec 14 17:54 RELEASE
lwx----- 2 omm wheel      4096 Dec 15 17:14 sbin
rw-r--r-- 1 root root    14904892 Dec 15 17:14 spark-archive-2x-arm.zip
rw-r--r-- 1 root root    138881100 Dec 15 17:15 spark-archive-2x-x86.zip
rw----- 1 omm wheel      244 Dec 14 17:57 version.properties
lwx----- 2 omm wheel      63 Dec 14 17:57 x86
lwx----- 2 omm wheel      42 Dec 14 17:54 yarn
```

- Step 4** Run the following command to check the .jar package on which the Spark of HDFS depends:

```
hdfs dfs -ls /user/spark/jars/8.3.1
```



Change the version number 8.3.1 as required.

Run the following commands to move the .jar package files from HDFS, for example, **/tmp**.

```
hdfs dfs -mv /user/spark/jars/8.3.1/spark-archive-arm.zip /tmp
```

```
hdfs dfs -mv /user/spark/jars/8.3.1/spark-archive-x86.zip /tmp
```

- Step 5** Run the following commands to upload the **spark-archive-arm.zip** and **spark-archive-x86.zip** packages in [Step 3](#) to the **/user/spark/jars/8.3.1** directory of HDFS:

```
hdfs dfs -put spark-archive-arm.zip /user/spark/jars/8.3.1/
```

```
hdfs dfs -put spark-archive-x86.zip /user/spark/jars/8.3.1/
```

After the upload is complete, delete local files **spark-archive-arm.zip** and **spark-archive-x86.zip**.

- Step 6** Perform [Step 1](#) to [Step 2](#) for other SparkResource nodes.

- Step 7** Log in to the web UI and restart the jdbcServer instance of Spark.

- Step 8** After the restart, update the client configuration. Copy *xx.jar* for the corresponding platform to the Spark installation directory  **\${install\_home}/Spark/spark/jars** on the client according to client server type (x86 or TaiShan).  **\${install\_home}** indicates the installation path of the client. Replace it with the actual one. If the local installation directory is **/opt/hadoopclient**, copy the corresponding *xx.jar* to the **/opt/hadoopclient/Spark/spark/jars** folder.

----End

### 1.21.5.5.14 What Should I Do If a Large Number of Directories Whose Names Start with **blockmgr-** or **spark-** Exist in the /tmp Directory on the Client Installation Node?

#### Question

After the system runs for a long time, there are many directories whose names start with **blockmgr-** or **spark-** in the **/tmp** directory on the node where the client is installed.

**Figure 1-311** Residual directory example

```
blockmgr-934dc20a-d5f0-4adf-a28f-c376ef0fe01d  
blockmgr-f514f38b-209c-4a65-985a-2a6c61d0ee00  
spark-33f95b4b-be82-4290-bde3-07b76c797085  
spark-988e28a7-0416-4115-8d6e-3a62a75f1f46
```

#### Answer

During the running of Spark tasks, the driver creates a local temporary directory whose name starts with **spark-** for storing service JAR packages and configuration files. In addition, the driver creates a local temporary directory with the name starting with **blockmgr-** for storing block data. The two directories are automatically deleted when the Spark application running is finished.

The path for storing the two directories is preferentially specified by the environment variable **SPARK\_LOCAL\_DIRS**. If the environment variable is not configured, use the value of **spark.local.dir** as the path for storing the directories. If the environment variable and the preceding parameter both are not configured, use the value of **java.io.tmpdir**. By default, **spark.local.dir** is set to **/tmp** on the client. Therefore, the **/tmp** directory is used by default.

In some special cases, for example, the driver process does not exit normally, for example, the **kill -9** command ends the process, or the Java virtual machine crashes. As a result, the directory cannot be deleted and remains in the system.

Currently, only the driver processes in yarn-client mode and local mode may confront the preceding problem. In yarn-cluster mode, the temporary directory of the process in the container is configured as the temporary directory of the container. When the container exits, the container automatically clears the directory. Therefore, this problem does not occur in yarn-cluster mode.

#### Solution

In Linux, you can configure automatic directory clearing for the **/tmp** temporary directory. Alternatively, you can change the value of **spark.local.dir** in the **spark-defaults.conf** configuration file on the client, specify the temporary directory to a specified directory, and configure a clear mechanism for the directory.

### 1.21.5.5.15 Error Code 139 Is Reported When Python Pipeline Runs in the Arm Environment

#### Question

Error code 139 is displayed when the pipeline of the Python plug-in is used on the TaiShan server. The error information is as follows:

```
subprocess exited with status 139
```

#### Answer

The python program uses both **libcrypto.so** and **libssl.so**. If the native library directory of Hadoop is added to **LD\_LIBRARY\_PATH**, the **libcrypto.so** in the **hadoop native** library is used and the **libssl.so** provided by the system is used (because the **hadoop native** directory does not contain this package). The versions of the two libraries do not match. As a result, a segment error occurs when the Python file is running.

#### Solution

Modify the **spark-default.conf** file in the **conf** directory of the Spark client. Clear the values of **spark.driver.extraLibraryPath**, **spark.yarn.cluster.driver.extraLibraryPath**, and **spark.executor.extraLibraryPath**.

### 1.21.5.5.16 What Should I Do If the Structured Streaming Task Submission Way Is Changed?

#### Question

When submitting a structured streaming task, users need to run the **--jars** command to specify the Kafka JAR package path, for example, **--jars /kafkadir/kafka-clients-x.x.x.jar,/kafkadir/kafka\_2.11-x.x.x.jar**. However, in the current version, users need to configure additional items. Otherwise, an error is reported, indicating that the class is not found.

#### Answer

The Spark kernel of the current version depends on the Kafka JAR package, which is used by the structured streaming. Therefore, when submitting a structured streaming task, you need to add the Kafka JAR package path to the library directory of the driver of this task to ensure that the driver can properly load the Kafka package.

#### Solution

1. The following operations need to be performed additionally when a structured streaming task in Yarn-client mode is submitted:

Copy the path of **spark.driver.extraClassPath** in the **spark-default.conf** file in the Spark client directory, and add the Kafka JAR package path to its end. When submitting a structured stream task, add the **--conf** statement to combine these two configuration items. For example, if the Kafka JAR package path is **/kafkadir**, you need to add **--conf**

- spark.driver.extraClassPath=/opt/hadoopclient/Spark/spark/conf:/opt/hadoopclient/Spark/spark/jars/\*:/opt/hadoopclient/Spark/spark/x86/\*:/kafkadir/\*** when submitting the task.
2. The following operations need to be performed additionally when a structured streaming task in **Yarn-cluster** mode is submitted:  
Copy the path of **spark.yarn.cluster.driver.extraClassPath** in the **spark-default.conf** file in the Spark client directory, and add relative paths of Kafka JAR packages to its end. When submitting a structured stream task, add the **--conf** statement to combine these two configuration items. For example, if the Kafka JAR package paths are **kafka-clients-x.x.x.jar** and **kafka\_2.11-x.x.x.jar**, you need to add **--conf spark.yarn.cluster.driver.extraClassPath=/home/huawei/Bigdata/common/runtime/security:/kafka-clients-x.x.x.jar:/kafka\_2.11-x.x.x.jar** when submitting the task.
  3. In the current version, the structured streaming of Spark does not support versions earlier than Kafka2.x. In the upgrade scenario, use the client of earlier versions.

## 1.21.5.5.17 Migrating Spark Streaming's Interconnection from Kafka 0.8 to Kafka 0.10

### Overview

Kafka 0.8 is not recommended for interconnection with Spark 2.3, and is not supported in Spark 3.0.0. If **spark-streaming-kafka-0-8** is used in Spark Streaming after Spark 2.3 is upgraded to Spark 3.1.1, you need to change the dependency to **spark-streaming-kafka-0-10** after the upgrade.

### Difference Analysis

- Maven dependency versions  
After Spark Streaming is interconnected to **spark-streaming-kafka-0-10**, the Maven dependency versions in the code change. The related dependencies are as follows:  
**org.apache.spark:**
  - **spark-core\_2.11:** Change **artifactId** to **spark-core\_2.12** as Scala is upgraded to 2.12.
  - **spark-streaming\_2.11:** Change **artifactId** to **spark-streaming\_2.12** as Scala is upgraded to 2.12.
  - **spark-streaming-kafka-0-8\_2.11:** Replace this with **spark-streaming-kafka-0-10\_2.12**.
  - **spark-streaming-kafkaWriter-0-8\_2.11:** Replace this with **spark-streaming-kafkaWriter-0-10\_2.12**.  
**org.apache.kafka:**
  - **kafka\_2.11:** Upgrade from version 0.8.2.1 to the corresponding version of the cluster.
  - **kafka-clients:** Add this dependency, and its version number should be the same as that of the cluster.
- Related interfaces

**spark-streaming-kafka-0-8** can obtain data from Kafka APIs **Receiver-based Approach** and **Direct Approach (No Receivers)**. In **spark-streaming-kafka-0-10**, only the method using **Direct Approach** is retained.

Differences exist in related APIs. For details, see the sample code **SparkStreamingKafka010ScalaExample** and **SparkStreamingKafka010JavaExample**

- Job submission mode

In security mode, **spark-streaming-kafka-0-8** does not support security authentication, but **spark-streaming-kafka-0-10** does. Security authentication is required when **spark-streaming-kafka-0-10** is used for task submission.

## Solution

- Maven dependency versions

Change the dependency versions referenced in the code. Add dependencies if no one exists.

### **org.apache.spark**

- **spark-core\_2.11**: Change **artifactId** to **spark-core\_2.12** and upgrade its version number to the one of the cluster as Scala is upgraded to 2.12.
- **spark-streaming\_2.11**: Change to **spark-streaming\_2.12**, change the version number to the one of Spark in the cluster, and upgrade the version number to the one of the cluster.
- **spark-streaming-kafka-0-8\_2.11**: Replace it with **spark-streaming-kafka-0-10\_2.12**, change the version number to the one of Spark in the cluster, and upgrade the version number to the one of the cluster.
- **spark-streaming-kafkaWriter-0-8\_2.11**: Replace it with **spark-streaming-kafkaWriter-0-10\_2.12**, change the version number to the one of Spark in the cluster, and upgrade the version number to the one of the cluster.

### **org.apache.kafka**

- **kafka\_2.11**: Change the version number from 0.8.2.1 to the one of Kafka in the cluster.
- **kafka-clients**: Add this dependency, and its version number should be the same as that of the cluster.

- For details about related APIs, see the sample code **SparkStreamingKafka010ScalaExample** and **SparkStreamingKafka010JavaExample**.

- Job submission mode

For clusters in security mode, configure security authentication as follows:

- Set the **allow.everyone.if.no.acl.found** parameter of Kafka Broker to **true**.
- Start Kafka producers. Brokers need to use port **21005**, for example, **IP address of the Kafka service Broker instance:21005**.
- The path of Spark Streaming's Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**, and the path of the Kafka dependency package is **\$SPARK\_HOME/jars/**

**streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the Spark Streaming's Kafka dependency package, for example, **--jars \${files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar);IFS=;echo "\${files[\*]}"}**.

- The running sample depends on the **kafka-clients** component. Therefore, if there is no **kafka-clients-xxx.jar** package in the **\$CLIENT/Spark/spark/jars** directory, copy the **kafka-clients-xxx.jar** file from the **\$CLIENT/Spark/spark/jars/streamingClient010** directory to the **\$CLIENT/Spark/spark/jars** directory. Download the JAR file of the **spark-streaming-kafkaWriter-0-10** cluster version from the Maven repository and save the file to the **\$CLIENT/Spark/spark/jars/streamingClient010** directory.
- When you run the sample, an error is reported indicating that the class does not exist. For details, see [Why the "Class Does not Exist" Error Is Reported While the SparkStresmingKafka Project Is Running?](#).
- New configurations need to be added and command parameters need to be modified.
  - Add the following configuration to the **\$SPARK\_HOME/conf/jaas.conf** file:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=false  
    useTicketCache=true  
    debug=false;  
};
```
  - Add the following configuration to the **\$SPARK\_HOME/conf/zk.conf** file:

```
KafkaClient {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab="/user.keytab"  
    principal=<Username>@<System domain name>  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

Use the **SASL\_PLAINTEXT** protocol port number (for example, **10.111.111.111:21007**) for brokers when consumers are started.

Use the **--files** command to specify the **jaas.conf** and **keytab** files, for example, **--files ./jaas.conf, ./user.keytab**.

## References

Spark Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher): <https://spark.apache.org/docs/3.1.1/streaming-kafka-0-10-integration.html>

Spark Streaming + Kafka Integration Guide (Kafka broker version 0.8.2.1 or higher): <https://spark.apache.org/docs/2.4.5/streaming-kafka-0-8-integration.html>

Modify the Java sample code. The following code snippet is for reference only.

```
FemaleInfoCollectionPrint.java  
// Parameter description:
```

```
// <checkPointDir> is the checkPoint directory.  
// <batchTime> is the interval for Streaming processing in batches.  
// <topics> is topics subscribed in Kafka. Multiple topics are separated by commas (,).  
// <brokers> is the Kafka address for obtaining metadata.  
  
import com.huawei.hadoop.security.KerberosUtil;  
import com.huawei.hadoop.security.LoginUtil;  
import scala.Tuple2;  
import scala.Tuple3;  
import org.apache.hadoop.conf.Configuration;  
import org.apache.kafka.clients.consumer.ConsumerRecord;  
import org.apache.spark.SparkConf;  
import org.apache.spark.api.java.Optional;  
import org.apache.spark.api.java.function.Function;  
import org.apache.spark.streaming.Duration;  
import org.apache.spark.streaming.Durations;  
import org.apache.spark.streaming.api.java.JavaDStream;  
import org.apache.spark.streaming.api.java.JavaInputDStream;  
import org.apache.spark.streaming.api.java.JavaPairDStream;  
import org.apache.spark.streaming.api.java.JavaStreamingContext;  
import org.apache.spark.streaming.kafka010.*;  
import java.util.*;  
  
public class FemaleInfoCollectionPrint {  
    public static void main(String[] args) throws Exception {  
        String userPrincipal = "sparkuser";  
        String userKeytabPath = "/opt/FIclient/user.keytab";  
        String krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf";  
  
        Configuration hadoopConf = new Configuration();  
        LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf);  
  
        String checkPointDir = args[0];  
        String batchTime = args[1];  
        String topics = args[2];  
        String brokers = args[3];  
  
        Duration batchDuration = Durations.seconds(Integer.parseInt(batchTime));  
  
        SparkConf conf = new SparkConf().setAppName("DataSightStreamingExample");  
        JavaStreamingContext jssc = new JavaStreamingContext(conf, batchDuration);  
  
        // Set the CheckPoint directory of Streaming.  
        jssc.checkpoint(checkPointDir);  
  
        // Assemble a Kafka topic list.  
        String[] topicArr = topics.split(",");  
        Set<String> topicsSet = new HashSet<String>(Arrays.asList(topicArr));  
        Map<String, Object> kafkaParams = new HashMap();  
        kafkaParams.put("bootstrap.servers", brokers);  
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");  
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");  
        kafkaParams.put("group.id", "DemoConsumer");  
        kafkaParams.put("security.protocol", "SASL_PLAINTEXT");  
        kafkaParams.put("sasl.kerberos.service.name", "kafka");  
        String principalName = KerberosUtil.getKrb5DomainRealm();  
        kafkaParams.put("kerberos.domain.name", "hadoop." + principalName);  
  
        LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();  
        ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicsSet, kafkaParams);  
  
        // Create a Kafka stream using brokers and topics.  
        // Receive data from Kafka and generate the corresponding DStream.  
        JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(jssc,  
            locationStrategy, consumerStrategy);  
  
        // Obtain the field attribute of each row.  
        JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {  
            @Override
```

```
public String call(ConsumerRecord<String, String> tuple2) throws Exception {
    return tuple2.value();
}
});

JavaDStream<Tuple3<String, String, Integer>> records = lines
.map(new Function<String, Tuple3<String, String, Integer>>() {
    public Tuple3<String, String, Integer> call(String line) throws Exception {
        String[] elems = line.split(",");
        return new Tuple3<String, String, Integer>(elems[0], elems[1], Integer.parseInt(elems[2])));
    }
});

// Filter the data information of the time that female netizens spend online.
JavaDStream<Tuple2<String, Integer>> femaleRecords = records
.filter(new Function<Tuple3<String, String, Integer>, Boolean>() {
    public Boolean call(Tuple3<String, String, Integer> line) throws Exception {
        if (line._2().equals("female")) {
            return true;
        } else {
            return false;
        }
    }
}).map(new Function<Tuple3<String, String, Integer>, Tuple2<String, Integer>>() {
    public Tuple2<String, Integer> call(Tuple3<String, String, Integer> stringStringIntegerTuple3)
        throws Exception {
        return new Tuple2<String, Integer>(stringStringIntegerTuple3._1(),
            stringStringIntegerTuple3._3());
    }
});

// Filter the data of netizens whose consecutive online duration exceeds the threshold.
JavaDStream<Tuple2<String, Integer>> upTimeUser = femaleRecords
.filter(new Function<Tuple2<String, Integer>, Boolean>() {
    public Boolean call(Tuple2<String, Integer> stringIntegerTuple2) throws Exception {
        if (stringIntegerTuple2._2() > 30) {
            return true;
        } else {
            return false;
        }
    }
});

// Print the result.
upTimeUser.print();

// Enable Streaming.
jssc.start();
jssc.awaitTermination();
}

}

JavaDstreamKafkaWriter.java
/*
 * Parameter description:
 * <checkPointDir> is the checkPoint directory.
 * <topics> is topics subscribed in Kafka. Multiple topics are separated by commas (,).
 * <brokers> is the Kafka address for obtaining metadata.
 */

import com.huawei.spark.streaming.kafka010.JavaDStreamKafkaWriterFactory;

import scala.collection.JavaConverters;

import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.streaming.Durations;
```

```
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;

import java.util.*;

public class JavaDstreamKafkaWriter {
    public static void main(String[] args) throws InterruptedException {
        if (args.length != 3) {
            System.err.println("Usage: JavaDstreamKafkaWriter <groupId> <brokers> <topic>");
            System.exit(1);
        }

        final String groupId = args[0];
        final String brokers = args[1];
        final String topic = args[2];

        SparkConf sparkConf = new SparkConf().setAppName("KafkaWriter");

        // Assemble a Kafka topic list.
        Map<String, Object> kafkaParams = new HashMap<String, Object>();
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("value.serializer", "org.apache.kafka.common.serialization.ByteArraySerializer");
        kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        kafkaParams.put("bootstrap.servers", brokers);
        kafkaParams.put("group.id", groupId);
        kafkaParams.put("auto.offset.reset", "smallest");

        // Create context of the Java Spark Streaming.
        JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));

        // Add data to be written to Kafka.
        List<String> sentData = new ArrayList();
        sentData.add("kafka_writer_test_msg_01");
        sentData.add("kafka_writer_test_msg_02");
        sentData.add("kafka_writer_test_msg_03");

        // Create a Java RDD queue.
        Queue<JavaRDD<String>> sent = new LinkedList();
        sent.add(ssc.sparkContext().parallelize(sentData));

        // Create a Java DStream for writing data.
        JavaDStream wStream = ssc.queueStream(sent);

        // Write data to Kafka.
        JavaDStreamKafkaWriterFactory.fromJavaDStream(wStream)
            .writeToKafka(
                JavaConverters.mapAsScalaMapConverter(kafkaParams).asScala(),
                new Function<String, ProducerRecord<String, byte[]>>() {
                    @Override
                    public ProducerRecord<String, byte[]> call(String s) throws Exception {
                        return new ProducerRecord(topic, s.toString().getBytes());
                    }
                });
    }

    ssc.start();
    ssc.awaitTermination();
}
}
```

Modify the Scala sample code. The following code snippet is for reference only.

```
FemaleInfoCollectionPrint.scala
// Parameter description:
// <checkPointDir> is the checkPoint directory.
// <batchTime> is the interval for Streaming processing in batches.
// <topics> is topics subscribed in Kafka. Multiple topics are separated by commas (,).
// <brokers> is the Kafka address for obtaining metadata.

import org.apache.hadoop.conf.Configuration
```

```
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.kafka010._

import com.huawei.hadoop.security.LoginUtil
import com.huawei.hadoop.security.KerberosUtil

object FemaleInfoCollectionPrint {
  def main(args: Array[String]) {
    val userPrincipal = "sparkuser"
    val userKeytabPath = "/opt/FIclient/user.keytab"
    val krb5ConfPath = "/opt/FIclient/KrbClient/kerberos/var/krb5kdc/krb5.conf"

    val hadoopConf: Configuration = new Configuration()
    LoginUtil.login(userPrincipal, userKeytabPath, krb5ConfPath, hadoopConf)

    val Array(checkPointDir, batchTime, topics, brokers) = args

    // Set up a Streaming startup environment.
    val sparkConf = new SparkConf().setAppName("KafkaWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(batchTime.toLong))

    // Set the CheckPoint directory of Streaming.
    //This parameter is mandatory because of existence of the window concept.
    ssc.checkpoint(checkPointDir)

    // Obtain the list of topics used by Kafka.
    val topicArr = topics.split(",")
    val topicSet = topicArr.toSet
    val principalName = KerberosUtil.getKrb5DomainRealm()
    val kafkaParams = Map[String, String](
      "bootstrap.servers" -> brokers,
      "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
      "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
      "group.id" -> "DemoConsumer",
      "security.protocol" -> "SASL_PLAINTEXT",
      "sasl.kerberos.service.name" -> "kafka",
      "kerberos.domain.name" -> ("hadoop." + principalName)
    );

    val locationStrategy = LocationStrategies.PreferConsistent
    val consumerStrategy = ConsumerStrategies.Subscribe[String, String](topicSet, kafkaParams)

    // Create a Kafka stream using brokers and topics.
    // Receive data from Kafka and generate the corresponding DStream.
    val stream = KafkaUtils.createDirectStream[String, String](ssc, locationStrategy, consumerStrategy)
    val lines = stream.transform( rdd =>
      rdd.map(r => (r.value))
    )

    // Obtain the field attribute of each row.
    val records = lines.map(getRecord)
    // Filter the data information of the time that female netizens spend online.
    val femaleRecords = records.filter(_.value._2 == "female")
    .map(x => (x.value._1, x.value._3))
    // Filter netizens whose consecutive online duration exceeds the threshold, and obtain the result.
    femaleRecords.filter(_.value._2 > 30).print()

    // Start Streaming.
    ssc.start()
    ssc.awaitTermination()
  }

  // Obtain field functions.
  def getRecord(line: String): (String, String, Int) = {
    val elems = line.split(",")
    val name = elems(0)
    val sexy = elems(1)
    val time = elems(2).toInt
    (name, sexy, time)
  }
}
```

```
}

}

DstreamKafkaWriter.scala
/*
 * Parameter description:
 * <checkPointDir> is the checkPoint directory.
 * <topics> is topics subscribed in Kafka. Multiple topics are separated by commas (,).
 * <brokers> is the Kafka address for obtaining metadata.
 */

import scala.collection.mutable
import scala.language.postfixOps

import com.huawei.spark.streaming.kafka010.KafkaWriter._
import org.apache.kafka.clients.producer.ProducerRecord
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.{Milliseconds, StreamingContext}

object DstreamKafkaWriter {
def main(args: Array[String]) {

if (args.length != 3) {
System.err.println("Usage: DstreamKafkaWriter <groupId> <brokers> <topic>")
System.exit(1)
}

val Array(groupId, brokers, topic) = args
val sparkConf = new SparkConf().setAppName("KafkaWriter")

// Fill the properties of Kafka.
val kafkaParams = Map[String, String](
"bootstrap.servers" -> brokers,
"value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
"key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
"value.serializer" -> "org.apache.kafka.common.serialization.ByteArraySerializer",
"key.serializer" -> "org.apache.kafka.common.serialization.StringSerializer",
"group.id" -> groupId,
"auto.offset.reset" -> "latest"
)

// Create the context of Streaming.
val ssc = new StreamingContext(sparkConf, Milliseconds(500));

// Add data to be written to Kafka.
val sentData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02",
"kafka_writer_test_msg_03")

// Create an RDD queue.
val sent = new mutable.Queue[RDD[String]]()
sent.enqueue(ssc.sparkContext.makeRDD(sentData))

// Create a DStream for writing data.
val wStream = ssc.queueStream(sent)

// Write data to Kafka.
wStream.writeToKafka(kafkaParams,
(x: String) => new ProducerRecord[String, Array[Byte]](topic, x.getBytes))

// Start Streaming.
ssc.start()
ssc.awaitTermination()
}
}
```

Modify the Producer sample code of Java and Scala. The following code snippet is for reference only.

```
StreamingExampleProducer.java
/*
 * Parameter description:
 * <topics> is topics subscribed in Kafka. Multiple topics are separated by commas (,).
 * <brokers> is the Kafka address for obtaining metadata.
 */

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
import java.io.*;
import java.util.Properties;

public class StreamingExampleProducer {
    public static void main(String[] args) throws IOException {
        if (args.length < 2) {
            printUsage();
        }
        String brokerList = args[0];
        String topic = args[1];
        String filePath = "/home/data/";
        Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokerList);
        props.put(ProducerConfig.CLIENT_ID_CONFIG, "DemoProducer");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        Producer<String, String> producer = new KafkaProducer<String, String>(props);

        for (int m = 0; m < Integer.MAX_VALUE / 2; m++) {
            File dir = new File(filePath);
            File[] files = dir.listFiles();
            if (files != null) {
                for (File file : files) {
                    if (file.isDirectory()) {
                        System.out.println(file.getName() + "This is a directory!");
                    } else {
                        BufferedReader reader = null;
                        reader = new BufferedReader(new FileReader(filePath + file.getName()));
                        String tempString = null;
                        while ((tempString = reader.readLine()) != null) {
                            // Judge empty lines.
                            if (!tempString.isEmpty()) {
                                producer.send(new ProducerRecord<String, String>(topic, tempString));
                            }
                        }
                        // Check that the stream is closed.
                        reader.close();
                    }
                }
            }
            try {
                Thread.sleep(3);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private static void printUsage() {
        System.out.println("Usage: {brokerList} {topic}");
    }
}
```

## 1.22 YARN Development Guide

## 1.22.1 Overview

### Intended Audience

This document is intended for development personnel who are experienced in Java development and want to develop YARN applications.

### YARN Introduction

YARN is a distributed resource management system that is used to improve resource usage in the distributed cluster environment. Resources include memory, I/O, network, and disk resources. YARN is developed to address the shortage of the original MapReduce framework. At the beginning, MapReduce committers periodically modified existing codes. As codes increase and because the original MapReduce framework was designed improperly, modification on the original MapReduce framework becomes more difficult. Therefore, MapReduce committers decided to re-design the MapReduce framework to provide a next-generation MapReduce (MRv2/Yarn) framework that supports high scalability, availability, reliability, backward compatibility, and resource usage. The next-generation MapReduce (MRv2/Yarn) framework supports more computing frameworks in addition to the MapReduce framework.

### Basic Concepts

- **ResourceManager (RM)**

ResourceManager is a global resource manager that manages and allocates resources in the system. ResourceManager consists of two components: Scheduler and Applications Manager.

- **ApplicationMaster (AM)**

Each application submitted by users includes an ApplicationMaster. The ApplicationMaster provides the following functions:

- Negotiates with the ResourceManager Scheduler to obtain resources (represented by Containers).
- Allocates resource to internal tasks.
- Communicates with NodeManager to start or stop tasks.
- Monitors all tasks with the running status, and applies for resources again for tasks when tasks fail to run to restart the tasks.

- **NodeManager (NM)**

NodeManager is the resource and task manager of each node. On one hand, NodeManager periodically reports resource usage of the local node and the running status of each Container to ResourceManager. On the other hand, NodeManager receives and processes requests from ApplicationMaster for starting or stopping Containers.

- **Container**

Container is a resource abstract in YARN. Container encapsulates multidimensional resources of a node, such as memory, CPU, disk, and network resources. When ApplicationMaster applies for resources from ResourceManager, ResourceManager returns resources in a Container to ApplicationMaster.

## 1.22.2 Interfaces

### 1.22.2.1 Command

You can use YARN commands to perform operations on YARN clusters, such as starting ResourceManager, submitting applications, killing applications, querying node status, and downloading container logs.

For details about the commands, see:

<http://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/YarnCommands.html>

### Common Commands

YARN commands can be used by common users and administrators. Common users can run a few of YARN commands, such as **jar** and **logs**. Administrators have the permission to run most YARN commands.

Users can run the following command to query usage of YARN.

**yarn --help**

Usage: Go to any directory on the YARN client, execute source command to import the environment variables, and run the command. The command format is as follows.

**yarn [--config confdir] COMMAND**

**Table 1-205** describes the commands.



The version 8.3.1 is used as an example. Replace it with the actual version number.

**Table 1-205** Common commands

| Command         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| resourcemanager | <p>Runs a ResourceManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user <b>omm</b> (the client must have the execute permission of user <b>omm</b>), export environment variables first.</p> <ul style="list-style-type: none"><li>• <b>export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_ResourceManager/etc</b></li><li>• <b>export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_ResourceManager/etc</b></li></ul> |

| Command                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nodemanager                  | <p>Runs a NodeManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user <b>omm</b> (the client must have the execute permission of user <b>omm</b>), export environment variables first.</p> <ul style="list-style-type: none"> <li>• <code>export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_NodeManager/etc</code></li> <li>• <code>export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_NodeManager/etc</code></li> </ul> |
| rmadmin                      | Runs administrator's tool for dynamically updating information.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| version                      | Displays the version.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| jar <jar>                    | Runs a JAR file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| logs                         | Obtains container logs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| classpath                    | Displays the class path of the Hadoop JAR package and other library files.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| daemonlog                    | Obtains or sets the service log level.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| CLASSNAME                    | Runs a class named by <i>CLASSNAME</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| top                          | Runs the cluster usage monitor tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| -Dmapreduce.job.hdfs-servers | <p>If OBS is connected, but the server still uses HDFS, you need to explicitly use this parameter in the command line to specify the HDFS address. The format is <code>hdfs:// {NAMESERVICE}</code>. Replace <i>NAMESERVICE</i> with the HDFS NameService name.</p> <p>If the current HDFS has multiple NameService instances, you need to specify all NameService instances and separate them with commas (,), for example, <code>hdfs://nameservice1,hdfs://nameservice2</code>.</p>                            |

## Superior Scheduler Command

Superior Scheduler Engine provides a CLI that displays detail information of Superior Scheduler Engine. For executing superior command we need to use the "`<HADOOP_HOME>/bin/superior`" script.

Here is the format of "superior" command.

```
<HADOOP_HOME>/bin/superior

Usage: superior [COMMAND | -help]
Where COMMAND is one of:
resourcepool      prints resource pool status
queue            prints queue status
```

|                    |                                                   |
|--------------------|---------------------------------------------------|
| application policy | prints application status<br>prints policy status |
|--------------------|---------------------------------------------------|

Most commands print help when invoked without parameters.

- Superior **resourcepool** command:

This command displays resourcepool and associated policy related status and configuration information.

#### NOTE

Superior resourcepool command can be used only by admin user and user with yarn admin rights.

Usage output:

```
>superior resourcepool

Usage: resourcepool [-help]
                  [-list]
                  [-status <resourcepoolname>]
-helpprints resource pool usage
-listprints all resource pool summary report
-status <resourcepoolname> prints status and configuration of specified
resource pool
```

- **resourcepool -list** prints resource pool summary in a table format. Here is an example:

```
> superior resourcepool -list
NAME      NUMBER_MEMBER   TOTAL_RESOURCE      AVAILABLE_RESOURCE
Pool1      4              vcores 30,memory 1000  vcores 21,memory 80
Pool2      100             vcores 100,memory 12800 vcores 30,memory 1000
default    2              vcores 64,memory 128   vcores 40,memory 28
```

- **resourcepool -status <resourcepoolname>** prints resource pool detail information in a list format. Here is an example:

```
> superior resourcepool -status default
NAME: default
DESCRIPTION: System generated resource pool
TOTAL_RESOURCE: vcores 64,memory 128
AVAILABLE_RESOURCE: vcores 40,memory 28
NUMBER_MEMBER: 2
MEMBERS: node1,node2
CONFIGURATION:
|-- RESOURCE_SELECT:
|__ RESOURCES:
```

- Superior **queue** command

This command displays hierarchy queue information.

Usage output:

```
>superior queue

Usage: queue [-help]
              [-list] [-e] [[-name <queue_name>] [-r|-c]]
              [-status <queue_name>]
-c           only work with -name <queue_name> option. If this
option is used, command will print information of
specified queue and its direct children.
-e           only work with -list or -list -name option. If
this option is used, command will print effective
state of specified queue and all of its
descendants.
-help        prints queue sub command usage
-list       prints queue summary report. This option can work
with -name <queue_name> and -r options.
-name <queue_name> print specified queue, this can work with -r
option. By default, it will print queue's own
```

- information. When -r is defined, command will print all of its descendant queues. When -c is defined, it will print its direct children queues.
- r only work with -name <queue\_name> option. If this option is used, command will print information of specified queue and all of its descendants.
  - status <queue\_name> prints status of specified queue
  - **queue -list** prints a table format of queue summary information. When command displays, command will display them based on queue hierarchy fashion. With SUBMIT ACL or ADMIN ACL rights for queue, user will be able to see queues. Here is an example:  

```
> superior queue -list
NAME      STATE      NRUN_APP    NPEND_APP    NRUN_CONTAINER
NPEND_REQUEST  RES_INUSE      RES_REQUEST
root      OPEN|ACTIVE   10        20          100          200          vcores 100,memory
1000     vcores 200,memory 2000
root.Q1    OPEN|ACTIVE   5         10          50           100          vcores 50,memory
500      vcores 100,memory 1000
root.Q1.Q11 OPEN|ACTIVE   5         10          50           100          vcores 50,memory
500      vcores 100,memory 1000
root.Q1.Q12 CLOSE|INACTIVE 0         0           0            0            vcores 0,memory
0        vcores 0,memory 0
root.Q2    OPEN|INACTIVE 5         10          50           100          vcores 50,memory
500      vcores 100,memory 1000
root.Q2.Q21 OPEN|ACTIVE   5         10          50           100          vcores 50,memory
500      vcores 100,memory 1000
```
  - **queue -list -name root.Q1** will display root.Q1 only.  

```
> superior queue -list -name root.Q1
NAME      STATE      NRUN_APP    NPEND_APP    NRUN_CONTAINER
NPEND_REQUEST  RES_INUSE      RES_REQUEST
root.Q1    OPEN|ACTIVE   5         10          50           100          vcores 50,memory
500      vcores 100,memory 1000
```
  - **queue -list -name root.Q1 -r** will print out root.Q1 and all of its descendants.  

```
> superior queue -list -name root.Q1 -r
NAME      STATE      NRUN_APP    NPEND_APP    NRUN_CONTAINER
NPEND_REQUEST  RES_INUSE      RES_REQUEST
root.Q1    OPEN|ACTIVE   5         10          50           100          vcores 50,memory
500      vcores 100,memory 1000
root.Q1.Q11 OPEN|ACTIVE   5         10          50           100          vcores 50,memory
500      vcores 100,memory 1000
root.Q1.Q12 CLOSE|INACTIVE 0         0           0            0            vcores 0,memory
0        vcores 0,memory 0
```
  - **queue -list -name root -c** will print out root and all of its direct children  

```
> superior queue -list -name root -c
NAME      STATE      NRUN_APP    NPEND_APP    NRUN_CONTAINER
NPEND_REQUEST  RES_INUSE      RES_REQUEST
root      OPEN|ACTIVE   10        20          100          200          vcores
100,memory 1000  vcores 200,memory 2000
root.Q1    OPEN|ACTIVE   5         10          50           100          vcores
500      vcores 100,memory 1000
root.Q2    OPEN|INACTIVE 5         10          50           100          vcores
500      vcores 100,memory 1000
```
  - **queue -status <queue\_name>** will display detail queue status and configuration.  

With SUBMIT ACL, user will be able to see details except ACLS of queue. User with ADMIN ACL rights for queue will be able to see queue details including ACL.

```
> superior queue -status root.Q1
NAME: root.Q1
OPEN_STATE:CLOSED
ACTIVE_STATE: INACTIVE
EOPEN_STATE: CLOSED
```

```
EACTIVE_STATE: INACTIVE
LEAF_QUEUE: Yes
NUMBER_PENDING_APPLICATION: 100
NUMBER_RUNNING_APPLICATION: 10
NUMBER_PENDING_REQUEST: 10
NUMBER_RUNNING_CONTAINER: 10
NUMBER_RESERVED_CONTAINER: 0
RESOURCE_REQUEST: vcores 3,memory 3072
RESOURCE_INUSE: vcores 2,memory 2048
RESOURCE_RESERVED: vcores 0,memory 0
CONFIGURATION:
|-- DESCRIPTION: Spark session queue
|-- MAX_PENDING_APPLICATION: 10000
|--MAX_RUNNING_APPLICATION: 1000
|--ALLOCATION_ORDER_POLICY: FIFO
|--DEFAULT_RESOURCE_SELECT: label1
|--MAX_MASTER_SHARE: 10%
|--MAX_RUNNING_APPLICATION_PER_USER : -1
|--MAX_ALLOCATION_UNIT: vcores 32,memory 12800
|--ACL_USERS: user1,user2
|--ACL_USERGROUPS: usergroup1,usergroup2
|-- ACL ADMINS: user1
|--ACL_ADMINGROUPS: usergroup1
```

- Superior ***application*** command

This command displays application related information.

Usage output:

```
>superior application

Usage: application [-help]
    [-list]
    [-status <application_id>]
-helpprints application sub command usage
-listprints all application summary report
-status <application_id> prints status of specified application
```

With the view access rights to application user can see application related information.

- ***application -list*** provides summary information of all application in a table format. Here is an example:

```
> superior application -list
   ID          QUEUE      USER  NRUN_CONTAINER
NPEND_REQUEST  NRSV_CONTAINER  RES_INUSE
RES_REQUEST    RES_RESERVED
application_1482743319705_0005  root.SEQ.queueB  hbase  1
100           0           vcores 1,memory 1536      vcores 2000,memory
409600         vcores 0,memory 0
application_1482743319705_0006  root.SEQ.queueB  hbase  0
1             0           vcores 0,memory 0      vcores 1,memory
1536          vcores 0,memory 0
```

- ***application -status <app\_id>*** command displays detail information of specified application. Here is an example:

```
> superior application -status application_1443067302606_0609
ID: application_1443067302606_0609
QUEUE: root.Q1.Q11
USER: cchen
RESOURCE_REQUEST: vcores 3,memory 3072
RESOURCE_INUSE: vcores 2,memory 2048
RESOURCE_RESERVED:vcores 1, memory 1024
NUMBER_RUNNING_CONTAINER: 2
NUMBER_PENDING_REQUEST: 3
NUMBER_RESERVED_CONTAINER: 1
MASTER_CONTAINER_ID: application_1443067302606_0609_01
MASTER_CONTAINER_RESOURCE: node1.domain.com
BLACKLIST: node5,node8
DEMANDS:
```

```
-- PRIORITY: 20
|-- MASTER: true
|-- CAPABILITY: vcores 2, memory 2048
|-- COUNT: 1
|-- RESERVED_RES : vcores 1, memory 1024
|-- RELAXLOCALITY: true
|-- LOCALITY: node1/1
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "application limit reached"
|-- ID: application_1443067302606_0609_03
|-- RESOURCE: node1.domain.com
|-- RESERVED_RES: vcores 1, memory 1024
|
|--PRIORITY: 1
|-- MASTER: false
|-- CAPABILITY: vcores 1, memory 1024
|-- COUNT: 2
|-- RESERVED_RES: vcores 0, memory 0
|-- RELAXLOCALITY: true
|-- LOCALITY: node1/1, node2/1, rackA/2
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "no available resource"
CONTAINERS:
|-- ID: application_1443067302606_0609_01
|-- RESOURCE: node1.domain.com
|-- CAPABILITY: vcores 1, memory 1024
|
|-- ID: application_1443067302606_0609_02
|-- RESOURCE: node2.domain.com
|-- CAPABILITY: vcores 1, memory 1024
```

- **Superior *policy* command**

This command displays policy related information.

 **NOTE**

Superior policy command can be used only by admin user and user with yarn admin rights.

**Usage output:**

```
>superior policy

Usage: policy [-help]
              [-list <resourcepoolname>] [-u] [-detail]
              [-status <resourcepoolname>]
-detailed      only work with -list option to show a
               summary information of resource pool
               distribution on queues, including reserve,
               minimum and maximum
-help          prints policy sub command usage
-list <resourcepoolname>  prints a summary information of resource
                           pool distribution on queue
-status <resourcepoolname> prints pool distribution policy
               configuration and status of specified
               resource pool
-u             only work with -list option to show a
               summary information of resource pool
               distribution on queues and also user
               accounts
```

- ***policy -list <resourcepoolname>*** print out a summary of queue distribution information. Here is an example:

```
>superior policy -list default
NAME: default
TOTAL_RESOURCE: vcores 16, memory 16384
AVAILABLE_RESOURCE: vcores 16, memory 16384

NAMERES_INUSERES_REQUEST
root.defaultvcores 0, memory 0 vcores 0, memory 0
```

```
root.productionvcores 0,memory 0vcores 0,memory 0  
root.production.BU1vcores 0,memory 0vcores 0,memory 0  
root.production.BU2 vcores 0,memory 0vcores 0,memory 0
```

- ***policy -list <resourcepoolname> -u*** prints out also user level summary information

```
> superior policy -list default -u  
NAME: default  
TOTAL_RESOURCE: vcores 16,memory 16384  
AVAILABLE_RESOURCE: vcores 16,memory 16384
```

```
NAMERES_INUSERES_REQUEST  
root.defaultvcores 0,memory 0vcores 0,memory 0  
root.default.[_others_]vcores 0,memory 0vcores 0,memory 0  
root.productionvcores 0,memory 0vcores 0,memory 0  
root.production.BU1vcores 0,memory 0vcores 0,memory 0  
root.production.BU1.[_others_]vcores 0,memory 0vcores 0,memory 0  
root.production.BU2vcores 0,memory 0vcores 0,memory 0  
root.production.BU2.[_others_]vcores 0,memory 0vcores 0,memory 0
```

- ***policy -status <resourcepoolname>*** print out policy detail of specified resource pool. Here is an example:

```
> superior policy -status pool1  
NAME: pool1  
TOTAL_RESOURCE: vcores 64,memory 128  
AVAILABLE_RESOURCE: vcores 40,memory 28  
QUEUES:  
|-- NAME: root.Q1  
|-- RESOURCE_USE: vcores 20, memory 1000  
|-- RESOURCE_REQUEST: vcores 2,memory 100  
|--RESERVE: vcores 10, memory 4096  
|--MINIMUM: vcore 11, memory 4096  
|--MAXIMUM: vcores 500, memory 100000  
|--CONFIGURATION:  
|-- SHARE: 50%  
|-- RESERVE: vcores 10, memory 4096  
|-- MINIMUM: vcores 11, memory 4096  
|-- MAXIMUM: vcores 500, memory 100000  
|-- QUEUES:  
|-- NAME: root.Q1.Q11  
|-- RESOURCE_USE: vcores 15, memory, 500  
|-- RESOURCE_REQUEST: vcores 1, memory 50  
|-- RESERVE: vcores 0, memory 0  
|-- MINIMUM: vcores 0, memory 0  
|-- MAXIMUM: vcores -1, memory -1  
|-- USER_ACCOUNTS:  
|-- NAME: user1  
|-- RESOURCE_USE: vcores 1, memory 10  
|-- RESOURCE_REQUEST: vcores 1, memory 50  
|  
|-- NAME: OTHERS  
|--RESOURCE_USE: vcores 0, memory 0  
|-- RESOURCE_REQUEST: vcores 0, memory 0  
|-- CONFIGURATION:  
|-- SHARE: 100%  
|-- USER_POLICY:  
|-- NAME: user1  
|-- WEIGHT: 10  
|  
|-- NAME: OTHERS  
|-- WEIGHT: 1  
|-- MAXIMUM: vcores 10, memory 1000
```

### 1.22.2.2 Java API

For details about YARN application programming interfaces (APIs), see:

<http://hadoop.apache.org/docs/r3.3.1/api/index.html>.

## Common Interfaces

Common YARN Java classes are as follows:

- **ApplicationClientProtocol**

This class is used between the client and ResourceManager. The client submits applications to ResourceManager, queries the running status of applications, and kills applications using this protocol.

**Table 1-206** Common interfaces of ApplicationClientProtocol

| Interface                                                                 | Description                                                                                                                               |
|---------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| forceKillApplication(KillApplicationRequest request)                      | The client requests ResourceManager to stop a submitted task through this interface.                                                      |
| getApplicationAttemptReport(GetApplicationAttemptReportRequest request)   | The client obtains the report of specified application attempts from ResourceManager through this interface.                              |
| getApplicationAttemptsReport(GetApplicationAttemptsReportRequest request) | The client obtains the report of all application attempts from ResourceManager through this interface.                                    |
| getApplicationReport(GetApplicationReportRequest request)                 | The client obtains an application report from ResourceManager through this interface.                                                     |
| getApplications(GetApplicationsRequest request)                           | The client obtains information about applications that meet filtering conditions from ResourceManager through this interface.             |
| getClusterMetrics(GetClusterMetricsRequest request)                       | The client obtains metrics of clusters from ResourceManager through this interface.                                                       |
| getClusterNodes(GetClusterNodesRequest request)                           | The client obtains information about all nodes in a cluster from ResourceManager through this interface.                                  |
| getContainerReport(GetContainerReportRequest request)                     | The client obtains the report of a Container from ResourceManager through this interface.                                                 |
| getContainers(GetContainersRequest request)                               | The client obtains the report of all Containers of an application attempt from ResourceManager through this interface.                    |
| getDelegationToken(GetDelegationTokenRequest request)                     | The client obtains the delegation token through this interface. The delegation token is used by the container to access related services. |

| Interface                                                               | Description                                                                                                                |
|-------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| getNewApplication(GetNewApplicationRequest request)                     | The client obtains a new application ID through this interface for submitting a new application.                           |
| getQueueInfo(GetQueueInfoRequest request)                               | The client obtains queue information from ResourceManager through this interface.                                          |
| getQueueUserAcls(GetQueueUserAclsInfoRequest request)                   | The client obtains queue access permission information about the current user from ResourceManager through this interface. |
| moveApplicationAcrossQueues(MoveApplicationAcrossQueuesRequest request) | This interface is used to move an application to a new queue.                                                              |
| submitApplication(SubmitApplicationRequest request)                     | The client submits a new application to ResourceManager through this interface.                                            |

- ApplicationMasterProtocol

This class is used between ApplicationMaster and ResourceManager. ApplicationMaster registers with ResourceManager, and applies for resources and obtains the running status of each task from ResourceManager using this protocol.

**Table 1-207** Common interfaces of ApplicationMasterProtocol

| Interface                                                           | Description                                                                                      |
|---------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| allocate(AllocateRequest request)                                   | ApplicationMaster submits a resource allocation request through this interface.                  |
| finishApplicationMaster(FinishApplicationMasterRequest request)     | ApplicationMaster notifies ResourceManager of running success or failure through this interface. |
| registerApplicationMaster(RegisterApplicationMasterRequest request) | ApplicationMaster registers with ResourceManager through this interface.                         |

- ContainerManagementProtocol

This class is used between ApplicationMaster and NodeManager. ApplicationMaster requests NodeManager to start or terminate a Container or queries the Container running status using this protocol.

**Table 1-208** Common interfaces of ContainerManagementProtocol

| Interface                                                 | Description                                                                                             |
|-----------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| getContainerStatuses(GetContainerStatusesRequest request) | ApplicationMaster obtains the current status of the Containers from NodeManager through this interface. |
| startContainers(StartContainersRequest request)           | ApplicationMaster requests NodeManager to start Containers through this interface.                      |
| stopContainers(StopContainersRequest request)             | ApplicationMaster requests NodeManager to stop a series of allocated Containers through this interface. |

### 1.22.2.3 REST API

#### Function Description

Users can use the application programming interface (API) of Representational State Transfer (REST) to query more information about YARN jobs. Currently, on the REST API provided by YARN, you can only query some resources or jobs. For details of the HTTP REST API, see the following official guidelines:

[http://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/  
WebServicesIntro.html](http://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/WebServicesIntro.html)

#### Preparing Running Environment

1. Install a client on the node. For example, install a client in the `/opt/hadoopclient` directory. See details in "Software Installation > Initial Configuration > Configuring Client > Installing a Client".
2. Go to the `/opt/hadoopclient` directory where the client is installed and run the following commands to initiate environment variables:

```
source bigdata_env  
kinit component service user
```



The validity duration of kinit authentication is 24 hours. After 24 hours, you need to re-authenticate the sample with the kinit to restart the sample.

3. HTTPS-based access is different from HTTP-based access. When you access Yarn using HTTPS, you must ensure that the SSL protocol supported by the `curl` command is supported by the cluster because SSL security encryption is used. If the cluster does not support the SSL protocol, change the SSL protocol in the cluster. For example, if the `Curl` command only supports the TLSv1 protocol, modify the protocol configuration performing following measures:

[Accessing FusionInsight Manager of an MRS Cluster](#) and choose **Cluster > Name of the desired cluster > Service > Yarn > Configuration > All Configurations**. Type `hadoop.ssl.enabled.protocols` in the research box, check whether the parameter value contains `TLSv1`. If the parameter value does not contain `TLSv1`, add `TLSv1` in the `hadoop.ssl.enabled.protocols`

configuration item, and clear the value of `ssl.server.exclude.cipher.list`. Otherwise, Yarn cannot be accessed by using HTTPS. Then click **Save Configuration** and select **Restart the affected services or instances**. Restart the service.

 **NOTE**

TLSv1 has security vulnerabilities. Exercise caution when using it.

## Procedure

### 1. Query the information of the jobs that run on the Yarn.

- Command:

```
curl -k -i --negotiate -u : "https://10-120-85-2:26001/ws/v1/cluster/apps/"
```

 **NOTE**

- **10-120-85-2**: host name of the active ResourceManager node.

You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the **Host Name of ResourceManager(Active)**.

- **26001**: port number of the ResourceManager.

You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and obtain the value of `yarn.resourcemanager.webapp.https.port`.

- If users have the admin permission, they can check jobs in some columns. To grant users the admin permission on the columns, see sections of "Creating a YARN Role" in the *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

 **NOTE**

If the current component uses Ranger for permission control, you need to configure permission management policies based on Ranger.

- The result is displayed as follows:

```
{
  "apps": [
    "app": [
      {
        "id": "application_1461743120947_0001",
        "user": "spark",
        "name": "Spark-JDBCServer",
        "queue": "default",
        "state": "RUNNING",
        "finalStatus": "UNDEFINED",
        "progress": 10,
        "trackingUI": "ApplicationMaster",
        "trackingUrl": "https://10-120-85-2:26001/proxy/application_1461743120947_0001/",
        "diagnostics": "AM is launched. ",
        "clusterId": 1461743120947,
        "applicationType": "SPARK",
        "applicationTags": "",
        "startedTime": 1461804906260,
        "finishedTime": 0,
        "elapsedTime": 6888848,
        "amContainerLogs": "https://10-120-85-2:26010/node/containerlogs/container_e12_1461743120947_0001_01_000001/spark",
        "amHostHttpAddress": "10-120-85-2:26010",
        "allocatedMB": 1024,
        "allocatedVCores": 1,
        "runningContainers": 1,
      }
    ]
  ]
}
```

```
"memorySeconds": 7053309,
"vcoreSeconds": 6887,
"preemptedResourceMB": 0,
"preemptedResourceVCores": 0,
"numNonAMContainerPreempted": 0,
"numAMContainerPreempted": 0,
"resourceRequests": [
  {
    "capability": {
      "memory": 1024,
      "virtualCores": 1
    },
    "nodeLabelExpression": "",
    "numContainers": 0,
    "priority": {
      "priority": 0
    },
    "relaxLocality": true,
    "resourceName": "*"
  }
],
"logAggregationStatus": "NOT_START",
"amNodeLabelExpression": ""
},
{
  "id": "application_1461722876897_0002",
  "user": "admin",
  "name": "QuasiMonteCarlo",
  "queue": "default",
  "state": "FINISHED",
  "finalStatus": "SUCCEEDED",
  "progress": 100,
  "trackingUI": "History",
  "trackingUrl": "https://10-120-85-2:26001/proxy/application_1461722876897_0002/",
  "diagnostics": "Attempt recovered after RM restart",
  "clusterId": 1461743120947,
  "applicationType": "MAPREDUCE",
  "applicationTags": "",
  "startedTime": 1461741052993,
  "finishedTime": 1461741079483,
  "elapsedTime": 26490,
  "amContainerLogs": "https://10-120-85-2:26010/node/containerlogs/
container_e11_1461722876897_0002_01_000001/admin",
  "amHostHttpAddress": "10-120-85-2:26010",
  "allocatedMB": -1,
  "allocatedVCores": -1,
  "runningContainers": -1,
  "memorySeconds": 158664,
  "vcoreSeconds": 52,
  "preemptedResourceMB": 0,
  "preemptedResourceVCores": 0,
  "numNonAMContainerPreempted": 0,
  "numAMContainerPreempted": 0,
  "amNodeLabelExpression": ""
}
]
}
```

- Result analysis:

On the interface, you can query the information of jobs that are running on the YARN and obtain the common information that is displayed in [Table 1](#).

**Table 1-209** Common information of jobs running on Yarn

| Information     | Description                                                  |
|-----------------|--------------------------------------------------------------|
| user            | Indicates the user who runs the job.                         |
| applicationType | Indicates the application types, such as MAPREDUCE or SPARK. |
| finalStatus     | Indicates whether a job is executed successfully.            |
| elapsedTime     | Indicates the time to run a job.                             |

## 2. Obtain the overall information of YARN resources.

- Command:

```
curl -k -i --negotiate -u : "https://10-120-85-102:26001/ws/v1/cluster/metrics"
```

- The result is displayed as follows:

```
{
  "clusterMetrics": {
    "appsSubmitted": 2,
    "appsCompleted": 1,
    "appsPending": 0,
    "appsRunning": 1,
    "appsFailed": 0,
    "appsKilled": 0,
    "reservedMB": 0,
    "availableMB": 23552,
    "allocatedMB": 1024,
    "reservedVirtualCores": 0,
    "availableVirtualCores": 23,
    "allocatedVirtualCores": 1,
    "containersAllocated": 1,
    "containersReserved": 0,
    "containersPending": 0,
    "totalMB": 24576,
    "totalVirtualCores": 24,
    "totalNodes": 3,
    "lostNodes": 0,
    "unhealthyNodes": 0,
    "decommissionedNodes": 0,
    "rebootedNodes": 0,
    "activeNodes": 3,
    "rmMainQueueSize": 0,
    "schedulerQueueSize": 0,
    "stateStoreQueueSize": 0
  }
}
```

- Result analysis:

On the interface, users can query the common information of jobs that are running in the cluster, as displayed in **Table 2**.

**Table 1-210** Common information of jobs running in the cluster

| Information   | Description                                            |
|---------------|--------------------------------------------------------|
| appsSubmitted | Indicates the number of jobs that have been submitted. |

| Information       | Description                                            |
|-------------------|--------------------------------------------------------|
| appsCompleted     | Indicates the number of jobs that have been completed. |
| appsPending       | Indicates the number of jobs that have been suspended. |
| appsRunning       | Indicates the number of jobs that are running.         |
| appsFailed        | Indicates the number of jobs that have failed.         |
| appsKilled        | Indicates the number of jobs that have been killed.    |
| totalMB           | Indicates the total memory of Yarn resources.          |
| totalVirtualCores | Indicates the total VCores of Yarn resources.          |

#### 1.22.2.4 REST APIs of Superior Scheduler

##### Function Description

The REST/HTTP server is part of Superior Scheduler on YARN Resource Manager host and leverage existing YARN resource manager web service port. In the section below, we will denote this YARN *address:port* as *SS\_REST\_SERVER*.

Below uses HTTPS as part of URL and only HTTPS will be supported.

##### Superior Scheduler Interfaces

- **Query Application**
  - Query \*all\* application within scheduler engine.
    - URL  
GET `https://<SS_REST_SERVER>/ws/v1/sscheduler/applications/list`
    -  **NOTE**

`SS_REST_SERVER` indicates *ResourceManager IP address:Port number*.

      - ResourceManager IP address: You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the service IP address of any ResourceManager.
      - Port: HTTPS port number of the ResourceManager. You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and view the value of `yarn.resourcemanager.webapp.https.port`.
  - Input  
None.

- **Output**

JSON Response:

```
{  
  "applicationlist": [  
    {  
      "id": "1020201_0123_12",  
      "queue": "root.Q1.Q11",  
      "user": "cchen",  
      "resource_request": {  
        "vcores" : 10,  
        "memory" : 100  
      },  
      "resource_inuse": {  
        "vcores" : 100,  
        "memory" : 2000  
      },  
      "number_running_container": 100,  
      "number_pending_request": 10  
    },  
    {  
      "id": "1020201_0123_15",  
      "queue": "root.Q2.Q21",  
      "user": "Jason",  
      "resource_request": {  
        "vcores" : 4,  
        "memory" : 100  
      },  
      "resource_inuse": {  
        "vcores" : 20,  
        "memory" : 200  
      },  
      "resource_reserved": {  
        "vcores" : 10,  
        "memory" : 100  
      },  
      "number_running_container": 20,  
      "number_pending_container": 4,  
      "number_reserved_container": 2  
    }  
  ]  
}
```

**Table 1-211** Parameters of all application

| Attribute                | Type   | Description                                                                 |
|--------------------------|--------|-----------------------------------------------------------------------------|
| applicationlist          | array  | Array of application IDs.                                                   |
| queue                    | String | Name of queue application.                                                  |
| user                     | String | Name of user who submits application.                                       |
| resource_request         | object | Currently requested resource, including vcores, memory, and so on.          |
| resource_inuse           | object | Currently resource in use, including vcores, memory, and so on.             |
| resource_reserved        | object | Currently reserved resource, including vcores, memory, and so on.           |
| number_running_container | int    | Total number of running containers. This maps to superior engine decisions. |

| Attribute                 | Type   | Description                                                                               |
|---------------------------|--------|-------------------------------------------------------------------------------------------|
| number_pending_request    | int    | Total number of pending requests, this is sum of all counts in all demands of allocation. |
| number_reserved_container | int    | Total number of reserved containers, this maps to superior engine decisions.              |
| id                        | String | Application ID.                                                                           |

- Query \*single\* application within scheduler engine.

- URL

```
GET https://<SS_REST_SERVER>/ws/v1/sscheduler/applications/
{application_id}
```

- Input

None.

- Output

JSON Response:

```
{
  "applicationlist": [
    {
      "id": "1020201_0123_12",
      "queue": "root.Q1.Q11",
      "user": "cchen",
      "resource_request": {
        "vcores": 3,
        "memory": 3072
      },
      "resource_inuse": {
        "vcores": 100,
        "memory": 2048
      },
      "number_running_container": 2,
      "number_pending_request": 3,
      "number_reserved_container": 1,
      "master_container_id": 23402_3420842,
      "master_container_resource": node1.domain.com
      "blacklist": [
        {
          "resource": "node5"
        },
        {
          "resource": "node8"
        }
      ],
      "demand": [
        {
          "priority": 1,
          "ismaster": true,
          "capability": {
            "vcores": 2,
            "memory": 2048
          },
          "count": 1,
          "relaxlocality": true,
          "locality": [
            {
              "target": "node1",
              "count": 1,
            }
          ]
        }
      ]
    }
  ]
}
```

```
        "strict": false
    }
],
"resourceselect": "label1",
"pending_reason": "application limit reached",
"reserved_resource": {
    "vcores":1,
    "memory":1024
},
"reservations":[
    "id": "23402_3420878",
    "resource": "node1.domain.com",
    "reservedAmount": {
        "vcores":1,
        "memory":1024
    }
]
},
{
    "priority": 1,
    "ismaster": false,
    "capability": {
        "vcores": 1,
        "memory": 1024
    },
    "count": 2,
    "relaxlocality": true,
    "locality": [
        {
            "target": "node1",
            "count": 1,
            "strict": false
        },
        {
            "target": "node2",
            "count": 1,
            "strict": false
        },
        {
            "target": "rackA",
            "count": 2,
            "strict": false
        }
    ],
    "resourceselect": "label1",
    "pending_reason": "no available resource"
}
],
"containers": [
    {
        "id": "23402_3420842",
        "resource": "node1.domain.com",
        "capability": {
            "vcores": 1,
            "memory": 1024
        }
    },
    {
        "id": "23402_3420853",
        "resource": "node2.domain.com",
        "capability": {
            "vcores": 1,
            "memory": 1024
        }
    }
]
```

- Exceptions  
Application not found.

**Table 1-212** Parameters of single application

| Attribute                 | Type    | Description                                                                               |
|---------------------------|---------|-------------------------------------------------------------------------------------------|
| application               | object  | Application object.                                                                       |
| id                        | String  | Application ID.                                                                           |
| queue                     | String  | Name of queue application.                                                                |
| user                      | String  | Name of user who submits application.                                                     |
| resource_request          | object  | Currently requested resource, including vcores, memory, and so on.                        |
| resource_inuse            | object  | Currently resource in use, including vcores, memory, and so on.                           |
| resource_reserved         | object  | Currently reserved resource, including vcores, memory, and so on.                         |
| number_running_container  | int     | Total number of running containers. This maps to superior engine decisions.               |
| number_pending_request    | int     | Total number of pending requests, this is sum of all counts in all demands of allocation. |
| number_reserved_container | int     | Total number of reserved containers, this maps to superior engine decisions.              |
| master_container_id       | String  | The master container ID.                                                                  |
| master_container_resource | String  | The host name that master container running on.                                           |
| demand                    | array   | Array of demand objects.                                                                  |
| priority                  | int     | Priority of demand.                                                                       |
| ismaster                  | boolean | Is the demand corresponding to "application master".                                      |
| capability                | object  | Capability object.                                                                        |
| vcores, memory, ..        | int     | Numeric consumable resource attributes, defining the allocation "unit" for this demand.   |
| count                     | int     | Number of unit required.                                                                  |
| relaxlocality             | boolean | Locality requirement is preference, and not mandatory if it cannot be satisfied.          |

| Attribute                              | Type    | Description                                                                       |
|----------------------------------------|---------|-----------------------------------------------------------------------------------|
| locality                               | object  | Locality object.                                                                  |
| target                                 | string  | Locality target's name (that is, node1, rack1..).                                 |
| count                                  | int     | Number of resource "unit" required with this locality requirement.                |
| strict                                 | boolean | If this particular locality is mandatory or not.                                  |
| resourceselect                         | String  | Resource selection expression for the demand.                                     |
| pending_reason                         | String  | Reason why this demand is outstanding.                                            |
| resource_reserved                      | object  | Currently reserved resource for this demand, including vcores, memory, and so on. |
| reservations                           | array   | Array of reserved container objects.                                              |
| reservations:id                        | String  | Reserved container ID.                                                            |
| reservations:resource                  | String  | Where the container is allocated.                                                 |
| reservations:reserveAmount             | object  | The reserved amount of this reservation.                                          |
| containers                             | array   | Array of allocated container objects.                                             |
| containers:id                          | String  | Container ID.                                                                     |
| containers:resource                    | String  | Where the container is allocated.                                                 |
| containers:capability                  | object  | Capability object.                                                                |
| containers:vcores,mem...<br>,memory... | int     | Numeric consumable resource attributes allocated to this container.               |

- Query Queue
  - Query \*all\* queues within scheduler engine, including leaf and all middle queues.
    - URL  
GET [https://<SS\\_REST\\_SERVER>/ws/v1/sscheduler/queues/list](https://<SS_REST_SERVER>/ws/v1/sscheduler/queues/list)
    - Input  
None.
    - Output  
JSON Response:

```
{  
    "queuelist": [  
        {  
            "name": "root.default",  
            "eopen_state": "OPEN",  
            "eactive_state": "ACTIVE",  
            "open_state": "OPEN",  
            "active_state": "ACTIVE",  
            "number_pending_application": 2,  
            "number_running_application": 10,  
            "number_pending_request": 2,  
            "number_running_container": 10,  
            "number_reserved_container": 1,  
            "resource_inuse" {  
                "vcores": 10,  
                "memory": 10240  
            },  
            "resource_request" {  
                "vcores": 2,  
                "memory": 2048  
            },  
            "resource_reserved" {  
                "vcores": 1  
                "memory": 1024  
            }  
        },  
        {  
            "name": "root.dev",  
            "eopen_state": "OPEN",  
            "eactive_state": "INACTIVE",  
            "open_state": "OPEN",  
            "active_state": "INACTIVE",  
            "number_pending_application": 2,  
            "number_running_application": 10,  
            "number_pending_request": 2,  
            "number_running_container": 10,  
            "number_reserved_container": 0,  
            "resource_inuse" {  
                "vcores": 10,  
                "memory": 10240  
            },  
            "resource_request" {  
                "vcores": 2,  
                "memory": 2048  
            },  
            "resource_reserved" {  
                "vcores": 0  
                "memory": 0  
            }  
        },  
        {  
            "name": "root.qa",  
            "eopen_state": "CLOSED",  
            "eactive_state": "ACTIVE",  
            "open_state": "CLOSED",  
            "active_state": "ACTIVE",  
            "number_pending_application": 2,  
            "number_running_application": 10,  
            "number_pending_request": 2,  
            "number_running_container": 10,  
            "number_reserved_container": 0,  
            "resource_inuse" {  
                "vcores": 10,  
                "memory": 10240  
            },  
            "resource_request" {  
                "vcores": 2,  
                "memory": 2048  
            }  
        }  
    ]  
}
```

```
        },
        "resource_reserved" {
            "vcores": 1
            "memory": 1024
        }
    },
]
}
```

**Table 1-213** Parameters of all queues

| Attribute                  | Type   | Description                                                                                                                                                              |
|----------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| queuelist                  | array  | Array of queue names.                                                                                                                                                    |
| name                       | String | Queue name.                                                                                                                                                              |
| open_state                 | String | This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request. |
| eopen_state                | String | This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request.       |
| active_state               | String | This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. An "INACTIVE" queue does not schedule any application.             |
| eactive_state              | String | This is outer state of queue. An effective state is combination of queue own state and its ancestors. An "INACTIVE" queue does not schedule any application.             |
| number_pending_application | int    | Total number of pending applications.                                                                                                                                    |
| number_running_application | int    | Total number of running applications.                                                                                                                                    |
| number_pending_request     | int    | Total number of pending request.                                                                                                                                         |
| number_running_container   | int    | Total number of running containers.                                                                                                                                      |
| numbert_reserved_container | int    | Total number of reserved containers.                                                                                                                                     |
| resource_request           | object | Pending resource requests within queue in a form of vcores, memory, and so on.                                                                                           |

| Attribute         | Type   | Description                                                                                                      |
|-------------------|--------|------------------------------------------------------------------------------------------------------------------|
| resource_inuse    | object | In use resource within queue in a form of vcores, memory, and so on.                                             |
| resource_reserved | object | Reserved resource within queue in form of vcores, memory, and so on.                                             |
| active_state      | String | Indicate either "ACTIVE" or "INACTIVE" state of the queue. An "INACTIVE" queue does not schedule any allocation. |

- Query \*single\* queues within scheduler engine, including leaf and all middle queues.

- URL

```
GET https://<SS_REST_SERVER>/ws/v1/sscheduler/queues/
{queueName}
```

- Input

None.

- Output

JSON Response:

```
{
  "queue": {
    "name": "root.default",
    "eopen_state": "CLOSED",
    "eactive_state": "INACTIVE",
    "open_state": "CLOSED",
    "active_state": "INACTIVE",
    "leaf_queue": true,
    "number_pending_application": 100,
    "number_running_application": 10,
    "number_pending_request": 10,
    "number_running_container": 10,
    "number_reserved_container": 1,
    "resource_inuse": {
      "vcores": 10,
      "memory": 10240
    },
    "resource_request": {
      "vcores": 2,
      "memory": 2048
    },
    "resource_reserved": {
      "vcores": 1,
      "memory": 1024
    }
  }

  "configuration": {
    "description": "Production spark queue",
    "max_pending_application": 10000,
    "max_running_application": 1000,
    "allocation_order_policy": "FIFO",
    "default_resource_select": "label1",
    "max_master_share": 10,
    "max_running_application_per_user": -1,
    "max_allocation_unit": {
      "vcores": 32,
      "memory": 128000
    },
  }
}
```

```
"user_acl": [
  {
    "user": "user1"
  },
  {
    "group": "group1"
  }
],
"admin_acl": [
  {
    "user": "user2"
  },
  {
    "group": "group2"
  }
]
```

- Exceptions  
Queue not found.

**Table 1-214** Parameters of single queues

| Attribute                  | Type    | Description                                                                                                                                                              |
|----------------------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| queue                      | object  | A queue object.                                                                                                                                                          |
| name                       | String  | Queue name.                                                                                                                                                              |
| description                | String  | Purpose of queue.                                                                                                                                                        |
| open_state                 | String  | This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request. |
| eopen_state                | String  | This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request.       |
| active_state               | String  | This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. An "INACTIVE" queue does not schedule any application.             |
| eactive_state              | String  | This is outer state of queue. An effective state is combination of queue own state and its ancestors. An "INACTIVE" queue does not schedule any application.             |
| leaf_queue                 | boolean | Indicate whether queue is leaf or middle. Yes means leaf queue.                                                                                                          |
| number_pending_application | int     | Number of pending application currently. In case of middle or parent queue, this is the aggregated number of all children queue.                                         |

| Attribute                        | Type   | Description                                                                                                                                                |
|----------------------------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| number_running_application       | int    | Number of running application currently. In case of middle or parent queue, this is the aggregated number of all children queue.                           |
| number_pending_request           | int    | Number of pending demand; sum of count from each outstanding demand. In case of middle/parent queue, this is the aggregated number of all children queues. |
| number_running_container         | int    | Number of running containers. In case of middle or parent queue, this is the aggregated number of all children queues.                                     |
| number_reserved_container        | int    | Number of reserved containers. In case of middle or parent queue, this is the aggregated number of children queues.                                        |
| resource_request                 | object | Pending resource requests within queue in a form of vcores, memory etc.                                                                                    |
| resource_inuse                   | object | In use resource within queue in a form of vcores and memory etc.                                                                                           |
| resource_reserved                | object | Reserved resources within queue in a form of vcores and memory etc.                                                                                        |
| configuration                    | object | A queue configuration object.                                                                                                                              |
| max_pending_application          | int    | Max number of pending application. In case of middle or parent queue, this is the aggregated number of all children queue.                                 |
| max_running_application          | int    | Max number of running application. In case of middle/parent queue, this is the aggregated number of all children queue.                                    |
| allocation_order_policy          | String | Allocation policy, can be FIFO, PRIORITY, or FAIR.                                                                                                         |
| max_running_application_per_user | int    | Maximum number of running application per user.                                                                                                            |
| max_master_share                 | String | Percentage of steady share of this queue.                                                                                                                  |
| max_allocation_unit              | object | Maximum allowed resource per container in vcores and memory format.                                                                                        |
| default_resource_select          | String | Default resource selection expression. It is used when an application does not specify one during its submission.                                          |
| user_acl                         | array  | Array of user, who have been given "user" rights on this queue.                                                                                            |

| Attribute | Type   | Description                                                      |
|-----------|--------|------------------------------------------------------------------|
| admin_acl | array  | Array of user, who have been given "admin" rights on this queue. |
| group     | String | User group name.                                                 |
| user      | String | User name.                                                       |

- Query Resource Pool
  - Query \*all\* resource pools within scheduler engine.
    - URL  
GET `https://<SS_REST_SERVER>/ws/v1/sscheduler/resourcepools/list`
    - Input  
None.
    - Output

```
JSON Response:
{
  "resourcepool_list": [
    {
      "name": "pool1",
      "description": "resource pool for crc",
      "number_member": 5,
      "members": [
        {
          "resource": "node1"
        },
        {
          "resource": "node2"
        },
        {
          "resource": "node3"
        },
        {
          "resource": "node4"
        },
        {
          "resource": "node5"
        }
      ],
      "available_resource": {
        "vcores": 60,
        "memory": 60000
      },
      "total_resource": {
        "vcores": 100,
        "memory": 128000
      },
      "configuration": {
        "resources": [
          {
            "resource": "node1"
          },
          {
            "resource": "node[2-5]"
          }
        ],
        "resource_select": "label1"
      }
    }
  ]
}
```

```
"name": "pool2",
"description": "resource pool for erc",
"number_member": 4
"members": [
{
"resource": "node6"
},
{
"resource": "node7"
},
{
"resource": "node8"
},
{
"resource": "node9"
}
],
"available_resource": {
"vcores": 56,
"memory": 48000
},
"total_resource": {
"vcores": 100,
"memory": 128000
},
"configuration": {
"resources": [
{
"resource": "node6"
},
{
"resource": "node[7-9]"
}
],
"resource_select": "label1"
},
{
"name": "default",
"description": "system-generated",
"number_member": 1,
"members": [
{
"resource": "node0"
}
],
"available_resource": {
"vcores": 8,
"memory": 8192
},
"total_resource": {
"vcores": 16,
"memory": 12800
}
}
]
```

**Table 1-215** Parameters of all resource pool

| Attribute         | Type   | Description                     |
|-------------------|--------|---------------------------------|
| resourcepool_list | array  | Array of resource pool objects. |
| name              | String | Resource pool name.             |

| Attribute          | Type   | Description                                                                   |
|--------------------|--------|-------------------------------------------------------------------------------|
| number_member      | int    | Number of members in resource pool.                                           |
| description        | string | Description of the resource pool.                                             |
| members            | array  | Array of resource name, which is current members of the resource pool.        |
| resource           | String | Resource name.                                                                |
| available_resource | object | Current available resource in this pool.                                      |
| vcores, memory     | int    | Numeric consumable resource attributes, available via this resource pool now. |
| total_resource     | object | total resource in this pool.                                                  |
| vcores, memory     | int    | Numeric consumable resource attributes, total via this resource pool now.     |
| configuration      | object | Configuration object.                                                         |
| resources          | array  | Array of resource name pattern configured.                                    |
| resource           | String | Resource name pattern.                                                        |
| resource_select    | String | Resource selection expression.                                                |

- Query \*single\* resource pools within scheduler engine.
  - URL  
GET `https://<SS_REST_SERVER>/ws/v1/sscheduler/resourcepools/{resourcepoolname}`
  - Input  
None.
  - Output

JSON Response:

```
{
  "resourcepool": {
    "name": "pool1",
    "description": "resource pool for crc",
    "number_member": 5
    "members": [
      {
        "resource": "node1"
      },
      {
        "resource": "node2"
      },
      {
        "resource": "node3"
      },
      {
        "resource": "node4"
      }
    ]
  }
}
```

```
        "resource": "node5"
    },
],
"available_resource": {
    "vcores": 60,
    "memory": 60000
},
"total_resource": {
    "vcores": 100,
    "memory": 128000
},
"configuration": {
    "resources": [
        {
            "resource": "node6"
        },
        {
            "resource": "node[7-9]"
        }
    ],
    "resource_select": "label1"
}
}
```

- Exceptions

Resource pool not found.

**Table 1-216** Parameters of single resource pool

| Attribute          | Type   | Description                                                                   |
|--------------------|--------|-------------------------------------------------------------------------------|
| resourcepool       | object | Resource pool object.                                                         |
| name               | String | Resource pool name.                                                           |
| description        | String | Description of the resource pool.                                             |
| number_member      | int    | Number of members in resource pool.                                           |
| members            | array  | Array of resource name, which is current members of the resource pool.        |
| resource           | String | Resource name.                                                                |
| available_resource | object | Current available resource in this pool.                                      |
| vcores, memory     | int    | Numeric consumable resource attributes, available via this resource pool now. |
| total_resource     | object | total resource in this pool.                                                  |
| vcores, memory     | int    | Numeric consumable resource attributes, total via this resource pool now.     |
| configuration      | object | Configuration object.                                                         |
| resources          | array  | Array of resource name pattern configured.                                    |
| resource           | String | Resource name pattern.                                                        |

| Attribute       | Type   | Description                    |
|-----------------|--------|--------------------------------|
| resource_select | String | Resource selection expression. |

- Query **policiesxmlconf**

- URL  
GET https://<SS\_REST\_SERVER>/ws/v1/sscheduler/policiesxmlconf/list
- Input  
None.
- Output

```
<policies>
  <policlist>
    <resourcepool>default</resourcepool>
    <queues>
      <name>default</name>
      <fullname>root.default</fullname>
      <share>20.0</share>
      <reserve>memory 0,vcores 0 : 0.0%</reserve>
      <minimum>memory 0,vcores 0 : 20.0%</minimum>
      <maximum>memory 0,vcores 0 : 100.0%</maximum>
      <defaultuser>
        <maximum>100.0%</maximum>
        <weight>1.0</weight>
      </defaultuser>
    </queues>
  </policlist>
</policies>
```

# 2 Normal Mode

## 2.1 ClickHouse Development Guide

### 2.1.1 Overview

#### 2.1.1.1 Introduction to ClickHouse

##### ClickHouse

ClickHouse is a column-based database oriented to online analysis and processing. It supports SQL query and provides good query performance. The aggregation analysis and query performance based on large and wide tables is excellent, which is one order of magnitude faster than other analytical databases.

Advantages for ClickHouse:

- High data compression ratio
- Multi-core parallel computing
- Vectorized computing engine
- Supporting nested data structure
- Supporting sparse indexes
- Supporting INSERT and UPDATE

##### ClickHouse application scenarios:

- Real-time data warehouse

The streaming computing engine (such as Flink) is used to write real-time data to ClickHouse. With the excellent query performance of ClickHouse, Multi-dimensional and multi-mode real-time query and analysis requests can be responded within subseconds.

- Offline query

Large-scale service data is imported to ClickHouse and constructs a large wide table with hundreds of millions to tens of billions of records and hundreds of dimensions. It supports personalized statistics collection and continuous

exploratory query and analysis at any time to assist business decision-making and provides excellent query experience.

## Introduction to the ClickHouse Development Interface

ClickHouse is developed using C++ and positioned as a DBMS. It supports HTTP and Native TCP network interface protocols and multiple driver modes such as JDBC and ODBC. You are advised to use [clickhouse-jdbc](#) of the community version for application development.

### 2.1.1.2 Basic Concepts

#### Concepts

- **cluster**

Cluster: Cluster is a logical concept in ClickHouse. It can be defined by users as required, which is different from the general understanding of cluster. Multiple ClickHouse nodes are loosely coupled and exist independently.

- **shards**

Shard: A shard is a horizontal division of a cluster. A cluster can consist of multiple shards.

- **replicas**

Replica: One shard can contain multiple replicas.

- **partition**

Partition: A partition is used for local replicas and can be considered as a vertical division.

- **MergeTree**

ClickHouse has a huge table engine system. As the basic table engine of the family system, MergeTree provides functions such as data partitioning, primary indexes, and secondary indexes. When creating a table, you need to specify the table engine. Different table engines determine the final character of a data table, for example, the features of a data table, the form how data is stored, and how data is loaded.

### 2.1.1.3 Development Process

[Figure 2-1](#) and [Table 2-1](#) describe the phases in the development process.

Figure 2-1 ClickHouse application development process

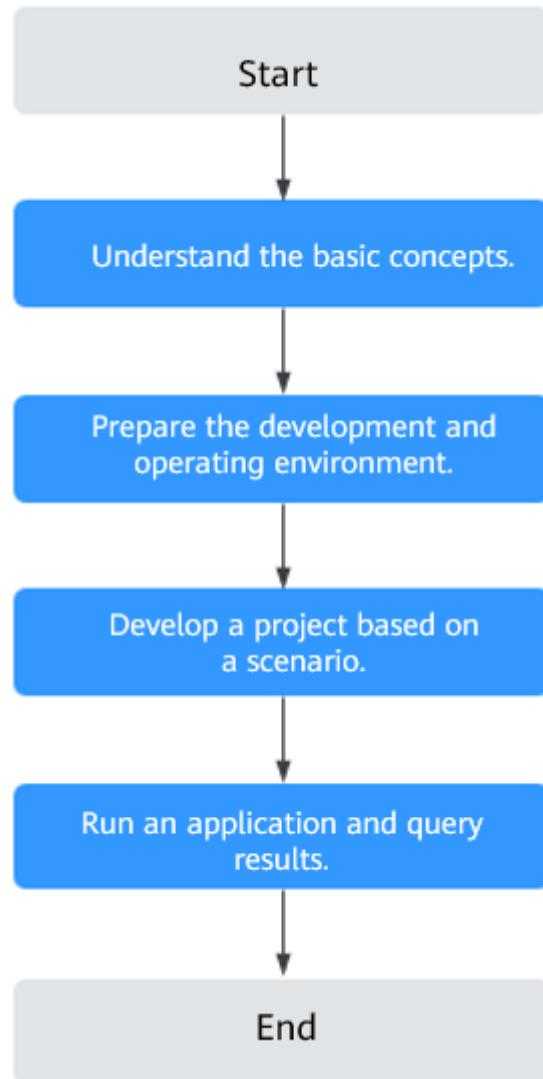


Table 2-1 Description of the ClickHouse application development process

Phase	Description	Reference
Understand the basic concepts.	Before application development, you need to understand basic concepts of ClickHouse.	<a href="#">Basic Concepts</a>
Prepare the development and operating environment.	ClickHouse applications can be developed in multiple languages, mainly Java. The IntelliJ IDEA tool is recommended. Configure the development environment based on the guide.	<a href="#">Preparing the Development and Operating Environment</a>

Phase	Description	Reference
Develop a project based on a scenario.	Sample projects are provided to help you quickly understand APIs of ClickHouse components.	<a href="#">Configuring and Importing a Sample Project</a>
Run application and query results.	You can check the application running status from the running results.	<a href="#">Commissioning Applications on Windows</a> <a href="#">Commissioning Applications on Linux</a>

## 2.1.2 Environment Preparations

### 2.1.2.1 Preparing the Development and Operating Environment

#### Preparing the Development Environment

[Table 2-2](#) describes the environment required for application development.

**Table 2-2** Development environment

Item	Description
Operating System (OS)	<ul style="list-style-type: none"><li>• Development environment: Windows 7 or later.</li><li>• Operating environment: Linux</li></ul> If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.

Item	Description
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
Installing and configuring IntelliJ IDEA	<p>Basic configuration of the development environment. You are advised to use version 2019.1 or other compatible versions.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li><li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li><li>• If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>

Item	Description
Apache Maven installation	Basic configuration of the development environment. This operation is used for project management throughout the lifecycle of software development.
Developer account preparation	Prepare a cluster user for application development and grant permissions to the user by referring to <a href="#">Preparing a Developer Account</a> .
7-zip	This tool is used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

## Preparing the Operating Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning application; configure the network connection, and commission the application in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#). In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.
  - b. Copy all items from the hosts file in the decompression directory to the hosts file on the node where the client is installed. Ensure that the network communication between the local PC and hosts listed in the hosts file in the decompression directory is normal.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client in a directory, for example, **/opt/hadoopclient**, on the node.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides. Ensure that the local host can communicate with each host in the cluster on the network.

### 2.1.2.2 Configuring and Importing a Sample Project

#### Context

Obtain the ClickHouse development sample project and import the project to IntelliJ IDEA to learn the sample project.

#### Prerequisites

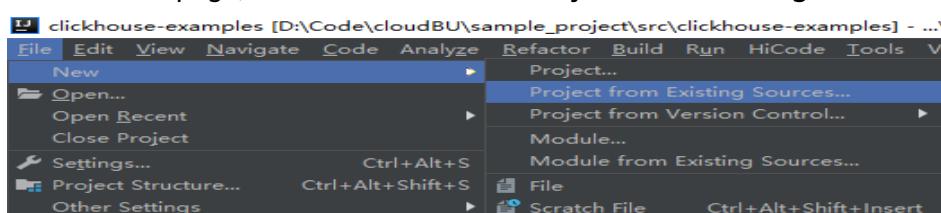
Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the cluster in the lower-right corner on the FusionInsight Manager page.

#### Scenarios

ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

#### Procedure

- Step 1** Obtain the sample project folder **clickhouse-examples** and Maven configurations in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
  1. On the IDEA page, choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-examples/ClickHouseJDBCJavaExample** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

**Step 3** After the project is imported, modify the **clickhouse-example.properties** file in the **conf** directory of the sample project based on the actual environment information.

```
loadBalancerIPList=
sslUsed=false
loadBalancerHttpPort=21425
loadBalancerHttpsPort=21426
CLICKHOUSE_SECURITY_ENABLED=true
user=
# Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or
environment variables.
password=
isMachineUser=false
isSupportMachineUser=false
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
clickhouse_dataSource_ip_list=
native_dataSource_ip_list=
```

**Table 2-3** Configuration description

Parameter	Default Value	Definition
loadBalancerIPList	-	<p>Mandatory. Set this parameter to the IP address list of LoadBalance.</p> <ul style="list-style-type: none"><li>• Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instance</b> and check the service IP addresses of all ClickHouseBalancer instances.</li><li>• Use commas (,) to separate multiple IP addresses, for example, <b>10.10.10.100,10.10.10.101</b>.</li><li>• If the IP address is an IPv6 address, convert it, for example, from <b>192:168:0:0:0:158:2</b> to <b>192:168::158:2</b>.</li></ul>

Parameter	Default Value	Definition
sslUsed	true	<p>Whether to enable SSL encryption. You are advised to set this parameter to <b>false</b> for clusters in common mode.</p> <p><b>Note:</b> If this parameter is set to <b>true</b>, SSL connections are used. Common clusters use non-SSL connections by default. If SSL connections are required, log in to FusionInsight Manager, choose <b>Cluster &gt; Services &gt; ClickHouse</b>, click the <b>Configuration</b> tab then the <b>All Configurations</b> sub-tab. On the page that is displayed, search for the <b>SSL_NONESSL_BOTH_ENABLE</b> configuration item, change its value to <b>true</b>, and restart all ClickHouse service instances.</p>
loadBalancerHttpPort	21425	<p>HTTP port number of LoadBalance. If <b>sslUsed</b> is set to <b>false</b>, this parameter cannot be left blank.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</p>
loadBalancerHttpsPort	21426	<p>HTTPS port number of LoadBalance. If <b>sslUsed</b> is set to <b>true</b>, this parameter cannot be empty.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</p>
CLICKHOUSE_SECURITY_ENABLED	true	<p>Whether to enable the security mode. Set this parameter to <b>false</b> for a cluster in normal mode.</p>
user	No default value	Developer user created in <a href="#">Table 1-3</a> .
password	No default value	<p>Password of a human-machine user. This parameter does not need to be set if a machine-machine user is used.</p> <p>Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.</p> <p><b>NOTE</b> If the human-machine user is created on FusionInsight Manager, you need to change the initial password upon the first login.</p>

Parameter	Default Value	Definition
isMachineUser	false	<p>Set this parameter to <b>true</b> if a machine-machine user is used for authentication.</p> <p>If this parameter is set to <b>true</b>, the values of <b>user</b> and <b>password</b> are the username and password of the machine-machine user.</p>
isSupportMachineUser	true	<p>Whether to support machine-machine user authentication. Value options are as follows:</p> <p><b>true</b>: Machine-machine user authentication is supported.</p> <p><b>false</b>: Machine-machine user authentication is not supported.</p> <p>To set this parameter, perform the following operations:</p> <p>Log in to FusionInsight Manager, choose <b>Cluster &gt; Services &gt; ClickHouse</b>, click the <b>Configuration</b> tab then the <b>All Configurations</b> sub-tab. In the navigation pane on the left, choose <b>ClickHouseServer(Role) &gt; Security</b>. On the page that is displayed, check the value of <b>SUPPORT_MACHINE_USER</b>.</p>
clusterName	default_c_luster	ClickHouse logical cluster name. Use the actual one.
databaseName	testdb	Name of the database to be created in the sample code project. You can change the database name based on the site requirements.
tableName	testtb	Name of the table to be created in the sample code project. You can change the table name based on the site requirements.
batchRows	10000	Number of data records written in a batch.
batchNum	10	Total number of batches in which data is written.

Parameter	Default Value	Definition
clickhouse_dataSource_ip_list	-	<p>Mandatory. Set this parameter to the IP address list and HTTP port number of the ClickHouseBalancer instance.</p> <ul style="list-style-type: none"><li>• Use commas (,) to separate multiple IP addresses. The format is as follows: <i>ip:port,ip:port,ip:port</i>.</li><li>• IP address: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instance</b> and check the service IP addresses of all ClickHouseBalancer instances.</li><li>• Port: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</li><li>• If the IP address is an IPv6 address, convert it to the IP address + Port number format. 192:168:0:0:0:158:2:21425 &gt; [192:168::158:2]:21425</li></ul>
native_dataSource_ip_list	-	<p>Mandatory. Set this parameter to the IP address list and TCP port number of the ClickHouseBalancer instance.</p> <ul style="list-style-type: none"><li>• Use commas (,) to separate multiple IP addresses. The format is as follows: <i>ip:port,ip:port,ip:port</i>.</li><li>• IP address: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instances</b> and check the service IP addresses of all ClickHouseBalancer instances.</li><li>• Port: Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>TCP Balancer Port</b> column.</li><li>• If the IP address is an IPv6 address, convert it to the IP address + Port number format. 192:168:0:0:0:158:2:21424 &gt; [192:168::158:2]:21424</li></ul>

#### NOTE

ClickHouse uses the LoadBalance-based deployment architecture to automatically distribute user access traffic to multiple backend nodes, expanding service capabilities to external systems and improving fault tolerance. When a client application requests a cluster, Nginx-based ClickHouseBalancer is used to distribute traffic. In this way, data read/write load and high availability of application access are guaranteed.

----End

### 2.1.2.3 Configuring and Importing a Transaction Sample Project

#### Context

Obtain the ClickHouse development sample project and import the project to IntelliJ IDEA to learn the sample project.

#### Prerequisites

Ensure that the difference between the local PC time and the cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the cluster in the lower-right corner on the FusionInsight Manager page.

#### Scenarios

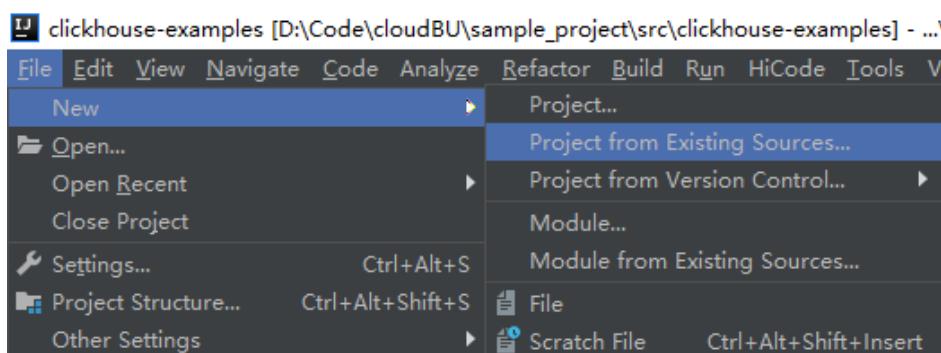
ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

#### Procedure

**Step 1** Obtain the sample project folder **clickhouse-examples** and Maven configurations in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. On the IDEA page, choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-examples/ClickHouseJDBC-Transaction-JavaExample** folder and click **OK**.

3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

**Step 3** After the project is imported, modify the **clickhouse-example.properties** file in the **conf** directory of the sample project based on the actual environment information.

```
loadBalancerIPList=
sslUsed=false
loadBalancerHttpPort=21425
loadBalancerHttpsPort=21426
CLICKHOUSE_SECURITY_ENABLED=true
user=
# Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or
environment variables.
password=
#Whether to use transactions.
useTransaction=true
#Whether to automatically submit SQL statements. The options are true (automatic) and false (manual).
Automatic submission is recommended. Due to the session forwarding mechanism of Balance, SQL
statements cannot be manually submitted by connecting to Balance.
autoCommit=true
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
```

**Table 2-4** Parameter description

Parameter	Default Value	Description
loadBalancerIPList	-	<p>Mandatory. Set this parameter to the IP address list of LoadBalance.</p> <ul style="list-style-type: none"><li>• Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Instance</b> and check the service IP addresses of all ClickHouseBalancer instances.</li><li>• Use commas (,) to separate multiple IP addresses, for example, <b>10.10.10.100,10.10.10.101</b>.</li><li>• If the IP address is an IPv6 address, convert it, for example, from <b>192:168:0:0:0:158:2</b> to <b>192:168::158:2</b>.</li></ul>

Parameter	Default Value	Description
sslUsed	true	<p>Whether to enable SSL encryption. You are advised to set this parameter to <b>false</b> for clusters in common mode.</p> <p><b>Note:</b> If this parameter is set to <b>true</b>, SSL connections are used. Common clusters use non-SSL connections by default. If SSL connections are required, log in to FusionInsight Manager, choose <b>Cluster &gt; Services &gt; ClickHouse</b>, click the <b>Configuration</b> tab then the <b>All Configurations</b> sub-tab. On the page that is displayed, search for the <b>SSL_NONESSL_BOTH_ENABLE</b> configuration item, change its value to <b>true</b>, and restart all ClickHouse service instances.</p>
loadBalancerHttpPort	21425	<p>HTTP port number of LoadBalance. If <b>sslUsed</b> is set to <b>false</b>, this parameter cannot be left blank.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</p>
loadBalancerHttpsPort	21426	<p>HTTPS port number of LoadBalance. If <b>sslUsed</b> is set to <b>true</b>, this parameter cannot be empty.</p> <p>Log in to FusionInsight Manager and choose <b>Cluster &gt; Services &gt; ClickHouse</b>. Click <b>Logical Cluster</b>, locate the row containing the desired logical cluster, and view <b>Ssl Port</b> in the <b>HTTP Balancer Port</b> column.</p>
CLICKHOUSE_SECURITY_ENABLED	true	<p>Whether to enable the security mode. Set this parameter to <b>false</b> for a cluster in normal mode.</p>
user	No default value	Developer user created in <a href="#">Table 1-3</a> .
password	No default value	<p>Password of a human-machine user. This parameter does not need to be set if a machine-machine user is used.</p> <p>Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.</p> <p><b>NOTE</b> If the human-machine user is created on FusionInsight Manager, you need to change the initial password upon the first login.</p>

Parameter	Default Value	Description
useTransaction	true	Whether to use transactions. Set this parameter to <b>true</b> . If this parameter is set to <b>true</b> , SQL statements use transactions.
autoCommit	true	Whether to submit data automatically. The available values are: <b>true</b> : The transactions are automatically submitted, which is equivalent to using implicit transactions. <b>false</b> : The transactions are manually submitted.
clusterName	default_c_luster	ClickHouse logical cluster name. Use the actual one.
databaseName	testdb	Name of the database to be created in the sample code project. You can change the database name based on the site requirements.
tableName	testtb	Name of the table to be created in the sample code project. You can change the table name based on the site requirements.
batchRows	10000	Number of data records written in a batch.
batchNum	10	Total number of batches in which data is written.

#### NOTE

ClickHouse uses the LoadBalance-based deployment architecture to automatically distribute user access traffic to multiple backend nodes, expanding service capabilities to external systems and improving fault tolerance. When a client application requests a cluster, Nginx-based ClickHouseBalancer is used to distribute traffic. In this way, data read/write load and high availability of application access are guaranteed.

----End

### 2.1.2.4 Configuring and Importing the Spring Boot Sample Project

#### Scenarios

To run the Spring Boot interface sample code of the ClickHouse component of MRS, perform the following operations.

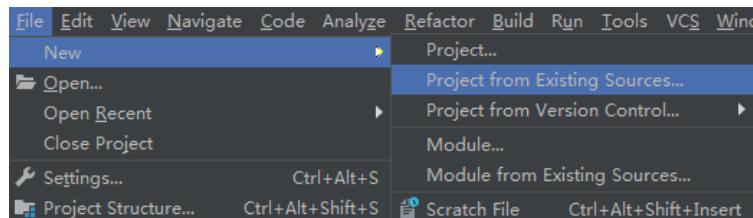
In the following example, an application is developed by compiling a Spring Boot project to connect to ClickHouse in Windows.

## Procedure

**Step 1** Obtain the sample project folder **clickhouse-rest-client-example** in the **src/springboot/clickhouse-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Choose **File > New > Project from Existing Sources....**



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-rest-client-example** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.

Select the recommended JDK version and click **Finish**.

**Step 3** In the IntelliJ IDEA development environment, open the **clickhouse-example.properties** file in the **conf** directory of the sample project, and change the values of the parameters listed in [Table 1-7](#).

**Step 4** (Optional) If machine-machine user authentication is used, download the **user.keytab** and **krb5.conf** authentication credential files of the machine-machine user by referring to [Step 7](#), and upload the files to the **conf** directory of the sample project.

----End

## 2.1.3 Application Development

### 2.1.3.1 Typical Application Scenario

This section describes the application development in a typical scenario, helping you quickly learn and master the ClickHouse development process and know key functions.

#### Scenario

ClickHouse enables you to perform common service operations by executing SQL statements. The SQL operations involved in sample codes include creating a database, creating a table, inserting data into a table, querying table data, and deleting a table.

Sample codes are described in the following sequence:

1. Setting properties
2. Establishing a connection

3. Creating a database
4. Creating a table
5. Inserting data into a table
6. Querying data
7. Deleting a table

### 2.1.3.2 Development Guideline

#### Development Guideline

As an independent DBMS system, ClickHouse allows you to use the SQL language to perform common operations. In the development program example, the clickhouse-jdbc API is used for description. The development process consists of the following parts:

- Setting properties: Sets the parameters for connecting to a ClickHouse service instance.
- Establishing a connection: Establishes a connection to the ClickHouse service instance.
- Creating a database: Creates a ClickHouse database.
- Creating a table: Creates a table in the ClickHouse database.
- Inserting data: Inserts data into a ClickHouse table.
- Querying data: Queries data in ClickHouse tables.
- Deleting a table: Deletes a ClickHouse table.

### 2.1.4 Sample Code

#### 2.1.4.1 Setting Properties

Set connection properties in the examples of creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files. The following code sets the socket timeout interval to 60 seconds:

```
Properties clickHouseProperties = new Properties();
clickHouseProperties.setProperty(ClickHouseClientOption.CONNECTION_TIMEOUT.getKey(),
Integer.toString(60000));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL.getKey(), Boolean.toString(true));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL_MODE.getKey(), "none");
```

If **sslUsed** in the **clickhouse-example.properties** configuration file in [Configuring and Importing a Sample Project](#) is set to **true**, set the following connection properties in the examples for creating connections in the **ClickhouseJDBCHaDemo**, **Demo**, **NativeJDBCHaDemo**, and **Util** files:

```
clickHouseProperties.setSsl(true);
clickHouseProperties.setSslMode("none");
```

#### 2.1.4.2 Establishing a Connection

The following code snippets are provided in the **initConnection** method of the **ClickhouseJDBCHaDemo** class. During connection creation, the user and

password configured in [Table 2-3](#) are used as authentication credentials. ClickHouse performs security authentication on the server with the user and password.

```
clickHouseProperties.setProperty(ClickHouseDefaults.USER.getKey(), userName);
clickHouseProperties.setProperty(ClickHouseDefaults.PASSWORD.getKey(), userPass);
try {
    clickHouseProperties.setProperty(ClickHouseClientOption.FAILOVER.getKey(), "21");
    clickHouseProperties.setProperty(ClickHouseClientOption.LOAD_BALANCING_POLICY.getKey(),
"roundRobin");
    balancedClickhouseDataSource = new ClickHouseDataSource(JDBC_PREFIX + UriList,
clickHouseProperties);
} catch (Exception e) {
    LOG.error("Failed to create balancedClickHouseProperties.");
    throw e;
}
```

### 2.1.4.3 Creating a Database

Run the **on cluster** statement to create a database whose name is the value of **databaseName** in [Table 2-3](#) in the cluster.

```
private void createDatabase(String databaseName, String clusterName) throws Exception {
    String createDbSql = "create database if not exists " + databaseName + " on cluster " + clusterName;
    util.exeSql(createDbSql);
}
```

### 2.1.4.4 Creating a Table

Run the **on cluster** statement to create the ReplicatedMerge and Distributed tables whose names are the same as the values of **tableName** in [Table 2-3](#).



If multiple ClickHouse services are installed in the cluster, for example, **clickhouse-1**, the ZooKeeper path is **/clickhouse-1**.

```
private void createTable(String databaseName, String tableName, String clusterName) throws Exception {
    String createSql = "create table " + databaseName + "." + tableName + " on cluster " + clusterName +
" (name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}'/" +
databaseName + "." + tableName + "," + "{replica}) partition by toYYYYMM(date) order by age";
    String createDisSql = "create table " + databaseName + "." + tableName + "_all" + " on cluster " +
clusterName + " as " + databaseName + "." + tableName + " ENGINE = Distributed(default_cluster," +
databaseName + "," + tableName + ", rand());"; ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(createSql);
    sqlList.add(createDisSql);
    util.exeSql(sqlList);
}
```

### 2.1.4.5 Inserting Data

The table created in [Creating a Table](#) has three fields of the String, UInt8, and Date types.

```
String insertSql = "insert into " + databaseName + "." + tableName + " values (?, ?, ?)";
PreparedStatement preparedStatement = connection.prepareStatement(insertSql);
long allBatchBegin = System.currentTimeMillis();
for (int j = 0; j < batchNum; j++) {
    for (int i = 0; i < batchRows; i++) {
        preparedStatement.setString(1, "huawei_" + (i + j * 10));
        preparedStatement.setInt(2, ((int) (Math.random() * 100)));
        preparedStatement.setDate(3, generateRandomDate("2018-01-01", "2021-12-31"));
        preparedStatement.addBatch();
    }
    long begin = System.currentTimeMillis();
```

```
        preparedStatement.executeBatch();
        long end = System.currentTimeMillis();
        log.info("Inert batch time is {} ms", end - begin);
    }
    long allBatchEnd = System.currentTimeMillis();
    log.info("Inert all batch time is {} ms", allBatchEnd - allBatchBegin);
```

## 2.1.4.6 Querying Data

Query statement 1 **querySql1** queries any 10 records in the **tableName** table created in [Creating a Table](#). Query statement 2 **querySql2** uses a built-in function to combine the year and month fields in the **tableName** table created in [Creating a Table](#).

```
private void queryData(String databaseName, String tableName) throws Exception {
    String querySql1 = "select * from " + databaseName + "." + tableName + "_all" + " order by age limit 10";
    String querySql2 = "select toYYYYMM(date),count(1) from " + databaseName + "." + tableName + "_all"
+ " group by toYYYYMM(date) order by count(1) DESC limit 10";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(querySql1);
    sqlList.add(querySql2);
    ArrayList<ArrayList<ArrayList<String>>> result = util.exeSql(sqlList);
    for (ArrayList<ArrayList<String>> singleResult : result) {
        for (ArrayList<String> strings : singleResult) {
            StringBuilder stringBuilder = new StringBuilder();
            for (String string : strings) {
                stringBuilder.append(string).append("\t");
            }
            log.info(stringBuilder.toString());
        }
    }
}
```

## 2.1.4.7 Deleting a Table

Delete the replica table and distributed table created in [Creating a Table](#).

```
private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " +
clusterName;
    String dropDisTableSql = "drop table if exists " + databaseName + "." + tableName + "_all" + " on cluster "
+ clusterName;
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(dropLocalTableSql);
    sqlList.add(dropDisTableSql);
    util.exeSql(sqlList);
}
```

## 2.1.4.8 Using BalanceDataSource

BalanceDataSource provides high-availability access connection APIs for applications. When detecting that a ClickHouseBalancer service is abnormal, applications can access the service by using other normal service instances. The following sample code provides examples of ClickHouse JDBC and Native JDBC using BalanceDataSource.

### NOTE

- ClickHouse JDBC uses HTTP to access ClickHouse, and Native JDBC uses TCP to access ClickHouse. For details about the access configurations, see the configuration description of `clickhouse.dataSource_ip_list` and `native.dataSource_ip_list` in [Configuring and Importing a Sample Project](#).
- Native JDBC does not support SSL-encrypted access.

## ClickHouse JDBC Example

### Initial configuration and connection.

```
public void initConnection() throws Exception {
    Properties properties = new Properties();
    String proPath = System.getProperty("user.dir") + File.separator + "conf"
        + File.separator + "clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(proPath));
    } catch (IOException e) {
        LOG.error("Failed to load properties file.");
        throw e;
    }
    Properties clickHouseProperties = new Properties();
    boolean isSec = Boolean.parseBoolean(properties.getProperty("CLICKHOUSE_SECURITY_ENABLED"));
    String UriList = properties.getProperty("clickhouse.dataSource_ip_list");
    String userName = properties.getProperty("user");
    boolean isMachineUser = Boolean.parseBoolean(properties.getProperty("isMachineUser"));
    if (isSec) {
        if (isMachineUser) {
            clickHouseProperties.setProperty("isMachineUser", "true");
            clickHouseProperties.setProperty("keytabPath", System.getProperty("user.dir") + File.separator +
                "conf" + File.separator + "user.keytab");
        }
        String userPass = properties.getProperty("password");
        clickHouseProperties.setProperty(ClickHouseDefaults.PASSWORD.getKey(), userPass);
        clickHouseProperties.setProperty(ClickHouseClientOption.SSL.getKey(), Boolean.toString(true));
        clickHouseProperties.setProperty(ClickHouseClientOption.SSL_MODE.getKey(), "none");
    }
    clickHouseProperties.setProperty(ClickHouseDefaults.USER.getKey(), userName);
    try {
        clickHouseProperties.setProperty(ClickHouseClientOption.FAILOVER.getKey(), "21");
        clickHouseProperties.setProperty(ClickHouseClientOption.LOAD_BALANCING_POLICY.getKey(),
            "roundRobin");
        balancedClickhouseDataSource = new ClickHouseDataSource(JDBC_PREFIX + UriList,
        clickHouseProperties);
    } catch (Exception e) {
        LOG.error("Failed to create balancedClickHouseProperties.");
        throw e;
    }
}
```

### Obtain the connection and send an SQL request.

```
public void queryData(String databaseName, String tableName) {
    String querySql1 = "select name,age from " + databaseName + "." + tableName + "_all" + " order by age
limit 10";
    try (ClickHouseConnection connection = balancedClickhouseDataSource.getConnection();
        ClickHouseStatement statement = connection.createStatement();
        ResultSet set = statement.executeQuery(querySql1)){
        while (set.next()) {
            LOG.info("Name is: " + set.getString(1) + ", age is: " + set.getString(2));
        }
    } catch (SQLException e) {
        //If the return code is 210, the LB server may be disconnected. Check whether the connection is
        available and try again.
        if (e.getErrorCode() == ClickHouseErrorCode.NETWORK_ERROR.code) {
            balancedClickhouseDataSource.actualize();
            try (ClickHouseConnection con = balancedClickhouseDataSource.getConnection());

```

```
        ClickHouseStatement st = con.createStatement();
        ResultSet rs = st.executeQuery(querySql1)) {
        while (rs.next()) {
            LOG.info("Name is: " + rs.getString(1) + ", age is: " + rs.getString(2));
        }
    } catch (SQLException exception) {
        LOG.error("Query data failed.");
    }
} else {
    LOG.error("Query data failed.", e);
}
}
```

## Native JDBC Example

Initial configuration and connection.

```
public void initConnection() throws IOException {
    Properties properties = new Properties();
    String proPath = System.getProperty("user.dir") + File.separator + "conf"
        + File.separator + "clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(proPath));
    } catch (IOException e) {
        LOG.error("Failed to load properties file.");
        throw e;
    }
    String UriList = properties.getProperty("native.dataSource.ip.List");
    //Whether to check connection availability during connection initialization.
    properties.put("is_check_connection", "true");
    balancedClickhouseDataSource = new BalancedClickhouseDataSource(JDBC_PREFIX + UriList, properties)
        //This method periodically checks whether the connection is available in the background and
        maintains a list of available connections. The following configuration indicates that the check is performed
        every 30 seconds.
        .scheduleActualization(30, TimeUnit.SECONDS);
}
```

Obtain the connection and send an SQL request.

```
public void queryData(String databaseName, String tableName) {
    String querySql1 = "select name,age from " + databaseName + "." + tableName + "_all" + " order by age
limit 10";
    try (ClickHouseConnection connection = balancedClickhouseDataSource.getConnection();
        ClickHouseStatement statement = connection.createStatement();
        ResultSet set = statement.executeQuery(querySql1)){
        while (set.next()) {
            LOG.info("Name is: " + set.getString(1) + ", age is: " + set.getString(2));
        }
    } catch (SQLException e) {
        //If the return code is 210, the LB server may be disconnected. Check whether the connection is
        available and try again.
        if (e.getErrorCode() == ClickHouseErrCode.NETWORK_ERROR.code()) {
            balancedClickhouseDataSource.actualize();
            try (Connection con = balancedClickhouseDataSource.getConnection();
                Statement statement = con.createStatement();
                ResultSet rs = st.executeQuery(querySql1)) {
                while (rs.next()) {
                    LOG.info("Name is: " + rs.getString(1) + ", age is: " + rs.getString(2));
                }
            } catch (SQLException exception) {
                LOG.error("Query data failed.");
            }
        } else {
            LOG.error("Query data failed.", e);
        }
    }
}
```

## 2.1.5 Application Commissioning

### 2.1.5.1 Commissioning Applications on Windows

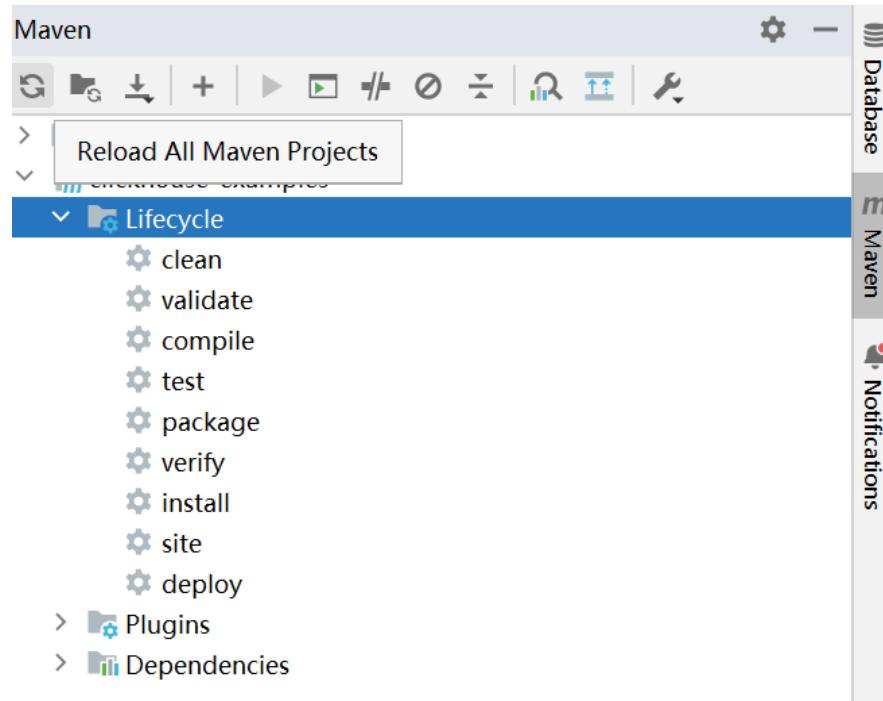
#### Writing and Running an Application

You can run an application in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

#### Procedure

- Step 1** Click **Reload All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

**Figure 2-2** Reloading projects

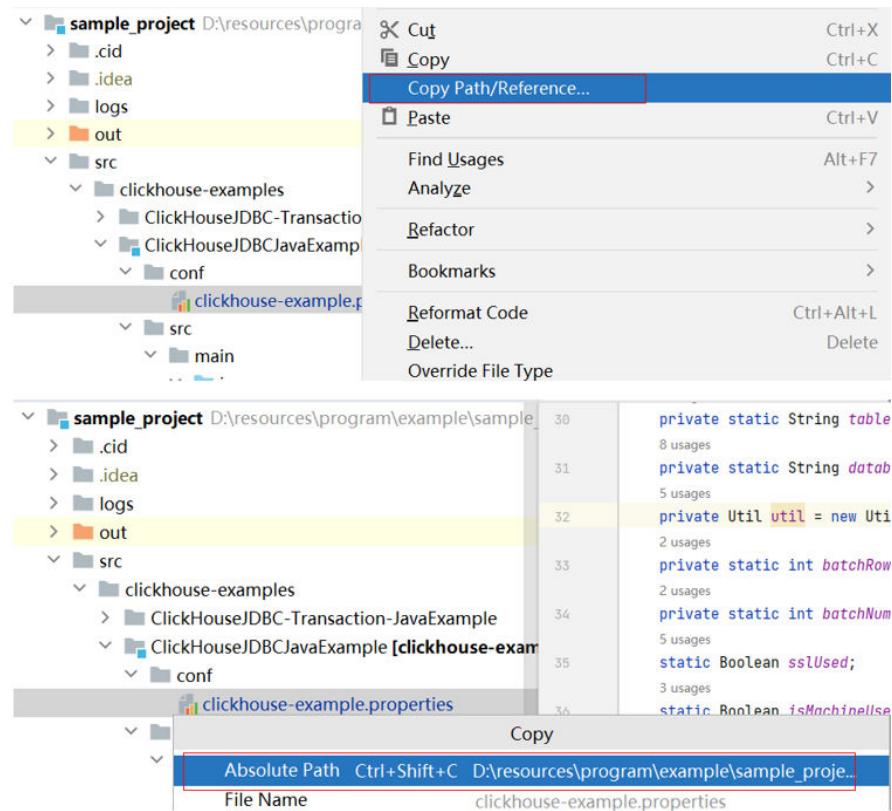


- Step 2** Copy the **clickhouse-example.properties** path on the IDEA page. Right-click **clickhouse-example.properties** and choose **Copy Path/Reference > Absolute Path** from the shortcut menu.

**NOTE**

Skip this step if you are commissioning the transaction sample project.

**Figure 2-3** Copying absolute path of the configuration file



**Step 3** In **Demo.java**, replace the path of **proPath** in the **getProperties()** method with the path **clickhouse-example.properties**.

**Figure 2-4** Replacing the path

```
private void getProperties() throws Exception {
    Properties properties = new Properties();
    String proPath = "D:\\resources\\program\\example\\sample_project\\src\\clickhouse-examples\\ClickHouseJDBCJavaExample\\conf\\clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(new File(proPath)));
    } catch (IOException e) {
        log.error("Failed to load properties file.");
        throw e;
    }
}
```

**Step 4** In **ClickhouseJDBCHaDemo.java**, replace the path of **proPath** in the **initConnection()** method with the path **clickhouse-example.properties**.

#### BOOK NOTE

Skip this step if you are commissioning the transaction sample project.

**Figure 2-5** Replacing the path

```
public void initConnection() throws Exception {
    Properties properties = new Properties();
    String proPath = "D:\\resources\\program\\example\\sample_project\\src\\clickhouse-examples\\ClickHouseJDBCJavaExample\\conf\\clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(proPath));
    } catch (IOException e) {
        LOG.error("Failed to load properties file.");
        throw e;
    }
}
```

**Step 5** If SSL is disabled in the **clickhouse-example.properties** configuration file, replace the **proPath** path in the **initConnection()** method with the **clickhouse-example.properties** path in **NativeJDBCHaDemo.java**.

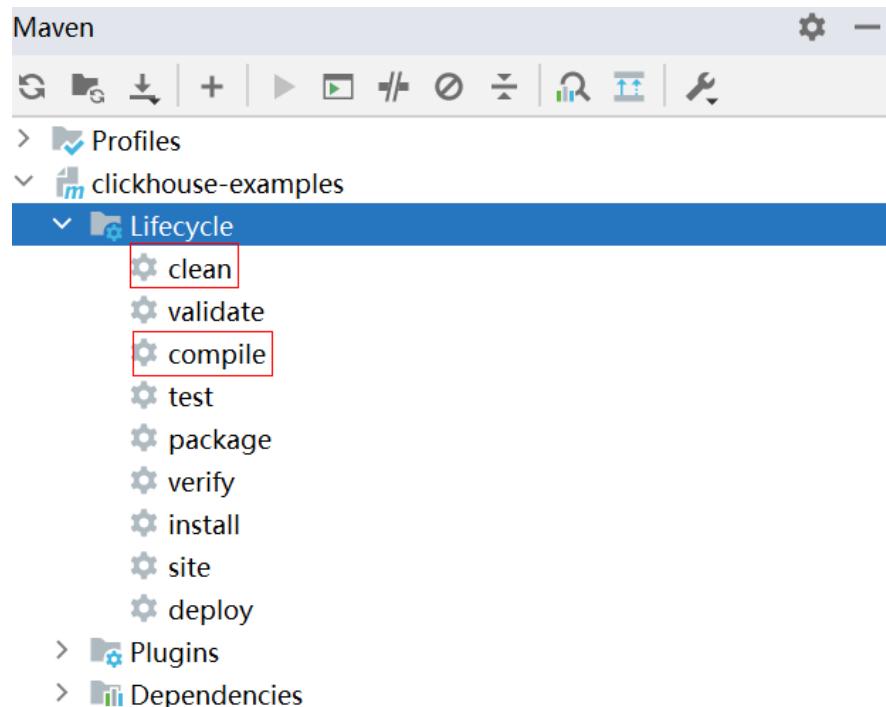
**Figure 2-6** Replacing the path

```
public void initConnection() throws IOException {
    Properties properties = new Properties();
    String proPath = "D:\\resources\\program\\example\\sample_project\\src\\clickhouse-examples\\ClickHouseJDBCJavaExample\\conf\\clickhouse-example.properties";
    try {
        properties.load(new FileInputStream(proPath));
    } catch (IOException e) {
        LOG.error("Failed to load properties file.");
        throw e;
    }
}
```

**Step 6** Compile the application.

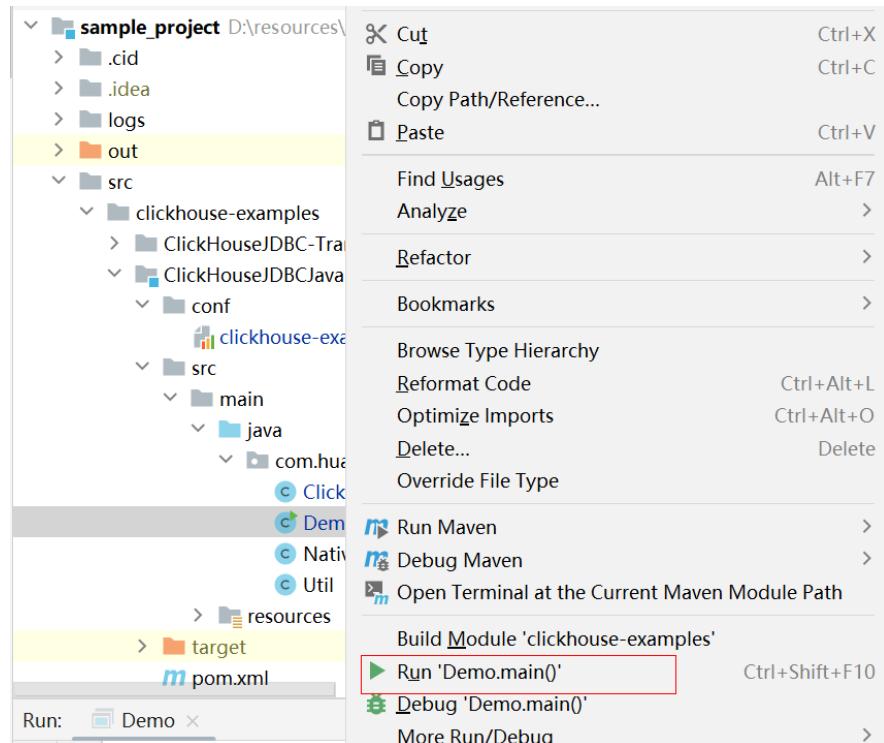
- Select **Maven > clickhouse-examples > Lifecycle > clean** and double-click **clean** to run the **clean** command of Maven.
- Select **Maven > clickhouse-examples > Lifecycle > compile** and double-click **compile** to run the **compile** command of Maven.

**Figure 2-7** Maven commands clean and compile



**Step 7** Click Run'Demo.main()' to run the application.

**Figure 2-8** Running the application



----End

## Viewing the Commissioning Result

After a ClickHouse application finishes running, you can use one of the following methods to view the running result:

- View the running result.
- View the **clickhouse-example.log** file in the **logs** directory to obtain the application running status.

After the complete sample of the **clickhouse-examples** finishes running, the following information is displayed on the console:

```
2023-09-19 16:20:48,344 | INFO | main | loadBalancerIPList is 192.168.5.132, loadBalancerHttpPort is 21422, user is ck_user, clusterName is default_cluster, isSec is true, password is xxx. | com.huawei.clickhouse.examples.Demo.main(Demo.java:42)
2023-09-19 16:20:48,350 | INFO | main | ckLbServerList current member is 0, ClickhouseBalancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Demo.getCKLbServerList(Demo.java:110)
2023-09-19 16:20:48,436 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:50,781 | INFO | main | Execute query:drop table if exists testdb.testtb on cluster default_cluster no delay | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:51,504 | INFO | main | Execute time is 723 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:51,511 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:51,897 | INFO | main | Execute query:drop table if exists testdb.testtb_all on cluster default_cluster no delay | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:52,421 | INFO | main | Execute time is 524 ms | com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:52,422 | INFO | main | Current load balancer is 192.168.5.132:21422 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:52,946 | INFO | main | Execute query:create database if not exists testdb on cluster default_cluster | com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
```

```
2023-09-19 16:20:53,405 | INFO | main | Execute time is 458 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:53,406 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:53,757 | INFO | main | Execute query:create table testdb.testtb on cluster default_cluster
(name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/'
testdb.testtb','{replica}') partition by toYYYYMM(date) order by age |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:54,243 | INFO | main | Execute time is 485 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:54,244 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:20:54,640 | INFO | main | Execute query:create table testdb.testtb_all on cluster
default_cluster as testdb.testtb ENGINE = Distributed(default_cluster,testdb,testtb, rand()); |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:20:55,175 | INFO | main | Execute time is 535 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:20:55,175 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.insertData(Util.java:143)
2023-09-19 16:20:58,868 | INFO | main | Insert batch time is 503 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:160)
2023-09-19 16:21:01,015 | INFO | main | Insert batch time is 631 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:160)
2023-09-19 16:21:02,521 | INFO | main | Inert all batch time is 4163 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:164)
2023-09-19 16:21:02,522 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:21:03,051 | INFO | main | Execute query:select * from testdb.testtb_all order by age limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:21:03,430 | INFO | main | Execute time is 379 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:21:03,433 | INFO | main | Current load balancer is 192.168.5.132:21422 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:68)
2023-09-19 16:21:03,760 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
testdb.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:73)
2023-09-19 16:21:04,361 | INFO | main | Execute time is 600 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:77)
2023-09-19 16:21:04,362 | INFO | main | name age date |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,362 | INFO | main | huawei_9 12 2021-04-20 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,362 | INFO | main | huawei_17 15 2021-05-23 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_5 24 2021-04-15 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_13 39 2020-07-04 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_3 49 2021-06-27 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_15 50 2020-06-26 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_11 53 2020-08-14 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_12 56 2021-12-19 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_19 57 2021-10-31 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | huawei_0 57 2020-03-01 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,363 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202105 3 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202110 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202104 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202008 2 |
```

```
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202007 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202106 2 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202012 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202109 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,364 | INFO | main | 202003 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:04,365 | INFO | main | 202011 1 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:158)
2023-09-19 16:21:05,044 | INFO | main | Name is: huawei_9, age is: 12 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,044 | INFO | main | Name is: huawei_17, age is: 15 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_5, age is: 24 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_13, age is: 39 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_3, age is: 49 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_15, age is: 50 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_11, age is: 53 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_12, age is: 56 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,045 | INFO | main | Name is: huawei_19, age is: 57 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
2023-09-19 16:21:05,046 | INFO | main | Name is: huawei_0, age is: 57 |
com.huawei.clickhouse.examples.ClickhouseJDBCHaDemo.queryData(ClickhouseJDBCHaDemo.java:78)
```

Process finished with exit code 0

### 2.1.5.2 Commissioning Applications on Linux

ClickHouse applications can run in a Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

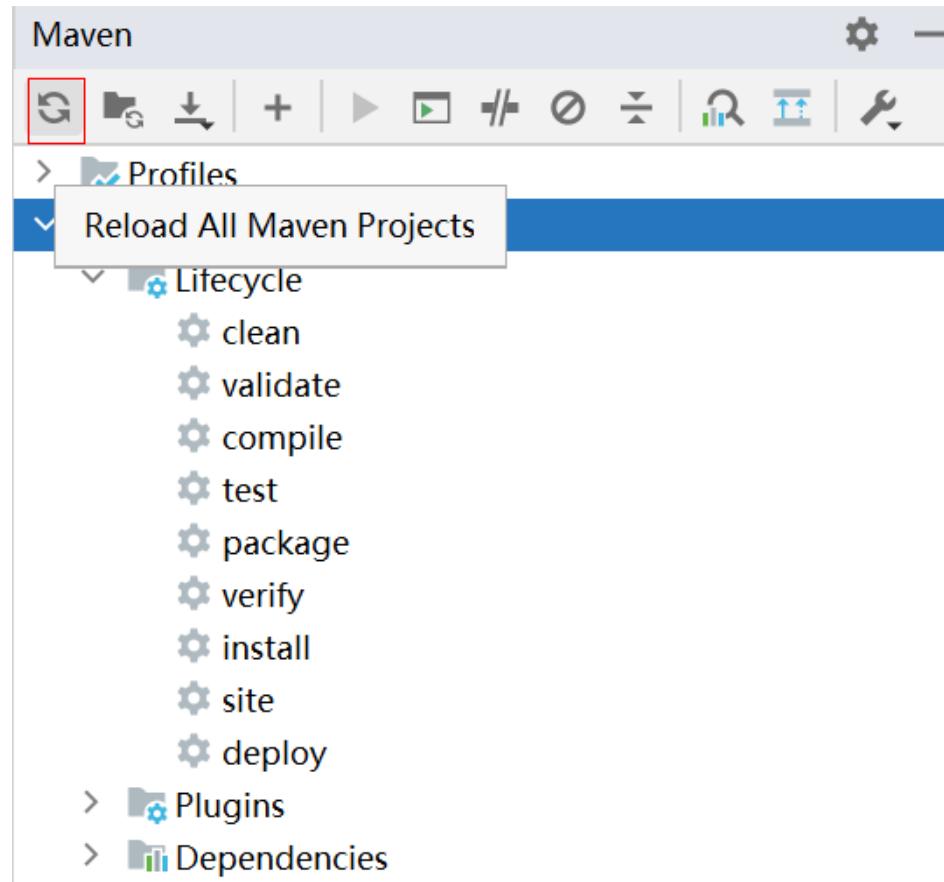
#### Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

#### Compile and run applications.

**Step 1** Click **Reload All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

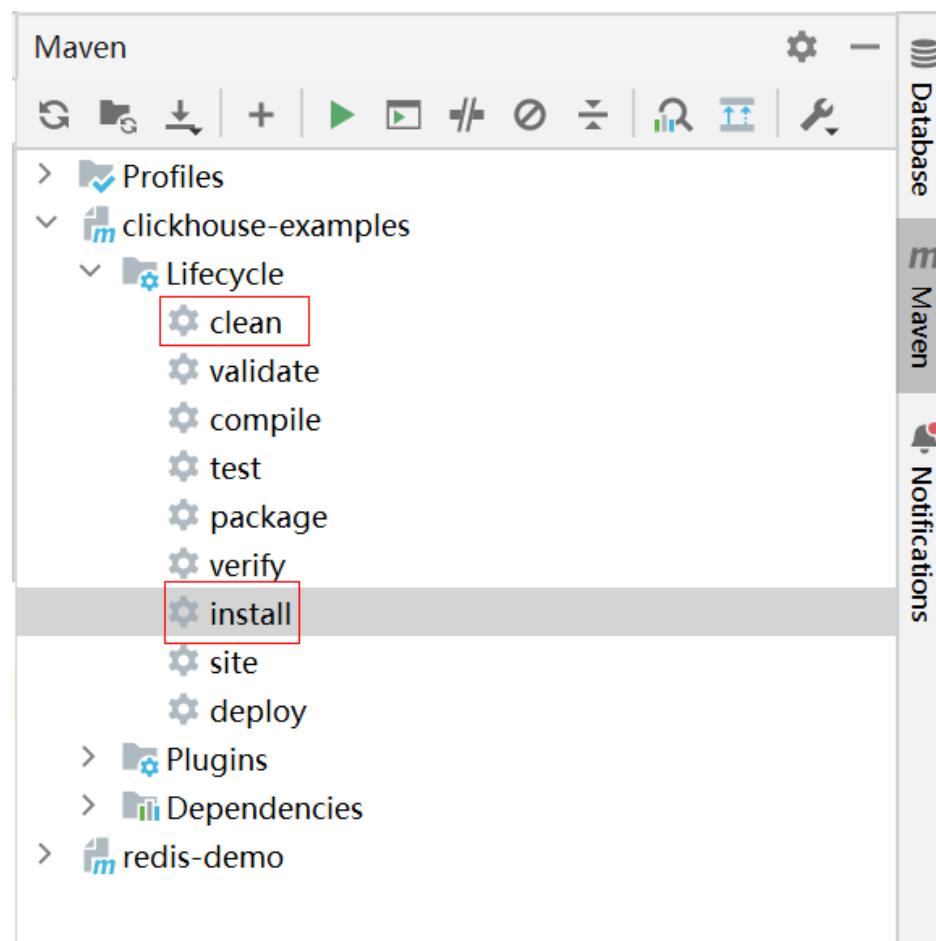
Figure 2-9 reload projects



**Step 2** Export the JAR file.

- Select **Maven > clickhouse-examples > Lifecycle > clean** and double-click **clean** to run the **clean** command of Maven.
- Select **Maven > clickhouse-examples > Lifecycle > install** and double-click **install** to run the **install** command of Maven.

Figure 2-10 Maven clean and install commands



- Step 3** Copy the **clickhouse-examples-\*jar** file from the **target** directory and the **conf** folder from the **clickhouse-examples** directory to the ClickHouse client installation directory, for example, *Client installation directory/JDBC* or *Client installation directory/JDBCTransaction*.

NOTE

The *Client installation directory/JDBC* directory is used to commission JDBC secondary samples.

The *Client installation directory/JDBCTransaction* directory is used to commission secondary transaction samples.

- Step 4** Log in to the client node, go to the directory where the JAR file is uploaded, and change the file permission to 700.

**cd** *Client installation directory/JDBC*

Or **cd** *Client installation directory/JDBCTransaction*

**chmod 700 clickhouse-examples-\*jar**

- Step 5** In the client directory where **clickhouse\_examples.jar** is stored, run the following commands to run the JAR file:

**source** *Client installation directory/bigdata\_env*

**cd** *Client installation directory/JDBC*

Or **cd Client installation directory/JDBCTransaction**

**java -jar clickhouse-examples-\*.jar**

----End

## Viewing Commissioning Results

After a ClickHouse application is run, you can use one of the following methods to view the running result:

- Viewing the command output
- Viewing ClickHouse logs

View the **logs clickhouse-example.log** file in the directory where the current JAR file is located, for example, *Client installation directory/JDBC/logs/ clickhouse-example.log* or *Client installation directory/JDBCTransaction/ logs/clickhouse-example.log*.

The execution result of the JAR file is as follows:

```
2021-06-10 20:53:56,028 | INFO | main | Current load balancer is 10.112.17.150:21426 |
com.huawei.clickhouse.examples.Util.insertData(Util.java:128)
2021-06-10 20:53:58,247 | INFO | main | Inert batch time is 1442 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:53:59,649 | INFO | main | Inert batch time is 1313 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:05,872 | INFO | main | Inert batch time is 6132 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:10,223 | INFO | main | Inert batch time is 4272 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:11,614 | INFO | main | Inert batch time is 1300 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:12,871 | INFO | main | Inert batch time is 1200 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:14,589 | INFO | main | Inert batch time is 1663 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:16,141 | INFO | main | Inert batch time is 1500 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:17,690 | INFO | main | Inert batch time is 1498 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:19,206 | INFO | main | Inert batch time is 1468 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:145)
2021-06-10 20:54:19,207 | INFO | main | Inert all batch time is 22626 ms |
com.huawei.clickhouse.examples.Util.insertData(Util.java:148)
2021-06-10 20:54:19,208 | INFO | main | Current load balancer is 10.112.17.150:21426 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-06-10 20:54:20,231 | INFO | main | Execute query:select * from mutong1.testtb_all order by age
limit 10 | com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-06-10 20:54:21,266 | INFO | main | Execute time is 1035 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-06-10 20:54:21,267 | INFO | main | Current load balancer is 10.112.17.150:21426 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:58)
2021-06-10 20:54:21,815 | INFO | main | Execute query:select toYYYYMM(date),count(1) from
mutong1.testtb_all group by toYYYYMM(date) order by count(1) DESC limit 10 |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:63)
2021-06-10 20:54:22,897 | INFO | main | Execute time is 1082 ms |
com.huawei.clickhouse.examples.Util.exeSql(Util.java:67)
2021-06-10 20:54:22,898 | INFO | main | name    age   date   |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,898 | INFO | main | huawei_266  0  2021-12-19   |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_2500  0  2021-12-29   |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_8980  0  2021-12-16   |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_671   0  2021-12-29   |
```

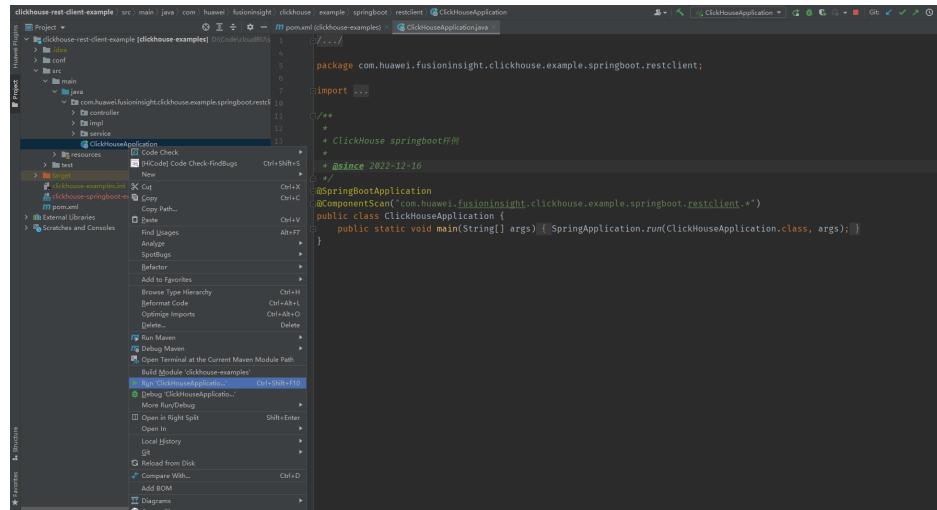
```
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_2225 0 2021-12-12 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_6040 0 2021-12-14 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_7294 0 2021-12-10 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,899 | INFO | main | huawei_1133 0 2021-12-25 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | huawei_3161 0 2021-12-21 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | huawei_3992 0 2021-11-25 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | toYYYYMM(date) count() |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201910 2247 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 202105 2213 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201801 2208 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,900 | INFO | main | 201803 2204 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201810 2167 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201805 2166 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201901 2164 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201908 2145 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 201912 2143 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
2021-06-10 20:54:22,901 | INFO | main | 202107 2137 |
com.huawei.clickhouse.examples.Demo.queryData(Demo.java:144)
```

## 2.1.5.3 Commissioning the Spring Boot Sample Project

### 2.1.5.3.1 Commissioning the Spring Boot Project in Windows

#### Compiling and Running Applications

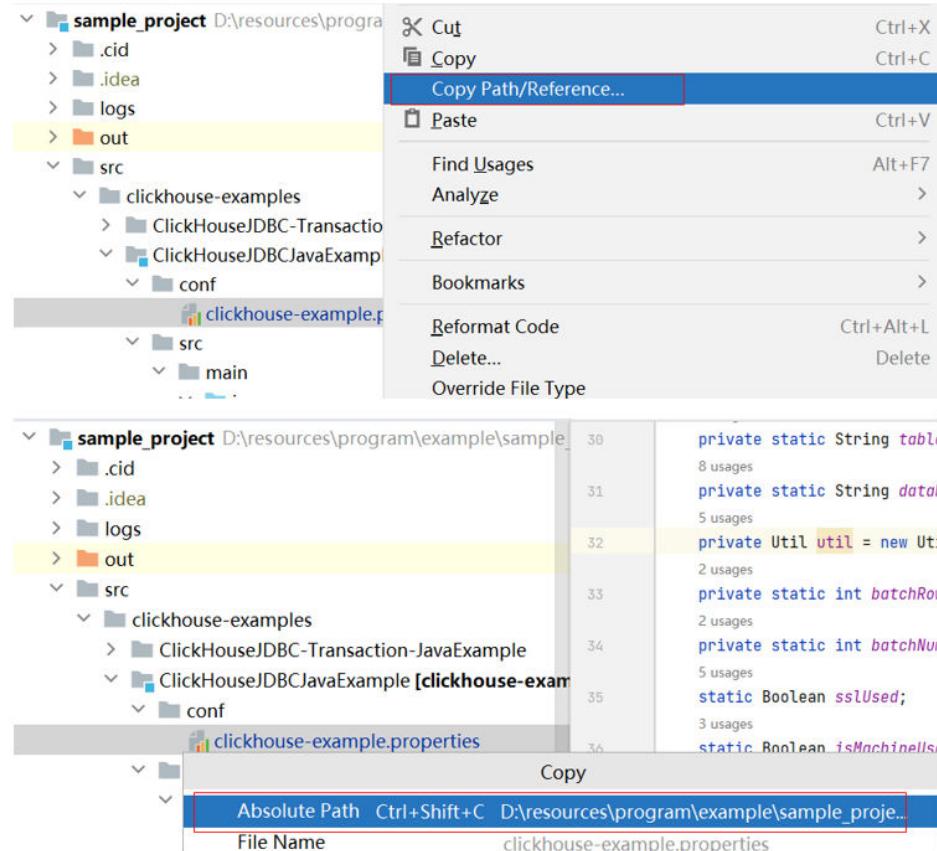
You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host. Right-click **ClickHouseApplication** in IntelliJ IDEA project **clickhouse-rest-client-examples** in the development environment, and then choose **Run ClickHouseApplication** from the shortcut menu to run the application project.



## Procedure

**Step 1** Copy the **clickhouse-example.properties** path on the IDEA page. Right-click **clickhouse-example.properties** and choose **Copy Path/Reference > Absolute Path** from the shortcut menu.

**Figure 2-11** Copying absolute path of the configuration file



**Step 2** In `ClickHouseFunc.java`, replace the path of `proPath` in the `getProperties()` method with `clickhouse-example.properties`.

**Figure 2-12** Replacing the path



----End

# **Viewing Commissioning Results**

1. After the ClickHouse Spring Boot service is started, the ClickHouse sample interface is used to trigger the running of the sample code. Enter the link of the specific operation to be performed in the address box of the browser, for example, <http://localhost:8080/clickhouse/executeQuery>. The following result is returned:  
ClickHouse springboot client runs normally.
  2. View the **clickhouse-springboot-example.log** file to obtain the application running status.

After the `clickhouse-springboot` sample is run, the following information is displayed on the console:

```
Driver registered
2023-02-06 17:58:32.665 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:35.494 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      :
Execute sql drop table if exists testdb.testtb on cluster default_cluster no delay, time is 216 ms
2023-02-06 17:58:35.495 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:36.074 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      :
Execute sql drop table if exists testdb.testtb_all on cluster default_cluster no delay, time is 196 ms
2023-02-06 17:58:36.074 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:36.765 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      :
Execute sql create database if not exists testdb on cluster default_cluster, time is 218 ms
2023-02-06 17:58:36.765 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:37.364 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      :
Execute sql create table testdb.testtb on cluster default_cluster (name String, age UInt8, date
Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/testdb.testtb','{replica}') partition by
toYYYYMM(date) order by age, time is 198 ms
2023-02-06 17:58:37.366 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:37.872 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      :
Execute sql create table testdb.testtb_all on cluster default_cluster as testdb.testtb ENGINE =
Distributed(default_cluster,testdb,testtb,rand()), time is 160 ms
2023-02-06 17:58:38.959 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 591 ms
2023-02-06 17:58:41.114 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 572 ms
2023-02-06 17:58:43.095 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 413 ms
2023-02-06 17:58:45.220 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 543 ms
2023-02-06 17:58:47.207 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 406 ms
2023-02-06 17:58:49.236 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 460 ms
2023-02-06 17:58:51.197 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 390 ms
2023-02-06 17:58:53.233 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 466 ms
2023-02-06 17:58:55.355 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 555 ms
2023-02-06 17:58:57.321 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Insert
batch time is 386 ms
2023-02-06 17:58:58.832 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Inert
all batch time is 20564 ms
2023-02-06 17:58:58.832 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:59.256 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      :
Execute sql select * from testdb.testtb_all order by age limit 10, time is 119 ms
2023-02-06 17:58:59.257 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      : Try
times is 0, current load balancer is 192.168.6.24:21422.
2023-02-06 17:58:59.972 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.restclient.impl.Util      :
Execute sql select toYYYYMM(date),count(1) from testdb.testtb_all group by toYYYYMM(date) order
by count(1) DESC limit 10, time is 289 ms
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc    :
huawei_234 0 2022-12-16
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc    :
huawei_1805 0 2022-12-21
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc    :
huawei_3359 0 2022-12-13
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc    :
huawei_4275 0 2022-12-09
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc    :
huawei_4307 0 2022-12-20
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc    :
huawei_4586 0 2022-12-02
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc    :
huawei_6326 0 2022-12-18
```

```
2023-02-06 17:58:59.973 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
huawei_9878 0 2022-12-06  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
huawei_2482 0 2022-12-18  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
huawei_2904 0 2022-12-25  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202201 8628  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202208 8587  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202205 8558  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202210 8547  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202207 8463  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202203 8379  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202204 8351  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202206 8300  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202209 8297  
2023-02-06 17:58:59.974 INFO 20556 --- [nio-8080-exec-1] c.h.f.c.e.s.r.impl.ClickHouseFunc :  
202212 8228
```

### 2.1.5.3.2 Commissioning the Spring Boot Project in Linux

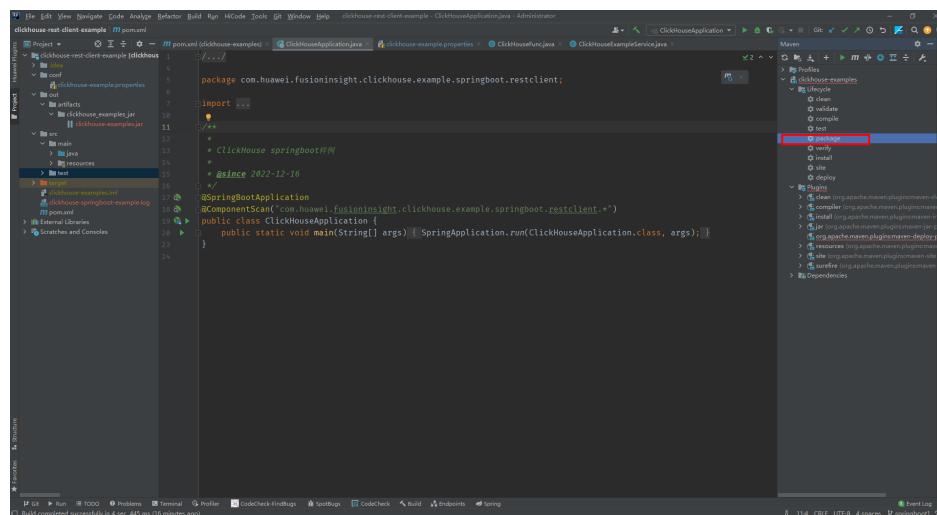
The ClickHouse Spring Boot applications can run in a Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

## Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

## Compiling and Running Applications

**Step 1** Click **Maven** on the right of IDEA, expand **Lifecycle**, and double-click **package** to package the current project.



**Step 2** Log in to the ClickHouse client node as user **root**, create a running directory, for example, **/opt/test**, and obtain the JAR package whose name contains **-with-dependencies** from the **target** directory of IDEA. Upload the JAR package and the **conf** folder in the IDEA to the **/opt/test** directory, as shown in the following figure.

```
[root@host-192-168-6-24 test]#  
[root@host-192-168-6-24 test]# pwd  
/opt/test  
[root@host-192-168-6-24 test]# ll  
total 25048  
-rw-r--r-- 1 root root 25644916 Feb 6 19:24 clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar  
drwxr-xr-x 2 root root 4096 Feb 6 18:03 conf  
[root@host-192-168-6-24 test]#
```

**Step 3** Run the following commands to set environment variables and run the JAR package:

**cd** *Client installation path*

```
source bigdata_env
```

```
cd /opt/test
```

```
java -jar clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

The command output is displayed as follows:

```
[root@host-192-168-6-24 test]# java -jar clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO ] 2023-02-06 19:42:04.200 main c.h.f.e.s.r.ClickHouseApplication : Starting ClickHouseApplication using Java 1.8.0_332 on host-192-168-6-24 with PID 42496 [/opt/test	clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar started by root in /opt/test]
[INFO ] 2023-02-06 19:42:04.203 main c.h.f.e.s.r.ClickHouseApplication : No active profile set, falling back to 1 default profile: "default"
[INFO ] 2023-02-06 19:42:04.396 INFO 24296 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
[INFO ] 2023-02-06 19:42:04.409 INFO 24296 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
[INFO ] 2023-02-06 19:42:04.410 INFO 24296 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.63]
[INFO ] 2023-02-06 19:42:04.410 INFO 24296 --- [main] org.apache.catalina.startup.Tomcat : Initialization completed in 1188 ms
[INFO ] 2023-02-06 19:42:04.476 INFO 24296 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
[INFO ] 2023-02-06 19:42:04.074 INFO 24296 --- [main] c.h.f.e.s.r.ClickHouseApplication : Started ClickHouseApplication in 2.718 seconds (JVM running for 3.07)
```

**Step 4** Call the Spring Boot sample API of ClickHouse to trigger running the sample code.

- Running method in Windows:

Open the browser, enter `http://IP address of the ClickHouse client node:8080/clickhouse/executeQuery` in the address box, and check the returned information.

ClickHouse springboot client runs normally.

- Running method in Linux:

Log in to the ClickHouse client node and run the following command to view shell logs and log files in Linux:

```
curl http://localhost:8080/clickhouse/executeQuery
```

## vi clickhouse-springboot-example.log

```
[root@host ~]# test # ll
total 25000
-rw-r--r-- 1 root root 25644915 Feb 6 10:24 clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
-rwxr-xr-x 2 root root 9384 Feb 6 15:45 clickhouse-springboot-example.log
drwxr-xr-x 2 root root 4096 Feb 6 15:03 .
[root@host ~]# java -jar clickhouse-springboot-example.log
2023-02-06 19:42:02.200 INFO 24296 -- [main] c.h.c.e.s.r.CClickHouseApplication : Starting ClickHouseApplication using Java 1.8.0_322 on host [REDACTED] with PID 24296 [/opt/test/clickhouse-examples-1.0-SNAPSHOT-jar-with-dependencies.jar started by root in /opt/test]
2023-02-06 19:42:02.300 [main] o.s.b.a.embedded.Tomcat : Tomcat initialized with port(s): 8080 (http)
2023-02-06 19:42:02.306 [main] o.s.b.a.embedded.Tomcat : Tomcat initialized with port(s): 8080 (http)
2023-02-06 19:42:02.309 [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-02-06 19:42:02.310 [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.63]
2023-02-06 19:42:02.316 [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-02-06 19:42:02.317 [main] o.w.s.c.WebApplicationContext : Root WebApplicationContext: initialization completed in 1188 ms
2023-02-06 19:42:02.318 [main] o.s.b.a.embedded.Tomcat : Tomcat started on port(s): 8080 [http://[REDACTED]:8080] with context path ''
2023-02-06 19:42:02.404 [main] c.h.c.e.s.ClickHouseApplication : Started ClickHouseApplication in 2.718 seconds (JVM running for 3.077 ms)
2023-02-06 19:43:53.323 [INFO] 24296 -- [http://nio-8080-exec-1].o.s.e.c.DispatcherServlet : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-02-06 19:43:53.324 [INFO] 24296 -- [http://nio-8080-exec-1].o.s.e.c.DispatcherServlet : Completed initialization in 2 ms
2023-02-06 19:43:53.326 [INFO] 24296 -- [http://nio-8080-exec-1].o.s.w.s.DispatcherServlet : Begin to execute query in clickhouse...
2023-02-06 19:43:53.375 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.ClickHouseExampleController : [dbBalancerPlist is [REDACTED], loadBalancerPort is [REDACTED]]
2023-02-06 19:43:53.381 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.ClickHouseFunc : cLbServerList current member is 0, ClickHouseBalancer is [REDACTED]
2023-02-06 19:43:53.391 [INFO] 24296 -- [http://nio-8080-exec-1].r.yandex.clickhouse.ClickHouseDriver : Driver registered
2023-02-06 19:43:54.257 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Try times is 0, current load balancer is [REDACTED]
2023-02-06 19:43:54.257 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Execute sql drop table if exists testdb.testtb on cluster default
2023-02-06 19:43:54.330 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Try times is 0, current load balancer is [REDACTED]
2023-02-06 19:43:54.330 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Execute sql drop table if exists testdb.testtb_all on cluster default
2023-02-06 19:43:54.377 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Try times is 0, current load balancer is [REDACTED]
2023-02-06 19:43:54.377 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Execute sql drop table if exists testdb.testtb_all on cluster default
2023-02-06 19:43:54.890 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Try times is 0, current load balancer is [REDACTED]
2023-02-06 19:43:54.890 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Execute sql drop table if exists testdb.testtb on cluster default
2023-02-06 19:43:55.131 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Try times is 0, current load balancer is [REDACTED]
2023-02-06 19:43:55.131 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Execute sql create database if not exists testdb on cluster default
2023-02-06 19:43:55.177 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Try times is 0, current load balancer is [REDACTED]
2023-02-06 19:43:55.177 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Execute sql create database if not exists testdb on cluster default
2023-02-06 19:43:55.188 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Try times is 0, current load balancer is [REDACTED]
2023-02-06 19:43:55.188 [INFO] 24296 -- [http://nio-8080-exec-1].c.h.f.c.e.s.restclient.impl.Util : Execute sql create database if not exists testdb on cluster default
```

----End

## 2.2 Doris Development Guide

## 2.2.1 Overview

## 2.2.1.1 Doris Introduction

Doris is a high-performance and real-time analytical database based on the MPP architecture. Doris is well known for its high speed and ease of use. It can return query results in massive data in sub-second response time and supports high-concurrency point query scenarios. It also supports complex analysis scenarios with high throughput. Based on this, Apache Doris can meet the requirements of report analysis, ad hoc query, unified data warehouse construction, and data lake federated query acceleration. Users can build applications such as user behavior analysis, A/B experiment platform, log retrieval analysis, user profile analysis, and order analysis.

Doris uses the MPP model, which is executed concurrently between nodes and within nodes. It is applicable to the distributed join of multiple large tables. Supports the vectorized query engine, adaptive query execution (AQE), optimization policy that combines CBO and RBO, and hot data cache query.

### 2.2.1.2 Common Concepts

In Doris, data is logically described in the form of tables.

- Row&Column

A table consists of rows and columns:

- Row: indicates a row of user data.
  - Column: describes different fields in a row of data.

Columns can be divided into two categories: Key and Value. From the business perspective, Key and Value can correspond to the dimension column and indicator column respectively. From the point of view of the aggregation model, rows with the same key column are aggregated into

one row. The aggregation mode of the Value column is specified by the user when creating a table.

- **Tablet&Partition**

In the storage engine of Doris, user data is horizontally divided into several data fragments (tablets, also called data buckets). Each Tablet contains several rows of data. Tablet data has no intersection and is physically stored independently.

Multiple tablets belong to different partitions logically. A tablet belongs to only one partition, and a partition contains several tablets. Because tablets are physically independent, partitions can be considered as physically independent. A tablet is the smallest physical storage unit for operations such as data movement and replication.

Multiple partitions form a table. A partition can be considered as the smallest logical management unit. Data can be imported and deleted only for one partition.

- **Data model**

Doris data models are classified into three types: Aggregate, Unique, and Duplicate.

- **Aggregate model**

When data is imported, rows with the same Key column are aggregated into one row, and the Value column is aggregated based on the AggregationType parameter. Currently, the AggregationType supports the following aggregation modes:

- SUM: Sum of values in multiple lines.
- Replace: Replace the value in the next batch of data.
- MAX: retain the maximum value.
- MIN: minimum value.

- **Unique model**

In some multidimensional analysis scenarios, users focus more on how to ensure the uniqueness of keys, that is, how to obtain the uniqueness constraint of the primary key. Therefore, the Unique data model is introduced.

- **Read-on-Merge**

The read-on-merge implementation of the Unique model can be completely replaced by the Replace mode in the Aggregate model. Its internal implementation and data storage are exactly the same.

- **Combine on write**

The unique model is implemented by combining data on write. The query performance of the unique model is closer to that of the duplicate model. Compared with the aggregate model, the unique model has higher query performance than the aggregate model in scenarios where primary key constraints are required, especially in aggregate queries and queries where a large amount of data needs to be filtered by indexes.

In the Unique table with the merge-on-write option enabled, the overwritten and updated data is marked and deleted during the import phase, and new data is written to a new file. During query, all data marked for deletion is filtered out at the file level, and the read data is the latest data. This eliminates the data aggregation process during read merge and supports pushdown of multiple predicates in many cases. Therefore, the performance can be greatly improved in many scenarios, especially in the case of aggregated queries.

- **Duplicate model**

In some multidimensional analysis scenarios, data has neither primary keys nor aggregation requirements. The Duplicate data model can be introduced to meet such requirements.

This data model is different from the Aggregate and Unique models. The data is stored exactly as the data in the import file, and there is no aggregation. Even if the two rows of data are identical, they are retained. The DUPLICATE KEY specified in the table creation statement is only used to specify the columns by which the underlying data is sorted.

- **Data Model Selection Suggestions**

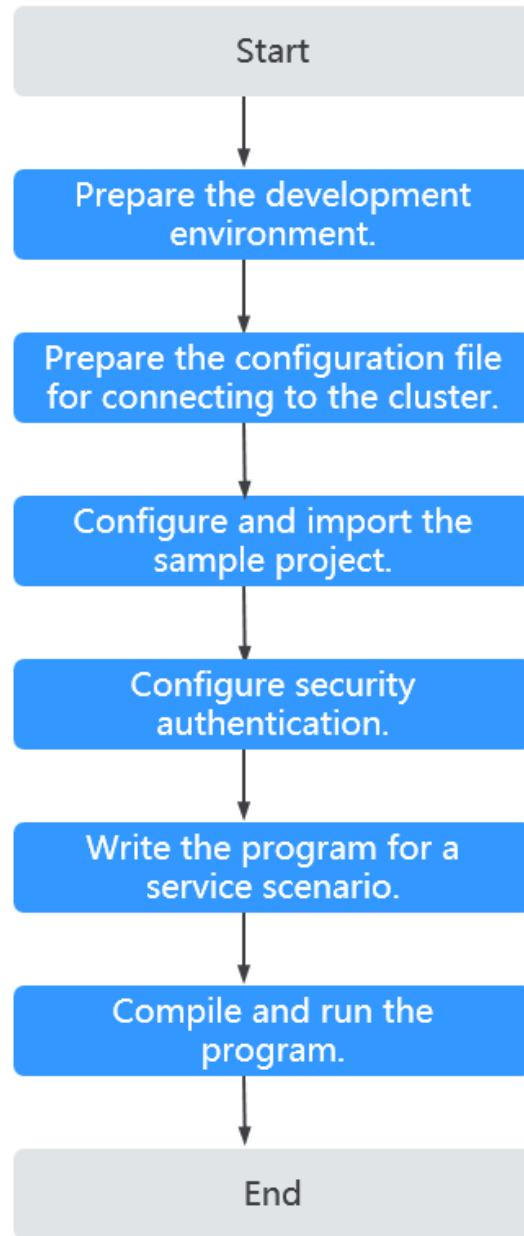
The data model is determined when the table is created and cannot be modified. Therefore, it is important to choose a suitable data model.

- i. The Aggregate model greatly reduces the amount of data to be scanned and the amount of query calculation during aggregation query through pre-aggregation. It is suitable for the report query scenario with fixed mode. However, this model is not friendly to count(\*) queries. In addition, the aggregation mode of the Value column is fixed. Therefore, the semantic correctness needs to be considered when performing other types of aggregation queries.
- ii. The Unique model ensures the uniqueness of primary key constraints in scenarios where unique primary key constraints are required. However, the query advantages of pre-aggregation such as ROLLUP cannot be taken advantage of.
  - 1) For users who have high performance requirements for aggregated queries, the merge-on-write implementation introduced in version 1.2 is recommended.
  - 2) The Unique model supports only the entire row update. If the unique primary key constraint and some columns need to be updated, (e.g., importing multiple source tables into a Doris table), you can use the Aggregate model and set the aggregation type of the non-primary key column to Replaced\_IF\_NOT\_NULL.
  - 3) Duplicate is applicable to Ad-hoc queries in any dimension. Although the pre-aggregation feature cannot be used, it is not restricted by the aggregation model and can take advantage of the column-store model. (Only the relevant columns are read, not all the Key columns).

### 2.2.1.3 Development Process

The following figure shows the phases in the development process.

Figure 2-13 Doris application development process



**Table 2-5 HBase application development process description**

Phase	Description	Reference Document
Prepare the development environment.	Before developing an application, you need to prepare the development environment. You are advised to use the Java language and IntelliJ IDEA tool for development. In addition, you need to complete the initial configuration of the JDK and Maven.	<a href="#">Preparing for Development Environment</a>
Prepare the Configuration File for Connecting to the Cluster.	During application development or running, you need to connect to the MRS cluster through cluster configuration files. The configuration file contains user files used for security authentication. You can obtain related content from the created MRS cluster.  Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured.	<a href="#">Preparing the Configuration Files for Connecting to the Cluster</a>
Configure and import the sample project.	Doris provides multiple sample programs in different scenarios. You can obtain sample projects and import them to the local development environment for program learning.	<a href="#">Configuring and Importing JDBC or Stream Load Sample Projects</a> <a href="#">Configuring and Importing SpringBoot Sample Projects</a>

Phase	Description	Reference Document
Configure security authentication.	When using the JDBC or Spring Boot interface to connect to Doris, you need to configure a user with Doris administrator rights for security authentication.	<a href="#">Configuring and Importing JDBC or Stream Load Sample Projects</a> <a href="#">Configuring and Importing SpringBoot Sample Projects</a>
Develop programs based on business scenarios.	Develop programs based on actual service scenarios and invoke component interfaces to implement corresponding functions.	<a href="#">Using Doris Through JDBC and DBalancer</a>
Compile and run the application.	You can commission and run the program in the local Windows development environment, or compile the program into a JAR package and submit it to the Linux node for running.	<a href="#">Application Commissioning</a>

#### 2.2.1.4 Doris Sample Project Introduction

To obtain the MRS sample project, visit <https://github.com/huaweicloud/huaweicloud-mrs-example>. Switch to the version branch that matches the MRS cluster, download the package to the local PC, and decompress the package to obtain the sample code project of each component.

Currently, MRS provides the following Doris-related sample projects:

**Table 2-6** Doris-related sample projects

Sample Project Location	Function Description
doris-examples/doris-jdbc-example	Application development example of Doris data read and write operations. You can invoke Doris APIs to create user tables, insert data into tables, query table data, and delete tables. For details, see <a href="#">Using Doris Through JDBC and DBalancer</a> .

Sample Project Location	Function Description
springboot/doris-examples	SpringBoot application development example for Doris data read and write operations. This section provides an example for connecting Doris to SpringBoot. For details, see <a href="#">Configuring and Importing SpringBoot Sample Projects</a> .
doris-examples/doris-stream-load-example	Application development example of the Doris Stream Load API for importing local CSV files. This section provides an example of using the Stream Load API to import data from a local CSV file to a Doris table. For details about the example, see <a href="#">Loading Data to a Doris Table with Stream Load</a> .

## 2.2.2 Environment Preparation

### 2.2.2.1 Preparing for Development Environment

[Table 2-7](#) describes the environment required for secondary development.

**Table 2-7** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>

Item	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
JUnit plug-in installation	Basic configuration for the development environment.

Item	Description
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.  Huawei provides Huawei Mirrors for you to download all dependency JAR files of sample projects. However, you need to download the rest dependency open-source JAR files from the Maven central repository or other custom repository address. For details, see <a href="#">Configuring Huawei Open-Source Mirrors</a> .
7-zip	Used to decompress .zip and .rar packages. The 7-Zip 16.04 is supported.

### 2.2.2.2 Preparing the Configuration Files for Connecting to the Cluster

#### Preparing a Developer Account

If Kerberos authentication is not enabled for an MRS cluster, you need to prepare a user who has the permission to perform operations on related components for program authentication.

The following Doris permission configuration example is for reference only. You can adjust the permission configuration based on service requirements in actual service scenarios.

**Step 1** Log in to the node where the MySQL client is installed and connect to the Doris service as user **admin**.

```
mysql -uadmin -PDatabase connection port -hDoris FE instance IP address
```



#### NOTE

- The default password of the **admin** user is empty.
- The database connection port is the query connection port of the Doris FE. The default port is 29982. You can also log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and query the **query\_port** parameter of the Doris service.
- To view the IP address of a Doris FE instance, log in to FusionInsight Manager of the MRS cluster and choose **Cluster > Services > Doris > Instances**.
- You can also use the MySQL connection software or Doris WebUI to connect to the database.

**Step 2** Run the following command to create a role:

```
Create Role dorisrole;
```

**Step 3** Run the following command to grant permissions to the role. For details, see "Doris Permission Management" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*. For example, to grant the ADMIN\_PRIV permission to the role, run the following command:

```
GRANT ADMIN_PRIV ON *.* TO ROLE 'dorisrole';
```

**Step 4** Run the following commands to create a user and bind the user to the role:

```
CREATE USER 'dorisuser'@'%' IDENTIFIED BY 'password' DEFAULT ROLE  
'dorisrole';
```

Commands carrying authentication passwords pose security risks. Disable historical command recording before running such commands to prevent information leakage.

----End

## Configuring the Network of the Running Environment

Nodes used for program debugging or running must be able to communicate with nodes within the MRS cluster, and the host domain names must be configured.

- Scenario 1: Configure the network connection between the local Windows environment and MRS cluster nodes.
  - a. Log in to FusionInsight Manager, click **Download Client** in the upper right corner of **Homepage**, select **Configuration Files Only** for **Select Client Type**, and click **OK**. After the client file package is generated, download the client to the local PC as prompted and decompress it.  
For example, if the client configuration file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Then, continue to decompress this file.
  - b. Copy the **hosts** file content from the decompression directory to the **hosts** file of the local PC.

### NOTE

- If you need to debug the application in the local Windows environment, ensure that the local PC can communicate with the hosts listed in the **hosts** file.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- When configuring IPv6 hosts on the local PC running Windows, add **%** and the interface index of the network interface card (NIC) to the end of the IPv6 address. The interface index can be obtained by running the **ipconfig** command.

The following is an example:

fec1:0:0:e505:8:99:5:1%10

- Scenario 2: Configure the network communication between the Linux environment and MRS cluster nodes.
  - a. Install the MRS cluster client on nodes.  
For example, the client installation directory can be **/opt/hadoopclient**.
  - b. Obtain the configuration files.
    - i. Log in to FusionInsight Manager, click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.

- ii. Log in to the active OMS node as user **root** and go to the directory where the client configuration file is stored. The default directory is **/tmp/FusionInsight-Client/**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf  
FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles
```

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If they are not contained, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed.

### 2.2.2.3 Configuring and Importing JDBC or Stream Load Sample Projects

#### Background

To run the JDBC or Stream Load API sample code of the Doris component, perform the following operations.

#### Procedure

- Step 1** Obtain the sample project folder **doris-jdbc-example** or **doris-stream-load-example** in the **src\doris-examples** directory in the sample code decompression directory by referring to [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the sample project to the IntelliJ IDEA development environment.
  1. On the menu bar of IntelliJ IDEA, choose **File > Open...** to display the Open File or Project dialog box.
  2. In the displayed dialog box, select the **doris-jdbc-example** or **doris-stream-load-example** folder and click OK. In Windows, the full path of the folder does not contain spaces.

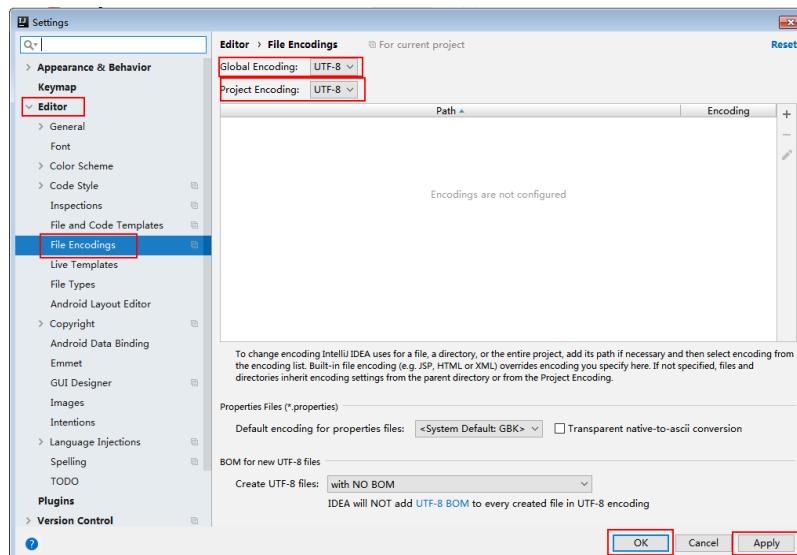
 NOTE

- This section describes how to develop an application that connects to the Doris service in JDBC mode in Windows.
  - Set the **DORIS\_MY\_USER** and **DORIS\_MY\_PASSWORD** environment variables in the local environment variables. Store the environment variables in ciphertext and decrypt the environment variables to ensure security.
    - **DORIS\_MY\_USER** indicates the user name for logging in to the Doris.
    - **DORIS\_MY\_PASSWORD** indicates the password for logging in to the Doris.
- If the error message "Could not connect to *IP address of the FE instance*.MySQL protocol query connection port" is displayed during the execution of the sample program, the configured local environment variables may not take effect. In this case, restart the computer for the settings to take effect.
- After the **jdbc-example** sample project is imported, modify the following parameters:
    - In the code, change `xxx` in **HOST = "xxx"** to the IP address of the master FE node of the Doris. To obtain the IP address of the master FE node, choose **Cluster > Services > Doris** on Manager and view the **Host Where Leader Locates**.
    - In the code, change `xxx` where **PORT = "xxx"** to the MySQL query connection port of Doris. The default port is 29982. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **query\_port**.

**Step 3** Set the encoding format of the IntelliJ IDEA text file to solve the problem of garbled characters.

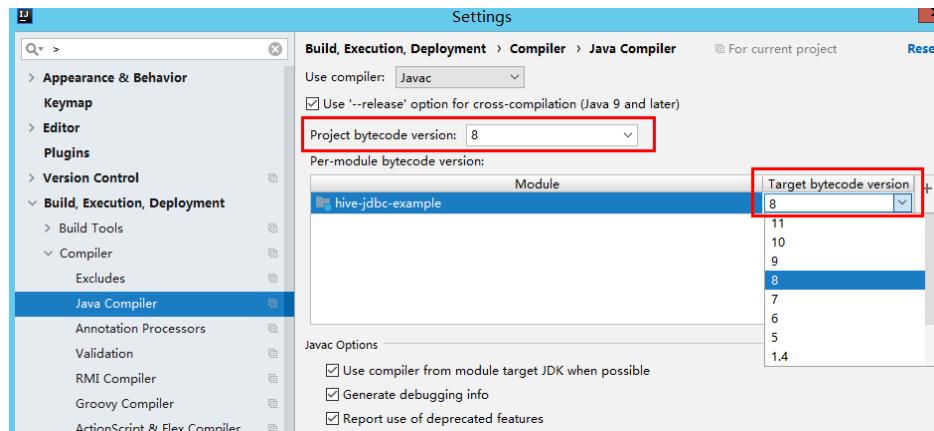
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.  
The Settings window is displayed.
2. In the navigation tree on the left, choose **Editor > File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 2-14](#).

**Figure 2-14** Setting the Encoding Format of IntelliJ IDEA

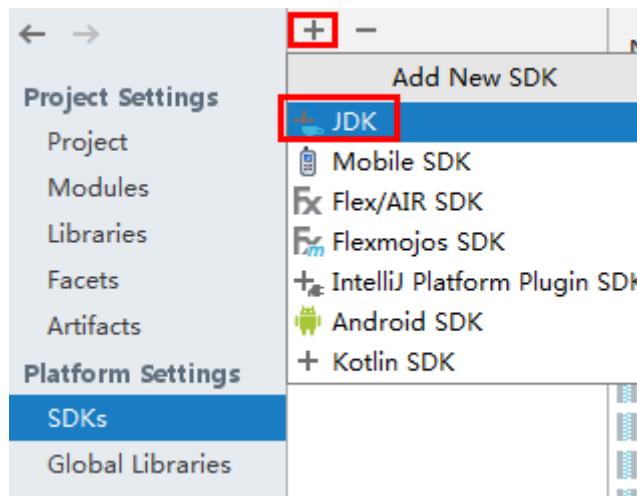


**Step 4** Set the JDK of the project.

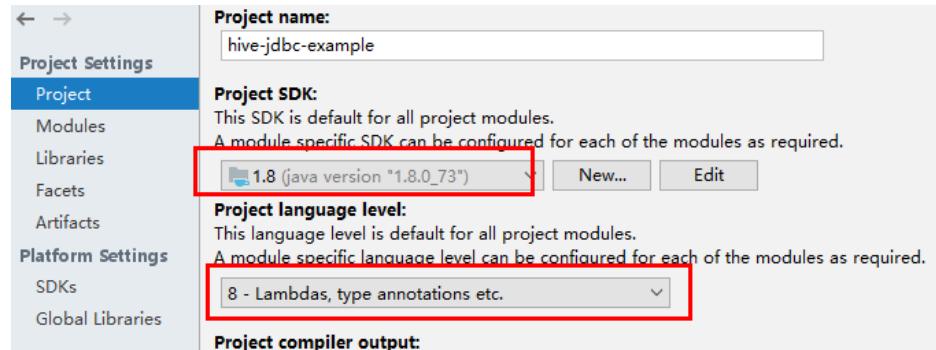
1. On the IntelliJ IDEA menu bar, choose **File > Settings....** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **doris-jdbc-example**.



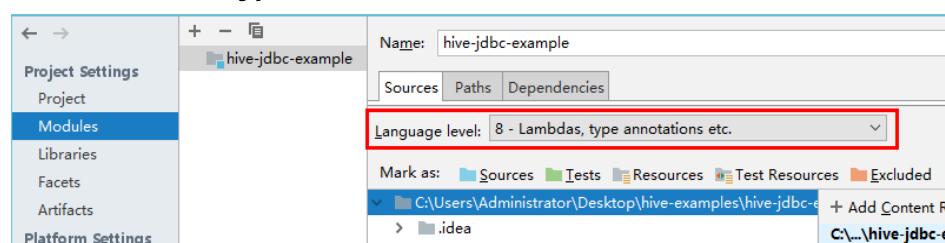
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



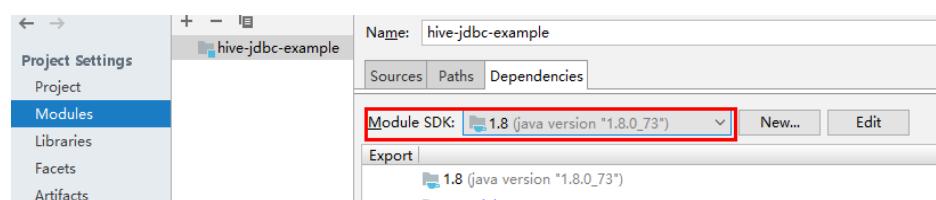
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in **SDKs**.

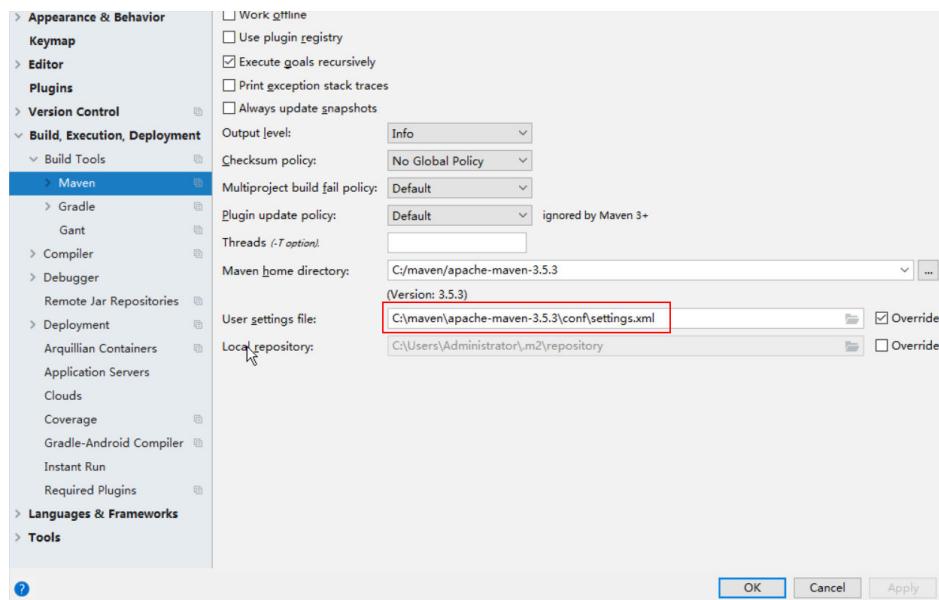


11. Click **Apply** and **OK**.

#### Step 5 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

Figure 2-15 Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

#### 2.2.2.4 Configuring and Importing SpringBoot Sample Projects

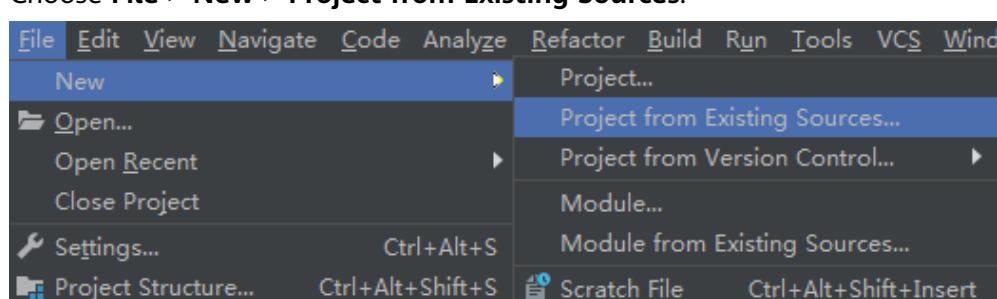
##### Background

To run the SpringBoot API sample code of the Doris component of MRS, perform the following operations:

This section describes how to develop an application that connects to the Doris service in SpringBoot mode in Windows.

##### Procedure

- Step 1** Obtain the sample project folder **doris-rest-client-example** in the **src/springboot/doris-examples** directory in the sample code decompression directory by referring to [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
1. Choose **File > New > Project from Existing Sources**.



2. In the **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **doris-rest-client-example** folder and click **OK**.
3. Confirm the subsequent configuration and click **Next**. If there is no special requirement, use the default value.

Select the recommended JDK version and click **Finish**.

#### NOTE

- This section describes how to develop an application that connects to the Doris service in SpringBoot mode in Windows.
- Set the **DORIS\_MY\_USER** and **DORIS\_MY\_PASSWORD** environment variables in the local environment variables. Store the environment variables in ciphertext and decrypt the environment variables to ensure security.
  - **DORIS\_MY\_USER** indicates the user name for logging in to the Doris.
  - **DORIS\_MY\_PASSWORD** indicates the password for logging in to the Doris.If the error message "Could not connect to *IP address of the FE instance*.MySQL protocol query connection port" is displayed during the execution of the sample program, the configured local environment variables may not take effect. In this case, restart the computer for the settings to take effect.
- After the **doris-rest-client-example** sample project is imported, modify the following parameters:
  - In the code, change `xxx` in `HOST = "xxx"` to the IP address of the master FE node of the Doris. To obtain the IP address of the master FE node, choose **Cluster > Services > Doris** on Manager and view the **Host Where Leader Locates**.
  - In the code, change `xxx` where `PORT = "xxx"` to the MySQL query connection port of Doris. The default port is 29982. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **query\_port**.

----End

#### NOTE

- This section describes how to develop an application that connects to the Doris service in JDBC mode in Windows.
- After the **jdbc-example** sample project is imported, modify the following parameters:
- Change the value of `xxx` under `USER = "xxx"` in the code to the actual development user, for example, `developuser`.
- Change the value of `xxx` under `PASSWD = "xxx"` in the code to the actual password of the development user.
- In the code, change `xxx` in `HOST = "xxx"` to the IP address of the master FE node of the Doris. You can obtain the IP address of the master FE node by choosing **Cluster > Service > Doris** on FusionInsight Manager and checking the host where the leader resides.
- In the code, change `xxx` where `PORT = "xxx"` to the MySQL query connection port of Doris. The default port is 29982. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Service > Doris > Configuration**, and search for **query\_port**.

## 2.2.3 Doris Application Development

### 2.2.3.1 Using Doris Through JDBC and DBalancer

### 2.2.3.1.1 Service Scenario Description

This section describes how to quickly learn the Doris development process and understand key interface functions.

#### Scenario

Doris can use SQL statements to perform common service operations. The SQL operations involved in the code example include creating a database, creating a table, inserting table data, querying table data, deleting a table, and deleting a database (DBalancer).

The code sample operation process is as follows:

1. Connect to Doris using JDBC or DBalancer.
2. Create a database.
3. Create a table in the database.
4. Inserts data into a table.
5. Query table data.
6. Delete the table.
7. Delete a database. (DBalancer)

### 2.2.3.1.2 Application Development Approach

#### Function Decomposition

Doris is compatible with the MySQL protocol. Common operations can be performed by using the SQL language.

The development process consists of the following parts:

- Establish a connection: Establish a connection to the Doris service instance.
- Create Library: Create a Doris database.
- Create Table: Create a table in the Doris database.
- Insert Data: Insert data into the Doris table.
- Querying data: Query the Doris table data.
- Deleting a table: You can delete a created Doris table.
- Deleting a database (DBalancer): You can delete a created Doris database.

### 2.2.3.1.3 Setting Up a Connection

#### Connecting Doris Using JDBC

The following code snippet uses the **createConnection** method of the **JDBCExample** class in the **com/huawei/bigdata/doris/example** package.

```
private static Connection createConnection() throws Exception {  
    Connection connection = null;  
    try {  
        Class.forName(JDBC_DRIVER);  
        String dbUrl = String.format(DB_URL_PATTERN, HOST, PORT);  
        connection = DriverManager.getConnection(dbUrl, USER, PASSWD);  
    } catch (Exception e) {
```

```
        logger.error("Init doris connection failed.", e);
        throw new Exception(e);
    }
    return connection;
}
```

In the preceding code snippet, HOST and PORT indicate the IP address of the master FE node and the MySQL query connection port for Doris, respectively.

- To obtain the IP address of the master FE node, log in to FusionInsight Manager, choose **Cluster > Services > Doris**, and check the **Host Where Leader Locates**.
- The default MySQL query connection port is 29982. To obtain the port number, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **query\_port**.

```
private static final String HOST = "192.168.67.78";
private static final int PORT = 29982;
```

## Connecting Doris Using DBalancer

The following code snippet uses the **createConnection** method on the **DBalancerJDBCExample** class in the **com.huawei/bigdata/doris/example** package.

```
private static Connection createConnection() throws Exception {
    Connection connection = null;
    try {
        Class.forName(JDBC_DRIVER);
        String dbUrl = String.format(DB_URL_PATTERN, HOST, PORT);
        connection = DriverManager.getConnection(dbUrl, USER, PASSWD);
    } catch (Exception e) {
        logger.error("Init doris connection failed.", e);
        throw new Exception(e);
    }
    return connection;
}
```

In the preceding code snippet, HOST and PORT indicate the IP address of the DBalancer node and DBalancer TCP access port set in the following example.

- To obtain the IP address of the DBalancer node, log in to FusionInsight Manager and choose **Cluster > Services > Doris > Instances**.
- The default TCP access port of DBalancer is 29992. To obtain the port number, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **balancer\_tcp\_port**.

```
private static final String HOST = "192.168.20.37";
private static final int PORT = 29992;
```

### 2.2.3.1.4 Creating a Database

Run the SQL statement in Java JDBC mode or Java DBalancer to create the database corresponding to the *dbName* variable in the cluster.

```
String createDatabaseSql = "create database if not exists dbName";
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

### 2.2.3.1.5 Creating a Table

Run the SQL statement in Java JDBC or Java DBalancer mode to create the table corresponding to *tableName* in the database corresponding to the *dbName* variable in the cluster.

```
String createTableSql = "create table if not exists " + dbName + "." + tableName + " (\n" +
    "c1 int not null,\n" +
    "c2 int not null,\n" +
    "c3 string not null\n" +
    ") engine=olap\n" +
    "unique key(c1, c2)\n" +
    "distributed by hash(c1) buckets 1";
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

### 2.2.3.1.6 Inserting Data

Run the following SQL statement in Java JDBC or Java DBalancer mode to insert data into the *dbName.tableName* table of the cluster.

```
String insertTableSql = "insert into " + dbName + "." + tableName + " values(?, ?, ?)";
private static void insert(Connection connection, String sql) throws Exception {
    int INSERT_BATCH_SIZE = 10;
    try(PreparedStatement stmt = connection.prepareStatement(sql)) {
        for (int i =0; i < INSERT_BATCH_SIZE; i++) {
            stmt.setInt(1, i);
            stmt.setInt(2, i * 10);
            stmt.setString(3, String.valueOf(i * 100));
            stmt.addBatch();
        }
        stmt.executeBatch();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

### 2.2.3.1.7 Querying Data

Use Java JDBC to run SQL statements to query data in the *dbName.tableName* table in the cluster.

```
String querySql = "select * from " + dbName + "." + tableName + " limit 10";
private static void query(Connection connection, String sql) throws Exception {
    try (Statement stmt = connection.createStatement());
        ResultSet resultSet = stmt.executeQuery(sql);
        ResultSetMetaData md = resultSet.getMetaData();
        int columnCount = md.getColumnCount();
        StringBuffer stringBuffer = new StringBuffer();
        logger.info("Start to print query result.");
        for (int i = 1; i <= columnCount; i++) {
            stringBuffer.append(md.getColumnName(i));
            stringBuffer.append(" ");
        }
        logger.info(stringBuffer.toString());

        while (resultSet.next()) {
            stringBuffer = new StringBuffer();
            for (int i = 1; i <= columnCount; i++) {
                stringBuffer.append(resultSet.getObject(i));
            }
        }
    }
}
```

```
        stringBuffer.append(" ");
    }
    logger.info(stringBuffer.toString());
}
} catch (Exception e) {
    logger.error("Execute sql {} failed.", sql, e);
    throw new Exception(e);
}
}
```

### 2.2.3.1.8 Deleting a table

Run the SQL statement in Java JDBC mode to delete the *dbName.tableName* table from the cluster.

```
String dropSql = "drop table " + dbName + "." + tableName;
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

### 2.2.3.1.9 Deleting a Database

Run the SQL statement using Java DBalancer to delete the *dbName* database from the cluster.

```
String dropDb = "drop database " + dbName;
public static void execDDL(Connection connection, String sql) throws Exception {
    try (PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.execute();
    } catch (Exception e) {
        logger.error("Execute sql {} failed.", sql, e);
        throw new Exception(e);
    }
}
```

## 2.2.3.2 Loading Data to a Doris Table with Stream Load

The Doris Stream Load sample program imports local CSV file data to a Doris table.

### Example Code

Modify the following parameters in the sample code based on the site requirements:

- **HOST**: IP address of the master FE node of Doris. To obtain the IP address of the master FE node, log in to FusionInsight Manager, choose **Cluster > Services > Doris**, and check **Host Where Leader Locates**.
- **PORT**: HTTPS port of the Doris FE service. The default value is **29980**. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > Doris**, click **Configurations**, and search for **http\_port**.
- **JDBC\_PORT**: The value is the MySQL protocol query connection port of the Doris. The default port is **29982**. To obtain the port, log in to FusionInsight Manager, choose **Cluster > Services > Doris > Configurations**, and search for **query\_port**.

- **DATABASE:** database of the table storing imported data
- **TABLE:** name of the Doris table that stores imported data
- **getHttpPost:** path of the CSV file to be imported

```
public class DorisStreamLoader {

    // FE IP Address
    private final static String HOST = "192.168.13.178";
    // If Kerberos authentication is enabled for the cluster (the cluster is in security mode), FE port is the value
    // of https_port; If Kerberos authentication is disabled for the cluster (the cluster is in normal mode), FE port
    // is the value of http_port.
    private final static int PORT = 29980;

    private final static int JDBC_PORT = 29982;
    // db name
    private final static String DATABASE = "test_2";
    // table name
    private final static String TABLE = "doris_test_sink";

    private static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DB_URL_PATTERN = "jdbc:mysql://%s:%d?rewriteBatchedStatements=true";

    private static final String USER = System.getenv("DORIS_MY_USER");
    private static final String PASSWD = System.getenv("DORIS_MY_PASSWORD");

    // If Kerberos authentication is enabled for the cluster (the cluster is in security mode), Start the url with
    // https; If Kerberos authentication is disabled for the cluster (the cluster is in normal mode), start the url
    // with http.
    private final static String loadUrl = String.format("http://%s/api/%s/%s/_stream_load",
            HOST, PORT, DATABASE, TABLE);

    // Call the Curl method.
    public static String execCurl(String[] cmd) {
        ProcessBuilder process = new ProcessBuilder(cmd);
        Process p;
        try {
            p = process.start();
            BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));
            StringBuilder builder = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
                builder.append(line);
                builder.append(System.getProperty("line.separator"));
            }
            return builder.toString();
        } catch (Exception e) {
            System.out.print("error");
        }
        return null;
    }

    public static void initTable(){
        String createDatabaseSql = "create database if not exists "+DATABASE;

        String createTableSql = "create table if not exists " + DATABASE + "." + TABLE + " (\n" +
            " `id` int NULL COMMENT '\"',\n" +
            " `number` int NULL COMMENT '\"',\n" +
            " `price` DECIMAL(12,2) NULL COMMENT '\"',\n" +
            " `skuname` varchar(40) NULL COMMENT '\"',\n" +
            " `skudesc` varchar(200) NULL COMMENT '\"'\n" +
            " ) ENGINE=OLAP\n" +
            " DUPLICATE KEY(`id')\n" +
            " COMMENT \"Offering information table\"\n" +
            " DISTRIBUTED BY HASH(`id') BUCKETS 1\n" +
            " PROPERTIES (\n" +
            " \"replication_num\" = \"3\",\\n" +
            " \"in_memory\" = \"false\",\\n" +

```

```
" \"storage_format\" = \"V2\"\n" +  
" );";  
try (Connection connection = createConnection()) {  
// Create a database.  
System.out.println("Start create database.");  
execDDL(connection, createDatabaseSql);  
System.out.println("Database created successfully.");  
// Create a table.  
System.out.println("Start create table.");  
execDDL(connection, createTableSql);  
System.out.println("Table created successfully.");  
} catch (Exception e) {  
System.out.println("Execute doris operation failed.");  
}  
}  
  
private static Connection createConnection() throws Exception {  
Connection connection = null;  
try {  
Class.forName(JDBC_DRIVER);  
String dbUrl = String.format(DB_URL_PATTERN, HOST, JDBC_PORT);  
connection = DriverManager.getConnection(dbUrl, USER, PASSWD);  
} catch (Exception e) {  
System.out.println("Init doris connection failed.");  
throw new Exception(e);  
}  
return connection;  
}  
  
public static void execDDL(Connection connection, String sql) throws Exception {  
try (PreparedStatement statement = connection.prepareStatement(sql)) {  
statement.execute();  
} catch (Exception e) {  
System.out.println("Execute sql {} failed.");  
throw new Exception(e);  
}  
}  
  
// Call the API.  
public static String getHttpPost(String csvPath) {  
  
String[] cmdList = {"curl", "-k", "--location-trusted", "-u" + USER + ":" + PASSWD, "-H", "expect:100-  
continue", "-H", "column_separator:", "-T",  
csvPath,  
loadUrl};  
  
String responseMsg = execCurl(cmdList);  
System.out.println("curl" + responseMsg);  
  
return responseMsg;  
}  
  
public static void main(String[] args) throws IOException {  
initTable();  
String path = DorisStreamLoader.class.getClassLoader().getResource("test.csv").getPath();  
path = URLDecoder.decode(path, "UTF-8");  
File file = new File(path);  
String filePath = file.getAbsolutePath();  
// In the Linux scenario, upload the test.csv file in the resource directory to the Linux background and  
specify the file path in getHttpPost.  
getHttpPost(filePath);  
}  
}
```

## 2.2.4 Application Commissioning

## 2.2.4.1 Commissioning an Application in Windows

### 2.2.4.1.1 Compiling and Running an Application

#### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

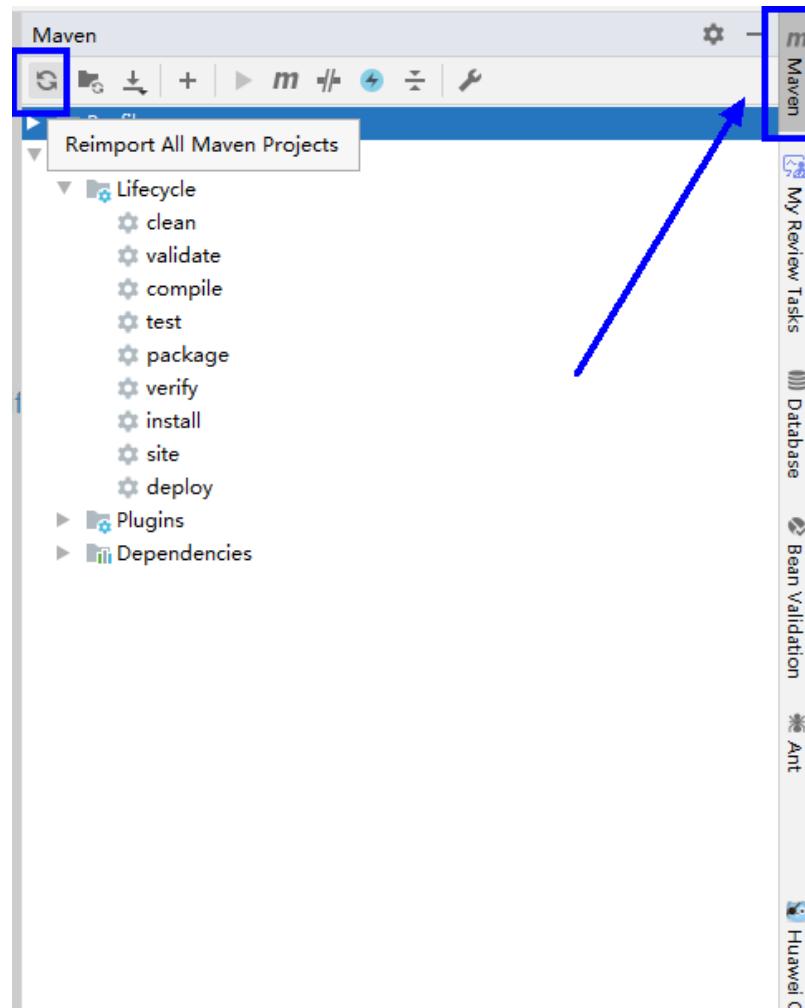
#### NOTE

- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

#### Procedure

**Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

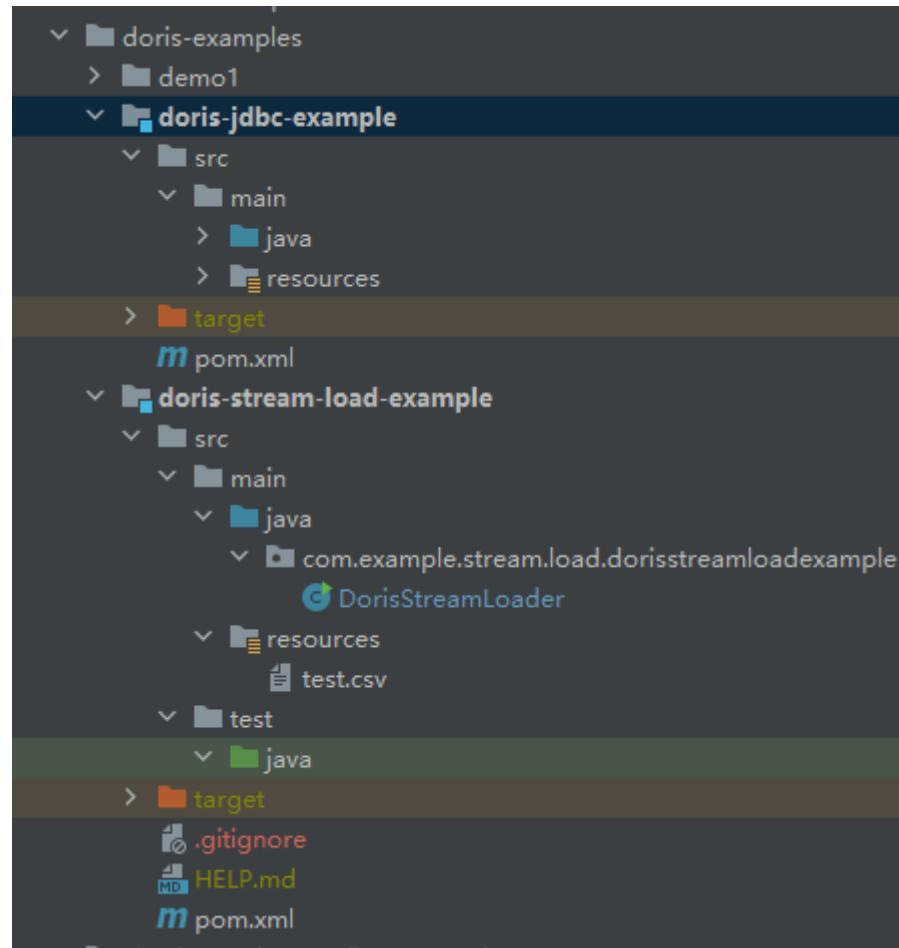
Figure 2-16 reimport projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user.  
See [Figure 2-17](#).

Figure 2-17 Directory list of Doris to be compiled



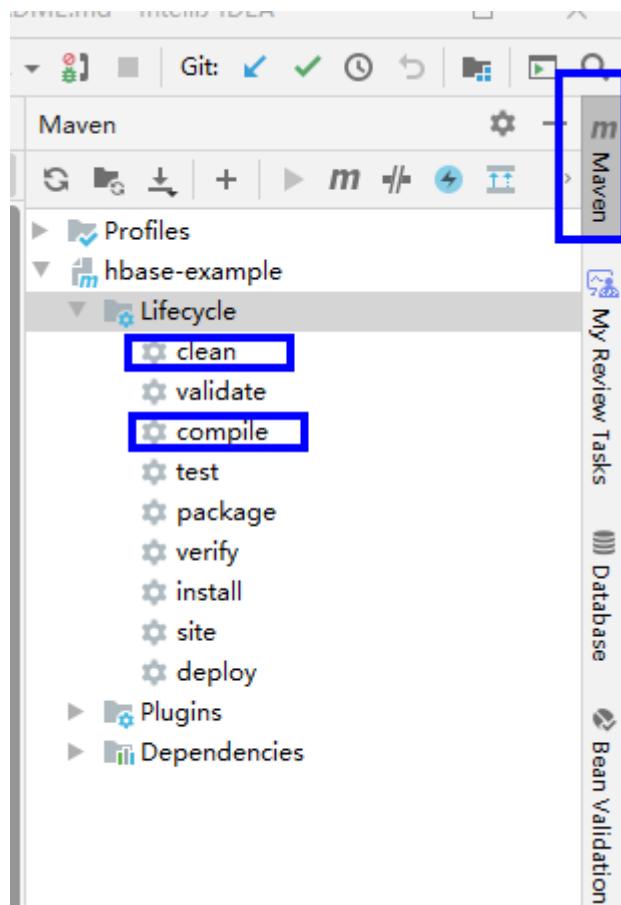
1. Compile an application.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

Figure 2-18 clean and compile tools of Maven



- Method 2:

Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

When the compilation is complete, the **Build Success** message is printed and the target directory is generated.

Figure 2-19 doris-jdbc-example compiled

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.241 s  
[INFO] Finished at: 2023-08-17T23:09:46+08:00  
[INFO] -----  
PS D:\gitSource\sample_project\src\doris-examples\doris-jdbc-example>
```

Figure 2-20 doris-stream-load-example compiled

```
[INFO] ----- [ jar ] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ doris-stream-load-example ---
[INFO] Deleting D:\doris-stream-load-example\target
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time:  0.190 s
[INFO] Finished at: 2024-06-12T11:26:09+08:00
[INFO] -----
```

 NOTE

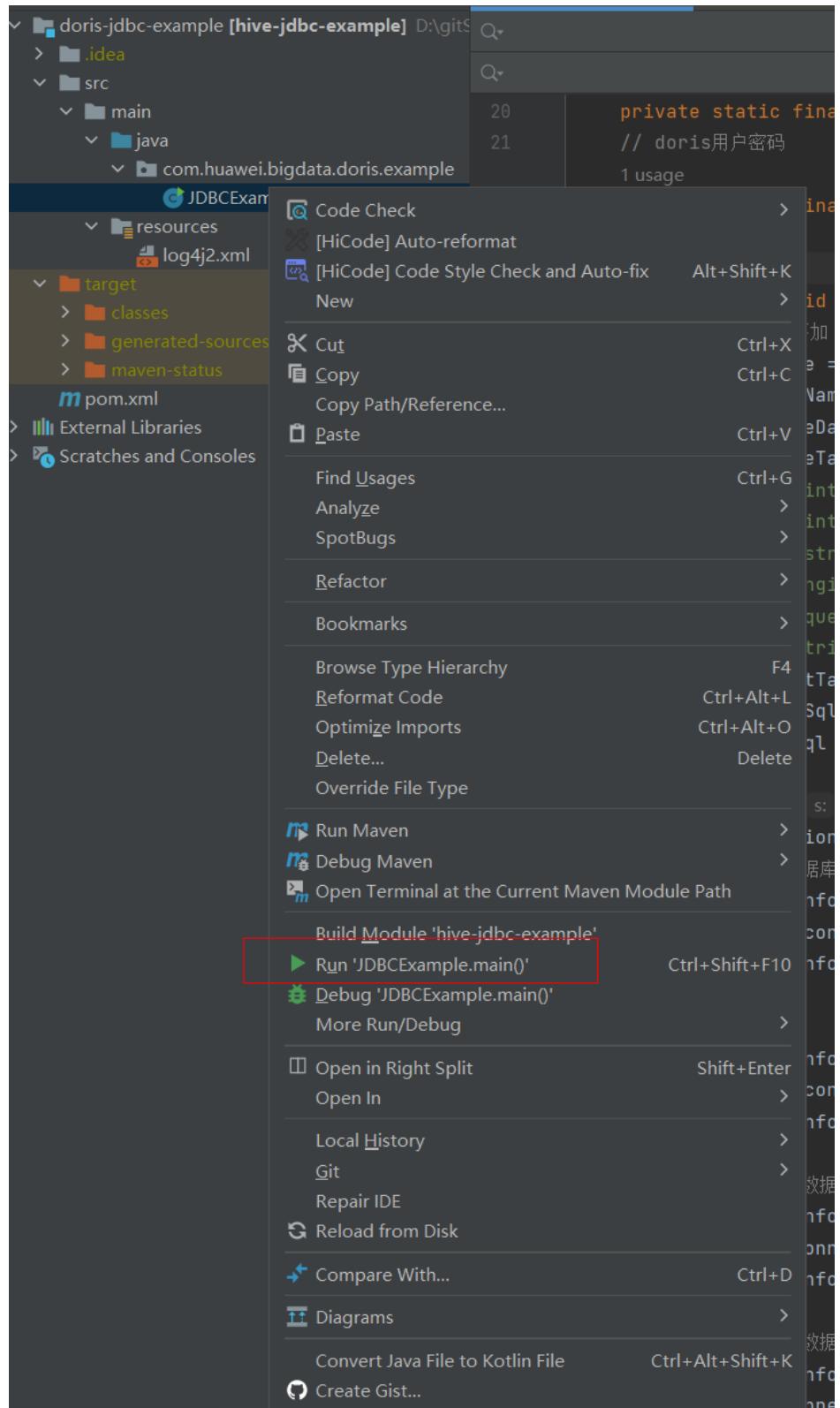
If the error message "Please use -source 7 or later to enable try-with-resources" is displayed during compilation, add the following content to the **properties** tag in the **pom.xml** configuration file:

```
<maven.compiler.source>8</maven.compiler.source>
<maven.compiler.target>8</maven.compiler.target>
```

2. Run the program.

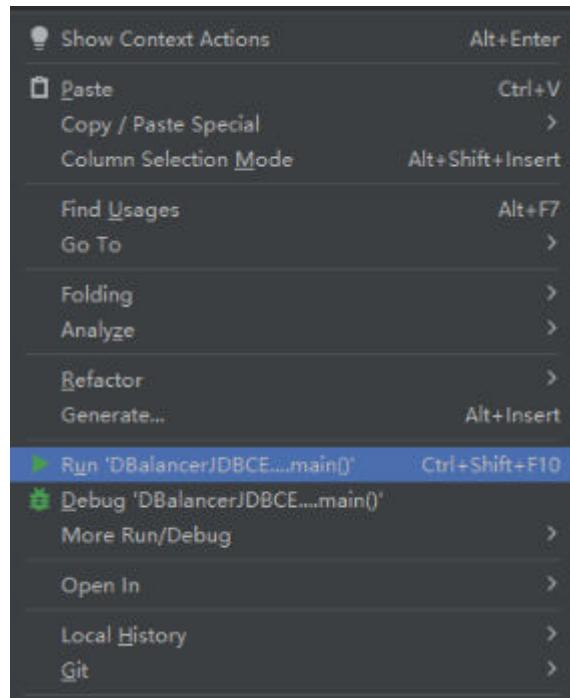
- Right-click the **JDBCExample.java** file and choose **Run 'JDBCExample.main()'**.

Figure 2-21 Run the Doris application.



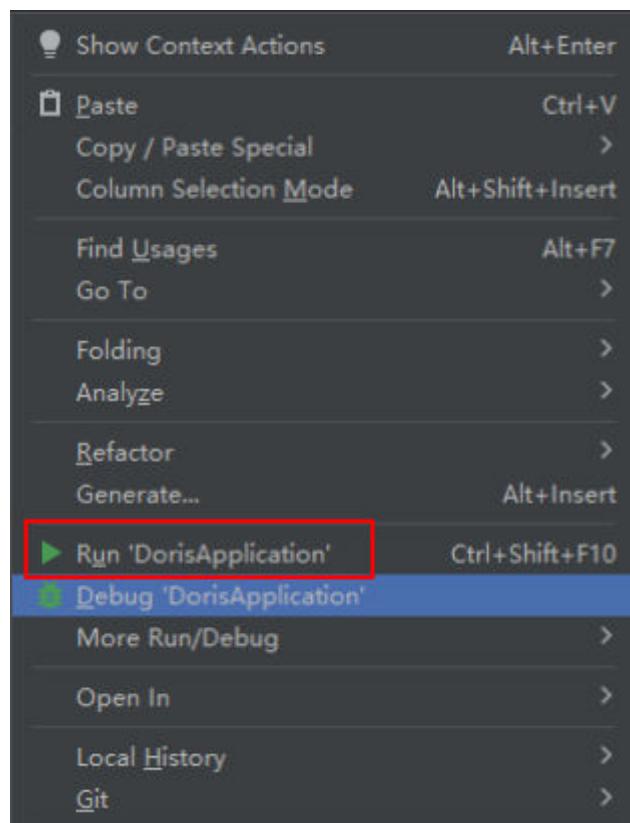
- Right-click the **DBalancerJDBCExample.java** file and choose **Run 'DBalancerJDBCExample.main()'**.

Figure 2-22 Run the DorisDBalancer application.



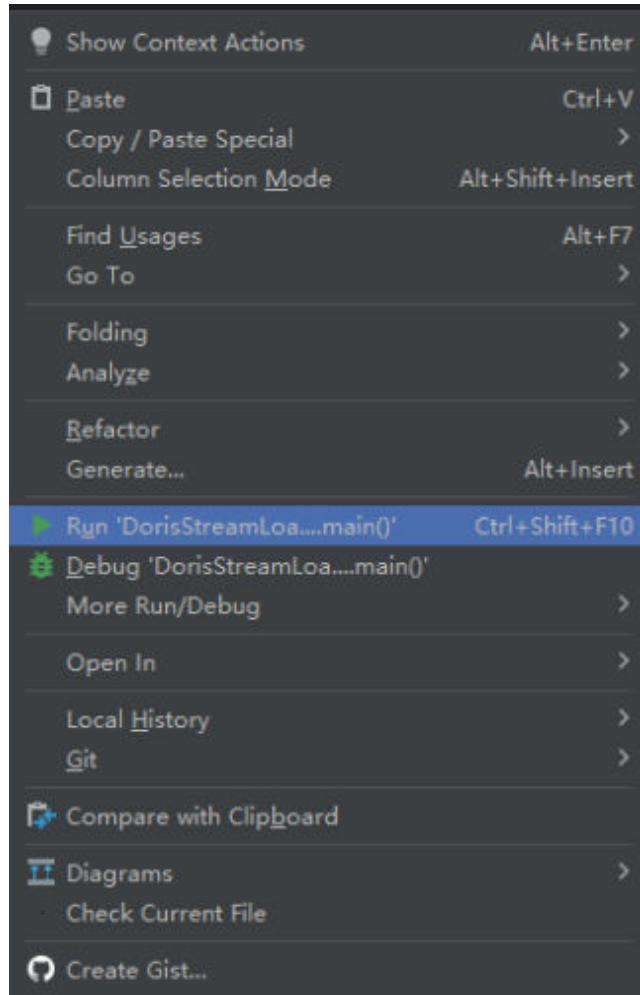
- Right-click the **DorisApplication.java** file and choose **Run 'DorisApplication'**.

Figure 2-23 Run the Doris SpringBoot application.



- Right-click the **DorisStreamLoader.java** file and choose **Run 'DorisStreamLoader.main()'**.

Figure 2-24 Run the Doris Stream Load application.



----End

#### 2.2.4.1.2 Viewing Windows Commissioning Results

##### Operation Scenario

After a Doris application finishes running, you can use one of the following methods to view the running result:

- Check the IntelliJ IDEA.
- Check the Doris log.

##### Procedure

- After the **JDBCExample.java** of **doris-jdbc-example** sample is successfully executed, the following information is displayed:

```
2023-08-17 23:13:13,473 | INFO | main | Start execute doris example. |  
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:41)  
2023-08-17 23:13:13,885 | INFO | main | Start create database. |
```

```
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:44)
2023-08-17 23:13:13,949 | INFO | main | Database created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:46)
2023-08-17 23:13:13,950 | INFO | main | Start create table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:49)
2023-08-17 23:13:14,132 | INFO | main | Table created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:51)
2023-08-17 23:13:14,133 | INFO | main | Start to insert data into the table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:54)
2023-08-17 23:13:14,733 | INFO | main | Inserting data to the table succeeded. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:56)
2023-08-17 23:13:14,733 | INFO | main | Start to query table data. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:59)
2023-08-17 23:13:15,079 | INFO | main | Start to print query result. |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:121)
2023-08-17 23:13:15,079 | INFO | main | c1 c2 c3 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:126)
2023-08-17 23:13:15,079 | INFO | main | 0 0 0 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 1 10 100 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 2 20 200 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 3 30 300 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 4 40 400 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 5 50 500 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 6 60 600 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 7 70 700 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 8 80 800 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 9 90 900 |
com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | Querying table data succeeded. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:61)
2023-08-17 23:13:15,081 | INFO | main | Start to delete the table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:64)
2023-08-17 23:13:15,114 | INFO | main | Table deleted successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:66)
2023-08-17 23:13:15,124 | INFO | main | Doris example execution successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:71)
```

Process finished with exit code 0

- After the **DBalancerJDBCExample.java** of **doris-jdbc-example** sample is successfully executed, the following information is displayed:

```
2024-06-12 14:22:08,716 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:131)
2024-06-12 14:22:08,716 | INFO | main | c1 c2 c3 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:136)
2024-06-12 14:22:08,717 | INFO | main | 0 0 0 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 1 10 100 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 2 20 200 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 3 30 300 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 4 40 400 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 5 50 500 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 6 60 600 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 7 70 700 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 8 80 800 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 9 90 900 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 0 0 0 | com.huawei.bigdata.doris.example.DBalancerJDBCExample.query(DBalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | Start to delete the table. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:66)
2024-06-12 14:22:08,718 | INFO | main | Table deleted successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:71)
2024-06-12 14:22:08,836 | INFO | main | Start to delete the database. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:74)
2024-06-12 14:22:08,961 | INFO | main | Database deleted successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:76)
2024-06-12 14:22:08,962 | INFO | main | Doris example execution successfully. | com.huawei.bigdata.doris.example.DBalancerJDBCExample.main(DBalancerJDBCExample.java:81)
```

- Running result of interconnection between Doris and SpringBoot
  - The result of running the Doris JDBC sample application using SpringBoot is as follows:

Enter **http://IP address of the node that runs the sample:8080/doris/example/executesql** in the address box of the browser. IDEA prints logs. The following figure shows the returned information.

**Figure 2-25** Doris JDBC returned running information



```
localhost:8080/doris/example/executesql
=====
Doris Example Start =====
Start create database. Database created successfully. Start create table. Table created successfully. Start to insert data into the table. Inserting data to the table succeeded. Start to query table data. Query result: c1 c2 c3 0 0 0 1 10 100 2.20 200 3 30 300 4 40 400 5 50 500 6 60 600 7 70 700 8 80 800 9 90 900 Querying table data succeeded. Start to delete the table. Table deleted successfully.
=====
Doris Example End =====
```

- The result of running the Doris DBalancer sample application using SpringBoot is as follows:

**Figure 2-26** Doris DBalancer returned running information



```
=====
Doris Example Start =====
10 100 2 20 200 3 30 300 4 40 400 5 50 500 6 60 600 7 70 700 8 80 800 9 90 900 Querying table data succeeded. Start to delete the table. Table deleted successfully. Start to delete the database. Database deleted successfully.
=====
Doris Example End =====
```

- If the **doris-stream-load-example** sample is successfully executed, the following information is displayed:

```
Start create database.
Database created successfully.
Start create table.
Table created successfully.
{
  "TxnId": 27,
  "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
  "Comment": "",
  "TwoPhaseCommit": "false",
  "Status": "Success",
  "Message": "OK",
  "NumberTotalRows": 4,
  "NumberLoadedRows": 4,
  "NumberFilteredRows": 0,
  "NumberUnselectedRows": 0,
  "LoadBytes": 141,
  "LoadTimeMs": 387,
  "BeginTxnTimeMs": 58,
  "StreamLoadPutTimeMs": 137,
  "ReadDataTimeMs": 0,
  "WriteDataTimeMs": 94,
  "CommitAndPublishTimeMs": 95
}
curl{
  "TxnId": 27,
  "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
  "Comment": "",
  "TwoPhaseCommit": "false",
  "Status": "Success",
  "Message": "OK",
  "NumberTotalRows": 4,
  "NumberLoadedRows": 4,
  "NumberFilteredRows": 0,
  "NumberUnselectedRows": 0,
  "LoadBytes": 141,
  "LoadTimeMs": 387,
  "BeginTxnTimeMs": 58,
  "StreamLoadPutTimeMs": 137,
  "ReadDataTimeMs": 0,
  "WriteDataTimeMs": 94,
  "CommitAndPublishTimeMs": 95
}
```

Connect to Doris on the MySQL client and run the following command to view the imported data:

```
select * from doris_test_sink;
```

```
mysql> select * from doris_test_sink;
+-----+-----+-----+-----+
| id   | number | price | skuname | skudesc      |
+-----+-----+-----+-----+
| 10001 |    12  | 13.30 | test1   | this is atest
| 10002 |   100  | 15.30 | test2   | this is atest
| 10003 |   102  | 16.30 | test3   | this is atest
| 10004 |   120  | 17.30 | test4   | this is atest
+-----+-----+-----+-----+
4 rows in set (0.02 sec)
```

## 2.2.4.2 Commissioning an Application in Linux

### 2.2.4.2.1 Compiling and Running Applications

#### Scenario

After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

#### Prerequisites

- You have installed a JDK in the Linux environment. The version of the JDK must be consistent with that of the JDK used by IntelliJ IDEA to export the JAR file.
- If the host where the Linux environment is deployed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the Linux environment is deployed. The host names and IP addresses must be mapped one on one.

## Compiling and Running JDBC, DBalancer and SpringBoot Programs

### Step 1 Export a JAR file.

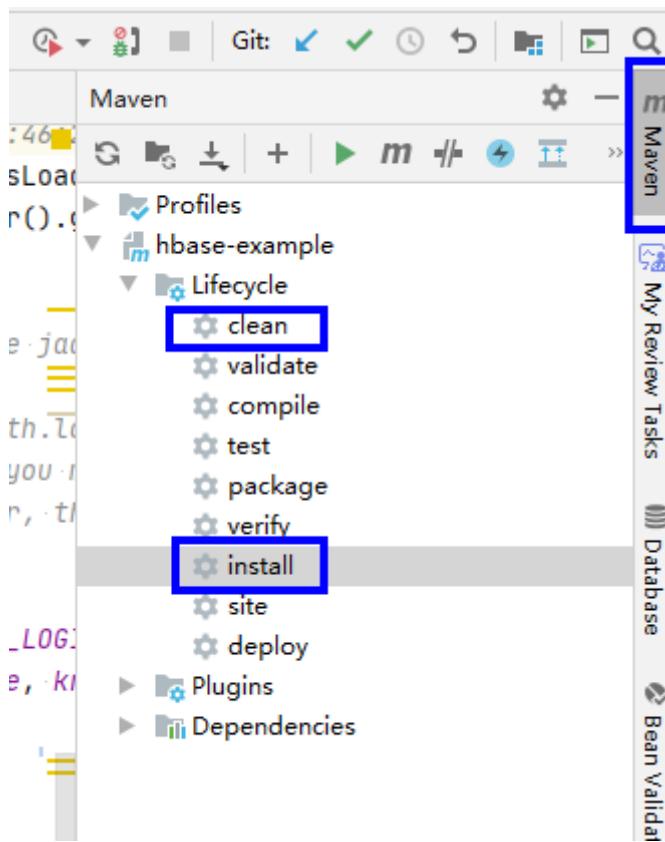
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 2-27 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

**Step 2** Decide whether to run Doris to connect to SpringBoots.

- If yes, perform the following steps to run the sample:
  - Create a running directory in the Linux environment and save the **doris-rest-client-example-\*jar** file in the **target** directory to this directory.
  - Switch to the running directory and run the following command to run the JAR package:  
**java -jar doris-rest-client-example-\*jar-with-dependencies.jar**
- If no, go to **Step 4**.

**Step 3** Export the JAR package on which the sample project depends.

Go to the directory where the **pom.xml** file is stored in the **Terminal** window at the bottom of the IDEA or in other command line tools. Run the following command:

**mvn dependency:copy-dependencies -DoutputDirectory=lib**

The **lib** folder is generated in the directory where the **pom.xml** file is stored. The **lib** folder contains the JAR packages on which the sample project depends.

**Step 4** Prepare the dependency JAR file and configuration file.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create the subdirectory **lib**. Export the JAR package on which the sample project depends, and upload the JAR package generated in **Step 1** and JAR package generated in **Step 3** to the **lib** directory of the Linux operating system.
2. In the **/opt/test** root directory, create the **run.sh** script, modify the following content, and save the file:

```
#!/bin/sh  
BASEDIR=`cd $(dirname $0);pwd`  
cd ${BASEDIR}  
for file in ${BASEDIR}/lib/*.jar  
do  
i_cp=$i_cp:$file  
echo "$file"  
done  
  
java -cp ${i_cp} com.huawei.bigdata.doris.example.JDBCExample
```

**com.huawei.bigdata.doris.example.JDBCExample** is used as an example. Replace it with the actual code.

**Step 5** Configure the environment variables **DORIS\_MY\_USER** and **DORIS\_MY\_PASSWORD**.

1. Add the following content to the **/etc/profile** file and save the file:  
*DORIS\_MY\_USER=Username for accessing Doris*  
*DORIS\_MY\_PASSWORD=Password for accessing Doris*
2. Run the following command to apply the environment variables:  
**source /etc/profile**

**Step 6** Go to **/opt/test** and run the following command to run the JAR file:

**sh run.sh**

----End

## Compiling and Running the Stream Load Program

- Step 1** Create a directory in the Linux environment, for example, **/opt/test**, and upload the **/src/main/resources/test.csv** file prepared in the **doris-stream-load-example** sample project obtained in **Step 1** to the directory.
- Step 2** Change the value of **getHttpPost** in the **doris-stream-load-example** sample code to **/opt/test/test.csv**.
- Step 3** Export a JAR file.

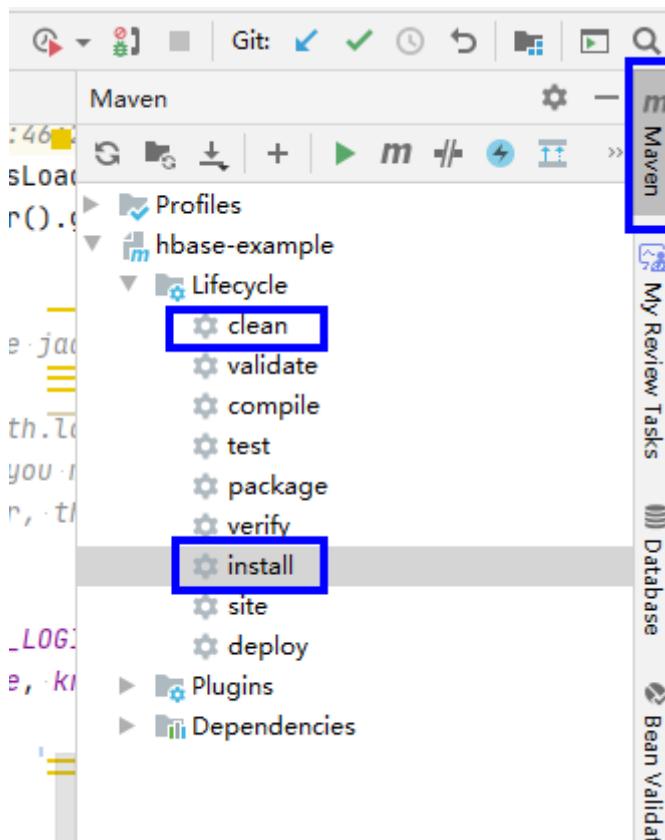
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 2-28 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

**Step 4** Place **doris-stream-load-example-\*jar** in the **target** directory to the directory created in **Step 1**, for example, **/opt/test**.

**Step 5** Switch to the JDK directory and run the JAR package.

```
java -jar doris-stream-load-example-*jar-with-dependencies.jar
```

----End

#### 2.2.4.2.2 Viewing Linux Commissioning Results

- After the **JDBCExample.java** of **doris-jdbc-example** sample is successfully executed, the following information is displayed:

```
2023-08-17 23:13:13,473 | INFO | main | Start execute doris example. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:41)
2023-08-17 23:13:13,885 | INFO | main | Start create database. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:44)
2023-08-17 23:13:13,949 | INFO | main | Database created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:46)
2023-08-17 23:13:13,950 | INFO | main | Start create table. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:49)
2023-08-17 23:13:14,132 | INFO | main | Table created successfully. |
com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:51)
```

```
2023-08-17 23:13:14,133 | INFO | main | Start to insert data into the table. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:54)
2023-08-17 23:13:14,733 | INFO | main | Inserting data to the table succeeded. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:56)
2023-08-17 23:13:14,733 | INFO | main | Start to query table data. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:59)
2023-08-17 23:13:15,079 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:121)
2023-08-17 23:13:15,079 | INFO | main | c1 c2 c3 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:126)
2023-08-17 23:13:15,079 | INFO | main | 0 0 0 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 1 10 100 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 2 20 200 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 3 30 300 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 4 40 400 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 5 50 500 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 6 60 600 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,080 | INFO | main | 7 70 700 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 8 80 800 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | 9 90 900 | com.huawei.bigdata.doris.example.JDBCExample.query(JDBCExample.java:134)
2023-08-17 23:13:15,081 | INFO | main | Querying table data succeeded. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:61)
2023-08-17 23:13:15,081 | INFO | main | Start to delete the table. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:64)
2023-08-17 23:13:15,114 | INFO | main | Table deleted successfully. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:66)
2023-08-17 23:13:15,124 | INFO | main | Doris example execution successfully. | com.huawei.bigdata.doris.example.JDBCExample.main(JDBCExample.java:71)
```

- After the `DBalancerJDBCExample.java` of `doris-jdbc-example` sample is successfully executed, the following information is displayed:

```
2024-06-12 14:22:08,716 | INFO | main | Start to print query result. | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:131)
2024-06-12 14:22:08,716 | INFO | main | c1 c2 c3 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:136)
2024-06-12 14:22:08,717 | INFO | main | 0 0 0 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:142)
2024-06-12 14:22:08,717 | INFO | main | 1 18 108 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 2 28 208 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 3 39 309 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 4 49 408 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,717 | INFO | main | 5 59 508 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 6 69 608 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 7 79 708 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 8 89 808 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | 9 99 908 | com.huawei.bigdata.doris.example.ObalancerJDBCExample.query(@BalancerJDBCExample.java:144)
2024-06-12 14:22:08,718 | INFO | main | Querying table data succeeded. | com.huawei.bigdata.doris.example.ObalancerJDBCExample.main(@BalancerJDBCExample.java:66)
2024-06-12 14:22:08,718 | INFO | main | Start to delete the table. | com.huawei.bigdata.doris.example.ObalancerJDBCExample.main(@BalancerJDBCExample.java:69)
2024-06-12 14:22:08,836 | INFO | main | Table deleted successfully. | com.huawei.bigdata.doris.example.ObalancerJDBCExample.main(@BalancerJDBCExample.java:77)
2024-06-12 14:22:08,836 | INFO | main | Start to delete the database. | com.huawei.bigdata.doris.example.ObalancerJDBCExample.main(@BalancerJDBCExample.java:76)
2024-06-12 14:22:08,961 | INFO | main | Database deleted successfully. | com.huawei.bigdata.doris.example.ObalancerJDBCExample.main(@BalancerJDBCExample.java:76)
2024-06-12 14:22:08,962 | INFO | main | Doris example execution successfully. | com.huawei.bigdata.doris.example.ObalancerJDBCExample.main(@BalancerJDBCExample.java:81)
```

- Running result of interconnection between Doris and SpringBoot
    - The result of running the Doris JDBC sample application using SpringBoot is as follows:  
Enter **http://IP address of the node that runs the sample:8080/doris/example/executesql** in the address box of the browser. IDEA prints logs. The following figure shows the returned information.

**Figure 2-29** Doris JDBC returned running information



- The result of running the Doris DBalancer sample application using SpringBoot is as follows:

**Figure 2-30** Doris DBalancer returned running information

- ```
***** Doris Example Start ***** Start create database. Database created successfully. Start create table. Table created successfully. Start to insert data into the table. Inserting data to the table succeeded. Start to query table data. Query result: c1 c2 c3 0 0 0
10 100 2 20 200 3 30 300 4 40 400 5 50 500 6 60 600 7 70 700 8 80 800 9 90 900 Querying table data succeeded. Start to delete the table. Table deleted successfully. start to delete the database. Database deleted successfully. ***** Doris Example End *****
```
- If the **doris-stream-load-example** sample is successfully executed, the following information is displayed:

```
Start create database.
Database created successfully.
Start create table.
Table created successfully.
{
    "TxnId": 27,
    "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 4,
    "NumberLoadedRows": 4,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 141,
    "LoadTimeMs": 387,
    "BeginTxnTimeMs": 58,
    "StreamLoadPutTimeMs": 137,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 94,
    "CommitAndPublishTimeMs": 95
}
curl{
    "TxnId": 27,
    "Label": "3661e112-f0e9-4aaa-80c3-4c2b7aefeebf",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 4,
    "NumberLoadedRows": 4,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 141,
    "LoadTimeMs": 387,
    "BeginTxnTimeMs": 58,
    "StreamLoadPutTimeMs": 137,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 94,
    "CommitAndPublishTimeMs": 95
}
```

Connect to Doris on the MySQL client and run the following command to view the imported data:

```
select * from doris_test_sink;
```

```
mysql> select * from doris_test_sink;
+-----+-----+-----+-----+
| id   | number | price | skuname | skudesc      |
+-----+-----+-----+-----+
| 10001 |    12 | 13.30 | test1   | this is atest
| 10002 |   100 | 15.30 | test2   | this is atest
| 10003 |   102 | 16.30 | test3   | this is atest
| 10004 |   120 | 17.30 | test4   | this is atest
+-----+-----+-----+-----+
4 rows in set (0.02 sec)
```

## 2.3 Elasticsearch Development Guide

### 2.3.1 Overview

#### 2.3.1.1 Application Development Overview

##### Introduction to Elasticsearch

Elasticsearch, a Lucene-powered search server, provides a distributed full-text search and analysis engine with multi-user capabilities. Designed for big data scenarios, Elasticsearch is easy to install and use and supports stable, fast, and reliable real-time search and analysis.

In addition to the basic functions, such as timestamp-based filtering and exact math provided by traditional relational databases, Elasticsearch can perform full-text search, process synonyms, score documents based on the correlation, and generate analysis and aggregation results based on the same data. Data can be processed in real time if a large number of working processes do not exist.

#### 2.3.1.2 Concepts

##### Basic Concepts

- **cluster**

There are multiple nodes in a cluster, one of which is the master node and the others are the slave nodes. Master node and slave node are defined within a cluster. In Elasticsearch, decentralization is posed. That is, there is no "central node" of the Elasticsearch cluster to external systems. Nodes in an Elasticsearch cluster can be logically considered as an entity. In this way, communication with any node in Elasticsearch is equivalent to the communication with the Elasticsearch cluster.

- **shards**

Indicates the index shard. Elasticsearch can divide a complete index into multiple shards. In this way, a large index can be split into multiple shards and distributed to different nodes, implementing distributed search. The number of shards can only be specified before the index is created and cannot be changed after the index is created.

- **replicas**

Indicates the index replica. Elasticsearch allows you to set multiple replicas for an index. Replication is to improve the fault tolerance of the system. When a shard of a node is damaged or lost, the system can restore it using its replica. In addition, replication is to improve the query efficiency of Elasticsearch. Elasticsearch automatically balances the search requests.

- **recovery**

Indicates data restoration or data redistribution. When a node is added or deleted, Elasticsearch redistributes index shards based on the load of the corresponding physical server. When a faulty node is restarted, data restoration is also performed.

- **river**

Indicates a data source of Elasticsearch. It is also a method of synchronizing data from other storage modes (such as the database) to Elasticsearch. **river** is a plug-in Elasticsearch service. The river plug-in reads data from a river data source and indexes it to Elasticsearch. Official rivers include those from CouchDB, RabbitMQ, Twitter, and Wikipedia.

- **gateway**

Indicates the storage mode of an Elasticsearch index snapshot. By default, Elasticsearch stores an index in the memory. When the memory is full, Elasticsearch persistently saves the index to the local hard disk. A gateway stores index snapshots. When the corresponding Elasticsearch cluster is stopped and then restarted, the index backup data is read from the gateway. Elasticsearch supports multiple types of gateways, including local file systems (default), distributed file systems, and Hadoop HDFS.

- **discovery.zen**

Indicates the automatic node discovery mechanism of Elasticsearch. Elasticsearch is a P2P system. It searches for existing nodes through broadcast and then communicates with nodes through multicast protocols. Elasticsearch supports P2P interaction.

- **Index**

Indicates the data storage mode of Elasticsearch and is similar to a table in relational database .

- **Document**

Is similar to the row of a relational database. Each document of the same index has a unique ID.

- **Field**

Is similar to the column in a relational database, which is the smallest unit of Elasticsearch data storage.

### 2.3.1.3 Application Development Process

[Figure 2-31](#) shows the Elasticsearch application development process.

Figure 2-31 Elasticsearch application development process

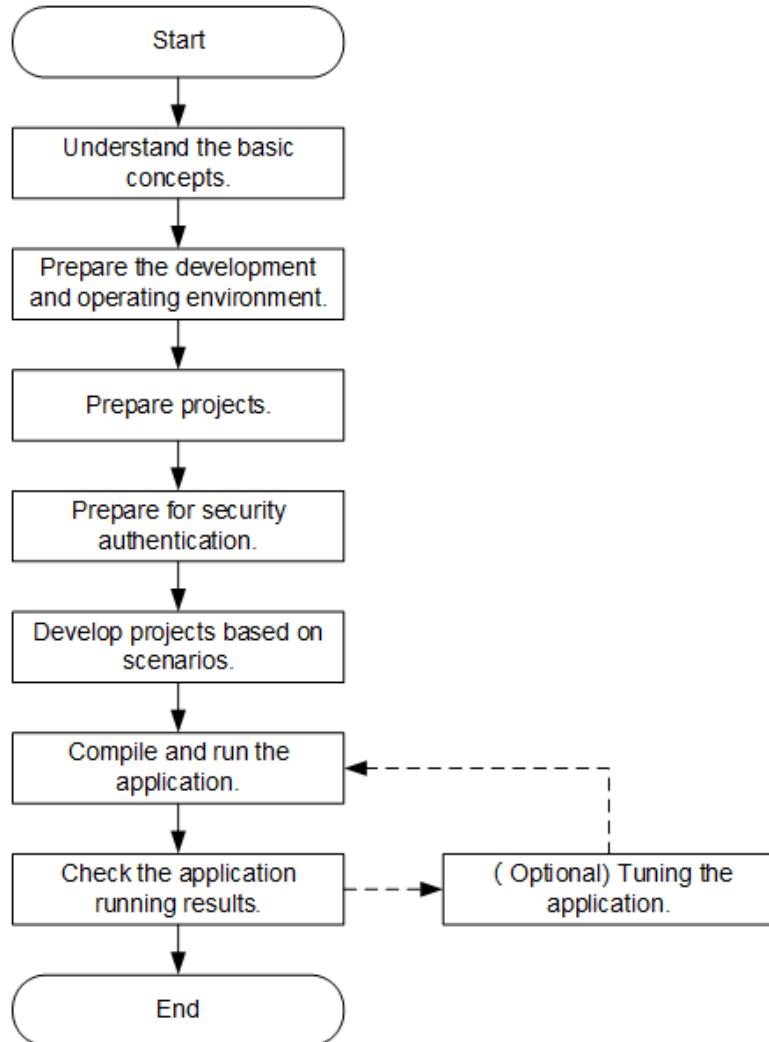


Table 2-8 Description of the Elasticsearch application development process

| Phase   | Description  | Reference   |
|---|--|---|
| Understand basic concepts.                        | Before application development, learn basic concepts of Elasticsearch and understand scenario requirements.  | <a href="#">Concepts</a>  |
| Development Environment and Operating Environment | Elasticsearch applications support RESTful development using Java. You are advised to use the IDEA tool to configure development environments. Elasticsearch runs on an Elasticsearch client. Install and configure the client according to the guide. | <a href="#">Preparing the Development Environment and Operating Environment</a> |

| Phase                                    | Description  | Reference   |
|--|--|---|
| Prepare a project.                       | Elasticsearch provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create an Elasticsearch project according to the guide. | <a href="#">Configuring and Importing Sample Projects</a> |
| Develop a project based on the scenario. | Provide a sample project using Java language, covering an entire process for sample projects from query index to deletion index.   | <a href="#">Development Process</a>                       |
| Compile and run applications .           | You can compile the developed application and submit it for running.   | <a href="#">Commissioning Process</a>                     |
| View application running results.        | Application running results are stored under a user-defined path. You can also view the application running status on the UI.  | <a href="#">Commissioning Process</a>                     |

## 2.3.2 Environment Preparation

### 2.3.2.1 Preparing the Development Environment and Operating Environment Preparing Development Environment

**Table 2-9** describes the development environment and operating environment required for application development.

**Table 2-9** Development environment

| Item | Description   |
|------|---|
| OS   | <ul style="list-style-type: none"><li>• Development environment: Windows (Windows 7 or later is supported.)</li><li>• Operating environment: Windows or Linux. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul> |

| Item                                     | Description   |
|--|---|
| Installing JDK                           | <p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul> |
| Installing and configuring IntelliJ IDEA | <p>Tool used for developing GraphBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li><li>• If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li><li>• If you use Open JDK, ensure that the JDK configured in IntelliJ IDEA is Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>  |
| Installing the JUnit plug-in             | Basic configurations of the development environment.  |

| Item                  | Description  |
|-----------------------|--|
| Installation of Maven | Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development. |
| 7-zip                 | It is a tool used to decompress *.zip and *.rar files. The 7-Zip 16.04 is supported.   |

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) and choose **Homepage > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Elasticsearch\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the Elasticsearch sample project.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

- a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
- b. **Accessing FusionInsight Manager of an MRS Cluster.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig \Elasticsearch\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:  

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Elasticsearch/config/* root@/P address of the client node:/opt/hadoopclient/conf
```
- c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

### 2.3.2.2 Configuring and Importing Sample Projects

#### 2.3.2.2.1 Overview

##### Background

Obtain the Elasticsearch development sample project. Import the project to IntelliJ IDEA for learning.

##### Prerequisites

Ensure that the time difference between the local PC and the cluster is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. Time of the cluster can be viewed in the lower-right corner on FusionInsight Manager.

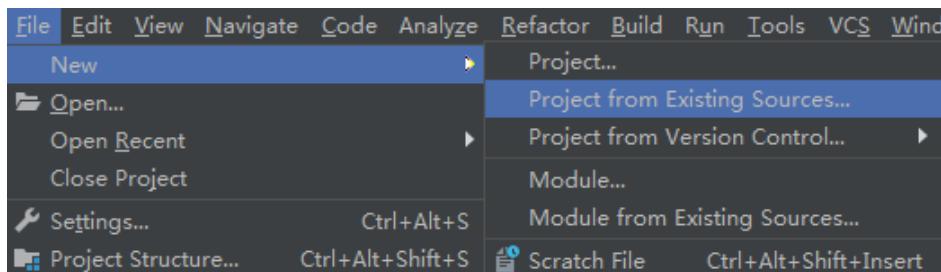
### 2.3.2.2 Importing a RestClient Sample Project

#### Procedure

**Step 1** Obtain the sample project folder **elasticsearch-rest-client-example** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Choose **File > New > Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **elasticsearch-rest-client-example** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

**Step 3** Find the configuration file **es-rest-client-example.properties** exists in the client folder **Elasticsearch\config** directory. Copy the parameter values in the file to the **esParams.properties** file in the **conf** directory of the sample project.

**Step 4** In the IntelliJ IDEA development environment, open the **esParams.properties** file in the **conf** directory of the sample project, and change the values of the parameters listed in [Table 2-10](#) as required.

**Table 2-10** Configuration table

| Parameter      | Default Value                                     | Description   |
|----------------|---|---|
| esServerHost   | Examples:<br>ip1:port1,ip2:port<br>2,ip3:port3... | Specifies the list of combinations of the IP addresses of nodes in the Elasticsearch cluster and the HTTP ports of the Elasticsearch instances installed on the nodes, except EsMaster instances. |
| connectTimeout | 5000  | Specifies the timeout interval of the connection between the client and the server, in milliseconds.  |
| socketTimeout  | 60,000  | Specifies the server response obtaining timeout interval of the client, in milliseconds.  |

| Parameter                | Default Value | Description   |
|--------------------------|---------------|---|
| connectionRequestTimeout | 100000        | Indicates that the connection timeout interval is obtained from the connection pool, in milliseconds.   |
| maxConnPerRoute          | 100           | Maximum number of connections per route.  |
| maxConnTotal             | 1000          | Maximum number of connections   |
| isSecureMode             | false         | Indicates whether the client is in security mode. The value <b>true</b> indicates that the security mode is enabled, and the value <b>false</b> indicates that the normal mode is enabled.  |
| sslEnabled               | false         | Indicates whether SSL encryption is enabled on the client. The value <b>true</b> indicates that SSL encryption is enabled, and the value <b>false</b> indicates that SSL encryption is disabled. The configuration takes effect only in security mode. In common mode, set this parameter to false. |
| principal                | esuser        | Specifies the authentication subject. The recommended format is "username". You can also use the format of "username@System domain name".   |
| snifferEnable            | true          | Indicates whether the Rest client enables the sniffing function. The value <b>true</b> indicates that the sniffing function is enabled, and the value <b>false</b> indicates that the sniffing function is disabled.  |
| customJaasPath           | null          | User-defined path of the <b>jaas.conf</b> file, which must contain specific file name. If this parameter is not set, the system automatically generates a path.   |

 NOTE

The **principal** and **customJaasPath** parameter is mandatory in security mode and can be ignored in normal mode.

**Step 5** Set **isSecureMode** in the **esParams.properties** file in the **conf** directory of the sample project to **false**, and enable the normal mode.

 NOTE

- If an error is reported during sample code importing, use a later IntelliJ IDEA version.
- The HTTP request function of the EsMaster instance is disabled. The value of the esServerHost parameter does not contain the EsMaster instance, that is, the 24148 port. Otherwise, the request fails.
- Set **esServerHosts** to the list of combinations. The combinations contain the IP addresses of all nodes in the Elasticsearch cluster and the HTTP ports of all Elasticsearch instances installed on the nodes, for example, ip1: Port1, ip2:port2, ip3:port3. You can query the port number as follows: On Manager, choose **Cluster > Name of the desired cluster > Service > Elasticsearch > Configuration > All Configurations**. Click **Port** of the corresponding instance and check the value of **SERVICE\_PORT**.
- To check whether the Elasticsearch cluster is in security mode or normal mode, log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Service > Elasticsearch > Configuration > All Configurations**, and search for **ELASTICSEARCH\_SECURITY\_ENABLE** parameter. If this parameter can be queried and its value is **true**, the Elasticsearch cluster is in security mode. If this parameter is not found or its value is **false**, the Elasticsearch cluster is in normal mode.

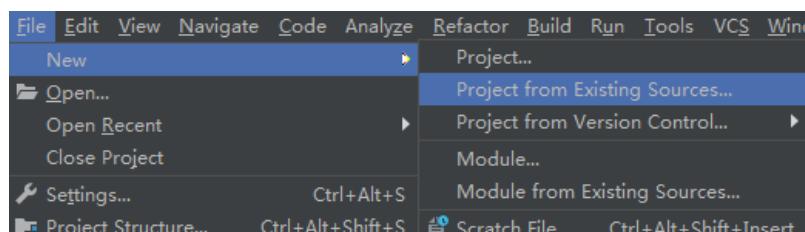
----End

### 2.3.2.2.3 Importing a SpringBoot Sample Project

#### Procedure

- Step 1** Obtain the sample project folder **elasticsearch-rest-client-example** in the **src/springboot/elasticsearch-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the sample project to the IntelliJ IDEA development environment in the application development environment.

1. Choose **File > New > Project from Existing Sources....**



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **elasticsearch-rest-client-example** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

- Step 3** Find configuration file **es-rest-client-example.properties** in the **Elasticsearch \config** directory of the client folder. Copy the parameter values in the file to the **esParams.properties** file in the **conf** directory of the sample project.
- Step 4** In the IntelliJ IDEA development environment, open the **esParams.properties** file in the **conf** directory of the sample project, and change the values of the parameters listed in the following table.

**Table 2-11** Configuration table

| Name                     | Default Value                    | Description  |
|--------------------------|----------------------------------|--|
| esServerHost             | ip1:port1,ip2:port2,ip3:port3... | List of combinations of the IP addresses of nodes in the Elasticsearch cluster and the HTTP ports of the Elasticsearch instances installed on the nodes, except EsMaster instances.  |
| connectTimeout           | 5000                             | Timeout interval of the connection between the client and the server, in milliseconds.   |
| socketTimeout            | 60000                            | Server response obtaining timeout interval of the client, in milliseconds.   |
| connectionRequestTimeout | 100000                           | Connection timeout interval is obtained from the connection pool, in milliseconds.   |
| maxConnPerRoute          | 100                              | Maximum number of connections on each route  |
| maxConnTotal             | 1000                             | Maximum number of connections  |
| isSecureMode             | false                            | Whether the client is in security mode. The value <b>true</b> indicates that the security mode is enabled, and the value <b>false</b> indicates that the normal mode is enabled.   |
| sslEnabled               | false                            | Whether to enable SSL encryption on the client. <b>true</b> indicates that encryption is enabled, and <b>false</b> indicates that encryption is disabled. This configuration takes effect only in security mode. For normal clusters, set this parameter to <b>false</b> . |
| principal                | esuser                           | Authentication subject. The options are as follows: <ul style="list-style-type: none"><li>• <b>Username</b> (recommended)</li><li>• <i>Username@System domain name</i></li></ul>   |
| snifferEnable            | true                             | Whether the REST Client enables the sniffing function. The value <b>true</b> indicates that the sniffing function is enabled, and the value <b>false</b> indicates that the sniffing function is disabled.   |
| customJaasPath           | N/A                              | User-defined path of the <b>jaas.conf</b> file, which must contain specific file name. If this parameter is not set, the system automatically generates a path.  |

 NOTE

The **principal** and **customJaasPath** parameters are mandatory in security mode and can be ignored in normal mode.

**Step 5** Set **isSecureMode** in the **esParams.properties** file in the **conf** directory of the sample project to **false** and enable the normal mode.

 NOTE

- If an error is reported during sample code importing, use the latest Eclipse version.
- The HTTP request function of the EsMaster instance is disabled. The value of the **esServerHost** parameter cannot contain the EsMaster instance, that is, the 24148 port. Otherwise, the request fails.
- Set **esServerHost** to a list of combinations of IP addresses of nodes in the installed Elasticsearch cluster and HTTP ports of Elasticsearch instances installed on the nodes, for example, *ip1:port1,ip2:port2,ip3:port3...*. To query the port number, choose **Cluster > Service > Elasticsearch** on FusionInsight Manager, click **Configuration**, click **All Configurations**, click **Port** of the corresponding instance, and check the value of **SERVER\_PORT**.
- To check whether the current Elasticsearch cluster is in security mode or common mode, log in to FusionInsight Manager, choose **Cluster**, click the name of the desired cluster, and choose **Services > Elasticsearch**. Click **Configuration**, click **All Configurations**, and search for the **ELASTICSEARCH\_SECURITY\_ENABLE** parameter. If the parameter can be queried and its value is **true**, the Elasticsearch cluster is in security mode. If this parameter cannot be queried or the value of this parameter is **false**, the Elasticsearch cluster is in normal mode.

----End

## 2.3.3 Development Process

### 2.3.3.1 Typical Application Scenarios

You can quickly learn and master the Elasticsearch development process and know key API functions in a typical application scenario.

#### Scenarios

Assume that you want to develop an application to search for all library information, provide information about books related to search keywords, and grade the books by score. The search function can be implemented by Elasticsearch. The search process is as follows:

1. Connecting the client to a cluster
2. Querying the health status
3. Checking whether a specified index exists
4. Creating an index for specifying the number of shards
5. Writing index data
6. Writing data in batches
7. Batch write data to specified routes
8. Querying index information
9. Deleting an index

10. Deleting documents in an index
11. Refreshing an index
12. Using a multi-thread sample for query
13. Pre-sorting indices

### 2.3.3.2 Low Level RestClient Sample Java Code

#### 2.3.3.2.1 Connecting the Elasticsearch Client to an Elasticsearch Cluster

##### Function Description

Before using APIs provided by Elasticsearch, you need to obtain the Elasticsearch client. Connect the Elasticsearch client to the desired Elasticsearch cluster by setting the IP address and port number of the cluster.

###### NOTE

Call **RestClient.close()** to close requested resources after the Elasticsearch operation is complete.

```
public static void main(String[] args) {  
    HwRestClient hwRestClient = new HwRestClient();  
    RestClient restClient = hwRestClient.getRestClient();  
    try {  
        /.../  
    } catch (Exception e) {  
        LOG.error("There are exceptions occurred.", e);  
    } finally {  
        if (restClient != null) {  
            try {  
                restClient.close();  
                LOG.info("Close the client successful.");  
            } catch (Exception e1) {  
                LOG.error("Close the client failed.", e1);  
            }  
        }  
    }  
}
```

###### NOTE

By default, the HwRestClient reads the following configuration files from the **conf** directory in the code running path: **esParams.properties**, **krb5.conf**, and **user.keytab**.

The configuration file path can be customized. For example, if the code runs in the Windows operating system, and the configuration files are stored in the root directory of drive D, you can set **HwRestClient hwRestClient** to **new HwRestClient("D:\\")**.

#### 2.3.3.2.2 Querying the Health Status of an Elasticsearch Cluster

##### Function Description

**QueryClusterInfo.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/cluster** directory is used to query the health status of the Elasticsearch cluster in information about the Elasticsearch cluster.

```
/**  
 * Query the cluster's information  
 */
```

```
public static void queryClusterInfo(RestClient restClientTest) {  
  
    Response response;  
    try {  
        Request request=new Request("GET", "/_cluster/health");  
        //Process the returned result to display it more intuitively  
        request.addParameter("pretty", "true");  
        response = restClientTest.performRequest(request);  
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {  
            LOG.info("QueryClusterInfo successful.");  
        } else {  
            LOG.error("QueryClusterInfo failed.");  
        }  
        LOG.info("QueryClusterInfo response entity is : " + EntityUtils.toString(response.getEntity()));  
    } catch (Exception e) {  
        LOG.error("QueryClusterInfo failed, exception occurred.", e);  
    }  
}
```

### 2.3.3.2.3 Checking Whether the Specified Index Exists

#### Function

**IndexIsExist.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/exist** directory is used to check whether a specified index exists.

```
/**  
 * Check the existence of the index or not.  
 */  
public static boolean exist(RestClient restClientTest, String index) {  
  
    Response response;  
    try {  
        Request request = new Request("HEAD", "/" + index);  
        request.addParameter("pretty", "true");  
        response = restClientTest.performRequest(request);  
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {  
            LOG.info("Check index successful,index is exist : {}.", index);  
  
            return true;  
        }  
        if (HttpStatus.SC_NOT_FOUND == response.getStatusLine().getStatusCode()) {  
            LOG.info("Index is not exist : {}.", index);  
  
            return false;  
        }  
    } catch (Exception e) {  
        LOG.error("Check index failed, exception occurred.", e);  
    }  
  
    return false;  
}
```

### 2.3.3.2.4 Creating an Index with the Desired Number of Shards

#### Function Description

**CreateIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/index** directory is used to create an index of numbers of specified primary shards and replica shards. Set

variables *shardNum* and *replicaNum* in the following code, that is, the number of primary shards and the number of replica shards.

```
/*
 * Create one index with customized shard number and replica number.
 */
public static void createIndexWithShardNum(RestClient restClientTest, String index) {

    Response rsp;

    int shardNum = 3;
    int replicaNum = 1;
    String jsonString =
        "{" + "\"settings\":{\"number_of_shards\":\"" + shardNum + "\",\"number_of_replicas\":\"" +
        replicaNum + "\"}}";

    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
    try {
        Request request = new Request("PUT", "/" + index);
        request.addParameter("pretty", "true");
        request.setEntity(entity);
        rsp = restClientTest.performRequest(request);
        if (HttpStatus.SC_OK == rsp.getStatusLine().getStatusCode()) {
            LOG.info("CreateIndexWithShardNum successful.");
        } else {
            LOG.error("CreateIndexWithShardNum failed.");
        }
        LOG.info("CreateIndexWithShardNum response entity is : {}.", EntityUtils.toString(rsp.getEntity()));
    } catch (Exception e) {
        LOG.error("CreateIndexWithShardNum failed, exception occurred.", e);
    }
}
```

### 2.3.3.2.5 Writing Index Data

#### Function Description

**PutData.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/index** directory is used to insert data into a specified index.

The variable *jsonString* is data to be inserted and can be customized.

```
/*
 * Write one document into the index
 */
public static void putData(RestClient restClientTest, String index, String type, String id) {

    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    esMap.put("name", "Happy");
    esMap.put("author", "Alex Yang");
    esMap.put("pubinfo", "Beijing,China");
    esMap.put("pubtime", "2020-01-01");
    esMap.put("description", "Elasticsearch is a highly scalable open-source full-text search and analytics engine.");
    String jsonString = gson.toJson(esMap);

    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
    Response rsp;

    try {
        Request request = new Request("POST", "/" + index + "/" + type + "/" + id);
        request.addParameter("pretty", "true");
    }
```

```
request.setEntity(entity);
rsp = restClientTest.performRequest(request);
if (HttpStatus.SC_OK == rsp.getStatusLine().getStatusCode() || HttpStatus.SC_CREATED ==
rsp.getStatusLine().getStatusCode()) {
    LOG.info("PutData successful.");
} else {
    LOG.error("PutData failed.");
}
LOG.info("PutData response entity is : {}.", EntityUtils.toString(rsp.getEntity()));
} catch (Exception e) {
    LOG.error("PutData failed, exception occurred.", e);
}
}
```

### 2.3.3.2.6 Writing Data in Batches

#### Function

**Bulk.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/bulk** directory is used to insert data into a specified index in batches.

**totalRecordNumber** indicates the total number of documents to be inserted, and **oneCommit** indicates the number of documents to be inserted at a time. Data added to **esMap** is the data to be written into the index.

```
/*
 * Send a bulk request
 */
private static void bulk(RestClient restClientTest, String index) {

    //Total number of documents to be written
    long totalRecordNum = 10000;
    //Number of documents written in a bulk
    long oneCommit = 1000;
    long circleNumber = totalRecordNum / oneCommit;
    StringEntity entity;
    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    String str = "{ \"index\" : { \"_index\" : \"\" + index + "\" } }";
    for (int i = 1; i <= circleNumber; i++) {
        StringBuilder builder = new StringBuilder();
        for (int j = 1; j <= oneCommit; j++) {
            esMap.clear();
            esMap.put("name", "Linda");
            esMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
            esMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
            esMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
            esMap.put("cur_time", System.currentTimeMillis());
            String strJson = gson.toJson(esMap);
            builder.append(str).append("\n");
            builder.append(strJson).append("\n");
        }
        entity = new StringEntity(builder.toString(), ContentType.APPLICATION_JSON);
        entity.setContentEncoding("UTF-8");
        Response response;
        try {
            Request request = new Request("PUT", "/_bulk");
            request.addParameter("pretty", "true");
            request.setEntity(entity);
            response = restClientTest.performRequest(request);
            if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
                LOG.info("Already input documents : " + oneCommit * i);
            } else {

```

```
        LOG.error("Bulk failed.");
    }
    LOG.info("Bulk response entity is : " + EntityUtils.toString(response.getEntity()));
} catch (Exception e) {
    LOG.error("Bulk failed, exception occurred.", e);
}
}
```

### 2.3.3.2.7 Batch write data to specified routes

## Function

**BulkRoutingSample.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/bulk` directory is used to set routing for each request in the bulk, route documents with the same routing to the same segment, and specify routing during query to narrow the query range and improve the query speed.

```
/*
 * Number of documents to be written.
 */
private static final int NUM_OF_DOC = 100;

/**
 * Use bulk to write data and specify routing.
 */
private static void bulkWithRouting(RestClient restClient, String indexName) {
    StringEntity entity;
    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    String str = "{ \"index\" : { \"_index\" : \"" + indexName + "\",";
    StringBuilder sb = new StringBuilder();
    int age;
    for (int i = 1; i <= NUM_OF_DOC; i++) {
        esMap.clear();
        esMap.put("user", "Linda" + i);
        age = ThreadLocalRandom.current().nextInt(18, 100);
        esMap.put("age", age);
        esMap.put("postDate", "2020-01-01");
        esMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
        esMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
        String strJson = gson.toJson(esMap);
        // Route documents of the same age to the same shard based on the age.
        sb.append(str).append("\"routing\":\"").append(age).append("\")").append("\n");
        sb.append(strJson).append("\n");
    }
    entity = new StringEntity(sb.toString(), ContentType.APPLICATION_JSON);
    entity.setContentType("UTF-8");
    Response response;
    try {
        Request request = new Request("PUT", "_bulk");
        request.addParameter("pretty", "true");
        request.setEntity(entity);
        response = restClient.performRequest(request);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("Bulk routing is successful.");
        } else {
            LOG.error("Bulk routing is failed.");
        }
    } catch (IOException e) {
        LOG.error("Bulk routing is failed, exception occurred.", e);
    }
}
```

### 2.3.3.2.8 Querying Index Information

#### Function Description

**QueryData.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/search** directory is used to query document information under specified **index**, **type**, and **id**.

```
/*
 * Query all data of one index.
 */
private static void queryData(RestClient restClientTest, String index, String type, String id){

    Response response;
    try {
        Request request = new Request("GET", "/" + index + "/" + type + "/" + id);
        request.addParameter("pretty", "true");
        response = restClientTest.performRequest(request);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("QueryData successful.");
        } else {
            LOG.error("QueryData failed.");
        }
        LOG.info("QueryData response entity is : {}.", EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
        LOG.error("QueryData failed, exception occurred.", e);
    }
}
```

### 2.3.3.2.9 Deleting an Index

#### Function Description

**DeleteIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/delete** directory is used to delete a specified index.

```
/*
 * Delete one index
 */
public static void deleteIndex(RestClient restClientTest, String index) {

    Response response;
    try {
        Request request = new Request("DELETE", "/" + index + "?&pretty=true");
        response = restClientTest.performRequest(request);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("DeleteIndex successful.");
        } else {
            LOG.error("DeleteIndex failed.");
        }
        LOG.info("DeleteIndex response entity is : {}.", EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
        LOG.error("DeleteIndex failed, exception occurred.", e);
    }
}
```

### 2.3.3.2.10 Deleting Some Documents in the Index

#### Function

**DeleteSomeDocumentsInIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/delete** directory is used to refresh a specified index.

```
/*
 * Delete some documents by query in one index
 */
public static void deleteSomeDocumentsInIndex(RestClient restClientTest, String index, String field, String value) {
    String jsonString =
        "\n" + "\"query\": {" + "\n" + " \\"match\\": { \"\\":\"" + field + "\"\\":\"" + value + "\"}\\n" + " }\\n" + "}";
    Map<String, String> params = Collections.singletonMap("pretty", "true");
    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
    Response rsp;
    try {
        rsp = restClientTest.performRequest("POST", "/" + index + "/_delete_by_query", params, entity);
        if (HttpStatus.SC_OK == rsp.getStatusLine().getStatusCode()) {
            LOG.info("DeleteSomeDocumentsInIndex successful.");
        }
        else {
            LOG.error("DeleteSomeDocumentsInIndex failed.");
        }
        LOG.info("DeleteSomeDocumentsInIndex response entity is : " +
EntityUtils.toString(rsp.getEntity()));
    }
    catch (Exception e) {
        LOG.error("DeleteSomeDocumentsInIndex failed, exception occurred.", e);
    }
}
```

### 2.3.3.2.11 Deleting All Documents in the Index

#### Function

**DeleteAllDocumentsInIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/delete** directory is used to refresh a specified index.

```
/*
 * Delete all documents by query in one index
 */
public static void deleteAllDocumentsInIndex(RestClient restClientTest, String index) {

    String jsonString = "\n" + "\"query\": {" + "\n" + " \\"match_all\\": {}\\n" + " }\\n" + "}";

    HttpEntity entity = new NStringEntity(jsonString, ContentType.APPLICATION_JSON);
    Response response;
    try {
        response = restClientTest.performRequest("POST", "/" + index + "/_delete_by_query", params,
entity);
        if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
            LOG.info("DeleteAllDocumentsInIndex successful.");
        }
        else {
            LOG.error("DeleteAllDocumentsInIndex failed.");
        }
        LOG.info("DeleteAllDocumentsInIndex response entity is :
{}.",EntityUtils.toString(response.getEntity()));
    }
    catch (Exception e) {
```

```
        LOG.error("DeleteAllDocumentsInIndex failed, exception occurred.", e);
    }
```

### 2.3.3.2.12 Refreshing an Index

#### Refreshing a Single Index

Flush.java in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/flush` directory is used to refresh a specified index.

```
/**
 * Flush one index data to storage and clearing the internal transaction log
 */
public static void flushOneIndex(RestClient restClientTest, String index) {
    Response flushRsp;

    try {
        Request request=new Request("POST", "/" + index + "/_flush");
        request.addParameter("pretty", "true");
        flushRsp = restClientTest.performRequest(request);
        LOG.info(EntityUtils.toString(flushRsp.getEntity()));

        if (HttpStatus.SC_OK == flushRsp.getStatusLine().getStatusCode()) {
            LOG.info("Flush one index successful.");
        } else {
            LOG.error("Flush one index failed.");
        }
        LOG.info("Flush one index response entity is : {}.",EntityUtils.toString(flushRsp.getEntity()));
    } catch (Exception e) {
        LOG.error("Flush one index failed, exception occurred.", e);
    }
}
```

#### Refreshing all Indexes

Flush.java in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/flush` directory is used to refresh all indexes.

```
/**
 * Flush all indexes data to storage and clearing the internal transaction log
 * The user who performs this operation must belong to the "supergroup" group.
 */
private static void flushAllIndexes(RestClient restClientTest) {
    Map<String, String> params = Collections.singletonMap("pretty", "true");
    Response flushRsp;
    try {
        flushRsp = restClientTest.performRequest("POST", "/_all/_flush",params);
        if (HttpStatus.SC_OK == flushRsp.getStatusLine().getStatusCode()) {
            LOG.info("Flush all indexes successful.");
        } else {
            LOG.error("Flush all indexes failed.");
        }
        LOG.info("Flush all indexes response entity is : " + EntityUtils.toString(flushRsp.getEntity()));
    } catch (Exception e) {
        LOG.error("Flush all indexes failed, exception occurred.", e);
    }
}
```



#### NOTE

Only users in the supergroup group can perform this operation.

### 2.3.3.2.13 Multi-Thread Example

#### Function

**MultithreadRequest.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/multithread` directory is used to create a thread class **sendRequestThread**, rewrite a **run** method, implement a **bulk** request, and write data in batches.

```
/*
 * Thread that sends bulk request
 */
public static class sendRequestThread implements Runnable {

    private RestClient restClientTh;
    private HwRestClient hwRestClient;
    @Override
    public void run() {
        LOG.info("Thread begin.");
        try {

            hwRestClient = new HwRestClient();
            restClientTh = hwRestClient.getRestClient();
            LOG.info("Thread name: " + Thread.currentThread().getName());
            bulk(restClientTh, "huawei1", "type1");
        } catch (Exception e) {
            LOG.error("SendRequestThread has exception.", e);
        } finally {
            if (restClientTh != null) {
                try {
                    restClientTh.close();
                    LOG.info("Close the client successful in thread : " + Thread.currentThread().getName());
                } catch (IOException e) {
                    LOG.error("Close the client failed.", e);
                }
            }
        }
    }
}
```

In the main method, the **fixedThreadPool** thread pool is created. By specifying **threadPoolSize** and **jobNumber**, multiple threads are supported to send **bulk** requests and data can be concurrently written in batches.

```
public static void main(String[] args) throws Exception {
    LOG.info("Start to do multithread request !");
    ExecutorService fixedThreadPool;
    HwRestClient hwRestClient = new HwRestClient();
    RestClient restClientThNew;
    restClientThNew = hwRestClient.getRestClient();
    int threadPoolSize = 2;
    int jobNumber = 5;
    try {
        fixedThreadPool = Executors.newFixedThreadPool(threadPoolSize);
        for (int i = 0; i < jobNumber; i++) {
            fixedThreadPool.execute(new sendRequestThread());
        }
        fixedThreadPool.shutdown();
    } catch (Exception e) {
        LOG.error("There are exceptions in sendRequestThread.", e);
    } finally {
        if (restClientThNew != null) {
            try {
                restClientThNew.close();
                LOG.info("Close the client successful.");
            } catch (Exception e1) {
                LOG.error("Close the client failed.", e1);
            }
        }
    }
}
```

```
        }
    }
}
```

### 2.3.3.2.14 Pre-sorting Indices

#### Function Description

The `IndexSorting.java` file in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/lowlevel/index` directory is used to configure the fields to be sorted when an index is created, implement the **bulk** request, write data in batches, and query the top N documents.

During index creation, specify the fields to be sorted and specify whether each field is sorted in descending or ascending order.

```
private static void createIndexWithSorting(RestClient restClient, String index) {
    Response rsp;
    String jsonString;
    jsonString = "{\"settings\":{\"number_of_shards\":\"3\", \"number_of_replicas\":\"1\",
        + "\"sort.field\": [\"height\", \"weight\"], \"sort.order\": [\"desc\", \"asc\"]},\"mappings\":{"
        + "\"properties\":{\"height\":{\"type\":\"float\"},\"weight\":{\"type\":\"float\"}}}}";
    HttpEntity entity = new StringEntity(jsonString, ContentType.APPLICATION_JSON);
    try {
        Request request = new Request("PUT", "/" + index);
        request.addParameter("pretty", "true");
        request.setEntity(entity);
        rsp = restClient.performRequest(request);
        if (rsp.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            LOG.info("Create index with sorting successfully.");
        } else {
            LOG.error("Create index with sorting failed.");
        }
        LOG.info("Create index with sorting response entity is {}.", EntityUtils.toString(rsp.getEntity()));
    } catch (IOException | ParseException e) {
        LOG.error("Failed to create index with sorting, exception occurred.", e);
    }
}
```

Then, write data in batches.

```
public static void bulk(RestClient restClientTest, String index) {
    // Total number of documents to be written
    long totalRecordNum = 10000;
    // Number of documents written in each bulk. The recommended size is 5 MB to 15 MB.
    long oneCommit = 1000;
    long circleNumber = totalRecordNum / oneCommit;
    StringEntity entity;
    Gson gson = new Gson();
    Map<String, Object> esMap = new HashMap<>();
    String str = "{ \"index\" : { \"_index\" : \"" + index + "\" } }";

    for (int i = 1; i <= circleNumber; i++) {
        StringBuilder builder = new StringBuilder();
        for (int j = 1; j <= oneCommit; j++) {
            esMap.clear();
            esMap.put("name", "Linda");
            esMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
            esMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
            esMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
            esMap.put("cur_time", System.currentTimeMillis());

            String strJson = gson.toJson(esMap);
            builder.append(str).append(System.lineSeparator());
            builder.append(strJson).append(System.lineSeparator());
        }
    }
}
```

```
        }
        entity = new StringEntity(builder.toString(), ContentType.APPLICATION_JSON);
        entity.setContentTypeEncoding("UTF-8");
        Response response;
        try {
            Request request = new Request("PUT", "/_bulk");
            request.addParameter("pretty", "true");
            request.setEntity(entity);
            response = restClientTest.performRequest(request);
            if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
                LOG.info("Already input documents : {}.", oneCommit * i);
            } else {
                LOG.error("Bulk failed.");
            }
            LOG.info("Bulk response entity is : {}.", EntityUtils.toString(response.getEntity()));
        } catch (IOException | ParseException e) {
            LOG.error("Bulk failed, exception occurred.", e);
        }
    }
}
```

Perform the refresh operation so that the data can be queried after being written in batches.

```
private static void refresh(RestClient restClient, String index) {
    try {
        Map<String, String> params = Collections.singletonMap("pretty", Boolean.TRUE.toString());
        Response rsp = restClient.performRequest(buildRequest("POST", "/" + index + "/_refresh", params,
null));
        if (rsp.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            LOG.info("Refresh sorting index successfully.");
        }
    } catch (IOException e) {
        LOG.error("Failed to refresh sorting index, exception occurred.", e);
    }
}
```

Then, you can query the sorting field. By default, the top 10 documents are obtained.

```
private static void queryData(RestClient restClient, String index) {
    Response response;
    String jsonStr;
    jsonStr = "{\"sort\":[{\"height\": \"desc\", \"weight\": \"asc\"}],\"track_total_hits\": false}";
    StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
    try {
        Request request = new Request("GET", "/" + index + "/_search");
        request.addParameter("pretty", "true");
        request.setEntity(entity);
        response = restClient.performRequest(request);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            LOG.info("Query sorting index successfully.");
        } else {
            LOG.error("Failed to query sorting index.");
        }
        LOG.info("Query sorting index response entity is {}.", EntityUtils.toString(response.getEntity()));
    } catch (IOException | ParseException e) {
        LOG.error("Failed to query sorting index, exception occurred.", e);
    }
}
```

### 2.3.3.3 High Level RestClient Sample Code

### 2.3.3.3.1 Connecting the Client to a Cluster

#### Function

You can obtain the client and connect to a specific Elasticsearch cluster by setting the IP address and port. This is necessary before you use the API provided by the Elasticsearch.

##### NOTE

Call **RestHighLevelClient.close()** to close requested resources after the Elasticsearch service operation is complete.

```
public static void main(String[] args) {
    RestHighLevelClient highLevelClient = null;
    HwRestClient hwRestClient = new HwRestClient();
    try {
        highLevelClient = new RestHighLevelClient(hwRestClient.getRestClientBuilder());
        ...
    } finally {
        try {
            if (highLevelClient != null) {
                highLevelClient.close();
            }
        } catch (IOException e) {
            LOG.error("Failed to close RestHighLevelClient.", e);
        }
    }
}
```

##### NOTE

- By default, the HwRestClient reads the following configuration files from the **conf** directory in the code running path: **esParams.properties**, **krb5.conf**, and **user.keytab**.
- The configuration file path can be customized. For example, if the code runs in the Windows operating system, and the configuration files are stored in the root directory of drive D, you can set **HwRestClient hwRestClient** to **new HwRestClient("D:\\")**.

### 2.3.3.2 Querying Cluster Information

#### Function

**QueryClusterInfo.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/cluster** directory is used to query information about the Elasticsearch cluster.

```
/**
 * Get cluster information
 */
public static void queryClusterInfo(RestHighLevelClient highLevelClient) {

    try {
        MainResponse response = highLevelClient.info(RequestOptions.DEFAULT);
        ClusterName clusterName = response.getClusterName();
        LOG.info("ClusterName:[{}], clusterUuid:[{}], nodeName:[{}], version:[{}].", new Object[]
        { clusterName.value(),
            response.getClusterUuid(), response.getNodeName(), response.getVersion().toString() });
    } catch (Exception e) {
        LOG.error("QueryClusterInfo is failed,exception occurred.", e);
    }
}
```

### 2.3.3.3 Writing Index Data

Data can be written to a specified index by specifying data sources in different formats, such as a character string in the JSON format, a data source in the Map format, and a data source in the XContentBuilder format.

#### Writing Data by Using the Character String in the JSON Format

**IndexByJson.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index` directory is used to write data into a specified index by using a data source in the JSON format.

The variable `jsonString` is data to be inserted and can be customized.

```
/*
 * Create or update index by json
 */
public static void indexByJson(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        IndexRequest indexRequest = new IndexRequest(index, type, id);
        String jsonString = "{" + "\"user\":\"kimchy1\"," + "\"age\":100," + "\"postDate\"
\":\\"2020-01-01\",
        + "\"message\":\"trying out Elasticsearch\"," + "\"reason\":\"daily update\",
        + "\"innerObject1\":\"Object1\"," + "\"innerObject2\":\"Object2\",
        + "\"innerObject3\":\"Object3\"," + "\"uid\":\"11\"}";
        indexRequest.source(jsonString, XContentType.JSON);
        IndexResponse indexResponse = highLevelClient.index(indexRequest, RequestOptions.DEFAULT);

        LOG.info("IndexByJson response is {}.", indexResponse.toString());
    } catch (Exception e) {
        LOG.error("IndexByJson is failed,exception occurred.", e);
    }
}
```

#### Writing Data in the Map Format

**IndexByMap.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index` directory is used to write data into a specified index by using a data source in the Map format.

The variable `dataMap` is data to be inserted and can be customized.

```
/*
 * Create or update index by map
 */
public static void indexByMap(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        Map<String, Object> dataMap = new HashMap<>();
        dataMap.put("user", "kimchy2");
        dataMap.put("age", "200");
        dataMap.put("postDate", new Date());
        dataMap.put("message", "trying out Elasticsearch");
        dataMap.put("reason", "daily update");
        dataMap.put("innerObject1", "Object1");
        dataMap.put("innerObject2", "Object2");
        dataMap.put("innerObject3", "Object3");
        dataMap.put("uid", "22");
        IndexRequest indexRequest = new IndexRequest(index, type, id).source(dataMap);
        IndexResponse indexResponse = highLevelClient.index(indexRequest, RequestOptions.DEFAULT);

        LOG.info("IndexByMap response is {}.", indexResponse.toString());
    } catch (Exception e) {
        LOG.error("IndexByMap is failed,exception occurred.", e);
    }
}
```

## Writing Data in the XContentBuilder Format

**IndexByXContentBuilder.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index** directory is used to write data into a specified index by using a data source in the XContentBuilder format.

The variable *builder* is data to be inserted and can be customized.

```
/*
 * Create or update index by XContentBuilder
 */
public static void indexByXContentBuilder(RestHighLevelClient highLevelClient, String index, String type, String id) {

    try {
        XContentBuilder builder = XContentFactory.jsonBuilder();
        builder.startObject();
        {
            builder.field("user", "kimchy3");
            builder.field("age", "300");
            builder.field("postDate", "2020-01-01");
            builder.field("message", "trying out Elasticsearch");
            builder.field("reason", "daily update");
            builder.field("innerObject1", "Object1");
            builder.field("innerObject2", "Object2");
            builder.field("innerObject3", "Object3");
            builder.field("uid", "33");
        }
        builder.endObject();
        IndexRequest indexRequest = new IndexRequest(index, type, id).source(builder);
        IndexResponse indexResponse = highLevelClient.index(indexRequest, RequestOptions.DEFAULT);

        LOG.info("IndexByXContentBuilder response is {}", indexResponse.toString());
    } catch (Exception e) {
        LOG.error("IndexByXContentBuilder is failed,exception occurred.", e);
    }
}
```

### 2.3.3.3.4 Operations in Batches

## Function

**Bulk.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/bulk** directory is used to perform operations in batches, for example, create an index, update an index, or delete an index.

```
/*
 * Bulk request can be used to execute multiple index,update or delete
 */
public static void bulk(RestHighLevelClient highLevelClient, String index, String type) {

    try {
        Map<String, Object> jsonMap = new HashMap<>();
        for (int i = 1; i <= 100; i++) {
            BulkRequest request = new BulkRequest();
            for (int j = 1; j <= 1000; j++) {
                jsonMap.clear();
                jsonMap.put("user", "Linda");
                jsonMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
                jsonMap.put("postDate", "2020-01-01");
                jsonMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
                jsonMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
            }
            request.add(new IndexRequest(index, type, Integer.toString(i), jsonMap));
        }
        highLevelClient.bulk(request);
    } catch (IOException e) {
        LOG.error("bulk error, exception occurred.", e);
    }
}
```

```
        request.add(new IndexRequest(index, type).source(jsonMap));
    }
    BulkResponse bulkResponse = highLevelClient.bulk(request, RequestOptions.DEFAULT);

    if (RestStatus.OK.equals((bulkResponse.status()))) {
        LOG.info("Bulk is successful");
    } else {
        LOG.info("Bulk is failed");
    }
}
} catch (Exception e) {
    LOG.error("Bulk is failed,exception occurred.", e);
}
}
```

### 2.3.3.3.5 Batch write data to specified routes

#### Function

**BulkRoutingSample.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/bulk` directory is used to set routing for each request in the bulk, route documents with the same routing to the same segment, and specify routing during query to narrow the query range and improve the query speed.

```
/**
 * Number of documents to be written.
 */
private static final int NUM_OF_DOC = 100;

/**
 * Use bulk to write data and specify routing.
 */
private static void bulkWithRouting(RestHighLevelClient highLevelClient, String indexName) {
    try {
        Map<String, Object> jsonMap = new HashMap<>();
        BulkRequest request = new BulkRequest();
        int age;
        for (int i = 1; i <= NUM_OF_DOC; i++) {
            jsonMap.clear();
            jsonMap.put("user", "Linda" + i);
            age = ThreadLocalRandom.current().nextInt(18, 100);
            jsonMap.put("age", age);
            jsonMap.put("postDate", "2020-01-01");
            jsonMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
            jsonMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
            // Route documents of the same age to the same shard based on the age.
            request.add(new IndexRequest(indexName).source(jsonMap).routing(String.valueOf(age)));
        }
        BulkResponse bulkResponse = highLevelClient.bulk(request, RequestOptions.DEFAULT);
        if (RestStatus.OK.equals(bulkResponse.status())) {
            LOG.info("Bulk routing is successful.");
        } else {
            LOG.info("Bulk routing is failed.");
        }
    } catch (IOException e) {
        LOG.error("Bulk routing is failed, exception occurred.", e);
    }
}
```

### 2.3.3.6 Updating Index Information

#### Function

**Update.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/update` directory is used to update the index.

```
/*
 * Update index
 */
public static void update(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        XContentBuilder builder = XContentFactory.jsonBuilder();
        builder.startObject();
        { // update information
            builder.field("postDate", new Date());
            builder.field("reason", "update again");
        }
        builder.endObject();
        UpdateRequest request = new UpdateRequest(index, type, id).doc(builder);
        UpdateResponse updateResponse = highLevelClient.update(request, RequestOptions.DEFAULT);

        LOG.info("Update response is {}", updateResponse.toString());
    } catch (Exception e) {
        LOG.error("Update is failed,exception occurred.", e);
    }
}
```

### 2.3.3.7 Querying Index Information

#### Function

**GetIndex.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/search` directory is used to query information about a document with specified **index**, **type**, and **id**.

```
/*
 * Get index information
 */
private static void getIndex(RestHighLevelClient highLevelClient, String index, String type, String id) {
    try {
        GetRequest getRequest = new GetRequest(index, type, id);
        String[] includes = new String[] { "message", "test*" };
        String[] excludes = Strings.EMPTY_ARRAY;
        FetchSourceContext fetchSourceContext = new FetchSourceContext(true, includes, excludes);
        getRequest.fetchSourceContext(fetchSourceContext);
        getRequest.storedFields("message");
        GetResponse getResponse = highLevelClient.get(getRequest, RequestOptions.DEFAULT);

        LOG.info("GetIndex response is {}", getResponse.toString());
    } catch (Exception e) {
        LOG.error("GetIndex is failed,exception occurred.", e);
    }
}
```

### 2.3.3.8 Searching for Document Information

#### Function

**Search.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/search` directory is used to query related document information in a specified index. You can customize

specified sorting, filtering, highlighting, feedback data range, timeout interval, and the like.

```
/*
 * Search some information in index
 */
public static void search(RestHighLevelClient highLevelClient, String index) {
    try {
        SearchRequest searchRequest = new SearchRequest(index);
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.termQuery("user", "kimchy1"));

        //Specifying Sorting
        searchSourceBuilder.sort(new FieldSortBuilder("_doc").order(SortOrder.ASC));

        // Source filter
        String[] includeFields = new String[] { "message", "user", "innerObject" };
        String[] excludeFields = new String[] { "postDate" };
        searchSourceBuilder.fetchSource(includeFields, excludeFields);

        // Control which fields get included or
        // excluded
        // Request Highlighting
        HighlightBuilder highlightBuilder = new HighlightBuilder();
        HighlightBuilder.Field highlightUser = new HighlightBuilder.Field("user");
        // Create a field highlighter for the user field
        highlightBuilder.field(highlightUser);
        searchSourceBuilder.highlighter(highlightBuilder);
        searchSourceBuilder.from(0);
        searchSourceBuilder.size(2);
        searchSourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS));
        searchRequest.source(searchSourceBuilder);
        SearchResponse searchResponse = highLevelClient.search(searchRequest, RequestOptions.DEFAULT);

        LOG.info("Search response is {}.", searchResponse.toString());
    } catch (Exception e) {
        LOG.error("Search is failed, exception occurred.", e);
    }
}
```

### 2.3.3.3.9 Searching for Document Information by Using a Cursor

#### Function

**SearchScroll.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/searchscroll** directory is used to query related document information in a specified index by using a cursor. The method returns a **scrollId**. You can use the **scrollId** to clear the cursor. For details about the clearing method, see [Clearing a Cursor](#).

```
/*
 * Send a search scroll request
 */
public static String searchScroll(RestHighLevelClient highLevelClient, String index) {
    String scrollId = null;
    try {
        final Scroll scroll = new Scroll(TimeValue.timeValueMinutes(1L));
        SearchRequest searchRequest = new SearchRequest(index);
        searchRequest.scroll(scroll);
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchQuery("title", "Elasticsearch"));
        searchRequest.source(searchSourceBuilder);

        SearchResponse searchResponse = highLevelClient.search(searchRequest, RequestOptions.DEFAULT);
        scrollId = searchResponse.getScrollId();
    }
```

```
SearchHit[] searchHits = searchResponse.getHits().getHits();
LOG.info("SearchHits is {}", searchResponse.toString());

while (searchHits != null && searchHits.length > 0) {
    SearchScrollRequest scrollRequest = new SearchScrollRequest(scrollId);
    scrollRequest.scroll(scroll);
    searchResponse = highLevelClient.searchScroll(scrollRequest, RequestOptions.DEFAULT);
    scrollId = searchResponse.getScrollId();
    searchHits = searchResponse.getHits().getHits();
    LOG.info("SearchHits is {}", searchResponse.toString());
}
} catch (Exception e) {
    LOG.error("SearchScroll is failed,exception occurred.", e);
    return null;
}
return scrollId;
}
```

### 2.3.3.3.10 Clearing a Cursor

#### Function

**SearchScroll.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/searchscroll** directory includes the method **clearScroll(highLevelClient,scrollId)**, which is used to clear a cursor specified by **scrollId**.

```
/**
 * Clear a search scroll
 */
public static void clearScroll(RestHighLevelClient highLevelClient, String scrollId) {
    ClearScrollRequest clearScrollRequest = new ClearScrollRequest();
    clearScrollRequest.addScrollId(scrollId);
    ClearScrollResponse clearScrollResponse;
    try {
        clearScrollResponse = highLevelClient.clearScroll(clearScrollRequest, RequestOptions.DEFAULT);
        if (clearScrollResponse.isSucceeded()) {
            LOG.info("ClearScroll is successful.");
        } else {
            LOG.error("ClearScroll is failed.");
        }
    } catch (IOException e) {
        LOG.error("ClearScroll is failed,exception occurred.", e);
    }
}
```

### 2.3.3.3.11 Deleting an Index

#### Function

**DeleteIndex.java** in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/delete** directory is used to delete a specified index.

```
/**
 * Delete the index

*/
public static void deleteIndex(RestHighLevelClient highLevelClient, String index) {
    try {
        DeleteIndexRequest request = new DeleteIndexRequest(index);
        DeleteIndexResponse deleteResponse = highLevelClient.indices().deleteIndex(request,
RequestOptions.DEFAULT);
```

```
        if (deleteResponse.isAcknowledged()) {
            LOG.info("Delete index is successful");
        } else {
            LOG.info("Delete index is failed");
        }
    } catch (Exception e) {
        LOG.error("Delete index : {} is failed, exception occurred.", index, e);
    }
}
```

### 2.3.3.12 Multi-Thread Request

#### Function

**MultithreadRequest.java** in the `elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/multithread` directory is used to create a thread class `sendRequestThread`, rewrite a `run` method, implement a **bulk** request, and write data in batches.

```
/*
 * The thread that send a bulk request
 */
public static class sendRequestThread implements Runnable {

    private RestHighLevelClient highLevelClientTh;
    @Override
    public void run() {
        LOG.info("Thread begin.");

        HwRestClient hwRestClient = new HwRestClient();
        try {
            highLevelClientTh = new RestHighLevelClient(hwRestClient.getRestClientBuilder());
            LOG.info("Thread name: " + Thread.currentThread().getName());
            bulk(highLevelClientTh, "huawei1", "type1");
        } catch (Exception e) {
            LOG.error("SendRequestThread had exception.", e);
        } finally {
            if (highLevelClientTh != null) {
                try {
                    highLevelClientTh.close();
                    LOG.info("Close the highLevelClient successful in thread : " +
Thread.currentThread().getName());
                } catch (IOException e) {
                    LOG.error("Close the highLevelClient failed.", e);
                }
            }
        }
    }
}
```

In the main method, the **fixedThreadPool** thread pool is created. By specifying **threadPoolSize** and **jobNumber**, multiple threads are supported to send bulk requests and data can be concurrently written in batches.

```
public static void main(String[] args) throws Exception {
    LOG.info("Start to do bulk request !");

    ExecutorService fixedThreadPool;
    int threadPoolSize = 2;
    int jobNumber = 5;
    if (HighLevelUtils.getConfig()) {
        try {
            fixedThreadPool = Executors.newFixedThreadPool(threadPoolSize);
        }
```

```
        for (int i = 0; i < jobNumber; i++) {
            fixedThreadPool.execute(new sendRequestThread());
        }
        fixedThreadPool.shutdown();
    } catch (Exception e) {
        LOG.error("SendRequestThread is failed,exception occurred.", e);
    }
} else {
    LOG.error("Failed to get configuration!");
}
}
```

### 2.3.3.13 BulkProcessor Batch Import Example

#### Description

The **BulkProcessorSample.java** file in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/bulk** directory is used to instruct users to use BulkProcessor to import data to the database in batches.

BulkProcessor initialization:

```
private static BulkProcessor getBulkProcessor(RestHighLevelClient highLevelClient) {
    BulkProcessor.Listener listener = new BulkProcessor.Listener() {
        @Override
        public void beforeBulk(long executionId, BulkRequest bulkRequest) {
            int numberOfActions = bulkRequest.numberOfActions();
            LOG.info("Executing bulk {} with {} requests.", executionId, numberOfActions);
        }

        @Override
        public void afterBulk(long executionId, BulkRequest bulkRequest, BulkResponse bulkResponse) {
            if (bulkResponse.hasFailures()) {
                LOG.warn("Bulk {} executed with failures.", executionId);
            } else {
                LOG.info("Bulk {} completed in {} milliseconds.", executionId,
bulkResponse.getTook().getMillis());
            }
        }

        @Override
        public void afterBulk(long executionId, BulkRequest bulkRequest, Throwable throwable) {
            LOG.error("Failed to execute bulk.", throwable);
        }
    };

    BiConsumer<BulkRequest, ActionListener<BulkResponse>> bulkConsumer =
        (request, bulkListener) -> highLevelClient.bulkAsync(request, RequestOptions.DEFAULT,
bulkListener);

    BulkProcessor bulkProcessor = BulkProcessor.builder(bulkConsumer, listener)
        .setBulkActions(onceBulkMaxNum)
        .setBulkSize(new ByteSizeValue(onceBulkMaxSize, ByteSizeUnit.MB))
        .setConcurrentRequests(concurrentRequestsNum)
        .setFlushInterval(TimeValue.timeValueSeconds(flushTime))
        .setBackoffPolicy(BackoffPolicy.constantBackoff(TimeValue.timeValueSeconds(1L), maxRetry))
        .build();

    LOG.info("Init bulkProcess successfully.");
    return bulkProcessor;
}
```

BulkProcessor import example:

```
private void singleThreadBulk() {
    //single thread
```

```

int bulkTime = 0;
while (bulkTime++ < totalNumberForThread) {
    Map<String, Object> dataMap = new HashMap<>();
    dataMap.put("date", "2019/12/9");
    dataMap.put("text", "the test text");
    dataMap.put("title", "the title");
    bulkProcessor.add(new IndexRequest(indexName, indexType).source(dataMap));
}
LOG.info("This thread bulks successfully, the thread name is {}.", Thread.currentThread().getName());
}

```

### 2.3.3.14 Pre-sorting Indices

#### Function Description

The **IndexSorting.java** file in the **elasticsearch-rest-client-example/src/main/java/com/huawei/fusioninsight/elasticsearch/example/highlevel/index** directory is used to configure the fields to be sorted when an index is created, implement the **bulk** request, write data in batches, and query the top N documents.

During index creation, specify the fields to be sorted and specify whether each field is sorted in descending or ascending order.

```

private static void createIndexWithSorting(RestHighLevelClient highLevelClient, String index) {
    try {
        CreateIndexRequest indexRequest = new CreateIndexRequest(index);
        indexRequest.settings(
            Settings.builder().put("index.number_of_shards", 3)
                .put("index.number_of_replicas", 1)
                .putList("index.sort.field", "height", "weight")
                .putList("index.sort.order", SortOrder.DESC.toString(), SortOrder.ASC.toString()));
        indexRequest.mapping("{\"properties\": {\"height\": {\"type\": \"float\"},"
            + "\"weight\": {\"type\": \"float\"}}}", XContentType.JSON);
        CreateIndexResponse response = highLevelClient.indices().create(indexRequest,
RequestOptions.DEFAULT);
        if (response.isAcknowledged() || response.isShardsAcknowledged()) {
            LOG.info("Create index with sorting successfully.");
        }
    } catch (IOException e) {
        LOG.error("Failed to create index with sorting, exception occurred.", e);
    }
}

```

Then, write data in batches.

```

public static void bulk(RestHighLevelClient highLevelClient, String index) {
    try {
        Map<String, Object> jsonMap = new HashMap<>();
        for (int i = 1; i <= 10; i++) {
            BulkRequest request = new BulkRequest();
            for (int j = 1; j <= 1000; j++) {
                jsonMap.clear();
                jsonMap.put("user", "Linda");
                jsonMap.put("age", ThreadLocalRandom.current().nextInt(18, 100));
                jsonMap.put("postDate", "2020-01-01");
                jsonMap.put("height", (float) ThreadLocalRandom.current().nextInt(140, 220));
                jsonMap.put("weight", (float) ThreadLocalRandom.current().nextInt(70, 200));
                request.add(new IndexRequest(index).source(jsonMap));
            }
            BulkResponse bulkResponse = highLevelClient.bulk(request, RequestOptions.DEFAULT);

            if (RestStatus.OK.equals((bulkResponse.status()))) {
                LOG.info("Bulk is successful");
            } else {
                LOG.info("Bulk is failed");
            }
        }
    }
}

```

```
        }
    }
} catch (IOException e) {
    LOG.error("Bulk is failed,exception occurred.", e);
}
}
```

Perform the refresh operation so that the data can be queried after being written in batches.

```
private static void refresh(RestHighLevelClient highLevelClient, String index) {
    try {
        RefreshRequest refRequest = new RefreshRequest(index);
        highLevelClient.indices().refresh(refRequest, RequestOptions.DEFAULT);
        LOG.info("Refresh sorting index successfully.");
    } catch (IOException e) {
        LOG.error("Failed to refresh sorting index, exception occurred.", e);
    }
}
```

Then, you can query the sorting field. By default, the top 10 documents are obtained.

```
private static void queryData(RestHighLevelClient highLevelClient, String index) {
    try {
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.sort(new FieldSortBuilder("height").order(SortOrder.DESC));
        searchSourceBuilder.sort(new FieldSortBuilder("weight").order(SortOrder.ASC));
        // Source filter
        String[] includeFields = new String[]{"weight", "height", "age"};
        String[] excludeFields = new String[]{};
        searchSourceBuilder.fetchSource(includeFields, excludeFields);
        // Control which fields get included or excluded
        // Request Highlighting
        HighlightBuilder highlightBuilder = new HighlightBuilder();
        HighlightBuilder.Field highlightUser = new HighlightBuilder.Field("height");
        highlightBuilder.field(highlightUser);
        searchSourceBuilder.highlighter(highlightBuilder);
        searchSourceBuilder.timeout(new TimeValue(1, TimeUnit.SECONDS));
        SearchRequest searchRequest = new SearchRequest(index);
        searchRequest.source(searchSourceBuilder);
        SearchResponse searchResponse = highLevelClient.search(searchRequest, RequestOptions.DEFAULT);
        LOG.info("Search response is {}", searchResponse.toString());
    } catch (IOException e) {
        LOG.error("Failed to query sorting index, exception occurred.", e);
    }
}
```

## 2.3.4 Commissioning Process

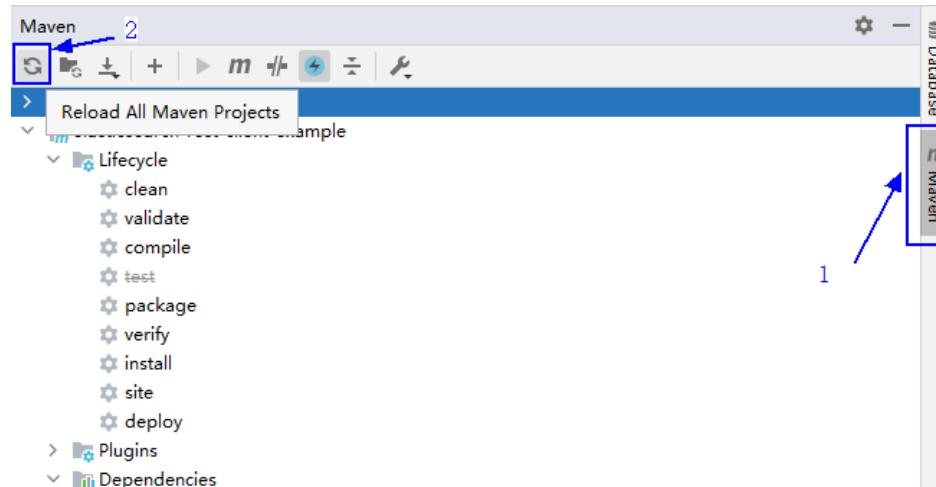
### 2.3.4.1 Commissioning Applications on Windows

#### 2.3.4.1.1 Commissioning a RestClient Application

#### Compiling and Running Applications

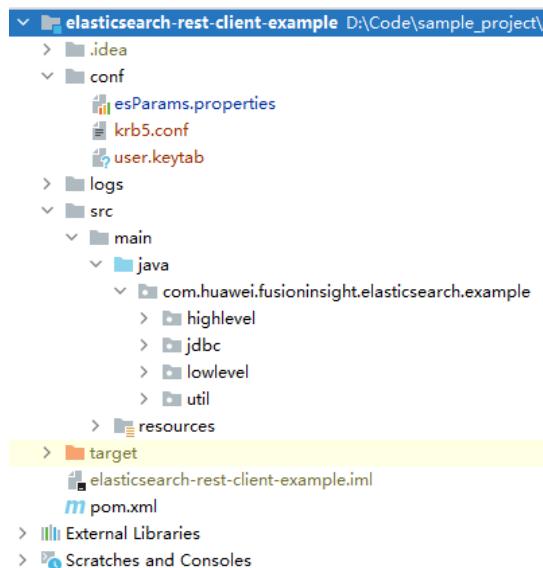
You can run an application on Windows after code development is complete. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

1. Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.



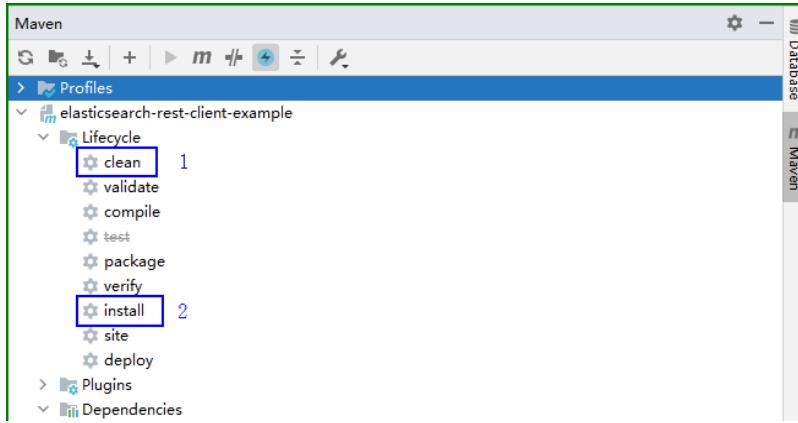
2. Compile and run an application.

- The following figure shows the file list after the configuration file has been placed and the code has been modified to match the login user. (The authentication files krb5 and keytab are not required in common mode.)



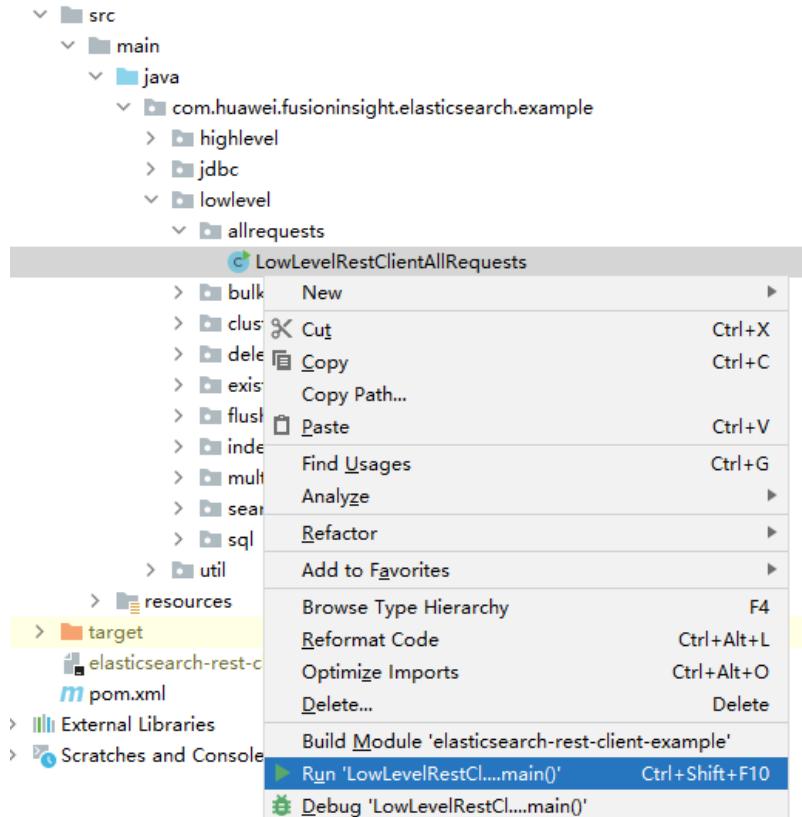
- Compile an application.

Double-click **clean** and **install** to run the **maven clean** and **maven install** commands. After the compilation is complete, the message "Build Success" is displayed.

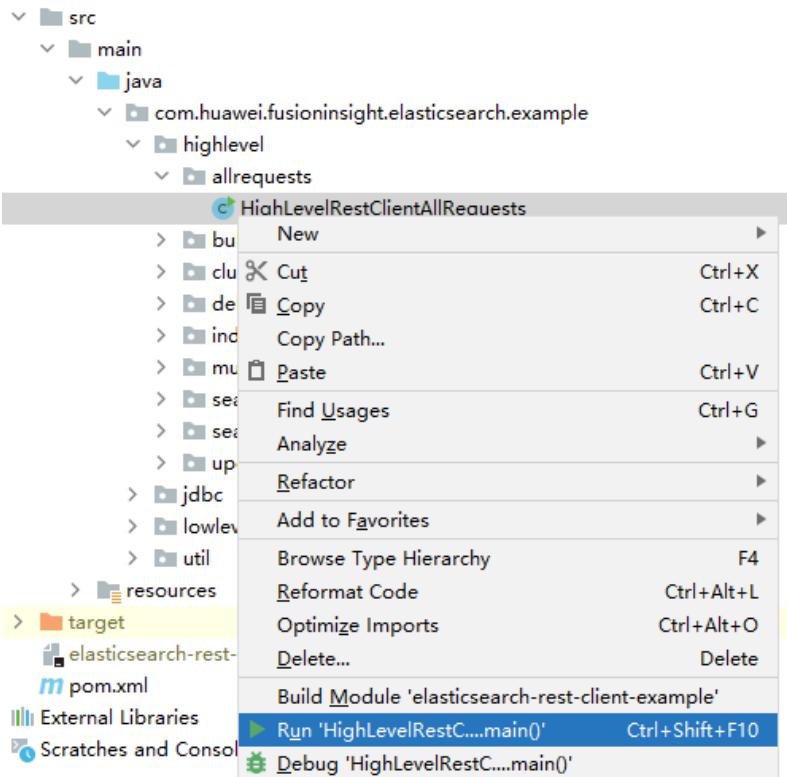


- c. Run the program.

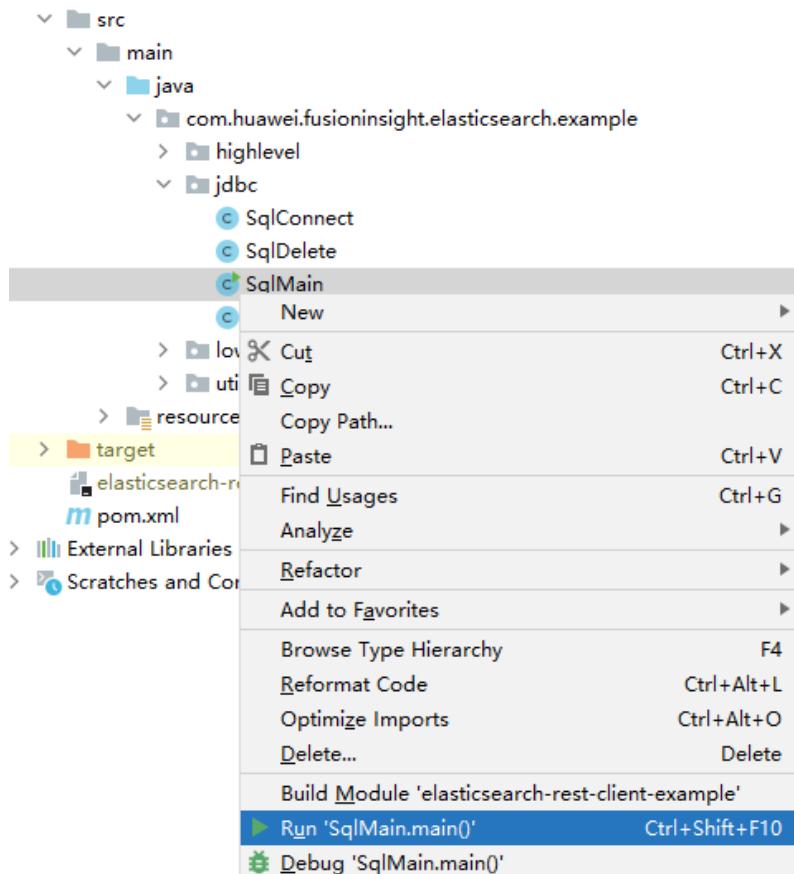
Right-click the **LowLevelRestClientAllRequests** file and choose **Run 'Low....main()'** from the shortcut menu to run all samples of the Low Level Rest Client.



Right-click the **HighLevelRestClientAllRequests** file and choose **Run 'High....main()'** from the shortcut menu to run all samples of the High Level Rest Client.



Right-click the **SqlMain** file and choose **Run 'SqlMain.main()'** from the shortcut menu to run all samples of the JDBC Rest Client.



## Viewing Commissioning Results

After you run an Elasticsearch application, you can use either of the following methods to view the running status:

- View the running result.
- View Elasticsearch logs, that is, the **elasticsearch-rest-client-example/logs/es-example.log** log file in the **logs** directory.

After a complete sample of the Low Level RestClient is run, some operation results are displayed on the console:

```
2019-01-10 22:05:44,091 | INFO | main | Start to do low level rest client request ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.main(LowLevelRestClientAllRequests.java:429)
2019-01-10 22:05:44,096 | INFO | main | esServerHost:10.1.1.1:24100 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:73)
2019-01-10 22:05:44,097 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:74)
2019-01-10 22:05:44,097 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:75)
2019-01-10 22:05:44,098 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:76)
2019-01-10 22:05:44,098 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:77)
2019-01-10 22:05:44,102 | INFO | main | shardNum:5 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:78)
2019-01-10 22:05:44,102 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:79)
2019-01-10 22:05:44,102 | INFO | main | isSecureMode:false |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:80)
2019-01-10 22:05:44,103 | INFO | main | oneCommit:5 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:81)
2019-01-10 22:05:44,103 | INFO | main | totalRecordNumber:10 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:82)
2019-01-10 22:05:44,103 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:83)
2019-01-10 22:05:47,246 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:05:47,246 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:05:47,246 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:05:47,249 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:05:47,251 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:05:47,251 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:05:47,426 | INFO | main | The Low Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.getRestClient(LowLevelRestClientAllRequests.java:162)
2019-01-10 22:05:47,587 | INFO | main | QueryClusterInfo successful. |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.queryClusterInfo(LowLevelRestClientAllRequests.java:178)
2019-01-10 22:05:47,588 | INFO | main | QueryClusterInfo response entity is : {
```

```
"cluster_name" : "elasticsearch_cluster",
"status" : "yellow",
"timed_out" : false,
"number_of_nodes" : 3,
"number_of_data_nodes" : 1,
"active_primary_shards" : 15,
"active_shards" : 15,
"relocating_shards" : 0,
"initializing_shards" : 0,
"unassigned_shards" : 15,
"delayed_unassigned_shards" : 0,
"number_of_pending_tasks" : 0,
"number_of_in_flight_fetch" : 0,
"task_max_waiting_in_queue_millis" : 0,
"active_shards_percent_as_number" : 50.0
}
|
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.queryClusterInfo(LowLevelRestClientAllRequests.java:182)
2019-01-10 22:05:47,648 | INFO | main | Index is not exist : huawei1 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.indexIsExist(LowLevelRestClientAllRequests.java:202)
2019-01-10 22:05:47,823 | INFO | main | CreateIndexWithShardNum successful. |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.createIndexWithShardNum(LowLevelRestClientAllRequests.java:227)
2019-01-10 22:05:47,824 | INFO | main | CreateIndexWithShardNum response entity is : {
    "acknowledged" : true,
    "shards_acknowledged" : true,
    "index" : "huawei1"
}
|
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.createIndexWithShardNum(LowLevelRestClientAllRequests.java:231)
```

After a complete sample of the High Level RestClient is run, some operation results are displayed on the console:

```
2019-01-10 22:04:08,933 | INFO | main | Start to do high level rest client request ! |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.main(HighLevelRestClientAllRequests.java:455)
2019-01-10 22:04:08,937 | INFO | main | esServerHost:10.1.1.1:24100 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:94)
2019-01-10 22:04:08,937 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:95)
2019-01-10 22:04:08,938 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:96)
2019-01-10 22:04:08,938 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:97)
2019-01-10 22:04:08,938 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:98)
2019-01-10 22:04:08,941 | INFO | main | shardNum:5 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:99)
2019-01-10 22:04:08,941 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:100)
2019-01-10 22:04:08,942 | INFO | main | isSecureMode:false |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:101)
2019-01-10 22:04:08,942 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProperties(HighLevelRestClientAllRequests.java:102)
2019-01-10 22:04:12,035 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:04:12,035 | INFO | main | systemSecConfig is false |
```

```
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:04:12,035 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:04:12,037 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:04:12,038 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:04:12,038 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:04:12,812 | INFO | main | The High Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.getHigh
LevelRestClient(HighLevelRestClientAllRequests.java:176)
2019-01-10 22:04:13,030 | INFO | main | ClusterName:[elasticsearch_cluster], clusterUuid:
[TTTr6K7laQKuY3yZU28QiZg], nodeName:[EsNode1@192.168.61.68], version:[6.1.3]. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.queryCl
usterInfo(HighLevelRestClientAllRequests.java:188)
2019-01-10 22:04:13,405 | INFO | main | IndexByJson response is
IndexResponse[index=huawei1,type=type1,id=1,version=1,result=created,seqNo=0,primaryTerm=1,shards={"t
otal":2,"successful":1,"failed":0}] |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.indexBy
son(HighLevelRestClientAllRequests.java:208)
2019-01-10 22:04:13,545 | INFO | main | IndexByMap response is
IndexResponse[index=huawei1,type=type1,id=2,version=1,result=created,seqNo=0,primaryTerm=1,shards={"t
otal":2,"successful":1,"failed":0}] |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.indexBy
Map(HighLevelRestClientAllRequests.java:232)
2019-01-10 22:04:13,595 | INFO | main | IndexByXContentBuilder response is
IndexResponse[index=huawei1,type=type1,id=3,version=1,result=created,seqNo=0,primaryTerm=1,shards={"t
otal":2,"successful":1,"failed":0}] |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.indexBy
XContentBuilder(HighLevelRestClientAllRequests.java:263)
2019-01-10 22:04:13,683 | INFO | main | Update response is
UpdateResponse[index=huawei1,type=type1,id=1,version=2,seqNo=1,primaryTerm=1,result=updated,shards=
ShardInfo[total=2, successful=1, failures=[]]] |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.update(
HighLevelRestClientAllRequests.java:285)
2019-01-10 22:04:13,857 | INFO | main | Bulk is successful |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.bulk(Hi
ghLevelRestClientAllRequests.java:313)
2019-01-10 22:04:13,905 | INFO | main | GetIndex response is
{"_index":"huawei1","_type":"type1","_id":"1","found":false} |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.getIndex(HighLevelRestClientAllRequests.java:336)
2019-01-10 22:04:14,022 | INFO | main | SearchIndex response is {"took":1,"timed_out":false,"_shards": {
"total":5,"successful":5,"skipped":0,"failed":0}, "hits":{"total":0,"max_score":null,"hits":[]}} |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.search(HighLevelRestClientAllRequests.java:373)
2019-01-10 22:04:14,077 | INFO | main | SearchHits is
{"_scroll_id":"DnF1ZXJ5VGhbkZldGNoBQAAAAAAAAAQFkw1aGxQeF9xU2wycjAwRkxFSDBjaHcAAAAAAA
EhZMNWhsUHhfCVNsMnlwMEZMRUgwY2h3AAAAAAAABEWTDvobFB4X3FTbDjyMDBGEVIMGNodwAAAA
AAAAATFkw1aGxQeF9xU2wycjAwRkxFSDBjaHcAAAAAAAABZMNWhsUHhfCVNsMnlwMEZMRUgwY2h3","_
took":10,"timed_out":false,"_shards":{"total":5,"successful":5,"skipped":0,"failed":0}, "hits": {
"total":0,"max_score":null,"hits":[]}} |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.searchS
croll(HighLevelRestClientAllRequests.java:395)
2019-01-10 22:04:14,133 | INFO | main | ClearScroll is successful. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.clearScr
oll(HighLevelRestClientAllRequests.java:424)
2019-01-10 22:04:14,224 | INFO | main | Delete index is successful |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.deleteIn
dex(HighLevelRestClientAllRequests.java:443)
```

After a complete sample of the JDBC Rest Client is run, some operation results are displayed on the console:

```
2022-03-31 10:33:33,886 | INFO | main |
esServerHost:192.168.64.53:24100,192.168.64.55:24100,192.168.64.52:24100 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:173)
2022-03-31 10:33:33,892 | INFO | main | connectTimeout:5000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:174)
```

```
2022-03-31 10:33:33,893 | INFO | main | socketTimeout:60000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:175)
2022-03-31 10:33:33,895 | INFO | main | connectionRequestTimeout:100000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:176)
2022-03-31 10:33:33,897 | INFO | main | maxConnPerRouteTotal:100 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:177)
2022-03-31 10:33:33,897 | INFO | main | maxConnTotal:1000 |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:178)
2022-03-31 10:33:33,897 | INFO | main | isSecureMode:false |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:179)
2022-03-31 10:33:33,898 | INFO | main | sslEnabled:false |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:180)
2022-03-31 10:33:33,898 | INFO | main | principal:null |
org.elasticsearch.hwclient.HwRestClient.getConfig(HwRestClient.java:181)
2022-03-31 10:33:34,034 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:221)
2022-03-31 10:33:34,035 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:222)
2022-03-31 10:33:34,037 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:221)
2022-03-31 10:33:34,037 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:222)
2022-03-31 10:33:34,037 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.<init>(RestClientBuilder.java:245)
2022-03-31 10:33:34,039 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:221)
2022-03-31 10:33:34,039 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:222)
2022-03-31 10:33:37,568 | INFO | main | Create index successful by high level client. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.createIndexForHighLevel(SqlMain.java:77)
2022-03-31 10:33:38,067 | INFO | main | Create index successful by high level client. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.createIndexForHighLevel(SqlMain.java:77)
2022-03-31 10:33:39,204 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:39,658 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:40,125 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:40,344 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:40,530 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:40,798 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:40,984 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:41,175 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:41,382 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:41,742 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:42,258 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:42,421 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:42,580 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:42,772 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:42,910 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:43,059 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:43,159 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:43,328 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:43,457 | INFO | main | Bulk is successful. |
```

```
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:43,557 | INFO | main | Bulk is successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain.insertData(SqlMain.java:58)
2022-03-31 10:33:43,969 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.basicSearch(SqlSearch.java:30)
2022-03-31 10:33:44,149 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.insteadFieldSearch(SqlSearch.java:57)
2022-03-31 10:33:44,252 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.distinctSearch(SqlSearch.java:71)
2022-03-31 10:33:44,387 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.whereAndLimitSearch(SqlSearch.java:86)
2022-03-31 10:33:44,517 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.groupByAndAliasSearch(SqlSearch.java:100)
2022-03-31 10:33:45,308 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.functionsSearch(SqlSearch.java:114)
2022-03-31 10:33:45,451 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.havingSearch(SqlSearch.java:129)
2022-03-31 10:33:45,583 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.orderBySearch(SqlSearch.java:143)
2022-03-31 10:33:46,101 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.joinSearch(SqlSearch.java:158)
2022-03-31 10:33:46,213 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.subQuerySearch(SqlSearch.java:174)
2022-03-31 10:33:46,336 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.matchSearch(SqlSearch.java:188)
2022-03-31 10:33:46,511 | INFO | main | Search succeed. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlSearch.querySearch(SqlSearch.java:203)
2022-03-31 10:33:46,644 | INFO | main | Delete data successful. |
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlDelete.deleteData(SqlDelete.java:36)
2022-03-31 10:33:46,850 | INFO | main | Delete index is successful. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.delete.DeleteIndex.deleteIndex(DeleteIndex.java:36)
)
2022-03-31 10:33:47,103 | INFO | main | Delete index is successful. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.delete.DeleteIndex.deleteIndex(DeleteIndex.java:36)
```

Process finished with exit code 0

## Log Description

The log level is **INFO** by default. You can view more detailed information by changing the log level (**TRACE**, **DEBUG**, **INFO**, **WARN**, and **ERROR**). You can modify the **log4j.properties** file to change log levels, for example:

```
# This sets the global logging level and specifies the appenders
log4j.rootLogger=INFO,theConsoleAppender,R

# settings for the console appender
log4j.appender.theConsoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.theConsoleAppender.Threshold=INFO
log4j.appender.theConsoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.theConsoleAppender.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n

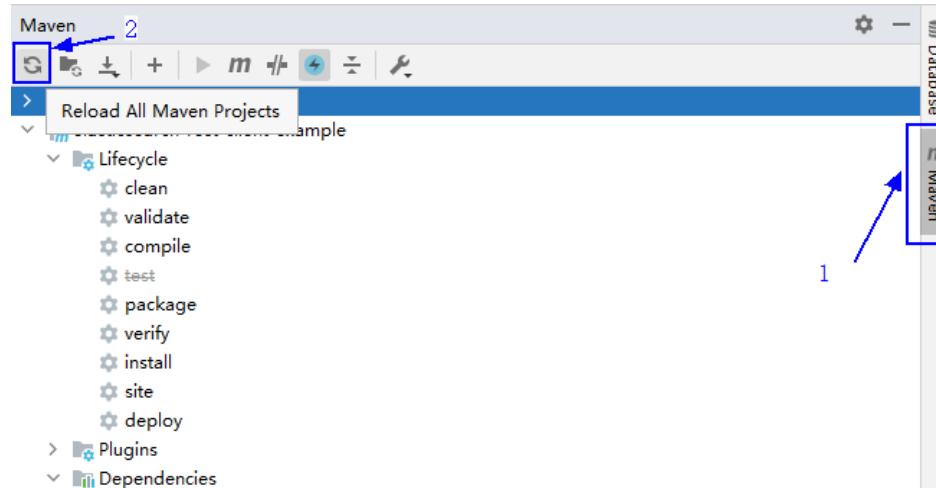
# R is set to be a File appender using a PatternLayout.
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.Append=true
log4j.appender.R.Threshold=debug
log4j.appender.R.MaxFileSize=10240KB
log4j.appender.R.MaxBackupIndex=10
log4j.appender.R.File=logs/es-example.log
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n
```

### 2.3.4.1.2 Commissioning a SpringBoot Program

#### Compiling and Running Applications

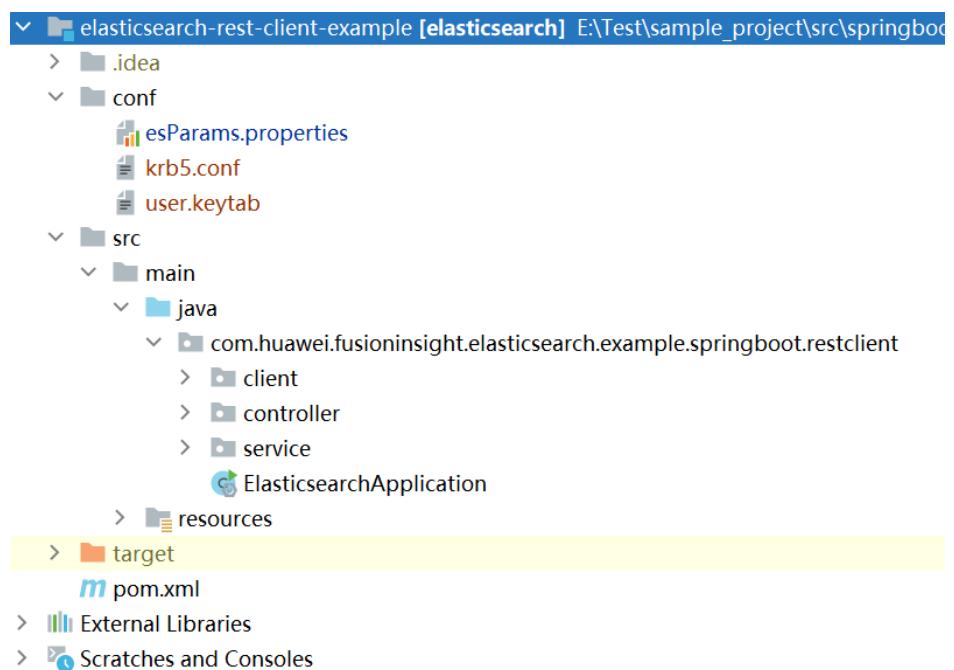
You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

1. Click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependency.



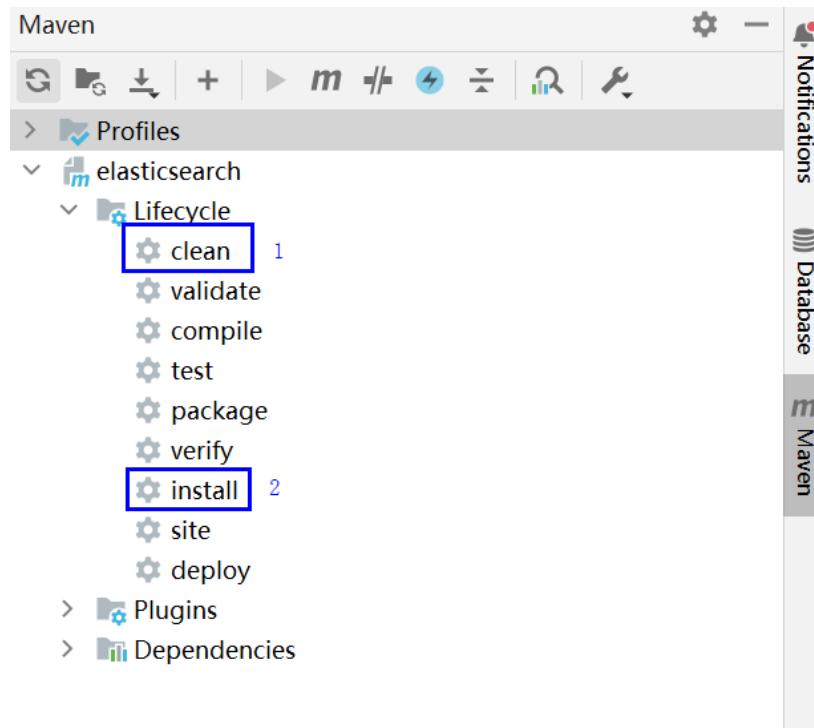
2. Compile and run an application.

- a. The following figure shows the file list after the configuration file has been placed and the code has been modified to match the login user. (Authentication files **krb5** and **keytab** are not required in normal clusters.)



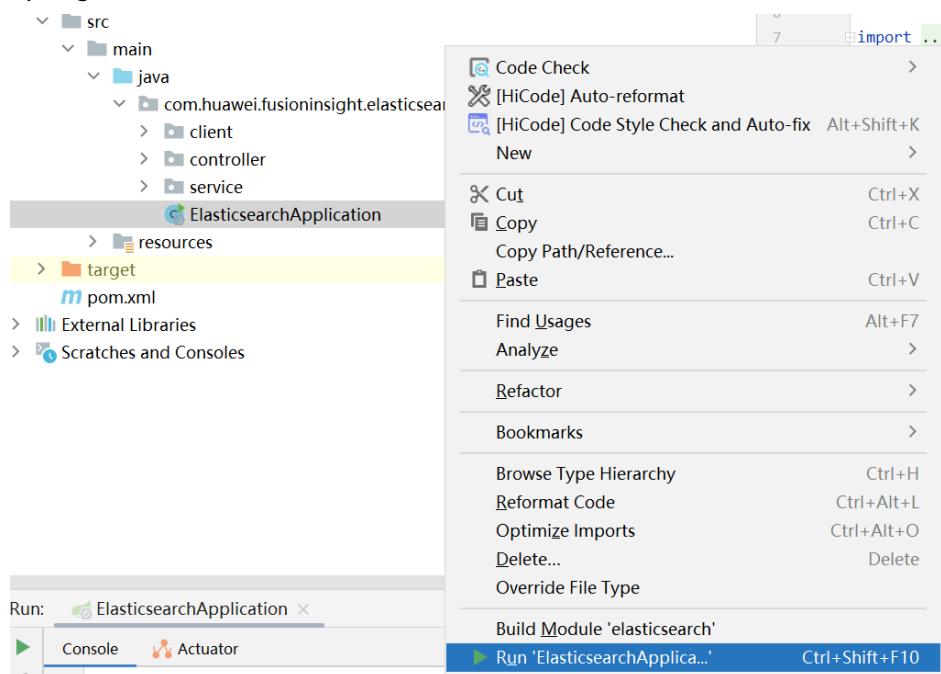
- b. Compile an application.

Double-click **clean** and **install** to run the **maven clean** and **maven install** commands. After the compilation is complete, "Build Success" is displayed.



c. Run the program.

Right-click **ElasticsearchApplication** and choose **Run'ElasticsearchApplication...'** from the shortcut menu to start the SpringBoot service.



3. Call the SpringBoot sample API of Elasticsearch to trigger the running of the sample code.

After the SpringBoot service is started, enter the link of the specific operation to be performed in the address box of the browser. For example, to query the cluster health status, enter **http://localhost:8080/elasticsearch/cluster/health** in the address box of the browser. The following result is returned:

```
{"cluster_name":"elasticsearch_cluster","status":"green","timed_out":false,"number_of_nodes":6,"number_of_data_nodes":3,"active_primary_shards":11,"active_shards":22,"relocating_shards":0,"initializing_shards":0,"unassigned_shards":0,"delayed_unassigned_shards":0,"number_of_pending_tasks":0,"number_of_in_flight_fetch":0,"task_max_waiting_in_queue_millis":0,"active_shards_percent_as_number":100.0}
```

**Table 2-12** lists the operations that can be performed when Elasticsearch interconnects with SpringBoot.

**Table 2-12** Operations for interconnecting Elasticsearch with SpringBoot

| Method                 | Browser Link   | Description  |
|------------------------|--|--|
| getElasticsearch       | http://localhost:8080/elasticsearch/   | Obtain cluster information such as clusterName, clusterUuid, and nodeName. |
| clusterHealth          | http://localhost:8080/elasticsearch/cluster/health   | Obtain the cluster health status.  |
| indexByJson            | http://localhost:8080/elasticsearch/indexByJson?index= <i>Index name</i>                   | Write data in JSON format.   |
| indexByMap             | http://localhost:8080/elasticsearch/indexByMap?index= <i>Index name</i>                    | Write data in the Map format.  |
| indexByXContentBuilder | http://localhost:8080/elasticsearch/indexByXContentBuilder?index= <i>Index name</i>        | Write data in the XContentBuilder format.                                  |
| update                 | http://localhost:8080/elasticsearch/update?index= <i>Index name</i> &id= <i>ID value</i>   | Update index data.   |
| bulk                   | http://localhost:8080/elasticsearch/bulk?index= <i>Index name</i>                          | Write data in batches.   |
| getIndex               | http://localhost:8080/elasticsearch/getIndex?index= <i>Index name</i> &id= <i>ID value</i> | Query index information.   |
| search                 | http://localhost:8080/elasticsearch/search?index= <i>Index name</i>                        | Search for related document information under a specified index.           |

| Method      | Browser Link   | Description   |
|-------------|--|---|
| scroll      | <code>http://localhost:8080/elasticsearch/scroll?index=/index name</code>      | Search for document information under a specified index using a cursor. You can disable the cursor by modifying <b>scrollId</b> . |
| deleteIndex | <code>http://localhost:8080/elasticsearch/deleteIndex?index=/index name</code> | Delete an index.  |

## 2.3.4.2 Commissioning Applications in Linux

### 2.3.4.2.1 Commissioning a RestClient Application

Elasticsearch application programs can run in a Linux environment. After the code development is complete, you can upload a .jar package to the prepared Linux running environment, to run the applications.

#### Prerequisites

JDK is installed in the Linux environment. The version must be the same as that of the JDK used to export the .jar package from IntelliJ IDEA, and Java environment variables have been set.

### Compiling and Running the Program

#### Step 1 Export a JAR package.

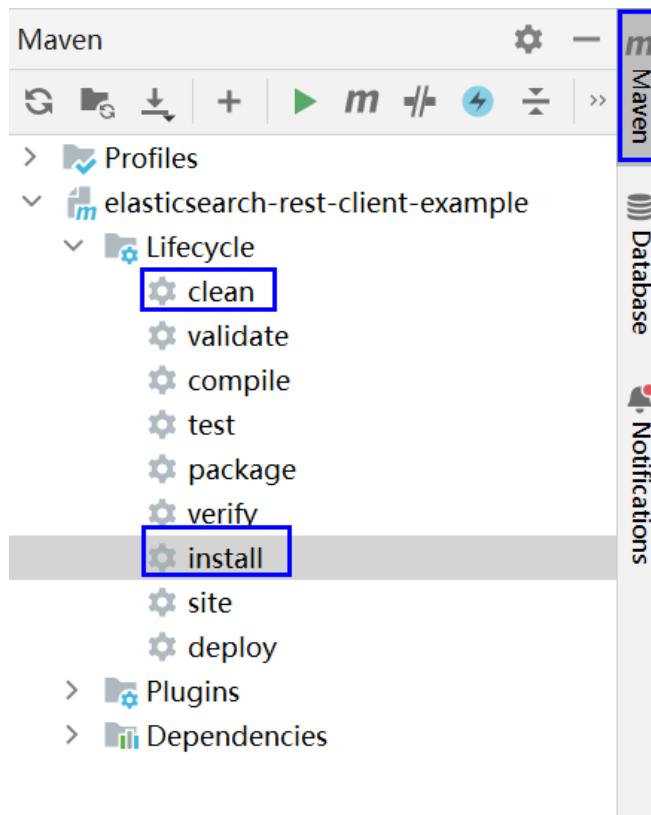
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 2-32 Maven tools: clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 2-33 Enter mvn clean compile in the IDEA Terminal text box.



After the compilation is completed, the message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

### Step 2 Generate the **libs** on which the running depends.

Go to the local root directory of the project and run the following command in the Windows command prompt window to generate the **libs** on which the running depends:

**mvn dependency:copy-dependencies -DoutputDirectory=libs**

### Step 3 Run the .jar package

- Copy the directories **libs** and **conf** required for the development environment to any directory (For example, /opt/elasticsearch-example) in the Linux

operating environment. Then, copy the **.jar** package generated in the application environment to **/opt/elasticsearch-rest-client-example/libs/**.

 NOTE

Modify the **esParams.properties** file in the **conf** directory based on the site requirements. For details, see [Table 2-10](#).

2. Switch to the code directory cd /opt/elasticsearch-rest-client-example.

3. Run the following Java command:

- Run Low Level RestClient code

```
java -cp /opt/elasticsearch-rest-client-example/conf/*:/opt/
elasticsearch-rest-client-example/libs/*
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests
```

- Run High Level RestClient code

```
java -cp /opt/elasticsearch-rest-client-example/conf/*:/opt/
elasticsearch-rest-client-example/libs/*
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests
```

- Run JDBC Rest Client code

```
java -cp /opt/elasticsearch-rest-client-example/conf/*:/opt/
elasticsearch-rest-client-example/libs/*
com.huawei.fusioninsight.elasticsearch.example.jdbc.SqlMain
```

----End

## Viewing the Commissioning Result

After the Elasticsearch ends running, you can view the running status using either of the following methods:

- View the running result.
- View Elasticsearch logs, that is, the **/opt/elasticsearch-rest-client-example/logs/es-example.log** log file in the **logs** directory.

After a complete sample of the Low Level RestClient is run, operation results are as follows:

```
2019-01-10 22:05:44,091 | INFO | main | Start to do low level rest client request ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.main(LowLevelRestClientAllRequests.java:429)
2019-01-10 22:05:44,096 | INFO | main | esServerHost:10.1.1.1:24100 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:73)
2019-01-10 22:05:44,097 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:74)
2019-01-10 22:05:44,097 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:75)
2019-01-10 22:05:44,098 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:76)
2019-01-10 22:05:44,098 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:77)
2019-01-10 22:05:44,102 | INFO | main | shardNum:5 |
```

```
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:78)
2019-01-10 22:05:44,102 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:79)
2019-01-10 22:05:44,102 | INFO | main | isSecureMode:false |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:80)
2019-01-10 22:05:44,103 | INFO | main | oneCommit:5 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:81)
2019-01-10 22:05:44,103 | INFO | main | totalRecordNumber:10 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:82)
2019-01-10 22:05:44,103 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.initProperties(LowLevelRestClientAllRequests.java:83)
2019-01-10 22:05:47,246 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:05:47,246 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:05:47,246 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:05:47,249 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:05:47,251 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:05:47,251 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:05:47,426 | INFO | main | The Low Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.getRestClient(LowLevelRestClientAllRequests.java:162)
2019-01-10 22:05:47,587 | INFO | main | QueryClusterInfo successful. |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.queryClusterInfo(LowLevelRestClientAllRequests.java:178)
2019-01-10 22:05:47,588 | INFO | main | QueryClusterInfo response entity is : {
    "cluster_name" : "elasticsearch_cluster",
    "status" : "yellow",
    "timed_out" : false,
    "number_of_nodes" : 3,
    "number_of_data_nodes" : 1,
    "active_primary_shards" : 15,
    "active_shards" : 15,
    "relocating_shards" : 0,
    "initializing_shards" : 0,
    "unassigned_shards" : 15,
    "delayed_unassigned_shards" : 0,
    "number_of_pending_tasks" : 0,
    "number_of_in_flight_fetch" : 0,
    "task_max_waiting_in_queue_millis" : 0,
    "active_shards_percent_as_number" : 50.0
}
|
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.queryClusterInfo(LowLevelRestClientAllRequests.java:182)
2019-01-10 22:05:47,648 | INFO | main | Index is not exist : huawei1 |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.indexIsExist(LowLevelRestClientAllRequests.java:202)
2019-01-10 22:05:47,823 | INFO | main | CreateIndexWithShardNum successful. |
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.createIndexWithShardNum(LowLevelRestClientAllRequests.java:227)
2019-01-10 22:05:47,824 | INFO | main | CreateIndexWithShardNum response entity is : {
    "acknowledged" : true,
    "shards_acknowledged" : true,
    "index" : "huawei1"
}
|
com.huawei.fusioninsight.elasticsearch.example.lowlevel.allrequests.LowLevelRestClientAllRequests.createIndexWithShardNum(LowLevelRestClientAllRequests.java:231)
```

After a complete sample of the High Level RestClient is run, some operation results are displayed on the console:

```
2019-01-10 22:04:08,933 | INFO | main | Start to do high level rest client request ! |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.main(Hi
ghLevelRestClientAllRequests.java:455)
2019-01-10 22:04:08,937 | INFO | main | esServerHost:10.1.1.1:24100 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:94)
2019-01-10 22:04:08,937 | INFO | main | maxRetryTimeoutMillis:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:95)
2019-01-10 22:04:08,938 | INFO | main | connectTimeout:5000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:96)
2019-01-10 22:04:08,938 | INFO | main | socketTimeout:60000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:97)
2019-01-10 22:04:08,938 | INFO | main | connectionRequestTimeout:3000 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:98)
2019-01-10 22:04:08,941 | INFO | main | shardNum:5 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:99)
2019-01-10 22:04:08,941 | INFO | main | replicaNum:1 |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:100)
2019-01-10 22:04:08,942 | INFO | main | isSecureMode:false |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:101)
2019-01-10 22:04:08,942 | INFO | main | principal:esuser@<system domain name> |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.initProp
erties(HighLevelRestClientAllRequests.java:102)
2019-01-10 22:04:12,035 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:04:12,035 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:04:12,035 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:04:12,037 | INFO | main | esSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:181)
2019-01-10 22:04:12,038 | INFO | main | systemSecConfig is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:182)
2019-01-10 22:04:12,038 | INFO | main | isSecureMode is false |
org.elasticsearch.client.RestClientBuilder.refreshSecureMode(RestClientBuilder.java:183)
2019-01-10 22:04:12,812 | INFO | main | The High Level Rest Client has been created ! |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.getHigh
LevelRestClient(HighLevelRestClientAllRequests.java:176)
2019-01-10 22:04:13,030 | INFO | main | ClusterName:[elasticsearch_cluster], clusterUuid:
[TTTr6K7laQKuY3yZU28QiZg], nodeName:[EsNode1@192.168.61.68], version:[6.1.3]. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.queryCl
usterInfo(HighLevelRestClientAllRequests.java:188)
2019-01-10 22:04:13,405 | INFO | main | IndexByJson response is
IndexResponse[index=huawei1,type=type1,id=1,version=1,result=created,seqNo=0,primaryTerm=1,shards={"t
otal":2,"successful":1,"failed":0}]. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.indexBy
Json(HighLevelRestClientAllRequests.java:208)
2019-01-10 22:04:13,545 | INFO | main | IndexByMap response is
IndexResponse[index=huawei1,type=type1,id=2,version=1,result=created,seqNo=0,primaryTerm=1,shards={"t
otal":2,"successful":1,"failed":0}]. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.indexBy
Map(HighLevelRestClientAllRequests.java:232)
2019-01-10 22:04:13,595 | INFO | main | IndexByXContentBuilder response is
IndexResponse[index=huawei1,type=type1,id=3,version=1,result=created,seqNo=0,primaryTerm=1,shards={"t
otal":2,"successful":1,"failed":0}]. |
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.indexBy
XContentBuilder(HighLevelRestClientAllRequests.java:263)
2019-01-10 22:04:13,683 | INFO | main | Update response is
UpdateResponse[index=huawei1,type=type1,id=1,version=2,seqNo=1,primaryTerm=1,result=updated,shards=
```

```
ShardInfo{total=2, successful=1, failures=[]}]. |  
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.update(  
HighLevelRestClientAllRequests.java:285)  
2019-01-10 22:04:13,857 | INFO | main | Bulk is successful |  
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.bulk(Hi  
ghLevelRestClientAllRequests.java:313)  
2019-01-10 22:04:13,905 | INFO | main | GetIndex response is  
{"_index":"huawei1","_type":"type1","_id":"1","found":false}|  
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.getIndex(HighLevelRestClientAllRequests.java:336)  
2019-01-10 22:04:14,022 | INFO | main | SearchIndex response is {"took":1,"timed_out":false,"_shards":  
{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":0,"max_score":null,"hits":[]}}.|  
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.search(  
HighLevelRestClientAllRequests.java:373)  
2019-01-10 22:04:14,077 | INFO | main | SearchHits is  
{"_scroll_id":"DnF1ZXJ5VGhlbkZldGN0BQAAAAAAAAAQFkw1aGxQeF9xU2wycjAwRkxFSDBjaHcAAAAAAA  
EhZMNWhsUHhfcVNsMnlwMEZMRUgwY2h3AAAAAAAABEWTDVobFB4X3FTbDJyMDBGTEVIMGNodwAAAA  
AAAAATFkw1aGxQeF9xU2wycjAwRkxFSDBjaHcAAAAAAAABZMNWhsUHhfcVNsMnlwMEZMRUgwY2h3","  
took":10,"timed_out":false,"_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":  
{"total":0,"max_score":null,"hits":[]}}|  
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.searchS  
croll(HighLevelRestClientAllRequests.java:395)  
2019-01-10 22:04:14,133 | INFO | main | ClearScroll is successful. |  
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.clearScr  
oll(HighLevelRestClientAllRequests.java:424)  
2019-01-10 22:04:14,224 | INFO | main | Delete index is successful |  
com.huawei.fusioninsight.elasticsearch.example.highlevel.allrequests.HighLevelRestClientAllRequests.deleteIn  
dex(HighLevelRestClientAllRequests.java:443)
```

### 2.3.4.2.2 Commissioning a SpringBoot Program

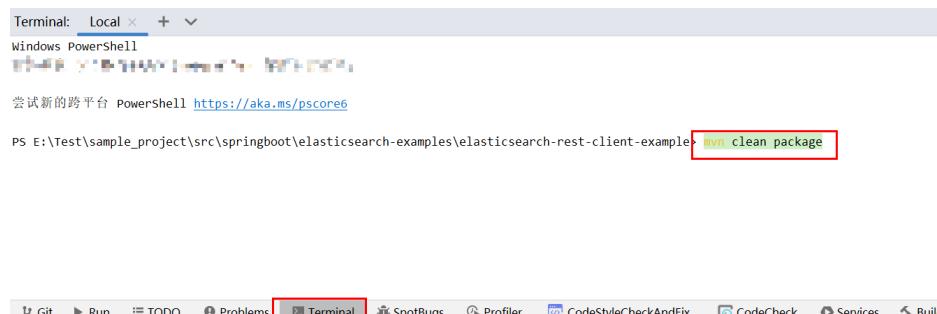
Applications that interconnect Elasticsearch with SpringBoot can run in the Linux environment. After the application code is developed, you can upload the JAR file to the prepared Linux environment.

## Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

## Compiling and Running Applications

**Step 1** Click Terminal in the lower left corner of the IDEA page to access the terminal. Run the **mvn clean package** command to compile the package.



If the command output contains "Build Success", the compilation is successful, as shown in the following figure. After the compilation is successful, the es-springboot.jar file is generated in the target directory of the sample project.

```
[INFO] -- maven-jar-plugin:3.2.2:jar (default-jar) @ elasticsearch ...
[INFO] Building jar: E:\Test\sample_project\src\springboot\elasticsearch-examples\elasticsearch-rest-client-example\target\es-springboot.jar
[INFO]
[INFO] -- spring-boot-maven-plugin:1.3.2.RELEASE:repackage (repackage) @ elasticsearch ...
[INFO]
[INFO] -- spring-boot-maven-plugin:1.3.2.RELEASE:repackage (default) @ elasticsearch ...
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 4.836 s
[INFO] Finished at: 2023-02-27T10:34:23+08:00
[INFO]
PS E:\Test\sample_project\src\springboot\elasticsearch-examples\elasticsearch-rest-client-example>
```

**Step 2** Create a directory on Linux as the running directory, for example, /opt/es-springboot, and save the es-springboot.jar and conf folders in the target directory to the directory. Modify the esParams.properties file in the conf directory and save the user credential files krb5.conf and user.keytab of Elasticsearch to the conf directory.

#### NOTE

Modify the esParams.properties configuration file in the conf directory based on the site requirements. For details, see [Table 2-10](#).

**Step 3** Run the JAR file.

1. Switch to the code function directory.

**cd /opt/es-springboot**

2. Run the java command to start the SpringBoot service.

**java -jar es-springboot.jar conf/**

3. Start another terminal on the current node to access the Elasticsearch client.

**cd Client installation directory**

**source bigdata\_env**

**kinit esuser**

4. Perform the curl operation of Elasticsearch. For example, if you run the **curl http://localhost:8080/elasticsearch/cluster/health** command to query the cluster health status, the following information is displayed:

```
{"cluster_name":"elasticsearch_cluster","status":"green","timed_out":false,"number_of_nodes":6,"number_of_data_nodes":3,"active_primary_shards":11,"active_shards":22,"relocating_shards":0,"initializing_shards":0,"unassigned_shards":0,"delayed_unassigned_shards":0,"number_of_pending_tasks":0,"number_of_in_flight_fetch":0,"task_max_waiting_in_queue_millis":0,"active_shards_percent_as_number":100.0}
```

For details about other operations, see [Table 2-13](#).

**Table 2-13** Curl operations for interconnecting Elasticsearch with SpringBoot

| Method           | Curl Commands on the Linux Client                         | Description  |
|------------------|---|--|
| getElasticsearch | curl 'http://localhost:8080/elasticsearch/'               | Obtain cluster information such as clusterName, clusterUuid, and nodeName. |
| clusterHealth    | curl 'http://localhost:8080/elasticsearch/cluster/health' | Obtain the cluster health status.  |

| Method                 | Curl Commands on the Linux Client   | Description   |
|------------------------|---|---|
| indexByJson            | curl 'http://localhost:8080/elasticsearch/indexByJson?index=/Index name'            | Write data in JSON format.  |
| indexByMap             | curl 'http://localhost:8080/elasticsearch/indexByMap?index=/Index name'             | Write data in the Map format.   |
| indexByXcontentBuilder | curl 'http://localhost:8080/elasticsearch/indexByXContentBuilder?index=/Index name' | Write data in the XContentBuilder format.   |
| update                 | curl 'http://localhost:8080/elasticsearch/update?index=/Index name&id=/D value'     | Update index data.  |
| bulk                   | curl 'http://localhost:8080/elasticsearch/bulk?index=/Index name'                   | Write data in batches.  |
| getIndex               | curl 'http://localhost:8080/elasticsearch/getIndex?index=/Index name&id=/D value'   | Query index information.  |
| search                 | curl 'http://localhost:8080/elasticsearch/search?index=/Index name'                 | Search for related document information under a specified index.  |
| scroll                 | curl 'http://localhost:8080/elasticsearch/scroll?index=/Index name'                 | Search for document information under a specified index using a cursor. You can disable the cursor by modifying <b>scrollId</b> . |
| deleteIndex            | curl 'http://localhost:8080/elasticsearch/deleteIndex?index=/Index name'            | Delete an index.  |

----End

## 2.4 Flink Development Guide

### 2.4.1 Overview

## 2.4.1.1 Application Development

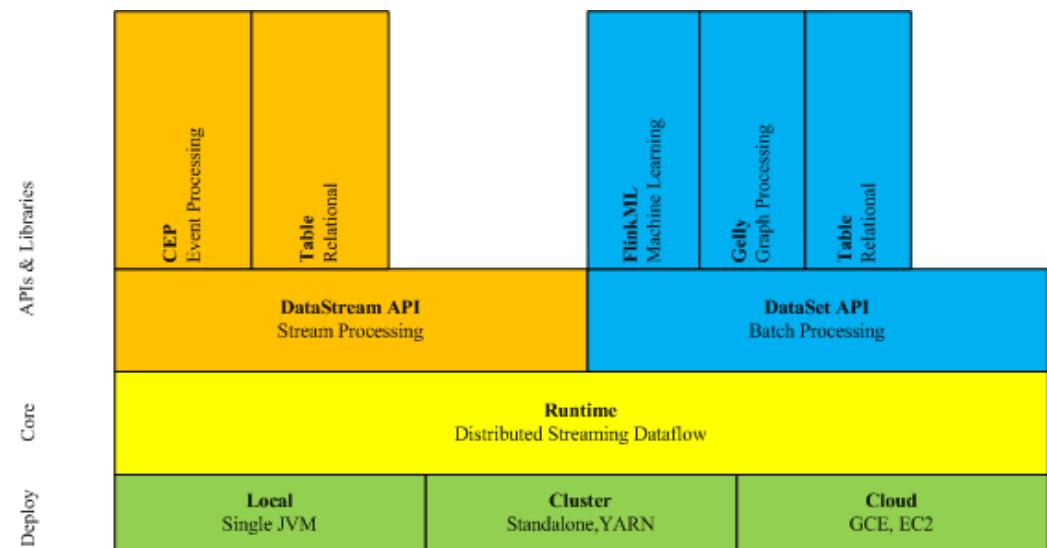
### Overview

Flink is a unified computing framework that supports both batch processing and stream processing. It provides a stream data processing engine that supports data distribution and parallel computing. Flink features stream processing and is a top open-source stream processing engine in the industry.

Flink provides high-concurrency pipeline data processing, millisecond-level latency, and high reliability, making it suitable for low-latency data processing.

[Figure 2-34](#) shows the technology stack of Flink.

**Figure 2-34** Technology stack of Flink



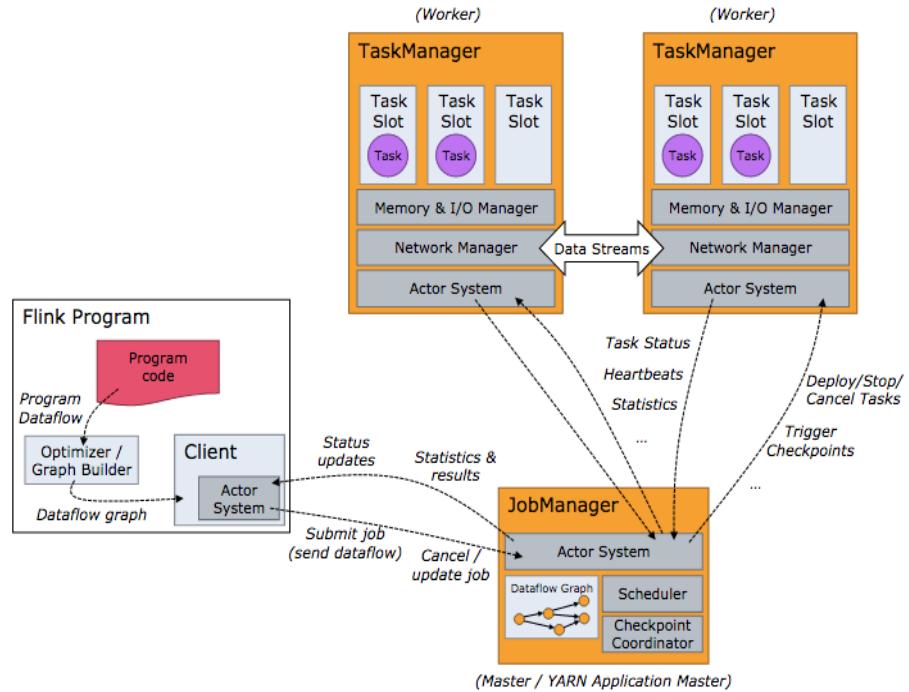
The following lists the key features of Flink in the current version:

- DataStream
- Checkpoint
- Window
- Job Pipeline
- Configuration Table

### Architecture

[Figure 2-35](#) shows the architecture of Flink.

**Figure 2-35 Flink architecture**



As shown in [Figure 2-35](#), the entire Flink system consists of three parts:

- **Client**  
Flink client is used to submit jobs (streaming jobs) to Flink.
- **TaskManager**  
TaskManager (also called worker) is a service execution node of Flink. It executes specific tasks. A Flink system could have multiple TaskManagers. These TaskManagers are equivalent to each other.
- **JobManager**  
JobManager (also called master) is a management node of Flink. It manages all TaskManagers and schedules tasks submitted by users to specific TaskManagers. In high-availability (HA) mode, multiple JobManagers are deployed. Among these JobManagers, one of which is selected as the leader, and the others are standby.

Flink provides the following features:

- **Low latency**  
Millisecond-level processing capability.
- **Exactly once**  
Asynchronous snapshot mechanism, ensuring that all data is processed only once.
- **High availability**  
Leader/Standy JobManagers, preventing single point of failure (SPOF).
- **Scale out**

Manual scale out supported by TaskManagers.

## Flink Development APIs

Flink DataStream API can be developed using Scala and Java languages, as shown in [Table 2-14](#).

**Table 2-14** Flink DataStream API

| Function  | Description  |
|-----------|--|
| Scala API | API in Scala, which can be used for data processing, such as filtering, joining, windowing, and aggregation. The Scala API is recommended for development because Scala is concise and easy to understand. |
| Java API  | API in Java, which can be used for data processing, such as filtering, joining, windowing, and aggregation.  |

### 2.4.1.2 Basic Concepts

#### Basic Concepts

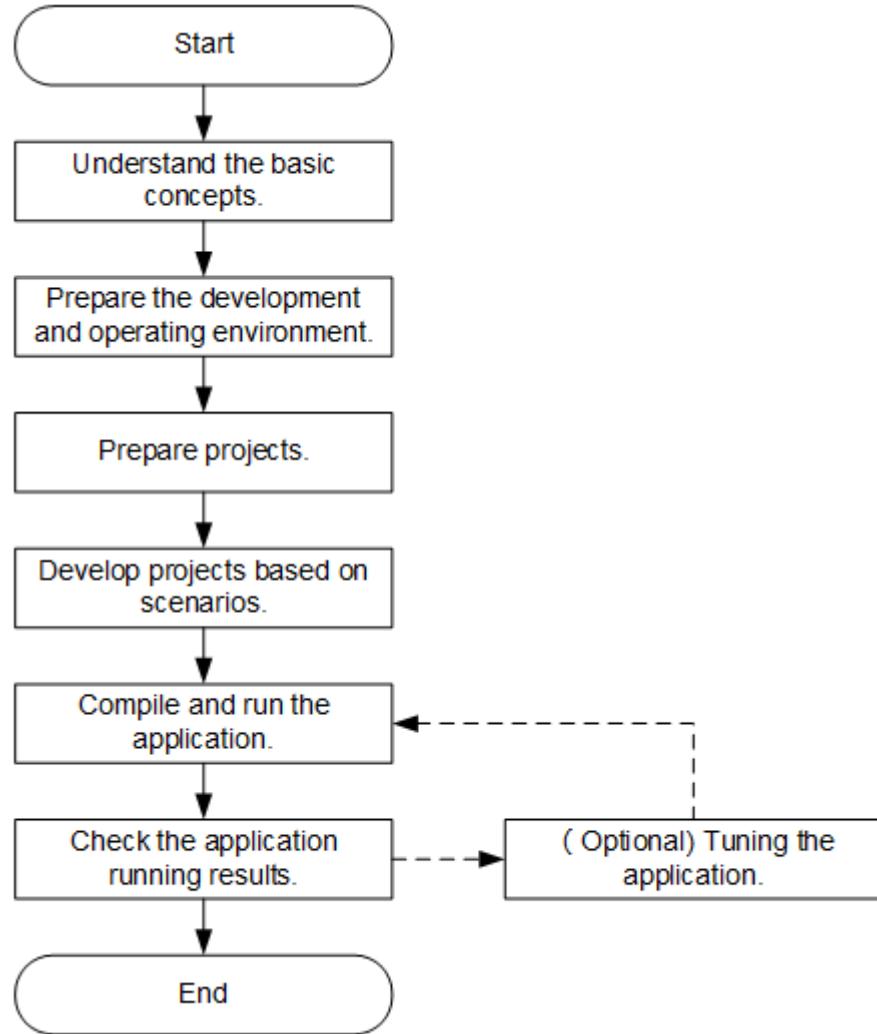
- **DataStream**  
DataStream is the minimum unit of Flink processing and is one of core concepts of Flink. DataStreams are initially imported from external systems in formats of socket, Kafka, and files. After being processed by Flink, DataStreams are exported to external systems in formats of socket, Kafka, and files.
- **Data Transformation**  
A data transformation is a data processing unit that transforms one or multiple DataStreams into a new DataStream.  
Data transformation can be classified as follows:
  - One-to-one transformation, for example, map.
  - One-to-zero, one-to-one, or one-to-multiple transformation, for example, flatMap.
  - One-to-zero or one-to-one transformation, for example, filter.
  - Multiple-to-one transformation, for example, union.
  - Transformation of multiple aggregations, for example, window and keyby.
- **Checkpoint**  
Checkpoint is the most important Flink mechanism to ensure reliable data processing. Checkpoints ensure that all application statuses can be recovered from a checkpoint in case of failure occurs and data is processed exactly once.
- **Savepoint**  
Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. After the upgrade, you can set the task status to the savepoint storage status and start the restoration, ensuring data continuity.

### 2.4.1.3 Development Process

#### Development Process of a Flink Application

[Figure 2-36](#) shows the Flink development process:

**Figure 2-36** Development process of a Flink application



**Table 2-15** Description of the development process

| Phase                          | Description   | Reference                      |
|--------------------------------|---|--------------------------------|
| Understand the basic concepts. | Before the development process, you are advised to gain a basic understanding of Flink. | <a href="#">Basic Concepts</a> |

| Phase  | Description   | Reference  |
|--|---|--|
| Prepare the development and operating environment. | Flink applications can be developed using Scala or Java. You are advised to use IntelliJ IDEA tool to configure the development environment as instructed, based on your development language.<br><br>The running environment of Flink is the Flink client. Install and configure the client as instructed. | <a href="#">Preparing the Development and Operating Environment</a>  |
| Prepare projects.                                  | Flink provides sample projects for you to import and learn. Alternatively, you can create a Flink project as instructed.  | <a href="#">Configuring and Importing Sample Projects</a><br><a href="#">Creating a Project (Optional)</a> |
| Develop the project.                               | Sample projects in Scala and Java are provided to help you quickly understand programming interfaces of Flink components.   | <a href="#">Developing an Application</a>  |
| Compile and run the application.                   | Guidance is provided for you to compile and run a developed application.  | <a href="#">Compiling and Running Applications</a>   |
| Check running results.                             | Application running results are stored in a path specified by you. You can also view application running status through Apache Flink Dashboard.   | <a href="#">Viewing the Debugging Result</a>   |
| Tune the application.                              | Tune the application to meet certain service requirements.<br><br>After the tuning is complete, the application needs to be compiled and run again.   | "Flink Performance Tuning" in the Component Operation Guide  |

## 2.4.2 Environment Preparation

### 2.4.2.1 Preparing the Development and Operating Environment

#### Preparing the Development Environment

[Table 2-16](#) describes the environment required for secondary development.

**Table 2-16** Development environment

| Item                                | Description   |
|-------------------------------------|---|
| OS                                  | <ul style="list-style-type: none"><li>• Development environment: Windows OS</li><li>• Operating environment: Linux OS<br/>If the program needs to be commissioned locally, the operating environment must be able to communicate with the cluster service plane.</li></ul>  |
| JDK installation                    | <p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul> |
| IDEA installation and configuration | Used for developing Flink applications. The required version is 14.1.7.   |
| Scala installation                  | Basic configuration for the Scala development environment. The required version is 2.11.7.  |
| Scala plug-in installation          | Basic configuration for the Scala development environment. The required version is 1.5.4.   |
| Maven installation                  | Basic configuration for the development environment. Maven is used for project management throughout the lifecycle of software development.   |

| Item    | Description  |
|---------|--|
| 7-zip   | Used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported. |
| Python3 | Used to run Flink Python jobs. Python 3.7 or later is required.            |

## Preparing the Operating Environment

During application development, prepare the code running and commissioning environment to verify that the application is running properly.

If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.

1. Install the client in a directory, for example, **/opt/hadoopclient**, on the node. Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
2. Log in to FusionInsight Manager. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Flink\config** directory to the **conf** directory where the compiled JAR file is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Flink/config/* root@IP address of the client node:/opt/hadoopclient/conf
```

[Table 2-17](#) describes the main configuration files.

**Table 2-17** Configuration files

| File            | Function                         |
|-----------------|----------------------------------|
| core-site.xml   | Configures Flink parameters.     |
| hdfs-site.xml   | Configures HDFS parameters.      |
| yarn-site.xml   | Configures YARN parameters.      |
| flink-conf.yaml | Flink client configuration file. |

3. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client is located, to ensure that the local host can communicate with each host in the cluster.

4. (Optional) To run a Python job, perform the following additional configurations:

- a. Log in to the node where the Flink client is installed as the **root** user and run the following command to check whether Python 3.7 or a later has been installed:

```
python3 -V
```

- b. Go to the python 3 installation path, for example, **/srv/pyflink-example**, and install the **virtualenv**:

```
cd /srv/pyflink-example
```

```
virtualenv venv --python=python3.x
```

```
source venv/bin/activate
```

- c. Copy the **Flink/flink/opt/python/apache-flink-\*.tar.gz** file from the client installation directory to **/srv/pyflink-example**:

```
cp /Client installation directory/Flink/flink/opt/python/apache-flink-*.tar.gz /srv/pyflink-example
```

- d. Install the dependency package. If the following command output is displayed, the installation is successful:

```
python -m pip install apache-flink-libraries-*.tar.gz
```

```
python -m pip install apache-flink- Version number.tar.gz
```

```
...
```

```
Successfully built apache-flink  
Installing collected packages: apache-flink  
Attempting uninstall: apache-flink  
    Found existing installation: apache-flink x.xx.x  
    Uninstalling apache-flink-x.xx.x:  
        Successfully uninstalled apache-flink-x.xx.x  
Successfully installed apache-flink-x.xx.x
```

#### 2.4.2.2 Configuring and Importing Sample Projects

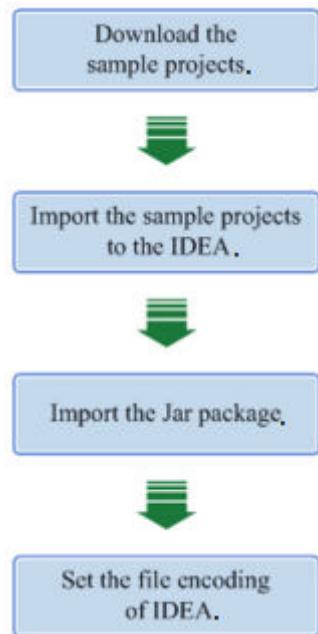
##### Scenario

Flink provides sample projects for multiple scenarios, including Java and Scala sample projects, to help you quickly learn Flink projects.

Methods to import Java and Scala projects are the same.

The following example describes how to import Java sample code. [Figure 2-37](#) shows the operation process.

Figure 2-37 Process of importing sample projects



## Procedure

**Step 1** Obtain the sample project folder **flink-examples-normal** in the **src\flink-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

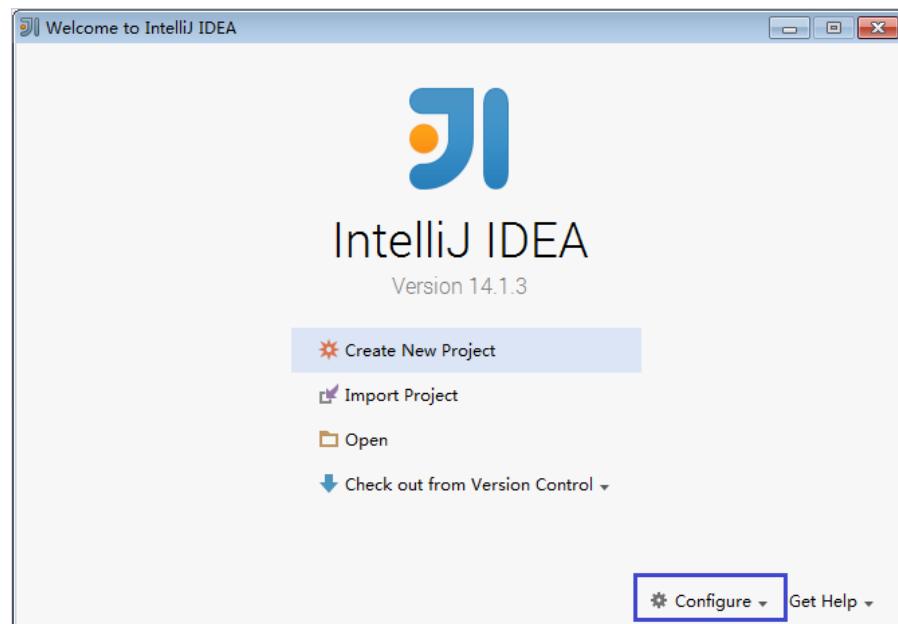
 NOTE

- In security mode, obtain the sample project **flink-examples-security** from the **src\flink-examples** folder.
- In normal mode, obtain the sample project **flink-examples-normal** from the **src\flink-examples** folder.

**Step 2** Before importing the sample project, configure JDK for IntelliJ IDEA.

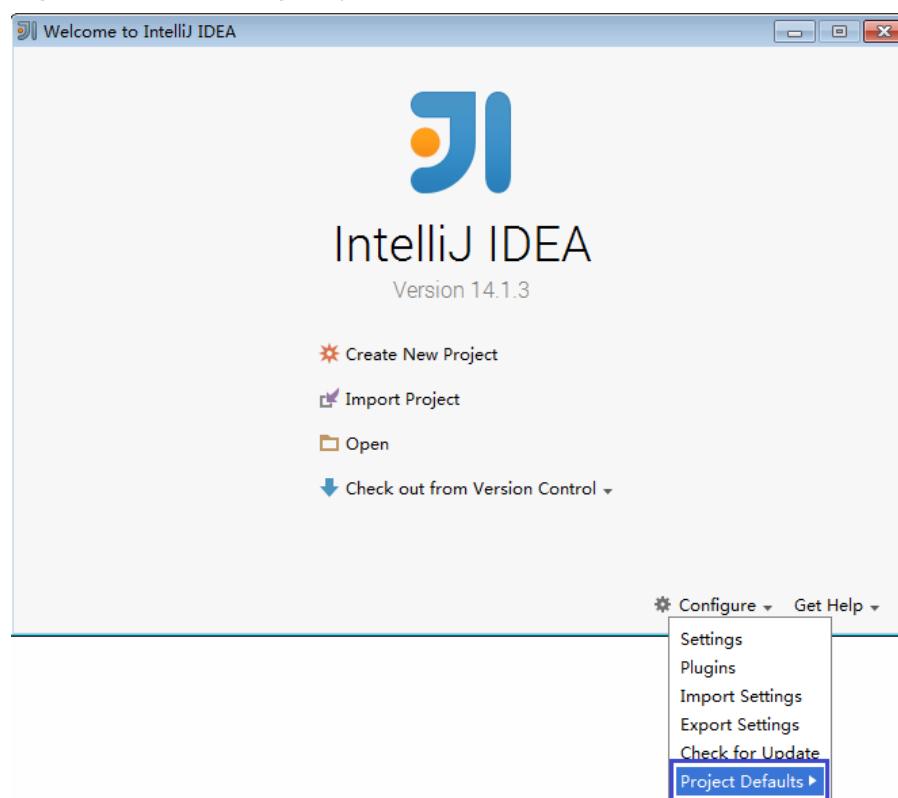
1. Start IntelliJ IDEA and click **Configure**.

Figure 2-38 Choosing Configure



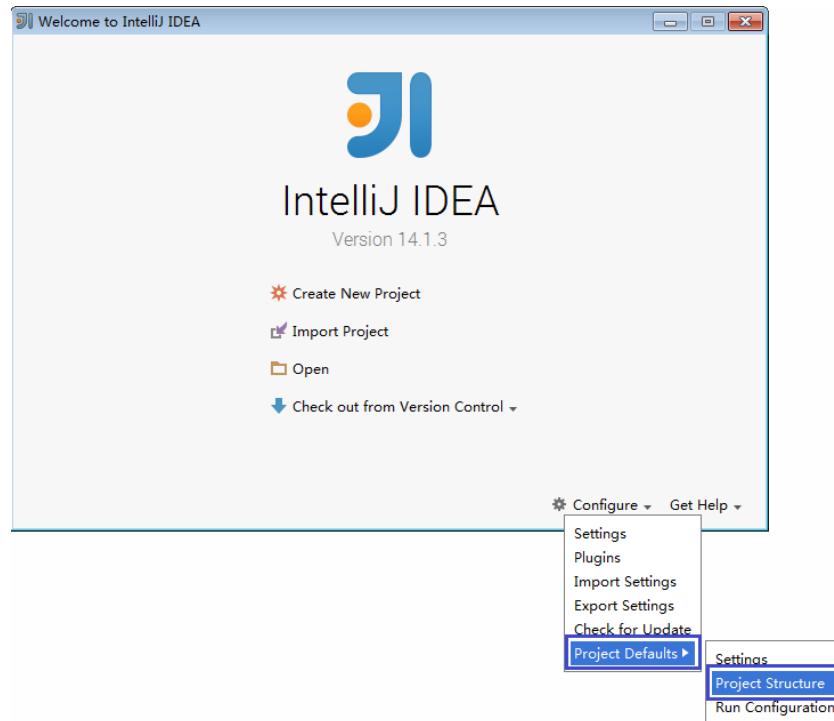
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 2-39 Choosing Project Defaults



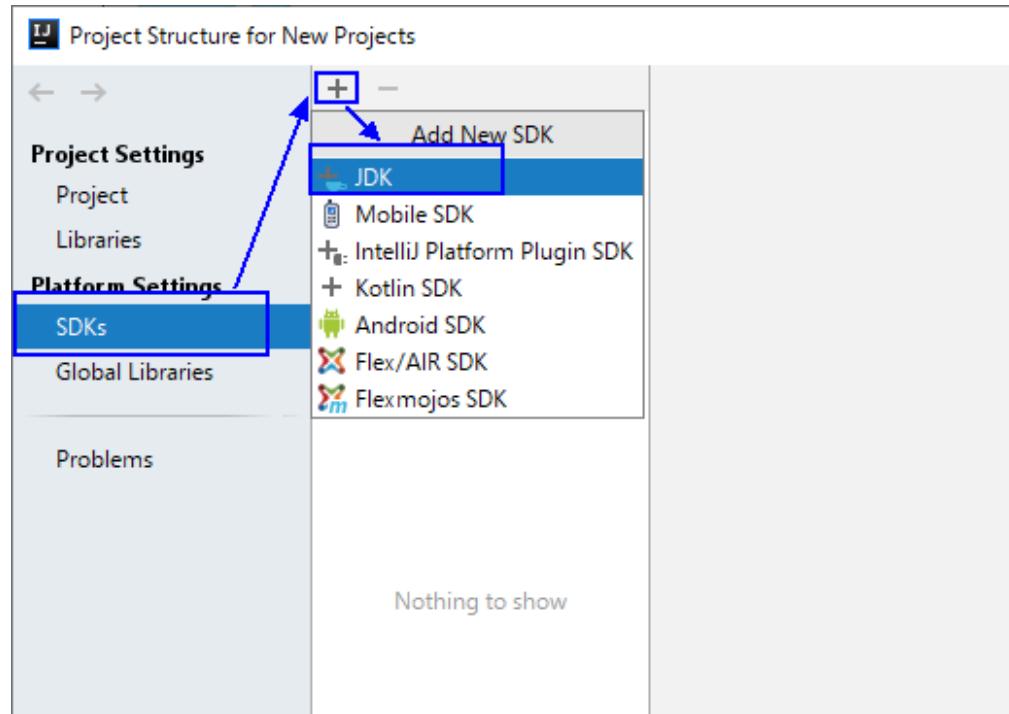
3. Choose **Project Structure** from the **Project Defaults** submenu.

Figure 2-40 Project Defaults



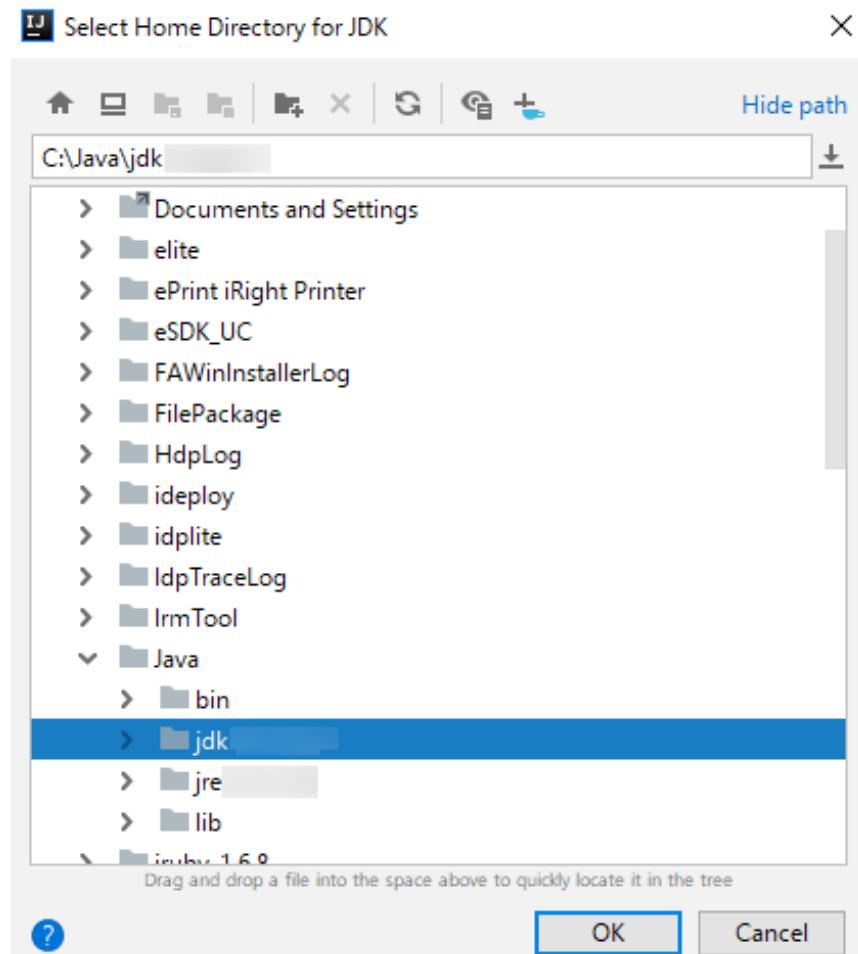
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 2-41 Adding the JDK



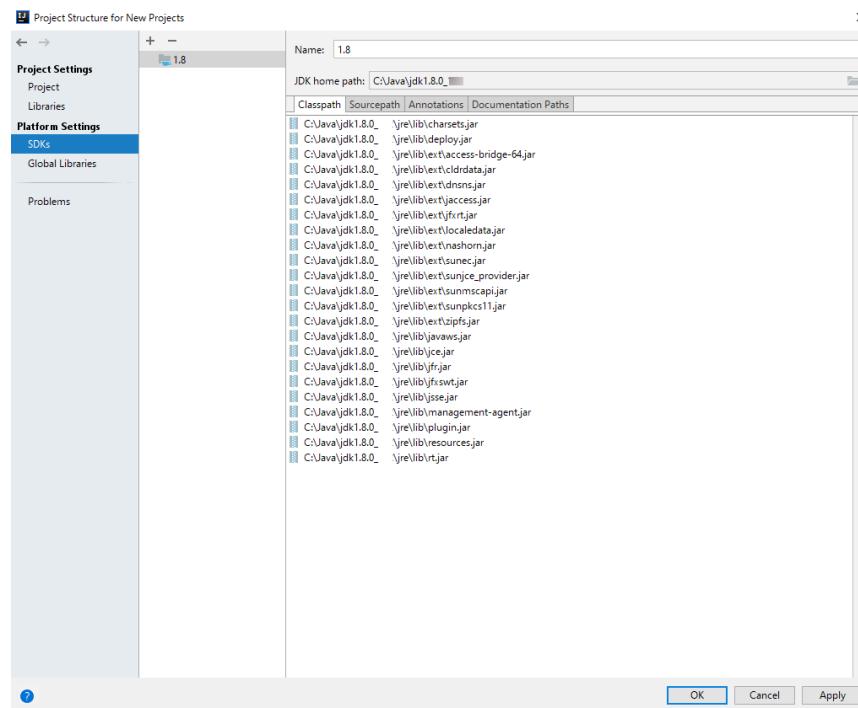
5. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 2-42 Selecting the JDK directory



6. After the JDK is selected, click **OK** to complete the configuration.

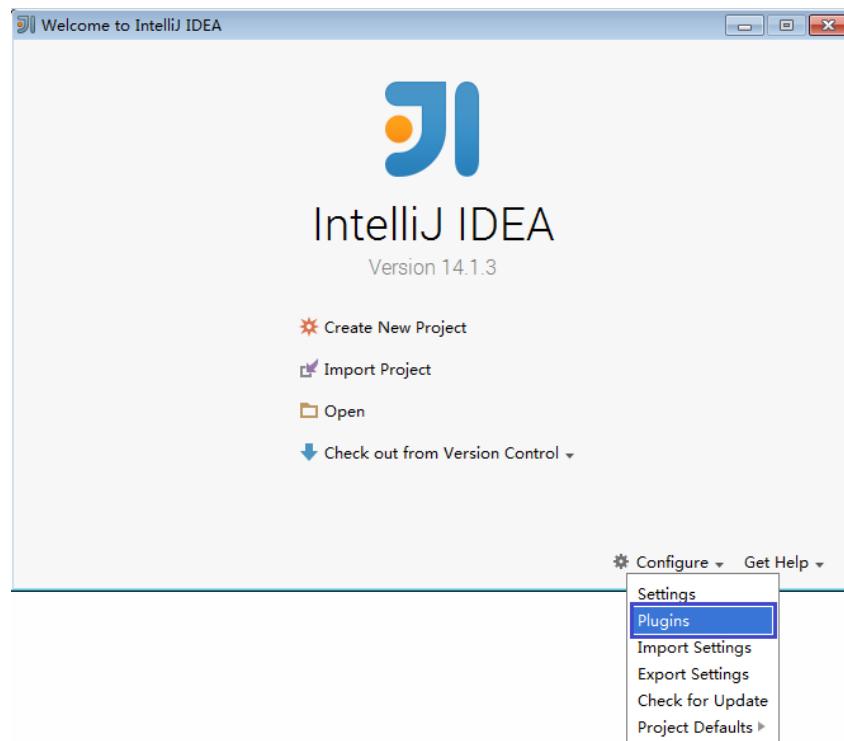
Figure 2-43 Completing the JDK configuration



**Step 3** (Optional) If a Scala sample project is imported, install Scala plug-ins in IntelliJ IDEA.

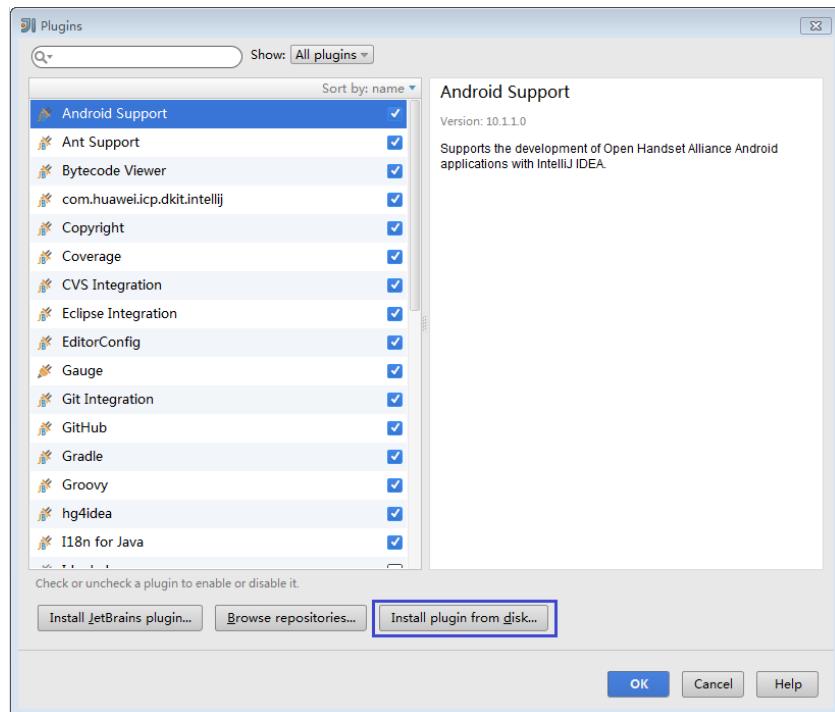
1. Choose **Plugins** from the **Configure** drop-down list.

Figure 2-44 Plugins



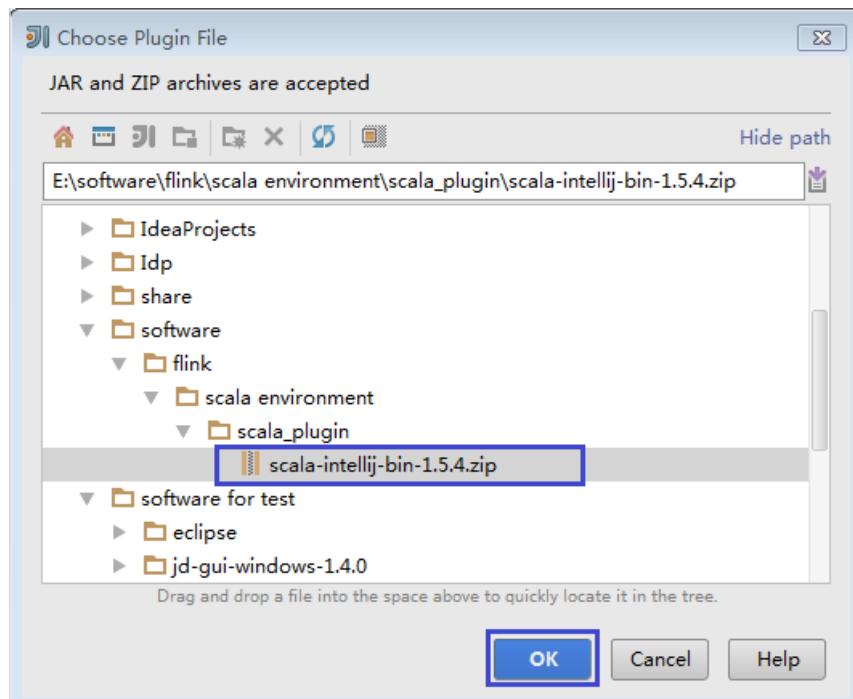
2. On the **Plugins** page, click **Install plugin from disk**.

Figure 2-45 Installing plugins from disk



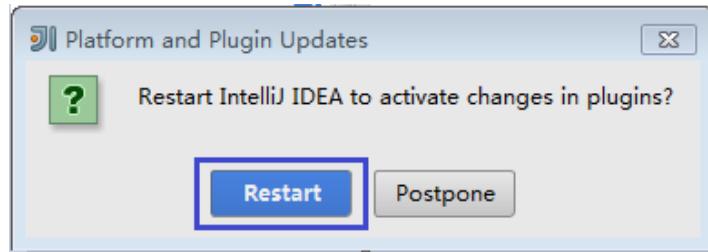
3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.

Figure 2-46 Choose Plugin File



4. On the **Plugins** page, click **Apply** to install the Scala plugins.
5. On the **Platform and Plugin Updates** page that is displayed, click **Restart** to make the configurations take effect.

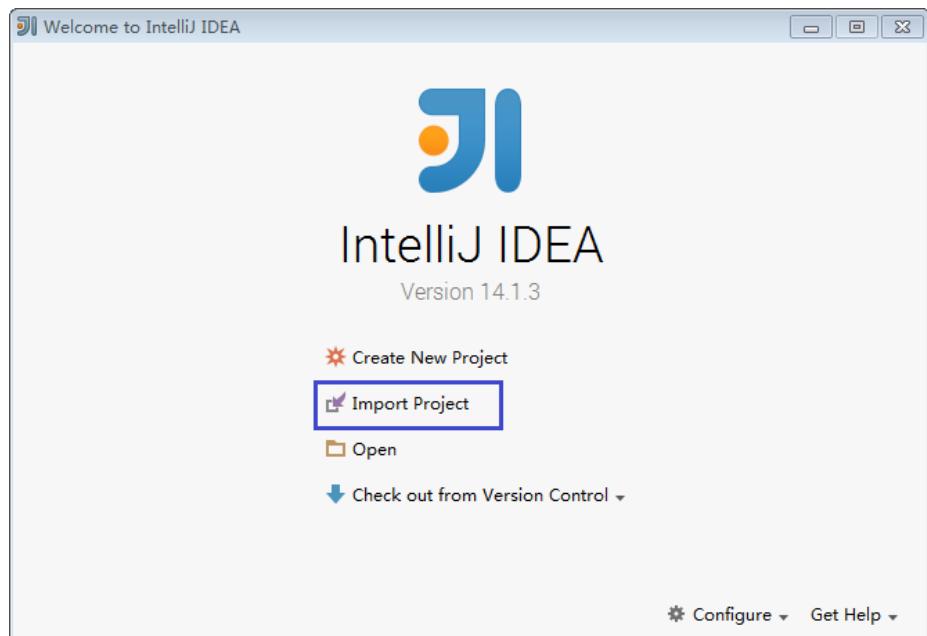
Figure 2-47 Platform and Plugin Updates



**Step 4** Import the Java sample project to IDEA.

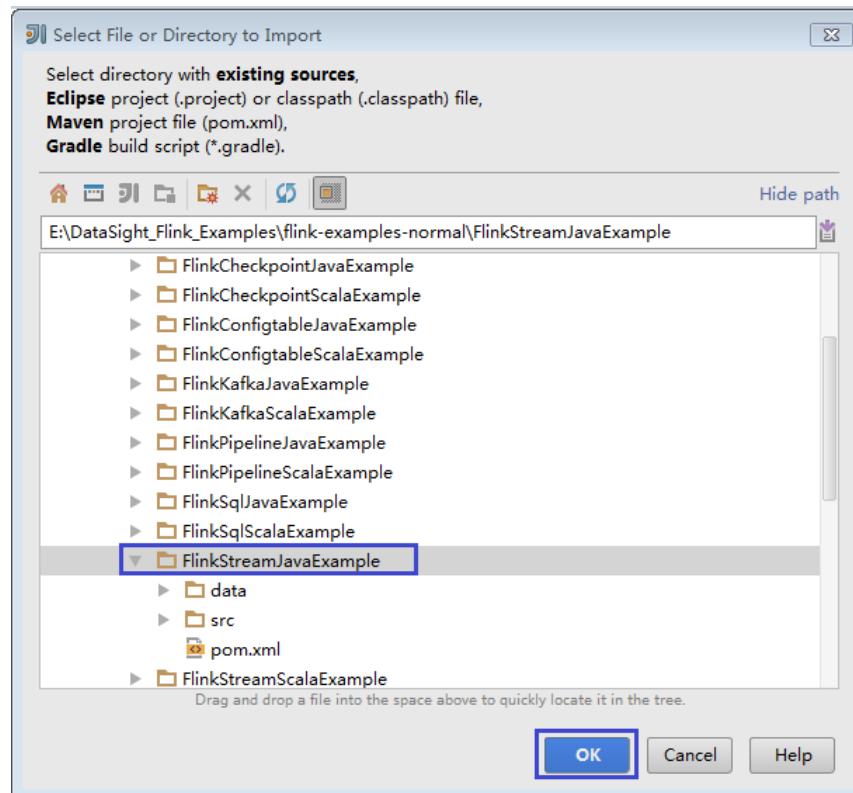
1. Start IntelliJ IDEA. On the **Quick Start** page, select **Import Project**. Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Choose **File > Import project...** to import a project.

Figure 2-48 Importing a project (on the Quick Start page)



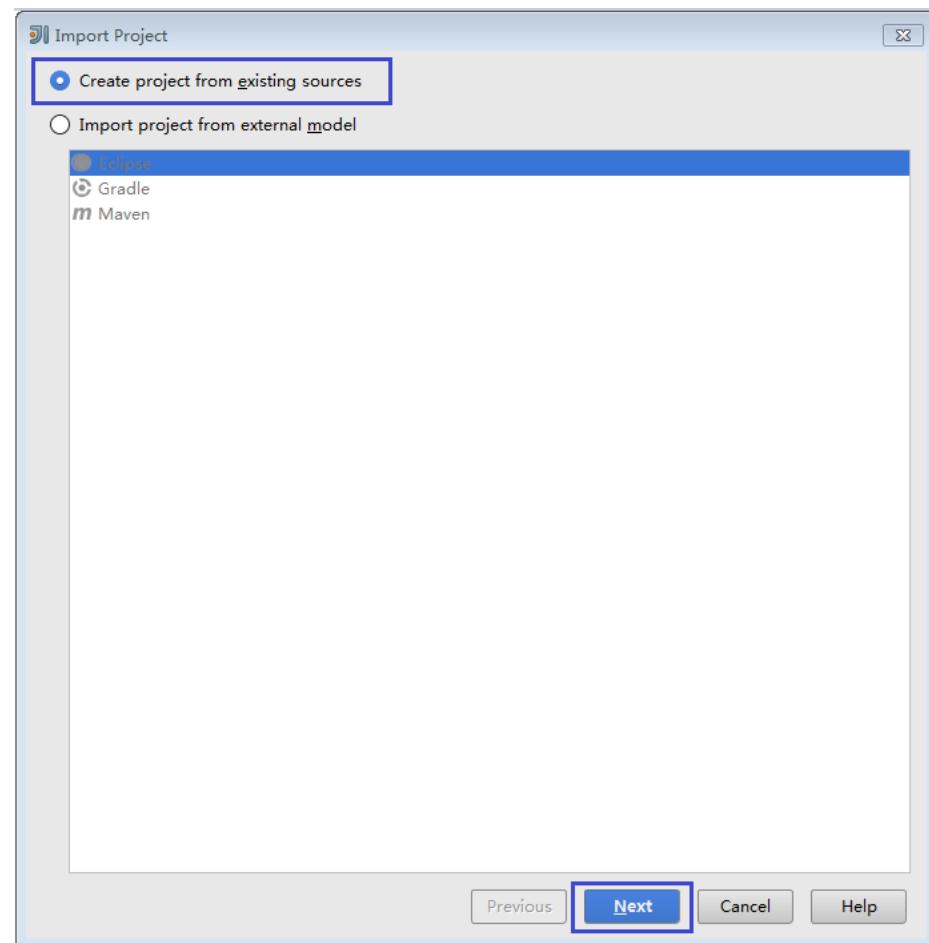
2. Select the directory for storing the imported projects and click **OK**.

Figure 2-49 Select File or Directory to Import



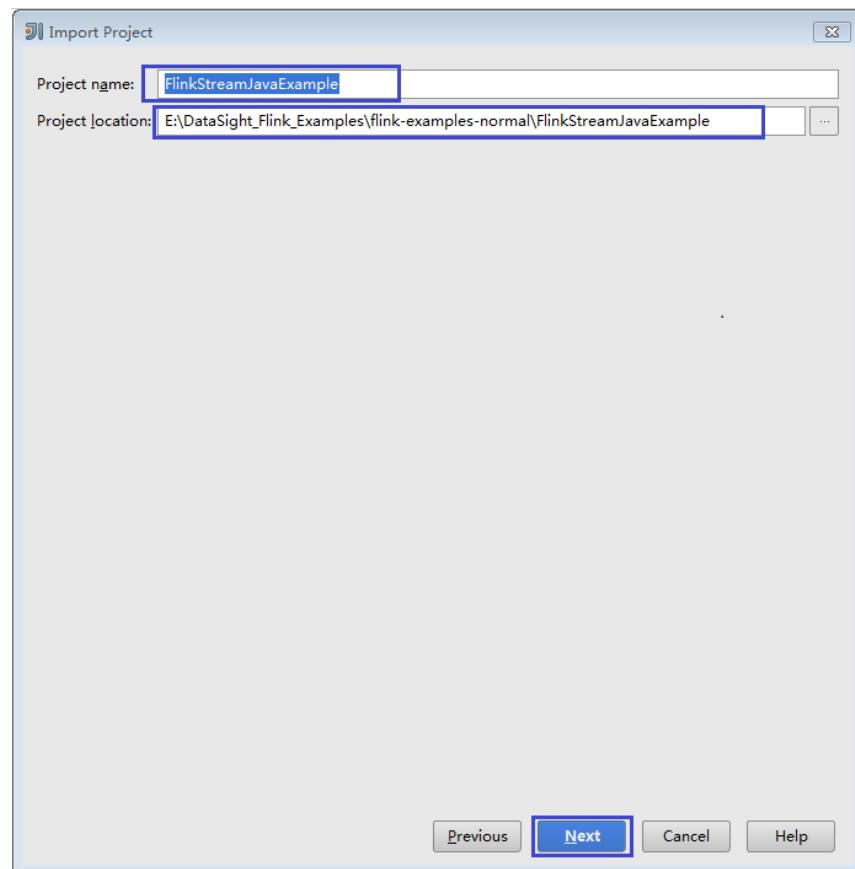
3. Select **Create project from existing sources** and click **Next**.

Figure 2-50 Create project from existing sources



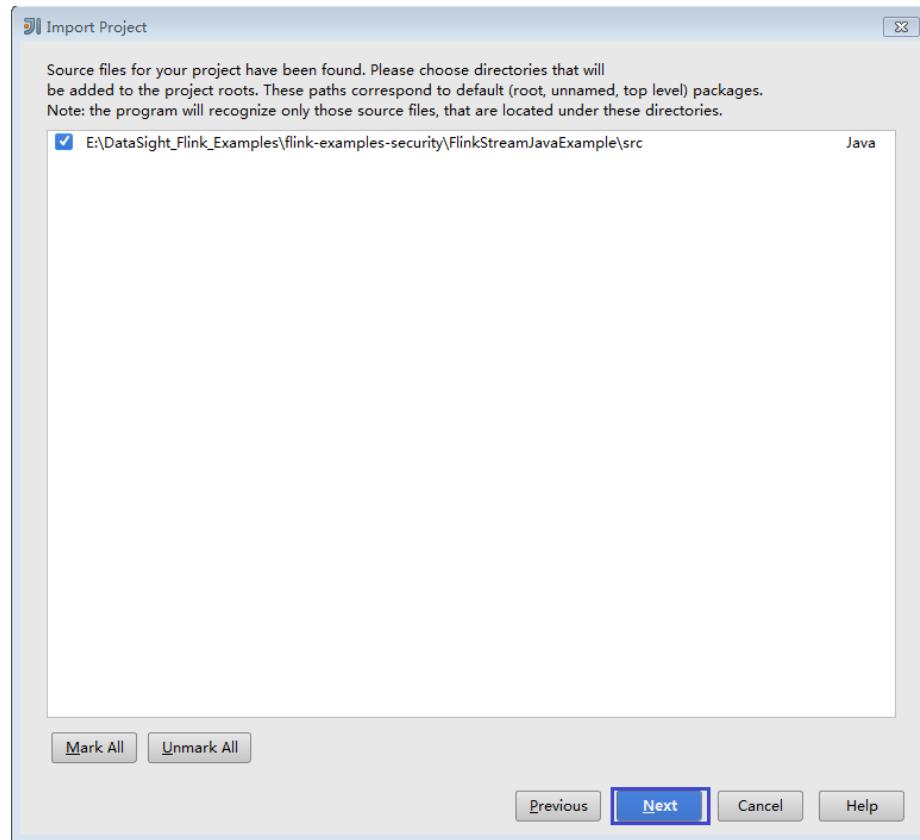
4. Confirm the project location and project name, and click **Next**.

Figure 2-51 Import Project



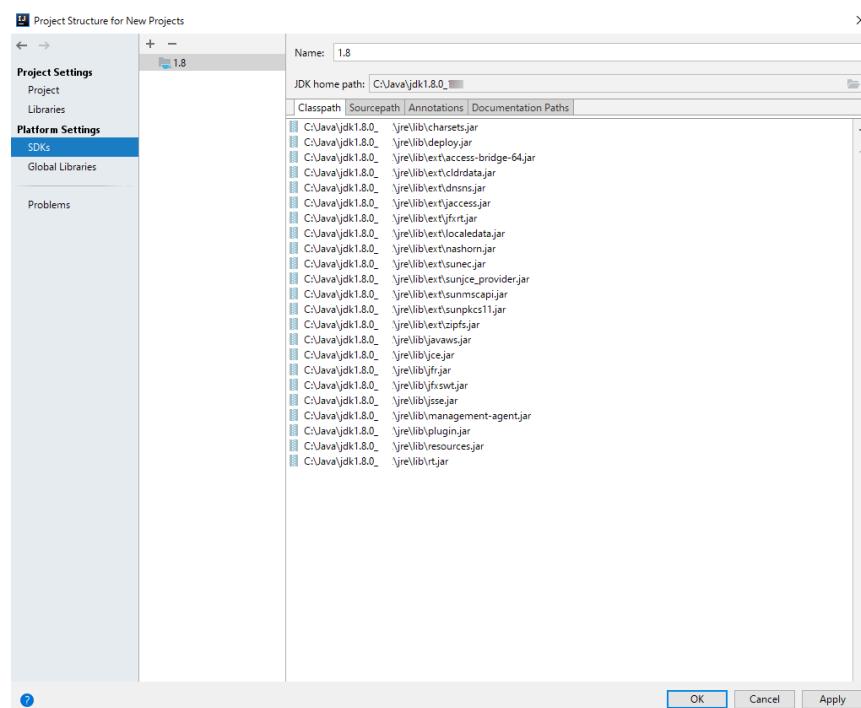
5. Retain the default value of the **root** directory for the project to be imported and click **Next**.

Figure 2-52 Import Project



6. Retain the default dependency library that is automatically recognized by IDEA and the suggested module structure, and click **Next**.
7. Confirm the JDK to be used by the project and click **Next**.

Figure 2-53 Select project SDK



8. After the import is complete, click **Finish**. The imported sample project is displayed on the IDEA home page.

**Figure 2-54** Completing the import

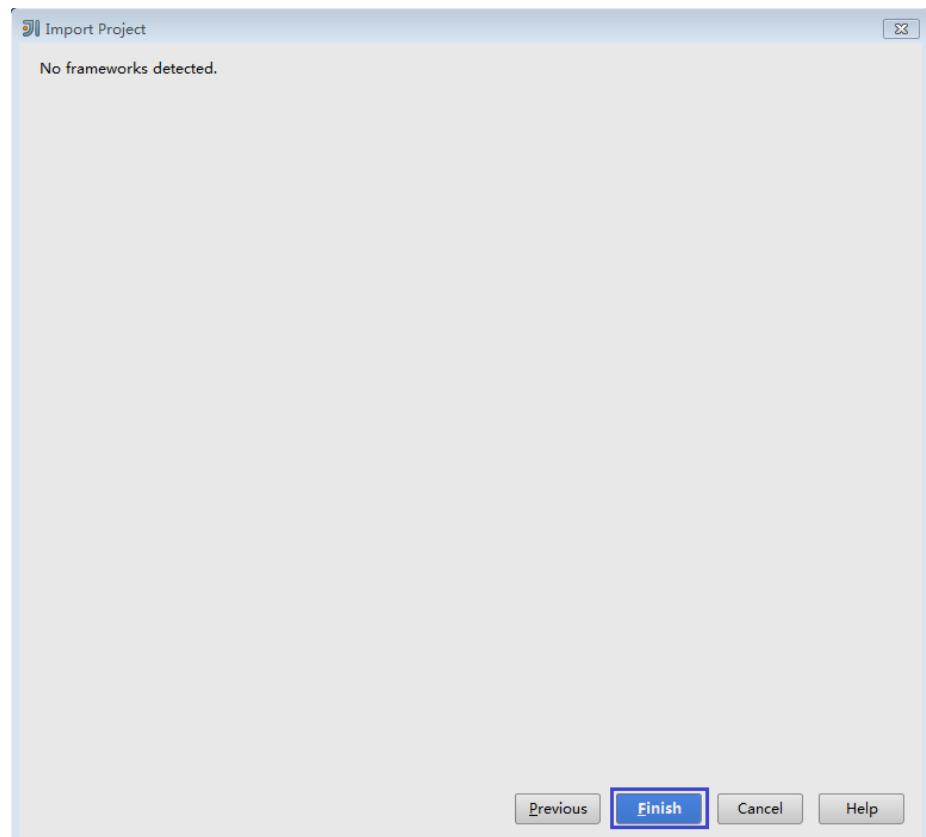
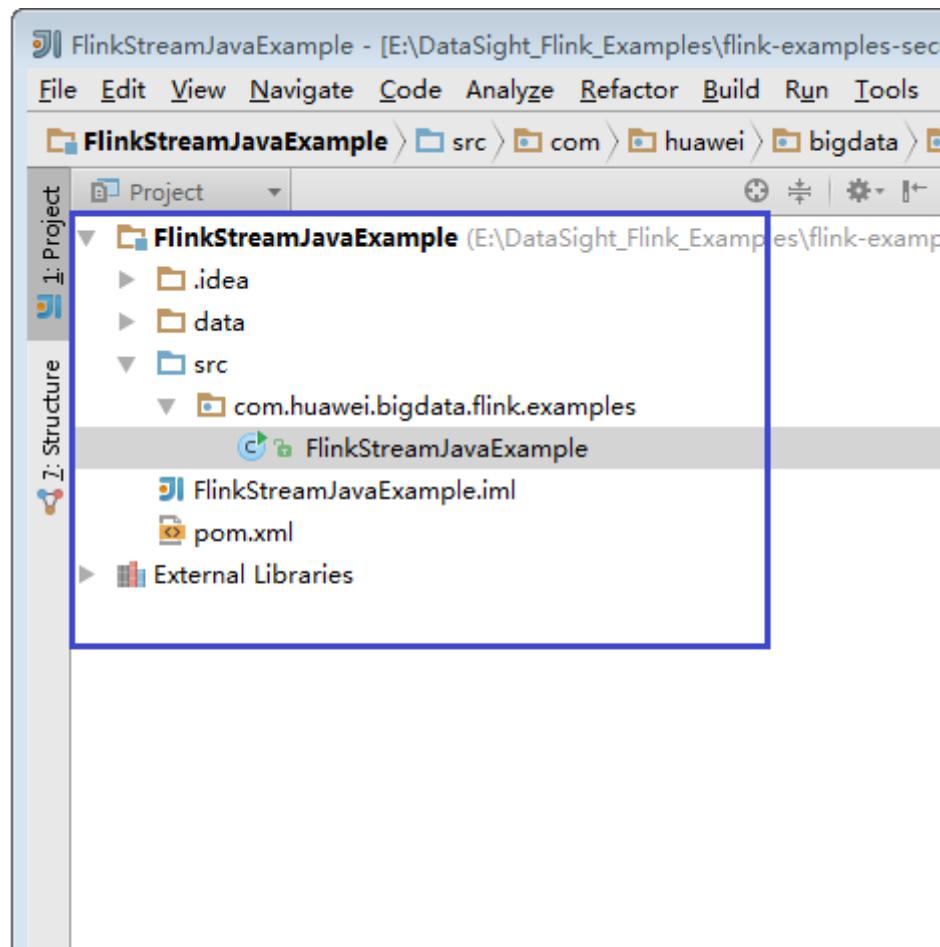


Figure 2-55 Imported project



**Step 5** Import the dependency JAR file for the sample project.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured.

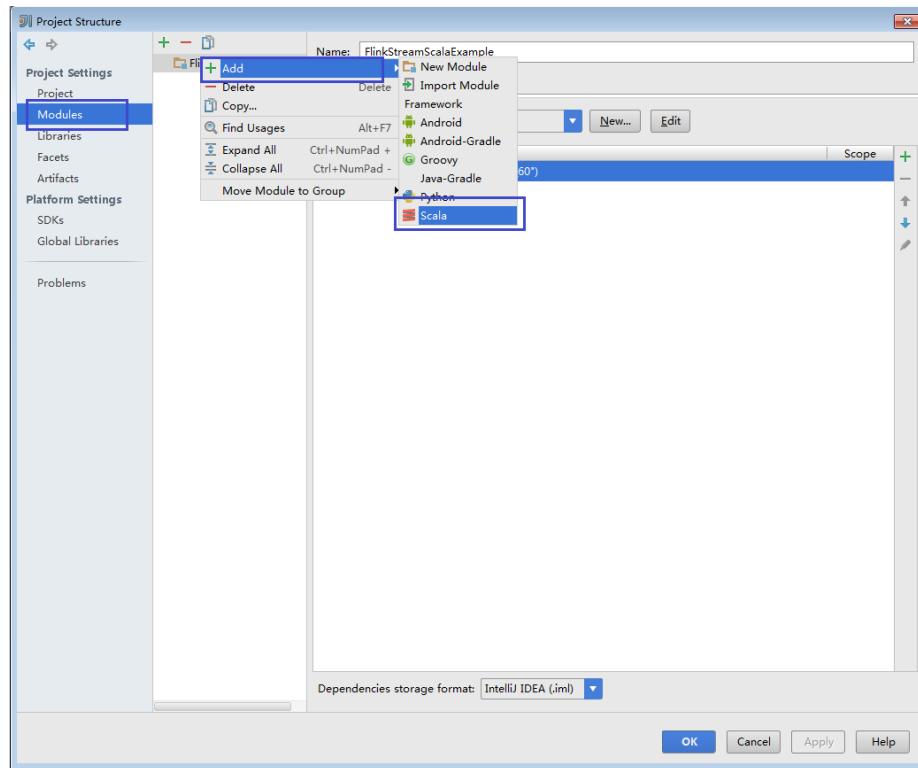
**NOTE**

If other FusionInsight components such as Kafka and Redis are used in the sample code, obtain them from the installation directory of these FusionInsight components. For details about dependency packages of sample projects, see [Reference information about the dependency package for running the sample project](#).

**Step 6** (Optional) If a Scala sample application is imported, configure a language for the project.

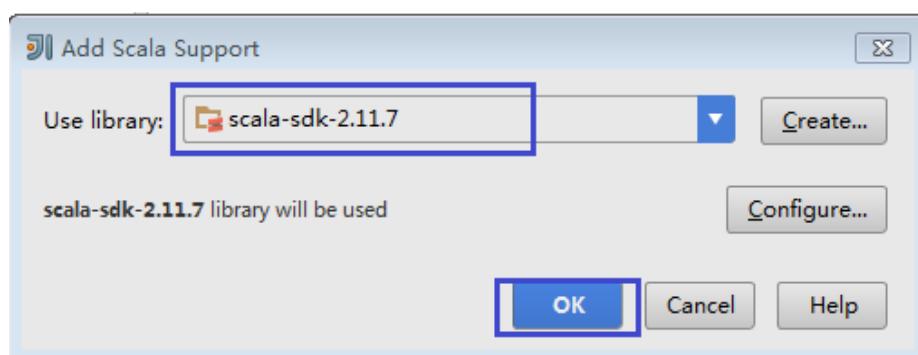
1. On the IDEA home page, choose **File > Project Structures...** to go to the **Project Structure** page.
2. Choose **Modules**, right-click a project name, and choose **Add > Scala**.

Figure 2-56 Selecting Scala



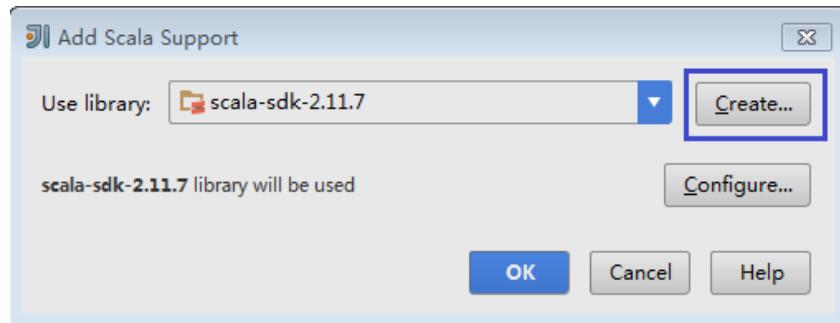
3. Wait until IDEA identifies Scala SDK, select the dependency JAR files in the **Add Scala Support** dialog box, and then click **OK**.

Figure 2-57 Add Scala Support



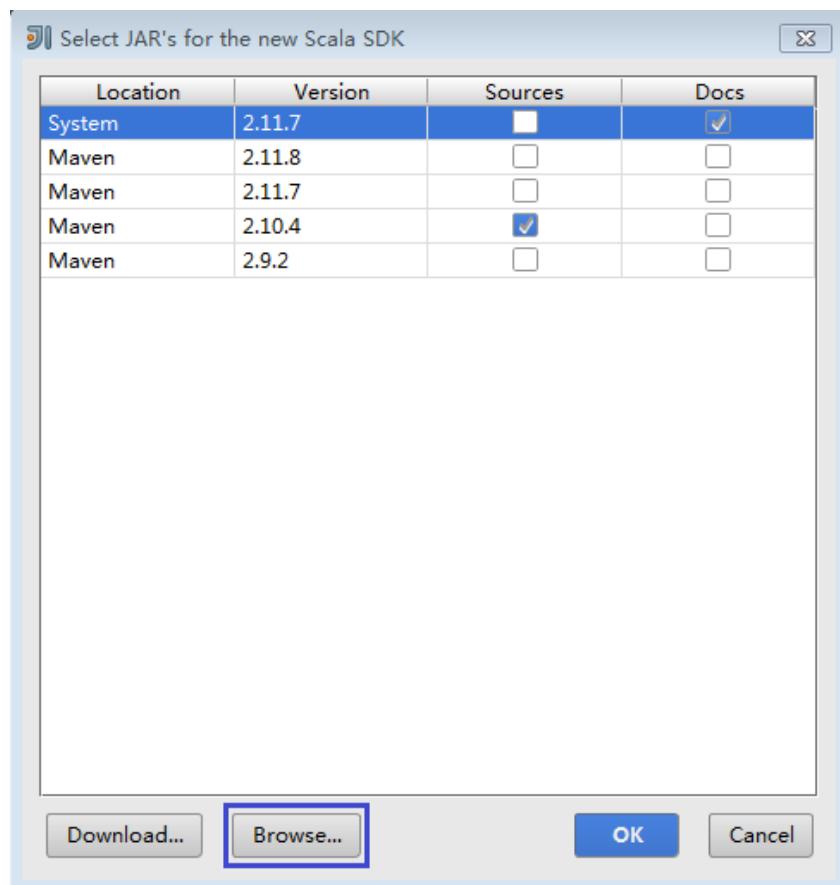
4. If IDEA fails to identify Scala SDK, create a Scala SDK.
  - a. Click **Create....**

**Figure 2-58 Create...**



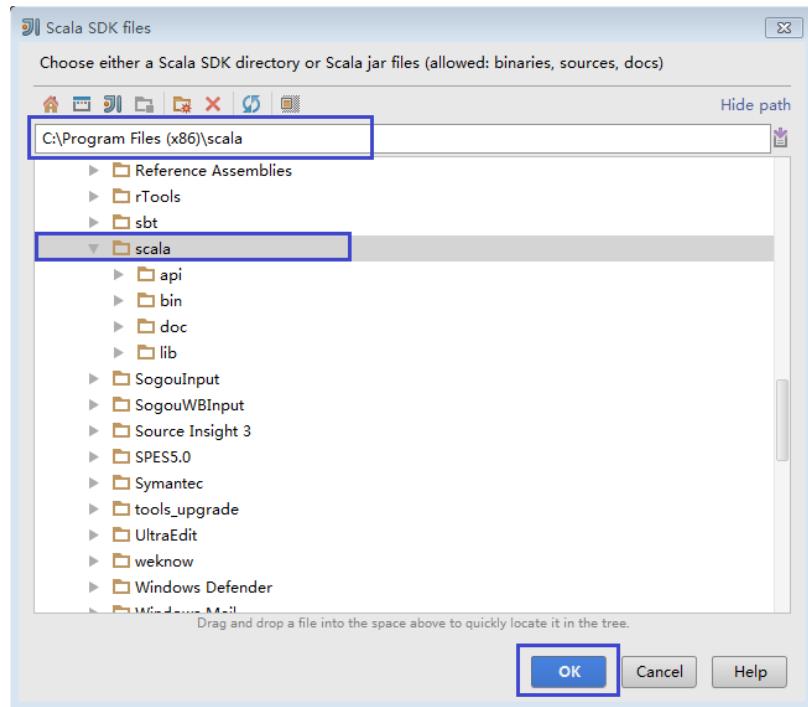
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse....**

**Figure 2-59 Select JAR's for the new Scala SDK**



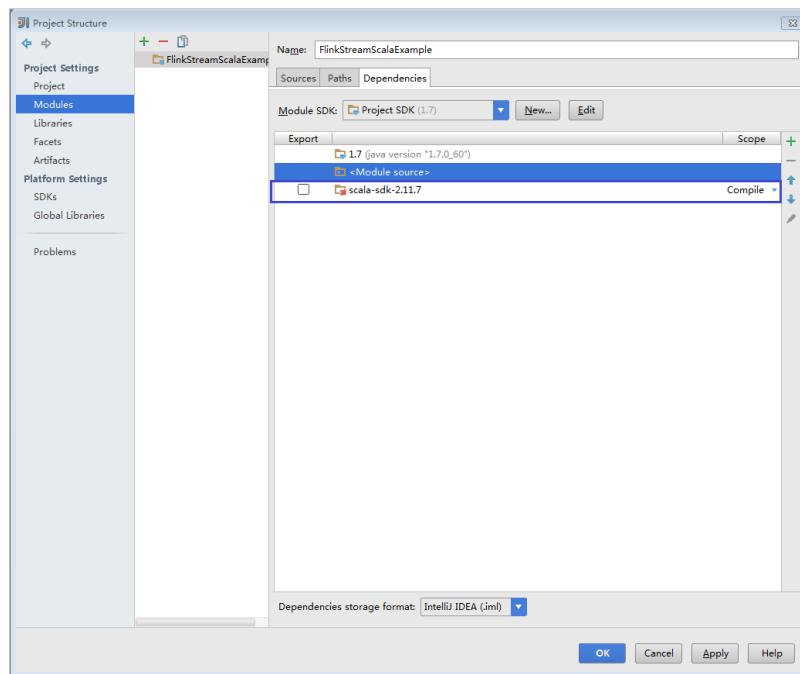
- c. On the **Scala SDK files** page, select the **scala sdk** directory and click **OK**.

Figure 2-60 Scala SDK files



5. Click OK.

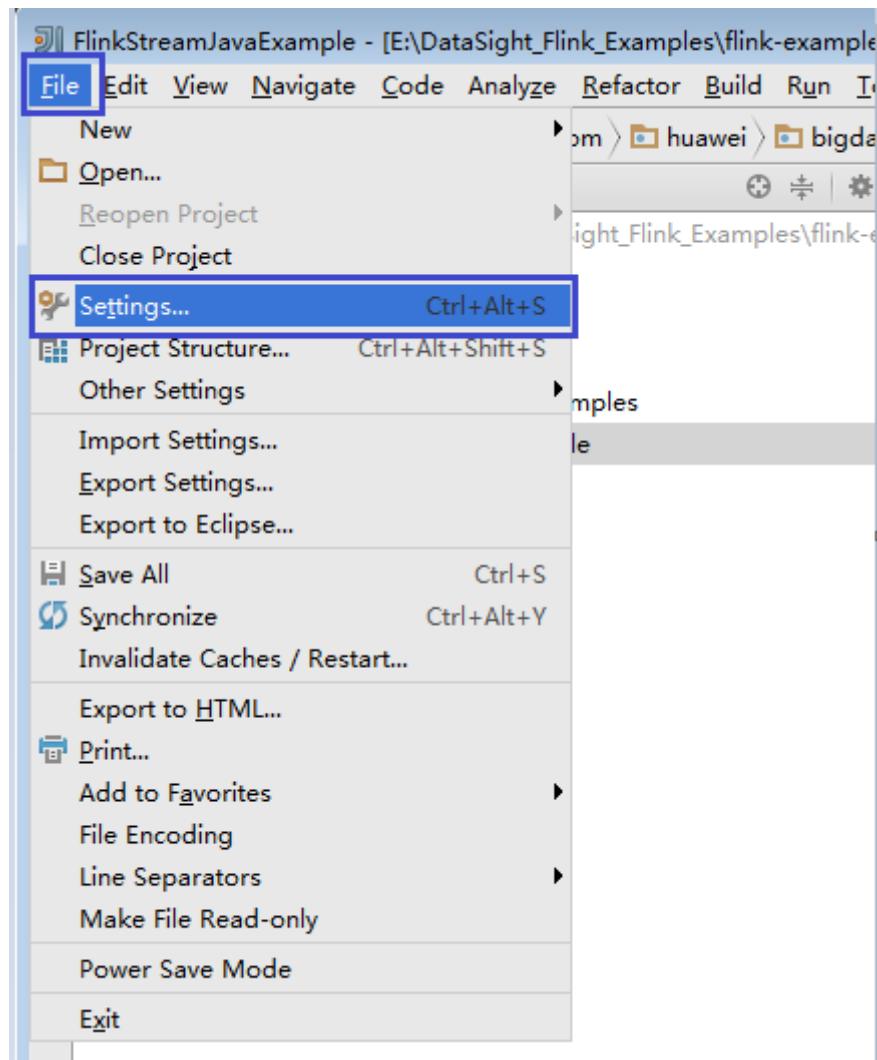
Figure 2-61 Configuration successful



**Step 7** Configure the text file encoding format of IDEA to prevent garbled characters.

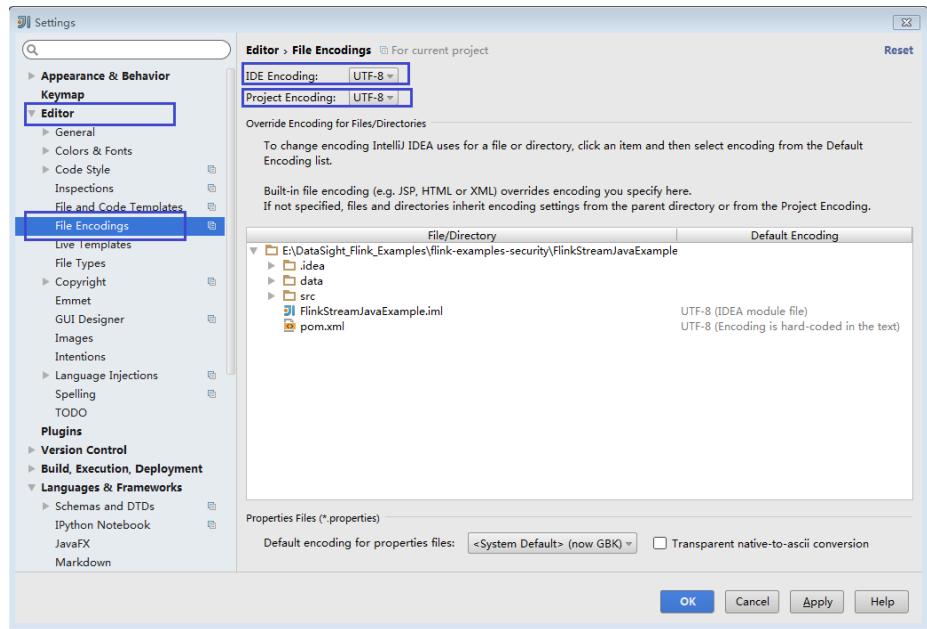
1. On the IDEA home page, choose **File > Settings....**

Figure 2-62 Choosing Settings



2. Configure encoding.
  - a. On the **Settings** page, unfold **Editor**, and choose **File Encodings**.
  - b. Select **UTF-8** from the **IDE Encoding** and **Project Encoding** drop-down lists on the right.
  - c. Click **Apply**.
  - d. Click **OK** to complete the encoding settings.

Figure 2-63 Settings



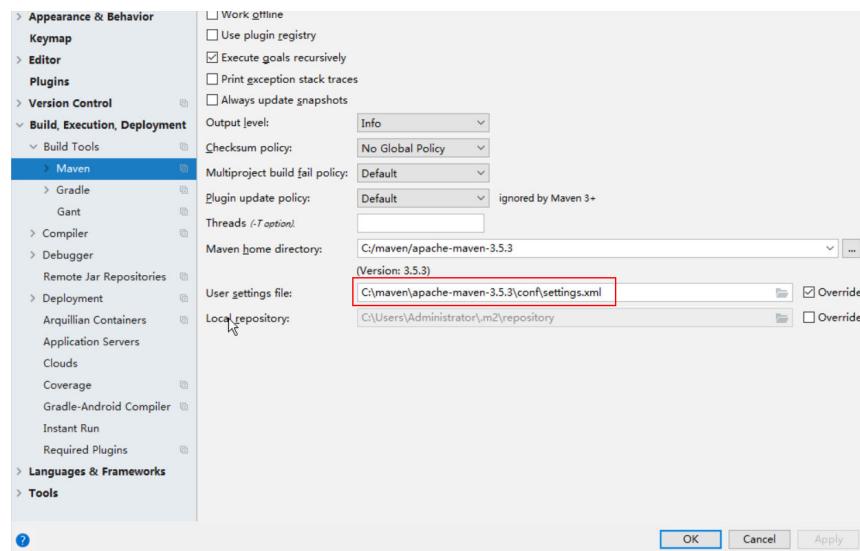
NOTE

- Obtain related dependency packages from the Flink server installation directory.
- Obtain Kafka dependency packages from the Kafka environment.
- Obtain Redis dependency packages from the Redis environment.
- For details about the dependency packages, see [Reference information about the dependency package for running the sample project](#).

**Step 8** Configure Maven.

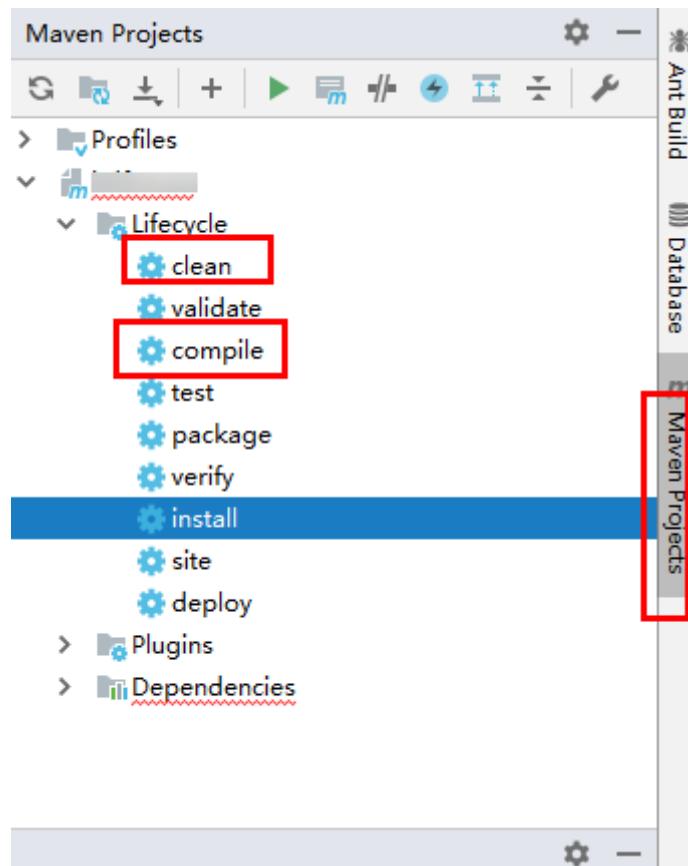
1. Add the configuration information such as the address of the open-source image repository to the `setting.xml` configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the `settings.xml` file is stored. Ensure that the directory is `<Local Maven installation directory>\conf\settings.xml`.

Figure 2-64 Directory for storing the settings.xml file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.
5. On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 2-65 Maven Projects page



----End

## Reference information about the dependency package for running the sample project

The **lib** and **opt** directories of the Flink client contain Flink JAR files. By default, the **lib** directory contains the Flink core JAR file, and the **opt** directory contains the JAR file (for example, **flink-connector-kafka\*.jar**) for connecting to external components. If it is required during application development, manually copy the related JAR files to the **lib** directory.

The dependency packages of the sample projects provided by Flink are as follows:

**Table 2-18** Dependency package for running the sample project

| Sample project   | Dependency package  | Obtaining Dependency Packages  |
|--|---|--|
| <ul style="list-style-type: none"><li>• Sample project of the DataStream application</li><li>• Sample project of the asynchronous checkpoint mechanism application</li></ul> | flink-dist_*.jar  | Can be obtained from <b>lib</b> in the installation directory of the Flink client or server.   |
| <ul style="list-style-type: none"><li>• Sample projects of Submitting SQL Jobs Using Flink Jar</li><li>• Sample projects of FlinkServer REST API</li></ul>                   | <ul style="list-style-type: none"><li>• flink-dist_*.jar</li><li>• flink-table_*.jar</li></ul>              | Can be obtained from <b>lib</b> in the installation directory of the Flink client or server.   |
| Sample projects of the application for producing and consuming data in Kafka   | <ul style="list-style-type: none"><li>• kafka-clients-*.jar</li><li>• flink-connector-kafka_*.jar</li></ul> | <ul style="list-style-type: none"><li>• <b>kafka-clients-*.jar</b> is provided by Kafka and can be obtained from the <b>lib</b> directory in the installation directory of the Kafka client or server.</li><li>• <b>flink-connector-kafka_*.jar</b> can be obtained from <b>opt</b> in the installation directory of the Flink client or server.</li></ul> |
| Sample projects of pipeline  | <ul style="list-style-type: none"><li>• flink-connector-netty_*.jar</li><li>• flink-dist_*.jar</li></ul>    | <ul style="list-style-type: none"><li>• <b>flink-connector-netty_*.jar</b> can be obtained from the <b>lib</b> folder generated after the secondary development sample code is compiled.</li><li>• <b>flink-dist_*.jar</b> can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li></ul>                          |

| Sample project   | Dependency package   | Obtaining Dependency Packages   |
|--|--|---|
| Sample projects of JOIN between configuration tables and streams | <ul style="list-style-type: none"> <li>commons-pool2-*jar</li> <li>flink-dist-*jar</li> <li>jredisclient-*jar</li> <li>super-csv-2.4.0.jar</li> <li>wcc_krb5-*jar</li> </ul> | <ul style="list-style-type: none"> <li><b>commons-pool2-*jar, jredisclient-*jar, and wcc_krb5-*jar</b> are provided by Redis and can be obtained from the <b>lib</b> directory in the installation directory of Redis server.</li> <li>flink-dist-*jar can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li> <li>You can obtain <b>super-csv-2.4.0.jar</b> from the <b>lib</b> directory of the configuration table sample code or download it from <a href="https://github.com/super-csv/super-csv/releases/download/v2.4.0/super-csv-distribution-2.4.0-bin.zip">https://github.com/super-csv/super-csv/releases/download/v2.4.0/super-csv-distribution-2.4.0-bin.zip</a>.</li> </ul> |
| Sample projects of Stream SQL JOIN                               | <ul style="list-style-type: none"> <li>kafka-clients-*jar</li> <li>flink-connector-kafka-*jar</li> <li>flink-dist-*jar</li> <li>flink-table-*jar</li> </ul>                  | <ul style="list-style-type: none"> <li><b>kafka-clients-*jar</b> is provided by Kafka and can be obtained from the <b>lib</b> directory in the installation directory of the Kafka client or server.</li> <li>flink-connector-kafka-*jar can be obtained from <b>opt</b> in the installation directory of the Flink client or server.</li> <li>flink-dist-*jar, flink-table-*jar can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li> </ul>  |
| Sample projects of HBase   | <ul style="list-style-type: none"> <li>flink-connector-hbase-*jar</li> <li>flink-dist-*jar</li> <li>flink-table-*jar</li> <li>hbase-clients-*jar</li> </ul>                  | <ul style="list-style-type: none"> <li>flink-connector-hbase-*jar can be obtained from <b>opt</b> in the installation directory of the Flink client or server.</li> <li>flink-dist-*jar, flink-table-*jar can be obtained from <b>lib</b> in the installation directory of the Flink client or server.</li> <li><b>hbase-clients-*jar</b> is provided by HBase and can be obtained from the <b>lib</b> directory in the installation directory of the HBase client or server.</li> </ul>  |
| Sample projects of Hudi  | hbase-unsafe-*jar  | Can be obtained from the <b>lib</b> folder generated after the secondary development sample code is compiled.   |

### 2.4.2.3 Creating a Project (Optional)

#### Scenarios

IntelliJ IDEA can be used to create a Flink project. A Scala project is used as an example to illustrate how to create a Flink project.

## Procedure

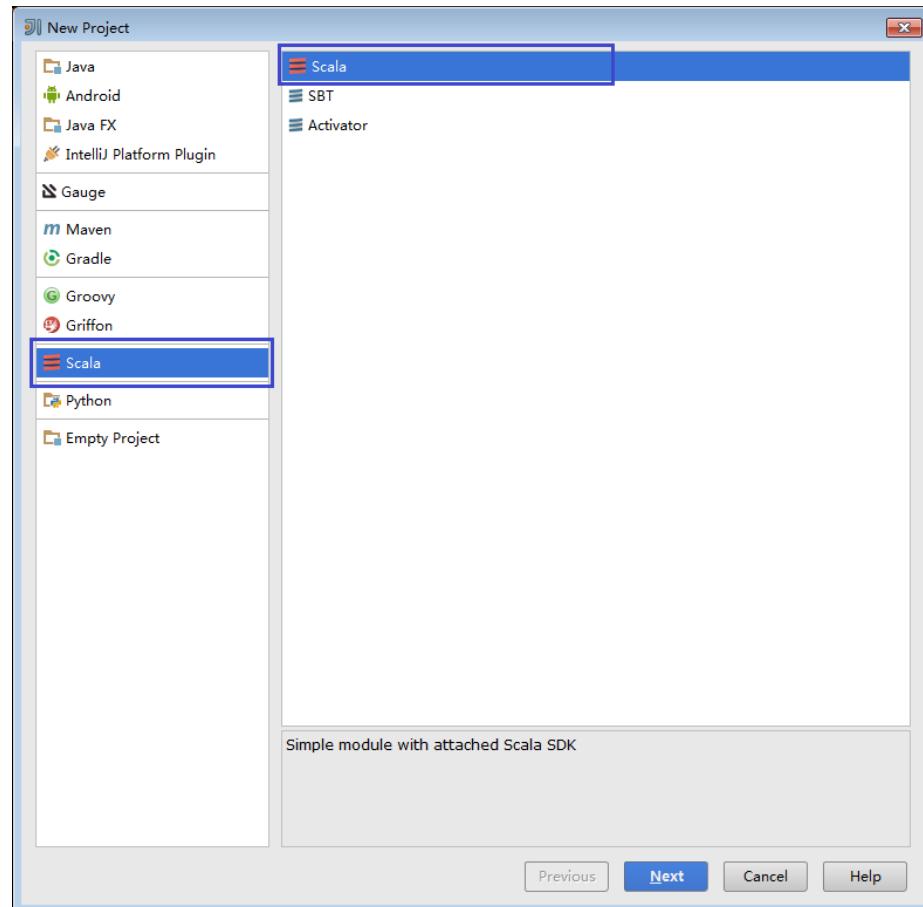
**Step 1** Start the IDEA and choose **Create New Project**.

**Figure 2-66** Creating a project



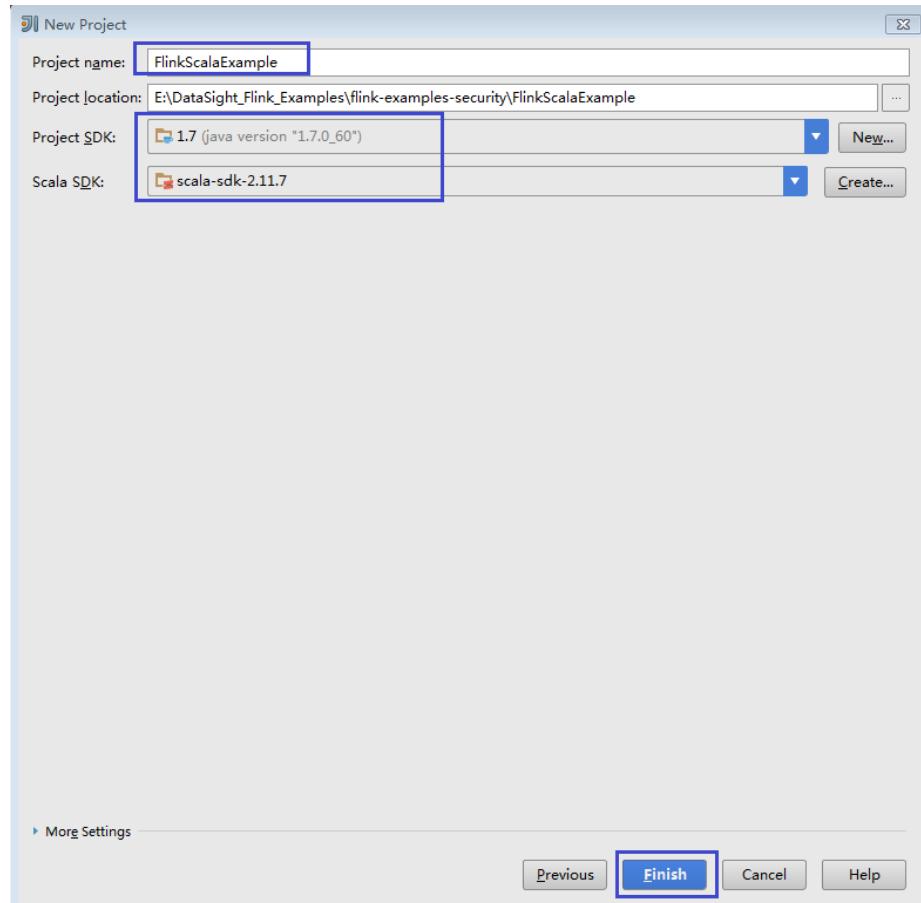
**Step 2** On the **New Project** page, choose a Scala development environment, choose **Scala Module**, and then click **Next**. If you want to create a Java project, choose corresponding parameters of Java.

Figure 2-67 Selecting a development environment



**Step 3** On the New Project page, enter the project name and project location, select the JDK version and Scala SDK, and then click **Finish**.

Figure 2-68 Filling in project information



----End

#### 2.4.2.4 Configuring a Spring Boot Sample Project

##### Scenarios

Run the sample code of the Spring Boot interface in FusionInsight MRS Hive. Currently, GaussDB(DWS) and Elasticsearch SpringBoot sample projects are supported.

In the following example, an application is developed in Linux to connect GaussDB(DWS) to Flink using Spring Boot.

##### NOTE

Before running the GaussDB(DWS) sample, log in to the node where GaussDB(DWS) is deployed to create an empty table **test\_lzh1** for receiving data. The creation command is as follows:

```
create table test_lzh1 (id integer not null);
```

##### Procedure

**Step 1** Install a client that contains only the Flink service.

The following example shows how to install the Flink client on a node in the cluster:

1. Log in to FusionInsight Manager. Choose **Cluster > Services > Flink > More > Download Client**.
2. In the dialog box that is displayed, select **Complete Client** for **Select Client Type**, select the platform type that matches the architecture of the node where the client is to install, Select **Server** from the **Select Download Location**, and click **OK**.

The generated file is stored in the **/tmp/FusionInsight-Client** directory on the active management node by default.

The name of the client software package is in the following format:  
**FusionInsight\_Cluster\_<Cluster ID>\_Flink\_Client.tar**. The following content uses the cluster ID **1** as an example.

3. Log in to the server where the client is to be installed as the client installation user.
4. Go to the directory where the installation package is stored and run the following commands to decompress the package:

```
cd /tmp/FusionInsight-Client
```

```
tar -xvf FusionInsight_Cluster_1_Flink_Client.tar
```

5. Run the following command to verify the decompressed file and check whether the command output is consistent with the information in the **sha256** file:

```
sha256sum -c FusionInsight_Cluster_1_Flink_ClientConfig.tar.sha256
```

6. Decompress the obtained installation file.

```
tar -xvf FusionInsight_Cluster_1_Flink_ClientConfig.tar
```

7. Go to the directory where the installation package is stored, and run the following command to install the client to a specified directory (absolute path), for example, **/opt/hadoopclient**:

```
cd /tmp/FusionInsight-Client/FusionInsight_Cluster_1_Flink_ClientConfig  
.install.sh /opt/hadoopclient
```

8. Configure security authentication and encryption. For details, see "Using Flink from Scratch" in the *Component Operation Guide*.

**Step 2** Obtain the sample project folder **flink-dws-sink-example** from **src\springboot\flink-examples** in the directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

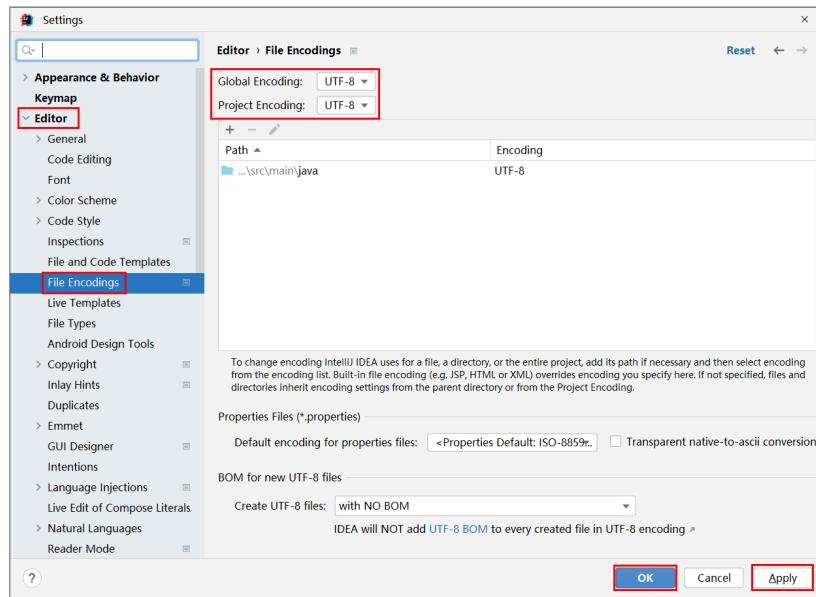
**Step 3** Import the sample project to the IntelliJ IDEA development environment.

1. On the menu bar of IntelliJ IDEA, choose **File > Open....**
2. In the **Open File or Project** dialog box that is displayed, select the **flink-dws-sink-example** folder and click **OK**.

**Step 4** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

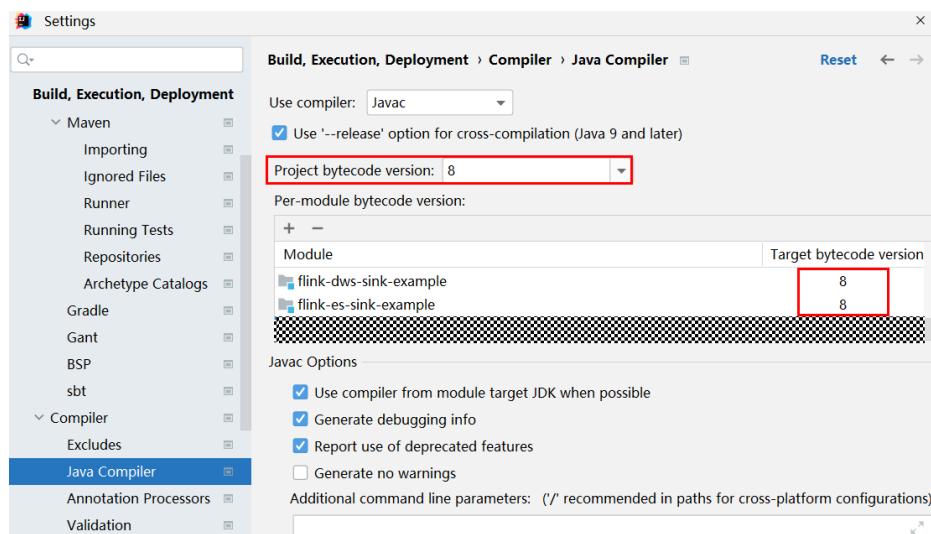
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. In the left navigation pane, choose **Editor > File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**.

**Figure 2-69** Setting the IntelliJ IDEA coding format

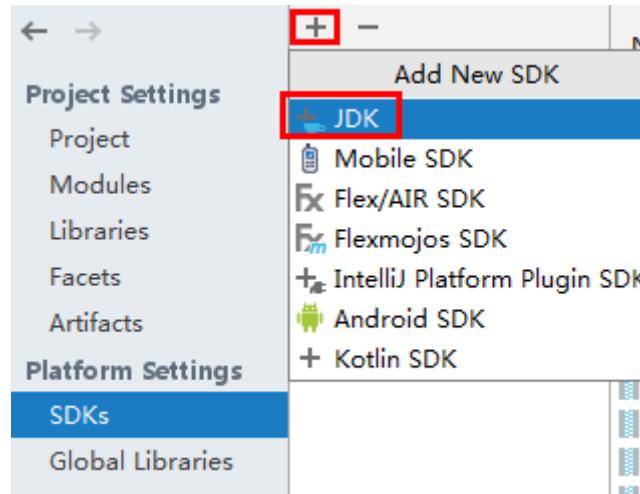


### Step 5 Set the JDK of the project.

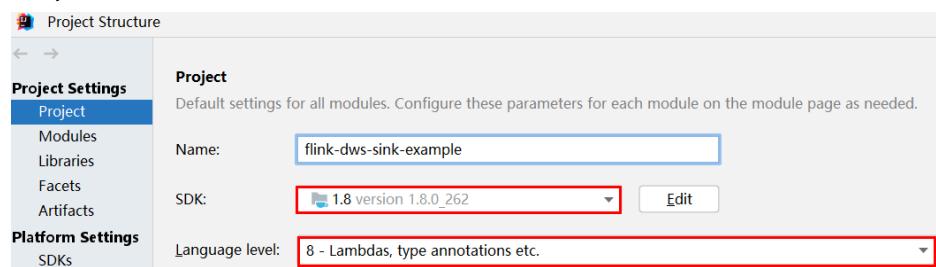
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **flink-dws-sink-example** to **8**.



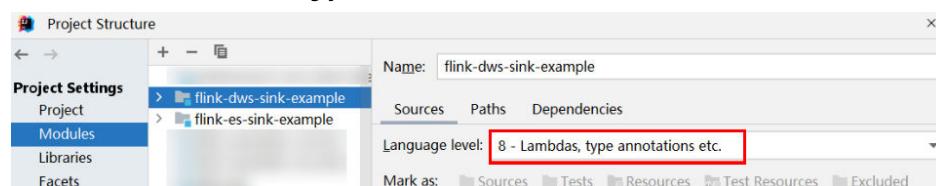
3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.



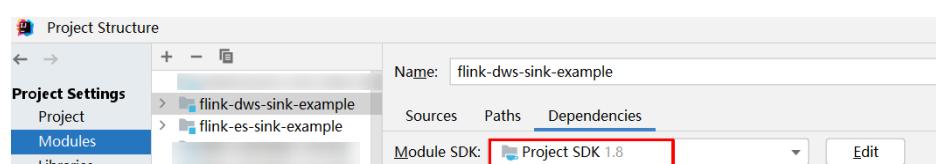
6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc..**



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.



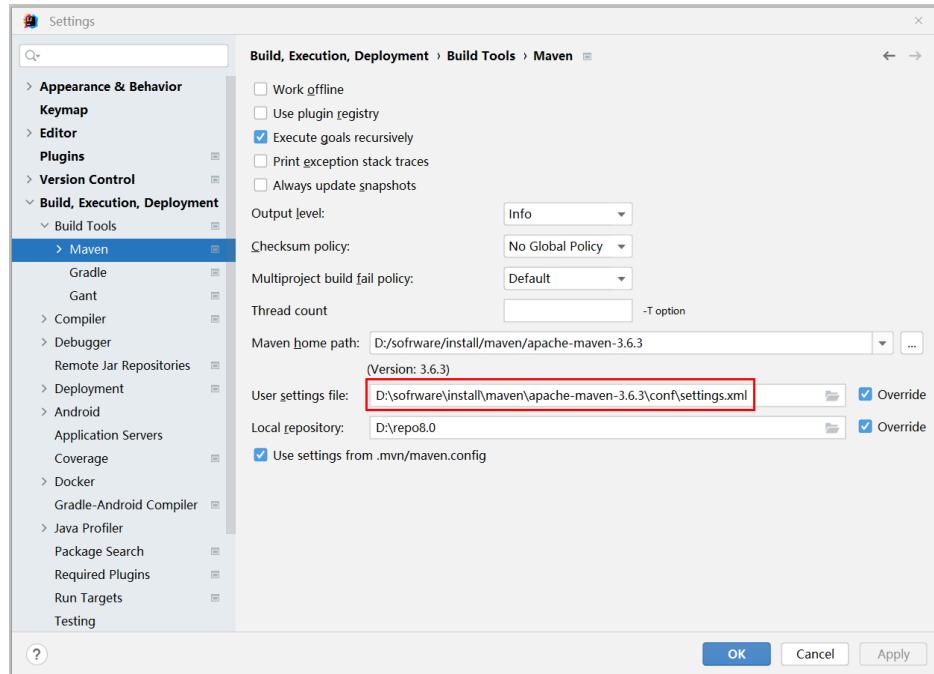
11. Click **Apply** and then **OK**.

#### Step 6 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).

2. On the IntelliJ IDEA page, choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

**Figure 2-70** Directory for storing the settings.xml file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply** and then **OK**.

**Step 7** Find the **application.properties** file in **src\main\resources** and add the following content to the file:

- Run the GaussDB(DWS) sample

```
spring.datasource.dws.url=jdbc:postgresql://IP address of the GaussDB(DWS) node:8000/postgres
spring.datasource.dws.username=dbadmin
spring.datasource.dws.password=Password of dbadmin
spring.datasource.dws.driver=org.postgresql.Driver
```
- Run the Elasticsearch sample

```
spring.datasource.es.host=Service IP address of Elasticsearch EsNode
spring.datasource.es.port=Elasticsearch EsNode Port
spring.datasource.es.scheme=http
```

#### NOTE

- log in to FusionInsight Manager and choose **Cluster > Services > Elasticsearch**. On the page that is displayed, click **Instance** and view the service IP addresses of all EsNode instances in the Elasticsearch cluster.
- Log in to FusionInsight Manager, choose **Cluster > Services > Elasticsearch > Configuration > All Configurations**, search for **SERVER\_PORT**, and view the EsNode port.

----End

## 2.4.3 Developing an Application

### 2.4.3.1 DataStream Application

#### 2.4.3.1.1 Scenarios

##### Scenarios

Develop a DataStream application of Flink to perform the following operations on logs about dwell durations of netizens for shopping online.



The DataStream application can run in both the Windows environment and the Linux environment.

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours in total in a real time manner.
- The first column in the log file records names, the second column records genders, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday. The log file can be obtained from the **data** directory of the sample project.

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

##### Data Planning

Data of DataStream sample project is stored in a **.txt** file.

Place the **log1.txt** and **log2.txt** in two directories, for example, **/opt/log1.txt** and **/opt/log2.txt**.

### NOTE

- If the data file is stored in the local file system, the data file must be stored in the specified directory on all nodes where Yarn NodeManager is deployed, and the running user access permission must be set.
- Alternatively, store the data file on HDFS and set the file read path in the program to the HDFS path, for example, `hdfs://hacluster/path/to/file`.

## Development Approach

Collect the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

1. Read text data, generate DataStreams, and parse data to generate UserRecord information.
2. Filter the data about the time that female netizens spend online.
3. Perform keyby operation based on the name and gender, and collect the time that female netizens spend online within a time window.
4. Filter data about users whose consecutive online duration exceeds the threshold, and obtain the result.

### 2.4.3.1.2 Java Sample Code

## Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

## DataStream FlinkStreamJavaExample Sample Code

The following code segment is an example. For details, see `com.huawei.bigdata.flink.examples.FlinkStreamJavaExample`.

```
//Parameter description:  
//<filePath> indicates paths where text is read. Paths are separated by commas (,).  
//<windowTime> indicates the period covered by statistics window. The unit is minute.  
public class FlinkStreamJavaExample {  
    public static void main(String[] args) throws Exception {  
        //Print reference command for executing flink run.  
        System.out.println("use command as: ");  
        System.out.println("./bin/flink run --class  
com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/test.jar --filePath /opt/log1.txt,/opt/  
log2.txt --windowTime 2");  
        System.out.println("*****");  
        System.out.println("<filePath> is for text file to read data, use comma to separate");  
        System.out.println("<windowTime> is the width of the window, time as minutes");  
        System.out.println("*****");  
  
        //Paths where text is read. Paths are separated by commas (,)  
        final String[] filePaths = ParameterTool.fromArgs(args).get("filePath", "/opt/log1.txt,/opt/  
log2.txt").split(",");  
        assert filePaths.length > 0;  
  
        //windowTime specifies the size of the time window. By default, a window with 2 minutes as the size  
        can read all data in a text file.  
        final int windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2);  
    }  
}
```

```
//Build the execution environment and run eventTime to process window data.  
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);  
env.setParallelism(1);  
  
//Read DataStream of the text.  
DataStream<String> unionStream = env.readTextFile(filePaths[0]);  
if (filePaths.length > 1) {  
    for (int i = 1; i < filePaths.length; i++) {  
        unionStream = unionStream.union(env.readTextFile(filePaths[i]));  
    }  
}  
  
//Convert data, build the logic for the entire data process, calculate, and print results.  
unionStream.map(new MapFunction<String, UserRecord>() {  
    @Override  
    public UserRecord map(String value) throws Exception {  
        return getRecord(value);  
    }  
}).assignTimestampsAndWatermarks(  
    new Record2TimestampExtractor()  
).filter(new FilterFunction<UserRecord>() {  
    @Override  
    public boolean filter(UserRecord value) throws Exception {  
        return value.sex.equals("female");  
    }  
}).keyBy(  
    new UserRecordSelector()  
).window(  
    TumblingEventTimeWindows.of(Time.minutes(windowTime))  
).reduce(new ReduceFunction<UserRecord>() {  
    @Override  
    public UserRecord reduce(UserRecord value1, UserRecord value2)  
        throws Exception {  
        value1.shoppingTime += value2.shoppingTime;  
        return value1;  
    }  
}).filter(new FilterFunction<UserRecord>() {  
    @Override  
    public boolean filter(UserRecord value) throws Exception {  
        return value.shoppingTime > 120;  
    }  
}).print();  
  
//Call execute to trigger the execution.  
env.execute("FemaleInfoCollectionPrint java");  
}  
  
//Build the keyword of keyBy as the grouping basis.  
private static class UserRecordSelector implements KeySelector<UserRecord, Tuple2<String, String>> {  
    @Override  
    public Tuple2<String, String> getKey(UserRecord value) throws Exception {  
        return Tuple2.of(value.name, value.sex);  
    }  
}  
  
//Parse the row data of the text and build UserRecord data structure.  
private static UserRecord getRecord(String line) {  
    String[] elems = line.split(",");  
    assert elems.length == 3;  
    return new UserRecord(elems[0], elems[1], Integer.parseInt(elems[2]));  
}  
  
//Defines UserRecord data structure and rewrite toString printing method.  
public static class UserRecord {  
    private String name;  
    private String sex;  
    private int shoppingTime;
```

```

public UserRecord(String n, String s, int t) {
    name = n;
    sexy = s;
    shoppingTime = t;
}

public String toString() {
    return "name: " + name + " sexy: " + sexy + " shoppingTime: " + shoppingTime;
}

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and watermark.
private static class Record2TimestampExtractor implements
AssignerWithPunctuatedWatermarks<UserRecord> {

    //add tag in the data of datastream elements
    @Override
    public long extractTimestamp(UserRecord element, long previousTimestamp) {
        return System.currentTimeMillis();
    }

    //give the watermark to trigger the window to execute, and use the value to check if the window
    elements are ready
    @Override
    public Watermark checkAndGetNextWatermark(UserRecord element, long extractedTimestamp) {
        return new Watermark(extractedTimestamp - 1);
    }
}
}

```

The following is the command output:

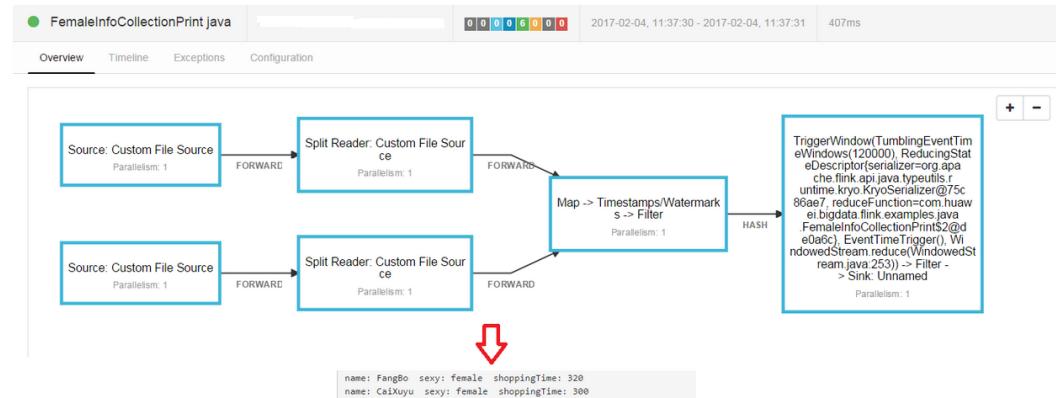
```

name: FangBo sexy: female shoppingTime: 320
name: CaiXuyu sexy: female shoppingTime: 300

```

**Figure 2-71** shows the process of execution.

**Figure 2-71** Process of execution



### 2.4.3.1.3 Scala Sample Code

#### Function Description

Collect the information about female netizens who continuously spend more than 2 hour in online shopping and print the result.

## DataStream FlinkStreamScalaExample Sample Code

The following code is an example. For details, see  
[com.huawei.bigdata.flink.examples.FlinkStreamScalaExample](#).

```
//Parameter description
//filePath indicates paths where text is read. Paths are separated by commas (,).
//windowTime indicates the period covered by statistics window. The unit is minute.
object FlinkStreamScalaExample {
def main(args: Array[String]) {
    //Print the reference command used to execute flink run.
    System.out.println("use command as:")
    System.out.println("./bin/flink run --class")
    com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/test.jar --filePath /opt/log1.txt,/opt/
log2.txt --windowTime 2")
    System.out.println("*****")
    System.out.println("<filePath> is for text file to read data, use comma to separate")
    System.out.println("<windowTime> is the width of the window, time as minutes")
    System.out.println("*****")

    //Paths where text is read. Paths are separated by commas (,)
    val filePaths = ParameterTool.fromArgs(args).get("filePath",
        "/opt/log1.txt,/opt/log2.txt").split(",").map(_.trim)
    assert(filePaths.length > 0)

    //windowTime specifies the size of the time window. By default, a window with 2 minutes as the size can
    read all data in a text file.
    val windowTime = ParameterTool.fromArgs(args).getInt("windowTime", 2)

    //Build the execution environment and run eventTime to process window data.
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
    env.setParallelism(1)
    //Read DataStream of the text.
    val unionStream = if (filePaths.length > 1) {
        val firstStream = env.readTextFile(filePaths.apply(0))
        firstStream.union(filePaths.drop(1).map(it => env.readTextFile(it)): _*)
    } else {
        env.readTextFile(filePaths.apply(0))
    }

    //Convert data, build the logic for the entire data process, calculate, and print results.
    unionStream.map(getRecord(_))
        .assignTimestampsAndWatermarks(new Record2TimestampExtractor)
        .filter(_.sexy == "female")
        .keyBy("name", "sexy")
        .window(TumblingEventTimeWindows.of(Time.minutes(windowTime)))
        .reduce((e1, e2) => UserRecord(e1.name, e1.sex, e1.shoppingTime + e2.shoppingTime))
        .filter(_shoppingTime > 120).print()

    //Call execute to trigger the execution
    env.execute("FemaleInfoCollectionPrint scala")
}

//Parse the row data of the text and build UserRecord data structure.
def getRecord(line: String): UserRecord = {
    val elems = line.split(",")
    assert(elems.length == 3)
    val name = elems(0)
    val sexy = elems(1)
    val time = elems(2).toInt
    UserRecord(name, sexy, time)
}

//Defines UserRecord data structure.
case class UserRecord(name: String, sexy: String, shoppingTime: Int)

//Build class that inherits AssignerWithPunctuatedWatermarks to configure eventTime and waterMark.
```

```
private class Record2TimestampExtractor extends AssignerWithPunctuatedWatermarks[UserRecord] {

    //add tag in the data of datastream elements
    override def extractTimestamp(element: UserRecord, previousTimestamp: Long): Long = {
        System.currentTimeMillis()
    }

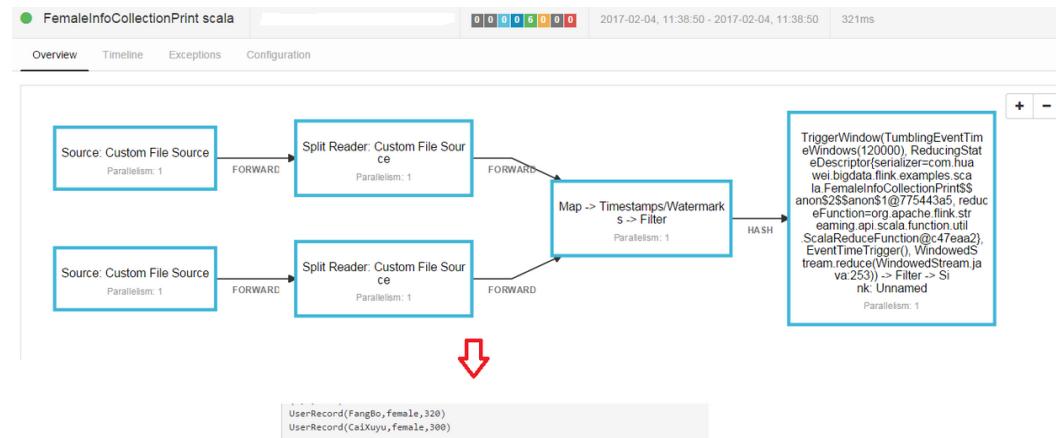
    //give the watermark to trigger the window to execute, and use the value to check if the window
    //elements are ready
    def checkAndGetNextWatermark(lastElement: UserRecord,
                                  extractedTimestamp: Long): Watermark = {
        new Watermark(extractedTimestamp - 1)
    }
}
```

The following is the command output:

```
UserRecord(FangBo,female,320)
UserRecord(CaiXuyu,female,300)
```

[Figure 2-72](#) shows the process of execution.

**Figure 2-72** Process of execution



### 2.4.3.2 Interconnecting with Kafka

#### 2.4.3.2.1 Scenarios

##### Scenarios

Assume that a Flink service receives one word record every 1 second.

Develop a Flink application that can generate output of prefixed message contents.

##### Data Planning

Sample project data of Flink is stored in Kafka. A user with Kafka permission can send data to Kafka and receive data from it.

1. Ensure that clusters, including HDFS, YARN, Flink, and Kafka are successfully installed.

## 2. Create a topic.

The format of the command is following:

```
bin/kafka-topics.sh --create --bootstrap-server <Kafka cluster IP address:21005> --command-config config/client.properties --partitions {partitionNum} --replication-factor {replicationNum} --topic {Topic}
```

**Table 2-19**

| parameter                  | Description   |
|----------------------------|---|
| {Kafka cluster IP address} | Service IP address of the Broker node in the Kafka cluster. |
| {PartitionNum}             | The number of partitions for the topic.                     |
| {ReplicationNum}           | The number of copies of each partition for the topic.       |
| {Topic}                    | The topic name.   |

For example, run the following command on the Kafka client, and the topic name is **topic1**.

```
bin/kafka-topics.sh --create --bootstrap-server  
10.96.101.32:21005,10.96.101.251:21005,10.96.101.177:21005 --partitions 5 --replication-factor 1 --topic topic1
```

## Development Approach

1. Start the Flink Kafka Producer to send data to Kafka.
2. Start Flink Kafka Consumer to receive data from Kafka. Ensure that topics of Kafka Consumer are consistent with that of Kafka Producer.
3. Add prefix to the data content and print the result.

### 2.4.3.2.2 Java Sample Code

#### Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

#### Sample Code

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see `com.huawei.bigdata.flink.examples.WriteIntoKafka` and `com.huawei.bigdata.flink.examples.ReadFromKafka`.

```
//producer code  
public class WriteIntoKafka {  
    public static void main(String[] args) throws Exception {  
        //Print the reference command of flink run.  
        System.out.println("use command as: ");  
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
```

```
" /opt/test.jar --topic topic-test --bootstrap.servers 10.91.8.218:21005");
System.out.println("*****");
System.out.println("<topic> is the kafka topic name");
System.out.println("<bootstrap.servers> is the ip:port list of brokers");
System.out.println("*****");

//Build the execution environment.
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
//Configure the parallelism.
env.setParallelism(1);
//Parse the execution parameter.
ParameterTool paraTool = ParameterTool.fromArgs(args);
//Build the StreamGraph and write data generated by customized source into Kafka.
DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
messageStream.addSink(new FlinkKafkaProducer<>(paraTool.get("topic"),
    new SimpleStringSchema(),
    paraTool.getProperties()));
//Call execute to trigger the execution.
env.execute();
}

//Customize source to continuously generate messages every one second.
public static class SimpleStringGenerator implements SourceFunction<String> {
    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
    long i = 0;

    @Override
    public void run(SourceContext<String> ctx) throws Exception {
        while (running) {
            ctx.collect("element-" + (i++));
            Thread.sleep(1000);
        }
    }

    @Override
    public void cancel() {
        running = false;
    }
}

//consumer code
public class ReadFromKafka {
    public static void main(String[] args) throws Exception {
        //Print the reference command of flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");

        //Build the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Configure the parallelism.
        env.setParallelism(1);
        //Parse the execution parameter.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Build the StreamGraph, read data from Kafka and print the result in another row.
        DataStream<String> messageStream = env.addSource(new FlinkKafkaConsumer<>(paraTool.get("topic"),
            new SimpleStringSchema(),
            paraTool.getProperties()));
        messageStream.rebalance().map(new MapFunction<String, String>() {
            @Override
            public String map(String s) throws Exception {
                return "Flink says " + s + System.getProperty("line.separator");
            }
        });
    }
}
```

```
}).print();
//Call execute to trigger the execution.
env.execute();
}
```

### 2.4.3.2.3 Scala Sample Code

#### Function Description

In a Flink application, call API of the **flink-connector-kafka** module to produce and consume data.

#### Sample Code

Following is the main logic code of Kafka Consumer and Kafka Producer.

For the complete code, see `com.huawei.bigdata.flink.examples.WriteIntoKafka` and `com.huawei.bigdata.flink.examples.ReadFromKafka`.

```
//Code of producer
object WriteIntoKafka {
  def main(args: Array[String]) {
    //Print the reference command of flink run.
    System.out.println("use command as:")
    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
      " /opt/test.jar --topic topic-test --bootstrap.servers 10.91.8.218:21005")
    System.out.println("*****")
    System.out.println("<topic> is the kafka topic name")
    System.out.println("<bootstrap.servers> is the ip:port list of brokers")
    System.out.println("*****")

    //Build the execution environment.
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    //Configure the parallelism.
    env.setParallelism(1)
    //Parse the execution parameter.
    val paraTool = ParameterTool.fromArgs(args)
    //Build the StreamGraph and write data generated by customized source into Kafka.
    val messageStream: DataStream[String] = env.addSource(new SimpleStringGenerator)
    messageStream.addSink(new FlinkKafkaProducer(
      paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties()))
    //Call execute to trigger the execution.
    env.execute
  }
}

//Customize source to continuously generate messages every one second.
class SimpleStringGenerator extends SourceFunction[String] {
  var running = true
  var i = 0

  override def run(ctx: SourceContext[String]) {
    while (running) {
      ctx.collect("element-" + i)
      i += 1
      Thread.sleep(1000)
    }
  }

  override def cancel() {
    running = false
  }
}

//consumer code
```

```
object ReadFromKafka {
    def main(args: Array[String]) {
        //Print the reference command of flink run.
        System.out.println("use command as:")
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005")
        System.out.println("*****")
        System.out.println("<topic> is the kafka topic name")
        System.out.println("<bootstrap.servers> is the ip:port list of brokers")
        System.out.println("*****")

        //Build the execution environment
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        //Configure the parallelism.
        env.setParallelism(1)
        //Parse the execution parameter.
        val paraTool = ParameterTool.fromArgs(args)
        //Build the StreamGraph, read data from Kafka and print the result in another row.
        val messageStream = env.addSource(new FlinkKafkaConsumer(
            paraTool.get("topic"), new SimpleStringSchema, paraTool.getProperties))
        messageStream
            .map(s => "Flink says " + s + System.getProperty("line.separator")).print()
        //Call execute to trigger the execution
        env.execute()
    }
}
```

### 2.4.3.3 Asynchronous Checkpoint Mechanism

#### 2.4.3.3.1 Scenarios

##### Scenarios

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status. In this case, when the application is recovered from failure, all operator statuses are the same.

##### Data Planning

1. Customized operators generate about 10000 pieces of data per second.
2. Generated data is of four tuples (Long, String, String, Integer).
3. Statistic results are printed on the devices.
4. Printed data is of the long type.

##### Development Approach

1. The source operator sends 10000 pieces of data and injects the data to the window operator every second.
2. The window operator calculates the data volume of preceding 4 seconds at the interval of one second.
3. The statistics is printed to the device at the interval of one second. Please refer to the specific [Viewing the Debugging Result](#)
4. The checkpoint is triggered at the interval of 6 seconds and the checkpoint result is stored in HDFS.

### 2.4.3.3.2 Java Sample Code

#### Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

#### Sample Code

##### 1. Snapshot data

The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

```
import java.io.Serializable;

//The class is a part of the snapshot and is used to store user-defined statuses.
public class UDFState implements Serializable {
    private long count;

    //Initialize user-defined statuses.
    public UDFState() {
        count = 0L;
    }

    //Configure self-defined statuses.
    public void setState(long count) {
        this.count = count;
    }

    //Obtain user-defined statuses.
    public long getState() {
        return this.count;
    }
}
```

##### 2. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the count variable.

```
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.source.RichSourceFunction;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

//The class is the source operator with checkpoint.
public class SEventSourceWithChk extends RichSourceFunction<Tuple4<Long, String, String, Integer>>
implements ListCheckpointed<UDFState> {
    private Long count = 0L;
    private boolean isRunning = true;
    private String alphabet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnpqrstuvwxyz
wxyzABCDEFGHIJKLMNPQRSTUVWXYZ0987654321";

    //The main logic of the operator is to inject 10000 tuples to the StreamGraph.
    public void run(SourceContext<Tuple4<Long, String, String, Integer>> ctx) throws Exception {
        Random random = new Random();
        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple4.of(random.nextLong(), "hello-" + count, alphabet, 1))
            }
        }
    }
}
```

```
        count++;
    }
    Thread.sleep(1000);
}
}

//Call this when the task is canceled.
public void cancel() {
    isRunning = false;
}

//Customize a snapshot.
public List<UDFState> snapshotState(long l, long ll) throws Exception {
    UDFState udfState = new UDFState();
    List<UDFState> listState = new ArrayList<UDFState>();
    udfState.setState(count);
    listState.add(udfState);
    return listState;
}

//Restore data from customized snapshots.
public void restoreState(List<UDFState> list) throws Exception {
    UDFState udfState = list.get(0);
    count = udfState.getState();
}
}
```

### 3. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```
import org.apache.flink.api.java.tuple.Tuple;
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.windowing.WindowFunction;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;

import java.util.ArrayList;
import java.util.List;

//The class is the window operator with checkpoint.
public class WindowStatisticWithChk implements WindowFunction<Tuple4<Long, String, String, Integer>, Long, Tuple, TimeWindow>, ListCheckpointed<UDFState> {
    private Long total = 0L;

    //The implementation logic of the window operator, which is used to calculate the number of tuples in a window.
    void apply(Tuple key, TimeWindow window, Iterable<Tuple4<Long, String, String, Integer>> input,
               Collector<Long> out) throws Exception {
        long count = 0L;
        for (Tuple4<Long, String, String, Integer> event : input) {
            count++;
        }
        total += count;
        out.collect(count);
    }

    //Customize snapshot.
    public List<UDFState> snapshotState(Long l, Long ll) {
        List<UDFState> listState = new ArrayList<UDFState>();
        UDFState udfState = new UDFState();
        udfState.setState(total);
        listState.add(udfState);
        return listState;
    }

    //Restore data from customized snapshots.
    public void restoreState(List<UDFState> list) throws Exception {
        UDFState udfState = list.get(0);
    }
}
```

```
        total = udfState.getState();
    }
}
```

#### 4. Application code

The code is about the definition of StreamGraph and is used to implement services. The processing time is used as the timestamp for triggering the window.

```
import org.apache.flink.runtime.state.filesystem.FsStateBackend;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;

public class FlinkProcessingTimeAPIChkMain {
    public static void main(String[] args) throws Exception{
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        //Set configurations and enable checkpoint.
        env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"));
        env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
        env.getCheckpointConfig.setCheckpointInterval(6000);

        //Application logic.
        env.addSource(new SEventSourceWithChk()
            .keyBy(0)
            .window(SlidingProcessingTimeWindows.of(Time.seconds(4), Time.seconds(1)))
            .apply(new WindowStatisticWithChk())
            .print()

        env.execute();
    }
}
```

### 2.4.3.3.3 Scala Sample Code

#### Function Description

Assume that you want to collect data volume in the window covering preceding 4 seconds at the interval of one second, and achieve strict consistency of status.

#### Sample Code

##### 1. Formats of sent data.

```
case class SEvent(id: Long, name: String, info: String, count: Int)
```

##### 2. Snapshot data

The snapshot data is used to store number of data pieces recorded by operators during creation of snapshots.

```
//User-defined statuses.

class UDFState extends Serializable{
    private var count = 0L

    //Configure user-defined statuses.
    def setState(s: Long) = count = s

    //Obtain user-defined statuses.
    def getState = count
}
```

##### 3. Data source with checkpoints

Code of the source operator. The code can be used to send 10000 pieces after each pause of one second. When a snapshot is created, number of sent data pieces is recorded in UDFState. When the snapshot is used for restoration, the number of sent data pieces recorded in UDFState is read and assigned to the *count* variable.

```

import java.util
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed
import org.apache.flink.streaming.api.functions.source.RichSourceFunction
import org.apache.flink.streaming.api.functions.source.SourceFunction.SourceContext

//The class is the source operator with checkpoint.
class SEventSourceWithChk extends RichSourceFunction[SEvent] with ListCheckpointed[UDFState]{
    private var count = 0L
    private var isRunning = true
    private val alphabet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnpqrstuvwxyz
wxyzABCDEFGHIJKLMNPQRSTUWZYX0987654321"

    //The logic of the operator is to inject 10000 tuples to the StreamGraph.
    override def run(sourceContext: SourceContext[SEvent]): Unit = {
        while(isRunning) {
            for (i <- 0 until 10000) {
                sourceContext.collect(SEvent(1, "hello-"+count, alphabet,1))
                count += 1L
            }
            Thread.sleep(1000)
        }
    }

    //Call this when the task is canceled.
    override def cancel(): Unit = {
        isRunning = false;
    }

    override def close(): Unit = super.close()

    //Create a snapshot.
    override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {
        val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]
        val udfState = new UDFState
        udfState.setState(count)
        udfList.add(udfState)
        udfList
    }

    //Obtain status from the snapshot.
    override def restoreState(list: util.List[UDFState]): Unit = {
        val udfState = list.get(0)
        count = udfState.getState
    }
}

```

#### 4. Definition of window with checkpoint.

This code is about the window operator and is used to calculate number or tuples in the window.

```
import java.util
import org.apache.flink.api.java.tuple.Tuple
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed
import org.apache.flink.streaming.api.scala.function.WindowFunction
import org.apache.flink.streaming.api.windowing.windows.TimeWindow
import org.apache.flink.util.Collector

//The class is the window operator with checkpoint.
class WindowStatisticWithChk extends WindowFunction[SEvent, Long, Tuple, TimeWindow] with
ListCheckpointed[UDFState]{
    private var total = 0L
```

```

//The implementation logic of the window operator, which is used to calculate the number of
//tuples in a window.
override def apply(key: Tuple, window: TimeWindow, input: Iterable[SEvent], out: Collector[Long]): Unit = {
    var count = 0L
    for (event <- input) {
        count += 1L
    }
    total += count
    out.collect(count)
}

//Customize a snapshot.
override def snapshotState(l: Long, l1: Long): util.List[UDFState] = {
    val udfList: util.ArrayList[UDFState] = new util.ArrayList[UDFState]
    val udfState = new UDFState
    udfState.setState(total)
    udfList.add(udfState)
    udfList
}

//Restore data from customized snapshots.
override def restoreState(list: util.List[UDFState]): Unit = {
    val udfState = list.get(0)
    total = udfState.getState
}
}

```

## 5. Application code

The code is about the definition of StreamGraph and is used to implement services. The **event time** is used as the timestamp for triggering the window.

```

import com.huawei.rt.flink.core.{SEvent, SEventSourceWithChk, WindowStatisticWithChk}
import org.apache.flink.contrib.streaming.state.RocksDBStateBackend
import org.apache.flink.streaming.api.functions.AssignerWithPeriodicWatermarks
import org.apache.flink.streaming.api.{CheckpointingMode, TimeCharacteristic}
import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.api.watermark.Watermark
import org.apache.flink.streaming.api.windowing.assigners.SlidingEventTimeWindows
import org.apache.flink.streaming.api.windowing.time.Time
import org.apache.flink.api.scala._
import org.apache.flink.runtime.state.filesystem.FsStateBackend
import org.apache.flink.streaming.api.environment.CheckpointConfig.ExternalizedCheckpointCleanup

object FlinkEventTimeAPIChkMain {
    def main(args: Array[String]): Unit ={
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"))
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
        env.getConfig.setAutoWatermarkInterval(2000)
        env.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
        env.getCheckpointConfig.setCheckpointInterval(6000)

        //Application logic.
        env.addSource(new SEventSourceWithChk)
            .assignTimestampsAndWatermarks(new AssignerWithPeriodicWatermarks[SEvent] {
                //Configure watermark.
                override def getCurrentWatermark: Watermark = {
                    new Watermark(System.currentTimeMillis())
                }
                //Add timestamp for each tuple.
                override def extractTimestamp(t: SEvent, l: Long): Long = {
                    System.currentTimeMillis()
                }
            })
            .keyBy(0)
            .window(SlidingEventTimeWindows.of(Time.seconds(4), Time.seconds(1)))
            .apply(new WindowStatisticWithChk)
            .print()
        env.execute()
    }
}

```

```
}
```

## 2.4.3.4 Job Pipeline Program

### 2.4.3.4.1 Scenario

The concurrency of NettySource and the concurrency NettySink must be the same. Otherwise, the connection cannot be created.

#### Scenario

Assume that there are three jobs, a publisher and two subscribers. The publisher generates 10000 pieces of data each second. Data is sent by the NettySink operator from the publisher job to downstream subscribers which subscribe a same piece of data.

#### Data Planning

1. The publisher uses customized operators to generate about 10000 pieces of data per second.
2. The data contains two attributes, which are of integer and string types.
3. Configuration file
  - nettyconnector.registerserver.topic.storage: (Mandatory) Configures the path (on a third-party server) to information about IP address, port numbers, and concurrency of NettySink. For example:  
nettyconnector.registerserver.topic.storage: /flink/nettyconnector
  - nettyconnector.sinkserver.port.range: (Mandatory) Configures the range of port numbers of NettySink. For example:  
nettyconnector.sinkserver.port.range: 28444-28943
  - nettyconnector.sinkserver.subnet: Configure the network domain. For example:  
nettyconnector.sinkserver.subnet: 10.162.0.0/16

#### 4. Description of APIs

- RegisterServer API

RegisterServerHandler stores information such as IP address, port number, and concurrency of NettySink for the connection with NettySource. Following APIs are provided for users:

```
public interface RegisterServerHandler {  
  
    /**
     * Start the RegisterServer
     * @param The configuration indicates the configuration type of Flink.  

     */  
    void start(Configuration configuration) throws Exception;  
    /**
     * Create a topic node (directory) on the RegisterServer
     * @param The topic indicates the name of the topic node
     */  
    void createTopicNode(String topic) throw Exception;  
}
```

```
*Register the information to a topic node (directory)
* @param topic @param The topic indicates the directory to be registered with
* @param The registerRecord indicates the information to be registered
*/
void register(String topic, RegisterRecord registerRecord) throws Exception;
/**
     *Delete the topic node.
     *@param The topic indicates the topic to be deleted.
     */
void deleteTopicNode(String topic) throws Exception;
/**
     * Deregister the registration inDeregister the registration information formation
     * @param The topic indicates the topic where the registration information locates.
     * @param The recordId indicates the ID of the registration information to be deregistered
     */
void unregister(String topic, int recordId) throws Exception;
/**
     * Query information
     * @param Topic where the query information locates
     * @param The recordId indicates the ID of the query information
     */
RegisterRecord query(String topic, int recordId) throws Exception;
/**
     * Query whether a topic exist.
     * @param topic
     */
Boolean isExist(String topic) throws Exception;
/**
     *Disable RegisterServerHandler.
     */
void shutdown() throws Exception;
```

In addition to the preceding APIs, Flink provides ZookeeperRegisterHandler.

- NettySink operator

```
Class NettySink(String name,
String topic,
RegisterServerHandler registerServerHandler,
int numberOfSubscribedJobs)
```

- name: Name of a current NettySink.
- topic: The topic that generates data for the current NettySink. Different NettySinks must use different topics. Otherwise, the subscription may be disordered and data transmission may be abnormal.
- registerServerHandler: Handler of the registration server.
- numberOfSubscribedJobs: Specific number of jobs that subscribe the current NettySink. NettySink sends data only when all subscribers are connected to NettySink.

- NettySource operator

```
Class NettySource(String name,
String topic,
RegisterServerHandler registerServerHandler)
```

- name: name of the NettySource. The NettySource must be unique (concurrency excluded). Otherwise, connection with NettySink may be conflicted.

- topic: topic of subscribed NettySink.
- registerServerHandler: Handler of the registration server.

 NOTE

The concurrency of NettySource and the concurrency NettySink must be the same. Otherwise, the connection cannot be created.

## Development Approach

1. There are three jobs, one publisher and two subscribers.
2. The publisher transforms generated data into byte[] and send them to the subscribers.
3. After receiving the byte[], subscribers transform data into the string type and print sampled data.

### 2.4.3.4.2 Java Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- com.huawei.bigdata.flink.examples.UserSource.
- com.huawei.bigdata.flink.examples.TestPipelineNettySink.
- com.huawei.bigdata.flink.examples.TestPipelineNettySource1.
- com.huawei.bigdata.flink.examples.TestPipelineNettySource2.

1. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;

import java.io.Serializable;

public class UserSource extends RichParallelSourceFunction<Tuple2<Integer, String>> implements Serializable {

    private boolean isRunning = true;

    public void open(Configuration configuration) throws Exception {
        super.open(configuration);
    }

    /**
     * Data generation function, which is used to generate 10000 pieces of data each second.
     */
    public void run(SourceContext<Tuple2<Integer, String>> ctx) throws Exception {

        while(isRunning) {
            for (int i = 0; i < 10000; i++) {
                ctx.collect(Tuple2.of(i, "hello-" + i));
            }
            Thread.sleep(1000);
        }
    }

    public void close() {
```

```
        isRunning = false;
    }

    public void cancel() {
        isRunning = false;
    }
}
```

## 2. Code for the publisher:

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.sink.NettySink;
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;

public class TestPipelineNettySink {

    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency of job to 2.
        env.setBufferTimeout(2);

        //Create a ZookeeperRegisterServerHandler.
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
        // Add a customized source operator.
        env.addSource(new UserSource()
            .keyBy(0)
            .map(new MapFunction<Tuple2<Integer,String>, byte[]>() {
                //Transform the to-be-sent data into a byte array.

@Override
        public byte[] map(Tuple2<Integer, String> integerStringTuple2) throws Exception {
            return integerStringTuple2.f1.getBytes();
        }
    }).addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2));//NettySink
transmits the data.

        env.execute();
    }
}
```

## 3. Code for the first subscriber.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.source.NettySource;
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;

import java.nio.charset.Charset;

public class TestPipelineNettySource1 {

    public static void main(String[] args) throws Exception{

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Set the concurrency of job to 2.
        env.setParallelism(2);

        // Create a ZookeeperRegisterServerHandler.
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
        //Add a NettySource operator to receive messages from the publisher.
        env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler))
            .map(new MapFunction<byte[], String>()
```

```
// Transform the received byte stream into character strings
@Override
    public String map(byte[] bytes) {
        return new String(bytes, Charset.forName("UTF-8"));
    }
}).print();

env.execute();
}
}
```

#### 4. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.netty.source.NettySource;
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler;

import java.nio.charset.Charset;

public class TestPipelineNettySource2 {

    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Set the concurrency of job to 2.
        env.setParallelism(2);

        //Create a ZookeeperRegisterServerHandler.
        ZookeeperRegisterServerHandler zkRegisterServerHandler = new ZookeeperRegisterServerHandler();
        //Add a NettySource operator to receive messages from the publisher.
        env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler))
            .map(new MapFunction<byte[], String>() {
                //Transform the received byte array into character strings.
                @Override
                public String map(byte[] bytes) {
                    return new String(bytes, Charset.forName("UTF-8"));
                }
            }).print();

        env.execute();
    }
}
```

#### 2.4.3.4.3 Scala Sample Code

Following is the main logic code for demonstration.

For details about the complete code, see the following:

- com.huawei.bigdata.flink.examples.UserSource.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySink.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySource1.
- com.huawei.bigdata.flink.examples.TestPipeline\_NettySource2.

##### 1. Code for sending messages:

```
package com.huawei.bigdata.flink.examples

case class Inforamtion(index: Int, content: String) {

    def this() = this(0, "")
}
```

##### 2. The publisher customizes source operators to generate data.

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.configuration.Configuration
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction
import org.apache.flink.streaming.api.functions.source.SourceFunction.SourceContext

class UserSource extends RichParallelSourceFunction[Inforamtion] with Serializable{

    var isRunning = true

    override def open(parameters: Configuration): Unit = {
        super.open(parameters)
    }

    // Generate 10000 pieces of data each second.
    override def run(sourceContext: SourceContext[Inforamtion]) = {

        while (isRunning) {
            for (i <- 0 until 10000) {
                sourceContext.collect(Inforamtion(i, "hello-" + i));
            }
            Thread.sleep(1000)
        }
    }

    override def close(): Unit = super.close()

    override def cancel() = {
        isRunning = false
    }
}
```

### 3. Code for the publisher:

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.sink.NettySink
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

object TestPipeline_NettySink {

    def main(args: Array[String]): Unit = {

        val env = StreamExecutionEnvironment.getExecutionEnvironment
        // Set the concurrency of job to 2.
        env.setParallelism(2)
        //Set Zookeeper as the registration server
        val zkRegisterServerHandler = new ZookeeperRegisterServerHandler
        //Add a user-defined operator to generate data.
        env.addSource(new UserSource) .keyBy(0).map(x=>x.content.getBytes)//Transform the to-be-sent data
        into a byte array.
        .addSink(new NettySink("NettySink-1", "TOPIC-2", zkRegisterServerHandler, 2))//Add NettySink
        operator to send data.
        env.execute()
    }
}
```

### 4. Code for the first subscriber

```
package com.huawei.bigdata.flink.examples

import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment
import org.apache.flink.streaming.connectors.netty.source.NettySource
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler
import org.apache.flink.streaming.api.scala._

import scala.util.Random
```

```
object TestPipeline_NettySource1 {  
    def main(args: Array[String]): Unit = {  
        val env = StreamExecutionEnvironment.getExecutionEnvironment  
        // Set the concurrency of job to 2.  
        env.setParallelism(2)  
        //Set ZooKeeper as the registration server  
        val zkRegisterServerHandler = new ZookeeperRegisterServerHandler  
        //Add a NettySource operator to receive messages from the publisher.  
        env.addSource(new NettySource("NettySource-1", "TOPIC-2", zkRegisterServerHandler))  
            .map(x => (1, new String(x)))//Add a NettySource operator to receive messages from the publisher.  
            .filter(x => {  
                Random.nextInt(50000) == 10  
            })  
            .print  
  
        env.execute()  
    }  
}
```

## 5. Code for the second subscriber.

```
package com.huawei.bigdata.flink.examples  
  
import org.apache.flink.streaming.api.scala.StreamExecutionEnvironment  
import org.apache.flink.streaming.connectors.netty.source.NettySource  
import org.apache.flink.streaming.connectors.netty.utils.ZookeeperRegisterServerHandler  
import org.apache.flink.streaming.api.scala._  
  
import scala.util.Random  
  
object TestPipeline_NettySource2 {  
    def main(args: Array[String]): Unit = {  
        val env = StreamExecutionEnvironment.getExecutionEnvironment  
        //Set the concurrency of job to 2.  
        env.setParallelism(2)  
        //Set the concurrency of job to 2.  
        val zkRegisterServerHandler = new ZookeeperRegisterServerHandler  
        //Add NettySource operator and receive data.  
  
        env.addSource(new NettySource("NettySource-2", "TOPIC-2", zkRegisterServerHandler))  
            .map(x=>(2, new String(x)))//Transform the received byte array into character strings.  
  
            .filter(x=>{  
                Random.nextInt(50000) == 10  
            })  
            .print()  
  
        env.execute()  
    }  
}
```

## 2.4.3.5 JOIN Operation between Configuration Tables and Streams

### 2.4.3.5.1 Scenario Description

#### Scenario

Assume that there is a log text file about dwell durations of netizens for shopping online and a CSV table about netizen information. Develop a Flink application that achieves following functions:

- Collects statistics on female netizens who spend more than 2 hours in total for online shopping during the weekend in real time.  
The name fields in the log file and the CSV table can be used as keywords, based on which the two files are joined.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration (in minutes). Three columns are separated by commas (,).

**data.txt:** logs of netizen dwell duration on weekends

```
LA,female,20
YA,male,10
GA,male,5
CA,female,50
LAA,male,20
FA,female,50
LA,female,20
YA,male,10
GA,male,50
CA,female,50
FA,female,60
LA,female,20
YA,male,10
CA,female,50
FA,female,50
GA,male,5
CA,female,50
LB,male,20
CA,female,50
FA,female,50
LA,female,20
YA,male,10
FA,female,50
GA,male,50
CA,female,50
FA,female,60
NotExist,female,200
```

- **configtable.csv:** contains netizens' personal information. Fields starting from the first column to the ninth column are the name, age, company name, work place, academic degree, years of working, mobile number, domicile place, and school of graduation. These columns are separated by commas (,).

```
username,age,company,workLocation,educational,workYear,phone,nativeLocation,school
LA,25,C1,W1,college,5,1312345678,N1,S1 university
YA,26,C2,W2,master,6,1312345679,N2,S2 university
GA,27,C3,W3,college,7,1312345680,N3,S3 university
CA,28,C4,W4,master,8,1312345681,N4,S4 university
LB,29,C5,W5,doctor,9,1312345682,N5,S5 university
FA,30,C6,W6,doctor,10,1312345683,N6,S6 university
```

## Data Preparation

The stream data of the example project is saved in a TXT file, and the configuration table is in CSV format.

1. Ensure that clusters, including HDFS, YARN, non-secure Redis, and Flink are successfully installed.
2. Create a Redis cluster. For details about Redis, see "Component Operation Guide" > "Using Redis" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).
3. Modify the **import.properties** and **read.properties** files, which are in the **config** directory specified in the sample code.

#### NOTE

In the Scala sample code, the IP addresses and port numbers of all the instances in the logical cluster must be configured for the **Redis\_IP\_Port** parameter.

The formula for calculating the port number of the Redis instance is:  $22400 + \text{Instance ID} - 1$ .

To obtain the instance ID, choose **Cluster > Name of the desired cluster > Service > Redis > Redis Manager** on FusionInsight Manager and click the target Redis cluster name. For example, the port numbers of the Redis instances corresponding to the role R1 and role R2 are 22400 ( $22400 + 1 - 1 = 22400$ ) and 22401 ( $22400 + 2 - 1 = 22401$ ), respectively.

- The following is an example of configurations in the **import.properties** file:

```
#path to read csv files, it can be file or directory  
CsvPath=config/configtable.csv  
  
#csv file headers exist in file first line or not  
CsvHeaderExist=true  
#csv file headers, also the redis field names  
#Notice: if CsvHeaderExist false, you must set it, if CsvHeaderExist true, it read from csv file  
ColumnNames=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school  
#redis hostname/ip and port when you need to connect to redis  
Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400  
# redis ssl on  
Redis_ssl_on=false
```

- The following is an example of configurations in the **read.properties** file:

```
#the redis field names, configure which fields you need to read, Notice you need English field  
names  
ReadFields=username,age,company,workLocation,educational,workYear,phone,nativeLocation,sch  
ool  
#redis hostname/ip and port when you need to connect to redis  
Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400  
# redis ssl on  
Redis_ssl_on=false
```

4. Create the **config** directory on the Flink client and save the **data.txt**, **configtable.csv**, **import.properties** and **read.properties** files to the **config** directory, for example, `/opt/hadoopclient/Flink/flink/config/configtable.csv`.

#### NOTE

The **data.txt** and **configtable.csv** files are in the **data** directory specified in the sample code.

## Development Guideline

Collect the detailed information about female netizens who continuously spend more than 2 hours on online shopping on the current weekend.

To achieve the objective, the process is as follows:

- Modify the configuration of **import.properties** and **read.properties**, configure fields in the CSV file, fields read by Redis, and information about Redis nodes.
- Import the **configtable.csv** table to Redis for storage.
- Read text data, generate DataStreams, and parse data to generate OriginalRecord information.

- Call the function of asynchronous I/O, use the OriginalRecord username field as the keyword to query personal information in Redis, and transform the information into UserRecord.
- Filter the data about the time that female netizens spend online.
- Perform keyby operation based on names, and collect the time that female netizens spend online within a time window.
- Filter data about netizens whose consecutive online duration exceeds the threshold, and obtain the results.

#### 2.4.3.5.2 Java Sample Code

##### Description

Collect the information from the log file about female netizens who continuously spend more than 2 hours in online shopping, collect personal information from the CSV table, and use the names of netizens as the keywords for joining the log file and the CSV table.

##### Sample Code

The following code snippet is used as an example for importing the **configtable.csv** file to Redis clusters in normal mode. For the complete code, see [com.huawei.bigdata.flink.examples.RedisDataImport](#).

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.java.utils.ParameterTool;

import org.supercsv.cellprocessor.constraint.NotNull;
import org.supercsv.cellprocessor.ift.CellProcessor;
import org.supercsv.io.CsvBeanReader;
import org.supercsv.io.ICsvBeanReader;
import org.supercsv.prefs.CsvPreference;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;

import java.io.File;
import java.io.FileReader;
import java.util.*;

/**
 * Read data from csv file and import to redis.
 */
public class RedisDataImport {

    private static final int MAX_ATTEMPTS = 2;

    private static final int TIMEOUT = 60000;

    public static void main(String[] args) throws Exception {
        // print comment for command to use run flink
        System.out.println("use command as: \n" +
            "java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/FlinkConfigtableJavaExample.jar" +
            " com.huawei.bigdata.flink.examples.RedisDataImport --configPath <config filePath>" +
            "*****\n" +
            "<config filePath> is for configure file to load\n" +
            "you may write following content into config filePath: \n" +
            "CsvPath=config/configtable.csv\n" +
            "CsvHeaderExist=true\n" +
            "\n" +
            "ColumnNames=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school\n"
    }
}
```

```
+        "Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
+*****");
+
// read all configures
final String configureFilePath = ParameterTool.fromArgs(args).get("configPath", "config/
import.properties");
final String csvFilePath = ParameterTool.fromPropertiesFile(configureFilePath).get("CsvPath", "config/
configtable.csv");
final boolean isHasHeaders =
ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("CsvHeaderExist", true);
final String csvScheme = ParameterTool.fromPropertiesFile(configureFilePath).get("ColumnNames");
final String redisIPPort = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_IP_Port");
final boolean ssl = ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("Redis_ssl_on",
false);

// init redis client
Set<HostAndPort> hosts = new HashSet<HostAndPort>();
for (String hostAndPort : redisIPPort.split(",")) {
    hosts.add(new HostAndPort(hostAndPort.split(":")[0], Integer.parseInt(hostAndPort.split(":")[1])));
}

JedisPoolConfig poolConfig = new JedisPoolConfig();
final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();
final JedisCluster client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS,"","");
poolConfig, ssl, socketFactory, null,null, null);

// get all files under csv file path
ArrayList<File> files = getListFiles(csvFilePath);
System.out.println("Read file or directory under " + csvFilePath
+ ", total file num: " + files.size() + ", columns: " + csvScheme);

// run read csv file and analyze it
for (int index = 0; index < files.size(); index++) {
    readWithCsvBeanReader(files.get(index).getAbsolutePath(), csvScheme, isHasHeaders, client);
}
client.close();
System.out.println("Data import finish!!!");

}

public static ArrayList<File> getListFiles(Object obj) {
    File directory = null;
    if (obj instanceof File) {
        directory = (File) obj;
    } else {
        directory = new File(obj.toString());
    }
    ArrayList<File> files = new ArrayList<File>();
    if (directory.isFile()) {
        files.add(directory);
        return files;
    } else if (directory.isDirectory()) {
        File[] fileArr = directory.listFiles();
        for (int i = 0; i < fileArr.length; i++) {
            File fileOne = fileArr[i];
            files.addAll(getListFiles(fileOne));
        }
    }
    return files;
}

/**
 * Sets up the processors used for read csv. There are 9 CSV columns. Empty
 * columns are read as null (hence the NotNull() for mandatory columns).
 *
 * @return the cell processors
 */
private static CellProcessor[] getProcessors() {
```

```
final CellProcessor[] processors = new CellProcessor[] {
    new NotNull(), // username
    new NotNull(), // age
    new NotNull(), // company
    new NotNull(), // workLocation
    new NotNull(), // educational
    new NotNull(), // workYear
    new NotNull(), // phone
    new NotNull(), // nativeLocation
    new NotNull(), // school
};

return processors;
}

private static void readWithCsvBeanReader(String path, String csvScheme, boolean isSkipHeader,
JedisCluster client) throws Exception {
    ICsvBeanReader beanReader = null;
    try {
        beanReader = new CsvBeanReader(new FileReader(path), CsvPreference.STANDARD_PREFERENCE);

        // the header elements are used to map the values to the bean (names must match)
        final String[] header = isSkipHeader ? beanReader.getHeader(true) : csvScheme.split(",");
        final CellProcessor[] processors = getProcessors();

        UserInfo userinfo;
        while( (userinfo = beanReader.read(UserInfo.class, header, processors)) != null ) {
            System.out.println(String.format("lineNo=%s, rowNo=%s, userinfo=%s",
                beanReader.getLineNumber(),
                beanReader.getRowNumber(), userinfo));

            // set redis key and value
            client.hmset(userinfo.getKeyValue(), userinfo.getMapInfo());
        }
    } finally {
        if( beanReader != null ) {
            beanReader.close();
        }
    }
}

// define the UserInfo structure
public static class UserInfo {
    private String username;
    private String age;
    private String company;
    private String workLocation;
    private String educational;
    private String workYear;
    private String phone;
    private String nativeLocation;
    private String school;

    public UserInfo() {

    }

    public UserInfo(String nm, String a, String c, String w, String e, String wy, String p, String nl, String sc) {
        username = nm;
        age = a;
        company = c;
        workLocation = w;
        educational = e;
        workYear = wy;
        phone = p;
        nativeLocation = nl;
        school = sc;
    }
}
```

```
}

public String toString() {
    return "UserInfo----[username: " + username + " age: " + age + " company: " + company
           + " workLocation: " + workLocation + " educational: " + educational
           + " workYear: " + workYear + " phone: " + phone + " nativeLocation: " + nativeLocation + "
school: " + school + "]";
}

// get key
public String getKeyValue() {
    return username;
}

public Map<String, String> getMapInfo() {
    Map<String, String> info = new HashMap<String, String>();
    info.put("username", username);
    info.put("age", age);
    info.put("company", company);
    info.put("workLocation", workLocation);
    info.put("educational", educational);
    info.put("workYear", workYear);
    info.put("phone", phone);
    info.put("nativeLocation", nativeLocation);
    info.put("school", school);
    return info;
}

/**
 * @return the username
 */
public String getUsername() {
    return username;
}

/**
 * @param username
 *          the username to set
 */
public void setUsername(String username) {
    this.username = username;
}

/**
 * @return the age
 */
public String getAge() {
    return age;
}

/**
 * @param age
 *          the age to set
 */
public void setAge(String age) {
    this.age = age;
}

/**
 * @return the company
 */
public String getCompany() {
    return company;
}

/**
 * @param company
 *          the company to set
 */
}
```

```
public void setCompany(String company) {
    this.company = company;
}

/**
 * @return the workLocation
 */
public String getWorkLocation() {
    return workLocation;
}

/**
 * @param workLocation
 *      the workLocation to set
 */
public void setWorkLocation(String workLocation) {
    this.workLocation = workLocation;
}

/**
 * @return the educational
 */
public String getEducational() {
    return educational;
}

/**
 * @param educational
 *      the educational to set
 */
public void setEducational(String educational) {
    this.educational = educational;
}

/**
 * @return the workYear
 */
public String getWorkYear() {
    return workYear;
}

/**
 * @param workYear
 *      the workYear to set
 */
public void setWorkYear(String workYear) {
    this.workYear = workYear;
}

/**
 * @return the phone
 */
public String getPhone() {
    return phone;
}

/**
 * @param phone
 *      the phone to set
 */
public void setPhone(String phone) {
    this.phone = phone;
}

/**
 * @return the nativeLocation
 */
public String getNativeLocation() {
    return nativeLocation;
}
```

```
    }

    /**
     * @param nativeLocation
     *      the nativeLocation to set
     */
    public void setNativeLocation(String nativeLocation) {
        this.nativeLocation = nativeLocation;
    }

    /**
     * @return the school
     */
    public String getSchool() {
        return school;
    }

    /**
     * @param school
     *      the school to set
     */
    public void setSchool(String school) {
        this.school = school;
    }
}
```

The following code snippet is used as an example. The function of the following code is to read stream data from the **log.txt** file, use netizen names as keywords to query from Redis clusters in normal mode, perform the JOIN operation, and print the results. For the complete code, see [com.huawei.bigdata.flink.examples.FlinkConfigtableJavaExample](#).

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.common.functions.FilterFunction;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.common.functions.ReduceFunction;
import org.apache.flink.api.java.functions.KeySelector;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.shaded.com.google.common.cache.CacheBuilder;
import org.apache.flink.shaded.com.google.common.cache.CacheLoader;
import org.apache.flink.shaded.com.google.common.cache.LoadingCache;
import org.apache.flink.streaming.api.datastream.AsyncDataStream;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.functions.AssignerWithPunctuatedWatermarks;
import org.apache.flink.streaming.api.functions.async.AsyncFunction;
import org.apache.flink.streaming.api.functions.async.RichAsyncFunction;
import org.apache.flink.streaming.api.functions.async.collector.AsyncCollector;
import org.apache.flink.streaming.api.watermark.Watermark;
import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;

import java.util.*;
import java.util.concurrent.TimeUnit;
/**
 * Read stream data and join from configure table from redis.
 */
public class FlinkConfigtableJavaExample {

    private static final int MAX_ATTEMPTS = 2;

    private static final int TIMEOUT = 60000;
```

```
public static void main(String[] args) throws Exception {
    // print comment for command to use run flink
    System.out.println("use command as: \n" +
        "./bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableJavaExample" +
        " -m yarn-cluster -yt /opt/config -yn 3 -yjm 1024 -ytm 1024 " +
        "/opt/FlinkConfigtableJavaExample.jar --dataPath config/data.txt" +
        "*****\n" +
        "Especially you may write following content into config filePath, as in config/read.properties: \n"
    +
    "ReadFields=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school\n" +
    "Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
    "*****");
}

// set up the execution environment
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
env.setParallelism(1);

// get configure and read data and transform to OriginalRecord
final String dataPath = ParameterTool.fromArgs(args).get("dataPath", "config/data.txt");
DataStream<OriginalRecord> originalStream = env.readTextFile(
    dataPath
).map(new MapFunction<String, OriginalRecord>() {
    @Override
    public OriginalRecord map(String value) throws Exception {
        return getRecord(value);
    }
}).assignTimestampsAndWatermarks(
    new Record2TimestampExtractor()
).disableChaining();

// read from redis and join to the whole user information
AsyncFunction<OriginalRecord, UserRecord> function = new AsyncRedisRequest();
// timeout set to 2 minutes, max parallel request num set to 5, you can modify this to optimize
DataStream<UserRecord> result = AsyncDataStream.unorderedWait(
    originalStream,
    function,
    2,
    TimeUnit.MINUTES,
    5);

// data transform
result.filter(new FilterFunction<UserRecord>() {
    @Override
    public boolean filter(UserRecord value) throws Exception {
        return value.sex.equals("female");
    }
}).keyBy(
    new UserRecordSelector()
).window(
    TumblingEventTimeWindows.of(Time.seconds(30))
).reduce(new ReduceFunction<UserRecord>() {
    @Override
    public UserRecord reduce(UserRecord value1, UserRecord value2)
        throws Exception {
        value1.shoppingTime += value2.shoppingTime;
        return value1;
    }
}).filter(new FilterFunction<UserRecord>() {
    @Override
    public boolean filter(UserRecord value) throws Exception {
        return value.shoppingTime > 120;
    }
}).print();

// execute program
env.execute("FlinkConfigtable java");
```

```
}

private static class UserRecordSelector implements KeySelector<UserRecord, String> {
    @Override
    public String getKey(UserRecord value) throws Exception {
        return value.name;
    }
}

// class to set watermark and timestamp
private static class Record2TimestampExtractor implements
AssignerWithPunctuatedWatermarks<OriginalRecord> {

    // add tag in the data of datastream elements
    @Override
    public long extractTimestamp(OriginalRecord element, long previousTimestamp) {
        return System.currentTimeMillis();
    }

    // Give the watermark to trigger the window to start execution, and use the value to check if the
    // window elements are ready.
    @Override
    public Watermark checkAndGetNextWatermark(OriginalRecord element, long extractedTimestamp) {
        return new Watermark(extractedTimestamp - 1);
    }
}

private static OriginalRecord getRecord(String line) {
    String[] elems = line.split(",");
    assert elems.length == 3;
    return new OriginalRecord(elems[0], elems[1], Integer.parseInt(elems[2]));
}

public static class OriginalRecord {
    private String name;
    private String sexy;
    private int shoppingTime;

    public OriginalRecord(String n, String s, int t) {
        name = n;
        sexy = s;
        shoppingTime = t;
    }
}

public static class UserRecord {
    private String name;
    private int age;
    private String company;
    private String workLocation;
    private String educational;
    private int workYear;
    private String phone;
    private String nativeLocation;
    private String school;
    private String sexy;
    private int shoppingTime;

    public UserRecord(String nm, int a, String c, String w, String e, int wy, String p, String nl, String sc,
String sx, int st) {
        name = nm;
        age = a;
        company = c;
        workLocation = w;
        educational = e;
        workYear = wy;
        phone = p;
        nativeLocation = nl;
        school = sc;
    }
}
```

```

        sexy = sx;
        shoppingTime = st;
    }

    public void setInput(String input_nm, String input_sx, int input_st) {
        name = input_nm;
        sexy = input_sx;
        shoppingTime = input_st;
    }

    public String toString() {
        return "UserRecord----name: " + name + " age: " + age + " company: " + company
            + " workLocation: " + workLocation + " educational: " + educational
            + " workYear: " + workYear + " phone: " + phone + " nativeLocation: " + nativeLocation + "
school: " + school
            + " sexy: " + sexy + " shoppingTime: " + shoppingTime;
    }
}

public static class AsyncRedisRequest extends RichAsyncFunction<OriginalRecord, UserRecord>{
    private String fields = "";
    private transient JedisCluster client;
    private LoadingCache<String, UserRecord> cacheRecords;

    @Override
    public void open(Configuration parameters) throws Exception {
        super.open(parameters);

        // init cache builder
        cacheRecords = CacheBuilder.newBuilder()
            .maximumSize(10000)
            .expireAfterAccess(7, TimeUnit.DAYS)
            .build(new CacheLoader<String, UserRecord>() {
                public UserRecord load(String key) throws Exception {
                    //load from redis
                    return loadFromRedis(key);
                }
            });
    }

    // Get configurations from config/read.properties. You must put this with commands:
    // ./bin/yarn-session.sh -t config -jm 1024 -tm 1024 or
    // ./bin/flink run -m yarn-cluster -yt config -yn 3 -yjm 1024 -ytm 1024 /opt/test.jar
    String configPath = "config/read.properties";
    fields = ParameterTool.fromPropertiesFile(configPath).get("ReadFields");
    final String hostPort = ParameterTool.fromPropertiesFile(configPath).get("Redis_IP_Port");
    final boolean ssl = ParameterTool.fromPropertiesFile(configPath).getBoolean("Redis_ssl_on", false);

    // create jedisCluster client
    Set<HostAndPort> hosts = new HashSet<HostAndPort>();
    for (String node : hostPort.split(",")) {
        hosts.add(new HostAndPort(node.split(":")[0], Integer.parseInt(node.split(":")[1])));
    }

    JedisPoolConfig poolConfig = new JedisPoolConfig();
    final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();
    client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS, "", "", poolConfig, ssl,
    socketFactory, null, null, null);

    System.out.println("JedisCluster init, getClusterNodes: " + client.getClusterNodes().size());
}

@Override
public void close() throws Exception {
    super.close();

    if (client != null) {
        System.out.println("JedisCluster close!!!!");
    }
}

```

```
        client.close();
    }

    public UserRecord loadFromRedis(final String key) throws Exception {
        if (client.getClusterNodes().size() <= 0) {
            System.out.println("JedisCluster init failed, getClusterNodes: " + client.getClusterNodes().size());
        }
        if (!client.exists(key)) {
            System.out.println("test-----cannot find data to key: " + key);
            return new UserRecord(
                "null",
                0,
                "null",
                "null",
                "null",
                "null",
                0,
                "null",
                "null",
                "null",
                "null",
                "null",
                0);
        } else {
            // get some fields
            List<String> values = client.hmget(key, fields.split(","));
            System.out.println("test-----key: " + key + " get some fields: " + values.toString());
            return new UserRecord(
                values.get(0),
                Integer.parseInt(values.get(1)),
                values.get(2),
                values.get(3),
                values.get(4),
                Integer.parseInt(values.get(5)),
                values.get(6),
                values.get(7),
                values.get(8),
                "null",
                0);
        }
    }

    public void asyncInvoke(final OriginalRecord input, final AsyncCollector<UserRecord> collector) throws
Exception {
        // Set key string, if your key is more than one column, build your key string with columns.
        String key = input.name;
        UserRecord info = cacheRecords.get(key);
        info.setInput(input.name, input.sex, input.shoppingTime);
        collector.collect(Collections.singletonList(info));
    }
}
```

#### 2.4.3.5.3 Scala Sample Code

##### Description

Collect the information from the log file about female netizens who continuously spend more than 2 hours in online shopping, collect personal information from the CSV table, and use the names of netizens as the keywords for joining the log file and the CSV table.

##### Sample Code

The following code snippet is used as an example for importing the **configurable.csv** file to Redis clusters in normal mode. For the complete code, see [com.huawei.bigdata.flink.examples.RedisDataImport](#).

```
package com.huawei.bigdata.flink.examples;

import org.apache.flink.api.java.utils.ParameterTool;

import org.supercsv.cellprocessor.constraint.NotNull;
import org.supercsv.cellprocessor.ift.CellProcessor;
import org.supercsv.io.CsvBeanReader;
import org.supercsv.io.ICsvBeanReader;
import org.supercsv.prefs.CsvPreference;
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;

import java.io.File;
import java.io.FileReader;
import java.util.*;

/**
 * Read data from csv file and import to redis.
 */
public class RedisDataImport {

    public static void main(String[] args) throws Exception {
        // print comment for command to use run flink
        System.out.println("use command as: \n" +
            "java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/FlinkConfigtableScalaExample.jar" +
            " com.huawei.bigdata.flink.examples.RedisDataImport --configPath <config filePath>" +
            "*****\n" +
            "<config filePath> is for configure file to load\n" +
            "you may write following content into config filePath: \n" +
            "CsvPath=config/configtable.csv\n" +
            "CsvHeaderExist=true\n" +
            "Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
            "*****");

        // read all configures
        final String configureFilePath = ParameterTool.fromArgs(args).get("configPath", "config/
import.properties");
        final String csvFilePath = ParameterTool.fromPropertiesFile(configureFilePath).get("CsvPath", "config/
configtable.csv");
        final boolean isHasHeaders =
ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("CsvHeaderExist", true);
        final String csvScheme = ParameterTool.fromPropertiesFile(configureFilePath).get("ColumnNames");
        final String redisIPPort = ParameterTool.fromPropertiesFile(configureFilePath).get("Redis_IP_Port");
        final boolean ssl = ParameterTool.fromPropertiesFile(configureFilePath).getBoolean("Redis_ssl_on",
false);

        // init redis client
Set<HostAndPort> hosts = new HashSet<HostAndPort>();
for (String hostAndPort : redisIPPort.split(",")) {
    hosts.add(new HostAndPort(hostAndPort.split(":")[0], Integer.parseInt(hostAndPort.split(":")[1])));
}

JedisPoolConfig poolConfig = new JedisPoolConfig();
final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();
final JedisCluster client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS,"","");
poolConfig, ssl, socketFactory, null,null, null);

        // get all files under csv file path
ArrayList<File> files = getListFiles(csvFilePath);
System.out.println("Read file or directory under " + csvFilePath
+ ", total file num: " + files.size() + ", columns: " + csvScheme);

        // run read csv file and analyze it
        for (int index = 0; index < files.size(); index++) {
            readWithCsvBeanReader(files.get(index).getAbsolutePath(), csvScheme, isHasHeaders, client);
        }
    }
}
```

```
        }
        client.close();
        System.out.println("Data import finish!!!!");
    }

    public static ArrayList<File> getListFiles(Object obj) {
        File directory = null;
        if (obj instanceof File) {
            directory = (File) obj;
        } else {
            directory = new File(obj.toString());
        }
        ArrayList<File> files = new ArrayList<File>();
        if (directory.isFile()) {
            files.add(directory);
            return files;
        } else if (directory.isDirectory()) {
            File[] fileArr = directory.listFiles();
            for (int i = 0; i < fileArr.length; i++) {
                File fileOne = fileArr[i];
                files.addAll(getListFiles(fileOne));
            }
        }
        return files;
    }

    /**
     * Sets up the processors used for read csv. There are 9 CSV columns. Empty
     * columns are read as null (hence the NotNull() for mandatory columns).
     *
     * @return the cell processors
     */
    private static CellProcessor[] getProcessors() {
        final CellProcessor[] processors = new CellProcessor[] {
            new NotNull(), // username
            new NotNull(), // age
            new NotNull(), // company
            new NotNull(), // workLocation
            new NotNull(), // educational
            new NotNull(), // workYear
            new NotNull(), // phone
            new NotNull(), // nativeLocation
            new NotNull(), // school
        };
        return processors;
    }

    private static void readWithCsvBeanReader(String path, String csvScheme, boolean isSkipHeader,
JedisCluster client) throws Exception {
    ICsvBeanReader beanReader = null;
    try {
        beanReader = new CsvBeanReader(new FileReader(path), CsvPreference.STANDARD_PREFERENCE);

        // the header elements are used to map the values to the bean (names must match)
        final String[] header = isSkipHeader ? beanReader.getHeader(true) : csvScheme.split(",");
        final CellProcessor[] processors = getProcessors();

        UserInfo userinfo;
        while( (userinfo = beanReader.read(UserInfo.class, header, processors)) != null ) {
            System.out.println(String.format("lineNo=%s, rowNo=%s, userinfo=%s",
beanReader.getLineNumber(),
            beanReader.getRowNumber(), userinfo));

            // set redis key and value
            client.hmset(userinfo.getKeyValue(), userinfo.getMapInfo());
        }
    }
    finally {

```

```
        if( beanReader != null ) {
            beanReader.close();
        }
    }

// define the UserInfo structure
public static class UserInfo {
    private String username;
    private String age;
    private String company;
    private String workLocation;
    private String educational;
    private String workYear;
    private String phone;
    private String nativeLocation;
    private String school;

    public UserInfo() {
    }

    public UserInfo(String nm, String a, String c, String w, String e, String wy, String p, String nl, String sc) {
        username = nm;
        age = a;
        company = c;
        workLocation = w;
        educational = e;
        workYear = wy;
        phone = p;
        nativeLocation = nl;
        school = sc;
    }

    public String toString() {
        return "UserInfo----[username: " + username + " age: " + age + " company: " + company
               + " workLocation: " + workLocation + " educational: " + educational
               + " workYear: " + workYear + " phone: " + phone + " nativeLocation: " + nativeLocation + "
school: " + school + "]";
    }

    // get key
    public String getKeyValue() {
        return username;
    }

    public Map<String, String> getMapInfo() {
        Map<String, String> info = new HashMap<String, String>();
        info.put("username", username);
        info.put("age", age);
        info.put("company", company);
        info.put("workLocation", workLocation);
        info.put("educational", educational);
        info.put("workYear", workYear);
        info.put("phone", phone);
        info.put("nativeLocation", nativeLocation);
        info.put("school", school);
        return info;
    }

    /**
     * @return the username
     */
    public String getUsername() {
        return username;
    }
}
```

```
/*
 * @param username
 *      the username to set
 */
public void setUsername(String username) {
    this.username = username;
}

/*
 * @return the age
 */
public String getAge() {
    return age;
}

/*
 * @param age
 *      the age to set
 */
public void setAge(String age) {
    this.age = age;
}

/*
 * @return the company
 */
public String getCompany() {
    return company;
}

/*
 * @param company
 *      the company to set
 */
public void setCompany(String company) {
    this.company = company;
}

/*
 * @return the workLocation
 */
public String getWorkLocation() {
    return workLocation;
}

/*
 * @param workLocation
 *      the workLocation to set
 */
public void setWorkLocation(String workLocation) {
    this.workLocation = workLocation;
}

/*
 * @return the educational
 */
public String getEducational() {
    return educational;
}

/*
 * @param educational
 *      the educational to set
 */
public void setEducational(String educational) {
    this.educational = educational;
}
```

```
/*
 * @return the workYear
 */
public String getWorkYear() {
    return workYear;
}

/**
 * @param workYear
 *      the workYear to set
 */
public void setWorkYear(String workYear) {
    this.workYear = workYear;
}

/*
 * @return the phone
 */
public String getPhone() {
    return phone;
}

/**
 * @param phone
 *      the phone to set
 */
public void setPhone(String phone) {
    this.phone = phone;
}

/*
 * @return the nativeLocation
 */
public String getNativeLocation() {
    return nativeLocation;
}

/**
 * @param nativeLocation
 *      the nativeLocation to set
 */
public void setNativeLocation(String nativeLocation) {
    this.nativeLocation = nativeLocation;
}

/*
 * @return the school
 */
public String getSchool() {
    return school;
}

/**
 * @param school
 *      the school to set
 */
public void setSchool(String school) {
    this.school = school;
}
}
```

The following code snippet is used as an example. The function of the following code is to read stream data from the **data.txt** file, use netizen names as keywords to query from Redis clusters in normal mode, perform the JOIN operation, and print the results. For the complete code, see [com.huawei.bigdata.flink.examples.FlinkConfigurableScalaExample](#).

```
package com.huawei.bigdata.flink.examples

import java.util.HashSet
import java.util.Set
import java.util.concurrent.TimeUnit

import org.apache.flink.api.java.utils.ParameterTool
import org.apache.flink.streaming.api.TimeCharacteristic
import org.apache.flink.streaming.api.functions.AssignerWithPunctuatedWatermarks
import org.apache.flink.streaming.api.scala._
import org.apache.flink.streaming.api.scala.async.AsyncCollector
import org.apache.flink.streaming.api.watermark.Watermark
import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows
import org.apache.flink.streaming.api.windowing.time.Time
import redis.clients.jedis.HostAndPort
import redis.clients.jedis.JedisCluster

import scala.concurrent.{ExecutionContext, Future}

<**
 * Read stream data and join from configure table from redis.
 */
object FlinkConfigtableScalaExample {

    private val MAX_ATTEMPTS = 2

    private val TIMEOUT = 60000

    def main(args: Array[String]) {
        // print comment for command to use run flink
        System.out.println("use command as: \n" +
            "./bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableScalaExample" +
            " -m yarn-cluster -yt /opt/config -yn 3 -yjm 1024 -ymt 1024 " +
            "/opt/FlinkConfigtableScalaExample.jar --dataPath config/data.txt" +
            "*****\n" +
            "Especially you may write following content into config filePath, as in config/read.properties: \n" +
            "ReadFields=username,age,company,workLocation,educational,workYear,phone,nativeLocation,school\n" +
+
            "Redis_IP_Port=SZV1000064084:22400,SZV1000064082:22400,SZV1000064085:22400\n" +
            "*****")
        // set up the execution environment
        val env = StreamExecutionEnvironment.getExecutionEnvironment
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
        env.setParallelism(1)

        // get configure and read data and transform to OriginalRecord
        val dataPath = ParameterTool.fromArgs(args).get("dataPath", "config/data.txt")
        val originalStream = env.readTextFile(dataPath)
            .map(it => getRecord(it)).assignTimestampsAndWatermarks(new
        Record2TimestampExtractor).disableChaining()

        // read from redis and join to the whole user information
        val resultStream = AsyncDataStream.unorderedWait(
            originalStream,
            2,
            TimeUnit.MINUTES,
            5) {
            (input, collector: AsyncCollector[UserRecord]) =>
            Future {
                // Get configurations from config/read.properties. You must put this with commands:
                // ./bin/yarn-session.sh -t /opt/config -jm 1024 -tm 1024 or
                // ./bin/flink run -m yarn-cluster -yt /opt/config -yn 3 -yjm 1024 -ymt 1024 /opt/test.jar
                val configPath = "config/read.properties"
                val fields = ParameterTool.fromPropertiesFile(configPath).get("ReadFields")
                val hostPort = ParameterTool.fromPropertiesFile(configPath).get("Redis_IP_Port")
                val ssl = ParameterTool.fromPropertiesFile(configPath).getBoolean("Redis_ssl_on", false)
            }
        }
    }
}
```

```
// create jedisCluster client
val hosts: Set[HostAndPort] = new HashSet[HostAndPort]()
hostPort.split(",").foreach(it => hosts.add(new HostAndPort(it.split(":").apply(0),
Integer.parseInt(it.split(":").apply(1))))))

val poolConfig = new jedisPoolConfig
val socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory
val client = new jedisCluster(hosts, TIMEOUT, TIMEOUT, MAX_ATTEMPTS, "", "", poolConfig, ssl,
socketFactory, null, null, null)

if (client.getClusterNodes.size() <= 0) {
    System.out.println("JedisCluster init failed, getClusterNodes: " + client.getClusterNodes.size())
}
// Set key string, if your key is more than one column, build your key string with columns.
val key = input.name
if (!client.exists(key)) {
    System.out.println("test-----cannot find data to key: " + key)
    collector.collect(Seq(new UserRecord(
        input.name,
        0,
        "null",
        "null",
        "null",
        0,
        "null",
        "null",
        "null",
        input.sex,
        input.shoppingTime)))
} else {
    val values = client.hmget(key, fields.split(",:_"))
    System.out.println("test-----key: " + key + " get some fields: " + values.toString())
    collector.collect(Seq(new UserRecord(
        values.get(0),
        Integer.parseInt(values.get(1)),
        values.get(2),
        values.get(3),
        values.get(4),
        Integer.parseInt(values.get(5)),
        values.get(6),
        values.get(7),
        values.get(8),
        input.sex,
        input.shoppingTime)))
}
client.close()
} (ExecutionContext.global)

// data transform
resultStream.filter(_.sex == "female")
.keyBy("name")
.window(TumblingEventTimeWindows.of(Time.seconds(30)))
.reduce((e1, e2) => UserRecord(e1.name, e2.age, e2.company, e2.workLocation, e2.educational,
e2.workYear,
e2.phone, e2.nativeLocation, e2.school, e2.sex, e1.shoppingTime + e2.shoppingTime))
.filter(_shoppingTime > 120).print()

// execute program
env.execute("FlinkConfigtable scala")
}

// get enums of record
def getRecord(line: String): OriginalRecord = {
    val elems = line.split(",")
    assert(elems.length == 3)
    val name = elems(0)
    val sex = elems(1)
```

```
val time = elems(2).toInt
OriginalRecord(name, sexy, time)
}

// the scheme of record read from txt
case class OriginalRecord(name: String, sexy: String, shoppingTime: Int)

case class UserRecord(name: String, age: Int, company: String, workLocation: String, educational: String,
workYear: Int,
    phone: String, nativeLocation: String, school: String, sexy: String, shoppingTime: Int)

// class to set watermark and timestamp
private class Record2TimestampExtractor extends AssignerWithPunctuatedWatermarks[OriginalRecord] {

    // add tag in the data of datastream elements
    override def extractTimestamp(element: OriginalRecord, previousTimestamp: Long): Long = {
        System.currentTimeMillis()
    }

    // Give the watermark to trigger the window to start execution, and use the value to check if the window
    // elements are ready.
    def checkAndGetNextWatermark(lastElement: OriginalRecord,
                                extractedTimestamp: Long): Watermark = {
        new Watermark(extractedTimestamp - 1)
    }
}
```

### 2.4.3.6 Stream SQL Join Program

#### 2.4.3.6.1 Scenario

##### Scenario

Assume that a Flink service (service 1) receives a message every second. The message records the basic information about a user, including the name, gender, and age. Another Flink service (service 2) receives a message irregularly, and the message records the name and career information about the user.

To meet the requirements of some services, the Flink application is developed to achieve the following function: uses the username recorded in the message received by service 2 as the keyword to jointly query two pieces of service data.

##### Data Planning

- The data of service 1 is stored in the Kafka component. Service 1 sends data (requiring Kafka user rights) to and receives data from the Kafka component. For details about how to configure Kafka, see the data planning section of [Data Planning](#).
- Service 2 receives messages using the socket. You can run the **netcat** command to input the analog data source.
  - Run the **netcat -l -p <port>** command to start a simple text server.
  - After starting the application to connect to the port monitored by **netcat**, enter the data information to the netcat terminal.

## Development Approach

1. Start the Flink Kafka Producer application to send data to Kafka.
2. Start the Flink Kafka Consumer application to receive data from Kafka, construct **Table1**, and ensure that the Topic is the same as that of Producer.
3. Read data from the socket and construct **Table2**.
4. Use Flink SQL to query and print **Table1** and **Table2**.

### 2.4.3.6.2 Java Sample Code

#### Function

In the Flink application, this code invokes the flink-connector-kafka module's API to generate and consume data.

#### Sample Code

If the user needs to use FusionInsight Kafka interconnected with the security mode before the development, obtain the **kafka-client-\*jar** JAR file from the FusionInsight client directory.

The following lists Producer, Consumer, and the main logic code used by Flink Stream SQL Join.

For the complete code, see  
[com.huawei.bigdata.flink.examples.WriteIntoKafka](#) and  
[com.huawei.bigdata.flink.examples.SqlJoinWithSocket](#).

1. A piece of user information is generated in Kafka every second. The user information includes the name, age, and gender.

```
//Producer code
public class WriteIntoKafka {
    public static void main(String[] args) throws Exception {
        //Print the command reference for flink run.
        System.out.println("use command as: ");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21005");
        System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
            " /opt/test.jar --topic topic-test -bootstrap.servers 10.91.8.218:21007 --security.protocol
        SASL_PLAINTEXT --sasl.kerberos.service.name kafka");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");

        //Construct the execution environment.
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        //Set the concurrency.
        env.setParallelism(1);
        //Parse the running parameters.
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Construct a flow diagram and write the data generated from self-defined sources to Kafka.
        DataStream<String> messageStream = env.addSource(new SimpleStringGenerator());
        FlinkKafkaProducer<String> producer = new FlinkKafkaProducer<>(paraTool.get("topic"),
            new SimpleStringSchema(),
            paraTool.getProperties());
        producer.setWriteTimestampToKafka(true);
        messageStream.addSink(producer);
        //Invoke execute to trigger the execution.
        env.execute();
    }
}
```

```
//Customize the sources and generate a message every second.
public static class SimpleStringGenerator implements SourceFunction<String> {
    static final String[] NAME = {"Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James"};
    static final String[] SEX = {"MALE", "FEMALE"};
    static final int COUNT = NAME.length;
    boolean running = true;
    Random rand = new Random(47);
    @Override
    //Use rand to randomly generate a combination of the name, gender, and age.
    public void run(SourceContext<String> ctx) throws Exception {
        while (running) {
            int i = rand.nextInt(COUNT);
            int age = rand.nextInt(70);
            String sexy = SEX[rand.nextInt(2)];
            ctx.collect(NAME[i] + "," + age + "," + sexy);
            thread.sleep(1000);
        }
    }
    @Override
    public void cancel() {
        running = false;
    }
}
```

2. Generate **Table1** and **Table2**, use Join to jointly query **Table1** and **Table2**, and print the output result.

```
public class SqJoinWithSocket {
    public static void main(String[] args) throws Exception{
        final String hostname;
        final int port;
        System.out.println("use command as: ");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket" +
                " /opt/test.jar --topic topic-test -bootstrap.servers xxxx.xxx.xxx.xxx:21005 --hostname
xxxx.xxx.xxx.xxx --port xxx");
        System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket" +
                " /opt/test.jar --topic topic-test -bootstrap.servers xxxx.xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT --sasl.kerberos.service.name kafka"
                + "--hostname xxx.xxx.xxx.xxx --port xxx");
        System.out.println("*****");
        System.out.println("<topic> is the kafka topic name");
        System.out.println("<bootstrap.servers> is the ip:port list of brokers");
        System.out.println("*****");
        try {
            final ParameterTool params = ParameterTool.fromArgs(args);
            hostname = params.has("hostname") ? params.get("hostname") : "localhost";
            port = params.getInt("port");
        } catch (Exception e) {
            System.err.println("No port specified. Please run 'FlinkStreamSqJoinExample " +
                    "--hostname <hostname> --port <port>', where hostname (localhost by default) " +
                    "and port is the address of the text server");
            System.err.println("To start a simple text server, run 'netcat -l -p <port>' and " +
                    "type the input text into the command line");
            return;
        }
        EnvironmentSettings fsSettings =
EnvironmentSettings.newInstance().useOldPlanner().inStreamingMode().build();
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env, fsSettings);
        //Perform processing based on EventTime.
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.setParallelism(1);
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        //Use Stream1 to read data from Kafka.
        DataStream<Tuple3<String, String, String>> kafkaStream = env.addSource(new
FlinkKafkaConsumer<>(paraTool.get("topic"),
        new SimpleStringSchema(),
        paraTool.getProperties()).map(new MapFunction<String, Tuple3<String, String, String>>()
{
```

```

@Override
public Tuple3<String, String, String> map(String s) throws Exception {
    String[] word = s.split(",");
    return new Tuple3<>(word[0], word[1], word[2]);
}
});
//Register Stream1 as Table1.
tableEnv.registerDataStream("Table1", kafkaStream, "name, age, sexy, proctime.proctime");
//Use Stream2 to read data from the socket.
DataStream<Tuple2<String, String>> socketStream = env.socketTextStream(hostname, port,
"\n").
map(new MapFunction<String, Tuple2<String, String>>() {
    @Override
    public Tuple2<String, String> map(String s) throws Exception {
        String[] words = s.split("\s");
        if (words.length < 2) {
            return new Tuple2<>();
        }
        return new Tuple2<>(words[0], words[1]);
    }
});
//Register Stream2 as Table2.
tableEnv.registerDataStream("Table2", socketStream, "name, job, proctime.proctime");
//Run SQL Join to perform a combined query.
Table result = tableEnv.sqlQuery("SELECT t1.name, t1.age, t1.sex, t2.job, t2.proctime as shiptime
\n" +
    "FROM Table1 AS t1\n" +
    "JOIN Table2 AS t2\n" +
    "ON t1.name = t2.name\n" +
    "AND t1.proctime BETWEEN t2.proctime - INTERVAL '1' SECOND AND t2.proctime +
INTERVAL '1' SECOND");
//Convert the query result into the stream and print the output.
tableEnv.toAppendStream(result, Row.class).print();
env.execute();
}
}

```

#### 2.4.3.6.3 Scala Sample Code

##### Function

In a Flink application, call the API of the flink-connector-kafka module to produce and consume data.

##### Sample Code

If you need to interconnect with Kafka in security mode before application development, **kafka-clients-\*jar** of FusionInsight is required. You can obtain the JAR file from the Kafka client directory.

The following example provides the main code logic of Producer, Consumer, and Flink Stream SQL Join.

Complete code is provided in **com.huawei.bigdata.flink.examples.WriteIntoKafka** and **com.huawei.bigdata.flink.examples.SqlJoinWithSocket**.

1. Produce a piece of user information in Kafka every second. The user information includes the name, age, and gender.

```

//Producer

object WriteIntoKafka {
    def main(args: Array[String]): Unit = {

```

```

System.out.println("use command as: ")
System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka" +
" /opt/test.jar --topic topic-test -bootstrap.servers xxx.xxx.xxx:21005")
System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxx.xxx.xxx:21007 --security.protocol
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka")

System.out.println("*****")
System.out.println("<topic> is the kafka topic name")
System.out.println("<bootstrap.servers> is the ip:port list of brokers")
System.out.println("*****")
val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setParallelism(1)

val paraTool = ParameterTool.fromArgs(args)

val messageStream = env.addSource(new WriteIntoKafka.SimpleStringGenerator)
val producer = new FlinkKafkaProducer[String](paraTool.get("topic"), new SimpleStringSchema,
paraTool.getProperties)

producer.setWriteTimestampToKafka(true)

messageStream.addSink(producer)
env.execute
}

/**
 * String source class
 *
 */
object SimpleStringGenerator {
  private[examples] val NAME = Array("Carry", "Alen", "Mike", "Ian", "John", "Kobe", "James")
  private[examples] val SEX = Array("MALE", "FEMALE")
  private[examples] val COUNT = NAME.length
}

class SimpleStringGenerator extends SourceFunction[String] {
  private[examples] var running = true
  private[examples] val rand = new Random(47)

  @throws[Exception]
  override def run(ctx: SourceFunction.SourceContext[String]): Unit = {
    while (running) {
      val i = rand.nextInt(SimpleStringGenerator.COUNT)
      val age = rand.nextInt(70)
      val sexy = SimpleStringGenerator.SEX(rand.nextInt(2))
      ctx.collect(SimpleStringGenerator.NAME(i) + "," + age + "," + sexy)
      Thread.sleep(1000)
    }
  }

  override def cancel(): Unit = {
    running = false
  }
}

```

2. Generate Table1 and Table2, use **Join** to query both Table1 and Table2, and print the output.

```

object SqJoinWithSocket {
  def main(args: Array[String]): Unit = {
    var hostname: String = null
    var port = 0
    System.out.println("use command as: ")
    System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxxx.xxx.xxx:21005 --hostname xxx.xxx.xxx --port
xxx")
    System.out.println("flink run --class com.huawei.bigdata.flink.examples.SqJoinWithSocket /opt/
test.jar --topic" + " topic-test -bootstrap.servers xxxx.xxx.xxx:21007 --security.protocol

```

```
SASL_PLAINTEXT" + " --sasl.kerberos.service.name kafka--hostname xxx.xxx.xxx.xxx --port xxx")  
  
System.out.println("*****  
System.out.println("<topic> is the kafka topic name")  
System.out.println("<bootstrap.servers> is the ip:port list of brokers")  
System.out.println("*****")  
try {  
    val params = ParameterTool.fromArgs(args)  
    hostname = if (params.has("hostname")) params.get("hostname")  
    else "localhost"  
    port = params.getInt("port")  
} catch {  
    case e: Exception =>  
        System.err.println("No port specified. Please run 'FlinkStreamSqlJoinExample " + "--hostname  
<hostname> --port <port>', where hostname (localhost by default) " + "and port is the address of the  
text server")  
        System.err.println("To start a simple text server, run 'netcat -l -p <port>' and " + "type the input  
text into the command line")  
        return  
}  
  
val fsSettings = EnvironmentSettings.newInstance.inStreamingMode.build  
val env = StreamExecutionEnvironment.getExecutionEnvironment  
val tableEnv = StreamTableEnvironment.create(env, fsSettings)  
  
env.getConfig.setAutoWatermarkInterval(200)  
env.setParallelism(1)  
val paraTool = ParameterTool.fromArgs(args)  
  
val kafkaStream = env.addSource(new FlinkKafkaConsumer[String](paraTool.get("topic"), new  
SimpleStringSchema, paraTool.getProperties)).map(new MapFunction[String, Tuple3[String, String,  
String]]())  
{@throws[Exception]  
override def map(str: String): Tuple3[String, String, String] = {  
    val word = str.split(",")  
    new Tuple3[String, String, String](word(0), word(1), word(2))  
}  
})  
  
tableEnv.createTemporaryView("Table1", kafkaStream, $("name"), $($("age")), $($("sexy")), $\n  
("proctime").proctime)  
  
val socketStream = env.socketTextStream(hostname, port, "\n").map(new MapFunction[String,  
Tuple2[String, String]]())  
{@throws[Exception]  
override def map(str: String): Tuple2[String, String] = {  
    val words = str.split("\\s")  
    if (words.length < 2) return new Tuple2[String, String]  
    new Tuple2[String, String](words(0), words(1))  
}  
})  
  
tableEnv.createTemporaryView("Table2", socketStream, $("name"), $($("job")), $\n  
("proctime").proctime)  
  
val result = tableEnv.sqlQuery("SELECT t1.name, t1.age, t1.sex, t2.job, t2.proctime as shiptime\n" +  
"FROM Table1 AS t1\n" + "JOIN Table2 AS t2\n" + "ON t1.name = t2.name\n" + "AND t1.proctime  
BETWEEN t2.proctime - INTERVAL '1' SECOND AND t2.proctime + INTERVAL '1' SECOND")  
  
tableEnv.toAppendStream(result, classOf[Row]).print  
env.execute  
}  
}
```

### 2.4.3.7 Submitting a SQL Job Using Flink Jar

#### 2.4.3.7.1 Scenario Description

##### Description

If SQL statements of a job are frequently modified, submit Flink SQL statements in Flink Jar mode to reduce your workload.

##### Development Guideline

Use the current sample to submit and execute specified SQL statements. Use semicolons (;) to separate multiple statements.

#### 2.4.3.7.2 Java Sample Code

The core logic for submitting SQL statements is as follows. Currently, only **CREATE** and **INSERT** statements can be submitted. For details about the complete code, see `com.huawei.bigdata.flink.examples.FlinkSQLExecutor`.

```
public class FlinkSQLExecutor {  
    public static void main(String[] args) throws IOException {  
        System.out.println("----- begin init -----");  
        final String sqlPath = ParameterTool.fromArgs(args).get("sql", "config/redisSink.sql");  
        final StreamExecutionEnvironment streamEnv =  
            StreamExecutionEnvironment.getExecutionEnvironment();  
        EnvironmentSettings bsSettings = EnvironmentSettings.newInstance().inStreamingMode().build();  
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(streamEnv, bsSettings);  
        StatementSet statementSet = tableEnv.createStatementSet();  
        String sqlStr = FileUtils.readFileToString(FileUtils.getFile(sqlPath), "utf-8");  
        String[] sqlArr = sqlStr.split(";;");  
        for (String sql : sqlArr) {  
            sql = sql.trim();  
            if (sql.toLowerCase(Locale.ROOT).startsWith("create")) {  
                System.out.println("-----\nexecuteSql=\n" + sql);  
                tableEnv.executeSql(sql);  
            } else if (sql.toLowerCase(Locale.ROOT).startsWith("insert")) {  
                System.out.println("-----\ninsert=\n" + sql);  
                statementSet.addInsertSql(sql);  
            }  
        }  
        System.out.println("----- begin exec sql -----");  
        statementSet.execute();  
    }  
}
```



##### NOTE

Copy the dependency package required by the current sample, that is, the JAR package in the **lib** file after compilation, to the **lib** folder on the client.

The following uses Kafka in a normal cluster as an example to describe how to submit SQL statements:

```
create table kafka_sink  
(  
    uuid varchar(20),  
    name varchar(10),  
    age int,  
    ts timestamp(3),  
    p varchar(20)  
) with (  
    'connector' = 'kafka',  
    'topic' = 'input2',  
    'properties.bootstrap.servers' = 'IP address of the Kafka broker instance',  
    'properties.group.id' = 'testGroup2',
```

```
'scan.startup.mode' = 'latest-offset',
'format' = 'json'
);
create TABLE datagen_source
(
    uuid varchar(20),
    name varchar(10),
    age int,
    ts timestamp(3),
    p varchar(20)
) WITH (
    'connector' = 'datagen',
    'rows-per-second' = '1'
);
INSERT INTO kafka_sink
SELECT *
FROM datagen_source;
```

## 2.4.3.8 FlinkServer REST API JavaExample

### 2.4.3.8.1 Scenario Description

#### Scenario

Call the FlinkServer RESTful API to create tenants.

#### Data Preparation

- Prepare information required for creating a tenant, for example, **tenantId** is **92**, **tenantName** is **test92**, and **remark** is **test tenant remark1**.
- If you run this sample program in Windows, add the host names and IP addresses of all nodes where FlinkServer resides to the **C:\Windows\System32\drivers\etc\hosts** file.
- To call the RESTful API for starting a job, you need to prepare the FlinkServer job, for example, whose ID is **cb4fa1b385ec4b3bb35f0d1ae3990857**.

#### Development Guideline

1. Configure user authentication information.
2. Log in as a user.
3. Send requests.

### 2.4.3.8.2 Java sample code for invoking the FlinkServer REST API to create a tenant

#### Description

Call the FlinkServer RESTful API to create tenants.

#### Sample Code

```
public class TestCreateTenants {
    public static void main(String[] args) {
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        final String hostName = paraTool.get("hostName"); // Replace hostName in the hosts file with the
actual host name.

        String url = "https://"+hostName+":28943/flink/v1/tenants";
```

```
String jsonstr = "{" +  
    "\n\t\"tenantId\":\"92\"," +  
    "\n\t\"tenantName\":\"test92\"," +  
    "\n\t\"remark\":\"test tenant remark\"," +  
    "\n\t\"updateUser\":\"test_updateUser\"," +  
    "\n\t\"createUser\":\"test_createUser\\"" +  
    "\n}";  
  
try {  
    System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", false));  
} catch (Exception e) {  
    System.out.println(e);  
}  
}  
}
```

#### 2.4.3.8.3 Calling a FlinkServer REST API to Start a Job

### Function

This topic describes how to start a FlinkServer job by calling a FlinkServer REST API.

### Sample Code

The following example uses the job whose ID is **cb4fa1b385ec4b3bb35f0d1ae3990857** to describe how to call the API for starting a job on FlinkServer. The complete code is as follows:

```
public class TestJobsAction {  
    public static void main(String[] args) {  
        ParameterTool paraTool = ParameterTool.fromArgs(args);  
        final String hostName = paraTool.get("hostName"); // Host name  
        final String keytab = paraTool.get("keytab"); // Path for storing user.keytab  
        final String krb5 = paraTool.get("krb5"); // Path for storing krb5.conf  
        final String principal = paraTool.get("principal"); // Authentication user  
  
        System.setProperty("java.security.krb5.conf", krb5);  
        String url = "https://" + hostName + ":28943/flink/v1/1/jobs/action";  
  
        // Modify the corresponding jsonstr.  
        String jsonstr = "{" +  
            "\n\t\"jobId\":\"cb4fa1b385ec4b3bb35f0d1ae3990857\"," +  
            "\n\t\"action\":\"start\\"" +  
            "\n}";  
        try {  
            LoginClient.getInstance().setConfigure(url, principal, keytab, "");  
            LoginClient.getInstance().login();  
            System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", false));  
        } catch (Exception e) {  
            System.out.println(e);  
        }  
    }  
}
```

#### 2.4.3.8.4 Accessing Flinkserver RESTful API as a Proxy User

### Function

Call the FlinkServer RESTful API as a proxy user. Use a proxy to access the API as a FlinkServer administrator to obtain common user permissions.

## Sample Code

Assume that the tenant user is **test92**, the tenant ID is **92**, and the user name is **flinkserveradmin** with FlinkServer administrator permissions. The following code is a complete example.

```
public class TestCreateTenants {  
    public static void main(String[] args) {  
        ParameterTool paraTool = ParameterTool.fromArgs(args);  
        final String hostName = paraTool.get("hostName"); // Replace hostName in the hosts file with the  
actual host name.  
        final String keytab = paraTool.get("keytab"); // Path for storing user.keytab  
        final String krb5 = paraTool.get("krb5"); // Path for storing krb5.conf  
        final String principal = paraTool.get("principal"); // Authentication user  
  
        System.setProperty("java.security.krb5.conf", krb5);  
        String url = "https://" + hostName + ".28943/flink/v1/tenants";  
        String jsonstr = "{" +  
            "\n\t\"tenantId\":\"92\","  
            "\n\t\"tenantName\":\"test92\","  
            "\n\t\"remark\":\"test tenant remark1\","  
            "\n\t\"updateUser\":\"test_updateUser1\","  
            "\n\t\"createUser\":\"test_createUser1\""  
            "\n"}";  
  
        try {  
            LoginClient.getInstance().setConfigure(url, principal, keytab, "");  
            LoginClient.getInstance().login(); // Log in as the FlinkServer administrator.  
  
            String proxyUrl = "https://" + hostName + ".28943/flink/v1/proxyUserLogin"; // Call the proxy user API  
to obtain the common user token.  
            String result = HttpClientUtil.doPost(proxyUrl, "{\n" +  
                "\t\"realUser\": \"flinkserveradmin\"\n" +  
                "}", "utf-8", true);  
            Gson gson = new Gson();  
            JsonObject jsonObject = gson.fromJson(result, JsonObject.class);  
            String token = jsonObject.get("result").toString();  
            token = "hadoop_auth=" + token;  
  
            System.out.println(HttpClientUtil.doPost(url, jsonstr, "utf-8", true, token));  
        } catch (Exception e) {  
            System.out.println(e);  
        }  
    }  
}
```

### 2.4.3.9 Flink Reading Data from and Writing Data to HBase

#### 2.4.3.9.1 Scenario Description

#### Typical Scenario Description

Use Flink API jobs to read data from and write data to HBase.

#### Data Preparation

Prepare the HBase configuration file and download the cluster configuration on FusionInsight Manager to obtain the **hbase-site.xml** file.

## Development Guideline

1. Writes data to HBase:
  - a. Specify the parent directory of the **hbase-site.xml** file. Flink Sink can obtain the HBase connection.
  - b. Use the connection to determine whether a table exists. If it does not, create one.
  - c. Convert received data into Put objects and writes the Put objects to HBase.
2. Reads data from HBase:
  - a. Specify the parent directory of the **hbase-site.xml** file. Flink Source can obtain the HBase connection.
  - b. Use the connection to determine whether a table exists. If it does not, the job fails. In this case, you need to create a table in HBase shell or an upstream job.
  - c. Read data from HBase, converts result data into Row objects, and sends the Row objects to downstream operators.

### 2.4.3.9.2 Java Sample Code

#### Description

Call Flink APIs to read data from and write data to HBase.

#### Sample Code

The following example shows the main logic code of WriteHBase and ReadHBase.

For details about the complete code, see  
[com.huawei.bigdata.flink.examples.WriteHBase](#) and  
[com.huawei.bigdata.flink.examples.ReadHBase](#).

- Main logic code of WriteHBase

```
public static void main(String[] args) throws Exception {
    System.out.println("use command as: ");
    System.out.println(
        "./bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase"
        + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");
    System.out.println(
        "*****");
    System.out.println("<tableName> hbase tableName");
    System.out.println("<confDir> hbase conf dir");
    System.out.println(
        "*****");
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.setParallelism(1);
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    DataStream<Row> messageStream = env.addSource(new SimpleStringGenerator());
    messageStream.addSink(
        new HBaseWriteSink(paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
    env.execute("WriteHBase");
}

private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
    LOG.info("Create HBase configuration.");
    org.apache.hadoop.conf.Configuration hbaseConf =
        HBaseConfigurationUtil.getHBaseConfiguration();
```

```
if (confDir != null) {
    File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
    if (hbaseSite.exists()) {
        LOG.info("Add hbase-site.xml");
        hbaseConf.addResource(new Path(hbaseSite.getPath()));
    }
    File coreSite = new File(confDir + File.separator + "core-site.xml");
    if (coreSite.exists()) {
        LOG.info("Add core-site.xml");
        hbaseConf.addResource(new Path(coreSite.getPath()));
    }
    File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
    if (hdfsSite.exists()) {
        LOG.info("Add hdfs-site.xml");
        hbaseConf.addResource(new Path(hdfsSite.getPath()));
    }
}
LOG.info("HBase configuration created successfully.");
return hbaseConf;
}

private static class HBaseWriteSink extends RichSinkFunction<Row> {
    private Connection conn;
    private BufferedMutator bufferedMutator;
    private String tableName;
    private final byte[] serializedConfig;
    private Admin admin;
    private org.apache.hadoop.conf.Configuration hbaseConf;
    private long flushTimeIntervalMillis = 5000; //5s
    private long preFlushTime;

    public HBaseWriteSink(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
        this.tableName = sourceTable;
        this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
    }

    private void deserializeConfiguration() {
        LOG.info("Deserialize HBase configuration.");
        hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
            serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
        LOG.info("Deserialization successfully.");
    }

    private void createTable() throws IOException {
        LOG.info("Create HBase Table.");
        if (admin.tableExists(TableName.valueOf(tableName))) {
            LOG.info("Table already exists.");
            return;
        }
        // Specify the table descriptor.
        TableDescriptorBuilder htd =
TableDescriptorBuilder.newBuilder(TableName.valueOf(tableName));
        // Set the column family name to f1.
        ColumnFamilyDescriptorBuilder hcd =
ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("f1"));
        // Set data encoding methods. HBase provides DIFF,FAST_DIFF,PREFIX
        hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
        // Set compression methods, HBase provides two default compression
        // methods:GZ and SNAPPY
        hcd.setCompressionType(Compression.Algorithm.SNAPPY);
        htd.setColumnFamily(hcd.build());
        try {
            admin.createTable(htd.build());
        } catch (IOException e) {
            if (!(e instanceof TableExistsException)
                || !admin.tableExists(TableName.valueOf(tableName))) {
                throw e;
            }
            LOG.info("Table already exists, ignore.");
        }
    }
}
```

```
        }
        LOG.info("Table created successfully.");
    }

@Override
public void open(Configuration parameters) throws Exception {
    LOG.info("Write sink open");
    super.open(parameters);
    deserializeConfiguration();
    conn = ConnectionFactory.createConnection(hbaseConf);
    admin = conn.getAdmin();
    createTable();
    bufferedMutator = conn.getBufferedMutator(TableName.valueOf(tableName));
    preFlushTime = System.currentTimeMillis();
}

@Override
public void close() throws Exception {
    LOG.info("Close HBase Connection.");
    try {
        if (admin != null) {
            admin.close();
            admin = null;
        }
        if (bufferedMutator != null) {
            bufferedMutator.close();
            bufferedMutator = null;
        }
        if (conn != null) {
            conn.close();
            conn = null;
        }
    } catch (IOException e) {
        LOG.error("Close HBase Exception:", e);
        throw new RuntimeException(e);
    }
    LOG.info("Close successfully.");
}

@Override
public void invoke(Row value, Context context) throws Exception {
    LOG.info("Write data to HBase.");
    Put put = new Put(Bytes.toBytes(value.getField(0).toString()));
    put.addColumn(Bytes.toBytes("f1"), Bytes.toBytes("q1"),
    (Bytes.toBytes(value.getField(1).toString())));
    bufferedMutator.mutate(put);

    if (preFlushTime + flushTimeIntervalMillis >= System.currentTimeMillis()) {
        LOG.info("Flush data to HBase.");
        bufferedMutator.flush();
        preFlushTime = System.currentTimeMillis();
        LOG.info("Flush successfully.");
    } else {
        LOG.info("Skip Flush.");
    }

    LOG.info("Write successfully.");
}
}

public static class SimpleStringGenerator implements SourceFunction<Row> {
    private static final long serialVersionUID = 2174904787118597072L;
    boolean running = true;
    long i = 0;
    Random random = new Random();

    @Override
    public void run(SourceContext<Row> ctx) throws Exception {
        while (running) {

```

```
        Row row = new Row(2);
        row.setField(0, "rk" + random.nextLong());
        row.setField(1, "v" + random.nextLong());
        ctx.collect(row);
        Thread.sleep(1000);
    }
}

@Override
public void cancel() {
    running = false;
}
}
```

- Main logic code of ReadHBase

```
public static void main(String[] args) throws Exception {
    System.out.println("use command as:");
    System.out.println("./bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase"
        + " /opt/test.jar --tableName t1 --confDir /tmp/hbaseConf");

    System.out.println(
        "*****");
    System.out.println("<tableName> hbase tableName");
    System.out.println("<confDir> hbase conf dir");
    System.out.println(
        "*****");
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.setParallelism(1);
    ParameterTool paraTool = ParameterTool.fromArgs(args);
    DataStream<Row> messageStream =
        env.addSource(
            new HBaseReaderSource(
                paraTool.get("tableName"), createConf(paraTool.get("confDir"))));
    messageStream
        .rebalance()
        .map(
            new MapFunction<Row, String>() {
                @Override
                public String map(Row s) throws Exception {
                    return "Flink says " + s + System.getProperty("line.separator");
                }
            })
        .print();
    env.execute("ReadHBase");
}

private static org.apache.hadoop.conf.Configuration createConf(String confDir) {
    LOG.info("Create HBase configuration.");
    org.apache.hadoop.conf.Configuration hbaseConf =
    HBaseConfigurationUtil.getHBaseConfiguration();
    if (confDir != null) {
        File hbaseSite = new File(confDir + File.separator + "hbase-site.xml");
        if (hbaseSite.exists()) {
            LOG.info("Add hbase-site.xml");
            hbaseConf.addResource(new Path(hbaseSite.getPath()));
        }
        File coreSite = new File(confDir + File.separator + "core-site.xml");
        if (coreSite.exists()) {
            LOG.info("Add core-site.xml");
            hbaseConf.addResource(new Path(coreSite.getPath()));
        }
        File hdfsSite = new File(confDir + File.separator + "hdfs-site.xml");
        if (hdfsSite.exists()) {
            LOG.info("Add hdfs-site.xml");
            hbaseConf.addResource(new Path(hdfsSite.getPath()));
        }
    }
    LOG.info("HBase configuration created successfully.");
    return hbaseConf;
}
```

```
}

private static class HBaseReaderSource extends RichSourceFunction<Row> {

    private Connection conn;
    private Table table;
    private Scan scan;
    private String tableName;
    private final byte[] serializedConfig;
    private Admin admin;
    private org.apache.hadoop.conf.Configuration hbaseConf;

    public HBaseReaderSource(String sourceTable, org.apache.hadoop.conf.Configuration conf) {
        this.tableName = sourceTable;
        this.serializedConfig = HBaseConfigurationUtil.serializeConfiguration(conf);
    }

    @Override
    public void open(Configuration parameters) throws Exception {
        LOG.info("Read source open");
        super.open(parameters);
        deserializeConfiguration();
        conn = ConnectionFactory.createConnection(hbaseConf);
        admin = conn.getAdmin();
        if (!admin.tableExists(TableDescriptor.create(tableName).getTableName())) {
            throw new IOException("table does not exist.");
        }
        table = conn.getTable(TableDescriptor.create(tableName).getTableName());
        scan = new Scan();
    }

    private void deserializeConfiguration() {
        LOG.info("Deserialize HBase configuration.");
        hbaseConf = HBaseConfigurationUtil.deserializeConfiguration(
            serializedConfig, HBaseConfigurationUtil.getHBaseConfiguration());
        LOG.info("Deserialization successfully.");
    }

    @Override
    public void run(SourceContext<Row> sourceContext) throws Exception {
        LOG.info("Read source run");
        try (ResultScanner scanner = table.getScanner(scan)) {
            Iterator<Result> iterator = scanner.iterator();
            while (iterator.hasNext()) {
                Result result = iterator.next();
                String rowKey = Bytes.toString(result.getRow());
                byte[] value = result.getValue(Bytes.toBytes("f1"), Bytes.toBytes("q1"));
                Row row = new Row(2);
                row.setField(0, rowKey);
                row.setField(1, Bytes.toString(value));
                sourceContext.collect(row);
                LOG.info("Send data successfully.");
            }
        }
        LOG.info("Read successfully.");
    }

    @Override
    public void close() throws Exception {
        closeHBase();
    }

    private void closeHBase() {
        LOG.info("Close HBase Connection.");
        try {
            if (admin != null) {
                admin.close();
                admin = null;
            }
        }
    }
}
```

```
        }
        if (table != null) {
            table.close();
            table = null;
        }
        if (conn != null) {
            conn.close();
            conn = null;
        }
    } catch (IOException e) {
    LOG.error("Close HBase Exception:", e);
    throw new RuntimeException(e);
}
LOG.info("Close successfully.");
}

@Override
public void cancel() {
    closeHBase();
}
}
```

### 2.4.3.10 Flink Reading Data from and Writing Data to Hudi

#### 2.4.3.10.1 Scenario Description

#### Typical Scenario Description

In this example, the job generates one data record per second, writes the data to the Hudi table, and reads and prints the data in the Hudi table.

#### Development Guideline

1. Write data to Hudi:
  - a. Generate data through a random data generation class.
  - b. Convert the generated data into **DataStream<RowData>**.
  - c. Write data to the Hudi table.
2. Read data from Hudi:
  - a. Read data from the Hudi table.
  - b. Combine the read data into the JSON format and print the data.

#### 2.4.3.10.2 Java Sample Code

#### Description

Call Flink APIs to read data from and write data to Hudi.

#### Sample Code

The following example shows the main logic code of **WriteIntoHudi** and **ReadFromHudi**.

For details about the complete code, see  
**com.huawei.bigdata.flink.examples.WriteIntoHudi** and  
**com.huawei.bigdata.flink.examples.ReadFromHudi**.

- Main logic code of WriteIntoHudi

```

public class WriteIntoHudi {
    public static void main(String[] args) throws Exception {
        System.out.println("use command as: ");
        System.out.println(
            "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteIntoHudi"
            + " /opt/test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/"
            + "flinkHudi/hudiTable");
        System.out.println(
            "*****");
        System.out.println("<hudiTableName> is the hudi table name. (Default value is hudiSinkTable)");
        System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://"
            + "hacluster/tmp/flinkHudi/hudiTable)");
        System.out.println(
            "*****");
        System.out.println(
            "*****");

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(1);
        env.getCheckpointConfig().setCheckpointInterval(10000);
        ParameterTool paraTool = ParameterTool.fromArgs(args);
        DataStream<RowData> stringDataStreamSource = env.addSource(new SimpleStringGenerator()
            .map(new MapFunction<Tuple5<String, String, Integer, String, String>, RowData>() {
                @Override
                public RowData map(Tuple5<String, String, Integer, String, String> tuple5) throws
                    Exception {
                    GenericRowData rowData = new GenericRowData(5);
                    rowData.setField(0, StringData.fromString(tuple5.f0));
                    rowData.setField(1, StringData.fromString(tuple5.f1));
                    rowData.setField(2, tuple5.f2);
                    rowData.setField(3, TimestampData.fromTimestamp(Timestamp.valueOf(tuple5.f3)));
                    rowData.setField(4, StringData.fromString(tuple5.f4));
                    return rowData;
                }
            }));
        String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
        String targetTable = paraTool.get("hudiTableName", "hudiSinkTable");
        Map<String, String> options = new HashMap<>();
        options.put(FlinkOptions.PATH.key(), basePath);
        options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
        options.put(FlinkOptions.PRECOMBINE_FIELD.key(), "ts");
        options.put(FlinkOptions.INDEX_BOOTSTRAP_ENABLED.key(), "true");
        HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
            .column("uuid VARCHAR(20)")
            .column("name VARCHAR(10)")
            .column("age INT")
            .column("ts TIMESTAMP(3)")
            .column("p VARCHAR(20)")
            .pk("uuid")
            .partition("p")
            .options(options);
        builder.sink(stringDataStreamSource, false); // The second parameter indicating whether the
        input data stream is bounded
        env.execute("Hudi_Sink");
    }
    public static class SimpleStringGenerator implements SourceFunction<Tuple5<String, String,
        Integer, String, String>> {
        private static final long serialVersionUID = 2174904787118597072L;
        boolean running = true;
        Integer i = 0;

        @Override
        public void run(SourceContext<Tuple5<String, String, Integer, String, String>> ctx) throws
            Exception {
            while (running) {
                i++;
                String uuid = "uuid" + i;
                String name = "name" + i;
                Integer age = new Integer(i);
                String ts = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
            }
        }
    }
}

```

```
HH:mm:ss"));
        String p = "par" + i % 5;
        Tuple5<String, String, Integer, String, String> tuple5 = Tuple5.of(uuid, name, age, ts, p);
        ctx.collect(tuple5);
        Thread.sleep(1000);
    }
}
@Override
public void cancel() {
    running = false;
}
}
```

- Main logic code of ReadFromHudi

```

public class ReadFromHudi {
    public static void main(String[] args) throws Exception {
        System.out.println("use command as: ");
        System.out.println(
            "./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadFromHudi"
            + " /opt/test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/"
            flinkHudi/hudiTable"
            + " --read.start-commit 20221206111532"
        );
        System.out.println(
            "*****");
        System.out.println("<hudiTableName> is the hoodie table name. (Default value is"
        hudiSourceTable)");
        System.out.println("<hudiPath> Base path for the target hoodie table. (Default value is hdfs://"
        hacluster/tmp/flinkHudi/hudiTable)");
        System.out.println("<read.start-commit> Start commit instant for reading, the commit time"
        format should be 'yyyyMMddHHmmss'. (Default value is earliest)");
        System.out.println(
            "*****");
    }

    ParameterTool paraTool = ParameterTool.fromArgs(args);
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    String basePath = paraTool.get("hudiPath", "hdfs://hacluster/tmp/flinkHudi/hudiTable");
    String targetTable = paraTool.get("hudiTableName", "hudiSourceTable");
    String startCommit = paraTool.get(FlinkOptions.READ_START_COMMIT.key(),
FlinkOptions.START_COMMIT_EARLIEST);
    Map<String, String> options = new HashMap();
    options.put(FlinkOptions.PATH.key(), basePath);
    options.put(FlinkOptions.TABLE_TYPE.key(), HoodieTableType.MERGE_ON_READ.name());
    options.put(FlinkOptions.READ_AS_STREAMING.key(), "true"); // This option enables streaming
read.
    options.put(FlinkOptions.READ_START_COMMIT.key(), startCommit); // specifies the start
commit instant time
    HoodiePipeline.Builder builder = HoodiePipeline.builder(targetTable)
        .column("uuid VARCHAR(20)")
        .column("name VARCHAR(10)")
        .column("age INT")
        .column("ts TIMESTAMP(3)")
        .column("p VARCHAR(20)")
        .pk("uuid")
        .partition("p")
        .options(options);

    DataStream<RowData> rowDataDataStream = builder.source(env);
    rowDataDataStream.map(new MapFunction<RowData, String>() {
        @Override
        public String map(RowData rowData) throws Exception {
            StringBuilder sb = new StringBuilder();
            sb.append("{");
            sb.append("\"uuid\":\"").append(rowData.getString(0)).append("\",\"");
            sb.append("\"name\":\"").append(rowData.getString(1)).append("\",\"");
            sb.append("\"age\":").append(rowData.getInt(2)).append(",");
            sb.append("\"ts\":\"").append(rowData.getTimestamp(3, 0)).append("\",\"");
            sb.append("\"p\":\"").append(rowData.getString(4)).append("\",\"");
            sb.append("}");
        }
    });
}

```

```
        return sb.toString();
    }
}).print();
env.execute("Hudi_Source");
}
}
```

### 2.4.3.11 Python Development Examples

#### 2.4.3.11.1 Submitting a Regular Job Using Python

##### Description

Assume that you need to submit a Flink task to an MRS cluster. The main language used by the service platform is Python. The following content uses an example Python program to read and write Kafka jobs.

##### Python Sample Code

##### Function

Submit Flink Kafka read and write jobs to Yarn through Python APIs.

##### Sample Code

The main logic code in **pyflink-kafka.py** is provided. Before submitting the code, ensure that **file\_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see **pyflink-kafka.py** in **flink-examples/pyflink-example/pyflink-kafka**.

```
import os
import logging
import sys
from pyflink.common import JsonRowDeserializationSchema, JsonRowSerializationSchema
from pyflink.common.typeinfo import Types
from pyflink.datastream.connectors import FlinkKafkaProducer, FlinkKafkaConsumer
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import TableEnvironment, EnvironmentSettings
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
    # Change the SQL path before job submission.
    # file_path = "/opt/client/Flink/flink/insertData2kafka.sql"
    # file_path = os.getcwd() + "/../../../../../yarnship/insertData2kafka.sql"
    # file_path = "/opt/client/Flink/flink/conf/ssl/insertData2kafka.sql"
    file_path = "insertData2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("-----create-----")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
```

```

print("-----insert-----")
print(sqlStr)
statement_set.add_insert_sql(sqlStr)
statement_set.execute()
def read_write_kafka():
    # find kafka connector jars
    env = StreamExecutionEnvironment.get_execution_environment()
    env.set_parallelism(1)
    specific_jars = "file:///opt/client/Flink/flink/lib/flink-connector-kafka-xxx.jar"
    # specific_jars = "file://" + os.getcwd() + "/../../yarnship/flink-connector-kafka-xxx.jar"
    # specific_jars = "file:///opt/client/Flink/flink/conf/ssl/flink-connector-kafka-xxx.jar"
    # the sql connector for kafka is used here as it's a fat jar and could avoid dependency issues
    env.add_jars(specific_jars)
    kafka_properties = {'bootstrap.servers': '192.168.20.162:21005', 'group.id': 'test_group'}
    deserialization_schema = JsonRowDeserializationSchema.builder() \
        .type_info(type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_consumer = FlinkKafkaConsumer(
        topics='test_source_topic',
        deserialization_schema=deserialization_schema,
        properties=kafka_properties)
    print("-----read -----")
    ds = env.add_source(kafka_consumer)
    serialization_schema = JsonRowSerializationSchema.builder().with_type_info(
        type_info=Types.ROW([Types.INT(), Types.STRING()])).build()
    kafka_producer = FlinkKafkaProducer(
        topic='test_sink_topic',
        serialization_schema=serialization_schema,
        producer_config=kafka_properties)
    print("-----write-----")
    ds.add_sink(kafka_producer)
    env.execute("pyflink kafka test")
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format"%(message)s")
    print("-----insert data to kafka-----")
    exec_sql()
    print("-----read_write_kafka-----")
    read_write_kafka()

```

**Table 2-20** Parameters for submitting a regular job using Python

| Parameter         | Description  | Example  |
|-------------------|--|--|
| bootstrap.servers | Service IP address and port number of the Broker instance of Kafka   | 192.168.12.25:21005  |
| specific_jars     | <p>Package path: <i>Client installation directory/Flink/flink/lib/flink-connector-kafka-*.jar</i>. You are advised to use a full path.</p> <p><b>NOTE</b><br/>If a job needs to be submitted as yarn-application, replace the following path. Replace the JAR package version with the actual one.</p> <pre>specific_jars="file://" + os.getcwd() + "/../../yarnship/flink-connector-kafka-xxx-h0.cbu.mrs.xxx.jar"</pre> | specific_jars = file:///Client installation directory/Flink/flink/lib/flink-connector-kafka-xxx-h0.cbu.mrs.xxx.jar |

| Parameter | Description  | Example   |
|-----------|--|---|
| file_path | <p>Path of the <b>insertData2kafka.sql</b> file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client.</p> <p><b>NOTE</b><br/>If a job needs to be submitted as yarn-application, replace the following path:<br/><code>file_path = os.getcwd() + "/../../../../../yarnship/insertData2kafka.sql"</code></p> | file_path = /Client installation directory/Flink/flink/insertData2kafka.sql |

The following is an example SQL statement:

```
create table kafka_sink_table (
    age int,
    name varchar(10)
) with (
    'connector' = 'kafka',
    'topic' = 'test_source_topic', --Name of the topic written to Kafka. Ensure that the topic name is the same as that in the Python file.
    'properties.bootstrap.servers' = 'IP address of the Kafka broker instance.Kafka port number',
    'properties.group.id' = 'test_group',
    'format' = 'json'
);
create TABLE datagen_source_table (
    age int,
    name varchar(10)
) WITH (
    'connector' = 'datagen',
    'rows-per-second' = '1'
);
INSERT INTO
    kafka_sink_table
SELECT
    *
FROM
    datagen_source_table;
```

## Running the Program

**Step 1** Obtain **pyflink-kafka.py** and **insertData2kafka.sql** from the sample project **flink-examples/pyflink-example/pyflink-kafka**.

**Step 2** Package the prepared Python virtual environment by referring to [Preparing Development and Operating Environment](#) and obtain the **venv.zip** file.

```
zip -q -r venv.zip venv/
```

**Step 3** Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-kafka.py**, and **insertData2kafka.sql** files obtained in [Step 1](#) and [Step 2](#) to the client environment.

- Per-job: Upload the preceding files to *Client installation directory/Flink/flink*.
- yarn-application: Upload the preceding files and the **flink-connector-kafka-Actual version number.jar** package to *Client installation directory/Flink/flink/yarnship*.

**Step 4** Change the `specific_jars` path in `pyflink-kafka.py`.

- per-job: Change the path to the actual path of the SQL file, for example, `file:///Client installation directory/Flink/flink/lib/flink-connector-kafka-Actual version number.jar`.
  - yarn-application: Change to `file://" + os.getcwd() + "/../../..../yarnship/flink-connector-kafka-Actual version number.jar`.

## **Step 5 Change file\_path in pyflink-kafka.py.**

- per-job: Change the path to the actual path of the SQL file. For example:  
*Client installation directory/Flink/flink/insertData2kafka.sql*
  - yarn-application: Change the path to **os.getcwd () + "/../../../../yarnship/insertData2kafka.sql"**

**Step 6** Run the following command to specify the running environment:

```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

**Step 7** Run the following command to run the program:

- **Per-job:**  
./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py\_kafka -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-kafka.py

## Execution result:

- **yarn-application**  
./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py\_kafka -Dyarn.ship-files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-kafka

## Execution result:

-----End

### 2.4.3.11.2 Submitting a SQL Job Using Python

## Description

Assume that you need to submit Flink tasks to the MRS cluster, the main language used by the service platform is Python, and core service processing requires SQL. The following content provides an example to describe how to submit a SQL job using Python.

## Python Sample Code

### Function

Submit a Flink SQL job to Yarn through Python APIs.

### Sample Code

The main logic code in **pyflink-sql.py** is provided. Before submitting the code, ensure that **file\_path** is the path where the SQL statement to be executed. You are advised to use a full path.

For details about the complete code, see **pyflink-sql.py** in **flink-examples/pyflink-example/pyflink-sql**.

```
import logging
import sys
import os
from pyflink.table import (EnvironmentSettings, TableEnvironment)
def read_sql(file_path):
    if not os.path.isfile(file_path):
        raise TypeError(file_path + " does not exist")
    all_the_text = open(file_path).read()
    return all_the_text
def exec_sql():
    # Change the SQL path before job submission.
    file_path = "datagen2kafka.sql"
    sql = read_sql(file_path)
    t_env = TableEnvironment.create(EnvironmentSettings.in_streaming_mode())
    statement_set = t_env.create_statement_set()
    sqlArr = sql.split(";\n")
    for sqlStr in sqlArr:
        sqlStr = sqlStr.strip()
        if sqlStr.lower().startswith("create"):
            print("-----create-----")
            print(sqlStr)
            t_env.execute_sql(sqlStr)
        if sqlStr.lower().startswith("insert"):
            print("-----insert-----")
            print(sqlStr)
            statement_set.add_insert_sql(sqlStr)
    statement_set.execute()
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format"%(message)s")
    exec_sql()
```

**Table 2-21** Parameters for submitting a SQL job using Python

| Parameter | Description   | Example   |
|-----------|---|---|
| file_path | <p>Path of the <b>datagen2kafka.sql</b> file. You are advised to use a full path. Obtain the package from the secondary development sample code and upload it to the specified directory on the client.</p> <p><b>NOTE</b><br/>If a job needs to be submitted as <b>yarn-application</b>, replace the following path:<br/><code>file_path = os.getcwd() + "/../../../../../yarnship/datagen2kafka.sql"</code></p> | <code>file_path = /Client<br/>installation directory/<br/>Flink/flink/<br/>datagen2kafka.sql</code> |

The following is an example SQL statement:

```
create table kafka_sink (
    uuid varchar(20),
    name varchar(10),
    age int,
    ts timestamp(3),
    p varchar(20)
) with (
    'connector' = 'kafka',
    'topic' = 'input2',
    'properties.bootstrap.servers' = 'IP address of the Kafka broker instance.Kafka port number',
    'properties.group.id' = 'testGroup2',
    'scan.startup.mode' = 'latest-offset',
    'format' = 'json'
);
create TABLE datagen_source (
    uuid varchar(20),
    name varchar(10),
    age int,
    ts timestamp(3),
    p varchar(20)
) WITH (
    'connector' = 'datagen',
    'rows-per-second' = '1'
);
INSERT INTO
    kafka_sink
SELECT
    *
FROM
    datagen_source;
```

## Running the Program

**Step 1** Obtain **pyflink-sql.py** and **datagen2kafka.sql** from the sample project **flink-examples/pyflink-example/pyflink-sql**.

**Step 2** Package the prepared Python virtual environment by referring to [Preparing Development and Operating Environment](#) and obtain the **venv.zip** file.

```
zip -q -r venv.zip venv/
```

**Step 3** Log in to the active management node as the **root** user and upload **venv.zip**, **pyflink-sql.py**, and **datagen2kafka.sql** files obtained in **Step 1** and **Step 2** to the client environment.

- Per-job: Upload the preceding files to *Client installation directory/Flink/flink*.
- yarn-application: Upload the preceding files to *Client installation directory/Flink/flink/yarnship*.
- yarn-session: Upload the preceding files to *Client installation directory/Flink/flink/conf/ssl*.

**Step 4** Change **file\_path** in **pyflink-sql.py**.

- per-job: Change the path to the actual path of the SQL file. For example: *Client installation directory/Flink/flink/datagen2kafka.sql*
- yarn-application: Change the path to **os.getcwd() + "/../../../../../yarnship/datagen2kafka.sql"**
- yarn-session: Change the path to the actual path of the SQL file. For example: *Client installation directory/Flink/flink/conf/ssl//datagen2kafka.sql*

**Step 5** Run the following command to specify the running environment:

```
export PYFLINK_CLIENT_EXECUTABLE=venv.zip/venv/bin/python3
```

**Step 6** Run the following command to run the program:

- Per-job:

```
./bin/flink run --detached -t yarn-per-job -Dyarn.application.name=py_sql -pyarch venv.zip -pyexec venv.zip/venv/bin/python3 -py pyflink-sql.py
```

## Execution result:

```
2023-07-19 [09:48:57.500] [INFO] [dash] | Login successful for user 'admin' using Yaml file 'user-keycloak-admin.yaml' | org.apache.hadoop.security.KeycloakUserLoginInformation.LoginInformation@5e05a1f0 | java[129]
2023-07-19 [09:48:57.500] [INFO] [dash] | No path for 'user-keycloak-admin.yaml' found in classpath | org.apache.hadoop.security.KeycloakUserLoginInformation.LoginInformation@5e05a1f0 | java[129]
2023-07-19 [09:48:57.500] [INFO] [dash] | User 'admin' is now logged in | org.apache.hadoop.yarn.server.resourcemanager.webapp.YarnClusterApplicationNode | java[129]
2023-07-19 [09:49:59.144] [INFO] [dash] | found resource-type.yaml at file:///opt/gateway/HDFS/hadoop/resource-type.yaml | org.apache.hadoop.hdfs.ConfigParser.getResourceManagerInputPaths(Configuration.java:2007) | java[400]
2023-07-19 [09:49:59.200] [INFO] [dash] | found cluster-descriptor.yaml at file:///opt/gateway/HDFS/yarnClusterDescriptor | org.apache.hadoop.hdfs.ConfigParser.getResourceManagerInputPaths(Configuration.java:2007) | java[400]
2023-07-19 [09:49:59.200] [INFO] [dash] | Cluster specification ClusterSpecification@f0d474f4 | org.apache.hadoop.hdfs.ConfigParser.getResourceManagerInputPaths(Configuration.java:2007) | java[400]
2023-07-19 [09:49:59.200] [INFO] [dash] | Adding kerberos /org/user-keycloak | as the AM container local resource huk | org.apache.hadoop.yarn.libcству.DescriptorManager@400 | org.apache.flink.yarn.YarnClusterDescriptor.startAppMaster(YarnClusterDescriptor.java:111)
2023-07-19 [09:49:59.200] [INFO] [dash] | Adding kerberos /org/user-keycloak | as the AM container local resource huk | org.apache.hadoop.yarn.libcstu.DescriptorManager.startAppMaster(YarnClusterDescriptor.java:111)
2023-07-19 [09:49:59.200] [INFO] [dash] | Obtaining delegation tokens for HDFS and Hbase | org.apache.flink.yarn.util.ObtainDelegationTokens.java[100]
2023-07-19 [09:49:59.200] [INFO] [dash] | Obtaining delegation tokens for HDFS and Hbase | org.apache.hadoop.yarn.util.ObtainDelegationTokens.java[100]
2023-07-19 [09:49:59.200] [INFO] [dash] | Attempting to obtain kerberos security tokens for Hbase | org.apache.flink.yarn.util.ObtainDelegationTokens.java[240]
2023-07-19 [09:49:59.200] [INFO] [dash] | Attempting to obtain kerberos security tokens for HDFS | org.apache.flink.yarn.util.ObtainDelegationTokens.java[240]
2023-07-19 [09:49:59.200] [INFO] [dash] | HDFS security setting sample | org.apache.hadoop.yarn.util.ObtainDelegationTokens.java[240]
2023-07-19 [09:49:59.200] [INFO] [dash] | Submitting application master application@10650090917.042 | org.apache.flink.yarn.YarnClusterDescriptor.startAppMaster(YarnClusterDescriptor.java:1250)
2023-07-19 [09:49:59.200] [INFO] [dash] | Application master application@10650090917.042 has been submitted | org.apache.hadoop.yarn.libcstu.DescriptorManager.submitApplication(YarnClusterDescriptor.java:200)
2023-07-19 [09:49:59.200] [INFO] [dash] | Waiting for the cluster to be started | org.apache.flink.yarn.YarnClusterDescriptor.startAppMaster(YarnClusterDescriptor.java:1283)
2023-07-19 [09:49:59.200] [INFO] [dash] | Application master application@10650090917.042 has been started | org.apache.flink.yarn.YarnClusterDescriptor.startAppMaster(YarnClusterDescriptor.java:1281)
2023-07-19 [09:49:59.200] [INFO] [dash] | YARN application has been deployed successfully | org.apache.flink.yarn.YarnClusterDescriptor.startAppMaster(YarnClusterDescriptor.java:1281)
2023-07-19 [09:49:59.200] [INFO] [dash] | Application master application@10650090917.042 has been started in detached mode. In order to stop Yarn gracefully, use the following command |
| echo "stop" | > $YARN_APP_DIR/application_10650090917.042
2023-07-19 [09:49:59.200] [INFO] [dash] | Application master application@10650090917.042 will run via YARN's Web interface or |
2023-07-19 [09:49:59.200] [INFO] [dash] | YARN application will application@10650090917.042 |
2023-07-19 [09:49:59.200] [INFO] [dash] | Found web interface 168.20.28.32:8088 of application application@10650090917.042 | org.apache.flink.yarn.YarnClusterDescriptor.setClusterEntryPointInUrlToConfig(YarnClusterDescriptor.java:1054)
Cluster started Yarn cluster with application identifier application@10650090917.042
```

- yarn-application

```
./bin/flink run-application --detached -t yarn-application -Dyarn.application.name=py_sql -Dyarn.ship-  
files=/opt/client/Flink/flink/yarnship/ -pyarch yarnship/venv.zip -pyexec venv.zip/venv/bin/python3 -  
pyclientexec venv.zip/venv/bin/python3 -pyfs yarnship -pym pyflink-sql
```

## Execution result:

- yarn-session

Before starting the Yarn session, prepare the running environment by referring to [Preparing Development and Operating Environment](#). Run the following command to start yarn-session:

```
bin/yarn-session.sh -jm 1024 -tm 4096 -t conf/ssl/ -d
```

Run the following command to submit the job:

```
./bin/flink run --detached -t yarn-session -Dyarn.application.name=py_sql -Dyarn.application.id=application_1685505909197_0285 -pyarch conf/ssl/venv.zip -pyexec conf/ssl/venv.zip/venv/bin/python3 -py conf/ssl/pyflink-sql.py
```

## Execution result:

----End

## 2.4.4 Debugging the Application

### 2.4.4.1 Compiling and Running Applications

#### Scenario

After application code is developed, upload it to the Linux client to run applications. The procedures for running applications developed using Scala or Java are the same on the Flink client.

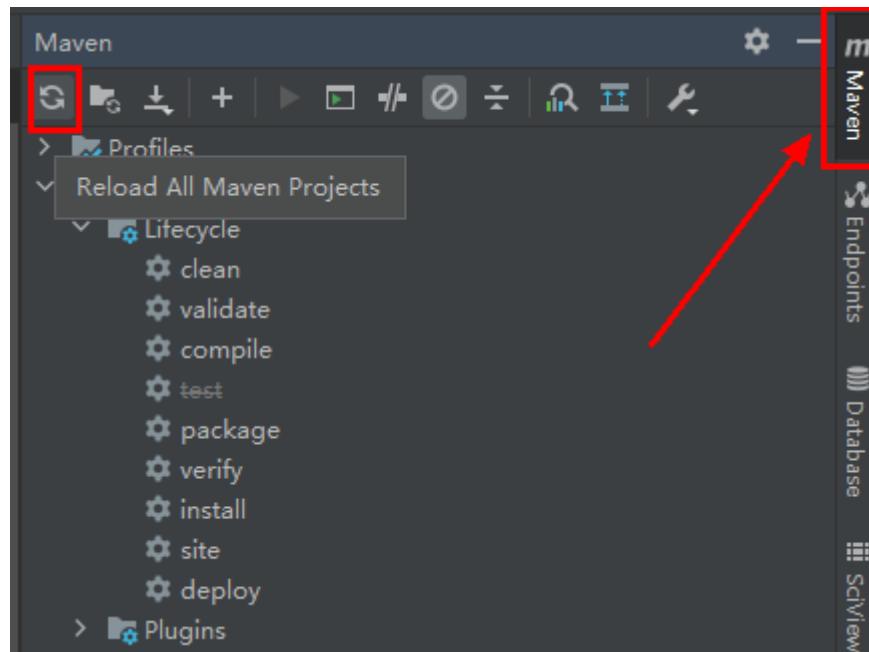


Flink applications of a YARN cluster can run only on Linux.

#### Procedure

- Step 1** In IntelliJ IDEA, click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependencies.

**Figure 2-73** Reload projects

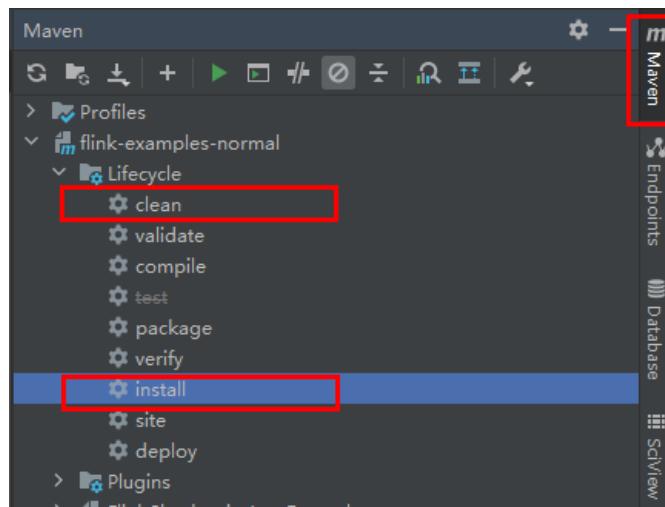


- Step 2** Compile and run the application.

Use either of the following two methods:

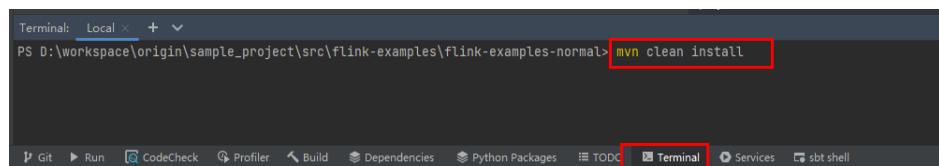
- Method 1:
  - a. Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.
  - b. Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 2-74 Maven clean and install



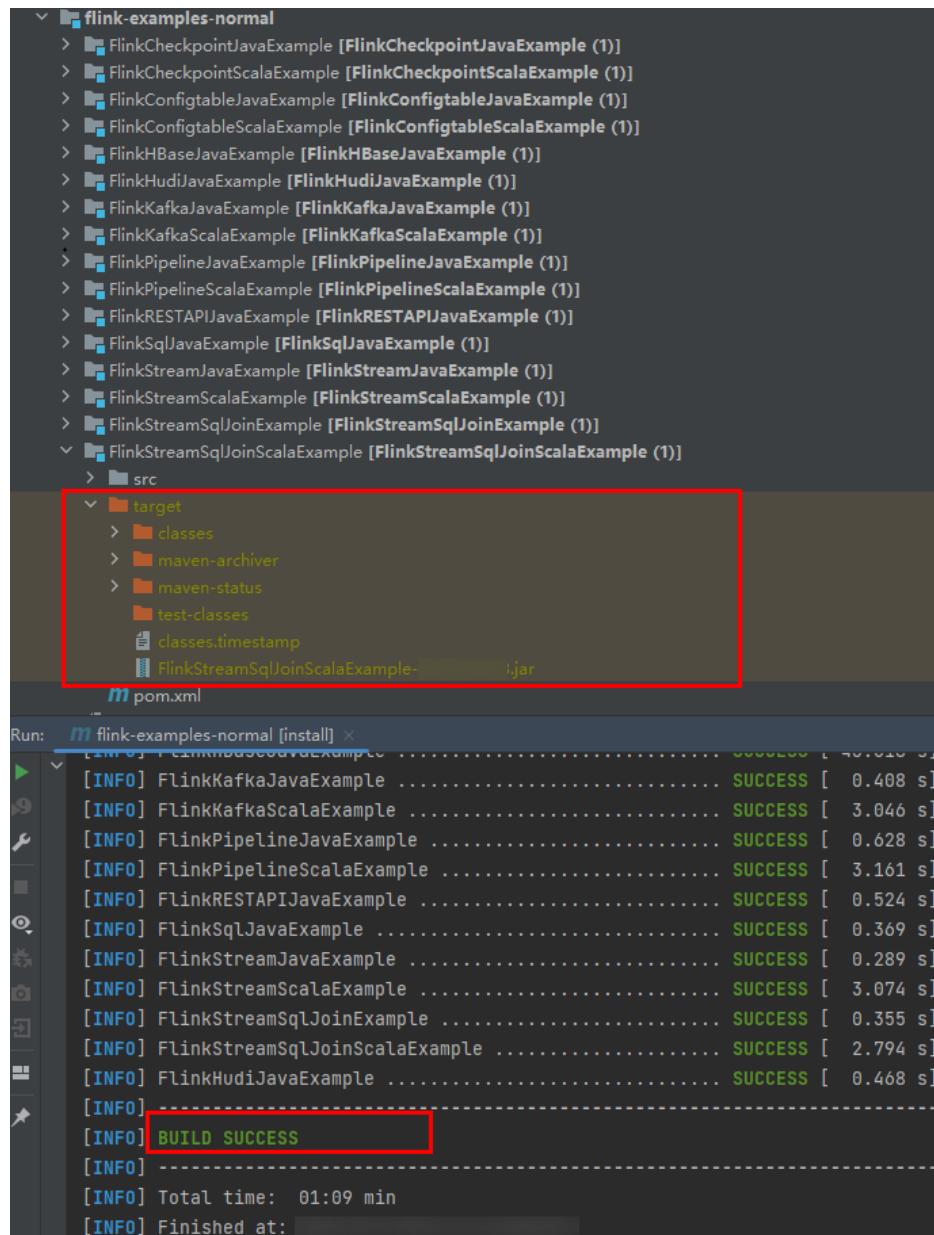
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 2-75 Entering **mvn clean install** in the IDEA Terminal text box



**Step 3** After the compilation is complete, the message "BUILD SUCCESS" is printed and the **target** directory is generated. Obtain the JAR package in the directory.

Figure 2-76 Compilation completed



**Step 4** Copy the JAR file generated in **Step 3** (for example, **FlinkStreamJavaExample.jar**) to the Flink runtime environment on Linux (that is, the Flink client), for example, **/opt/hadoopclient**. Create the **conf** directory in the directory and copy the required configuration file to **conf**. For details, see [Preparing the Operating Environment](#). Then, run the Flink application.

Start the Flink cluster before running the Flink applications on Linux. Run the yarn session command on the Flink client to start the Flink cluster. An example command is as follows:

```
bin/yarn-session.sh -jm 1024 -tm 4096
```

 NOTE

- Before running the **yarn-session.sh** command, copy the dependency package of the Flink application to the client directory **{client\_install\_home}/Flink/flink/lib**. For details about the dependency packages of the application, see [Reference information about the dependency package for running the sample project](#).
- The dependencies of different sample projects may conflict. When running a new sample project, you need to remove the dependencies copied from the old sample project to the **{client\_install\_home}/Flink/flink/lib** directory on the client.
- Run the **source bingdata\_env** command in the client installation directory before running the **yarn-session.sh** command.
- The **yarn-session.sh** command must be run in the */Flink client installation directory/Flink/flink* directory, for example, **/opt/hadoopclient/Flink/flink**.
- Do not restart the HDFS service or all DataNode instances during Flink job running. Otherwise, the job may fail and some temporary application data cannot be cleared.
- Ensure that the user permissions on the JAR file and configuration file are the same as those on the Flink client. For example, the user is **omm** and the permission is **755**.
- The memory size of TaskManagers specified by using the **-tm** command must be at least 4,096 MB.

• Running the DataStream sample application (in Scala or Java)

Open another window on the terminal. Go to the Flink client directory and invoke the **bin.flink run** script to run code.

– Java

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkStreamJavaExample /opt/hadoopclient/FlinkStreamJavaExample.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2
```

– Scala

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkStreamScalaExample /opt/hadoopclient/FlinkStreamScalaExample.jar --filePath /opt/log1.txt,/opt/log2.txt --windowTime 2
```

 NOTE

The **log1.txt** and **log2.txt** files must be stored on each node where NodeManager instances are deployed, and the permission is **755**.

**Table 2-22 Parameters**

| Parameter     | Description   |
|---------------|---|
| <filePath>    | File path in the local file system. The <b>/opt/log1.txt</b> and <b>/opt/log2.txt</b> files must be stored on every node. The default value can be retained or changed. |
| <>windowTime> | Duration of a time window, in minutes. The default value can be retained or changed.  |

- Running the sample application for producing and consuming data in Kafka (in Java or Scala)

Command for starting the application to produce data

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/hadoopclient/FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sasl.kerberos.service.name] [ssl.truststore.location] [ssl.truststore.password]
```

**Command for starting the application to consume data**

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/  
FlinkKafkaJavaExample.jar <topic> <bootstrap.servers> [security.protocol] [sasl.kerberos.service.name]  
[ssl.truststore.location] [ssl.truststore.password]
```

**Table 2-23 Parameters**

| Parameter         | Description   | Mandatory  |
|-------------------|---|--|
| topic             | Kafka topic name  | Yes  |
| bootstrap.server  | List of IP addresses or ports of broker clusters  | Yes  |
| security.protocol | The parameter can be set to <b>PLAINTEXT</b> (optional), <b>SASL_PLAINTEXT</b> , <b>SSL</b> , or <b>SASL_SSL</b> , corresponding to port <b>21005</b> , <b>21007</b> , <b>21008</b> , or <b>21009</b> of the FusionInsight Kafka cluster, respectively.<br>- If SASL is configured, <b>sasl.kerberos.service.name</b> must be set to <b>kafka</b> and the configuration items related to <b>security.kerberos.login</b> in <b>conf/flink-conf.yaml</b> must be configured.<br>- If SSL is configured, <b>ssl.truststore.location</b> (path of <b>truststore</b> ) and <b>ssl.truststore.password</b> (password of <b>truststore</b> ) must be configured. | No<br><b>NOTE</b><br>- If this parameter is not configured, Kafka is in non-security mode.<br>- If SSL needs to be configured, find more information about how to generate the <b>truststore.jks</b> file in <b>More Information &gt; External Interfaces &gt; SSL Encryption Function Used by a Client of Kafka Development Guide</b> . |

The following commands use ReadFromKafka as an example and the cluster domain name is **HADOOP.COM**:

- **Command 1:**  
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/  
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21005
- **Command 2:**  
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/  
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21005 --

- ```
security.protocol PLAINTEXT --sasl.kerberos.service.name kafka --kerberos.domain.name
hadoop.hadoop.com
```
- Command 3:  

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21008 --
security.protocol SSL --ssl.truststore.location /home/truststore.jks --ssl.truststore.password xxx
```
- Command 4:  

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromKafka /opt/hadoopclient/
FlinkKafkaJavaExample.jar --topic topic1 --bootstrap.servers 10.96.101.32:21005 --
security.protocol PLAINTEXT --sasl.kerberos.service.name kafka --ssl.truststore.location config/
truststore.jks --ssl.truststore.password xxx --kerberos.domain.name hadoop.hadoop.com
```
- Running the sample application of the asynchronous checkpoint mechanism (in Scala or Java)  

To diversify sample code, the processing time is used as a timestamp for data stream in Java, and the event time is used as a timestamp for data stream in Scala. The command reference is as follows:

Saving checkpoint snapshot information to HDFS

  - Java  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkProcessingTimeAPIMain /opt/
hadoopclient/FlinkCheckpointJavaExample.jar --chkPath hdfs://hacluster/flink/checkpoint/
```
  - Scala  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkEventTimeAPIChkMain /opt/
hadoopclient/FlinkCheckpointScalaExample.jar --chkPath hdfs://hacluster/flink/checkpoint/
```

 NOTE

  - Checkpoint source file path: **flink/checkpoint/fd5f5b3d08628d83038a30302b611/chk-X/4f854bf4-ea54-4595-a9d9-9b9080779ffe**  
**flink/checkpoint**: specified root directory  
**fd5f5b3d08628d83038a30302b611**: level-2 directory named after jobID  
**chk-X**: "X" indicates the checkpoint number, which is the level-3 directory  
**4f854bf4-ea54-4595-a9d9-9b9080779ffe**: a checkpoint source file
  - If Flink is in cluster mode, checkpoints store the file in HDFS. A local path can be used only when Flink is in local mode, facilitating commissioning.- Running the Pipeline sample application
  - Java
    - i. Start the publisher job.  

```
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipelineNettySink /opt/
hadoopclient/FlinkPipelineJavaExample.jar
```
    - ii. Start the subscriber Job1.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipelineNettySource1 /opt/
hadoopclient/FlinkPipelineJavaExample.jar
```
    - iii. Start the subscriber Job2.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipelineNettySource2 /opt/
hadoopclient/FlinkPipelineJavaExample.jar
```
  - Scala
    - i. Start the publisher job.  

```
bin/flink run -p 2 --class com.huawei.bigdata.flink.examples.TestPipeline_NettySink /opt/
hadoopclient/FlinkPipelineScalaExample.jar
```
    - ii. Start the subscriber Job1.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource1 /opt/
hadoopclient/FlinkPipelineScalaExample.jar
```

- iii. Start the subscriber Job2.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.TestPipeline_NettySource2 /opt/hadoopclient/FlinkPipelineScalaExample.jar
```
- Running the sample application for configuring JOIN between tables and streams (for example, FlinkConfigtableJavaExample and FlinkConfigtableScalaExample)

The execution modes of Java are as follows:

- yarn-session mode

- i. Import **configtable.csv** to the Redis cluster.  

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/FlinkConfigtableJavaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --configPath config/import.properties
```

- ii. Start the Flink cluster.

```
bin/yarn-session.sh -t config -jm 1024 -tm 4096
```

- iii. Stream data reads personal information from Redis using netizen names as keywords, joins the information, and generates the output.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableJavaExample /opt/hadoopclient/FlinkConfigtableJavaExample.jar --dataPath config/data.txt
```

- yarn-cluster mode

- i. Import **configtable.csv** to the Redis cluster.

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/FlinkConfigtableJavaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --configPath config/import.properties
```

- ii. Start the Flink cluster. Stream data reads personal information from Redis using netizen names as keywords, joins the information, and generates the output.

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableJavaExample -m yarn-cluster -yt config -yjm 1024 -ytm 4096 /opt/hadoopclient/FlinkConfigtableJavaExample.jar --dataPath config/data.txt
```

The execution modes of Scala are as follows:

- yarn-session mode

- i. Import **configtable.csv** to the Redis cluster.

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/FlinkConfigtableScalaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --configPath config/import.properties
```

- ii. Start the Flink cluster.

```
bin/yarn-session.sh -t config -jm 1024 -tm 4096
```

- iii. Stream data reads personal information from Redis using netizen names as keywords, joins the information, and generates the output.  

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableScalaExample /opt/hadoopclient/FlinkConfigtableScalaExample.jar --dataPath config/data.txt
```

- yarn-cluster mode

- i. Import **configtable.csv** to the Redis cluster.

```
java -cp /opt/hadoopclient/Flink/flink/lib/*:/opt/hadoopclient/FlinkConfigtableScalaExample.jar com.huawei.bigdata.flink.examples.RedisDataImport --configPath config/import.properties
```

- ii. Start the Flink cluster. Stream data reads personal information from Redis using netizen names as keywords, joins the information, and generates the output.

```
bin/flink run --class com.huawei.bigdata.flink.examples.FlinkConfigtableScalaExample -m yarn-cluster -yt config -yjm 1024 -ytm 4096 /opt/hadoopclient/FlinkConfigtableScalaExample.jar --dataPath config/data.txt
```

- Running the Stream SQL Join sample application

- Java

- i. Start the application to generate data for Kafka. For details about Kafka configuration, see [Running the sample application for producing and consuming data in Kafka \(in Java or Scala\)](#).

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/hadoopclient/FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.21005
```

- ii. Run the **netcat** command on any node in the cluster to wait for an application connection.

```
netcat -l -p 9000
```

 NOTE

If "command not found" is displayed, install NetCat and run the command again.

- iii. Start the application to receive socket data and perform a joint query.

```
bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/hadoopclient/FlinkStreamSqlJoinExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.21005 --hostname xxx.xxx.xxx.21005 --port 9000
```

- Scala

- i. Start the application to produce data in Kafka. For details about how to configure Kafka, see [Running the sample application for producing and consuming data in Kafka \(in Java or Scala\)](#).

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoKafka /opt/hadoopclient/FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.21005
```

- ii. Run the **netcat** command on any node in the cluster to wait for an application connection.

```
netcat -l -p 9000
```

- iii. Start the application to receive socket data and perform a joint query.

```
bin/flink run --class com.huawei.bigdata.flink.examples.SqlJoinWithSocket /opt/hadoopclient/FlinkStreamSqlJoinScalaExample.jar --topic topic-test --bootstrap.servers xxx.xxx.xxx.21005 --hostname xxx.xxx.xxx.21005 --port 9000
```

● Running the Flink HBase sample application

- yarn-session mode

- i. Start the Flink cluster.

```
./bin/yarn-session.sh -jm 1024 -tm 4096
```

- ii. Run the Flink application and enter parameters.

```
bin/flink run --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```

```
bin/flink run --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```

- yarn-cluster mode

```
bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```

```
bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadHBase /opt/client1/Flink/flink/FlinkHBaseJavaExample-xxx.jar --tableName xxx --confDir xxx
```

● Running the Flink Hudi sample application

- yarn-session mode

- i. Start the Flink cluster.

```
./bin/yarn-session.sh -jm 1024 -tm 4096
```

ii. Run the Flink application and enter parameters.

```
./bin/flink run --class com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/test.jar --  
hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable
```

```
./bin/flink run --class com.huawei.bigdata.flink.examples.ReadFromHudi /opt/test.jar --  
hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable --  
read.start-commit xxx
```

- yarn-cluster mode

```
./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.WriteIntoHudi /opt/  
test.jar --hudiTableName hudiSinkTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable
```

```
./bin/flink run -m yarn-cluster --class com.huawei.bigdata.flink.examples.ReadFromHudi /opt/  
test.jar --hudiTableName hudiSourceTable --hudiPath hdfs://hacluster/tmp/flinkHudi/hudiTable  
--read.start-commit xxx
```

- Run the REST APIs to create a tenant sample application. The TestCreateTenants application is used as an example.

- yarn-session mode

i. Start the Flink cluster.

```
./bin/yarn-session.sh -jm 1024 -tm 4096
```

ii. Run the Flink application and enter parameters.

```
./bin/flink run --class com.huawei.bigdata.flink.examples.TestCreateTenants /opt/  
hadoopclient/FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx
```

- yarn-cluster mode

```
./bin/flink run -m yarn-cluster -yjm 1024 -ytm 4096 --class  
com.huawei.bigdata.flink.examples.TestCreateTenants /opt/hadoopclient/  
FlinkRESTAPIJavaExample-xxx.jar --hostName xx-xx-xx-xx
```

- Run a SQL task to submit Flink JAR jobs.

- yarn-session mode

i. Start the Flink cluster.

```
./bin/yarn-session.sh -jm 1024 -tm 4096
```

ii. Run the Flink application and enter parameters.

```
bin/flink run -d --class com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/  
datagen2kafka.sql
```

- yarn-cluster mode

```
bin/flink run -m yarn-cluster -yjm 1024 -ytm 4096 -d --class  
com.huawei.mrs.FlinkSQLExecutor /opt/flink-sql-xxx.jar --sql ./sql/datagen2kafka.sql
```

----End

#### 2.4.4.2 Viewing the Debugging Result

##### Scenarios

After a Flink application completes running, you can view the running result, or use Apache Flink Dashboard to view application running status.

##### Procedure

- **View the running result of the Flink application.**

If you want to check the execution result, view the Stdout log of Task Manager on the Apache Flink Dashboard.

If the execution result is exported to a file or a location specified by Flink, view the result from the exported file or the location. The checkpoint, pipeline, and join between configuration tables and streams are used as examples.

- View checkpoint results and files

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Checkpoints** to view the submitted jobs as shown in [Figure 2-77](#). Choose **Task Managers > Stdout** to view the running result, as shown in [Figure 2-78](#).

**Figure 2-77** Submitted jobs

ID	Status	Target Time	Latest Acknowledgment	End-to-End Duration	Data Size	Batched During Acknowledgment
0	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
1	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
2	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
3	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
4	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
5	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
6	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
7	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
8	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
9	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
10	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
11	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
12	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
13	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
14	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
15	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
16	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
17	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
18	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
19	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
20	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
21	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
22	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
23	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
24	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
25	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
26	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0
27	Completed	2020-12-23 10:40:34	2020-12-23 10:40:34	00:00ms	0.0 MB	0.0

**Figure 2-78** Execution result

```

akka.tcp://flink@160.80.69.37:7742/user/taskmanager_0
Last Heartbeat: 20-12-23 10:42:40
container.e01.160.80.69.37.7742.0021.01.000002
Data Port: 32391 Free Slots / All Slots: 0 / 1 CPU Cores: 29 Physical Memory: 124 GB JVM Heap Size: 229 MB
Flink Managed Memory: 294 MB

Metrics Log Stdout
1 2020-12-23 10:40:34,17,272 | 1
2 2020-12-23 10:40:34,17,272 | 1
3 2020-12-23 10:40:34,17,272 | 1
4 2020-12-23 10:40:34,17,272 | 1
5 2020-12-23 10:40:34,17,272 | 1
6 2020-12-23 10:40:34,17,272 | 1
7 2020-12-23 10:40:34,17,272 | 1
8 2020-12-23 10:40:34,17,272 | 1
9 2020-12-23 10:40:34,17,272 | 1
10 2020-12-23 10:40:34,17,272 | 1
11 2020-12-23 10:40:34,17,272 | 1
12 2020-12-23 10:40:34,17,272 | 1
13 2020-12-23 10:40:34,17,272 | 1
14 2020-12-23 10:40:34,17,272 | 1
15 2020-12-23 10:40:34,17,272 | 1
16 2020-12-23 10:40:34,17,272 | 1
17 2020-12-23 10:40:34,17,272 | 1
18 2020-12-23 10:40:34,17,272 | 1
19 2020-12-23 10:40:34,17,272 | 1
20 2020-12-23 10:40:34,17,272 | 1
21 2020-12-23 10:40:34,17,272 | 1
22 2020-12-23 10:40:34,17,272 | 1
23 2020-12-23 10:40:34,17,272 | 1
24 2020-12-23 10:40:34,17,272 | 1
25 2020-12-23 10:40:34,17,272 | 1
26 2020-12-23 10:40:34,17,272 | 1
27 2020-12-23 10:40:34,17,272 | 1

```

- Run the `hdfs dfs -ls hdfs://hacluster/flink/checkpoint/` command to view the checkpoint snapshot information in the HDFS.

- View pipeline results

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in [Figure 2-79](#). Choose **Task Managers**. You can see two tasks, as shown in [Figure 2-80](#). Click any task, choose **Stdout** to view the output of the task, as shown in [Figure 2-81](#) and [Figure 2-82](#).

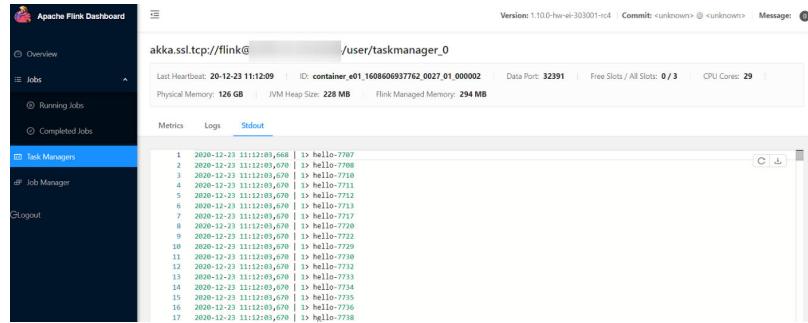
**Figure 2-79** Running jobs

Job Name	Start Time	Duration	End Time	Tasks	Status
Flink Streaming Job	2020-12-23 11:07:44	0m 0s	-	2	RUNNING
Flink Streaming Job	2020-12-23 11:07:58	4m 24s	-	2	RUNNING
Flink Streaming Job	2020-12-23 11:06:35	5m 8s	-	4	RUNNING

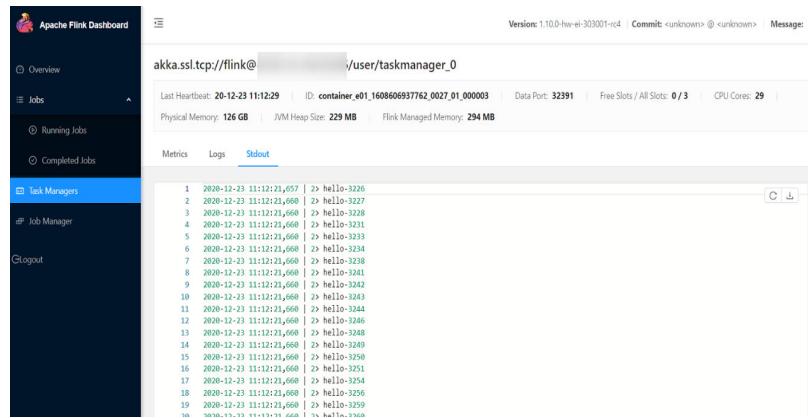
**Figure 2-80** Submitted tasks

Path, ID	Data Port	Last Heartbeat	All Slots	Free Slots	CPU Cores	Physical MEM	JVM MEM	Flink Managed MEM
com.flink.e01.160.80.69.37.7742.0021.01.000002	32391	20-12-23 11:11:49	3	0	29	124 GB	238 MB	294 MB
com.flink.e01.160.80.69.37.7742.0021.01.000002	32391	20-12-23 11:11:49	3	0	29	124 GB	238 MB	294 MB

**Figure 2-81** Output of task1



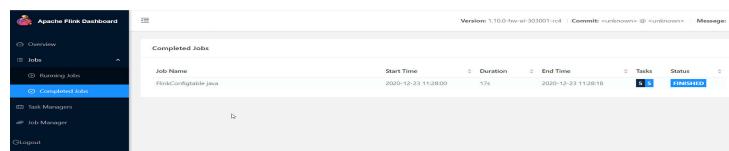
**Figure 2-82** Output of task2



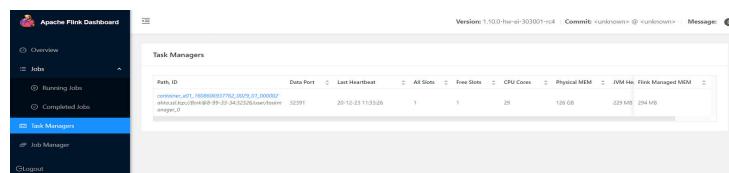
- View the JOIN result of configuration table and streams

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Completed Jobs** to view the completed job as shown in [Figure 2-83](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 2-84](#). Choose **Stdout** to view the running result, as shown in [Figure 2-85](#).

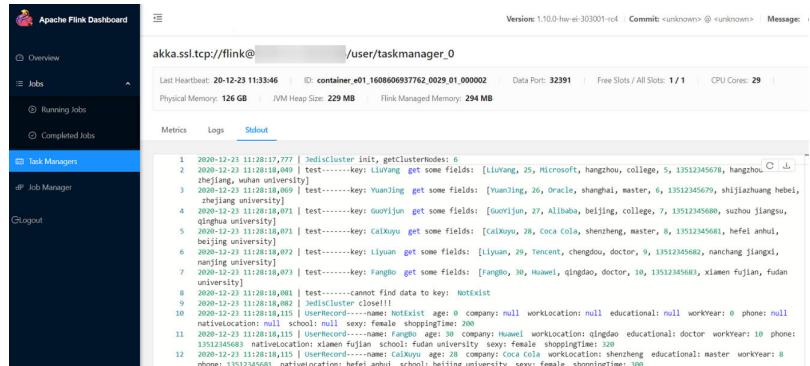
**Figure 2-83** Completed job



## **Figure 2-84 Submitted task**



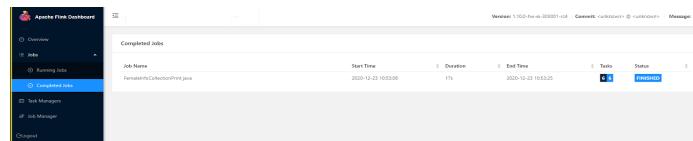
**Figure 2-85** Execution result



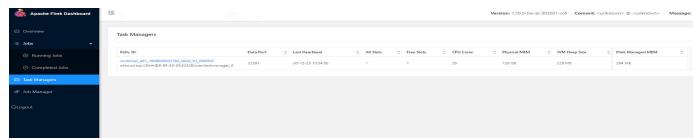
- View the result of DataStream

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Completed Jobs** to view the completed job as shown in [Figure 2-86](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 2-87](#). Choose **Stdout** to view the running result, as shown in [Figure 2-88](#).

**Figure 2-86** Completed job



## **Figure 2-87 Submitted task**



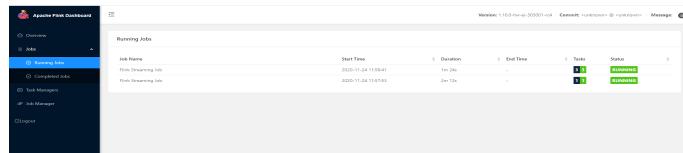
**Figure 2-88** Execution result



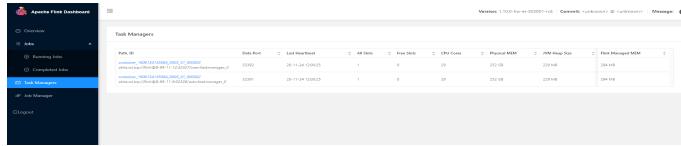
- View the result of stream sql join

- The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in [Figure 2-89](#). Choose **Task Managers**, you can see submitted task, as shown in [Figure 2-90](#). Choose **Stdout** to view the running result, as shown in [Figure 2-91](#).

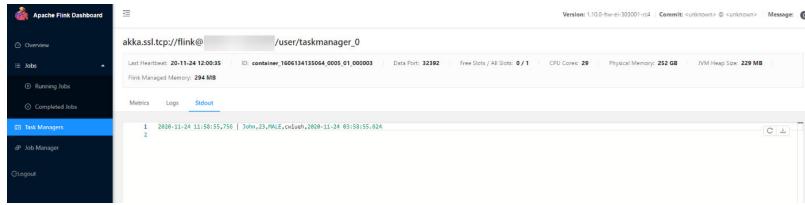
**Figure 2-89** Running jobs



**Figure 2-90 Submitted task**

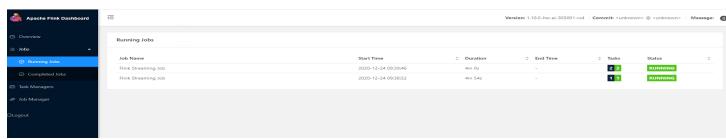


**Figure 2-91 Execution result**

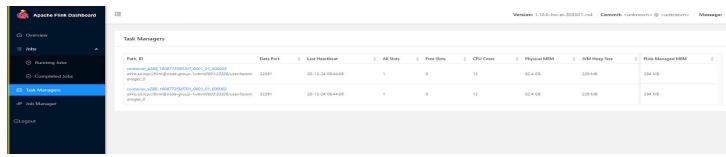


- **View the result of produce and consume data in Kafka**
  - The results are stored in the **taskmanager.out** file of Flink. User can log in to the WEB UI of the Yarn. Choose **Jobs > Running Jobs** to view the running jobs as shown in **Figure 2-92**. Choose **Task Managers**, you can see submitted task, as shown in **Figure 2-93**. Choose **Stdout** to view the running result, as shown in **Figure 2-94**.

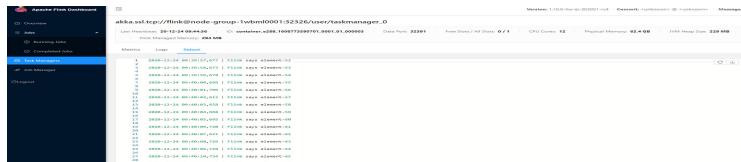
**Figure 2-92 Running jobs**



**Figure 2-93 Submitted task**



**Figure 2-94 Execution result**



- **Use Apache Flink Dashboard to view the running status of the Flink application.**

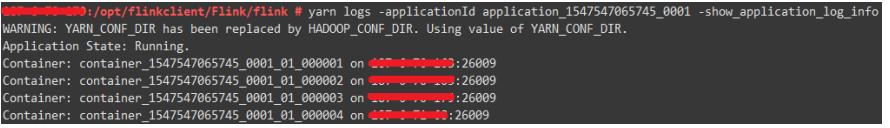
The Apache Flink Dashboard mainly includes Overview, Running Jobs, Completed Jobs, Task Managers, Job Manager and Logout and so on.

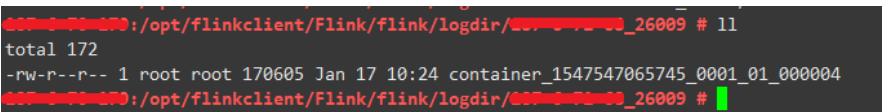
On In the YARN web UI, find the desired Flink application. Click the **ApplicationMaster** at the last column of the application to switch to the Apache Flink Dashboard.

View the print results of the program execution: find the corresponding **Task Manager** to see the corresponding **Stdout** tag log information.

- **View Flink logs.**

Three methods can be used to obtain Flink logs:

- Log in to the Apache Flink Dashboard and view logs of TaskManagers and JobManager.
- Log in to the YARN web UI to view logs about JobManager and GC. On the YARN web UI wind, find the desired Flink application. Click the **ID** of the application. On the switched page, click **Logs** in the Logs column.
- On the Yarn client, obtain or view logs of Task Managers and Job Manager.
  - i. Download and install the Yarn client, for example, in the /opt/hadoopclient directory.
  - ii. Use PuTTY to log in to the node where the client is installed as the client installation user.
  - iii. Run the following command to switch to the client installation directory:  
**cd /opt/hadoopclient**
  - iv. Run the following command to configure environment variables:  
**source bigdata\_env**
  - v. If the cluster employs the security mode, run the following command to authenticate the user. If the normal mode is used, skip this step.  
**kinit component service user**
  - vi. Run the following commands to obtain container information of the Flink cluster:  
**yarn logs -applicationId application\_\* -show\_application\_log\_info**  


```
[root@node01:~/opt/flinkclient/Flink/flink # yarn logs -applicationId application_1547547065745_0001 -show_application_log_info
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
Application State: Running.
Container: container_1547547065745_0001_01_000001 on [REDACTED]:26009
Container: container_1547547065745_0001_01_000002 on [REDACTED]:26009
Container: container_1547547065745_0001_01_000003 on [REDACTED]:26009
Container: container_1547547065745_0001_01_000004 on [REDACTED]:26009
```
  - vii. Run the following command to obtain run logs of the specified container. Generally, container\_\*\_000001 is the container where the Job Manager is running.  
**yarn logs -applicationId application\_\* --containerId container\_1547547065745\_0001\_01\_000004 -out logdir**  


```
[root@node01:~/opt/flinkclient/Flink/flink/logdir/container_1547547065745_0001_01_000004_26009 # ll
total 172
-rw-r--r-- 1 root root 170605 Jan 17 10:24 container_1547547065745_0001_01_000004
[root@node01:~/opt/flinkclient/Flink/flink/logdir/container_1547547065745_0001_01_000004_26009 #
```

After the command is executed, container run logs, including run logs of the Task Manager and Job Manager and GC logs, are downloaded to the local host.

- viii. You can also run a command to obtain the log with the specified name:

The following command is used to obtain the container log list.

**yarn logs -applicationId application\_\* -show\_container\_log\_info --containerId container\_1547547065745\_0001\_01\_000004**

LogFile	LogLength	LastModificationTime	LogAggregationType
container-localizer-syslog	184	Wed Jan 16 17:49:27 +0800 2019	LOCAL
taskmanager.log	131430	Wed Jan 16 17:49:35 +0800 2019	LOCAL
gc.log.0.current	17952	Thu Jan 17 10:22:26 +0800 2019	LOCAL
taskmanager.out	0	Wed Jan 16 17:49:31 +0800 2019	LOCAL
launch_container.sh	12232	Wed Jan 16 17:49:31 +0800 2019	LOCAL
directory.info	3661	Wed Jan 16 17:49:31 +0800 2019	LOCAL
taskmanager.err	1060	Wed Jan 16 17:49:31 +0800 2019	LOCAL
prelaunch.out	160	Wed Jan 16 17:49:31 +0800 2019	LOCAL
prelaunch.err	0	Wed Jan 16 17:49:31 +0800 2019	LOCAL

Download **taskmanager.log** to the local host.

```
yarn logs -applicationId application_* --containerId
container_1547547065745_0001_01_000004 -log_files
taskmanager.log -out localpath
```

#### 2.4.4.3 Running a Spring Boot Sample Project and Viewing Results

##### Running the Spring Boot Sample in CLI

**Step 1** Use Maven to run **install** on the IDEA.

If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR file containing the **flink-dws-sink-example-1.0.0-SNAPSHOT** field is generated in the **target** directory of the sample project.

```
[INFO] Dependency-reduced POM written at: D:\code\spring\sample_project\src\springboot\flink-examples\flink-dws-sink-example\dependency-reduced-pom.xml
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ flink-dws-sink-example ---
[INFO] Installing D:\code\spring\sample_project\src\springboot\flink-examples\flink-dws-sink-example\target\flink-dws-sink-example.jar to D:\soft\maven\local\org\springframework\boot\flink-dws-sink-example\flink-dws-sink-example-1.0.0-SNAPSHOT.jar
[INFO] Installing D:\code\spring\sample_project\src\springboot\flink-examples\flink-dws-sink-example\dependency-reduced-pom.xml to D:\soft\maven\local\org\springframework\boot\flink-dws-sink-example\dependency-reduced-pom.xml
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 15.581 s
[INFO] Finished at: 2023-08-18T11:39:01+08:00
[INFO]
[INFO] Process finished with exit code 0
```

**Step 2** On Linux, go to the client installation directory, for example, **/opt/client/Flink/flink/conf**, and save the JAR packages whose names contain **flink-dws-sink-example-1.0.0-SNAPSHOT** in the **target** directory generated in to this directory.



The Flink client running the Spring Boot sample must contain only the Flink service.

**Step 3** Run the following command to create a Yarn session:

```
yarn-session.sh -t ssl/ -nm "session-spring11" -d
```

```
[root@host1 conf]#
[root@host1 conf]#
[root@host1 conf]# yarn-session.sh -t ssl/ -nm "session-spring11" -d
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client123/Flink/flink/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client123/HDFS/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
2023-08-18 09:54:32,938 | INFO | [main] | Loading configuration property: akka.ask.timeout, 300 s | org.apache.actor.ActorConfig$Configuration$GlobalConfiguration.java:224)
2023-08-18 09:54:32,945 | INFO | [main] | Loading configuration property: akka.client-socket-worker-pool.configuration.loadYAMLResource(GlobalConfiguration.java:224)
2023-08-18 09:54:32,945 | INFO | [main] | Loading configuration property: akka.client-socket-worker-pool.configuration.loadYAMLResource(GlobalConfiguration.java:224)
2023-08-18 09:54:32,945 | INFO | [main] | Loading configuration property: akka.client-socket-worker-pool.configuration.loadYAMLResource(GlobalConfiguration.java:224)
```

**Step 4** Run the following command to start the SpringBoot service:

- Run the GaussDB(DWS) sample

```
flink run flink-dws-sink-example.jar
```

```
password for test@ADODR.COM:
[root@host1 conf]#
[root@host1 conf]# ./flink run flink-dws-sink-example.jar
OpenJDK 64-Bit Server VM warning: Cannot open file <LOG_DR>/gc.log due to No such file or directory

SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client123/Flink/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client123/HDFS/hadoop/share/hadoop/common/lib/slf4j-reloadable-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.jcl.Log4jLoggerFactory]
2023-08-18 11:32:46.153 | INFO | [main] | Found Yarn properties file under /opt/client123/Flink/tmp/.yarn-properties-root | org.apache.flink.yarn.cli.FlinkYarnSessionCl
i-sinit|[FlinkYarnSessionCli.java:293]
2023-08-18 11:32:46.153 | INFO | [main] | Found Yarn properties file under /opt/client123/Flink/tmp/.yarn-properties-root | org.apache.flink.yarn.cli.FlinkYarnSessionCl
i-sinit|[FlinkYarnSessionCli.java:293]
2023-08-18 11:32:46.474 | INFO | [main] | Login successful for user test using keytab file user.keytab. Keytab auto renewal enabled : false | org.apache.hadoop.security.UserGroupInformation.loginUserFromKeytab(UserGroupInformation.java:1129)


Spring Boot (v2.7.0)

2023-08-18 11:32:53.023 | INFO | [job-thread] | No path for the flink jar passed. Using the location of class org.apache.flink.yarn.YarnClusterDescriptor to locate the j
2023-08-18 11:32:53.121 | INFO | [job-thread] | Waiting over to 28 | org.apache.hadoop.yarn.client.ConfiguredRMFailoverProxyProvider.performFailover(ConfiguredRMFailoverProxyProvider.java:108)
2023-08-18 11:32:53.568 | INFO | [job-thread] | Found Web Interface 192-168-227-207:32261 of application 'application_1691142278253_0057' | org.apache.flink.yarn.YarnCl
usterDescriptor|YarnClusterDescriptorConfig|YarnClusterDescriptorConfig(yarnClusterDescriptor.java:1854)
has been submitted with jobID 9f93b9e0-90c0-4d82-2024-000000000000
```

- Run the Elasticsearch sample

**flink run flink-es-sink-example.jar**

-----End

## 2.4.5 More Information

### 2.4.5.1 Introduction to Common APIs

#### **2.4.5.1.1 Java**

To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended. For details about APIs, see MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1).

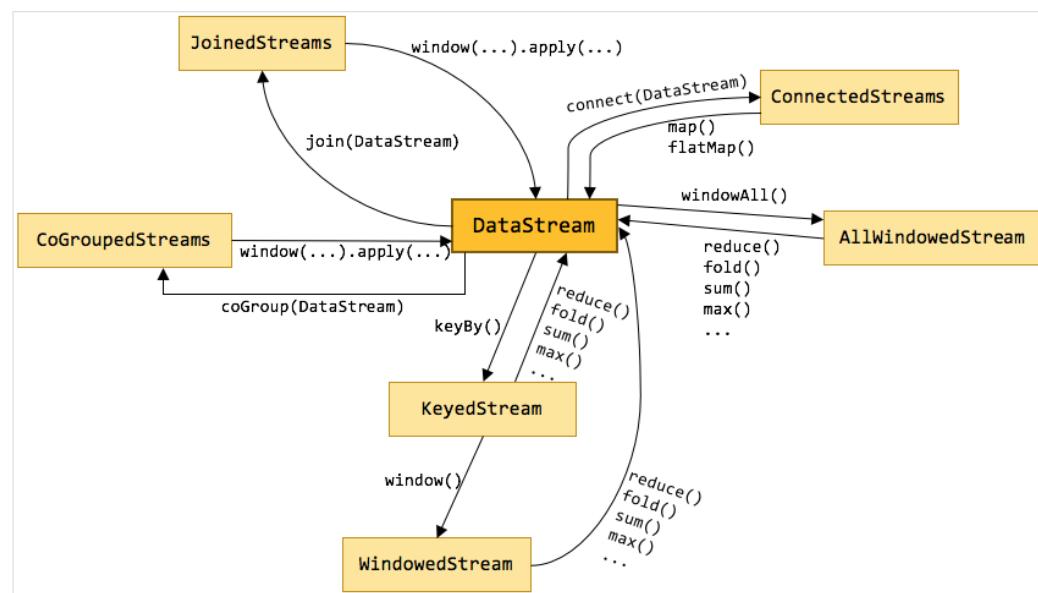
## Common APIs of Flink

Flink mainly uses the following APIs:

- StreamExecutionEnvironment: provides the execution environment, which is the basis of Flink stream processing.
  - DataStream: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
  - KeyedStream: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
  - WindowedStream: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.

- AllWindowedStream: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.
- ConnectedStreams: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- JoinedStreams: performs equijoin (which is performed when two values are equal, for example, a.id = b.id) operation to data in the window. The join operation is a special scenario of coGroup operation.
- CoGroupedStreams: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

**Figure 2-95** Conversion of Flink stream types



## Data Stream Source

**Table 2-24** APIs about data stream source

API	Description
<pre>public final &lt;OUT&gt; DataStreamSource&lt;OUT&gt; fromElements(OUT... data)</pre>	Obtain user-defined data of multiple elements as the data stream source. <ul style="list-style-type: none"> <li>• <b>type</b> indicates the data type of an element.</li> <li>• <b>data</b> indicates the data of multiple elements.</li> </ul>
<pre>public final &lt;OUT&gt; DataStreamSource&lt;OUT&gt; fromElements(Class&lt;OUT&gt; type, OUT... data)</pre>	

API	Description
public <OUT> DataStreamSource<OUT> fromCollection(Collection<OUT> data)	Obtain the user-defined data collection as the data stream source. <ul style="list-style-type: none"> <li>• <b>type</b> indicates the data type of elements in the collection.</li> </ul>
public <OUT> DataStreamSource<OUT> fromCollection(Collection<OUT> data, TypeInformation<OUT> typeInfo)	<ul style="list-style-type: none"> <li>• <b>typeInfo</b> indicates the type information obtained based on the element data type in the collection.</li> </ul>
public <OUT> DataStreamSource<OUT> fromCollection(Iterator<OUT> data, Class<OUT> type)	<ul style="list-style-type: none"> <li>• <b>data</b> indicates the iterator.</li> </ul>
public <OUT> DataStreamSource<OUT> fromCollection(Iterator<OUT> data, TypeInformation<OUT> typeInfo)	
public <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableIterator<OUT> iterator, Class<OUT> type)	Obtain the user-defined data collection as parallel data stream source. <ul style="list-style-type: none"> <li>• <b>type</b> indicates the data type of elements in the collection.</li> </ul>
public <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableIterator<OUT> iterator, TypeInformation<OUT> typeInfo)	<ul style="list-style-type: none"> <li>• <b>typeInfo</b> indicates the type information obtained based on the element data type in the collection.</li> </ul>
private <OUT> DataStreamSource<OUT> fromParallelCollection(SplittableIterator<OUT> iterator, TypeInformation<OUT> typeInfo, String operatorName)	<ul style="list-style-type: none"> <li>• <b>iterator</b> indicates the iterator that can be divided into multiple partitions.</li> </ul>
public DataStreamSource<Long> generate Sequence(long from, long to)	Obtain a sequence of user-defined data as the data stream source. <ul style="list-style-type: none"> <li>• <b>from</b> indicates the starting point of numbers.</li> <li>• <b>to</b> indicates the end point of numbers.</li> </ul>
public DataStreamSource<String> readTextFile(String filePath)	Obtain the user-defined text file from a specific path as the data stream source.
public DataStreamSource<String> readTextFile(String filePath, String charsetName)	<ul style="list-style-type: none"> <li>• <b>filePath</b> indicates the path of the text file.</li> <li>• <b>charsetName</b> indicates the encoding format.</li> </ul>

API	Description
public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath)	Obtain the user-defined file from a specific path as the data stream source. <ul style="list-style-type: none"><li>• <b>filePath</b> indicates the file path.</li></ul>
public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval)	<ul style="list-style-type: none"><li>• <b>inputformat</b> indicates the format of the file.</li><li>• <b>watchType</b> indicates the file processing mode, which can be <b>PROCESS_ONCE</b> or <b>PROCESS_CONTINUOUSLY</b>.</li><li>• <b>interval</b> indicates the interval for processing directories or files.</li></ul>
public <OUT> DataStreamSource<OUT> readFile(FileInputformat<OUT> inputformat, String filePath, FileProcessingMode watchType, long interval, TypeInformation<OUT> typeInformation)	
public DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter, long maxRetry)	Obtain user-defined socket data as the data stream source. <ul style="list-style-type: none"><li>• <b>hostname</b> indicates the host name of the socket server.</li></ul>
public DataStreamSource<String> socketTextStream(String hostname, int port, String delimiter)	<ul style="list-style-type: none"><li>• <b>port</b> indicates the listening port of the server.</li><li>• <b>delimiter</b> indicates the separator of messages.</li></ul>
public DataStreamSource<String> socketTextStream(String hostname, int port)	<ul style="list-style-type: none"><li>• <b>maxRetry</b> refers to the maximum retry times that can be triggered by abnormal connections.</li></ul>
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function)	Customize the <b>SourceFunction</b> and <b>addSource</b> methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction. <ul style="list-style-type: none"><li>• <b>function</b> indicates the user-defined <b>SourceFunction</b> function.</li></ul>
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, String sourceName)	<ul style="list-style-type: none"><li>• <b>sourceName</b> indicates the name of data source.</li></ul>
public <OUT> DataStreamSource<OUT> addSource(SourceFunction<OUT> function, TypeInformation<OUT> typeInfo)	<ul style="list-style-type: none"><li>• <b>typeInfo</b> indicates the type information obtained based on the element data type.</li></ul>

API	Description
<pre>public &lt;OUT&gt; DataStreamSource&lt;OUT&gt; addSource(SourceFunction&lt;OUT&gt; function, String sourceName, TypeInformation&lt;OUT&gt; typeInfo)</pre>	

## Data Output

Table 2-25 APIs about data output

API	Description
<code>public DataStreamSink&lt;T&gt; print()</code>	Print data as the standard output stream.
<code>public DataStreamSink&lt;T&gt; printToErr()</code>	Print data as the standard error output stream.
<code>public DataStreamSink&lt;T&gt; writeAsText(String path)</code>	Write data to a specific text file. <ul style="list-style-type: none"> <li>• <b>path</b> indicates the path of the text file.</li> </ul>
<code>public DataStreamSink&lt;T&gt; writeAsText(String path, WriteMode writeMode)</code>	<ul style="list-style-type: none"> <li>• <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b>.</li> </ul>
<code>public DataStreamSink&lt;T&gt; writeAsCsv(String path)</code>	Writhe data to a specific .csv file. <ul style="list-style-type: none"> <li>• <b>path</b> indicates the path of the text file.</li> </ul>
<code>public DataStreamSink&lt;T&gt; writeAsCsv(String path, WriteMode writeMode)</code>	<ul style="list-style-type: none"> <li>• <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b>.</li> </ul>
<code>public &lt;X extends Tuple&gt; DataStreamSink&lt;T&gt; writeAsCsv(String path, WriteMode writeMode, String rowDelimiter, String fieldDelimiter)</code>	<ul style="list-style-type: none"> <li>• <b>rowDelimiter</b> indicates the row separator.</li> <li>• <b>fieldDelimiter</b> indicates the column separator.</li> </ul>
<code>public DataStreamSink&lt;T&gt; writeToSocket(String hostName, int port, SerializationSchema&lt;T&gt; schema)</code>	Write data to the socket connection. <ul style="list-style-type: none"> <li>• <b>hostName</b> indicates the host name.</li> <li>• <b>port</b> indicates the port number.</li> </ul>
<code>public DataStreamSink&lt;T&gt; writeUsingOutputformat(Outputformat&lt;T&gt; format)</code>	Write data to a file, for example, a binary file.

API	Description
public DataStreamSink<T> addSink(SinkFunction<T> sinkFunction)	Export data in a user-defined manner. The flink-connectors allows the addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction.

## Filtering and Mapping

**Table 2-26** APIs about filtering and mapping

API	Description
public <R> SingleOutputStreamOperator<R> map(MapFunction<T, R> mapper)	Transform an element into another element of the same type.
public <R> SingleOutputStreamOperator<R> flatMap(FlatMapFunction<T, R> flatMapMapper)	Transform an element into zero, one, or multiple elements.
public SingleOutputStreamOperator<T> filter(FilterFunction<T> filter)	Run a Boolean function on each element and retain elements that return <b>true</b> .

## Aggregation

**Table 2-27** APIs about aggregation

API	Description
public KeyedStream<T, Tuple> keyBy(int... fields)	Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is internally implemented using hash partition. A KeyedStream is returned.
public KeyedStream<T, Tuple> keyBy(String... fields)	Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby.
public <K> KeyedStream<T, K> keyBy(KeySelector<T, K> key)	<ul style="list-style-type: none"> <li>• <b>fields</b> indicates the numbers of columns or names of member variables.</li> <li>• <b>key</b> indicates the user-defined basis for partitioning.</li> </ul>

API	Description
public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> reducer)	Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.
public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> folder)	Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.
public SingleOutputStreamOperator<T> sum(int positionToSum)	Calculate the sum in a KeyedStream in a rolling manner.
public SingleOutputStreamOperator<T> sum(String field)	<b>positionToSum</b> and <b>field</b> indicate calculating the sum of a specific column.
public SingleOutputStreamOperator<T> min(int positionToMin)	Calculate the minimum value in a KeyedStream in a rolling manner. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values.
public SingleOutputStreamOperator<T> min(String field)	<b>positionToMin</b> and <b>field</b> indicate calculating the minimum value of a specific column.
public SingleOutputStreamOperator<T> max(int positionToMax)	Calculate the maximum value in KeyedStream in a rolling manner. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values.
public SingleOutputStreamOperator<T> max(String field)	<b>positionToMax</b> and <b>field</b> indicate calculating the maximum value of a specific column.
public SingleOutputStreamOperator<T> minBy(int positionToMinBy)	Obtain the row where the minimum value of a column locates in a KeyedStream. <b>minBy</b> returns all elements of that row.
public SingleOutputStreamOperator<T> minBy(String positionToMinBy)	
public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)	<ul style="list-style-type: none"><li>• <b>positionToMinBy</b> indicates the column on which the minBy operation is performed.</li><li>• <b>first</b> indicates whether to return the first or last minimum value.</li></ul>
public SingleOutputStreamOperator<T> minBy(String field, boolean first)	

API	Description
public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)	Obtain the row where the maximum value of a column locates in a KeyedStream. maxBy returns all elements of that row.
public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)	
public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first)	
public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	<ul style="list-style-type: none"><li>• <b>positionToMaxBy</b> indicates the column on which the maxBy operation is performed.</li><li>• <b>first</b> indicates whether to return the first or last maximum value.</li></ul>

## DataStream Distribution

**Table 2-28** APIs about DataStream distribution

API	Description
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, int field)	Use a user-defined partitioner to select target task for each element. <ul style="list-style-type: none"><li>• <b>partitioner</b> indicates the user-defined method for repartitioning.</li><li>• <b>field</b> indicates the input parameters of partitioner.</li><li>• <b>keySelector</b> indicates the user-defined input parameters of partitioner.</li></ul>
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, String field)	
public <K> DataStream<T> partitionCustom(Partitioner<K> partitioner, KeySelector<T, K> keySelector)	
public DataStream<T> shuffle()	Randomly and evenly partition elements.
public DataStream<T> rebalance()	Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.
public DataStream<T> rescale()	Distribute elements into downstream subset in round-robin manner.

API	Description
public DataStream<T> broadcast()	Broadcast each element to all partitions.

## Project Capabilities

**Table 2-29** APIs about projecting

API	Description
public <R extends Tuple> SingleOutputStreamOperator<R> project(int... fieldIndexes)	Select some field subset from the tuple. <b>fieldIndexes</b> indicates some sequences of the tuple. <b>NOTE</b> Only tuple data type is supported by the project API.

## Configuring the eventtime Attribute

**Table 2-30** APIs about configuring the eventtime attribute

API	Description
public SingleOutputStreamOperator<T> assignTimestampsAndWatermarks(AssignerWithPeriodicWatermarks<T> timestampAndWatermarkAssigner)	Extract timestamp from records, enabling event time window to trigger computing.
public SingleOutputStreamOperator<T> assignTimestampsAndWatermarks(AssignerWithPunctuatedWatermarks<T> timestampAndWatermarkAssigner)	

**Table 2-31** lists differences of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs.

**Table 2-31** Difference of AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks APIs

Parameter	Description
AssignerWithPeriodicWatermarks	Generate Watermark based on the getConfig().setAutoWatermarkInterval(200L) timestamp of StreamExecutionEnvironment class.
AssignerWithPunctuatedWatermarks	Generate a Watermark each time an element is received. Watermarks can be different based on received elements.

## Iteration

**Table 2-32** APIs about iteration

API	Description
public IterativeStream<T> iterate()	In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator. <b>NOTE</b> <ul style="list-style-type: none"><li>This API is helpful to algorithms that require constant update of models.</li><li>long maxWaitTimeMillis: The timeout period of each round of iteration.</li></ul>
public IterativeStream<T> iterate(long maxWaitTimeMillis)	

## Stream Splitting

**Table 2-33** APIs about stream splitting

API	Description
public SplitStream<T> split(OutputSelector<T> outputSelector)	Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created.
public DataStream<OUT> select(String... outputNames)	Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.

## Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

**Table 2-34** lists APIs for generating windows.

**Table 2-34** APIs for generating windows

API	Description
public <W extends Window> WindowedStream<T, KEY, W> window(WindowAssigner<? super T, W> assigner)	Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds).
public <W extends Window> AllWindowedStream<T, W> windowAll(WindowAssigner<? super T, W> assigner)	Define windows in DataStreams.
public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size)	Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"><li>• <b>size</b> indicates the duration of the window.</li><li>• <b>slide</b> indicates the sliding time of window.</li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• WindowedStream and AllWindowedStream indicates two types of streams.</li><li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li></ul>
public WindowedStream<T, KEY, TimeWindow> timeWindow(Time size, Time slide)	

API	Description
public AllWindowedStream<T, TimeWindow> timeWindowAll(Time size)	Define time windows in partitioned DataStream, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters.
public AllWindowedStream<T, TimeWindow> timeWindowAll(Time size, Time slide)	<ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>
public WindowedStream<T, KEY, GlobalWindow> countWindow(long size)	Divide windows according to the number of elements and define windows in partitioned KeyedStreams.
public WindowedStream<T, KEY, GlobalWindow> countWindow(long size, long slide)	<ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• WindowedStream and AllWindowedStream indicates two types of streams.</li> <li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li> </ul>
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size)	Divide windows according to the number of elements and define windows in DataStreams.
public AllWindowedStream<T, GlobalWindow> countWindowAll(Time size, Time slide)	<ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>

**Table 2-35** lists APIs for operating windows.**Table 2-35** APIs for operating windows

Method	API	Description
Window	public <R> SingleOutputStrea- mOperator<R> apply(WindowFunction<T, R, K, W> function)	<p>Apply a general function to a window. The data in the window is calculated as a whole.</p> <ul style="list-style-type: none"> <li>• <b>function</b> indicates the window function to be executed.</li> <li>• <b>resultType</b> indicates the type of returned data.</li> </ul>

Method	API	Description
	public <R> SingleOutputStreamOperator<R> apply(WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)	
	public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)	Apply a reduce function to the window and return the result. <ul style="list-style-type: none"><li>• <b>reduceFunction</b> indicates the reduce function to be executed.</li></ul>
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function)	<ul style="list-style-type: none"><li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a reduce operation.</li></ul>
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, WindowFunction<T, R, K, W> function, TypeInformation<R> resultType)	<ul style="list-style-type: none"><li>• <b>resultType</b> indicates the type of returned data.</li></ul>
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function)	Apply a fold function to the window and return the result. <ul style="list-style-type: none"><li>• <b>initialValue</b> indicates the initial value.</li></ul>
	public <R> SingleOutputStreamOperator<R> fold(R initialValue, FoldFunction<T, R> function, TypeInformation<R> resultType)	<ul style="list-style-type: none"><li>• <b>foldFunction</b> indicates the fold function.</li><li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a fold operation.</li></ul>
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function)	<ul style="list-style-type: none"><li>• <b>resultType</b> indicates the type of returned data.</li></ul>

Method	API	Description
	public <ACC, R> SingleOutputStreamOperator<R> fold(ACC initialValue, FoldFunction<T, ACC> foldFunction, WindowFunction<ACC, R, K, W> function, TypeInformation<ACC> foldAccumulatorType, TypeInformation<R> resultType)	
Window All	public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function)	Apply a general function to a window. The data in the window is calculated as a whole.
	public <R> SingleOutputStreamOperator<R> apply(AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)	
	public SingleOutputStreamOperator<T> reduce(ReduceFunction<T> function)	Apply a reduce function to the window and return the result. <ul style="list-style-type: none"> <li>• <b>reduceFunction</b> indicates the reduce function to be executed.</li> </ul>
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function)	<ul style="list-style-type: none"> <li>• <b>AllWindowFunction</b> indicates triggering an operation to the window after a reduce operation.</li> </ul>
	public <R> SingleOutputStreamOperator<R> reduce(ReduceFunction<T> reduceFunction, AllWindowFunction<T, R, W> function, TypeInformation<R> resultType)	<ul style="list-style-type: none"> <li>• <b>resultType</b> indicates the type of returned data.</li> </ul>

Method	API	Description
	<pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; fold(R initialValue, FoldFunction&lt;T, R&gt; function)</pre>	Apply a fold function to the window and return the result. <ul style="list-style-type: none"><li>• <b>initialValue</b> indicates the initial value.</li><li>• <b>foldFunction</b> indicates the fold function.</li><li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a fold operation.</li><li>• <b>resultType</b> indicates the type of returned data.</li></ul>
	<pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; fold(R initialValue, FoldFunction&lt;T, R&gt; function, TypeInformation&lt;R&gt; resultType)</pre>	
	<pre>public &lt;ACC, R&gt; SingleOutputStreamOperator&lt;R&gt; fold(ACC initialValue, FoldFunction&lt;T, ACC&gt; foldFunction, AllWindowFunction&lt;ACC, R, W&gt; function)</pre>	
	<pre>public &lt;ACC, R&gt; SingleOutputStreamOperator&lt;R&gt; fold(ACC initialValue, FoldFunction&lt;T, ACC&gt; foldFunction, AllWindowFunction&lt;ACC, R, W&gt; function, TypeInformation&lt;ACC&gt; foldAccumulatorType, TypeInformation&lt;R&gt; resultType)</pre>	
Window and Window All	<pre>public SingleOutputStreamOperator&lt;T&gt; sum(int positionToSum)</pre>	Sum a specified column of the window data. <b>field</b> and <b>positionToSum</b> indicate a specific column of the data.
	<pre>public SingleOutputStreamOperator&lt;T&gt; sum(String field)</pre>	
	<pre>public SingleOutputStreamOperator&lt;T&gt; min(int positionToMin)</pre>	Calculate the minimum value of a specified column of the window data. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values.
	<pre>public SingleOutputStreamOperator&lt;T&gt; min(String field)</pre>	<b>positionToMin</b> and <b>field</b> indicate calculating the minimum value of a specific column.

Method	API	Description
	public SingleOutputStreamOperator<T> minBy(int positionToMinBy)	Obtain the row where the minimum value of a column locates in the window data. <b>minBy</b> returns all elements of that row.
	public SingleOutputStreamOperator<T> minBy(String positionToMinBy)	<ul style="list-style-type: none"> <li>• <b>positionToMinBy</b> indicates the column on which the <b>minBy</b> operation is performed.</li> </ul>
	public SingleOutputStreamOperator<T> minBy(int positionToMinBy, boolean first)	<ul style="list-style-type: none"> <li>• <b>first</b> indicates whether to return the first or last minimum value.</li> </ul>
	public SingleOutputStreamOperator<T> minBy(String field, boolean first)	
	public SingleOutputStreamOperator<T> max(int positionToMax)	Calculate the maximum value of a specified column of the window data. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values.
	public SingleOutputStreamOperator<T> max(String field)	<b>positionToMax</b> and <b>field</b> indicate calculating the maximum value of a specific column.
	The default public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy)//true	Obtain the row where the maximum value of a column locates in the window data. <b>maxBy</b> returns all elements of that row.
	The default public SingleOutputStreamOperator<T> maxBy(String positionToMaxBy)//true	<ul style="list-style-type: none"> <li>• <b>positionToMaxBy</b> indicates the column on which the <b>maxBy</b> operation is performed.</li> </ul>
	public SingleOutputStreamOperator<T> maxBy(int positionToMaxBy, boolean first)	<ul style="list-style-type: none"> <li>• <b>first</b> indicates whether to return the first or last maximum value.</li> </ul>
	public SingleOutputStreamOperator<T> maxBy(String field, boolean first)	

## Combining Multiple DataStreams

**Table 2-36** APIs about combining multiple DataStreams

API	Description
<pre>public final DataStream&lt;T&gt; union(DataStream&lt;T&gt;.. . streams)</pre>	<p>Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams.</p> <p><b>NOTE</b> If you perform union operation on a piece of data with itself, there are two copies of the same data.</p>
<pre>public &lt;R&gt; ConnectedStreams&lt;T, R&gt; connect(DataStream&lt;R &gt; dataStream)</pre>	Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.
<pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; map(CoMapFunction&lt;I N1, IN2, R&gt; coMapper)</pre>	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.
<pre>public &lt;R&gt; SingleOutputStreamOperator&lt;R&gt; flatMap(CoFlatMapFun ction&lt;IN1, IN2, R&gt; coFlatMapper)</pre>	Perform mapping operation, which is similar to flatMap operation in DataStream, on elements.

## Join Operation

**Table 2-37** APIs about join operation

API	Description
<pre>public &lt;T2&gt; JoinedStreams&lt;T, T2&gt; join(DataStream&lt;T2&gt; otherStream)</pre>	Join two DataStreams using a given key in a specified window.
<pre>public &lt;T2&gt; CoGroupedStreams&lt;T, T2&gt; coGroup(DataStream&lt; T2&gt; otherStream)</pre>	Co-group two DataStreams using a given key in a specified window.

### 2.4.5.1.2 Scala

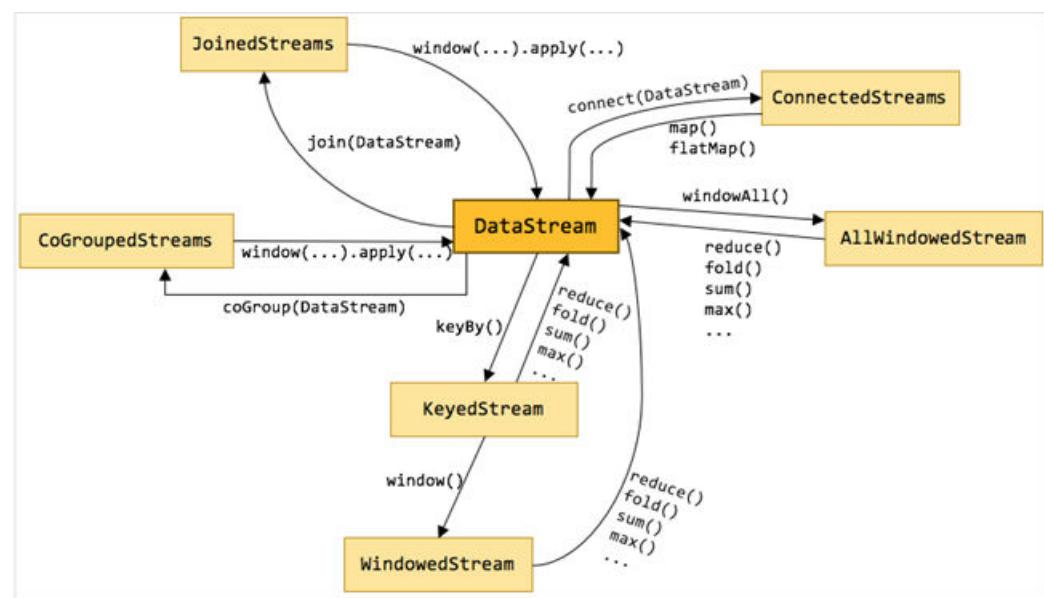
To avoid API compatibility or reliability issues after updates to the open-source Flink, recommended that APIs of the matching version are recommended. For details about APIs, see MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)

## Common APIs of Flink

Flink mainly uses the following APIs:

- StreamExecutionEnvironment: provides the execution environment, which is the basis of Flink stream processing.
- DataStream: represents streaming data. DataStream can be regarded as unmodifiable collection that contains duplicate data. The number of elements in DataStream are unlimited.
- KeyedStream: generates the stream by performing keyBy grouping operations on DataStream. Data is grouped based on the specified key value.
- WindowedStream: generates the stream by performing the window function on KeyedStream, sets the window type, and defines window triggering conditions.
- AllWindowedStream: generates streams by performing the window function on DataStream, sets the window type, defines window triggering conditions, and performs operations on the window data.
- ConnectedStreams: generates streams by connecting the two DataStreams and retaining the original data type, and performs the map or flatMap operation.
- JoinedStreams: performs equijoin operation to data in the window. The join operation is a special scenario of coGroup operation.
- CoGroupedStreams: performs the coGroup operation on data in the window. CoGroupedStreams can perform various types of join operations.

Figure 2-96 Conversion of Flink stream types



## Data Stream Source

**Table 2-38** APIs about data stream source

API	Description
def fromElements[T: TypeInformation] (data: T*): DataStream[T]	Obtain user-defined data of multiple elements as the data stream source. <b>data</b> is the specific data of multiple elements.
def fromCollection[T: TypeInformation] (data: Seq[T]): DataStream[T]	Obtain the user-defined data collection as input data stream.
def fromCollection[T: TypeInformation] (data: Iterator[T]): DataStream[T]	<b>data</b> can be a data collection or a data body that can be iterated.
def fromParallelCollection[T: TypeInformation] (data: SplittableIterator[T]): DataStream[T]	Obtain the user-defined data collection as parallel data stream source. <b>data</b> indicates the iterator that can be divided into multiple partitions.
def generateSequence(from: Long, to: Long): DataStream[Long]	Obtain a sequence of user-defined data as the data stream source. <ul style="list-style-type: none"><li>• <b>from</b> indicates the starting point of numbers.</li><li>• <b>to</b> indicates the end point of numbers.</li></ul>
def readTextFile(filePath: String): DataStream[String]	Obtain the user-defined text file from a specific path as the data stream source.
def readTextFile(filePath: String, charsetName: String): DataStream[String]	<ul style="list-style-type: none"><li>• <b>filePath</b> indicates the path of the text file.</li><li>• <b>charsetName</b> indicates the encoding format.</li></ul>
def readFile[T: TypeInformation] (inputFormat: FileInputFormat[T], filePath: String)	Obtain the user-defined file from a specific path as the data stream source.
def readFile[T: TypeInformation] (inputFormat: FileInputFormat[T], filePath: String, watchType: FileProcessingMode, interval: Long): DataStream[T]	<ul style="list-style-type: none"><li>• <b>filePath</b> indicates the file path.</li><li>• <b>inputFormat</b> indicates the format of the file.</li><li>• <b>watchType</b> indicates the file processing mode, which can be <b>PROCESS_ONCE</b> or <b>PROCESS_CONTINUOUSLY</b>.</li><li>• <b>interval</b> indicates the interval for processing directories or files.</li></ul>

API	Description
<pre>def socketTextStream(hostname: String, port: Int, delimiter: Char = '\n', maxRetry: Long = 0): DataStream[String]</pre>	<p>Obtain user-defined socket data as the data stream source.</p> <ul style="list-style-type: none"><li>• <b>hostname</b> indicates the host name of the socket server.</li><li>• <b>port</b> indicates the listening port of the server.</li><li>• <b>delimiter</b> and <b>maxRetry</b> are not supported by Scala APIs.</li></ul>
<pre>def addSource[T: TypeInformation](function: SourceFunction[T]): DataStream[T]</pre>	<p>Customize the <b>SourceFunction</b> and <b>addSource</b> methods to add data sources such as Kafka. The implementation method is the run method of SourceFunction.</p> <ul style="list-style-type: none"><li>• <b>function</b> indicates the user-defined <b>SourceFunction</b> function.</li><li>• Simplified format is supported by Scala.</li></ul>
<pre>def addSource[T: TypeInformation](function: SourceContext[T] =&gt; Unit): DataStream[T]</pre>	

## Data Output

Table 2-39 APIs about data output

API	Description
<pre>def print(): DataStreamSink[T]</pre>	Print data as the standard output stream.
<pre>def printToErr()</pre>	Print data as the standard error output stream.
<pre>def writeAsText(path: String): DataStreamSink[T]</pre>	Write data to a specific text file.
<pre>def writeAsText(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]</pre>	<ul style="list-style-type: none"><li>• <b>path</b> indicates the path of the text file.</li><li>• <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b>.</li></ul>

API	Description
def writeAsCsv(path: String): DataStreamSink[T]	Writhe data to a specific .csv file. • <b>path</b> indicates the path of the text file.
def writeAsCsv(path: String, writeMode: FileSystem.WriteMode): DataStreamSink[T]	• <b>writeMode</b> indicates the writing mode, which can be <b>OVERWRITE</b> or <b>NO_OVERWRITE</b> .
def writeAsCsv(path: String, writeMode: FileSystem.WriteMode, rowDelimiter: String, fieldDelimiter: String): DataStreamSink[T]	• <b>rowDelimiter</b> indicates the row separator. • <b>fieldDelimiter</b> indicates the column separator.
def writeUsingOutputFormat(format: OutputFormat[T]): DataStreamSink[T]	Write data to a file, for example, a binary file.
def writeToSocket(hostname: String, port: Integer, schema: SerializationSchema[T]): DataStreamSink[T]	Write data to the socket connection. • <b>hostName</b> indicates the host name. • <b>port</b> indicates the port number.
def addSink(sinkFunction: SinkFunction[T]): DataStreamSink[T]	Export data in a user-defined manner. The flink-connectors allows addSink method to support exporting data to Kafka. The implementation method is the invoke method of SinkFunction.
def addSink(fun: T => Unit): DataStreamSink[T]	

## Filtering and Mapping

Table 2-40 APIs about filtering and mapping

API	Description
def map[R: TypeInformation](fun: T => R): DataStream[R]	Transform an element into another element of the same type.
def map[R: TypeInformation](mapper: MapFunction[T, R]): DataStream[R]	
def flatMap[R: TypeInformation](flatMapMapper: FlatMapFunction[T, R]): DataStream[R]	Transform an element into zero, one, or multiple elements.
def flatMap[R: TypeInformation](fun: (T, Collector[R]) => Unit): DataStream[R]	
def flatMap[R: TypeInformation](fun: T => TraversableOnce[R]): DataStream[R]	

API	Description
def filter(filter: FilterFunction[T]): DataStream[T]	Run a Boolean function on each element and retain elements that return <b>true</b> .
def filter(fun: T => Boolean): DataStream[T]	

## Aggregation

**Table 2-41** APIs about aggregation

API	Description
def keyBy(fields: Int*): KeyedStream[T, JavaTuple]	Logically divide a stream into multiple partitions, each containing elements with the same key. The partitioning is internally implemented using hash partition. A KeyedStream is returned.
def keyBy(firstField: String, otherFields: String*): KeyedStream[T, JavaTuple]	
def keyBy[K: TypeInformation](fun: T => K): KeyedStream[T, K]	Return the KeyedStream after the KeyBy operation, and call the KeyedStream function, such as reduce, fold, min, minby, max, maxby, sum, and sumby. <ul style="list-style-type: none"><li>• <b>fields</b> indicates the IDs of certain columns</li><li>• <b>firstField</b> and <b>otherFields</b> are names of member variables.</li><li>• <b>key</b> indicates the user-defined basis for partitioning.</li></ul>
def reduce(fun: (T, T) => T): DataStream[T]	Perform reduce on KeyedStream in a rolling manner. Aggregate the current element and the last reduced value into a new value. The types of the three values are the same.
def reduce(reducer: ReduceFunction[T]): DataStream[T]	
def fold[R: TypeInformation] (initialValue: R)(fun: (R,T) => R): DataStream[R]	Perform fold operation on KeyedStream based on an initial value in a rolling manner. Aggregate the current element and the last folded value into a new value. The input and returned values of Fold can be different.
def fold[R: TypeInformation] (initialValue: R, folder: FoldFunction[T,R]): DataStream[R]	
def sum(position: Int): DataStream[T]	Calculate the sum in a KeyedStream in a rolling manner.
def sum(field: String): DataStream[T]	<b>position</b> and <b>field</b> indicate calculating the sum of a specific column.

API	Description
def min(position: Int): DataStream[T]	Calculate the minimum value in a KeyedStream in a rolling manner. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values.
def min(field: String): DataStream[T]	<b>position and field</b> indicate calculating the minimum value of a specific column.
def max(position: Int): DataStream[T]	Calculate the maximum value in KeyedStream in a rolling manner. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values.
def max(field: String): DataStream[T]	<b>position and field</b> indicate calculating the maximum value of a specific column.
def minBy(position: Int): DataStream[T]	Obtain the row where the minimum value of a column locates in a KeyedStream. <b>minBy</b> returns all elements of that row.
def minBy(field: String): DataStream[T]	<b>position and field</b> indicate the column on which the <b>minBy</b> operation is performed.
def maxBy(position: Int): DataStream[T]	Obtain the row where the maximum value of a column locates in a KeyedStream. <b>maxBy</b> returns all elements of that row.
def maxBy(field: String): DataStream[T]	<b>position and field</b> indicate the column on which the <b>maxBy</b> operation is performed.

## DataStream Distribution

**Table 2-42** APIs about DataStream distribution

API	Description
<code>def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], field: Int) : DataStream[T]</code>	Use a user-defined partitioner to select target task for each element. <ul style="list-style-type: none"><li>• <b>partitioner</b> indicates the user-defined method for repartitioning.</li><li>• <b>field</b> indicates the input parameters of partitioner.</li><li>• <b>keySelector</b> indicates the user-defined input parameters of partitioner.</li></ul>
<code>def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], field: String):DataStream[T]</code>	
<code>def partitionCustom[K: TypeInformation](partitioner: Partitioner[K], fun: T =&gt; K): DataStream[T]</code>	
<code>def shuffle: DataStream[T]</code>	Randomly and evenly partition elements.
<code>def rebalance: DataStream[T]</code>	Partition elements in round-robin manner, ensuring load balance of each partition. This partitioning is helpful to data with skewness.
<code>def rescale: DataStream[T]</code>	Distribute elements into downstream subset in round-robin manner. <b>NOTE</b> The method for checking the code is similar to the rebalance method.
<code>def broadcast: DataStream[T]</code>	Broadcast each element to all partitions.

## Configuring the eventtime Attribute

**Table 2-43** APIs about configuring the eventtime attribute

API	Description
<code>def assignTimestampsAndWatermarks(assigner: AssignerWithPeriodicWatermarks[T]): DataStream[T]</code>	Extract timestamp from records, enabling event time window to trigger computing.

API	Description
<code>def assignTimestampsAndWatermarks(assigner: AssignerWithPunctuatedWatermarks[T]): DataStream[T]</code>	

## Iteration

**Table 2-44** APIs about iteration

API	Description
<code>def iterate[R] (stepFunction: DataStream[T] =&gt; (DataStream[T], DataStream[R]),maxWaitTimeMillis:Long = 0,keepPartitioning: Boolean = false) : DataStream[R]</code>	<p>In the flow, create in a feedback loop to redirect the output of an operator to a preceding operator.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"> <li>• This API is helpful to algorithms that require constant update of models.</li> <li>• long maxWaitTimeMillis: The timeout period of each round of iteration.</li> </ul>
<code>def iterate[R, F: TypeInformation] (stepFunction: ConnectedStreams[T, F] =&gt; (DataStream[F], DataStream[R]),maxWaitTimeMillis:Long): DataStream[R]</code>	

## Stream Splitting

**Table 2-45** APIs about stream splitting

API	Description
<code>def split(selector: OutputSelector[T]): SplitStream[T]</code>	Use OutputSelector to rewrite select method, specify stream splitting basis (by tagging), and construct SplitStream streams. That is, a string is tagged for each element, so that a stream of a specific tag can be selected or created.
<code>def select(outputNames: String*): DataStream[T]</code>	Select one or multiple streams from a SplitStream. outputNames indicates the sequence of string tags created for each element using the split method.

## Window

Windows can be classified as tumbling windows and sliding windows.

- APIs such as Window, TimeWindow, CountWindow, WindowAll, TimeWindowAll, and CountWindowAll can be used to generate windows.
- APIs such as Window Apply, Window Reduce, Window Fold, and Aggregations on windows can be used to operate windows.
- Multiple Window Assigners are supported, including TumblingEventTimeWindows, TumblingProcessingTimeWindows, SlidingEventTimeWindows, SlidingProcessingTimeWindows, EventTimeSessionWindows, ProcessingTimeSessionWindows, and GlobalWindows.
- ProcessingTime, EventTime, and IngestionTime are supported.
- Timestamp modes EventTime: AssignerWithPeriodicWatermarks and AssignerWithPunctuatedWatermarks are supported.

**Table9 APIs for generating windows** lists APIs for generating windows.

**Table 2-46** APIs for generating windows

API	Description
<code>def window[W &lt;: Window](assigner: WindowAssigner[_ &gt;: T, W]): WindowedStream[T, K, W]</code>	Define windows in partitioned KeyedStreams. A window groups each key according to some characteristics (for example, data received within the latest 5 seconds. For details about windows, see: <a href="https://ci.apache.org/projects/flink/flink-docs-release-1.10/dev/event_time.html">https://ci.apache.org/projects/flink/flink-docs-release-1.10/dev/event_time.html</a> .
<code>def windowAll[W &lt;: Window](assigner: WindowAssigner[_ &gt;: T, W]): AllWindowedStream[T, W]</code>	Define windows in DataStreams.
<code>def timeWindow(size: time</code> <code>WindowedStream[T, K, TimeWindow]</code>	Define time windows in partitioned KeyedStreams, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"><li>• <b>size</b> indicates the duration of the window.</li><li>• <b>slide</b> indicates the sliding time of window.</li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• WindowedStream and AllWindowedStream indicates two types of streams.</li><li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li></ul>
<code>def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]</code>	

API	Description
def timeWindowAll(size: time AllWindowedStream[T, TimeWindow]	Define time windows in partitioned DataStream, select ProcessingTime or EventTime based on the environment.getStreamTimeCharacteristic() parameter, and determine whether the window is TumblingWindow or SlidingWindow depends on the number of parameters. <ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>
def countWindow(size: Long, slide: Long): WindowedStream[T, K, GlobalWindow]	Divide windows according to the number of elements and define windows in partitioned KeyedStreams. <ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>
def countWindow(size: Long): WindowedStream[T, K, GlobalWindow]	<b>NOTE</b> <ul style="list-style-type: none"> <li>• WindowedStream and AllWindowedStream indicates two types of streams.</li> <li>• If the API contains only one parameter, the window is TumblingWindow. If the API contains two or more parameters, the window is SlidingWindow.</li> </ul>
def countWindowAll(size: Long, slide: Long): AllWindowedStream[T, GlobalWindow]	Divide windows according to the number of elements and define windows in DataStreams. <ul style="list-style-type: none"> <li>• <b>size</b> indicates the duration of the window.</li> <li>• <b>slide</b> indicates the sliding time of window.</li> </ul>
def countWindowAll(size: Long): AllWindowedStream[T, GlobalWindow]	

**Table 2-47** lists APIs for operating windows.

**Table 2-47** APIs for operating windows

Method	API	Description
Window	def apply[R: TypeInformation] (function: WindowFunction[T, R, K, W]): DataStream[R]	Apply a general function to a window. The data in the window is calculated as a whole. <b>function</b> indicates the window function to be executed.
	def apply[R: TypeInformation] (function: (K, W, Iterable[T], Collector[R]) => Unit): DataStream[R]	

Method	API	Description
	<pre>def reduce(function: ReduceFunction[T]): DataStream[T]</pre>	Apply a reduce function to the window and return the result.
	<pre>def reduce(function: (T, T) =&gt; T): DataStream[T]</pre>	<ul style="list-style-type: none"> <li>• <b>reduceFunction</b> indicates the reduce function to be executed.</li> </ul>
	<pre>def reduce[R: TypeInformation] (preAggregator: ReduceFunction[T], function: WindowFunction[T, R, K, W]): DataStream[R]</pre>	<ul style="list-style-type: none"> <li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a reduce operation.</li> </ul>
	<pre>def reduce[R: TypeInformation] (preAggregator: (T, T) =&gt; T, windowFunction: (K, W, Iterable[T], Collector[R]) =&gt; Unit): DataStream[R]</pre>	
	<pre>def fold[R: TypeInformation] (initialValue: R, function: FoldFunction[T,R]): DataStream[R]</pre>	Apply a fold function to the window and return the result.
	<pre>def fold[R: TypeInformation] (initialValue: R)(function: (R, T) =&gt; R): DataStream[R]</pre>	<ul style="list-style-type: none"> <li>• <b>initialValue</b> indicates the initial value.</li> <li>• <b>foldFunction</b> indicates the fold function.</li> </ul>
	<pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: FoldFunction[T, ACC], function: WindowFunction[ACC, R, K, W]): DataStream[R]</pre>	<ul style="list-style-type: none"> <li>• <b>function of WindowFunction</b> indicates triggering an operation to the window after a fold operation.</li> </ul>
	<pre>def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, foldFunction: (ACC, T) =&gt; ACC, windowFunction: (K, W, Iterable[ACC], Collector[R]) =&gt; Unit): DataStream[R]</pre>	
Window All	<pre>def apply[R: TypeInformation] (function: AllWindowFunction[T, R, W]): DataStream[R]</pre>	Apply a general function to a window. The data in the window is calculated as a whole.

Method	API	Description
	def apply[R: TypeInformation] (function: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]	
	def reduce(function: ReduceFunction[T]): DataStream[T]	Apply a reduce function to the window and return the result. <ul style="list-style-type: none"> <li>• <b>reduceFunction</b> indicates the reduce function to be executed.</li> </ul>
	def reduce(function: (T, T) => T): DataStream[T]	
	def reduce[R: TypeInformation] (preAggregator: ReduceFunction[T], windowFunction: AllWindowFunction[T, R, W]): DataStream[R]	
	def reduce[R: TypeInformation] (preAggregator: (T, T) => T, windowFunction: (W, Iterable[T], Collector[R]) => Unit): DataStream[R]	
	def fold[R: TypeInformation] (initialValue: R, function: FoldFunction[T,R]): DataStream[R]	Apply a fold function to the window and return the result. <ul style="list-style-type: none"> <li>• <b>initialValue</b> indicates the initial value.</li> </ul>
	def fold[R: TypeInformation] (initialValue: R)(function: (R, T) => R): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: FoldFunction[T, ACC], windowFunction: AllWindowFunction[ACC, R, W]): DataStream[R]	
	def fold[ACC: TypeInformation, R: TypeInformation](initialValue: ACC, preAggregator: (ACC, T) => ACC, windowFunction: (W, Iterable[ACC], Collector[R]) => Unit): DataStream[R]	

Method	API	Description
Window and Window All	def sum(position: Int): DataStream[T]	Sum a specified column of the window data. <b>field</b> and <b>position</b> indicate a specific column of the data.
	def sum(field: String): DataStream[T]	
	def min(position: Int): DataStream[T]	Calculate the minimum value of a specified column of the window data. <b>min</b> returns the minimum value, without guarantee of the correctness of columns of non-minimum values.
	def min(field: String): DataStream[T]	<b>position</b> and <b>field</b> indicate calculating the minimum value of a specific column.
	def max(position: Int): DataStream[T]	Calculate the maximum value of a specified column of the window data. <b>max</b> returns the maximum value, without guarantee of the correctness of columns of non-maximum values.
	def max(field: String): DataStream[T]	<b>position</b> and <b>field</b> indicate calculating the maximum value of a specific column.
	def minBy(position: Int): DataStream[T]	Obtain the row where the minimum value of a column locates in the window data. <b>minBy</b> returns all elements of that row.
	def minBy(field: String): DataStream[T]	<b>position</b> and <b>field</b> indicate the column on which the <b>minBy</b> operation is performed.
	def maxBy(position: Int): DataStream[T]	Obtain the row where the maximum value of a column locates in the window data. <b>maxBy</b> returns all elements of that row.
	def maxBy(field: String): DataStream[T]	<b>position</b> and <b>field</b> indicate the column on which the <b>maxBy</b> operation is performed.

## Combining Multiple DataStreams

**Table 2-48** APIs about combining multiple DataStreams

API	Description
<code>def union(dataStreams: DataStream[T]*): DataStream[T]</code>	Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams. <b>NOTE</b> If you perform union operation on a piece of data with itself, there are two copies of the same data.
<code>def connect[T2] (dataStream: DataStream[T2]): ConnectedStreams[T, T2]</code>	Connect two DataStreams and retain the original type. The connect API method allows two DataStreams to share statuses. After ConnectedStreams is generated, map or flatMap operation can be called.
<code>def map[R: TypeInformation] (coMapper: CoMapFunction[IN1, IN2, R]): DataStream[R]</code>	Perform mapping operation, which is similar to map and flatMap operation in a ConnectedStreams, on elements. After the operation, the type of the new DataStreams is string.
<code>def map[R: TypeInformation](fun1: IN1 =&gt; R, fun2: IN2 =&gt; R): DataStream[R]</code>	
<code>def flatMap[R: TypeInformation] (coFlatMapper: CoFlatMapFunction[IN1, IN2, R]): DataStream[R]</code>	Perform union operation on multiple DataStreams to generate a new data stream containing all data from original DataStreams. <b>NOTE</b> If you perform union operation on a piece of data with itself, there are two copies of the same data.
<code>def flatMap[R: TypeInformation](fun1: (IN1, Collector[R]) =&gt; Unit, fun2: (IN2, Collector[R]) =&gt; Unit): DataStream[R]</code>	
<code>def flatMap[R: TypeInformation] (fun1: IN1 =&gt; TraversableOnce[R], fun2: IN2 =&gt; TraversableOnce[R]): DataStream[R]</code>	

## Join Operation

**Table 2-49** APIs about join operation

API	Description
<code>def join[T2] (otherStream: DataStream[T2]): JoinedStreams[T, T2]</code>	Join two DataStreams using a given key in a specified window. The key value of the join operation is specified by the <b>where</b> and <b>equalTo</b> method, indicating filtering data with equivalent conditions from two DataStreams.
<code>def coGroup[T2] (otherStream: DataStream[T2]): CoGroupedStreams[T, T2]</code>	Co-group two DataStreams using a given key in a specified window. The key value of the coGroup operation is specified by the where and equalTo method, indicating partitioning two DataStreams using equivalent conditions.

### 2.4.5.2 Overview of RESTful APIs

Flink has a monitoring API that can be used to query status and statistics of running jobs, as well as recent completed jobs. This monitoring API is used by Apache Flink Dashboard.

The monitoring API is a RESTful API that accepts HTTP GET requests and responds with JSON data. RESTful API is a set of APIs used to log in to the web server. In Flink, web server is a module of JobManager and shares the same process with JobManager. By default, the listening port of web server is 8081. If you want to change the listening port, modify **jobmanager.web.port** in the **flink-conf.yaml** file.

The *Netty* and the *Netty Router* library are used to handle REST requests and analysis URLs.

RESTful APIs are executed through HTTP requests.

The format of the HTTP requests is `http://<JobManager_IP>:<JobManager_Port><Path>`

The *JobManager\_IP* indicates the IP address of JobManager, *JobManager\_Port* indicates the listening port of JobManager, and *Path* indicates the path. For details, see [Table 2-50](#). For example, `http://10.162.181.57:32261/config`.

#### NOTE

If you want to modify the configuration file **flink-conf.yaml** of the Flink Client, add to-be-visited IP addresses (separated with commas) in **jobmanager.web.allow-access-address** and **jobmanager.web.access-control-allow-origin** parameters.

[Table 2-50](#) lists all RESTful API paths supported by Flink.

**Table 2-50** Paths supported by Flink

Path	Description
/config	Some information about the monitoring API and the server setup.
/logout	Some information about the logout.
/overview	Simple summary of the Flink cluster status.
/jobs	IDs of the jobs, grouped by status <i>running</i> , <i>finished</i> , <i>failed</i> , <i>canceled</i> .
/jobmanager/config	the configuration of the jobmanager.
/joboverview	Jobs, grouped by status, each with a small summary of its status.
/joboverview/running	Jobs, grouped by status, each with a small summary of its status. The same as <a href="#">/joboverview</a> , but containing only currently running jobs.
/joboverview/completed	Jobs, grouped by status, each with a small summary of its status. The same as <a href="#">/joboverview</a> , but containing only completed (finished, canceled, or failed) jobs.
/jobs/<jobid>	Summary of one job, listing dataflow plan, status, timestamps of state transitions, aggregate information for each vertex (operator).
/jobs/<jobid>/vertices	Currently the same as <a href="#">/jobs/&lt;jobid&gt;</a> .
/jobs/<jobid>/config	The user-defined execution config used by the job.
/jobs/<jobid>/exceptions	The non-recoverable exceptions that have been observed by the job. The truncated flag defines whether more exceptions occurred, but are not listed, because the response would otherwise get too big.
/jobs/<jobid>/accumulators	The aggregated user accumulators plus job accumulators.
/jobs/<jobid>/checkpoints	checkpoint stats for a job.
/jobs/<jobid>/metrics	a job a list of all available metrics.
/jobs/<jobid>/vertices/<vertexid>	Information about one specific vertex, with a summary for each of its subtasks.

Path	Description
/jobs/<jobid>/vertices/<vertexid>/subtasktimes	This request returns the timestamps for the state transitions of all subtasks of a given vertex. These can be used, for example, to create time-line comparisons between subtasks.
/jobs/<jobid>/vertices/<vertexid>/taskmanagers	TaskManager statistics for one specific vertex. This is an aggregation of subtask statistics returned by <code>/jobs/&lt;jobid&gt;/vertices/&lt;vertexid&gt;</code> .
/jobs/<jobid>/vertices/<vertexid>/accumulators	The aggregated user-defined accumulators, for a specific vertex.
/jobs/<jobid>/vertices/<vertexid>/checkpoints	checkpoint stats for a single job vertex.
/jobs/<jobid>/vertices/<vertexid>/backpressure	back pressure stats for a single job vertex and all its sub tasks.
/jobs/<jobid>/vertices/<vertexid>/metrics	a given task of the values for a set of metrics.
/jobs/<jobid>/vertices/<vertexid>/subtasks/accumulators	Gets all user-defined accumulators for all subtasks of a given vertex. These are the individual accumulators that are returned in aggregated form by the request <code>/jobs/&lt;jobid&gt;/vertices/&lt;vertexid&gt;/accumulators</code> .
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>	Summary of the current or latest execution attempt of a specific subtask. See below for a sample.
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>	Summary of a specific execution attempt of a specific subtask. Multiple execution attempts happen in case of failure/recovery.
/jobs/<jobid>/vertices/<vertexid>/subtasks/<subtasknum>/attempts/<attempt>/accumulators	The accumulators collected for one specific subtask during one specific execution attempt (multiple attempts happen in case of failure/recovery).
/jobs/<jobid>/plan	The dataflow plan of a job. The plan is also included in the job summary ( <code>/jobs/&lt;jobid&gt;</code> ).
/taskmanagers	Some information of the TaskManagers.
/taskmanagers/<taskmanagerid>/metrics	the metrics information of a TaskManager.

Path	Description
/taskmanagers/<taskmanagerid>/log	the log information of a TaskManager.
/taskmanagers/<taskmanagerid>/stdout	the stdout of a TaskManager.
/jobmanager/log	the log of the jobmanager.
/jobmanager/stdout	the stdout of the jobmanager.
/jobmanager/metrics	the metrics of the jobmanager.
/*	services requests to web frontend's static files, such as HTML, CSS, or JS files.

**Table 2-51** describes variables listed in **Table 2-50**.

**Table 2-51** Description of variables

Variable	Description
jobid	ID of jobs
vertexid	Vertices ID of the flow diagram.
subtasknum	Sum of subtasks.
attempt	Times of attempts
taskmanagerid	ID of TaskManager

### 2.4.5.3 Overview of Savepoints CLI

#### Overview

Savepoints are externally stored checkpoints that you can use to stop-and-resume or update your Flink programs. They use Flink's checkpoint mechanism to create a snapshot of the state of your streaming program and write the checkpoint meta data out to an external file system.

It is highly recommended that you adjust your programs as described in this section in order to be able to upgrade your programs in the future. The main required change is to manually specify operator IDs via the **uid(String)** method. These IDs are used to scope the state of each operator.

```
DataStream<String> stream = env
//Statefulesource(e.g.Kafka)withID
.addSource(new StatefulSource())
.uid("source-id") //IDforthesourceoperator
.shuffle()
//StatefulemapperwithID
.map(new StatefulMapper())
```

```
.uid("mapper-id") //IDforthemapper  
//Statelessprintingsink  
.print(); //Auto-generatedID
```

## Savepoint Recovery

If you do not specify the IDs manually, they will be generated automatically. You can automatically restore from the savepoint if these IDs do not change. The generated IDs depend on the structure of your program and are sensitive to program changes. Therefore, it is highly recommended to assign these IDs manually. When a savepoint is triggered, a single savepoint file will be created containing the checkpoint metadata. The actual checkpoint state will be kept around in the configured checkpoint directory, for example, with a FsStateBackend or RocksDBStateBackend:

1. Trigger a savepoint.

```
$ bin/flink savepoint jobId [targetDirectory]
```

This command will trigger a savepoint for the job with ID:*jobid*. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager. The targetDirectory is optional. If targetDirectory is not configured, the directory specified by **state.savepoints.dir** in the configuration file is used to store savepoint.

You can configure a default savepoint target directory via the **state.savepoints.dir** key in the **flink-conf.yaml** file.

```
# Default savepoint target directory
```

### NOTE

You are advised to configure targetDirectory to an HDFS path.

For example:

```
bin/flink savepoint 405af8c02cf6dc069a0f9b7a1f7be088 hdfs://savepoint.
```

2. Cancel a job with a savepoint.

```
$ bin/flink cancel -s [targetDirectory] jobId
```

This will atomically trigger a savepoint for the job with ID:*jobid* and cancel the job. Furthermore, you can specify a target file system directory to store the savepoint. The directory must be accessible by JobManager.

3. Resume jobs.

- Resume from a savepoint.

```
$ bin/flink run -s savepointPath [runArgs]
```

This command submits a job and specifies the savepoint path. The execution will resume from the respective savepoint state.

### NOTE

**runArgs** is a user-defined parameter with parameter format and name varying depending on users.

- Allow non-restored state.

```
$ bin/flink run -s savepointPath -n [runArgs]
```

By default the resume operation will try to map all state of the savepoint back to the program you are restoring with. If you dropped an operator, you can skip the state that cannot be mapped to the new program via -allowNonRestoredState (short: -n).

4. Dispose savepoints.

```
$ bin/flink savepoint -d savepointPath
```

This command disposes the savepoint stored in: *savepointPath*.

## Precautions

- Chained operators are identified by the ID of the first task. It is not possible to manually assign an ID to an intermediate chained task, for example, in the chain [ a -> b -> c ] only **a** can have its ID assigned manually, but not **b** or **c**. To work around this, you can manually define the task chains. To manually define chains, use the **disableChaining()** interface. See the following example:

```
env.addSource(new GetDataSource())
.keyBy(0)
.timeWindow(Time.seconds(2)).uid("window-id")
.reduce(_+_).uid("reduce-id")
.map(f->(f1)).disableChaining().uid("map-id")
.print().disableChaining().uid("print-id")
```
- During job upgrade, the data type of operators cannot be changed.

### 2.4.5.4 Introduction to Flink Client CLI

#### Common CLIs

Common Flink CLIs are as follows:

- yarn-session.sh**
  - You can run **yarn-session.sh** to start a standing Flink cluster to receive tasks submitted by clients. Run the following command to start a Flink cluster with three TaskManager instances:

```
bin/yarn-session.sh
```
  - Run the following command to obtain other parameters of **yarn-session.sh**:

```
bin/yarn-session.sh -help
```
- flink**
  - You can run Flink commands to submit a Flink job to a standing Flink cluster or to execute the job in single-server mode.
    - Run the following command to submit a Flink job to a standing Flink cluster:

```
bin/flink run ..//examples/streaming/WindowJoin.jar
```

#### NOTE

Before using the command to submit a task, you need to run **yarn-session.sh** to start the Flink cluster.

- Run the following command to execute a per-job YARN Cluster mode:

```
bin/flink run -m yarn-cluster ..//examples/streaming/WindowJoin.jar
```

#### NOTE

The **-m** **yarn-cluster** parameter is used to specify a job to independently start a Flink cluster.

- List scheduled and running jobs (including their JobIDs):

```
bin/flink list
```

- Cancel a job:  
`bin/flink cancel <jobID>`
- Stop a job (streaming jobs only):  
`bin/flink stop <jobID>`

#### NOTE

The difference between cancelling and stopping a (streaming) job is the following:

- Cancel a job: On a cancel call, the operators in a job immediately receive a cancel() method call to cancel them as soon as possible. If operators are not stopping after the cancel call, Flink will start interrupting the thread periodically until it stops.
- Stop a job: Stop is only available for jobs which use sources that implement the `StoppableFunction` interface. The job will keep running until all sources properly shut down. Stop a job is more graceful than cancel, but it may cause the job to stop failing.
- Run the following command to obtain other parameters of Flink commands:  
`bin/flink --help`

## Precautions

- If `yarn-session.sh` uses `-z` to configure the specified ZooKeeper NameSpace, you need to use `-yid` to specify the applicationID and use `-yz` to specify the ZooKeeper NameSpace when using `flink run`. The NameSpaces must the same.

Example:

```
bin/yarn-session.sh -z YARN101  
bin/flink run -yid application_****_*** -yz YARN101 examples/streaming/WindowJoin.jar
```

- If `yarn-session.sh` does not use `-z` to configure the specified ZooKeeper NameSpace, do not use `-yz` to specify the ZooKeeper NameSpace when using `flink run`.

Example:

```
bin/yarn-session.sh  
bin/flink run examples/streaming/WindowJoin.jar
```

- You can use `-yz` to specify a ZooKeeper NameSpace when using `flink run -m yarn-cluster` to start a cluster.
- A NameSpace cannot be shared by multiple clusters.
- If you use `-z` to specify a ZooKeeper NameSpace when starting a cluster or submitting a job, you need to use `-z` again to specify the NameSpace when deleting, stopping, or querying the job or triggering the savepoint.

## 2.4.5.5 FAQ

### 2.4.5.5.1 Savepoints-related Problems

1. Should I assign IDs to all operators of the job?

Strictly speaking, you can assign IDs to only operators with statuses because savepoints save only statuses of operators that have statuses and not operators without statuses.

However, in actual situations, you are advised to allocate IDs to all operators because some internal operators, such as window operators of Flink have statuses. Whether an operator has status or not is not obvious. If you are specific that an operator does not have status, you do not need to call `uid()` to allocate an ID to the operator.

2. What would be the impact if I add an operator with status while upgrading the job?

If you add an operator with status to the job, the status of the operator is not saved in the savepoint and thus the status cannot be recovered. The operator is processed as an operator without status and is executed from the start.

3. What would be the impact if I delete an operator with status while upgrading the job?

By default, savepoints attempt to recover all saved statuses. If the savepoint saves the status of the deleted operator, recovery fails.

You can run the following command and use the `-allowNonRestoredState (-n` in the following command) parameter to skip recovering the status of the deleted operator:

```
$ bin/flink run -s savepointPath -n [runArgs]
```

4. What would be the impact if I rearrange the sequence of operators with statuses?

- If you have allocated IDs to the operators, the statuses would be recovered normally.
- If you do not allocate IDs to the operators, IDs would be automatically allocated to the operators in the new sequence. Then, the status recovery would fail.

5. What would be the impact if I delete or add an operator without status or rearrange the sequence of operators without statuses?

- If you have allocated IDs to operators with statuses, operators without statuses do not affect status recovery from savepoints.
- If you do not allocate IDs to operators, operators with statuses may be allocated with new IDs due to the sequence change. This would cause status recovery failure.

6. What would be the impact if I change the operator concurrency during the status recovery?

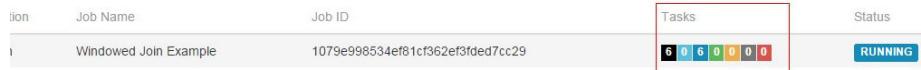
If the Flink version is higher than 1.2.0 and discarded status APIs, such as `checkpointed`, are not used, you can recover statuses from savepoints. Otherwise, statuses cannot be recovered.

#### 2.4.5.5.2 What If the Chrome Browser Cannot Display the Title

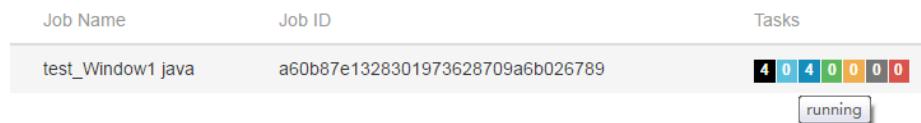
##### Question

What to do if the title is not displayed when I use Chrome browser to access the Apache Flink Dashboard? This section takes the Tasks field as an example. When the pointer is placed on a colorful box in Tasks, the title of the box is not displayed, as shown in [Figure 2-97](#). [Figure 2-98](#) shows the normal situation when the title is displayed.

**Figure 2-97 Title not displayed on the page**



**Figure 2-98 Title displayed normally**



## Answer

If the Chrome browser does not display the title, perform the following procedure:

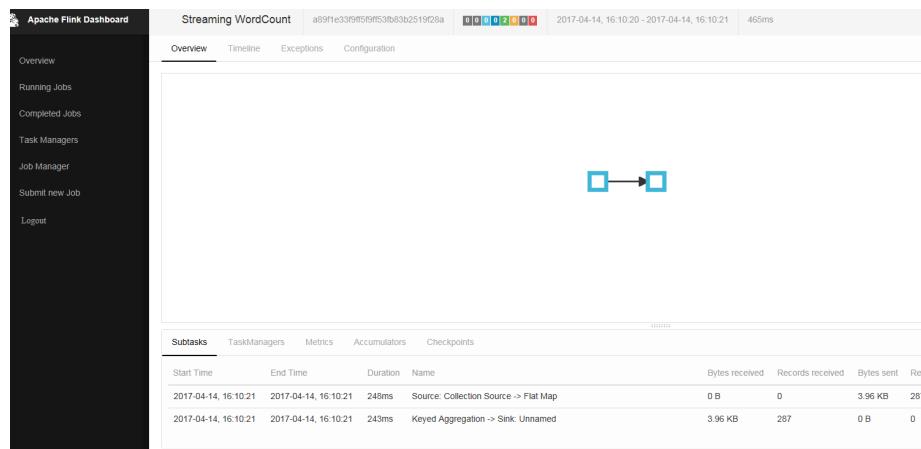
Check whether a tool that affects the tooltip display by the Chrome browser is running. If so, shut down the tool.

### 2.4.5.5.3 What If the Page Is Displayed Abnormally on Internet Explorer 10/11

## Question

What to do if Internet Explorer 10/11 does not display operator texts, as shown in [Figure 2-99](#)?

**Figure 2-99 Page displayed abnormally on Internet Explorer 10/11**



## Answer

Flink uses the `foreignObject` element to draw scalable vector graphics (SVGs) but Internet Explorer 10/11 does not support `foreignObject` and thus operators cannot be displayed normally. Google Chrome browser is supported.

#### 2.4.5.5.4 What If Checkpoint Is Executed Slowly in RocksDBStateBackend Mode When the Data Amount Is Large

##### Question

What to do if checkpoint is executed slowly in RocksDBStateBackend mode when the data amount is large?

##### Cause Analysis

Customized windows are used and the window state is ListState. There are many values under the same key. In the case of a new value, the merge operation of RocksDB is used. When calculation is triggered, all values under the key are read.

- The RocksDB mode is `merge() > merge() ... > merge() > read()`. Data reading in this mode consumes much time, as shown in [Figure 2-100](#).
- The source operator sends a large amount of data in a short period of time and the data keys are the same. The window operator fails to process data fast enough and barriers accumulate in the cache. The time consumed for snapshot preparation is too long and the window operator cannot report snapshot completion to CheckpointCoordinator in the specified time. Therefore, CheckpointCoordinator determines snapshot preparation failure, as shown in [Figure 2-101](#).

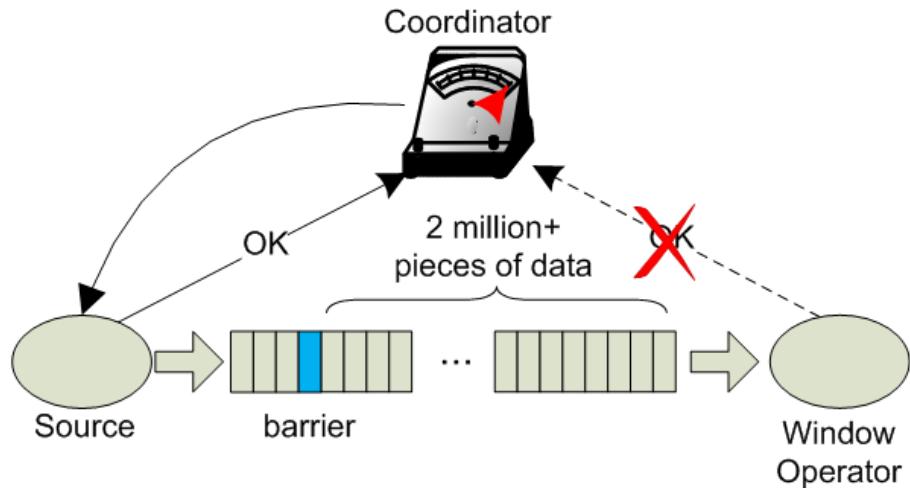
Figure 2-100 Time monitoring

```
send count: 200000 at:1489402491794
send count: 400000 at:1489402491874
send count: 600000 at:1489402491923
send count: 800000 at:1489402491963
send count: 1000000 at:1489402492006
send count: 1200000 at:1489402492045
send count: 1400000 at:1489402492094
send count: 1600000 at:1489402492134
send count: 1800000 at:1489402492173
send count: 2000000 at:1489402492260
=====Begin get at:1489402493099
=====End get at: 1489402534548
=====End iterator
((200000,400000),1489402534606)
snap in source
send count: 2200000 at:1489402535363
=====Begin get at:1489402535775
=====End get at: 1489402577386
=====End iterator
((400000,800000),1489402577414)
send count: 2400000 at:1489402577795
=====Begin get at:1489402578462
=====End get at: 1489402619442
=====End iterator
((600000,1200000),1489402619463)
send count: 2600000 at:1489402619930
=====Begin get at:1489402620571
=====End get at: 1489402660263
=====End iterator
((800000,1600000),1489402660282)
send count: 2800000 at:1489402660838
=====Begin get at:1489402661316
=====End get at: 1489402702431
=====End iterator
((1000000,2000000),1489402702450)
```

2 million pieces of data are sent within 466 ms

40S+  
39.7s is spent on reading data from RocksDB

Figure 2-101 Relationship



## Answer

Flink introduces the third-party software package RocksDB, whose defect causes the problem. You are advised to set checkpoint to FsStateBackend mode.

Set checkpoint to FsStateBackend mode in the application code as follows:

```
env.setStateBackend(new FsStateBackend("hdfs://hacluster/flink/checkpoint/"));
```

### 2.4.5.5.5 What If yarn-session Start Fails When blob.storage.directory Is Set to /home

## Question

When **blob.storage.directory** is set to **/home**, the **blobStore-UUID** file cannot be created in **/home**. This causes yarn-session start failure

## Answer

**Step 1** It is recommended that **blob.storage.directory** be set to **/tmp** or **/opt/huawei/Bigdata/tmp**.

**Step 2** When you set **blob.storage.directory** to a customized directory, manually grant permissions to the directory. This section takes the **admin** user of FusionInsight as an example.

1. Modify **conf/flink-conf.yaml** on the Flink client and run the **blob.storage.directory: /home/testdir/testdirdir/xxx** command.
2. Create the **/home/testdir** directory (level 1 is enough) and set the directory to be managed by the **admin** user.

```
SZV1000064084:/home # id admin  
uid=20000(admin) gid=9998(ficcommon) groups=9998(ficcommon),8003(System_administrator_186)  
SZV1000064084:/home # chown admin:ficcommon testdir/ -R
```

### NOTE

The **testdirdir/xxx** directory under the **/home/testdir/** directory is automatically created on each node when Flink cluster starts.

3. Run **./bin/yarn-session.sh -jm 2048 -tm 3072** on the client to check that yarn-session is normally started and the directory is successfully created.

```
SV1000064084:/home # ll testdir/
total 4
drwxr-x--- 3 admin ficommon 4096 Mar 13 11:55 testdirdir
SV1000064084:/home # ll testdir/testdirdir/
total 4
drwxr-x--- 4 admin ficommon 4096 Mar 13 11:55 xxx
SV1000064084:/home # ll testdir/testdirdir/xxx/
total 8
drwxr-x--- 2 admin ficommon 4096 Mar 13 11:55 blobStore-6fb3f049-ecf3-49ac-9fc9-95ad0aeeffd3
drwxr-x--- 2 admin ficommon 4096 Mar 13 11:55 blobStore-ad89b118-8545-4ece-8cae-1334b01de857
```

----End

#### 2.4.5.5.6 Why Does Non-static KafkaPartitioner Class Object Fail to Construct FlinkKafkaProducer010?

##### Question

After Flink kernel is upgraded to 1.3.0 or later versions, if Kafka calls the FlinkKafkaProducer010 that contains the non-static KafkaPartitioner class object as parameter to construct functions, an error is reported.

The error message is as follows:

```
org.apache.flink.api.common.InvalidProgramException: The implementation of the FlinkKafkaPartitioner is not serializable. The object probably contains or references non serializable fields.
```

##### Answer

In the 1.3.0 version of Flink, the FlinkKafkaDelegatePartitioner class is added, so that Flink allows APIs that use KafkaPartitioner, for example, FlinkKafkaProducer010 that contains KafkaPartitioner object, to construct functions.

The FlinkKafkaDelegatePartitioner class defines the member variable kafkaPartitioner.

```
private final KafkaPartitioner<T> kafkaPartitioner;
```

When Flink input parameter KafkaPartitioner constructs FlinkKafkaProducer010, the call stack is as follows:

```
FlinkKafkaProducer010(String topicId, KeyedSerializationSchema<T> serializationSchema, Properties
producerConfig, KafkaPartitioner<T> customPartitioner)
-> FlinkKafkaProducer09(String topicId, KeyedSerializationSchema<IN> serializationSchema, Properties
producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
---> FlinkKafkaProducerBase(String defaultTopicId, KeyedSerializationSchema<IN> serializationSchema,
Properties producerConfig, FlinkKafkaPartitioner<IN> customPartitioner)
-----> ClosureCleaner::clean(Object func, boolean checkSerializable)
```

Run the KafkaPartitioner object to construct a FlinkKafkaDelegatePartitioner object, and then check whether the object can be serializable. The ClosureCleaner::clean function is a static function. If the KafkaPartitioner object in a case is non-static, the ClosureCleaner::clean function cannot access the non-static member variable kafkaPartitioner in the KafkaDelegatePartitioner class and an exception is reported.

Either of the following methods can be used to solve the problem:

- Change the KafkaPartitioner class into static class.
- Use the FlinkKafkaProducer010 that contains FlinkKafkaPartitioner as the parameter to construct functions. In this case, FlinkKafkaDelegatePartitioner is not constructed and the exception about member variable is avoided.

#### 2.4.5.5.7 When I Use a Newly Created Flink User to Submit Tasks, Why Does the Task Submission Fail and a Message Indicating Insufficient Permission on ZooKeeper Directory Is Displayed?

##### Question

When I use a newly-created Flink user to submit tasks, the task submission fails because of insufficient permission on the ZooKeeper directory. The error message in the log is as follows:

```
NoAuth for /flink_base/flink/application_1499222480199_0013
```

##### Answer

1. Check whether the permission on the /flink\_base directory in ZooKeeper is 'world,'anyone: cdrwa; If no, change the permission on the /flink\_base directory to 'world,'anyone: cdrwa and go to **step 2**. If yes, go to **step 2**.
2. In the configuration file of Flink, the default value of `high-availability.zookeeper.client.acl` is `creator`, indicating that only the creator of the directory has permission on it. The user created later has no access to the /flink\_base/flink directory in ZooKeeper because only the user created earlier has permission on it.

To solve the problem, perform the following operation as the newly-created user:

- a. Check the configuration file `conf/flink-conf.yaml` on the client.
- b. Modify the parameter `high-availability.zookeeper.path.root` to the corresponding ZooKeeper directory, for example, `/flink2`.
- c. Submit tasks again.

#### 2.4.5.5.8 Why Cannot I Access the Apache Flink Dashboard?

##### Question

Why cannot I access the Apache Flink Dashboard through the URL: `http://IP address of JobManager:port of JobManager`.

##### Answer

The IP address of the computer you used has not been added to the whitelist of Apache Flink Dashboard. To solve this problem, modify the `conf/flink-conf.yaml` configuration file as follows:

1. Check whether the value of `jobmanager.web.ssl.enabled` is `false`. If not, set it to `false`.
2. Check whether the IP address of the computer you used has been added to the values of `jobmanager.web.access-control-allow-origin` and

**jobmanager.web.allow-access-address.** If the IP address has not been added, add it to the two parameters:

jobmanager.web.access-control-allow-origin:*IP address of the computer where the browser is installed*  
jobmanager.web.allow-access-address:*IP address of the computer where the browser is installed*

### 2.4.5.5.9 How Do I View the Debugging Information Printed Using System.out.println or Export the Debugging Information to a Specified File?

#### Question

System.out.println is added to the Flink service codes for printing debugging information. How do I view the debugging log? How do I export service logs to a specified file to differentiate service logs from run logs?

#### Answer

All run logs of Flink are printed to the local directory of Yarn. By default, all logs are exported to **taskmanager.log** in the local directory of Yarn container. All logs generated by invoking System.out will be exported to the **taskmanager.out** file. You can perform as follows to view the logs:

1. Log in to the native Flink web page.
2. Choose **Task Managers > Logs** or **Task Managers > Stdout** on the left to view log information.

Configure the function of printing service logs and Task Manager run logs separately:

#### NOTE

If service logs and Task Manager run logs are separately printed, service logs are not exported to the **taskmanager.log** file and cannot be viewed on the web page.

1. Modify the configuration file **logback.xml** in the **conf** directory of the client. Add the following log configuration information to the file. Modify the information in bold according to the actual situation.

```
<appender name="TEST" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>/path/test.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
        <fileNamePattern>/path/test.log.%i</fileNamePattern>
        <minIndex>1</minIndex>
        <maxIndex>20</maxIndex>
    </rollingPolicy>
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
        <maxFileSize>20MB</maxFileSize>
    </triggeringPolicy>
    <encoder>
```

```
<pattern>%d{"yyyy-MM-dd HH:mm:ss,SSS"} | %m %n</pattern>
</encoder>
</appender>

<logger name="com.huawei.bigdata.flink.examples" additivity="false">
    <level value="INFO"/>
    <appender-ref ref="TEST"/>
</logger>
```

2. Run **yarn-session.sh** to submit the task.



If the configuration file **logback.xml** contains `<file>/path/test.log</file>`, ensure that the user (configured **flink-conf.yaml**) used for running the task has the write and read permissions on the directory.

#### 2.4.5.5.10 Incorrect GLIBC Version

##### Question

When **State Backend** is set to **RocksDB** for a Flink task, the following error message is displayed:

```
Caused by: java.lang.UnsatisfiedLinkError: /srv/BigData/hadoop/data1/nm/usercache/**/appcache/application_***/rocksdb-lib-***/librocksdbjni-linux64.so: /lib64/libpthread.so.0: version `GLIBC_2.12` not found (required by /srv/BigData/hadoop/**/librocksdbjni-linux64.so)
at java.lang.ClassLoader$NativeLibrary.load(Native Method)
at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1965)
at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1890)
at java.lang.Runtime.load0(Runtime.java:795)
at java.lang.System.load(System.java:1062)
at org.rocksdb.NativeLibraryLoader.loadLibraryFromJar(NativeLibraryLoader.java:78)
at org.rocksdb.NativeLibraryLoader.loadLibrary(NativeLibraryLoader.java:56)
at
org.apache.flink.contrib.streaming.state.RocksDBStateBackend.ensureRocksDBIsLoaded(RocksDBStateBackend.java:734)
... 11 more
```

##### Possible Causes

The version of the system where the task runs and the version of the system where the compilation environment locates are different, resulting in the incompatibility of GLIBC versions.

##### Troubleshooting Method

Run the **strings /lib64/libpthread.so.0 | grep GLIBC** command to check whether the GLIBC version is earlier than 2.12.

##### Procedure

If the version of the GLIBC is too early, use the file of later version (2.12) to replace **libpthread-\* .so**. (This is a link file. You need to replace only the file that is linked to this link file.)

##### References

None

## 2.5 GraphBase Development Guide

### 2.5.1 Overview

#### 2.5.1.1 Application Development Overview

##### GraphBase Introduction

FusionInsight GraphBase is a distributed graph database based on HBase and Elasticsearch. It builds a property graph model for storage and provides powerful graph query, analysis, and traversal capabilities. It has the following features:

- The HBase-based distributed storage mechanism is provided to process massive data.
- The Spark distributed memory computing technology is provided to support quick data import.
- The Elasticsearch-based index mechanism is provided to quickly query data based on indexes.

##### Interface Type Introduction

- REST API Interface

REST APIs are developed using the Java language that is simple and easy. Therefore, you are advised to use the Java language to develop upper-layer applications. For details, see **GraphBase** in *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

The following table describes the functions provided by GraphBase after REST API is invoked.

Function	Description
User login and logout	This function can be used for access authentication.
Metadata read/write	This function can be used to add, delete, modify, and query metadata.
Metadata-based operations on vertices and edges	This function can be used to add, delete, modify, and query vertices and edges.
Filtering and querying relationships for a large amount of data	This function can be used for full graph query, conditional query, line expansion query, and path query for vertices and edges.
Index management	This function can be used to create and query full graph indexes and edge indexes, and query the status of asynchronous index tasks.

- Gremlin API Interface

Gremlin is the graph traversal language of Apache TinkerPop. Gremlin is a functional, data-flow language that enables users to succinctly express complex traversals on (or queries of) their application's property graph. Every Gremlin traversal is composed of a sequence of (potentially nested) steps. A step performs an atomic operation on the data stream. For details about Gremlin, visit <http://tinkerpop.apache.org/docs/3.3.2/reference/#traversal>.

Gremlin has the following three types of basic operations:

- map-step: converts objects in a data flow.
- filter-step: filters objects in a data flow.
- sideEffect-step: calculates data flows.

### 2.5.1.2 Basic Concepts

Like most graph databases, GraphBase uses property graphs for modeling. Based on the property graph models, GraphBase has the following basic concepts:

- **Vertex**: A vertex is also called a node, which is used to specify an entity object in the real world, such as a person.
- **Vertex label**: A vertex label indicates a node type that specifies the type of an entity object in the real world. Example: person. In GraphBase, a node has only one vertex label. The default value is **vertex**.
- **Edge**: An edge, also known as a relationship that is used to specify the connections between two entity objects (vertices in a graph) in the real world. For example, the relationship between two persons is friend. Edges in GraphBase are unidirectional, which starts from a vertex and ends at another. Therefore, a bidirectional edge has an incoming and an outgoing edge.
- **Edge label**: An edge label specifies the type of a relationship in the real world. For example, the relationship is friend.
- **Property**: A property describes some attribute of either a vertex or an edge in the format of key-value pair. Property key is used to describe the key in the key-value pair, property value describes a specific value. For example, a property key is **name**, the property value is **Zhang San**.

### 2.5.1.3 Development Process

This section describes how to develop a GraphBase application based on GraphBase REST API and Gremlin API.

[Figure 2-102](#) shows the development process, and [Table 2-52](#) describes each phase.

Figure 2-102 GraphBase application development process

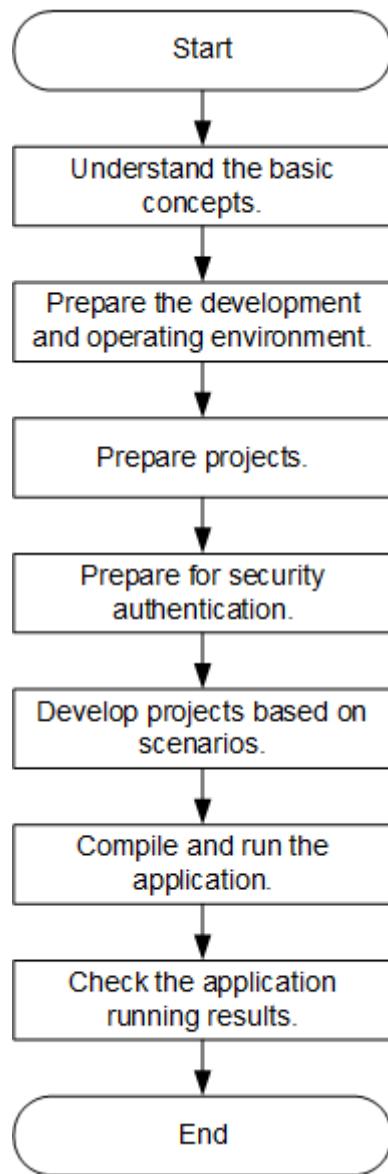


Table 2-52 GraphBase application development process

Phase	Description	Reference Document
Understand basic concepts about GraphBase.	Before application development, learn basic concepts of GraphBase, understand the scenario requirements, and design tables.	<a href="#">Basic Concepts</a>

Phase	Description	Reference Document
Prepare the development and running environment.	REST APIs and the Gremlin language are recommended for GraphBase application development. You can use the IntelliJ IDEA tool. The GraphBase running environment is the GraphBase client. Install and configure the client according to the guide.	<ul style="list-style-type: none"><li>For details about how to prepare the development environment for the REST API, see <a href="#">Preparing the Development and Operating Environment</a>.</li><li>For details about how to prepare the development environment for the Gremlin API, see <a href="#">Preparing the Development and Operating Environment</a>.</li></ul>
Prepare a project.	GraphBase provides sample projects for different scenarios. You can import a sample project to learn the application.	<ul style="list-style-type: none"><li>For details about how to prepare a project for the REST API, see <a href="#">Configuring and Importing Sample Projects</a>.</li><li>For details about how to prepare a project for the Gremlin API, see <a href="#">Configuring and Importing a Piece of Sample Code</a>.</li></ul>
Preparing for security authentication.	In normal mode of the cluster, security authentication must be performed for the REST API.	For details about how to perform security authentication for the REST API, see <a href="#">Preparing for Security Authentication</a> .

Phase	Description	Reference Document
Develop a project based on the scenario.	<ul style="list-style-type: none"><li>Sample projects for developing REST APIs are provided, including projects for creating and deleting graphs and performing full graph query, path query, and line expansion query.</li><li>The Gremlin syntax scenario is provided, including syntax scenarios for querying a specified graph object, querying one or more properties, traversing graph objects and paths.</li></ul>	<a href="#">Developing an Application</a>
Compile and run applications.	You can compile a developed application and submit it for running.	<ul style="list-style-type: none"><li><a href="#">Compiling and Running an Application</a></li><li><a href="#">Compiling and Running an Application</a></li></ul>
Check the application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	<ul style="list-style-type: none"><li><a href="#">Viewing Windows Commissioning Results</a></li><li><a href="#">Viewing Linux Commissioning Results</a></li></ul>

## 2.5.2 Environment Preparation

### 2.5.2.1 REST API Development Environment Preparation

#### 2.5.2.1.1 Preparing the Development and Operating Environment

[Table 2-53](#) describes the development and running environment required for secondary development.

**Table 2-53 Environment**

Preparation Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows 7 or later</li><li>• Operating environment: Windows OS or Linux OS. If the program needs local debugging, the operating environment must be able to communicate with the cluster service plane.</li></ul>
JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA	<p>Tool used for developing GraphBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you are using an IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If you are using an Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If you are using an OpenJDK, ensure that the JDK configured in IntelliJ IDEA is the OpenJDK.</li></ul>
Maven	Basic configuration of the development environment. It can be used for project management throughout the lifecycle of software development.

Preparation Item	Description
JUnit plug-ins	Basic configuration of the development environment.
User development	See <a href="#">Preparing a Developer Account</a> for configuration.
7-Zip	Tool used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

## Preparing the Operating Environment

During application development, prepare the environment for running and commissioning code to verify that the application can run properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required for commissioning; configure the network connection, and commission the application in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) and click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar**. Then, decompress this file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\GraphBase\config** and import the configuration file to the configuration file directory (usually the **conf** folder) of the GraphBase sample project.

The keytab file obtained during [Preparing a Developer Account](#) is also stored in this directory. **Table 2-54** describes the main configuration files.

**Table 2-54** Configuration files

File	Description
graphbase.properties	Parameters required for the GraphBase sample code
Person.xml	Files required by Schema

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where

the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

#### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, install the client in the **/opt/hadoopclient** directory.

The difference between the client time and the cluster time must be less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. Log in to FusionInsight Manager. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/GraphBase/config** directory to the **conf** directory where the compiled JAR file is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active OMS node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp GraphBase/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

[Table 2-55](#) describes the main configuration files.

**Table 2-55** Configuration files

File	Description
graphbase.properties	Parameters required for the GraphBase sample code
Person.xml	Files required by Schema

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is

no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

### 2.5.2.1.2 Configuring and Importing Sample Projects

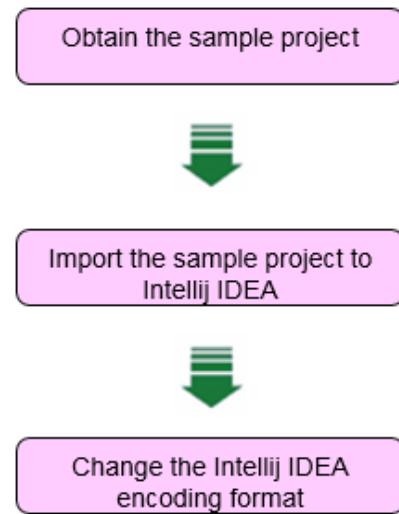
#### Prerequisites

You have installed the GraphBase client.

#### Scenarios

GraphBase provides sample projects for multiple scenarios to help customers quickly learn REST APIs. The following describes how to import the GraphBase sample code that can be imported to IntelliJ IDEA. [Figure 2-103](#) shows the importing process.

**Figure 2-103** Process of importing a sample project

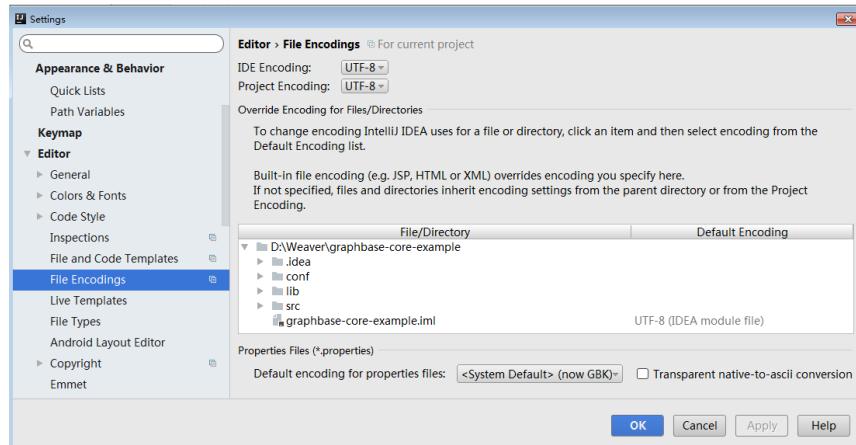


#### Procedure

- Step 1** Obtain the sample project folder **graphbase-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the sample project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > New > Project from Existing Sources**.
  2. Select the sample project folder **graphbase-core-example** and click **OK**.
  3. In the **Import Project** dialog box, select **Create project from existing sources**.
  4. Click **Next** until **JDK** is set to **jdk1.8.0\_402**. Then, click **Next**. On the displayed dialog box, click **Finish**.
- Step 3** Configure the text file encoding format of IntelliJ IDEA to avoid garbled characters.

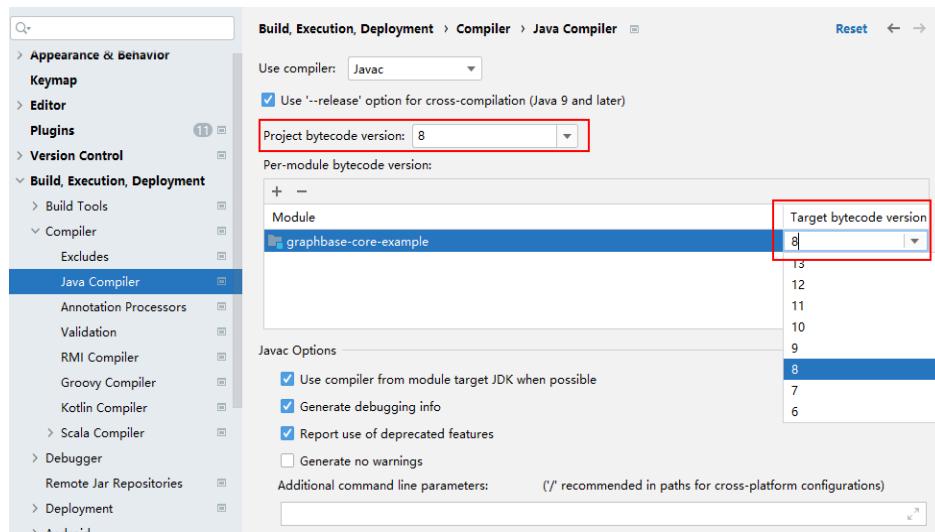
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.
2. In the displayed **Settings** page, choose **Editor > File Encoding** from the navigation pane on the left. On the displayed page, set **IDE Encoding** and **Project Encoding** to **UTF-8**, and click **Apply**. On the displayed page, click **OK**, as shown in [Figure 2-104](#).

**Figure 2-104** Setting the encoding format of IntelliJ IDEA

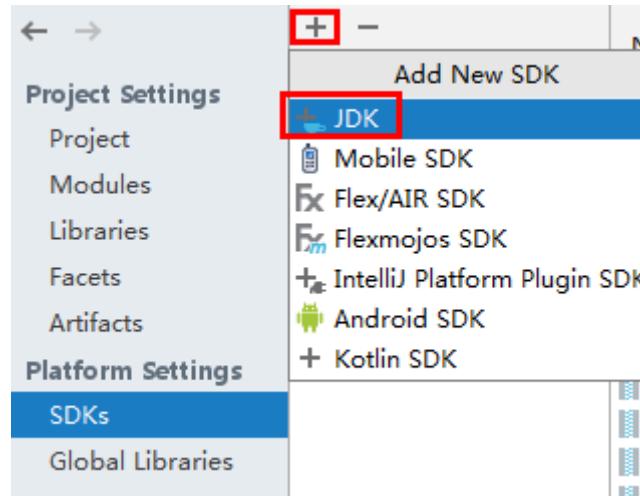


#### Step 4 Set the JDK of the project.

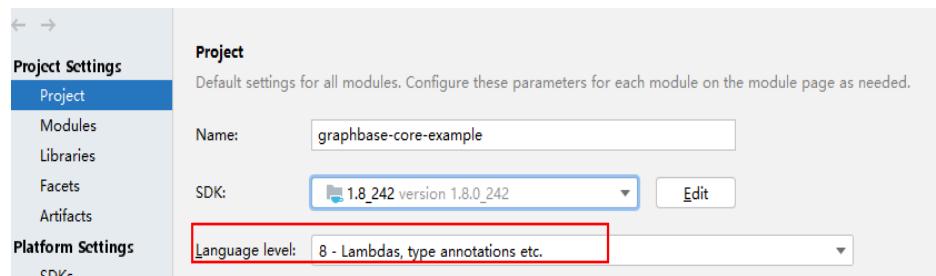
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **graphbase-core-example** to **8**.



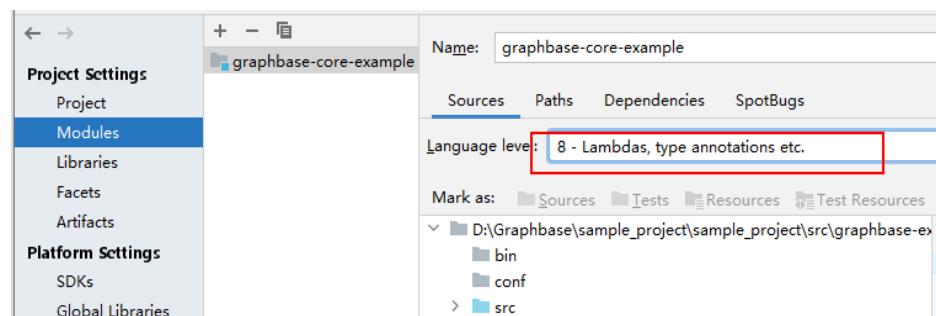
3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.



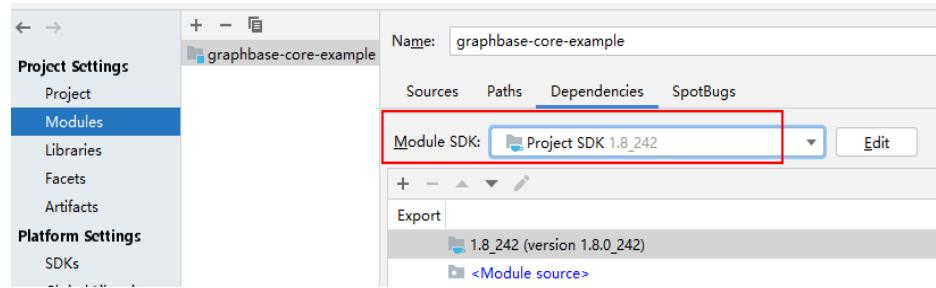
6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc..**



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.



11. Click **Apply** and then **OK**.

----End

### 2.5.2.1.3 Preparing for Security Authentication

GraphBase uses keytab authentication which never expires. You are advised to use keytab files for authentication on the client.

1. Modify the configuration data.

- In the **conf** directory of the root directory of the sample project, set the **graphbase.properties** configuration file, IP address, port number, and username as you need.

#### NOTE

The IP address and port number of the GraphBase service is the floating IP address and port number specified when the GraphBase component is installed. Log in to FusionInsight Manager, choose **Cluster** > *Name of the desired cluster* > **Services** > **GraphBase** > **Configurations** > **All Configurations**, and search for **loadbalancer.floatip**. The value of **loadbalancer.floatip** is the floating IP address, the value of **loadbalancer.https.port** is the port number.

Parameter	Value
<b>GraphBase-&gt;LoadBalancer</b>	
* <b>loadbalancer.floatip</b>	10 . 5 . 89 . 50
* <b>loadbalancer.https.port</b>	22380

**Note:** In an IPv6 network, the floating IP address must be enclosed by square brackets, for example, [IPv6 address].

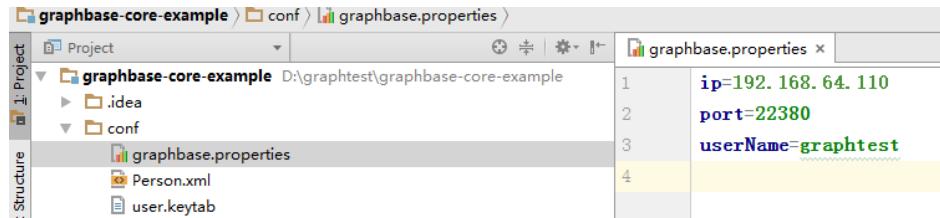
- Download the keytab file of the machine-machine user and copy it to the **conf** directory in the root directory of the sample project.

Log in to FusionInsight Manager, click **System**, and choose **Permission > User**. On the displayed page, click **More > Download Authentication Credential** to obtain the **user.keytab** authentication file.

#### NOTE

The authentication credential contains the **krb5.conf** file of Kerberos. If there are multiple clusters, select the **krb5.conf** file of the corresponding cluster.

- Modify the configuration file as follows.



## 2. Obtain Keytab authentication code.

Construct an HTTP client, log in to it with Kerberos authentication, obtain CSRF TOKEN information, and call RESTful APIs. The following code snippets are in the **GraphHttpClient** class of the **com.huawei.security** package.

```

/* step 1: create http client */
final X509TrustManager trustManager = new X509TrustManager() {
    @Override
    public final X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    @Override
    public final void checkClientTrusted(X509Certificate ax509certificate[], String s) throws
CertificateException {
        // to do nothing.
    }

    @Override
    public final void checkServerTrusted(X509Certificate ax509certificate[], String s) throws
CertificateException {
        // to do nothing.
    }
};

final javax.net.ssl.SSLContext context = javax.net.ssl.SSLContext.getInstance("TLS");
context.init(null, new TrustManager[]{trustManager}, null);
final ThreadSafeClientConnManager connectionManager = new ThreadSafeClientConnManager();
connectionManager.setMaxTotal(100);
connectionManager.getSchemeRegistry().register(new Scheme("https", 443, new SSLSocketFactory(
    context, SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER)));

CloseableHttpClient httpclient = new DefaultHttpClient(connectionManager);

/*
 * step 2:
 * usage: kerberos login based on username and keytab,
 * note: Password never expired
 */
HttpPost loginPost = new HttpPost(httpAuthInfo.getBaseUrl() + "/login");
MultipartEntityBuilder entityBuilder = MultipartEntityBuilder.create();
entityBuilder.addPart("username", new StringBody(httpAuthInfo.getUsername(),
ContentType.create("text/plain", Consts.UTF_8)));
entityBuilder.addBinaryBody("keytabfile", new File(httpAuthInfo.getKeytabFilePath()));
loginPost.setEntity(entityBuilder.build());
loginRsp = httpclient.execute(loginPost);
RestHelper.checkHttpRsp(loginRsp);

/* step 3: get CSRF-Token */
HttpGet httpGet = new HttpGet(httpAuthInfo.getBaseUrl() + "/graph/user/me");
CloseableHttpResponse csrfTokenRsp = httpClient.execute(httpGet);
}

```

## 2.5.2.2 Gremlin API Development Environment Preparation

### 2.5.2.2.1 Preparing the Development and Operating Environment

**Table 2-56** describes the development and running environment required for secondary development.

**Table 2-56** Environment

Preparation Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows 7 or later</li><li>• Operating environment: Windows OS or Linux OS. If the program needs local debugging, the operating environment must be able to communicate with the cluster service plane.</li></ul>
JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>

Preparation Item	Description
IntelliJ IDEA	<p>Tool used for developing GraphBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you are using an IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If you are using an Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If you are using an OpenJDK, ensure that the JDK configured in IntelliJ IDEA is the OpenJDK.</li></ul>
Maven	Basic configuration of the development environment. It can be used for project management throughout the lifecycle of software development.
JUnit plug-ins	Basic configuration of the development environment.
7-Zip	Tool used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

## Preparing the Operating Environment

During application development, prepare the environment for running and commissioning code to verify that the application can run properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required for commissioning; configure the network connection, and commission the application in Windows.
    - a. **Accessing FusionInsight Manager of an MRS Cluster** and click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar**. Then, decompress this file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

    - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\GraphBase\config** and import the configuration file to the configuration file directory (usually the **conf** folder) of the GraphBase sample project.
- The keytab file obtained during **Preparing a Developer Account** is also stored in this directory. **Table 2-57** describes the main configuration files.

**Table 2-57** Configuration files

File	Description
graphbase.properties	Parameters required for the GraphBase sample code
Person.xml	Files required by Schema

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, install the client in the **/opt/hadoopclient** directory.

The difference between the client time and the cluster time must be less than 5 minutes.

For details about how to install a cluster client, see [Installing a Client](#).

- b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then log in to the active management node as user **root**. Go to the decompression directory of the cluster client, and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/GraphBase/config** directory to the client node to the **conf** directory where the compiled JAR package is stored, for example, **/opt/hadoopclient/conf**, for subsequent commissioning.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active OMS node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp GraphBase/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

**Table 2-58** describes the main configuration files.

**Table 2-58** Configuration files

File	Description
log4j.properties	Gremlin log parameters
remote-objects.yaml	Detailed parameters for Gremlin query

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

### 2.5.2.2 Configuring and Importing a Piece of Sample Code

1. Obtain the sample project folder **gremlin-demo4j** in the **graphbase-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Import the sample code to IntelliJ IDEA. For details, see [Configuring and Importing Sample Projects](#).

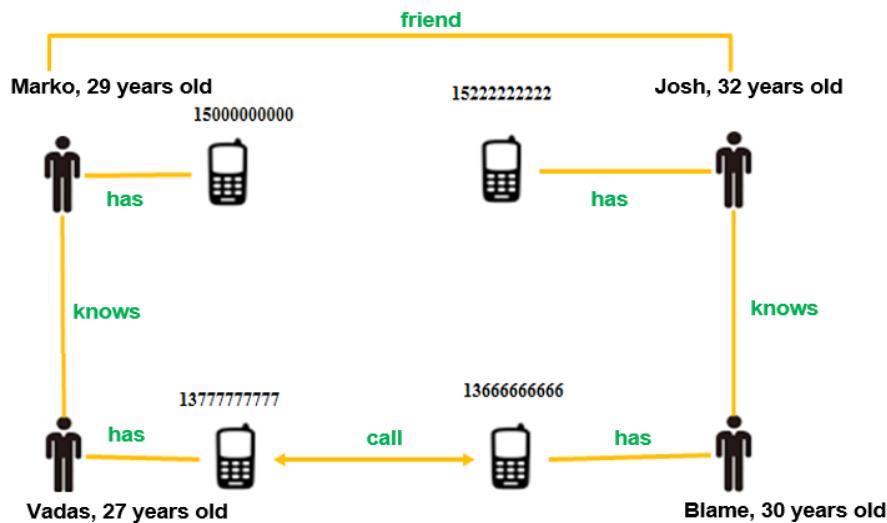
## 2.5.3 Developing an Application

### 2.5.3.1 Typical Application Scenario

You can quickly learn and master the GraphBase development process and know key interface functions in a typical application scenario.

#### Actual Scenario

Typical social relationships:



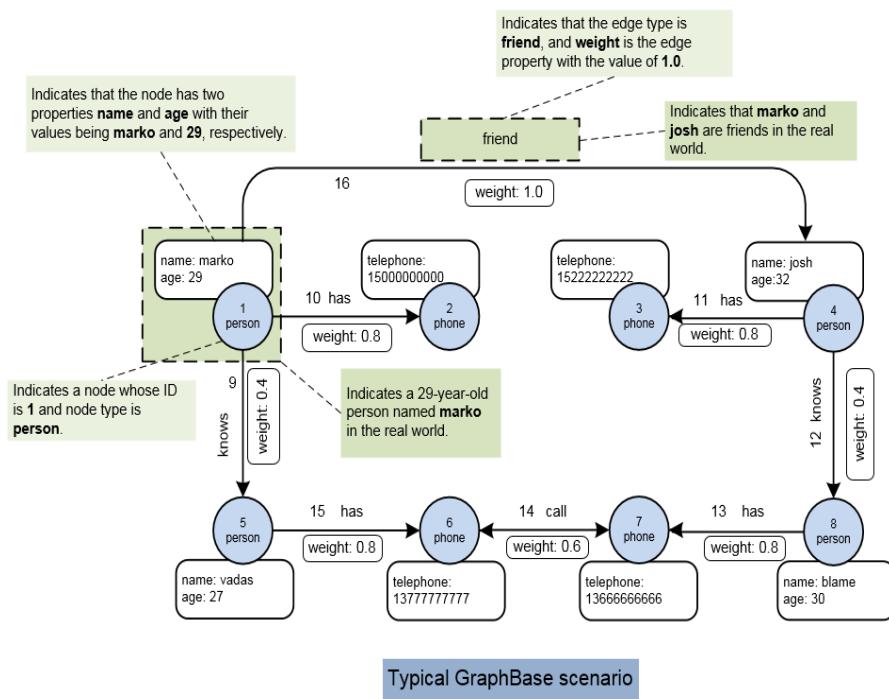
In the scenario, there are eight real world entities, four persons and four mobile phones.

There are eight relationships between the entity objects, including **friend**, **knows**, **call**, and **has**.

## Graph Modeling Scenario

The purpose of graph modeling is to map the real world relationships to GraphBase using a graph:

- **Vertex**: maps persons and mobile phones in the real world to GraphBase using vertices.
  - **Vertex label**: defines the types of persons and mobile phone in the real world in GraphBase. That is, people are labeled as **person**, and mobile phones are defined as **phone** in the graph.
  - **Edge**: maps the relationships between people and people, people and mobile phones, and mobile phones and mobile phones in the real world to GraphBase using edges.
  - **Edge label**: defines the relationships between entity objects in the real world in GraphBase. The definitions of these relationships in this scenario are as follows:
    - The friend relationship between people is defined as the edge label **friend**.
    - The acquaintance relationship is defined as the edge label **knows**.
    - The ownership between mobile phones and people is defined as **has**.
    - The communication relationship between mobile phones is defined as the edge label **call**.
  - **Property**: defines the feature information of each entity object in the real world in GraphBase. The name and age of a person is defined as the property **name** and **age**, respectively, and the mobile phone number is defined as the property **telephone**.
  - **New property**: Considering that GraphBase supports the value properties obtained by the evaluation system, the relationship weight property definition is added to indicate the importance of relationships. That is, the **weight** property is defined. It is considered that the value property has been obtained.
- The graph model is as follows.



## Development Guidelines

Based on the preceding scenario description, the basic operation process of creating a GraphBase service and querying data using the created service are as follows:

- Take **node 1 > node 2** in the scenario as an example, there are three elements included: node 1, node 2, and edge 10.
  - Create a schema using the two nodes, see instructions provided in [Creating a Schema](#).  
You need to create two vertex labels, **person** and **phone**, and an edge label, **has**.  
You need to create the properties of all nodes and edges: **name**, **age**, **telephone**, and **weight**.
  - (Optional) Create indexes for properties to improve query efficiency. For details, see [Creating an Index](#).
  - Create vertices and edges.  
Create vertex 1, vertex 2, and the edge 10. For details about how to create vertices and edges, see [Creating and Querying a Vertex or an Edge](#).
  - Query data.  
Method 1: Invoke a REST API for query.  
Invoke the full graph query interface to query vertices and edges. For details, see [Performing a Full Graph Query](#).  
Method 2: Perform data query using Gremlin statements. For details, see [Gremlin Console](#), [Gremlin Java](#) and [Gremlin Application Scenarios](#).  
You can create other nodes and edges in the same way.

- If the data volume is large, you can import the data in batches. The procedure is as follows:
  - a. Create a schema.  
Compile an XML file. For details, see [Compiling a Schema File \(XML\)](#).  
Create a schema by uploading the XML file. For details, see [Creating a Schema by Uploading an XML File](#).
  - b. Import data.  
Compile a data file (CSV), data description file (DESC), and a graph mapping rule file (.mapper). For details, see [Preparing Data Files](#).  
Import data in real time or in batches. For details, see [Importing Data](#).
  - c. Query data.  
Method 1: Invoke a REST API for query.  
Invoke the full graph query interface to query vertices and edges. For details, see [Performing a Full Graph Query](#).  
Method 2: Perform data query using Gremlin statements. For details, see [Gremlin Console](#), [Gremlin Java](#) and [Gremlin Application Scenarios](#).
- During data query, except vertex and edge query, you can query paths between vertices and perform line expansion queries. For details, see [Performing a Path Query](#) and [Performing a Line Expansion Query](#).

### 2.5.3.2 Preparing Data Files

#### 2.5.3.2.1 Compiling a Schema File (XML)

##### Defining the Schema File (XML)

In GraphBase, vertices and edges are entities. You need to create the metadata information for the entity labels, such as vertex labels, edge labels, property names, property value types. Such metadata information is the schema information of a graph.

1. Define a schema file **person.xml** based on the scenario described in [Typical Application Scenario](#) to create a schema file (containing vertexLabel, edgeLabel, and propertyKey).

The following code snippets are used as an example. For complete codes, see [GraphBase.examples.person-demo](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<schema>
    <vertexLabelList>
        <vertexLabel name="person" isPartitioned="false"/>
        <vertexLabel name="phone" isPartitioned="false"/>
    </vertexLabelList>
    <edgeLabelList>
        <edgeLabel name="friend"></edgeLabel>
        <edgeLabel name="knows"></edgeLabel>
        <edgeLabel name="call"></edgeLabel>
        <edgeLabel name="has"></edgeLabel>
    </edgeLabelList>
    <propertyKeyList>
        <propertyKey name="name" dataType="STRING"/>
        <propertyKey name="age" dataType="INTEGER"/>
        <propertyKey name="telephone" dataType="STRING"/>
    </propertyKeyList>
</schema>
```

```
<propertyKey name="weight" dataType="FLOAT"/>
</propertyKeyList>
<graphIndexList>
    <graphIndex name="name_index" elementCategory="VERTEX" type="COMPOSITE" unique="false">
        <keyTextTypeList name="name" textType="" />
    </graphIndex>
    <graphIndex name="age_index" elementCategory="VERTEX" type="MIXED" unique="false">
        <keyTextTypeList name="age" textType="DEFAULT" />
    </graphIndex>
    <graphIndex name="telephone_index" elementCategory="VERTEX" type="COMPOSITE" unique="false">
        <keyTextTypeList name="telephone" textType="" />
    </graphIndex>
    <graphIndex name="weight_index" elementCategory="EDGE" type="MIXED" unique="false">
        <keyTextTypeList name="weight" textType="DEFAULT" />
    </graphIndex>
</graphIndexList>
</schema>
```

- **vertexLabelList:** indicates the vertex label set. These labels are optional in the preceding XML file, and the label content can be empty.

**vertexLabel:** indicates the vertex label. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name:** indicates the vertex label name, which is mandatory.

- **edgeLabelList:** indicates the edge label list. This parameter can be missing from the preceding XML file, and the parameter value can also be left blank.

**edgeLabel:** indicates an edge label. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name:** indicates the edge label name, which is mandatory.

**primaryKeys:** indicates the primary key to which EdgeLabel are bound with. Multiple primary keys can be specified. This parameter is optional. If there are multiple edges between two vertices and they have the same primaryKeys value, the edges can be considered as the same. For example:

```
<edgeLabel name="friend">
    <primaryKeys>p1</primaryKeys>
    <primaryKeys>p2</primaryKeys>
</edgeLabel>
```

- **propertyKeyList:** indicates the property key set. This parameter can be missing from the preceding XML file, and the parameter value can also be left blank.

**propertyKey:** indicates a property key. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name:** indicates the property name, which is mandatory.

**dataType:** indicates the property value data type, which is optional. The default value is **STRING**.

- **graphIndexList:** indicates the index set. This parameter can be missing from the preceding XML file, and the parameter value can also be left blank.

**graphIndex**: indicates the index. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name**: indicates the index name, which is mandatory.

**elementCategory**: indicates the index entity type, which is mandatory. The options are **VERTEX** and **EDGE**, which is case insensitive.

**type**: indicates the index type, which is mandatory. The options are as follows: **COMPOSITE** (Composite index: internal exact match index with index data stored in HBase) or **MIXED** (mixed index: external mixed match index with index data stored in Elasticsearch) which is case insensitive.

The analyzer cannot be configured by using COMPOSITE indexes.

- If the property key of this index does not contain any Chinese, you do not need to configure the analyzer.
- You can configure the analyzer only for data types, such as TEXT and TEXTSTRING.
- The analyzer can be set for MIXED indexes:
  - ik\_max\_word (fine-grained)
  - ik\_smart (coarse-grained, recommended)

**keyTextTypeList**: indicates the property key list in text format. This parameter can be missing from the preceding XML file, but the parameter value cannot be left blank once the parameter exists.

**name**: indicates the property name, which is mandatory.

**textType**: indicates the index data type. The options are as follows:

- **TEXT**: indicates that the index data is of the text type.
- **TEXTSTRING**: indicates that the index data is of the text string type.
- **PREFIX\_TREE**: indicates that the index is of the geographical location type.
- **DEFAULT**: indicates the default index type. The value is case insensitive and can be empty. If this parameter is left empty, the default type is used.

## NOTICE

- GraphBase index categories:
  - Composite index: is the internal exact match index with the index data stored in HBase and does not support full-text search or splitter.
  - Mixed Index: is the external mixed match index with the index data stored in Elasticsearch and supports full-text search and splitter.
- Suggestions for using GraphBase indexes:
  - When a property is the vertex primary key, an internal index (Composite Index) must be created.
  - You are advised to create mixed indexes in common cases because the index search performance is better and the functions are and richer than the composite index.
  - The field definitions in the XML file labels are the same as those in the preceding labels. For details, see "Adding a Vertex Label", "Adding an Edge Label", "Adding a Property Key", "Adding a Graph Index", and "Vertex-centric Index Management" in "GraphBase" of the *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

## 2. Create metadata using the schema file.

REST APIs can be used to upload the schema file. For details, see [Creating a Schema by Uploading an XML File](#) in the sample project.

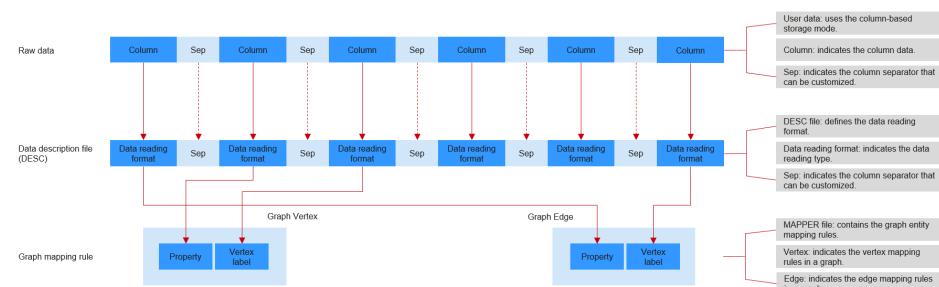
### 2.5.3.2.2 Compiling a Data File (CSV)

#### Data-Related Files

Data-related files are classified into the following types:

1. User data files, also called raw data files that are in CSV format.
2. Data description file is used to define the schema mapping relationship for data, and the file format is DESC.
3. Graph mapping rule file, which defines the mapping between raw data and entities in the graph database. The file format is MAPPER.

The following figure shows the relationship between the three types of files.



## Compiling Raw Data Files

Based on the instructions provided in [Typical Application Scenario](#), define the raw data file as follows (The following code snippets are used as an example. For complete codes, see [GraphBase.examples.person-demo](#)):

Note: The data in each column corresponds to the **index** value of properties in the graph mapping file (.mapper).

```
marko,29,,vadas,27,,knows,0.4,  
marko,29,150000000000,,,0.8,,  
josh,32,1522222222,,0.8,,  
josh,32,,blame,30,,knows,0.4,  
blame,30,136666666666,,,0.8,,  
,,136666666666,,137777777777,,0.6  
,,137777777777,,136666666666,,,0.6  
vadas,27,137777777777,,0.8,,  
marko,29,,josh,32,,friend,1.0,
```

### 2.5.3.2.3 Compiling a Data Description File (DESC)

#### Scenarios

Data description files (in DESC format) corresponding to CSV data files are required during data reading.

Based on the information provided in [Typical Application Scenario](#), design data description files as follows (The following code snippets are used as an example. For complete codes, see [GraphBase.examples.person-demo](#)):

```
{"delim":",",  
"includeRowHead":false,  
"attributes": [  
 {"name":"attr_0","type":"STRING","nominalList":[]},  
 {"name":"attr_1","type":"INTEGER","nominalList":[]},  
 {"name":"attr_2","type":"STRING","nominalList":[]},  
 {"name":"attr_3","type":"STRING","nominalList":[]},  
 {"name":"attr_4","type":"INTEGER","nominalList":[]},  
 {"name":"attr_5","type":"STRING","nominalList":[]},  
 {"name":"attr_6","type":"REAL","nominalList":[]},  
 {"name":"attr_7","type":"STRING","nominalList":[]},  
 {"name":"attr_8","type":"REAL","nominalList":[]},  
 {"name":"attr_9","type":"REAL","nominalList":[]} ]}
```

#### File Format Description

For details about the data description files (in DESC or HVDE format) corresponding to CSV data files, see [Table 2-59](#).

**Table 2-59** Data description file examples

File Format	Description
DESC	<p>The file contains <b>delim</b> and <b>attributes</b>.</p> <ul style="list-style-type: none"><li>• <b>delim</b>: defines the delimiter for data in the CSV data file. Ensure that you use the same delimiter in the CSV data file, otherwise, data fails to be imported, including commas (,), spaces, tabs, and user-defined delimiters. For example, a slash (/) or a colon (:).</li><li>• <b>includeRowHead</b>: specifies whether a header is included. If it is set to <b>false</b>, no header is included. If it is set to <b>true</b>, a header is included.</li><li>• <b>attributes</b>: includes <b>name</b>, <b>type</b>, <b>nominalList</b>, and <b>Format</b>.</li><li>• <b>type</b>: specifies the feature data type, which can be String, Integer, Real, Date, Time, or Timestamp.</li><li>• <b>name</b>: specifies the feature name.</li><li>• <b>nominalList</b>: specifies the enumerated value list.</li><li>• <b>alueFormat</b>: specifies the date format when <b>type</b> is set to <b>DateTime</b>, for example, yyyy-MM-dd HH:mm:ss.</li></ul> <p>Metadata is defined as follows:</p> <pre>{   "delim": ",",   "includeRowHead": false,   "attributes": [     {"name": "name", "type": "STRING", "nominalList": []},     {"name": "date", "type": "DATETIME", "nominalList": [], "valueFormat": "yyyy-MM-dd HH:mm:ss"} ]}</pre>

 **NOTE**

- A column name cannot contain special characters, for example, #@\*.
- The yyyyMMdd time format corresponds to year, month, and day. A complete time format is yyyyMMdd HH:mm:ss, corresponding to year, month, day, hour, minute, and second. The common time formats are yyyy/MM/dd and yyyy-MM-dd.

#### 2.5.3.2.4 Compiling a Graph Mapping Rule File (.mapper)

A graph mapping rule describes the mapping between the columns of a data set and the graph elements.

### Scenarios

Based on **Typical Application Scenario**, design the graph mapping rule file as follows (The following code snippets are used as an example. For complete codes, see **GraphBase.examples.person-demo**):

```
{  
  "entities": {  
    "entity1": {  
      "conceptIRI": "person",  
      "primaryKey": "name",  
      "label": "Person",  
      "type": "CLASS",  
      "valueFormat": "yyyy-MM-dd HH:mm:ss",  
      "valueType": "DATETIME",  
      "valueList": []  
    }  
  }  
}
```

```
"properties": {
    "name": {
        "index": 0
    },
    "age": {
        "index": 1
    }
},
"entity2": {
    "conceptIRI": "phone",
    "primaryKey": "telephone",
    "properties": {
        "telephone": {
            "index": 2
        }
    }
},
"entity3": {
    "conceptIRI": "person",
    "primaryKey": "name",
    "properties": {
        "name": {
            "index": 3
        },
        "age": {
            "index": 4
        }
    }
},
"entity4": {
    "conceptIRI": "phone",
    "primaryKey": "telephone",
    "properties": {
        "telephone": {
            "index": 5
        }
    }
},
"relationships": [
{
    "source": "entity1",
    "target": "entity2",
    "unique": true,
    "label": "has",
    "properties": {
        "weight": {
            "index": 6
        }
    }
},
{
    "source": "entity1",
    "target": "entity3",
    "unique": true,
    "labelType": "columnLookup",
    "labelColumn": {
        "index": 7
    },
    "properties": {
        "weight": {
            "index": 8
        }
    }
},
{
    "source": "entity2",
    "target": "entity4",
    "unique": true,
```

```
        "label": "call",
        "properties": {
            "weight": {
                "index": 9
            }
        }
    }]
}
```

## File Structure

The basic structure is as follows:

(The texts in bold are the defined property names, which are set by users.) The italic words can be entered by users based on scenarios and requirements. The underscored italic words indicate that the corresponding schemas must be created in the graph in advance.)

```
{
  "entities": {
    "entityName1": {
      "conceptIRI": "conceptName1",
      "primaryKey": "primaryKeyName1",
      "properties": {
        "propertyName1": {
          "type": "propertyValueType",
          "": ":{ }"
        },
        "propertyName2": {
          "type": "propertyValueType",
          "": ":{ }"
        }
      }
    },
    "entityName2": {
      "conceptIRIType": "constant",
      "conceptIRI": "conceptName2",
      "primaryKey": "primaryKeyName2",
      "properties": {
        "propertyName1": {
          ...
        }
      }
    }
  },
  "relationships": [
    {
      "source": "entityName1",
      "target": "entityName2",
      "label": "labelName1"
    },
    {
      "source": "entityName3",
      "target": "entityName4",
      "label": "labelName2",
      "properties": {
        "propertyName1": {
          "type": "propertyValueType",
          "": ":{ }"
        },
        "propertyName2": {
          "type": "propertyValueType",
          "": ":{ }"
        }
      }
    }
  ]
}
```

- **entities** (mandatory): specifies entity sets. Entities indicate vertices in a graph. An entity set contains multiple entities.
- **relationships** (mandatory): specifies relationship sets. Relationships indicate edges in a graph.
- **entityName** (mandatory): specifies the entity name, which identifies an entity object and is referenced when a relationship is defined. In a mapping rule, entities cannot use the same name.

---

**NOTICE**

The entity that is defined after a used name in the graph overwrites the previously defined entity with the same name.

- **conceptIRIType** (optional): specifies the conceptIRI (entity label) definition mode. The available values are **constant** and **columnLookup**. If this parameter is not defined, the default value **constant** is used.
  - **constant**  
Specifies that a constant is defined as conceptIRI.  
Parameter:  
**conceptIRI** (mandatory): specifies an entity label, that is, an entity category. For example, conceptIRI can be a person or an item. **conceptIRI** is a string.
  - **columnLookup**  
Defines a value as the **conceptIRI**.  
Parameter:  
**concept** (mandatory): specifies a label value. The definition method is the same as that used for defining a property value.
- **primaryKey** (mandatory): specifies a primary key, which is the unique identifier of an entity (vertex). For example, the ID card number of a person. The value of **primaryKey** must be a property name in properties, and the property must be of the single type or a combination of required and single. That is, the property value must be a column of values in the input data rather than a multi-condition combination value. In addition, Built-in indexes must be created for the property that is used as a primary key. Currently, only one value can be specified for **primaryKey**, and the value cannot contain multiple primary keys.
- **createIfAbsent** (optional): specifies whether to create an entity when the entity does not exist. That is, whether the entity needs to be imported again. If this parameter is not set, the default value **true** is used.
  - **true**  
If this parameter is set to **true**, the common process is used. Check whether each entity has been imported. If yes, update the properties. If no, create required entities and add corresponding properties for the entities.
  - **false**  
If this parameter is set to **false**, the entity does not need to be imported. You can skip the entity query and property update. If some entities in the

data are not imported before, the entities will not be imported, and the relationships associated with the entities are not imported. If an entity has been imported before, the relationships associated with the entity can still be imported.

Setting this parameter to **false** improves the data import speed and reduces unnecessary query verification. However, you need to set this parameter based on the actual service scenario. In the following two typical scenarios, you are advised to set this parameter to **false**.

- If an exception occurs when data is imported. After the exception is handled, you are advised to import the data again. For example, if an exception occurs when the second entity is imported after the first entity is successfully imported. In this case, you can set this parameter to **false** in the mapping rule of the first entity for the second import. After this configuration, the first entity will be skipped and the data import will start from the second entity where the exception occurred. If the last exception occurs during the relationship import, you can set this parameter to **false** in the mapping rules of all entities for this import. After this configuration, the task will skip the phase of importing entity data, and only the relationship data is imported. In this way, data import speed is greatly improved and the data integrity is not affected.
- Some entities of the data have been imported for other data before. In this case, you must ensure that the entity type IDs of the data exist in the graph and that you do not need to update the property values this time. For example, to import the data relationship table of a transaction, the card numbers in the table have been imported when the card number table is imported. In this case, the card numbers in the transaction exist in the graph, and you do not need to update other property values of the card number. You can set **createIfAbsent** to **false** in the mapping rule of the card number to speed up the import.
- **properties** (mandatory): specifies properties of entities, such as the height and gender of a person. When defining an entity, you must define at least one property as the primary key. Property names, values, and rules must be provided. **propertyName** is defined by users. Property values have multiple types which can be specified by **propertyValueType**. Currently, the following nine types are supported: **single**, **required**, **constant**, **formattedMultiColumn**, **geopoint**, and **geocircle** (mandatory), **geobox**, **mappedMultiColumn**, and **fallback**. If the type is not specified, the default value **single** is used. Multiple property types can be nested. The following uses **ColumnValue** as an example to describe how to nest a property method.
  - **single**  
A column of values in the data set is used as the property values. This is the most common scenario.  
Parameter:  
**index** (mandatory): specifies a column of subscripts corresponding to the property value.
  - **required**

If a property value is defined as **required**, the value cannot be empty. If the value is not defined as **required**, the value can be empty. This definition method needs to be nested with other definition methods.

Parameter:

**Column** (mandatory). The value is **ColumnValue**. This parameter is used to define a value that cannot be empty.

- **constant**

Specifies that the property value is a constant.

Parameter:

**value** (mandatory): The value must be a constant. The value data type can only be a basic type, for example, numeric, string, or Boolean.

- **formattedMultiColumn**

Multiple property values are combined in sequence based on the defined format.

Parameter:

**columns** (mandatory): specifies the definition of multiple values in List[*ColumnValue*] format. Note that the definition sequence is related to the combination sequence.

**format** (mandatory): specifies the combination format. Placeholders are used, for example, %s-%s, indicates that two strings are connected using a hyphen. %d indicates an integer, and %f indicates a floating point number.

- **geopoint**

Specifies the geographical location that indicates the location of a point.

Parameter:

**latitudeColumn** (mandatory): *ColumnValue* Specifies the latitude. The value must be a numeral ranging from -90 to 90.

**longitudeColumn** (mandatory): *ColumnValue* Specifies the longitude. The value must be a numeral ranging from -180 to 180.

- **geocircle**

Specifies a geographical circle, which is determined by the location of a point and its radius.

Parameter:

**latitudeColumn** (mandatory): *ColumnValue* Specifies the latitude. The value must be a numeral ranging from -90 to 90.

**longitudeColumn** (mandatory): *ColumnValue* Specifies the longitude. The value must be a numeral ranging from -180 to 180.

**radiusColumn** (mandatory): *ColumnValue* Specifies the radius. The value must be a numeral. The unit is km.

- **geobox**

Specifies a geographical frame (a trapezoid on a sphere). You can determine the scope of the trapezoid by using the latitude and longitude values in the lower left corner of the rectangle and those in the upper right corner of the trapezoid.

Parameter:

**southWestLatitudeColumn** (mandatory): *ColumnValue* Specifies the latitude of the southwest point. The value must be a numeral ranging from -90 to 90.

**southWestLongitudeColumn** (mandatory): *ColumnValue* Specifies the longitude of the southwest point. The value must be a numeral ranging from -180 to 180.

**northEastLatitudeColumn** (mandatory): *ColumnValue* Specifies the latitude of the northeast point. The value must be a numeral ranging from -90 to 90.

**northEastLongitudeColumn** (mandatory): *ColumnValue* Specifies the longitude of the northeast point. The value must be a numeral ranging from -180 to 180.

- **mappedMultiColumn**

Combines multiple values in sequence to obtain a property value. Based on the defined mapping relationship, if the property value meets the mapping relationship, obtain the mapping value based on rules described in the mapping and use the mapping value as the final property value. If the mapping relationship cannot be found, delete the last property value and then search for the mapping relationship. Repeat this operation until only the first property value is left. If no mapping relationship is found still, the default value **null** is returned.

Parameter:

**keyColumns** (mandatory): specifies the definition of multiple values in List[*ColumnValue*] format. Note that the definition sequence is related to the combination sequence.

**valueMap** (mandatory): The mapping relationship is in the following format: Map<*String*, *String*>. If a combined value does not meet the mapping requirement, the default value **null** is used. If you want to set the property value to a default value when the mapping rules are not met, you can set a null string as the key in the mapping relationship and set value to the default value.

**separator** (optional): specifies the separator. Separators are connectors between properties. **If this parameter is not specified, a colon (:) is used as the separator by default.**

If the property value is empty, the separator is also used. For example, if three values are defined for **keyColumns**, which are A, B, and C, the processing sequence is as follows:

- i. Combine the three values to obtain **A:B:C**.
- ii. Check whether there is a property whose key is **A:B:C** exists in the mapping. If the property exists, the mapping value is returned. If the property does not exist, delete the last value in the combination to obtain **A:B**.
- iii. Check whether there is a property whose key is **A:B** exists in the mapping. If the property exists, the mapping value is returned. If the property does not exist, delete the last value in the combination to obtain **A**.
- iv. Check whether the property whose key is **A** exists in the mapping. If the property does not exist, delete the last value **A**. In this case, an empty string is left.

- v. Check whether there is a null string mapping in the mapping (that is, the default value). If the mapping exists, the default value is returned. If no mapping is found, the **null** value is returned.
- **fallback**

In this definition method, two *ColumnValue* can be defined. One is active, and the other is the standby. If the active value is null or the value matches the corresponding **fallbackIf** condition, the standby value is used.

Parameter:

**Primarycolumn** (mandatory): *Columnvalue*. Specifies the first value. The property value takes precedence over the value of this parameter. Then, the system determines whether to obtain the standby value based on the condition.

**fallbackColumn** (required): *ColumnValue*. Specifies the standby value. When the primary value is empty or the value meets the **fallbackIf** condition, the value is used as the property value.

**fallbackIf** (mandatory): specifies the predicate, that is, the matching criteria. The following code uses Predicate as an example. Example:

```
"fallbackIf": {  
    "op": "stringEq",  
    "value": "unknown"  
}
```

Currently, the following nine basic conditional expression predicates are supported: **and**, **or**, **isNull**, **not**, **eq**, **stringEq**, **emptyStr**, **moreThan**, and **lessThan**.

- **and**

If multiple conditions are matched at the same time, the value is **true**.

Parameter:

**predicates** (mandatory): List[*Predicate*]: Specifies multiple conditions.

- **or**

If any condition is matched, the value is **true**.

Parameter:

**predicates** (mandatory): List[*Predicate*]: Specifies multiple conditions.

- **isNull**

If the value is empty, the value is **true**.

Specifies that no parameter is set.

- **not**

If criteria is not matched, the value is **true**.

Parameter:

**predicate** (mandatory): specifies the predicate, that is, a single condition.

- **eq**

If the values are the same, the value is **true**.

Parameter:

**value** (mandatory): specifies the value for judgment, which can be a basic type value.

- **stringEq**

If the strings are the same, the value is **true**. You can define whether the strings are case sensitive.

Parameter:

**value** (mandatory): specifies the value for judgment, which is a string.

**caseSensitive** (optional): specifies whether the value is case sensitive. The default value is **false** that is case insensitive.

- **emptyStr**

Checks whether the entered string is empty or all characters are spaces.  
Note: Empty strings are not null.

Specifies that no parameter is set.

- **moreThan**

Checks whether the entered value is greater than the specified value.

Parameter:

Value (mandatory): Specifies the value for judgment. The value must be a number, and the entered value must also be a number.

- **lessThan**

Checks whether the entered value is less than the specified value.

Parameter:

Value (mandatory): Specifies the value for judgment. The value must be a number, and the entered value must also be a number.

- **action**

A property rule is used to perform operations on the property value. The parameter is defined as an action. Currently, the following actions are supported:

**assign**: specifies that the property value is updated.

**assignIfAbsent**: specifies if the property does not exist, the property is updated. If the property does not exist, the property is not updated.

**assignIfBigger**: compares the existing value and a new value of the property to obtain the larger value and then updates the value. Only the numeric type and date type are supported.

**assignIfSmaller**: compares the existing value and a new value of the property to obtain the smaller value and then updates the value. Only the numeric type and date type are supported.

**assignSum**: sums up the existing value and a new value of the property, and then updates the value. Only the numeric type is supported.

If the property field is not set to **action**, the default value **assign** is used.

- The rules for filling in **relationships** are as follows:
  - *source* (mandatory): specifies the start entity. The value must be an entity name defined above.
  - *target* (mandatory): specifies the target entity. The value must be an entity name defined above.
  - *labelType* (optional): specifies the label (relationship) definition mode. The available values are **constant** and **columnLookup**. If this parameter is not defined, the default value **constant** is used.

- **constant**  
Specifies that a constant is defined as conceptIRI.  
Parameter:  
**label** (mandatory): specifies the edge label. The value must be an edge label defined in the graph.
  - **columnLookup**  
Defines a value as the label.  
Parameter:  
**labelColumn** (mandatory): specifies the edge label value. The definition method is the same as that for property values described in the previous content. For details, see the related information provided above.
  - **unique** (optional): specifies whether modification is required based on the original relationship. If this parameter is not set, the default value **false** is used.
    - **true**  
If this parameter is set to **true**, the system checks whether the label relationship exists between the source entity and target entity before you import the relationship. If no, add an edge. If yes, update the attribute based on the original relationship. If there are multiple label relationships between two entities, you can set the **uniqueFilter** field to update a relationship. If the field is not set, the attribute of a relationship corresponding to the label is used for update. If all relationships do not meet the condition, you need to add an edge and ignore the update. The **uniqueFilter** field contains **propertyKey** and **propertyValue**, which respectively indicate the key and value of a special attribute of the edge. Multiple attribute keys can be set in the **uniqueFilter** field. Use commas (,) to separate multiple attribute keys. You can set only **propertyKey** but not **propertyValue**.
- The query rule description is as follows:
- i. If **propertyKey** is configured but **propertyValue** is not configured, the corresponding column value in the data is used for query. If the column value is null, keys are used for query.
  - ii. If **propertyKey** and **propertyValue** are both configured but the value is not null, use the value. If the value is null, replace it with the data in the CSV file.
  - iii. If **propertyKey** and **propertyValue** are both configured but the value is null, only keys are used for query.
- The following is a configuration example of the uniqueFilter in common scenarios:
- The configuration of **uniqueFilter** in common scenarios is as follows:
- ```
"uniqueFilter" : {  
    "propertyKey" : "p1,p2"  
}
```
- **false**  
No query operation is performed before the relationship is imported. The relationship of the corresponding label is directly added, and the original relationship is not changed.

- **optimizeUniqueCheck** (optional): specifies whether to optimize the check of **unique**. This parameter is valid only when **unique** is set to **true**. If this parameter is not set, the default value **false** is used.
  - **true**  
If this parameter is set to **true**, the label relationships to be added between the entities are queried in batches, reducing the number of HBase requests and accelerating the query of **unique**.
  - **false**  
If this parameter is set to **false**, the query is performed only when a relationship is added for two entities.

### NOTICE

Note: When an entity has a large number of identical label relationships (for example, the call between a person and another), setting this parameter to **true** may lead to slow query speed because the amount of returned data is too large or even may cause query failure for insufficient memory. In practical use, estimate the number of entity edges based on the service and data. Use this parameter with caution.

- **Properties** (optional): specifies the properties corresponding to the label. The parameter setting rule is the same as that for the entity property.

## Example

Assume that the node input mapping rules are as follows. For the meanings of the mapping rules, see the comments on the right of these rules.

```
{  
    "entities": {  
        "person": { // Defines an entity and is named as person. This name is used only when relationships is defined and it does not need to be defined in the schema.  
            "conceptIRI": "person", // The entity label is person, and conceptIRIType is omitted here. The default value is constant, that is, the value of conceptIRI is directly defined. The other islabel is columnLookup. For details, see the employer entity described below.  
            "primaryKey": "UID", // The entity ID is the property value whose name is UID in properties below. You can use this ID to uniquely identify an entity in the graph.  
            "properties": { // Defines the properties of person. UID and Name are property names, and their values are property values.  
                "UID": {  
                    "type": "required", // If this parameter is set to required, the value cannot be empty. If the value is empty, an error is reported.  
                    "column": {  
                        "index": 0,  
                    }  
                },  
                "Name": {  
                    "type": "formattedMultiColumn", // formattedMultiColumn indicates that Name is composed of the values whose subscript are 1 and 2, respectively, in the format of %s-%s. For example, the values whose subscripts are 1 and 2 are zhang and san, respectively. Name is zhang-san.  
                    "columns": [  
                        {  
                            "index": 1  
                        },  
                        {  
                            "index": 2  
                        }  
                    ]  
                }  
            }  
        }  
    }  
}
```

```
        ],
        "format": "%s-%s"
    },
    "shortName": {
        "type": "mappedMultiColumn",      // mappedMultiColumn indicates that the value of shortName
is composed of the value whose subscript is 1 and the value whose subscript is 2, and the two values are
separated by a colon (:). It is used to check whether the combination result has a mapping in valueMap.
        "keyColumns": [      // If the mapping exists, use the mapping value. For example, if the value
whose subscript 1 is zhang and the value whose subscript is 2 is san, the combination is zhang:san. In
valueMap, the value of zhang:san is zs. Therefore, the value of shortName is zs.
9           {      // If the mapping does not exist, remove the last value of the combination result and then
search for the mapping. For example, if the value whose subscript is 1 is zhao, and the value whose
subscript is 2 is wu, the combination is zhao:wu.
            "index": 1      // If valueMap does not contain zhao:wu, remove wu, and only zhao is left.
valueMap still does not contain the keyword zhao. Delete zhao to obtain an empty string.
            },      // The value corresponding to an empty string in valueMap is no short name. Therefore,
the value of shortName is no short name.
            {      // If the mapping corresponding to an empty string is not defined in valueMap, the value of
shortName is null.
                "index": 2
            }
        ],
        "valueMap": {
            "": "no short name",
            "zhang:san": "zs",
            "li:si": "ls"
        },
        "separator": ":"      // This parameter is optional. If separator is not defined, the default value is a
colon(:).
    },
    "Birth Date": {
        "type": "single",      // single indicates that the value of Birth Date is the value whose subscript is
3. The property value type is single by default, which can be omitted. For example, the following property
Married, its property type is ommited.
        "index": 3
    },
    "Married": {
        "index": 4      // The type single is omitted.
    },
    "Deceased": {
        "type": "constant",      // constant indicates that all Deceased properties are set to false.
        "value": "false"
    },
    "Home Address": {
        "type": "geopoint",      // geopoint indicates that the value of Home Address is the location
corresponding to the combination of the value whose latitude is the value of subscript 5 and the value
whose longitude is the value of subscript 6.
        "latitudeColumn": {
            "index": 5
        },
        "longitudeColumn": {
            "index": 6
        }
    },
    "Home Area": {
        "type": "geocircle",      // geocircle indicates that the value of Home Area is the scope that the
latitude is subscript 7, the longitude is subscript 8, and the radius is subscript 9.
        "latitudeColumn": {
            "index": 7
        },
        "longitudeColumn": {
            "index": 8
        },
        "radiusColumn": {
            "index": 9
        }
    },
    "Country Area": {
        "type": "geobox",      // geobox indicates that the value of Country Area is a geographical area
    }
}
```

with the latitude ranges from the value of subscript 10 to that of subscript 12, the longitude ranges from the value of subscript 11 to that of subscript 13.

```

    "southWestLatitudeColumn": {
        "index": 10
    },
    "southWestLongitudeColumn": {
        "index": 11
    },
    "northEastLatitudeColumn": {
        "index": 12
    },
    "northEastLongitudeColumn": {
        "index": 13
    }
},
"employer": { // Defines an entity named employer
    "conceptIRIType": "columnLookup", // If conceptIRIType is set to columnLookup, a value is defined to indicate the conceptIRI. For example, the value of the subscript 14 is used as the conceptIRI.
        "concept": {"index": 14},
        "primaryKey": "companyName",
        "properties": {
            "companyName": {
                "index": 15
            },
            "locationAddress": {
                "type": "geopoint",
                "latitudeColumn": {
                    "index": 16
                },
                "longitudeColumn": {
                    "index": 17
                }
            },
            "locationPlace": {
                "type": "fallback", // fallback indicates that the value of primaryColumn is used preferentially as the property value. If the value of primaryColumn is empty or meets the fallbackIf condition, the value of fallbackColumn is used.
                    "primaryColumn":{ // If the value of subscript 18 is empty and the value of subscript 19 is guangdong, the value of locationPlace is guangdong.
                        "index": 18 // If the value of subscript 18 is unknown and the value of subscript 19 is guangdong, the fallbackIf condition is met. In this case, the value of locationPlace is guangdong.
                    },
                    // The value of subscript 18 is shenzhen, the value of subscript 19 is guangdong, the value shenzhen is neither empty nor meets the fallbackIf condition. In this case, the value of locationPlace is shenzhen.
                    "fallbackColumn":{
                        "index": 19
                    },
                    "fallbackIf": {
                        "op": "stringEq",
                        "value": "unknown"
                    }
                },
                "Earnings": {
                    "type": "required",
                    "column": {
                        "index": 20
                    }
                }
            },
            "home": {
                "conceptIRI": "location",
                "primaryKey": "house_number",
                "properties": {
                    "Population": {
                        "index": 22
                    }
                }
            }
}

```

```

    },
    "house_number": {
        "type": "required",
        "column": {
            "index": 21
        }
    }
},
"social utility": {
    "conceptIRI": "utility",
    "primaryKey": "userID",
    "properties": {
        "userID": {
            "index": 23
        }
    }
},
"relationships": [
    {
        "source": "person",      // indicates the relationship that the person lives at home.
        "target": "home",
        "label": "livesAt"
    },
    {
        "source": "employer",
        "target": "person",
        "label": "employs",
        "unique": true, // This parameter is optional. If unique is not defined, the default value false is used,
        // indicating that an employs relationship is added. If this parameter is set to true, the full-time property is
        // updated in the original relationship when the employs relationship already exists between employer and
        // person.
        "uniqueFilter":{ //This parameter is optional when unique is set to true. Based on this filter condition,
        // update an edge between the two vertices in the same label. This parameter is left blank when unique is set
        // to false.
            "propertyKey": "place",
            "propertyValue": "xian"
        },
        "properties": {      // Properties are not mandatory. For example, the preceding relationship does not
        // have properties.
            "full-time": {
                "index": 24
            }
        },
        {
            "source": "person",
            "target": "social utility",
            "labelType": "columnLookup", // If labelType is set to columnLookup, a value is defined to
            // indicate the label. For example, the value of the subscript 27 is defined and used as the label.
            "labelColumn": {
                "index": 27
            },
            "properties": {
                "twitter": {
                    "index": 25
                },
                "facebook": {
                    "index": 26
                }
            }
        }
    }
]
}

```

### 2.5.3.3 Importing Data

### 2.5.3.3.1 Importing Data in Batches Using Tools

 CAUTION

Data inconsistency occurs when GraphBase imports and deletes data at the same time. If you are using a data ingestion tool on the client, ensure that no batch deletion is executed at the same time.

The system automatically checks whether there are other asynchronous tasks when tasks are submitted through a REST API. You are advised to use the REST API **graphwriter** to submit batch import tasks.

## Prerequisites

1. Install clients for all components in the cluster.
2. A graph has been created. For details about how to create a graph, see [Creating or Deleting a Graph](#).
3. A schema file (for example, an XML file) has been prepared and uploaded. For details about how to upload a schema file, see [Creating a Schema by Uploading an XML File](#).
4. You have prepared the data file (for example, a CSV file). For details about how to prepare a data file, see the example on the GraphBase client in the `/opt/hadoopclient/GraphBase/examples/person-demo/` directory.  
For details about the data description file (DESC file), see [Compiling a Data Description File \(DESC\)](#).  
The graph mapping rule file (.mapper file) has been uploaded. For details, see [Compiling a Graph Mapping Rule File \(.mapper\)](#).

## Upload related files

1. Upload data files to the HDFS.  
Switch to the `/graphwriter` directory in the installation directory of the GraphBase client.  
`cd /opt/hadoopclient/GraphBase/graphwriter`  
Run the following command to execute environment variables:  
`source /opt/hadoopclient/bigdata_env`  
Log in to the system as a GraphBase user and enter the password (specified by the user).  
`kinit graphtest`  
Create a directory storing HDFS files. (If the directory exists, skip this operation.)  
`hdfs dfs -mkdir /user/graphtest`  
Upload the data files to the `/user/graphtest` directory (for example, **person-demo**) of the HDFS.  
`hdfs dfs -put ..//examples/person-demo/Person.csv ..//examples/person-demo/Person.mapper ..//examples/person-demo/Person.desc /user/graphtest`

The details are shown in the following figure.

```
hadoopNode2:/opt/graphbaseClient/GraphBase/graphwriter # hdfs dfs -ls /user/graphtest
Found 7 items
-rw-r--r--  3 root hadoop      100 2018-08-04 14:47 /user/graphtest/Person.csv
-rw-r--r--  3 root hadoop     740 2018-08-04 14:47 /user/graphtest/Person.csv.mapper
-rw-r--r--  3 root hadoop     368 2018-08-04 14:47 /user/graphtest/Person.desc
```

## Modifying the Configuration File

Set the **RESOURCE** parameter in the **resource.properties** file under **/opt/hadoopclient/GraphBase/graphwriter/conf/**. You can modify the parameter configuration as required or add advanced parameters for Spark. For example: You can add **--conf spark.default.parallelism** and set it to **105**.

When data is imported in batches, the Yarn resource queue in Spark parameters is the default queue. To change the default resource queue, add the **--conf spark.yarn.queue** parameter and set it to the specified queue name in the **RESOURCE** parameter.

## Importing Data

Execute the following script and transfer the *graphName* parameter (*graphName* is user-defined that must exist. If it does not exist, create the graph first.)

```
./bin/graphWriter.sh graphName /user/graphtest/Person.csv /user/graphtest/Person.csv.mapper /user/graphtest/Person.desc
```

Note: The data file path must be an HDFS path that is in the following format:  
*Data file*/*Data mapping file.mapper*/*Metadata file.desc*

If no metadata file exists, the file path is: *Data file*/*Data mapping file.mapper*

After the script is executed, the following information is displayed:

```
hadoopNode2:/opt/graphbaseClient/GraphBase/graphwriter # ./bin/graphWriter.sh graphName /user/graphtest/Person.csv /user/graphtest/Person.csv.mapper /user/graphtest/Person.desc
RESOURCE= --executor-memory 5g --driver-memory 5g --num-executors 5 --executor-cores 5
SERNAME= graphtest
spark.yarn.am.memory is set but does not apply in cluster mode.
spark.yarn.am.cores is set but does not apply in cluster mode.
918-08-06 11:31:00 [WARN] [main] [ ] unable to load native-hadoop library for your platform... using builtin-java classes where applicable | org.apache.hadoop.util.NativeCodeLoader.loadNativeCodeLoader.java:62
918-08-06 11:31:00.482 [WARN] [main] [spark.varn.am.extraJavaOptions will not take effect in cluster mode] org.apache.spark.Logging$class.loogWarning(Logging.scala:71)
```

### NOTE

During data import, if the feature data type (type attribute in attributes) configured in the data description file (.desc file) is different from the data type in the data file (for example, .csv file) to be imported, the batch import tool ignores data of different data types by default and imports only the data of the same type in the data description file and data file. For example, the type of data in a column defined in the data description file is "REAL", as indicated by the following: `{"name": "Attr_0", "type": "REAL", "nominalList": []}`. In the data file to be imported, the data of the corresponding column is the character string "mary". The data type of "mary" is STRING, which is inconsistent with the "REAL" type in the description file. In this case, when the batch import tool is used to import data, the data of this error type is ignored and will not be imported.

### 2.5.3.3.2 Importing Data in Real Time Using Tool

Data import in real time is implemented by using SparkStreaming. After SparkStreaming is started, data is read from the corresponding topic based on the configuration and then imported to GraphBase.

 CAUTION

Data inconsistency occurs when GraphBase imports and deletes data at the same time. If you are using a data ingestion tool on the client, ensure that no batch deletion is executed at the same time.

## Prerequisites

1. Install clients for all components in the cluster.
2. The schema file (that is, the XML file) has been uploaded. For details, see [Creating a Schema by Uploading an XML File](#).
3. The graph mapping rule file (.mapper file) has been uploaded. For details, see [Compiling a Graph Mapping Rule File \(.mapper\)](#).

## Modifying the Configuration File

The configuration file directory is `/opt/hadoopclient/GraphBase/graphstream/conf/`.

1. Modify the **graph-streaming.properties** file.

The parameter configuration is listed in the following table.

| Parameter                            | Description  | Example  |
|--------------------------------------|--|--|
| graph.user.name                      | Specifies the name of the user to be created.  | <b>graphtest</b>                                 |
| spark.streaming.checkpoint.directory | Specifies the HDFS directory of the SparkStreaming checkpoint. If the directory does not exist, create it in advance. (It is recommended that this directory be created separately because a large number of checkpoint files are generated during the program running.) | <b>/user/graphtest/graphstreaming/checkpoint</b> |
| spark.streaming.batch.time           | Specifies the time spent by SparkStreaming on data processing in batches. The unit is s.   | 1  |

| Parameter                        | Description  | Example   |
|----------------------------------|--|---|
| spark.streaming.partition.number | Specifies the number of partitions of the node where SparkStreaming resides.   | 50  |
| kafka.topic.name                 | Specifies the name of the topic where data needs to be consumed.   | Person.csv  |
| kafka.metadata.broker.list       | Specifies the IP address of the node where Kafka metadata resides (manual configuration is not required).  | 10.3.70.133:21005,10.3.71.67:21005,10.3.71.24:21005 |
| csv.mapper.hdfs.file             | Specifies the path of the Mapper file in the HDFS.   | /user/graphtest/graphstreaming/Person.csv.mapper    |
| kafka.save.failed.message        | Specifies whether to save failure messages. The available values are <b>false</b> and <b>true</b> . <b>false</b> indicates that the failure message is not saved, and <b>true</b> indicates the reverse. | false   |
| kafka.save.failed.topic.name     | Specifies that the failure messages are saved as Kafka topic names.  | default   |
| kafka.topic.message.separator    | Specifies a separator of Kafka messages.   | ,   |

## 2. Set the **RESOURCE** parameter.

In **resource.properties**, allocate resources based on YARN resource usage. Ensure that the resources to be allocated do not exceed the total resources of YARN.

```
RESOURCE='--executor-memory 4g --driver-memory 2g --num-executors 5 --executor-cores 5'.
```

When data is imported in batches, the Yarn resource queue in Spark parameters is the default queue. To change the default resource queue, add the **--conf spark.yarn.queue** parameter and set it to the specified queue name in the **RESOURCE** parameter.

## Uploading Related Files

Upload the customized Mapper file to the HDFS.

1. Execute environment variables as user **root** (`/opt/hadoopclient/` is the installation directory of the cluster client).

```
source /opt/hadoopclient/bigdata_env
```

2. Create a directory storing HDFS files. (If the directory exists, skip this operation.)

```
hdfs dfs -mkdir /user/graphtest/graphstreaming
```

3. Upload the MAPPER data file to the `/user/graphtest/graphstreaming` directory of HDFS (Take the **person-demo** directory as an example).

```
hdfs dfs -put /opt/hadoopclient/GraphBase/examples/person-demo/  
Person.csv.mapper /user/graphtest/graphstreaming
```

## Create a topic

The APIs or scripts of Kafka can be used to create topics. For details, see "Managing Kafka Themes" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

## Importing Data

1. Go to the **graphstream** directory.

```
cd /opt/hadoopclient/GraphBase/graphstream
```

2. Execute the following script and transfer the `graphbase` parameter (`graphbase` is user-defined that must exist. If it does not exist, create the graph first. Before you create the graph, ensure that all prerequisites have been completed):

```
./bin/graphStreaming.sh graphbase
```

If you need to customize the path of the configuration file, you can specify a local absolute path for the configuration file and copy the information in **graph-streaming.properties** to the configuration file. If the path is not specified, the **graph-streaming.properties** file in the **conf**/ directory is used as the configuration file by default. Example:

```
./bin/graphStreaming.sh graphbase /opt/hadoopclient/GraphBase/  
graphstream/conf/graph-streaming.properties
```

3. (Optional) Send a message to the **Person.csv** topic.

Send the message by following the following procedure:

- a. Upload data files. If the HDFS directory does not exist, create it in advance.

```
hdfs dfs -put ..//examples/person-demo/Person.csv /user/graphtest/  
graphstreaming/data
```

Run the **vi conf/graph-producer.properties** command to configure the **graph-producer.properties** file in **conf**. The parameter configuration in this file is listed as follows.

| Parameter                  | Description   | Example   |
|----------------------------|---|---|
| graph.user.name            | Specifies the name of the user to be created.   | <b>graphtest</b>                                    |
| kafka.topic.name           | Specifies the name of the topic where data needs to be consumed.  | Person.csv  |
| kafka.metadata.broker.list | Specifies the IP address of the node where Kafka metadata resides (manual configuration is not required). | 10.3.70.133:21005,10.3.71.67:21005,10.3.71.24:21005 |
| csv.data.hdfs.directory    | Specifies the directory for storing data in the HDFS.   | /user/graphtest/graphstreaming/data                 |

- b. Run the following command to send a message to a specified Kafka topic:

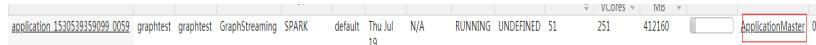
**./bin/graphProducer.sh**

#### BOOK NOTE

This mode is used to test the production data. You can use other methods too.

## Monitoring Tasks

1. Click **ApplicationMaster** on the YARN task page to go to the Spark task page.



2. Click **Streaming**.



3. Check the processing progress.

| Batch Time          | Input Size     | Scheduling Delay [?] | Processing Time [?] | Total Delay [?] |
|---------------------|----------------|----------------------|---------------------|-----------------|
| 2018/07/19 11:43:40 | 3579000 events | 42 min               | 3.6 min             | 45 min          |
| 2018/07/19 11:43:30 | 3690000 events | 38 min               | 3.9 min             | 42 min          |
| 2018/07/19 11:43:20 | 3608500 events | 34 min               | 3.7 min             | 38 min          |
| 2018/07/19 11:43:10 | 3294500 events | 31 min               | 3.4 min             | 35 min          |
| 2018/07/19 11:43:00 | 3675000 events | 27 min               | 3.9 min             | 31 min          |
| 2018/07/19 11:42:50 | 3673500 events | 24 min               | 3.8 min             | 28 min          |

## Tuning Parameters

- The product of the number of executors and the number of cores of each executor determines the task concurrency capability. The value of **spark.streaming.partition.number** determines the total number of tasks that can be started. If the resources are sufficient, you can set the executor and partition quantities to the same.
- spark.streaming.batch.time** of SparkStreaming and the data traffic of Kafka messages determine the amount of data that can be processed by

SparkStreaming at a certain time. The data is distributed on each partition. The data traffic value is negatively correlated with the batchtime value. If the data traffic is small, set **batchtime** to a larger value. Otherwise, set **batchtime** to a small value. In this way, the amount of data to be processed can be ensured moderate.

## Stopping a Job

After a SparkStreaming task is submitted, it keeps running and reads messages from Kafka. To stop the task, perform the following operations: Obtain the submitted task ID on the YARN web client. Run the **yarn application -kill \${appid}** command to kill the task. Example: **yarn application -kill \$Application\_1530539359099\_0053** If no data is queried after the data import, kill the task and collect logs. Search for **ERROR** in the logs, and identify the cause. Run the following command to collect logs:

```
yarn logs -applicationId ${appid} > log
```

### 2.5.3.3.3 Using the Hive Data Import Tool

---

 CAUTION

Data inconsistency occurs when GraphBase imports and deletes data at the same time. If you are using a data ingestion tool on the client, ensure that no batch deletion is executed at the same time.

---

## Prerequisites

1. Install clients for all components in the cluster.
2. Create a graph by referring to [Creating or Deleting a Graph](#).
3. Prepare and upload a schema file (for example, an XML file) by referring to [Creating a Schema by Uploading an XML File](#).
4. The Hive component has been installed in the cluster.
5. Prepare a Hive table. The field names in the figure can be different from those in the Hive table. Compile a graph mapping rule file (**.mapper** file) based on the data columns of vertices and edges in Hive. For details, see [Compiling a Graph Mapping Rule File \(.mapper\)](#).
6. For details about how to create Hive data, see section [Developing an Application](#).

## Modify the configuration file

The configuration file directory is **/opt/hadoopclient/GraphBase/graphwriter/conf/**.

1. Modify the **hiveTable.properties** file.

The parameter configuration is listed in the following table.

| Parameter        | Description   | Example       |
|------------------|---|---------------|
| table.namespace  | Indicates the namespace where the Hive table is located. Retain the <b>default</b> value.   | default       |
| table.tablename  | Indicates the Hive table name.  | table1        |
| table.columns    | Indicates the columns to be selected in the table. By default, the asterisk (*) indicates a global column. Multiple column names are separated by commas (,). | * or name,age |
| table.conditions | Indicates that you can write a condition expression to select the column that meets the condition. The format is the same as that of the WHERE clause.        | age=20        |

## 2. Set the **RESOURCE** parameter.

In **resource.properties**, allocate resources based on YARN resource usage. Ensure that the resources to be allocated do not exceed the total resources of YARN.

`RESOURCE="--executor-memory 4g --driver-memory 2g --num-executors 5 --executor-cores 5"`

When data is imported in real time, the YARN resource queue in Spark parameters is the default queue. To change the default resource queue, add the **--conf spark.yarn.queue=default** parameter and set it to the specified queue name in the **RESOURCE** parameter.

## Import data

Run the following script that contains the graph name (*graphName* is a user-defined graph name and must exist. If the graph name does not exist, create it first.) and the mapper file path (the mapper file path must be an HDFS path).

`./bin/hiveWriter.sh graphName /user/graphtest/xxx.mapper`

Example description:

```
CREATE EXTERNAL TABLE IF NOT EXISTS table1  
(
```

```
sourceld STRING,  
targetId STRING,  
attr1 STRING,  
attr2 STRING  
)  
ROW FORMAT delimited fields terminated by ','  
STORED AS TEXTFILE;
```

Create the preceding Hive table. **sourceld** corresponds to the primary key attribute of the start vertex, and **targetId** corresponds to the primary key attribute of the end vertex. If the start vertex and end vertex are not of the same type (for example, the start vertex is an ID card number and the end vertex is a mobile number), **entities** in the mapper file are defined as follows:

```
"entities": {  
  "entity1": {  
    "conceptIRI": "person",  
    "primaryKey": "idCard",  
    "properties": {  
      "name": {  
        "index": 0  
      }  
    }  
  },  
  "entity2": {  
    "conceptIRI": "phone",  
    "primaryKey": "phoneNum",  
    "properties": {  
      "name": {  
        "index": 1  
      }  
    }  
  }  
}
```

If the start vertex and end vertex are of the same type (for example, the start vertex is a mobile number and the end vertex is a mobile number), entities in the mapper file are defined as follows:

```
"entities": {  
  "entity1": {  
    "conceptIRI": "phone",  
    "primaryKey": "phoneNum",  
    "properties": {  
      "name": {  
        "index": 0  
      }  
    }  
  },  
  "entity2": {  
    "conceptIRI": "phone",  
    "primaryKey": "phoneNum",  
    "properties": {  
      "name": {  
        "index": 1  
      }  
    }  
  }  
}
```

#### 2.5.3.4 Data Export



##### NOTE

When exporting data, check whether the specified HDFS path exists. If it does not, check whether the default HDFS path exists. If it does, delete the path. Otherwise, an error is reported and the task fails.

## Prerequisites

Install clients for all components in the cluster.

## Configuring Data Collection

The mapper file is used to collect the parameter information required during the export. The following figure shows the relationships between the vertex with label set to Person and Tag and the edge with label set to hasInterest:



The mapper file of the Person vertex is as follows:

```
{  
    "entities": {  
        "entity1": {  
            "conceptIRI": "Person",           ----- This item specifies label (type) of the vertex.  
            "primaryKey": "personId",       ----- This item specifies primaryKey of the vertex.  
            "properties": {  
                "personId": {  
                    "index": 0                  ----- This item specifies the attribute key of the vertex.  
                },  
                "firstName": {  
                    "index": 1                  ----- This item specifies the attribute key of the vertex.  
                },  
                "lastName": {  
                    "index": 2                  ----- This item specifies the attribute key of the vertex.  
                },  
                "gender": {  
                    "index": 3                  ----- This item specifies the attribute key of the vertex.  
                },  
                "birthDay": {  
                    "index": 4                  ----- This item specifies the attribute key of the vertex.  
                },  
                "creationDate": {  
                    "index": 5                  ----- This item specifies the attribute key of the vertex.  
                },  
                "locationIP": {  
                    "index": 6                  ----- This item specifies the attribute key of the vertex.  
                },  
                "browserUsed": {  
                    "index": 7                  ----- This item specifies the attribute key of the vertex.  
                }  
            }  
        }  
    }  
}
```

The mapper file of the Tag vertex is as follows:

```
{  
    "entities": {  
        "entity1": {  
            "conceptIRI": "Tag",           ----- This item specifies label (type) of the vertex.  
            "primaryKey": "tagId",         ----- This item specifies primaryKey of the vertex.  
            "properties": {  
                "tagId": {  
                    "index": 0                  ----- This item specifies the attribute key of the vertex.  
                },  
                "name": {  
                    "index": 1                  ----- This item specifies the attribute key of the vertex.  
                }  
            }  
        }  
    }  
}
```

```

        "url": {
          "index": 2
        }
      }
    }
  }
}
----- This item specifies the attribute key of the vertex.

```

The mapper file of the hasInterest edge is as follows:

```

{
  "entities": {
    "sourceEntity": {
      "conceptIRI": "Person",
      "primaryKey": "personId",
      "createIfAbsent": false,
      "properties": {
        "personId": {
          "index": 0
        }
      }
    },
    "targetEntity": {
      "conceptIRI": "Tag",
      "primaryKey": "tagId",
      "createIfAbsent": false,
      "properties": {
        "tagId": {
          "index": 1
        }
      }
    }
  },
  "relationships": [
    {
      "source": "sourceEntity",
      "target": "targetEntity",
      "labelType": "constant",
      "label": "hasInterest"
    }
  ]
}
----- This item specifies the attribute of the outvertex.
----- This item specifies the label (type) of the outvertex.
----- This item specifies the primaryKey of the outvertex.
----- This item specifies the attribute key of the outvertex.
----- This item specifies the attribute of the invertex.
----- This item specifies the label (type) of the invertex.
----- This item specifies the primaryKey of the invertex.
----- This item specifies the attribute key of the invertex.
----- This item specifies the entity of the outvertex.
----- This item specifies the entity of the invertex.
----- This item specifies the label (type) of the edge

```

## Modifying Configuration Files

The configuration file directory is **/opt/hadoopclient/GraphBase/graphreader/conf/**.

### 1. Modify the **vertex.properties** file.

The parameter configurations are listed in the following table.

| Parameter | Definition  | Example   |
|-----------|---|-----------|
| graphname | Name of the graph instance to be operated. This parameter is mandatory. | graphbase |
| label     | Type of the vertex to be exported. This parameter is mandatory.         | Forum     |

| Parameter         | Definition  | Example                    |
|-------------------|---|----------------------------|
| propertykeylist   | Property key list of the vertex to be exported. This parameter is mandatory. The primary key must be specified. If multiple properties need to be exported, separate the property keys by commas (,). The primary key must be placed in the first place.  | forumId,title,creationDate |
| dest.storage.path | HDFS path for storing the data to be exported. This parameter is optional and not recommended. Assume that the current domain name is HADOOP.COM, by default, the data is stored in <b>/user/graphtest@HADOOP.COM/graphbase/vertex/Person</b> . In the preceding directory, <b>graphtest</b> indicates the user name, <b>graphbase</b> indicates the graph name, and <b>Person</b> indicates the vertex type. | /user/vertex               |

| Parameter | Definition  | Example   |
|-----------|---|---|
| delimiter | Separator between properties of the data to be exported. The separator can be a common character, a special character, or a unicode. The separators specified for exporting the incoming and outgoing vertices must be the same. If you do not configure this parameter, commas (,) are used as the separators by default. This parameter is optional.    | \u002a  |
| filters   | Filtering conditions of properties in the data to be exported. The data format is JSONArray. For details, see the description for <i>PropertyFilter</i> in section "GraphBase" > "REST" > "Performing Full Graph Query on Vertices" in <i>MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)</i> . This parameter is optional | [<br>{"propertyName":"age","predicate":">>","values":[27]},<br>{"propertyName":"name","predicate":string_contains _prefix","values":["m"]}] |

2. Modify the **edge.properties** file.

The parameter configuration is listed in the following table.

| Parameter | Definition  | Example   |
|-----------|---|-----------|
| graphname | Name of the diagram instance to be operated. This parameter is mandatory. | graphbase |

| Parameter            | Definition  | Example   |
|----------------------|---|-----------|
| label                | Type of the edge to be exported. This parameter is mandatory.   | hasMember |
| outvertex.label      | Type of the outvertex at the two vertices corresponding to the edge to be exported. This parameter is mandatory.  | Forum     |
| invertex.label       | Type of the invertex at the two vertices corresponding to the edge to be exported. This parameter is mandatory.   | Person    |
| outvertex.primarykey | The primary key of the incoming vertex of the edge to be exported. This parameter is mandatory.   | forumId   |
| invertex.primarykey  | The primary key of the outgoing vertex of the edge to be exported. This parameter is mandatory.   | personId  |
| propertykeylist      | Attribute key list of the edge to be exported. This parameter is optional. If multiple attributes need to be exported, use commas (,) to separate all attribute keys. | joinDate  |

| Parameter         | Definition   | Example    |
|-------------------|--|------------|
| dest.storage.path | HDFS path for storing the data to be exported. This parameter is optional and not recommended. Assume that the current domain name is HADOOP.COM, by default, the data is stored in <b>/user/graphtest@HADOO P.COM/graphbase/edge/friend</b> . In the preceding path, <b>graphtest</b> is the created username, <b>graphbase</b> is the graph name, and <b>friend</b> is the type of the edge. | /user/edge |
| delimiter         | Separator between attributes of the data to be exported. The separator can be a common character, a special character, or a unicode. The separators specified for exporting the incoming and outgoing vertices must be the same. If you do not configure this parameter, commas (,) are used as the separators by default. This parameter is optional.   | \uu02a     |

| Parameter | Definition   | Example   |
|-----------|--|---|
| filters   | Filtering conditions of properties in the data to be exported. The data format is JSONArray. For details, see the description for <i>PropertyFilter</i> in section "GraphBase" > "REST" > "Performing Full Graph Query on Vertices" in <i>MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)</i> . This parameter is optional. | [<br>{"propertyName":"age","predicate":">>","values":[27]},<br>{"propertyName":"name","predicate":}"string_contains _prefix","values":["m"]}<br>] |

3. Modify the **resource.properties** file.

The parameter configurations are listed in the following table.

| Parameter             | Definition   | Example   |
|-----------------------|--|---|
| BINDING_SPARK_SERVICE | Bound Spark service, which is used to adapt to multiple instances. | Spark   |
| RESOURCE              | --executor-memory  | Memory size allocated to each executor in Spark |
|                       | --driver-memory  | Memory size allocated to driver in Spark        |
|                       | --num-executors  | Number of executors started by Spark            |
|                       | --executor-cores   | Number of cores started in each executor        |

| Parameter | Definition  | Example |
|-----------|---|---------|
|           | --conf spark.hbase.obtainToken.enabled<br>Whether to dynamically obtain the token information of the HBase. | true    |
|           | --conf spark.inputFormat.cache.enabled<br>Whether to enable the HBase cache                                 | false   |

### NOTE

The following special characters can be used as separators.

\u0020 Space  
\u0021 ! Exclamation mark  
\u0022 " Double quotation marks  
\u0023 # Pond key  
\u0024 \$ Currency symbol  
\u0025 % Percentage symbol  
\u0026 & And sign  
\u0027 ' Single quotation mark  
\u0028 ( Open parenthesis  
\u0029 ) Close parenthesis  
\u002a \* Asterisk  
\u002b + Plus sign  
\u002c , Comma  
\u002d - Hyphen

## Parameter Optimization

### 1. Parameter Description

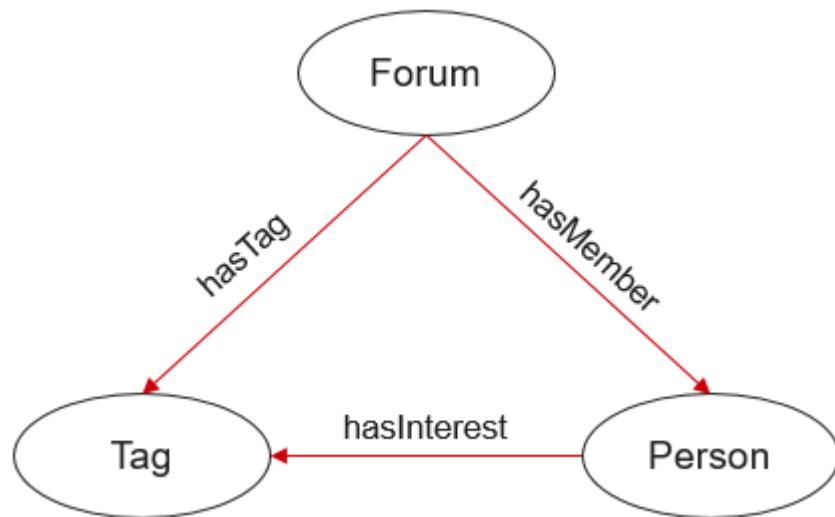
- In the **RESOURCE** configuration, the product of **num-executors** and **executor-cores** is the total number of **executor-core**, which determines the task concurrency capability.
- The product of the number of partitions and the number of nodes in the configuration item **graph.regions-per-server** in GraphBase is the total number of partitions, which determines the total number of started tasks.
- In the **RESOURCE** configuration, the product of **num-executors** and **executor-memory** is the total memory required by the task. The **yarn.nodemanager.resource.memory-mb** configuration item in Yarn determines the total memory that can be used.
- In the RESOURCE configuration, the value of **executor-memory** divided by **executor-cores** is the memory that can be used by each executor-core.

### 2. Parameter Optimization

- If the resources are sufficient, you can set the total number of executor-cores to be the same as the total number of partitions. In practice, you are advised to set **executor-cores** to a smaller value and increase the value of **num-executors**.
- The total memory configured for a task cannot exceed the total memory configured for Yarn, preventing memory overflow.
- If the memory is sufficient, you can increase the value of **executor-memory** to increase the memory size of each executor-core. You can also increase the value of **spark.yarn.executor.memoryOverhead** to dynamically increase executor memory to prevent performance deterioration caused by long GC time.

## Exporting Data

For example, the graph name is **graphbase** and the username is **graphtest**. The following figure shows the relationships between the vertex with label set to Forum, Person and Tag and the edge with label set to hasMember, hasInterest and hasTag.



1. Export the vertex.

Access the **graphreader** directory:

```
cd /opt/hadoopclient/GraphBase/graphreader
```

Run the following command to execute the environment variable:

```
source /opt/hadoopclient/bigdata_env
```

- Export a vertex whose type is Forum.

Copy the configuration file:

```
cp conf/vertex.properties conf/Forum.properties
```

Modify the **Forum.properties** configuration file.

```
vim conf/vertex.properties
```

For example:

```
propertykeylist = forumId,title,creationDate  
dest.storage.path =  
graphname = graphbase  
label = Forum
```

```
delimiter = \u002a  
filters =
```

Run the script to export data:

**./bin/graphReader.sh vertex Forum.properties**

If the command is executed successfully, the following information is displayed:

```
[root@8-5-199-1 graphreader]# ./bin/graphReader.sh vertex Forum.properties  
-----  
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2  
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false  
graphName=graphbase  
type=vertex  
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/Forum.properties  
  
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your  
platform... using builtin-java classes where applicable |  
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)  
spark.yarn.am.memory is set but does not apply in cluster mode.  
spark.yarn.am.cores is set but does not apply in cluster mode.  
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in  
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)  
[root@8-5-199-1 graphreader]#
```

- Export the vertex whose type is Person.

Copy the configuration file:

**cp conf/vertex.properties conf/Person.properties**

Modify the **Person.properties** configuration file:

**vim conf/Person.properties**

For example:

```
propertykeylist = personId,firstName,lastName,gender,birthDay  
dest.storage.path =  
graphname = graphbase  
label = Person  
delimiter = \u002a  
filters =
```

Run the script to export data:

**./bin/graphReader.sh vertex Person.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh vertex Person.properties  
-----  
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2  
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false  
graphName=graphbase  
type=vertex  
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/Person.properties  
  
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your  
platform... using builtin-java classes where applicable |  
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)  
spark.yarn.am.memory is set but does not apply in cluster mode.  
spark.yarn.am.cores is set but does not apply in cluster mode.  
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in  
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)  
[root@10-5-199-1 graphreader]#
```

- Export the vertex whose type is Tag.

Copy the configuration file:

**cp conf/vertex.properties conf/Tag.properties**

Modify the **Tag.properties** configuration file:

**vim conf/Tag.properties**

For example:

```
propertykeylist = tagId,name,url  
dest.storage.path =  
graphname = graphbase  
label = Tag  
delimiter = \u002a  
filters =
```

Run the script to export data:

**./bin/graphReader.sh vertex Tag.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh vertex Tag.properties  
-----  
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2  
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false  
graphName=graphbase  
type=vertex  
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/Tag.properties  
-----  
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your  
platform... using builtin-java classes where applicable |  
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)  
spark.yarn.am.memory is set but does not apply in cluster mode.  
spark.yarn.am.cores is set but does not apply in cluster mode.  
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in  
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)  
[root@10-5-199-1 graphreader]#
```

## 2. Export the edge.

Access **graphreader** directory:

**cd /opt/hadoopclient/GraphBase/graphreader**

Run the following command to execute the environment variable:

**source /opt/hadoopclient/bigdata\_env**

- Export the edge whose type is hasMember.

Copy the configuration file:

**cp conf/edge.properties conf/hasMember.properties**

Modify the **hasMember.properties** configuration file.

**vim conf/hasMember.properties**

For example:

```
invertex.primarykey = personId  
outvertex.primarykey = forumId  
propertykeylist =  
invertex.label = Person  
dest.storage.path =  
graphname = graphbase  
outvertex.label = Forum  
label = hasMember  
delimiter = \u002a  
filters =
```

Run the script to export data:

**./bin/graphReader.sh edge hasMember.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh edge hasMember.properties
-----
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false
graphName=graphbase
type=edge
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/hasMember.properties

2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable |
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)
spark.yarn.am.memory is set but does not apply in cluster mode.
spark.yarn.am.cores is set but does not apply in cluster mode.
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
[root@10-5-199-1 graphreader]#
```

- Export the edge whose type is hasInterest.

Copy the configuration file:

**cp conf/edge.properties conf/hasInterest.properties**

Modify the **hasInterest.properties** configuration file:

**vim conf/hasInterest.properties**

For example:

```
invertex.primarykey = tagId
outvertex.primarykey = personId
propertykeylist =
invertex.label = Tag
dest.storage.path =
graphname = graphbase
outvertex.label = Person
label = hasInterest
delimiter = \u002a
filters =
```

Run the script to export data:

**./bin/graphReader.sh edge hasInterest.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh edge hasInterest.properties
-----
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false
graphName=graphbase
type=edge
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/hasInterest.properties

2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable |
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)
spark.yarn.am.memory is set but does not apply in cluster mode.
spark.yarn.am.cores is set but does not apply in cluster mode.
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
[root@10-5-199-1 graphreader]#
```

- Export the edge whose type is hasTag.

Copy the configuration file:

**cp conf/edge.properties conf/hasTag.properties**

Modify the **hasTag.properties** configuration file:

**vim conf/hasTag.properties**

For example:

```
invertex.primarykey = tagId
outvertex.primarykey = forumId
propertykeylist =
invertex.label = Tag
dest.storage.path =
graphname = graphbase
outvertex.Label = Forum
label = hasTag
delimiter = \u002a
filters =
```

Run the script to export data:

**./bin/graphReader.sh edge hasTag.properties**

If the command is executed successfully, the following information is displayed:

```
[root@10-5-199-1 graphreader]# ./bin/graphReader.sh edge hasTag.properties
```

```
resourceInfo=--executor-memory 2g --driver-memory 2g --num-executors 20 --executor-cores 2
--conf spark.hbase.obtainToken.enabled=true --conf spark.inputFormat.cache.enabled=false
graphName=graphbase
type=edge
propertyFile=/opt/hadoopClient/GraphBase/graphreader/conf/hasTag.properties
```

```
2019-03-07 10:26:36,684 | WARN | main | Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable |
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)
spark.yarn.am.memory is set but does not apply in cluster mode.
```

```
spark.yarn.am.cores is set but does not apply in cluster mode.
2019-03-07 10:26:49,315 | WARN | main | spark.yarn.am.extraJavaOptions will not take effect in
cluster mode | org.apache.spark.Logging$class.logWarning(Logging.scala:71)
```

```
[root@10-5-199-1 graphreader]#
```

## Viewing Execution Results

- Check the execution process on Yarn.
  - Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Yarn**, and select **ResourceManager (Active)** in **Basic Information** on the **Overview** page. The Yarn WebUI is displayed.

The screenshot shows the Yarn ResourceManager Overview page. It displays a table of applications, each with a task ID, user, queue, application name, state, and other metrics like memory used and progress. There are four visible tasks, all in the 'RUNNING' state.

| Task ID                        | User | Queue   | Application Name        | State   | Memory Used (MB) | Progress (%) |
|--------------------------------|------|---------|-------------------------|---------|------------------|--------------|
| application_1552331280773_0001 | root | default | graphbase_1552331280773 | RUNNING | 1024             | 100          |
| application_1552331280773_0002 | root | default | graphbase_1552331280773 | RUNNING | 1024             | 100          |
| application_1552331280773_0003 | root | default | graphbase_1552331280773 | RUNNING | 1024             | 100          |
| application_1552331280773_0004 | root | default | graphbase_1552331280773 | RUNNING | 1024             | 100          |

- Click the task ID to view the task overview.

The screenshot shows the Yarn Application Overview page for task application\_1552331280773\_0001. It provides detailed information about the application, including its configuration, logs, and resource usage.

**Application Configuration:**

- Queue: default
- Queue Root: /
- Application Type: MapReduce
- Application Name: graphbase\_1552331280773
- Application ID: application\_1552331280773\_0001
- Non-Application ID: 1552331280773
- Final Status: FINISHED
- Final Application State: FINISHED
- Final Application Type: MAPREDUCE
- Final Application Name: graphbase\_1552331280773
- Final Application ID: application\_1552331280773\_0001
- Final Non-Application ID: 1552331280773

**Application Metrics:**

- Total Resource Requested (memory): 4096 MB
- Total Number of AM Requests Preempted: 0
- Resource Requested from Current Attempt: 1024 MB
- Number of Non-AM Requests Preempted: 0
- Number of Non-AM Requests from Current Attempt: 0
- Aggregate Resource Allocation: 1024 MB via 100 seconds, 100 msec

- c. Click **ApplicationMaster** to view the task execution progress.



2. Check the exported data on HDFS.

Log in to FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > HDFS**, and select **NameNode (Active)** in **Basic Information** on the **Overview** page. On the HDFS WebUI that is displayed, choose **Utilities > Browse the file system** to go to the root directory.

- Go to the Person vertex export directory and check the Person data.



#### Browse Directory

| Permission | Owner | Group  | Size     | Last Modified            | Replication | Block Size | Name       |
|------------|-------|--------|----------|--------------------------|-------------|------------|------------|
| -rwxr--r-- | dbc   | hadoop | 0 B      | Tue Mar 12 07:29:37 2019 | 3           | 128 MB     | _SUCCESS   |
| -rwxr--r-- | dbc   | hadoop | 16.23 KB | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00000 |
| -rwxr--r-- | dbc   | hadoop | 17.5 KB  | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00001 |
| -rwxr--r-- | dbc   | hadoop | 18.39 KB | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00002 |
| -rwxr--r-- | dbc   | hadoop | 19.43 KB | Tue Mar 12 07:29:35 2019 | 3           | 128 MB     | part-00003 |
| -rwxr--r-- | dbc   | hadoop | 14.3 KB  | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00004 |
| -rwxr--r-- | dbc   | hadoop | 14.37 KB | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00005 |
| -rwxr--r-- | dbc   | hadoop | 15.02 KB | Tue Mar 12 07:29:35 2019 | 3           | 128 MB     | part-00006 |
| -rwxr--r-- | dbc   | hadoop | 15.29 KB | Tue Mar 12 07:29:35 2019 | 3           | 128 MB     | part-00007 |
| -rwxr--r-- | dbc   | hadoop | 16.77 KB | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00008 |
| -rwxr--r-- | dbc   | hadoop | 15.49 KB | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00009 |
| -rwxr--r-- | dbc   | hadoop | 22.1 KB  | Tue Mar 12 07:29:31 2019 | 3           | 128 MB     | part-00010 |

- Go to the Tag vertex export directory and check Tag data.



#### Browse Directory

| Permission | Owner | Group  | Size     | Last Modified            | Replication | Block Size | Name       |
|------------|-------|--------|----------|--------------------------|-------------|------------|------------|
| -rwxr--r-- | dbc   | hadoop | 0 B      | Tue Mar 12 07:53:05 2019 | 3           | 128 MB     | _SUCCESS   |
| -rwxr--r-- | dbc   | hadoop | 20.91 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00000 |
| -rwxr--r-- | dbc   | hadoop | 21.13 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00001 |
| -rwxr--r-- | dbc   | hadoop | 29.84 KB | Tue Mar 12 07:53:17 2019 | 3           | 128 MB     | part-00002 |
| -rwxr--r-- | dbc   | hadoop | 26.68 KB | Tue Mar 12 07:53:31 2019 | 3           | 128 MB     | part-00003 |
| -rwxr--r-- | dbc   | hadoop | 22.78 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00004 |
| -rwxr--r-- | dbc   | hadoop | 21.36 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00005 |
| -rwxr--r-- | dbc   | hadoop | 21.89 KB | Tue Mar 12 07:53:31 2019 | 3           | 128 MB     | part-00006 |
| -rwxr--r-- | dbc   | hadoop | 22.34 KB | Tue Mar 12 07:53:31 2019 | 3           | 128 MB     | part-00007 |
| -rwxr--r-- | dbc   | hadoop | 21.95 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00008 |
| -rwxr--r-- | dbc   | hadoop | 25.27 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00009 |
| -rwxr--r-- | dbc   | hadoop | 28.73 KB | Tue Mar 12 07:53:18 2019 | 3           | 128 MB     | part-00010 |

- Go to the hasInterest edge export directory and check the hasInterest data.

| Browse Directory                   |       |        |           |                          |             |            |            |
|------------------------------------|-------|--------|-----------|--------------------------|-------------|------------|------------|
| /user/dbc/graphbase_17/edge/hadoop |       |        |           |                          |             |            | Go!        |
| Permission                         | Owner | Group  | Size      | Last Modified            | Replication | Block Size | Name       |
| -r--r--r--                         | dbc   | Hadoop | 0 B       | Tue Mar 12 08:22:55 2019 | 3           | 128 MB     | _SUCCESS   |
| -r--r--r--                         | dbc   | Hadoop | 6542 KB   | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00000 |
| -r--r--r--                         | dbc   | Hadoop | 72.06 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00001 |
| -r--r--r--                         | dbc   | Hadoop | 47.11 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00002 |
| -r--r--r--                         | dbc   | Hadoop | 70.57 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00003 |
| -r--r--r--                         | dbc   | Hadoop | 57.21 KB  | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00004 |
| -r--r--r--                         | dbc   | Hadoop | 80.04 KB  | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00005 |
| -r--r--r--                         | dbc   | Hadoop | 72.99 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00006 |
| -r--r--r--                         | dbc   | Hadoop | 100.39 KB | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00007 |
| -r--r--r--                         | dbc   | Hadoop | 67.7 KB   | Tue Mar 12 08:22:50 2019 | 3           | 128 MB     | part-00008 |
| -r--r--r--                         | dbc   | Hadoop | 60.81 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00009 |
| -r--r--r--                         | dbc   | Hadoop | 76.42 KB  | Tue Mar 12 08:22:49 2019 | 3           | 128 MB     | part-00000 |

### NOTE

Data export formats of the Date and Geoshape are as follows:

- Date

During the import, the data may be the time of the long type:  
1552894952164 or time of the date type: Mon Mar 18 15:43:34 GMT+08:00  
2019.

During the export, the format is 2016-03-18 15:45:18.

- Geoshape

Data to be imported may be longitude and latitude: 23,52 (longitude,  
latitude) or a circle area: 23,52,18 (longitude,latitude,radius).

After the data export, if a comma (,) is used as the separator, the data format  
remains unchanged: 23,52 (longitude, longitude) or 23,52,18  
(longitude,longitude,radius).

## 2.5.3.5 REST API Development

REST APIs are provided by GraphBase 8.3.1 and used by developers for secondary  
development based on the GraphBase platform.

The GraphBase API provides unified access for upper-layer applications and  
encapsulates the unified operations performed on the GraphBase platform  
through highly flexible REST APIs.

When an HTTP requester sets up a connection to the server and sends an HTTP  
request, no other requests can be sent from the connection until responses to the  
sent request are received.

## 2.5.3.6 REST API Invocation Examples and Running Results

### 2.5.3.6.1 Example

This section describes an example of creating a PropertyKey API. For details about  
how to create other APIs, see **GraphBase in MapReduce Service (MRS) 3.3.1-LTS**  
**API Reference (for Huawei Cloud Stack 8.3.1)**. The code snippet is in the  
GraphBaseRestExample class in the **com.huawei.util** package of the sample  
project. After code is run, the result is returned on the console.

 NOTE

1. Ensure that the input parameters and the invoked URL are correct.
2. Ensure that the entered IP address, port number, username, and password are correct.

1. Initialize the configuration of **graphbase.properties**. For details, see [Preparing for Security Authentication](#).

2. Obtain the connection information and perform security authentication.

```
GraphHttpClient client = null;
InputStream inputStream = null;
inputStream = new FileInputStream(System.getProperty("user.dir") + File.separator +
DEFAULT_CONFIG_FILE);
Properties p = new Properties();
p.load(inputStream);
HttpAuthInfo httpAuthInfo = HttpAuthInfo.newBuilder()
    .setIp(p.getProperty("ip"))
    .setPort(Integer.valueOf(p.getProperty("port")))
    .setService(RestApi.SERVICE)
    .setUsername(p.getProperty("userName"))
/*.setPassword(p.getProperty("password"))*/
    .setKeytabFile(System.getProperty("user.dir") + File.separator + KEY_TAB)
    .build();
client = GraphHttpClient.newKeytabClient(httpAuthInfo);
RestApi api = new RestApi(client);
```

3. Prepare the metadata-related classes, including PropertyKey, EdgeLabel, and VertexLabel in **com.huawei.entity** of the sample code.

4. Create the related REST APIs, for example, a property key API. (For details, see the sample project **com.huawei.graphbase.RestApi**.)

```
//Create a propertyKey interface.
public boolean addPropertyKey(PropertyKey propertyKey, String graphName) {
    JSONObject rspJson = null;
    try {
//Invoke the REST API URL and transfer the request parameter propertyKey.
        rspJson = sendHttpPostReq("/graph/" + graphName + "/schema/property-key",
RestHelper.toJsonString(propertyKey));
    } catch (JsonProcessingException e) {
        LOG.info(ERROR_MSG);
    }
//Parse the returned rspJson parameter.
    final boolean isSuccessfully = (null == rspJson ? false : rspJson.getString("code").equals("0"));
//Print logs to the console.
    String log = String.format("[%-s]: create property key[%s] %s",
        (isSuccessfully ? "SUCCESS" : "FAIL"),
        propertyKey.getName(),
        (isSuccessfully ? "successfully." : "fail."));
    System.out.println(log);
    return isSuccessfully;
}
//Send a request containing CSRF_TOKEN.
private JSONObject sendHttpPostReq(String uri, JSONObject reqJson) {
    return sendHttpPostReq(uri, reqJson.toString());
}

private JSONObject sendHttpPostReq(String uri) {
    return sendHttpPostReq(uri, "");
}

private JSONObject sendHttpPostReq(String uri, String reqJsonStr) {
    String fullUrl = buildUrl(uri);
    HttpPut httpPut = new HttpPut(fullUrl);
    httpPut.addHeader(CSRF_TOKEN, this.client.csrfToken);

    CloseableHttpResponse response = null;
    JSONObject rspJson = null;
```

```
try {
    if (reqJsonStr != null && !reqJsonStr.isEmpty()) {
        httpPut.setEntity(new StringEntity(reqJsonStr, ContentType.create("text/plain",
Consts.UTF_8)));
    }
    printHttpReqHeader(httpPut);
    printHttpReqBody(reqJsonStr);

    response = client.httpClient.execute(httpPut);
    RestHelper.checkHttpRsp(response);

    rspJson = new JSONObject(EntityUtils.toString(response.getEntity(), Charset.forName("UTF-8")));
    printHttpRspBody(rspJson);
} catch (Exception e) {
    LOG.info(e.getMessage());
    return null;
} finally {
    if (null != response) {
        try {
            response.close();
        } catch (IOException e) {
            // nothing to do
        }
    }
}
return rspJson;
}
```

### 2.5.3.6.2 Creating or Deleting a Graph

To invoke Rest APIs to perform operations on graphs, refer to the following examples provided in com.huawei.graphbase.GraphBaseRestExample:

Create a graph.

```
String graphName = "graphbase";
api.createGraph(graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph HTTP/1.1
REQ BODY: {"graphName":"graphbase"}
RSP BODY: {
    "msg": "create graph success.",
    "code": "0"
}
[SUCCESS]: create graph[graphbase] successfully.
```

Delete a graph.

```
String graphName = "graphbase";
api.deleteGraph(graphName);
```

The command output is as follows:

```
REQ HEADER: DELETE https://10.7.66.150:22380/graphbase/graph?graphName=graphbase HTTP/1.1
REQ BODY: null
RSP BODY: {
    "msg": "delete graph success.",
    "code": "0"
}
[SUCCESS]: delete graph[graphbase] successfully.
```

### 2.5.3.6.3 Creating a Schema by Uploading an XML File



**graphName** specifies the name of a graph. This parameter is required when the specified REST API is invoked.

In the example, the com.huawei.graphbase.GraphBaseRestExample class has defined global variables. You can modify the configuration as required.

```
String graphName = "graphbase";
```

1. Save the XML file defined in [Compiling a Schema File \(XML\)](#) to a local directory.
2. Import the sample project. For details, see [Configuring and Importing Sample Projects](#).
3. Invoke the REST API of the addSchema class in **com.huawei.graphbase.GraphBaseRestExample**.

```
File file = new File(System.getProperty("user.dir") + File.separator + SCHEMA_FILE);
api.addSchema(file, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema HTTP/1.1
RSP BODY: {
    "msg": "Make schema file successful",
    "code": "0"
}
[SUCCESS]: create schema successfully.
```

### 2.5.3.6.4 Creating a Schema

The following code snippets are used as an example. For complete codes, see **com.huawei.graphbase.GraphBaseRestExample**.

- Create a property key.

```
PropertyKey propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.String);
propertyKey.setName("name");
api.addPropertyKey(propertyKey, graphName);
propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.Integer);
propertyKey.setName("age");
api.addPropertyKey(propertyKey, graphName);
propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.String);
propertyKey.setName("telephone");
api.addPropertyKey(propertyKey, graphName);
propertyKey = new PropertyKey();
propertyKey.setDataType(DataType.Float);
propertyKey.setName("weight");
api.addPropertyKey(propertyKey, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/property-key
HTTP/1.1
REQ BODY: {"name":"name","dataType":"String"}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {
        "dataType": "String",
        "name": "name",
        "cardinality": "SINGLE"
    }
}
[SUCCESS]: create property key[name] successfully.
```

- Query a property key based on the name.

```
api.queryPropertyKey("name",graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/property-key/  
p_name HTTP/1.1  
RSP BODY: {  
  "msg": "success",  
  "code": "0",  
  "data": {  
    "dataType": "String",  
    "name": "name",  
    "cardinality": "SINGLE",  
    "ttl":0  
  }  
}  
[SUCCESS]: query property key[name] successfully.
```

- Query all property keys.

```
api.queryAllPropertyKey(graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/property-key  
HTTP/1.1  
RSP BODY: {  
  "msg": "success",  
  "code": "0",  
  "data": {"propertyKeyList": [  
    {  
      "dataType": "String",  
      "name": "name",  
      "cardinality": "SINGLE",  
      "ttl":0  
    },  
    {  
      "dataType": "Integer",  
      "name": "age",  
      "cardinality": "SINGLE",  
      "ttl":0  
    },  
    {  
      "dataType": "String",  
      "name": "telephone",  
      "cardinality": "SINGLE",  
      "ttl":0  
    },  
    {  
      "dataType": "Float",  
      "name": "weight",  
      "cardinality": "SINGLE",  
      "ttl":0  
    }  
  ]}  
}  
[SUCCESS]: query all property key successfully.
```

- Create a vertex label.

```
api.addVertexLabel("person",graphName);  
api.addVertexLabel("phone",graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/vertex-label  
HTTP/1.1  
REQ BODY: {"name":"person"}  
RSP BODY: {  
  "msg": "success",  
  "code": "0",  
  "data": {  
    "name": "person"  
  }  
}
```

```
}
```

[SUCCESS]: create vertex label[person] successfully.

REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/vertex-label  
HTTP/1.1

REQ BODY: {"name":"person"}

RSP BODY: {  
"msg": "success",  
"code": "0",  
"data": {  
"name": "phone"  
}  
}  
}

[SUCCESS]: create vertex label[phone] successfully.

- Query a vertex label based on the name.  
api.queryVertexLabel("person",graphName);

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/vertex-label/person  
HTTP/1.1
```

RSP BODY: {  
"msg": "success",  
"code": "0",  
"data": {  
"name": "person"  
}  
}  
}

[SUCCESS]: query vertex label[person] successfully.

- Query all vertex labels.  
api.queryAllVertexLabel(graphName);

The command output is as follows:

```
REQ HEADER: GET https://10.5.199.50:22380/graphbase/graph/graphbase/schema/vertex-label  
HTTP/1.1
```

RSP BODY: {  
"msg": "success",  
"code": "0",  
"data": {"vertexLabelList": [  
{  
"name": "person"  
},  
{  
"name": "phone"  
}  
]}  
}

[SUCCESS]: query all vertex label successfully.

- Create an edge label.

```
EdgeLabel edgeLabel = new EdgeLabel();  
edgeLabel.setName("friend");  
api.addEdgeLabel(edgeLabel,graphName);  
edgeLabel = new EdgeLabel();  
edgeLabel.setName("knows");  
api.addEdgeLabel(edgeLabel,graphName);  
edgeLabel = new EdgeLabel();  
edgeLabel.setName("call");  
api.addEdgeLabel(edgeLabel,graphName);  
edgeLabel = new EdgeLabel();  
edgeLabel.setName("has");  
api.addEdgeLabel(edgeLabel,graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/edge-label  
HTTP/1.1
```

REQ BODY: {"name":"friend"}

RSP BODY: {  
"msg": "success",  
"code": "0",  
"data": {

```
"multiplicity": "MULTI",
"name": "friend",
"ttl":0
}
}
[SUCCESS]: create edge label[friend] successfully.
```

- Query an edge label based on the name.

```
api.queryEdgeLabel("friend",graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/edge-label/
friendHTTP/1.1
RSP BODY: {
"msg": "success",
"code": "0",
"data": {
"multiplicity": "MULTI",
"name": "friend",
"ttl":0
}
}
[SUCCESS]: query edge label[friend] successfully.
```

## Periodic Data Deletion

GraphBase provides the following periodic data deletion functions:

- When creating a property key, you can set the **ttl** parameter for the property. The value must be an integer, in seconds. For example, if you set **ttl** for the **name** property to **120**, the **name** property of all vertices will be deleted 120s later after being created.

This parameter takes effect only for vertex properties, and does not take effect for edge properties.

**Note:** If a vertex has only one property and the **ttl** parameter is set for the property, the vertex cannot be queried using this property after the property is deleted. However, this vertex can be queried by performing full-graph traversal, and the vertex ID is also recorded on the corresponding edge of the vertex. You are not advised to configure **ttl** for the unique property of a vertex.

- When creating an edge label, you can set the **ttl** parameter for the edge label. The value must be an integer, in seconds. For example, if you set **ttl** for the **know** label to **120**, edges labeled by the **know** label will be deleted 120s later after being created.

**Note:** The **ttl** parameter is not allowed for data stored in Elasticsearch. For example, if a MIXED index is created for the property of an edge labeled by **know**, the index data of the edge cannot be deleted with the edge 120s later, and the deleted edge cannot be found using the index. To delete the index data, you are advised to create a MIXED index for the corresponding edge property and recreate the index. After the index is successfully recreated, delete the original MIXED index.

### NOTICE

If **ttl** is set to **0** or a negative number, the corresponding property or edges are not deleted. The default value is **0**.

### 2.5.3.6.5 Creating an Index

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

- Create an index.

#### NOTE

MIXED: specifies the mixed index that is a kind of external hybrid match index. This type of indexes searches vertices and edges using any combination of indexes created in the databases. The property key corresponding to such indexes is in the <key, value> format. The options of value are as follows: **TEXT**, **TEXTSTRING**, **PREFIX\_TREE**, and **DEFAULT** (these values are case insensitive.) External indexes are used for non-exact match queries.

COMPOSITE: specifies the composite index that is a kind of internal exact match index. This type of indexes retrieve vertices or edges using a fixed combination of one or more properties. Such indexes are accurate, simple, and efficient, and can greatly accelerate the retrieval speed, but have high requirements on the property sequence in the combination. The property key corresponding to such indexes is in the <key, ""> format. Built-in indexes are used for exact match queries.

You can create an index only after a schema is created and before a vertex or edge is created. Otherwise, data cannot be queried. You need to recreate indexes to query data.

```
//Create indexes for all vertices and edges.  
addIndex(api, graphName);  
private static void addIndex(RestApi api, String graphName) {  
    //Create a built-in index for the vertex property name.  
    GraphIndexReqObj graphIndexReqObj = new GraphIndexReqObj();  
    graphIndexReqObj.setElementCategory(ElementCategory.VERTEX);  
    graphIndexReqObj.setName("name_index");  
    graphIndexReqObj.setType(IndexType.COMPOSITE);  
    List<KeyTextType> keyTextTypeList1 = new ArrayList<>();  
    KeyTextType keyTextType = new KeyTextType();  
    keyTextType.setName("name");  
    KeyTextType.setTextType(""); //If IndexType is set to COMPOSITE, TextType is empty.  
    keyTextTypeList1.add(keyTextType);  
    graphIndexReqObj.setKeyTextTypeList(keyTextTypeList1);  
    api.addGraphIndex(graphIndexReqObj, graphName);  
    //Create a built-in index for the vertex property age.  
    graphIndexReqObj = new GraphIndexReqObj();  
    graphIndexReqObj.setElementCategory(ElementCategory.VERTEX);  
    graphIndexReqObj.setName("age_index");  
    graphIndexReqObj.setType(IndexType.COMPOSITE);  
    List<KeyTextType> keyTextTypeList2 = new ArrayList<>();  
    KeyTextType keyTextType = new KeyTextType();  
    keyTextType.setName("age");  
    KeyTextType.setTextType(""); //If IndexType is set to COMPOSITE, TextType is empty.  
    keyTextTypeList2.add(keyTextType);  
    graphIndexReqObj.setKeyTextTypeList(keyTextTypeList2);  
    api.addGraphIndex(graphIndexReqObj, graphName);  
    //Create a mixed index for the vertex property telephone.  
    graphIndexReqObj = new GraphIndexReqObj();  
    graphIndexReqObj.setElementCategory(ElementCategory.VERTEX);  
    graphIndexReqObj.setName("telephone_index");  
    graphIndexReqObj.setType(IndexType.COMPOSITE);  
    List<KeyTextType> keyTextTypeList3 = new ArrayList<>();  
    KeyTextType keyTextType = new KeyTextType();  
    keyTextType.setName("telephone");  
    KeyTextType.setTextType(""); //If IndexType is set to COMPOSITE, TextType is empty.  
    keyTextTypeList3.add(keyTextType);  
    graphIndexReqObj.setKeyTextTypeList(keyTextTypeList3);  
    api.addGraphIndex(graphIndexReqObj, graphName);  
    ...//Creates a mixed index for the edge property weight.  
    graphIndexReqObj = new GraphIndexReqObj();  
    graphIndexReqObj.setElementCategory(ElementCategory.EDGE);
```

```
graphIndexReqObj.setName("weight_index");
graphIndexReqObj.setType(IndexType.MIXED);
List<KeyTextType> keyTextTypeList4 = new ArrayList<>();
keyTextType = new KeyTextType();
keyTextType.setName("weight");
...KeyTextType.setTextType("DEFAULT");//If IndexType is set to MIXED, TextType is transferred based
on the information provided in the interface document.
keyTextTypeList4.add(keyTextType);
graphIndexReqObj.setKeyTextTypeList(keyTextTypeList4);
api.addGraphIndex(graphIndexReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph
HTTP/1.1
REQ BODY: {"name":"name_index","elementCategory":"VERTEX","unique":false,"keyTextTypeList": [
["name":"name","textType":""}], "type":"COMPOSITE"}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"indexName": "name_index"}
}
[SUCCESS]: create COMPOSITE graph index[name_index] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph
HTTP/1.1
REQ BODY: {"name":"age_index","elementCategory":"VERTEX","unique":false,"keyTextTypeList": [
["name":"age","textType":""]], "type":"COMPOSITE"}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"indexName": "age_index"}
}
[SUCCESS]: create COMPOSITE graph index[age_index] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph
HTTP/1.1
REQ BODY: {"name":"telephone_index","elementCategory":"VERTEX","unique":false,"keyTextTypeList": [
["name":"telephone","textType":"STRING"]], "type":"MIXED"}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"indexName": "telephone_index"}
}
[SUCCESS]: create MIXED graph index[telephone_index] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/graph
HTTP/1.1
REQ BODY: {"name":"weight_index","elementCategory":"EDGE","unique":false,"keyTextTypeList": [
["name":"weight","textType":"DEFAULT"]], "type":"MIXED"}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"indexName": "weight_index"}
}
[SUCCESS]: create MIXED graph index[weight_index] successfully.
```

- Recreate an index.

```
api.reCreateGraphIndex("name_index", graphName);
api.reCreateGraphIndex("age_index", graphName);
api.reCreateGraphIndex("telephone_index", graphName);
api.reCreateGraphIndex("weight_index", graphName);
```

The command output is as follows:

```
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/
name_index HTTP/1.1
REQ BODY:
RSP BODY: {
    "msg": "Asynchronous task submitted successfully.",
    "code": "0",
    "data": {"id": "1433552"}
}
[SUCCESS]: reCreate graph index[name_index] successfully.
```

```
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/  
age_index HTTP/1.1  
REQ BODY:  
RSP BODY: {  
    "msg": "Asynchronous task submitted successfully.",  
    "code": "0",  
    "data": {"id": "2482128"}  
}  
[SUCCESS]: reCreate graph index[age_index] successfully.  
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/  
telephone_index HTTP/1.1  
REQ BODY:  
RSP BODY: {  
    "msg": "Asynchronous task submitted successfully.",  
    "code": "0",  
    "data": {"id": "1135248"}  
}  
[SUCCESS]: reCreate graph index[telephone_index] successfully.  
REQ HEADER: PUT https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index/reindex/  
weight_index HTTP/1.1  
REQ BODY:  
RSP BODY: {  
    "msg": "Asynchronous task submitted successfully.",  
    "code": "0",  
    "data": {"id": "2183824"}  
}  
[SUCCESS]: reCreate graph index[weight_index] successfully.
```

- Query all indexes.

```
api.queryAllIndex(graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/schema/index HTTP/1.1  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"indexList": [  
        {  
            "elementCategory": "VERTEX",  
            "name": "name_index",  
            "propertyKeyStatusList": [{  
                "name": "name",  
                "status": "ENABLED"  
            }],  
            "type": "COMPOSITE"  
        },  
        {  
            "elementCategory": "VERTEX",  
            "name": "age_index",  
            "propertyKeyStatusList": [{  
                "name": "age",  
                "status": "ENABLED"  
            }],  
            "type": "COMPOSITE"  
        },  
        {  
            "elementCategory": "VERTEX",  
            "name": "telephone_index",  
            "propertyKeyStatusList": [{  
                "name": "telephone",  
                "status": "ENABLED"  
            }],  
            "type": "MIXED"  
        },  
        {  
            "elementCategory": "EDGE",  
            "name": "weight_index",  
            "propertyKeyStatusList": [{  
                "name": "weight",  
                "status": "ENABLED"  
            }]  
        }  
    ]}  
}
```

```
        ],
        "type": "MIXED"
    }
}
[SUCCESS]: query all index successfully.
```

### 2.5.3.6.6 Creating and Querying a Vertex or an Edge

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

- Create a vertex.

```
//Create four nodes labeled as person 1, person 4, person 5, and person 8.
addVertexPerson(api, graphName);
private static void addVertexPerson(RestApi api, String graphName) {
...//Create the node person 1.
    AddVertexReqObj addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("person");
    List<PropertyReqObj> propertyReqObjList1 = new ArrayList<>();
    PropertyReqObj propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("name");
    propertyReqObj.setValue("marko");
    propertyReqObjList1.add(propertyReqObj);
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("age");
    propertyReqObj.setValue(29);
    propertyReqObjList1.add(propertyReqObj);
    addVertexReqObj.setPropertyList(propertyReqObjList1);
    api.addVertex(addVertexReqObj, graphName);
//Create the node person 4.
    addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("person");
    List<PropertyReqObj> propertyReqObjList2 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("name");
    propertyReqObj.setValue("josh");
    propertyReqObjList2.add(propertyReqObj);
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("age");
    propertyReqObj.setValue(32);
    propertyReqObjList2.add(propertyReqObj);
    addVertexReqObj.setPropertyList(propertyReqObjList2);
    api.addVertex(addVertexReqObj, graphName);
...//Create the node person 5.
    addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("person");
    List<PropertyReqObj> propertyReqObjList3 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("name");
    propertyReqObj.setValue("vadas");
    propertyReqObjList3.add(propertyReqObj);
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("age");
    propertyReqObj.setValue(27);
    propertyReqObjList3.add(propertyReqObj);
    addVertexReqObj.setPropertyList(propertyReqObjList3);
    api.addVertex(addVertexReqObj, graphName);
...//Create the node person 8.
    addVertexReqObj = new AddVertexReqObj();
    addVertexReqObj.setVertexLabel("person");
    List<PropertyReqObj> propertyReqObjList4 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("name");
    propertyReqObj.setValue("blame");
    propertyReqObjList4.add(propertyReqObj);
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("age");
    propertyReqObj.setValue(30);
```

```
        propertyReqObjList4.add(propertyReqObj);
        addVertexReqObj.setPropertyList(propertyReqObjList4);
        api.addVertex(addVertexReqObj, graphName);
    }
    //Create four nodes labeled as phone 2, phone 3, phone 6, and phone 7, respectively.
    addVertexPhone(api, graphName);
    private static void addVertexPhone(RestApi api, String graphName) {
        //Create the node phone 2.
        AddVertexReqObj addVertexReqObj = new AddVertexReqObj();
        addVertexReqObj.setVertexLabel("phone");
        List<PropertyReqObj> propertyReqObjList1 = new ArrayList<>();
        PropertyReqObj propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("telephone");
        propertyReqObj.setValue("15000000000");
        propertyReqObjList1.add(propertyReqObj);
        addVertexReqObj.setPropertyList(propertyReqObjList1);
        api.addVertex(addVertexReqObj, graphName);
        //Create the node phone 3.
        addVertexReqObj = new AddVertexReqObj();
        addVertexReqObj.setVertexLabel("phone");
        List<PropertyReqObj> propertyReqObjList2 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("telephone");
        propertyReqObj.setValue("15222222222");
        propertyReqObjList2.add(propertyReqObj);
        addVertexReqObj.setPropertyList(propertyReqObjList2);
        api.addVertex(addVertexReqObj, graphName);
        //Create the node phone 6.
        addVertexReqObj = new AddVertexReqObj();
        addVertexReqObj.setVertexLabel("phone");
        List<PropertyReqObj> propertyReqObjList3 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("telephone");
        propertyReqObj.setValue("13777777777");
        propertyReqObjList3.add(propertyReqObj);
        addVertexReqObj.setPropertyList(propertyReqObjList3);
        api.addVertex(addVertexReqObj, graphName);
        //Create the node phone 7.
        addVertexReqObj = new AddVertexReqObj();
        addVertexReqObj.setVertexLabel("phone");
        List<PropertyReqObj> propertyReqObjList4 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("telephone");
        propertyReqObj.setValue("13666666666");
        propertyReqObjList4.add(propertyReqObj);
        addVertexReqObj.setPropertyList(propertyReqObjList4);
        api.addVertex(addVertexReqObj, graphName);
    }
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"marko"}, {"name":"age","value":29}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1261176"}
}
[SUCCESS]: add vertex[person] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"josh"}, {"name":"age","value":32}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1170064"}
}
[SUCCESS]: add vertex[person] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"vadas"},
```

```

{"name":"age","value":27}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1747680"}
}
[SUCCESS]: add vertex[person] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"person","propertyList":[{"name":"name","value":"blame"}, {"name":"age","value":30}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1478712"}
}
[SUCCESS]: add vertex[person] successfully.

REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"15000000000"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "2218640"}
}
[SUCCESS]: add vertex[phone] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"15222222222"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1297704"}
}
[SUCCESS]: add vertex[phone] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"13777777777"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "1508048"}
}
[SUCCESS]: add vertex[phone] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex HTTP/1.1
REQ BODY: {"vertexLabel":"phone","propertyList":[{"name":"telephone","value":"13666666666"}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "2556624"}
}
[SUCCESS]: add vertex[phone] successfully.

```

- Query a vertex based on the vertex ID.

```
String vertexId = getVertexIdByProperty(api,graphName,"person","name","marko");
api.queryVertex(vertexId, graphName);
```

The command output is as follows:

```

REQ HEADER: GET https://10.7.66.150:22380/graphbase/graph/graphbase/vertex/1261176 HTTP/1.1
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {
        "propertyList": [
            {
                "name": "name",
                "id": "99vz-r14o-4if9",
                "value": "marko"
            },
            {
                "name": "age",
                "id": "c30v-r14o-5wzp",
                "value": "29"
            }
        ]
    }
}
```

```
        }
    ],
    "id": "1261176",
    "vertexLabel": "person"
}
}

[SUCCESS]: query vertex[1261176] successfully.

● Create an edge.
//Create the following six edges: 9, 10, 11, 12, 13, 14, 15, and 16.
addEdges(api, graphName);
private static void addEdges(RestApi api, String graphName) {
...//Create the edge 9 in the scenario.
    AddEdgeReqObj addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("knows");
    addEdgeReqObj.setOutVertexId("1261176");
    addEdgeReqObj.setInVertexId("1747680");
    List<PropertyReqObj> propertyReqObjList1 = new ArrayList<>();
    PropertyReqObj propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("weight");
    propertyReqObj.setValue(0.4);
    propertyReqObjList1.add(propertyReqObj);
    addEdgeReqObj.setPropertyList(propertyReqObjList1);
    api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 10 in the scenario.
    addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("has");
    addEdgeReqObj.setOutVertexId("1261176");
    addEdgeReqObj.setInVertexId("2218640");
    List<PropertyReqObj> propertyReqObjList2 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("weight");
    propertyReqObj.setValue(0.8);
    propertyReqObjList2.add(propertyReqObj);
    addEdgeReqObj.setPropertyList(propertyReqObjList2);
    api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 11 in the scenario.
    addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("has");
    addEdgeReqObj.setOutVertexId("1170064");
    addEdgeReqObj.setInVertexId("1297704");
    List<PropertyReqObj> propertyReqObjList3 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("weight");
    propertyReqObj.setValue(0.8);
    propertyReqObjList3.add(propertyReqObj);
    addEdgeReqObj.setPropertyList(propertyReqObjList3);
    api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 12 in the scenario.
    addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("knows");
    addEdgeReqObj.setOutVertexId("1170064");
    addEdgeReqObj.setInVertexId("1478712");
    List<PropertyReqObj> propertyReqObjList4 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("weight");
    propertyReqObj.setValue(0.4);
    propertyReqObjList4.add(propertyReqObj);
    addEdgeReqObj.setPropertyList(propertyReqObjList4);
    api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 13 in the scenario.
    addEdgeReqObj = new AddEdgeReqObj();
    addEdgeReqObj.setEdgeLabel("has");
    addEdgeReqObj.setOutVertexId("1478712");
    addEdgeReqObj.setInVertexId("2556624");
    List<PropertyReqObj> propertyReqObjList5 = new ArrayList<>();
    propertyReqObj = new PropertyReqObj();
    propertyReqObj.setName("weight");
    propertyReqObj.setValue(0.8);
    propertyReqObjList5.add(propertyReqObj);
```

```

        addEdgeReqObj.setPropertyList(propertyReqObjList5);
        api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 14 in the scenario. Edge 14 is a bidirectional edge, and you need to specify an
incoming edge, and an outgoing edge.
        addEdgeReqObj = new AddEdgeReqObj();
        addEdgeReqObj.setEdgeLabel("call");
        addEdgeReqObj.setOutVertexId("2556624");
        addEdgeReqObj.setInVertexId("1508048");
        List<PropertyReqObj> propertyReqObjList6 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("weight");
        propertyReqObj.setValue(0.6);
        propertyReqObjList6.add(propertyReqObj);
        addEdgeReqObj.setPropertyList(propertyReqObjList6);
        api.addEdge(addEdgeReqObj, graphName);
        addEdgeReqObj = new AddEdgeReqObj();
        addEdgeReqObj.setEdgeLabel("call");
        addEdgeReqObj.setOutVertexId("1508048");
        addEdgeReqObj.setInVertexId("2556624");
        List<PropertyReqObj> propertyReqObjList7 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("weight");
        propertyReqObj.setValue(0.6);
        propertyReqObjList7.add(propertyReqObj);
        addEdgeReqObj.setPropertyList(propertyReqObjList7);
        api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 15 in the scenario.
        addEdgeReqObj = new AddEdgeReqObj();
        addEdgeReqObj.setEdgeLabel("has");
        addEdgeReqObj.setOutVertexId("1508048");
        addEdgeReqObj.setInVertexId("1747680");
        List<PropertyReqObj> propertyReqObjList8 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("weight");
        propertyReqObj.setValue(0.8);
        propertyReqObjList8.add(propertyReqObj);
        addEdgeReqObj.setPropertyList(propertyReqObjList8);
        api.addEdge(addEdgeReqObj, graphName);
...//Create the edge 16 in the scenario.
        addEdgeReqObj = new AddEdgeReqObj();
        addEdgeReqObj.setEdgeLabel("friend");
        addEdgeReqObj.setOutVertexId("1261176");
        addEdgeReqObj.setInVertexId("2016584");
        List<PropertyReqObj> propertyReqObjList9 = new ArrayList<>();
        propertyReqObj = new PropertyReqObj();
        propertyReqObj.setName("weight");
        propertyReqObj.setValue(1.0);
        propertyReqObjList9.add(propertyReqObj);
        addEdgeReqObj.setPropertyList(propertyReqObjList9);
        api.addEdge(addEdgeReqObj, graphName);
    }
}

```

The command output is as follows:

```

REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1261176","inVertexId":"1747680","edgeLabel":"knows","propertyList": [{"name":"weight","value":0.4}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "ew5r-r14o-bj9x-11gio"}
}
[SUCCESS]: add edge[knows] successfully.
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1
REQ BODY: {"outVertexId":"1261176","inVertexId":"2218640","edgeLabel":"has","propertyList": [{"name":"weight","value":0.8}]}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"id": "hpan-r14o-ecet-1bjww"}
}

```

```
[SUCCESS]: add edge[has] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1  
REQ BODY: {"outVertexId":"1170064","inVertexId":"1297704","edgeLabel":"has","propertyList":  
[{"name":"weight","value":0.8}]}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"id": "nqky-p2ts-ecet-rtbc"}  
}  
[SUCCESS]: add edge[has] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1  
REQ BODY: {"outVertexId":"1170064","inVertexId":"1478712","edgeLabel":"knows","propertyList":  
[{"name":"weight","value":0.4}]}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"id": "qjpu-p2ts-bj9x-vozc"}  
}  
[SUCCESS]: add edge[knows] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1  
REQ BODY: {"outVertexId":"1478712","inVertexId":"2556624","edgeLabel":"has","propertyList":  
[{"name":"weight","value":0.8}]}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"id": "gflz-vozc-ecet-1ispc"}  
}  
[SUCCESS]: add edge[has] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1  
REQ BODY: {"outVertexId":"2556624","inVertexId":"1508048","edgeLabel":"call","propertyList":  
[{"name":"weight","value":0.6}]}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"id": "kzq2-1ispc-cxud-wbm8"}  
}  
[SUCCESS]: add edge[call] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1  
REQ BODY: {"outVertexId":"1508048","inVertexId":"2556624","edgeLabel":"call","propertyList":  
[{"name":"weight","value":0.6}]}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"id": "nsuy-wbm8-cxud-1ispc"}  
}  
[SUCCESS]: add edge[call] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1  
REQ BODY: {"outVertexId":"1508048","inVertexId":"1747680","edgeLabel":"has","propertyList":  
[{"name":"weight","value":0.8}]}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"id": "qlzu-wbm8-ecet-11gio"}  
}  
[SUCCESS]: add edge[has] successfully.  
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge HTTP/1.1  
REQ BODY: {"outVertexId":"1261176","inVertexId":"1170064","edgeLabel":"friend","propertyList":  
[{"name":"weight","value":1.0}]}  
RSP BODY: {  
    "msg": "success",  
    "code": "0",  
    "data": {"id": "kifj-r14o-a4ph-p2ts"}  
}  
[SUCCESS]: add edge[friend] successfully.
```

- Query an edge based on the edge ID.

```
String edgeld = getEdgeldByProperty(api,graphName,"call","weight","0.6");  
api.queryEdge(edgeld, graphName);
```

The command output is as follows:

```
REQ HEADER: GET https://10.5.199.173:22380/graphbase/graph/graphbasejpps/edge/b8lu-mhlc-5x-mhig HTTP/1.1
RSP BODY: {
  "msg": "Success.",
  "code": "0",
  "data": {
    "propertyList": [
      {
        "name": "weight",
        "value": "0.6"
      }
    ],
    "inVertexId": "1049128",
    "edgeLabel": "call",
    "id": "b8lu-mhlc-5x-mhig",
    "outVertexId": "1049232"
  }
}
[SUCCESS]: query edge[b8lu-mhlc-5x-mhig] successfully.
```

### 2.5.3.6.7 Performing a Full Graph Query

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

- Perform full graph query for vertices.

```
//Perform full graph query for vertices in the following scenario: To query the users who are older than 29 years old and list the users by age in ascending order. The upper limit of the number of users is 3.
VertexSearchReqObj vertexSearchReqObj = new VertexSearchReqObj();
vertexSearchReqObj.setVertexLabel("person");
List<PropertyFilter> propertyFilterList = new ArrayList<>();
PropertyFilter propertyFilter = new PropertyFilter();
propertyFilter.setProperty("age");
propertyFilter.setPredicate(PropertyPredicate.GREATER_THAN);
List<Integer> values = new ArrayList();
values.add(29);
propertyFilter.setValues(values);
propertyFilterList.add(propertyFilter);
vertexSearchReqObj.setFilterList(propertyFilterList);
List<PropertyKeySort> sortList = new ArrayList<>();
PropertyKeySort propertyKeySort = new PropertyKeySort();
propertyKeySort.setPropertyKeyName("age");
propertyKeySort.setSortType("ASC");
sortList.add(propertyKeySort);
vertexSearchReqObj.setPropertyKeySortList(sortList);
vertexSearchReqObj.setLimit(3);
//Perform full graph query for vertices.
api.searchVertex(vertexSearchReqObj, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/vertex/search HTTP/1.1
REQ BODY: {"filterList":[{"propertyName":"age","predicate":">>","values": [29]}],"vertexLabel":"person","propertyKeySortList": [{"propertyKeyName":"age","sortType":"ASC"}],"limit":3}
RSP BODY: {
  "msg": "success",
  "code": "0",
  "data": {
    "totalHits": 2,
    "vertexList": [
      {
        "propertyList": [
          {
            "name": "name",
            "id": "atc7-vozc-4if9",
            "value": "blame"
          },
          {
            "name": "age",
            "value": "29"
          }
        ]
      }
    ]
  }
}
```

```

        "id": "dmh3-vozc-5wzp",
        "value": "30"
    }
],
"id": "1478712",
"vertexLabel": "person"
},
{
"propertyList": [
{
"name": "name",
"id": "9owi-p2ts-4if9",
"value": "josh"
},
{
"name": "age",
"id": "ci1e-p2ts-5wzp",
"value": "32"
}
],
"id": "1170064",
"vertexLabel": "person"
}
]
}
}

[SUCCESS]: search vertex [person] successfully.
=====
Description of the JSON data result
format=====
|---msg           message
|---code          code
|---data          Result data storage location
|   |---totalHits Number of vertices
|   |---vertexList Vertex set
|   |   |---vertexLabel Vertex label
|   |---id           Vertex ID
|   |---propertyList Property set
|   |   |---name       Property name
|   |   |---value      Property value
|   |---id           Property ID
=====
=====
```

- Perform full graph query for edges.

```
//Perform full graph query for edges in the following scenario: To query the edge that is labeled as
knows, and the value of the edge property weight is equal to or less than 0.6.
EdgeSearchReqObj edgeSearchReqObj = new EdgeSearchReqObj();
edgeSearchReqObj.setEdgeLabel("knows");
edgeSearchReqObj.setLimit(2);
List<PropertyFilter> propertyFilterList = new ArrayList<>();
PropertyFilter propertyFilter = new PropertyFilter();
propertyFilter.setProperty("weight");
propertyFilter.setPredicate(PropertyPredicate.LESS_THAN_EQUAL);
List<Float> values = new ArrayList();
values.add(new Float(0.6));
propertyFilter.setValues(values);
propertyFilterList.add(propertyFilter);
edgeSearchReqObj.setFilterList(propertyFilterList);
api.searchEdge(edgeSearchReqObj, graphName);
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/edge/search HTTP/1.1
REQ BODY: {"filterList": [{"propertyName": "weight", "predicate": "<=", "values": [0.6]}, {"edgeLabel": "knows", "limit": 2}]
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {
        "totalHits": 2,
        "edgeList": [
            {
                "id": "1478712",
                "label": "person",
                "source": "1170064",
                "target": "1478712",
                "value": "30"
            }
        ]
    }
}
```

```
{
  "propertyList": [
    {
      "name": "weight",
      "value": "0.4"
    }
  ],
  "inVertexId": "1747680",
  "edgeLabel": "knows",
  "outVertexId": "1261176"
},
{
  "propertyList": [
    {
      "name": "weight",
      "value": "0.4"
    }
  ],
  "inVertexId": "1478712",
  "edgeLabel": "knows",
  "outVertexId": "1170064"
}
]
}
}

[SUCCESS]: search edge [knows] successfully.
=====
Description of the JSON data result
format=====
|---msg          message
|---code         code
|---data         Result data storage location
|   |---totalHits   Number of edges
|   |---edgeList     Edge set
|       |---edgeLabel   Edge label
|       |---outVertexId Start vertex ID
|       |---inVertexId  End vertex ID
|       |---propertyList Property set
|           |---name      Property name
|           |---value     Property value
=====
=====
```

### 2.5.3.6.8 Performing a Path Query

- Graph traversal: Access the vertices of the graph along the side from any vertex of the graph. Graph traversal is a basic operation in graph databases, and it is similar to tree traversal. In a narrow sense, a vertex can be accessed only once in a graph traversal. The graph traversal described in this document is that in a broad sense.
- Graph path: indicates the set of vertices and edges that passed by during graph traversal. The number of edges in a path is called the path depth.

Use marko in [Typical Application Scenario](#) as an example, the paths formed by the vertices and edges found based on the first layer expansion query are as follows:

- marko -[friend]-> josh
- marko -[has]-> 150000000000
- marko -[knows]-> vadas

In this example, the person name or mobile number indicates the entity object. `-[relationship]->` is used to indicate the edge, and the relationship in the square brackets([]) indicates the edge label. The depth of these paths is 1.

Continue traversing based on the preceding path result, and obtain a second layer edge and a vertex of marko to form a new path. The original path is named the parent path of the new path, and the new path is named the sub-path of the original path.

Example:

- Original path (path A): marko -[friend]-> josh
- New path (path B): marko -[friend]-> josh -[has]-> 1522222222
- Path A is the parent path of path B, and path B is the sub-path of path A.
- Full path: indicates all graph paths between two vertices that meet specified conditions.

Scenario 1: Query the full path between node 1 and node 8.

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

```
allPathSearch(api, graphName);
private static void allPathSearch(RestApi api, String graphName) {
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "marko"));
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "blame"));
    pathSearchReqObj.setVertexIdList(vertexIdList);
    pathSearchReqObj.setLayer(7);
    api.searchPath(pathSearchReqObj, graphName);
}
```

The command output is as follows:

REQ HEADER: POST https://10.7.66.150:22380/graphbase/default/paths HTTP/1.1

REQ BODY: {"vertexIdList":["1793880","2842456"],"layer":7}

```
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"pathSet": [
        {
            "edgeList": [
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "0.8"
                        }
                    ],
                    "start": "2842456",
                    "end": "1683696",
                    "id": "a4x7-1ox94-700l-1035c",
                    "label": "has",
                    "type": "edge"
                },
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "0.6"
                        }
                    ],
                    "start": "1561040",
                    "end": "1683696",
                    "id": "3hkq-xgi8-5lg5-1035c",
                    "label": "call",
                    "type": "edge"
                },
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "0.4"
                        }
                    ],
                    "start": "2016584",
                    "end": "2842456",
                    "id": "6kdl-17808-46vp-1ox94",
                    "label": "knows",
                    "type": "edge"
                },
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "0.4"
                        }
                    ],
                    "start": "2016584",
                    "end": "2842456",
                    "id": "6kdl-17808-46vp-1ox94",
                    "label": "knows",
                    "type": "edge"
                }
            ]
        }
    ]}
```

```
        }],
        "start": "1793880",
        "end": "3065160",
        "id": "4inf-12g60-46vp-1tp3c",
        "label": "knows",
        "type": "edge"
    },
    {
        "propertyList": [
            {
                "name": "weight",
                "value": "0.6"
            }
        ],
        "start": "1683696",
        "end": "1561040",
        "id": "3pni-1035c-5lg5-xgi8",
        "label": "call",
        "type": "edge"
    },
    {
        "propertyList": [
            {
                "name": "weight",
                "value": "0.8"
            }
        ],
        "start": "3065160",
        "end": "1561040",
        "id": "9dih-1tp3c-700l-xgi8",
        "label": "has",
        "type": "edge"
    },
    {
        "propertyList": [
            {
                "name": "weight",
                "value": "1.0"
            }
        ],
        "start": "1793880",
        "end": "2016584",
        "id": "cy23-12g60-2sb9-17808",
        "label": "friend",
        "type": "edge"
    }
],
"start": "1793880",
"vertexList": [
    {
        "labelList": ["phone"],
        "propertyList": [
            {
                "name": "telephone",
                "value": "13666666666"
            }
        ],
        "id": "1683696",
        "type": "vertex"
    },
    {
        "labelList": ["person"],
        "propertyList": [
            {
                "name": "name",
                "value": "josh"
            },
            {
                "name": "age",
                "value": "32"
            }
        ],
        "id": "2016584",
        "type": "vertex"
    }
],
"labelList": ["person"],
```

```
"propertyList": [
    {
        "name": "name",
        "value": "vadas"
    },
    {
        "name": "age",
        "value": "27"
    }
],
"id": "3065160",
"type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "marko"
        },
        {
            "name": "age",
            "value": "29"
        }
],
"id": "1793880",
"type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "blame"
        },
        {
            "name": "age",
            "value": "30"
        }
],
"id": "2842456",
"type": "vertex"
},
{
    "labelList": ["phone"],
    "propertyList": [
        {
            "name": "telephone",
            "value": "13777777777"
        }
],
"id": "1561040",
"type": "vertex"
}
],
"end": "2842456",
"pathList": [
    [
        "cy23-12g60-2sb9-17808",
        "6kdl-17808-46vp-1ox94"
    ],
    [
        "4inf-12g60-46vp-1tp3c",
        "9dih-1tp3c-700l-xgi8",
        "3pni-1035c-5lg5-xgi8",
        "a4x7-1ox94-700l-1035c"
    ],
    [
        "4inf-12g60-46vp-1tp3c",
        "9dih-1tp3c-700l-xgi8",
        "3hkq-xgi8-5lg5-1035c",
        "3hkq-xgi8-5lg5-1035c"
    ]
]
```

```
        "a4x7-1ox94-700l-1035c"
    ]
}
}
[SUCCESS]: search path successfully.
=====
format=====
|---msg          message
|---code         code
|---data         Result data storage location
|---pathSet      All path data. Multiple vertex pairs correspond to multiple data structures.
|---start        Start node ID
|---end          End node ID
|---edgeList     Detailed data of all relationships in the data result
|---vertexList   Detailed data of all vertices included in the current data result
|---pathList     Path set. Each path stores only the relationship access sequence table. Only
relationship IDs are recorded.
=====
=====
```

Build a path based on the path result (applicable to all paths):

- Find a path from the path list. For example:

```
[  
    "cy23-12g60-2sb9-17808",  
    "6kdl-17808-46vp-1ox94"  
]
```

- Find an edge from the edge list based on the edge ID. For example:

Find the edge with the ID of cy23-12g60-2sb9-17808:

```
{  
    "propertyList": [{  
        "name": "weight",  
        "value": "1.0"  
    }],  
    "start": "1793880",  
    "end": "2016584",  
    "id": "cy23-12g60-2sb9-17808",  
    "label": "friend",  
    "type": "edge"  
}
```

- Identify the start vertex, obtain the end node ID based on the edge data in **b**, and obtain the vertex data from the vertex list. For example:

Start from the edge with the ID of cy23-12g60-2sb9-17808 to find the end node with the vertex ID of 2016584

```
{  
    "labelList": ["person"],  
    "propertyList": [  
        {  
            "name": "name",  
            "value": "josh"  
        },  
        {  
            "name": "age",  
            "value": "32"  
        }  
    ],  
    "id": "2016584",  
    "type": "vertex"  
}
```

- Repeat **b** and **c** until the data in **a** is exhausted.

The path in the preceding example is simplified as follows:

```
(1793880,marko) -[cy23-12g60-2sb9-17808:friend]-> (2016584,josh)  
-[6kdl-17808-46vp-1ox94:knows]-> (2842456:blame)
```

Vertices are marked in parentheses and in the format of (Vertex ID, Person name). Edges are marked in square brackets and in the format of -[Edge ID:Edge type]->

- Shortest path

It indicates the shortest path between two vertices, specifically the path with the smallest sum of relationship weights. This document considers that the path with the smallest sum of path relationship weights is not the shortest path. Instead, the shortest path must comply with the specified weight rule. If the weight calculation rule is not specified, the shortest path is the path with the minimum path depth by default. The shortest path complying with the relationship weighting rule is the weight-based shortest path.

Scenario 2: Query the shortest path from node 1 to node 8.

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

```
shortestPathSearch(api, graphName);
private static void shortestPathSearch(RestApi api, String graphName) {
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "marko"));
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "blame"));
    pathSearchReqObj.setVertexIdList(vertexIdList);
    pathSearchReqObj.setLayer(7);
    ...PathSearchReqObj.setOption("shortest");//If option is not transferred, the default value is all, that
    is, the full path.
    api.searchPath(pathSearchReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/default/paths HTTP/1.1
```

```
REQ BODY: {"vertexIdList":["1793880","2842456"],"layer":7,"option":"shortest"}
```

```
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"pathSet": [
        {
            "edgeList": [
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "0.4"
                        }
                    ],
                    "start": "2016584",
                    "end": "2842456",
                    "id": "6kdl-17808-46vp-1ox94",
                    "label": "knows",
                    "type": "edge"
                },
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "1.0"
                        }
                    ],
                    "start": "1793880",
                    "end": "2016584",
                    "id": "cy23-12g60-2sb9-17808",
                    "label": "friend",
                    "type": "edge"
                }
            ],
            "start": "1793880",
            "vertexList": [
                {
                    "labelList": ["person"],
                    "propertyList": [

```

```

        "name": "name",
        "value": "josh"
    },
    {
        "name": "age",
        "value": "32"
    }
],
"id": "2016584",
"type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "marko"
        },
        {
            "name": "age",
            "value": "29"
        }
    ],
    "id": "1793880",
    "type": "vertex"
},
{
    "labelList": ["person"],
    "propertyList": [
        {
            "name": "name",
            "value": "blame"
        },
        {
            "name": "age",
            "value": "30"
        }
    ],
    "id": "2842456",
    "type": "vertex"
}
],
"end": "2842456",
"pathList": [
    [
        "cy23-12g60-2sb9-17808",
        "6kdl-17808-46vp-1ox94"
    ]
]}
}
]
[SUCCESS]: search path successfully.
=====
Description of the JSON data result
format=====
|---msg           message
|---code          code
|---data          Result data storage location
|---pathSet       All path data. Multiple vertex pairs correspond to multiple data structures.
|---start         Start node ID
|---end           End node ID
|---edgeList      Detailed data of all relationships in the data result
|---vertexList    Detailed data of all vertices included in the current data result
|---pathList      Path set. Each path stores only the relationship access sequence table. Only
relationship IDs are recorded.
=====
=====
```

- Scenario 3: Query the paths where the **age** values of all nodes are greater than or equal to 29 between node 1 and node 8.

The following code snippets are used as an example. For complete codes, see [com.huawei.graphbase.GraphBaseRestExample](#).

```
vertexFilterPathSearch(api, graphName);
private static void vertexFilterPathSearch(RestApi api, String graphName) {
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "marko"));
    vertexIdList.add(getVertexIdByProperty(api, graphName, "person", "name", "blame"));
    pathSearchReqObj.setVertexIdList(vertexIdList);
    VertexFilter vertexFilter = new VertexFilter();
    List<PropertyFilter> propertyFilterList3 = new ArrayList<>();
    PropertyFilter propertyFilter = new PropertyFilter();
    propertyFilter.setProperty("age");
    propertyFilter.setPredicate(PropertyPredicate.GREATER_THAN_EQUAL);
    List<Integer> values2 = new ArrayList();
    values2.add(29);
    propertyFilter.setValues(values2);
    propertyFilterList3.add(propertyFilter);
    vertexFilter.setFilterList(propertyFilterList3);
    pathSearchReqObj.setVertexFilter(vertexFilter);
    pathSearchReqObj.setLayer(7);
    api.searchPath(pathSearchReqObj, graphName);
}
```

The command output is as follows:

```
REQ HEADER: POST https://10.7.66.150:22380/graphbase/default/paths HTTP/1.1
REQ BODY: {"vertexIdList":["1793880","2842456"],"layer":7,"vertexFilter":{"filterList": [{"propertyName":"age","predicate":">=","values":[29]}],"limit":0}}
RSP BODY: {
    "msg": "success",
    "code": "0",
    "data": {"pathSet": [
        {
            "edgeList": [
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "0.4"
                        }
                    ],
                    "start": "2016584",
                    "end": "2842456",
                    "id": "6kdl-17808-46vp-1ox94",
                    "label": "knows",
                    "type": "edge"
                },
                {
                    "propertyList": [
                        {
                            "name": "weight",
                            "value": "1.0"
                        }
                    ],
                    "start": "1793880",
                    "end": "2016584",
                    "id": "cy23-12g60-2sb9-17808",
                    "label": "friend",
                    "type": "edge"
                }
            ],
            "start": "1793880",
            "vertexList": [
                {
                    "labelList": ["person"],
                    "propertyList": [
                        {
                            "name": "name",
                            "value": "josh"
                        },
                        {
                            "name": "age",
                            "value": "32"
                        }
                    ],
                    "id": "2016584",
                    "type": "vertex"
                }
            ]
        }
    ]
}
```

```

},
{
  "labelList": ["person"],
  "propertyList": [
    {
      "name": "name",
      "value": "marko"
    },
    {
      "name": "age",
      "value": "29"
    }
  ],
  "id": "1793880",
  "type": "vertex"
},
{
  "labelList": ["person"],
  "propertyList": [
    {
      "name": "name",
      "value": "blame"
    },
    {
      "name": "age",
      "value": "30"
    }
  ],
  "id": "2842456",
  "type": "vertex"
}
],
"end": "2842456",
"pathList": [
  [
    "cy23-12g60-2sb9-17808",
    "6kdl-17808-46vp-1ox94"
  ]
]}
]
}
[SUCCESS]: search path successfully.
=====
Description of the JSON data result
format=====
|---msg           message
|---code          code
|---data          Result data storage location
|---pathSet       All path data. Multiple vertex pairs correspond to multiple data structures.
|   |---start      Start node ID
|   |---end        End node ID
|   |---edgeList    Detailed data of all relationships in the data result
|   |---vertexList  Detailed data of all vertices included in the current data result
|   |---pathList    Path set. Each path stores only the relationship access sequence table. Only
relationship IDs are recorded.
=====
=====
```

### 2.5.3.6.9 Performing a Line Expansion Query

- Line Expansion Query
  - Similar to graph traversal, a line expansion query starts from a specified vertex and traverses the incident edges to find the target vertex. In a graph traversal, the process of finding the vertices that have direct connection with the specified vertex is the first layer expansion for the vertex.
  - The graph traversal proceeds with the query from these vertices directly connected with the start vertex to find the vertices that are indirectly connected with the start vertex. This process is called second layer

expansion. In this way the graph traversal will expand n layers to find all vertices that are directly and indirectly connected with the start vertex.

```
//Scenario: Start from node 1 and expand two layers.  
//Layer 1 query condition: To query the vertices whose age is equal to or less than 29.  
//Layer 2 query condition: To query the vertices whose vertexlabel is phone.  
lineSearch(api, graphName);  
private static void lineSearch(RestApi api, String graphName) {  
    LineSearchReqObj lineSearchReqObj = new LineSearchReqObj();  
    List<Integer> vertexIdList = new ArrayList<>();  
    String vertexId = getVertexIdByProperty(api, graphName, "person", "name", "marko");  
    vertexIdList.add(Integer.valueOf(vertexId));  
    lineSearchReqObj.setVertexIdList(vertexIdList);  
    List<VertexFilter> vertexFilterList = new ArrayList<>();  
    VertexFilter vertexFilter = new VertexFilter();  
    List<PropertyFilter> propertyFilterList1 = new ArrayList<>();  
    PropertyFilter propertyFilter = new PropertyFilter();  
    propertyFilter.setProperty("age");  
    propertyFilter.setPredicate(PropertyPredicate.LESS_THAN_EQUAL);  
    List<Integer> values2 = new ArrayList();  
    values2.add(29);  
    propertyFilter.setValues(values2);  
    propertyFilterList1.add(propertyFilter);  
    vertexFilter.setFilterList(propertyFilterList1);  
    vertexFilterList.add(vertexFilter);  
    vertexFilter = new VertexFilter();  
    List<String> vertexLabelList = new ArrayList<>();  
    vertexLabelList.add("phone");  
    vertexFilter.setVertexLabelList(vertexLabelList);  
    vertexFilterList.add(vertexFilter);  
    lineSearchReqObj.setVertexFilterList(vertexFilterList);  
    lineSearchReqObj.setLayer(2);  
    lineSearchReqObj.setLimit(5);  
    api.searchLines(lineSearchReqObj, graphName);  
}
```

The command output is as follows:

REQ HEADER: POST https://10.7.66.150:22380/graphbase/graph/graphbase/lines HTTP/1.1

REQ BODY: {"vertexIdList": [1261176], "layer": 2, "vertexFilterList": [{"filterList": [{"propertyName": "age", "predicate": "<=", "values": [29]}], "limit": 0}, {"vertexLabelList": ["phone"], "limit": 0}], "limit": 5}

RSP BODY: {

```
    "msg": "success",  
    "code": "0",  
    "data": {  
        "edgeList": [  
            {  
                "propertyList": [{  
                    "name": "weight",  
                    "value": "0.4"  
                }],  
                "start": "1261176",  
                "end": "1747680",  
                "id": "ew5r-r14o-bj9x-11gio",  
                "label": "knows",  
                "type": "edge"  
            },  
            {  
                "propertyList": [{  
                    "name": "weight",  
                    "value": "0.8"  
                }],  
                "start": "1508048",  
                "end": "1747680",  
                "id": "qlzu-wbm8-ecet-11gio",  
                "label": "has",  
                "type": "edge"  
            }  
        ]  
    }  
}
```

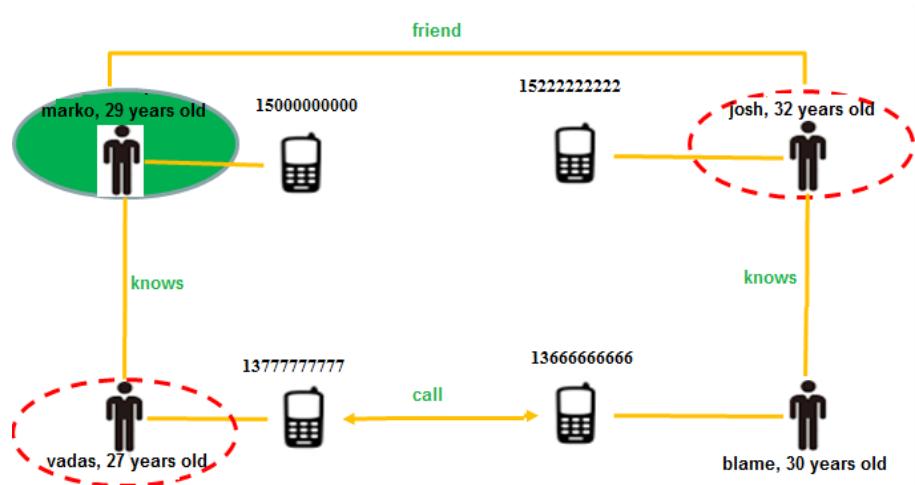
```
"vertexList": [
    {
        "labelList": ["person"],
        "propertyList": [
            {
                "name": "name",
                "value": "marko"
            },
            {
                "name": "age",
                "value": "29"
            }
        ],
        "id": "1261176",
        "type": "vertex"
    },
    {
        "labelList": ["person"],
        "propertyList": [
            {
                "name": "name",
                "value": "vadas"
            },
            {
                "name": "age",
                "value": "27"
            }
        ],
        "id": "1747680",
        "type": "vertex"
    },
    {
        "labelList": ["phone"],
        "propertyList": [
            {
                "name": "telephone",
                "value": "13777777777"
            }
        ],
        "id": "1508048",
        "type": "vertex"
    }
]
}
[SUCCESS]: search path successfully.
=====
===== Description of the JSON data result
=====
|---msg           message
|---code          code
|---data          Result data storage location
|---edgeList      All edges contained in the query result.
|---vertexList    All vertices contained in the query result.
=====
```

### 2.5.3.6.10 Analyzing the Typical Application Scenario

#### Direct Contact Search

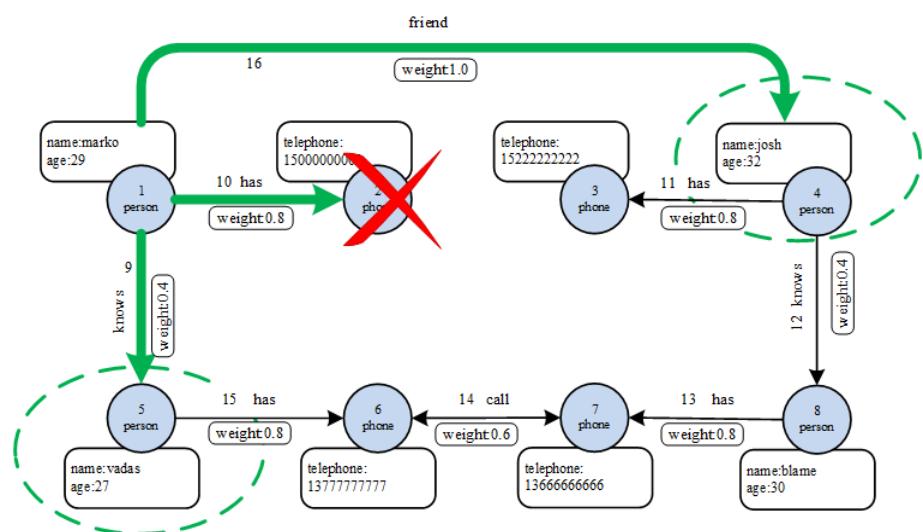
Direct contact: A person who has direct contact with a specified person.

The following graph shows the relationship between persons and their direct contacts in the typical scenario. (marko is used as an example. The direct contacts of marko are marked using the red dotted ovals.)



In GraphBase, the line expansion query function can be used to implement this scenario. **Figure 2-105** shows how to find the direct contacts for marko. (The green arrows indicate the expansion direction, the red cross mark indicates that the vertex does not meet the search condition, and the green dotted ovals indicate the search result.)

**Figure 2-105** Direct contact search in GraphBase



As shown in the preceding graph, josh and vadas are directly connected with marko. The related code is as follows:

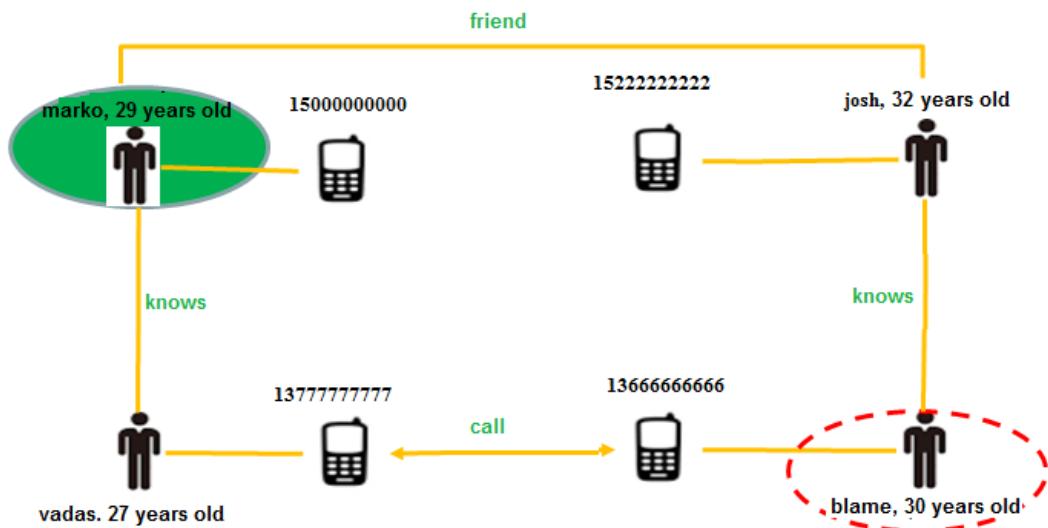
```
/*
 * <p> Search for the direct contact</p>
 * @param api REST API encapsulation
 * @param graphName Graph name
 * @param startNodeId Start node ID
 * NOTE: The search result is the response data of the end node.
 */
static String findDirectRelations(RestApi api, String graphName, String startNodeId)
{
    // Node preparation
    LineSearchReqObj lineSearchReqObj = new LineSearchReqObj();
    List<Integer> vertexIdList = new ArrayList<>();
    vertexIdList.add(Integer.valueOf(startNodeId));
    lineSearchReqObj.setVertexIdList(vertexIdList);
```

```
// Query depth
lineSearchReqObj.setLayer(1);
// End node condition (the label is person)
List<VertexFilter> vertexFilterList = new ArrayList<>();
VertexFilter vertexFilter = new VertexFilter();
List<String> vertexLabelList = new ArrayList<>();
vertexLabelList.add("person");
vertexFilter.setVertexLabelList(vertexLabelList);
vertexFilterList.add(vertexFilter);
lineSearchReqObj.setVertexFilterList(vertexFilterList);
// Perform line expansion query. (The query result is displayed as strings in JSON format.)
String result = api.searchLines(lineSearchReqObj, graphName);
return result;
}
```

## Indirect Contact Search

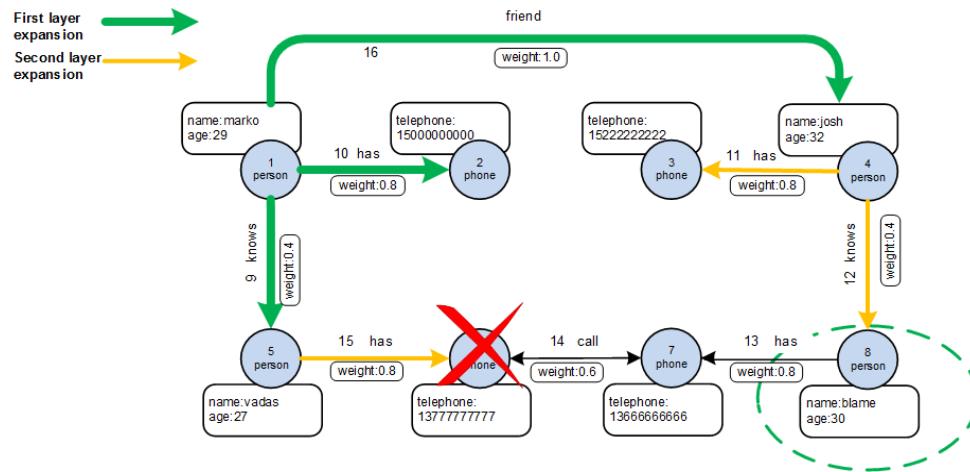
Indirect contact: A person who has indirect contact with a specified person.

The following graph shows the relationship between persons and their indirect contacts in the typical scenario. (marko is used as an example. The indirect contacts of marko are marked using the red dotted ovals.)



In GraphBase, the line expansion query can be used to find the indirect contact of marko in this scenario. [Figure 2-106](#) shows how the indirect contacts are searched in GraphBase. In the scenario, two layers are expanded: the green arrows indicate the query at the first layer, the yellow arrows indicate the query at the second layer. The red cross indicates that the vertex does not meet the search criteria, and the green dotted oval marks out the search result.

**Figure 2-106** Graph example of indirect contact search



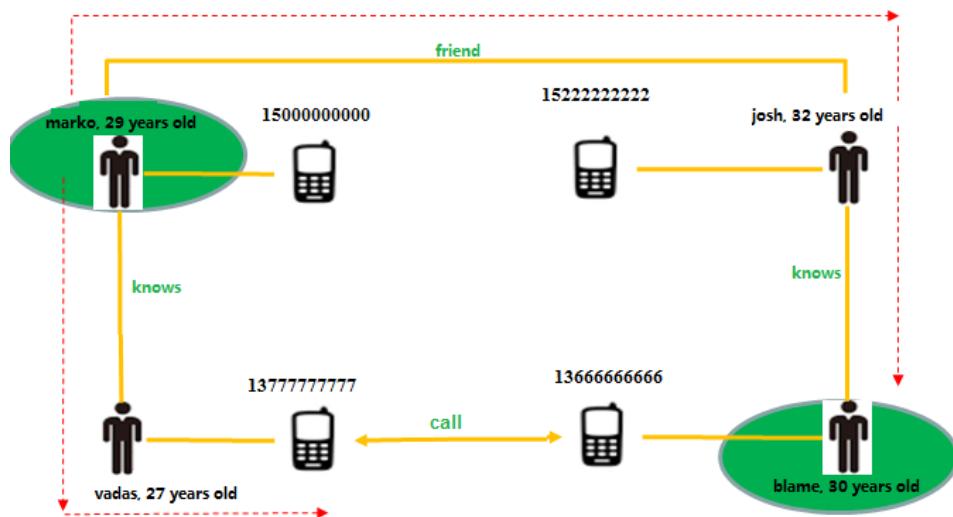
As shown in the preceding graph, only one link between marko and blame meets the search criteria. The related code is as follows:

```
/*
 * <p> Search for the indirect contact</p>
 * @param api REST API encapsulation
 * @param graphName Graph name
 * @param startNodeID Start node ID
 * NOTE: The search result is the response data of the end node.
 */
static String findIndirectRelations(RestApi api, String graphName, String startNodeID)
{
    // Node preparation
    LineSearchReqObj lineSearchReqObj = new LineSearchReqObj();
    List<Integer> vertexIdList = new ArrayList<>();
    vertexIdList.add(Integer.valueOf(startNodeID));
    lineSearchReqObj.setVertexIdList(vertexIdList);
    // Query depth
    lineSearchReqObj.setLayer(2);
    // End node condition (the vertex label is person)
    List<VertexFilter> vertexFilterList = new ArrayList<>();
    VertexFilter vertexFilter = new VertexFilter();
    List<String> vertexLabelList = new ArrayList<>();
    vertexLabelList.add("person");
    vertexFilter.setVertexLabelList(vertexLabelList);
    vertexFilterList.add(vertexFilter);
    lineSearchReqObj.setVertexFilterList(vertexFilterList);
    // Perform line expansion query. (The query result is displayed as strings in JSON format.)
    String result = api.searchLines(lineSearchReqObj, graphName);
    return result;
}
```

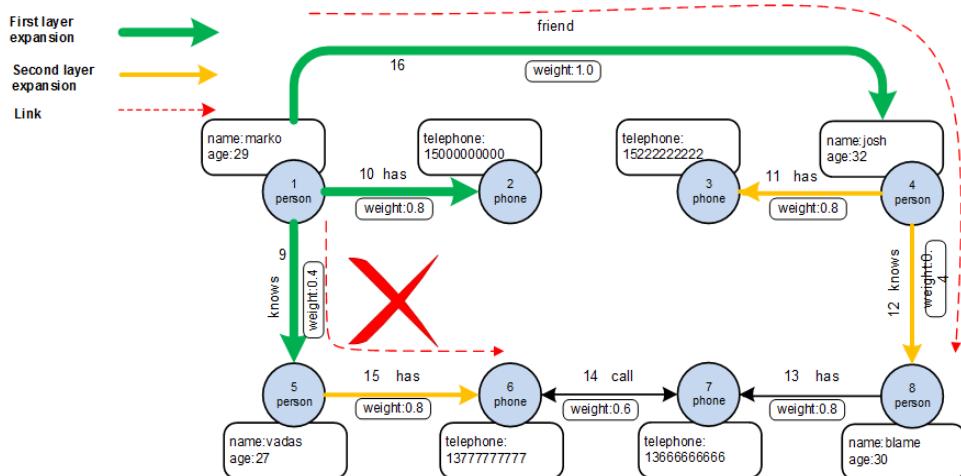
## Search for Link

Link refers to the link between two real world entities.

In the real world, the link between two persons is shown in the following graph. (The link between marko and blame is used as an example. The red dashed arrows indicate the link directions.)



In the link search scenario, the full path query function needs to be used. The following graph shows how to search for links in GraphBase. In this graph, the link between marko and blame is used as an example, and the path depth must be within two layers. The green arrows indicate the first layer expansion, the yellow arrow indicate the line expansion at the second layer, the red doted arrows indicate the link direction, and the red cross indicates that the link does not meet the requirements.



As shown in the preceding graph, only one link between marko and blame meets the search conditions. The related code is as follows:

```
/*
 * <p> Search for links </p>
 * @param api REST API encapsulation
 * @param graphName Graph name
 * @param startNodeId Start node ID
 * @param endNodeId End node ID
 * NOTE: The result data is the path set of the response data.
 */
static String findLinkedPaths(RestApi api, String graphName, String startNodeId, String endNodeId)
{
    // Node preparation
    PathSearchReqObj pathSearchReqObj = new PathSearchReqObj();
    List<String> vertexIdList = new ArrayList<>();
    vertexIdList.add(startNodeId);
    vertexIdList.add(endNodeId);
```

```
pathSearchReqObj.setVertexIdList(vertexIdList);
// Query depth.
pathSearchReqObj.setLayer(2);
// Perform full path query. (The query result is displayed as strings in JSON format).
String result = api.searchPath(pathSearchReqObj, graphName);
return result;
}
```

### 2.5.3.7 Gremlin Console

#### Prerequisites

You have installed all component clients required in the cluster.

#### Performing Query Using Gremlin Statements

1. Use PuTTY to log in to the node where the client is installed as user **root**.
2. Go to the **/GraphBase/gremlinserver/gremlin-console** directory where the GraphBase client is installed.

```
cd /opt/hadoopclient/GraphBase/gremlinserver/gremlin-console
```



In this example, the GraphBase client is installed in the /opt/hadoopclient directory.

3. Run the required environment variables as user **root**.  
**source /opt/hadoopclient/bigdata\_env**
4. Run the required environment variables as user **root**.  
**source /opt/hadoopclient/bigdata\_env**
5. Enter the Gremlin operation block.  
**sh bin/gremlin.sh**
6. Connect to the remote GraphBase.  
**:remote connect tinkerpop.server conf/remote-graphbase-gremlinserver.yaml**
7. Omit the remote command "**:**".  
**:remote console** (This operation is optional. If you perform this operation, you do not need to add "**:**" before the Gremlin language.)
8. Traverse a specified graph.  
**:remote config alias g Graph name**
9. Query a vertex, as shown in the following example. (For other application scenarios, see [Gremlin Application Scenarios](#).)  
gremlin> **:> g.V(13464736).valueMap()**  
==>[name:[zhangsan],age:[30]]  
gremlin> **:remote console**  
==>All scripts will now be sent to Gremlin Server - [szvphicprb00285/10.4.64.123:22375] - type  
'remote console' to return to local mode  
gremlin> **g.V(13464736).valueMap()**  
==>[name:[zhangsan],age:[30]]

```
[root@hd-123 gremlin-console]#source /opt/graphbaseClient/bigdata_env
[root@hd-123 gremlin-console]#kinit zky
Password for zky@HADOOP.COM:
[root@hd-123 gremlin-console]#sh bin/gremlin.sh
....o o
....o o-(3)-o o o-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
gremlin> :remote connect tinkerpop.server conf/remote-graphbase-gremlinserver.yaml
=>Configured hd-123/187.4.64.123:22381, hd-184/187.4.64.184:22381, HD-10/187.4.65.10:22381
gremlin> :remote config alias g gremlintest_1109
=>g:gremlintest_1109
gremlin> :remote console
=>All scripts will now be sent to Gremlin Server - [hd-123/187.4.64.123:22381, hd-184/187.4.64.184:22381, HD-10/187.4.65.10:22381]
gremlin> g.V().valueMap().limit(10)
=>[name:[blame],age:[30]]
=>[telephone:[13612493615]]
=>[telephone:[13729584211]]
=>[telephone:[13612493615]]
=>[telephone:[15023614521]]
=>[telephone:[15291463258]]
=>[name:[blame],age:[30]]
=>[name:[blame],age:[30]]
=>[telephone:[13612493615]]
=>[telephone:[15023614521]]
gremlin>
```

## 2.5.3.8 Gremlin Java

### 2.5.3.8.1 Connecting Gremlin Servers and Clients

- Modify the Gremlin client configuration file **remote-objects.yaml**.

Change the host name in hosts to the actual IP address.

```
hosts:[10.7.66.111,10.7.66.151,10.7.66.220]
port: 22381
connectionPool: {
  maxContentLength: 65536000,
  maxInProcessPerConnection: 60,
  maxSize: 16,
  resultIterationBatchSize: 960
}
protocol: gremlinserver
jaasEntry: GremlinConsole
serializer:
  { className: org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV1d0, config:
    { ioRegistries: [org.janusgraph.graphdb.tinkerpop.JanusGraphIoRegistry] } }
  serializer:
    { className: org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV1d0, config:
      { serializeResultToString: true } }
```

- Configure the mapping between the service IP Address of GraphServer and the host name in the local HOSTS file.

Directory of the HOSTS file on the Windows OS: **C:\Windows\System32\drivers\etc\hosts**

Directory of the HOSTS file on the Linux OS: **/etc/hosts**

```
#Mapping between the IP address and host name.
10.7.66.111 hd-111
10.7.66.151 hd-151
10.7.66.220 hd-220
```

- Connect the server and client.

Method 1:

```
//Connect the client and one or more Gremlin server instances.
Cluster cluster = GremlinClient.getInstance().getCluster();
//Create a client based on a cluster object.
Client client = cluster.connect().alias ("Graph name");
```

Method 2:

```
//Connect the client and one or more Gremlin server instances.
Cluster cluster = GremlinClient.getInstance().getCluster();
//Configure a remote graph traversal object on the client.
Graph graph = GremlinClient.getInstance().getGraph();
```

```
GraphTraversalSource g = graph.traversal().withRemote(DriverRemoteConnection.using (cluster,  
"Graph name"));
```

### 2.5.3.8.2 Running Gremlin Java API Example Code

- Submit Gremlin DSL statements as a client object.

```
//Submit Gremlin DSL statements synchronously.  
ResultSet resultSet = Client.submit("g.V().has('propertyKey', 'huawei').valueMap().limit(10)");  
//Submit Gremlin DSL statements asynchronously.  
Future<ResultSet> resultSet =  
Client.submitAsync("g.V().hasLabel('person').valueMap('name','age').limit(10)");  
//Generate the result.  
==>{name=[vadas], age=[27]}  
==>{name=[josh], age=[32]}  
==>{name=[blame], age=[30]}  
==>{name=[marko], age=[29]}
```

- Perform a graph traversal.

```
//Perform vertex traversal.  
Iterator<Map<String, Object>> iterator = g.V(ids).valueMap(properties).limit(10)  
//Perform edge traversal.  
Iterator<Map<String, Object>> iterator = g.E(ids).valueMap(properties).limit(10)  
//Generate the result.  
==>{name=[vadas], age=[27]}  
==>{name=[josh], age=[32]}  
==>{name=[blame], age=[30]}  
==>{name=[marko], age=[29]}
```

### 2.5.3.9 Gremlin Application Scenarios

#### 2.5.3.9.1 Querying a Specified Graph Object

- Query a specified vertex.

```
//Taverse all properties of the vertex based on the vertex ID.  
gremlin> g.V(1151560,1195696).valueMap()  
==>{name=[vadas], age=[27]}  
==>{name=[josh], age=[32]}
```

- Query a specified edge.

```
gremlin> g.E('fryh-oojs-93th-108hc', '102fq-pmls-6aol-vegw').valueMap()  
==>{weight=0.8}  
==>{weight=0.4}
```

#### 2.5.3.9.2 Queries One or More Properties

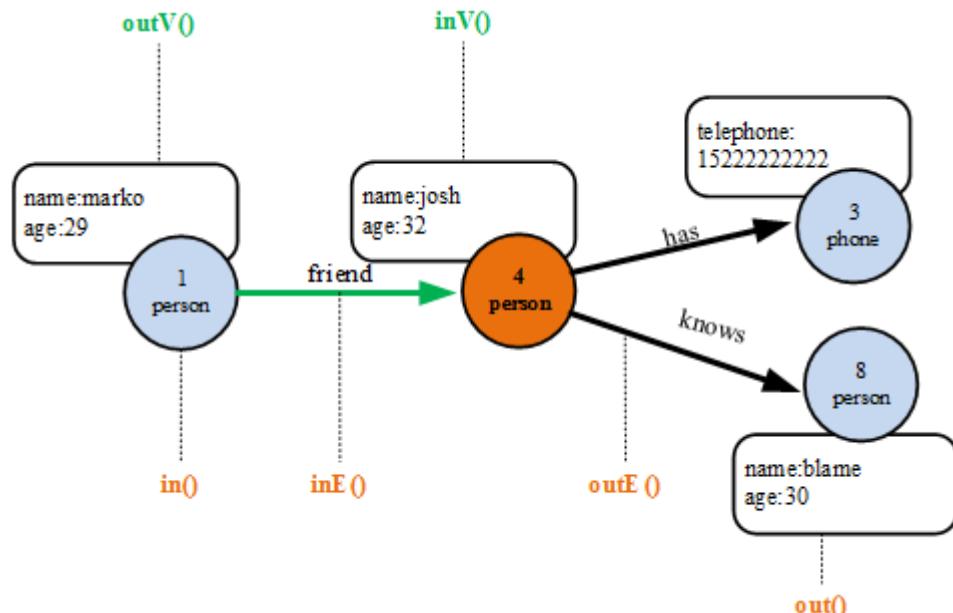
Querying one or more properties of a graph object.

```
gremlin> g.V().hasLabel('person').valueMap('name','age')  
==>{name=[vadas], age=[27]}  
==>{name=[josh], age=[32]}  
==>{name=[blame], age=[30]}  
==>{name=[marko], age=[29]}
```

#### 2.5.3.9.3 Traversing a Graph Object

| Graph Object<br>Traversal | Description   |
|---------------------------|---|
| Out(string )              | Move to the outgoing adjacent vertices given the edge labels. |

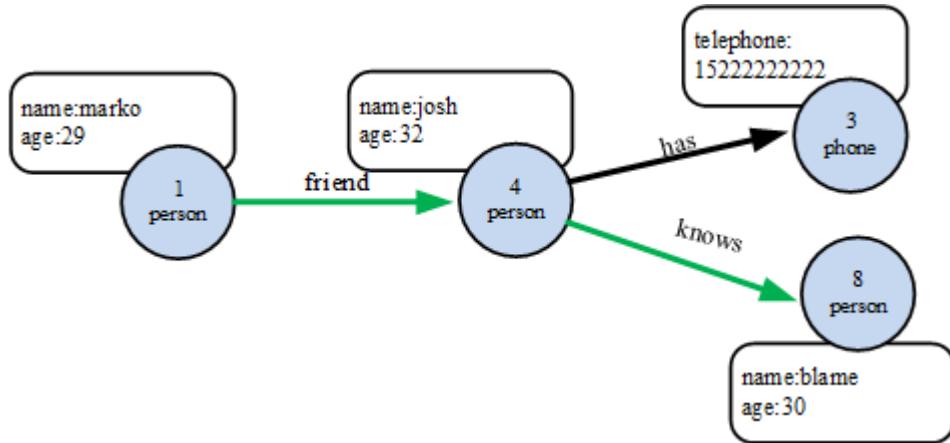
| Graph Object<br>Traversal | Description   |
|---------------------------|---|
| In(string )               | Move to the incoming adjacent vertices given the edge labels.                   |
| Both(string )             | Move to both the incoming and outgoing adjacent vertices given the edge labels. |
| OutE(string )             | Move to the outgoing incident edges given the edge labels.                      |
| InE(string )              | Move to the incoming incident edges given the edge labels.                      |
| BothE(string )            | Move to both the incoming and outgoing incident edges given the edge labels.    |
| outV()                    | Move to the outgoing vertex.  |
| inV()                     | Move to the incoming vertex.  |
| bothV()                   | Move to both vertices.  |



```
//Start from vertex 4 and traverse the edge to the out direction to obtain the incomming vertex of the edge.
gremlin> g.V(4).outE().inV()
==>v[3]
==>v[8]
//Move from vertex 4 to the out direction to obtain the peer vertex.
gremlin> g.V(4).out()
==>v[3]
==>v[8]
//Start from vertex 4 and traverse all edges to the out direction.
gremlin> g.V(4).outE()
==>e[10][4-has->3]
==>e[11][4-knows->8]
```

```
//Start from vertex 4 and traverse the edge to the out or in direction. The edge labels must be of a specified edge type, such as knows, has, or friend.
gremlin> g.V(4).bothE('knows','has','friend')
==>e[10][4-has->3]
==>e[11][4-knows->8]
==>e[8][1-friend->4]
```

#### 2.5.3.9.4 Traversing a Path



Function: path()

```
//Start from vertex 1 and traverse two layers to the out direction to obtain the peer vertex of the second layer.
gremlin> g.V(1).out().out().values('name')
==>blame
//Start from vertex 1 and traverse two layers to the out direction to return the track path of the traversal.
gremlin> g.V(1).out().out().values('name').path()
==>[v[1],v[4],v[8],blame]
```

#### 2.5.3.9.5 Filtering by Condition

- Filtering function: has()

| has                    | Description  |
|------------------------|--|
| has(key, value)        | Remove the traverser if its element does not have the provided key/value property.                     |
| has(label, key, value) | Remove the traverser if its element does not have the specified label and provided key/value property. |
| has(key, predicate)    | Remove the traverser if its element does not have a key value that satisfies the predicate.            |
| HasLabel(labels )      | Remove the traverser if its element does not have any of the labels.                                   |
| HasId(ids )            | Remove the traverser if its element does not have any of the IDs.                                      |
| HasKey(keys )          | Remove the traverser if the property does not have all of the provided keys.                           |

| has               | Description  |
|-------------------|--|
| HasValue(values ) | Remove the traverser if its property does not have all of the provided values. |
| has(key)          | Remove the traverser if its element does not have a value for the key.         |
| hasNot(key)       | Remove the traverser if its element has a value for the key.                   |

- Expression: predicate

| Predicate                | Description   |
|--------------------------|---|
| eq(object)               | Is the incoming object equal to the provided object?  |
| neq(object)              | Is the incoming object not equal to the provided object?  |
| lt(number)               | Is the incoming number less than the provided number?   |
| lte(number)              | Is the incoming number less than or equal to the provided number?                                   |
| gt(number)               | Is the incoming number greater than the provided number?  |
| gte(number)              | Is the incoming number less than or equal to the provided number?                                   |
| inside(number, number)   | Is the incoming number greater than the first provided number and less than the second?             |
| outside(number, number)  | Is the incoming number less than the first provided number or greater than the second?              |
| between(number, number)  | Is the incoming number greater than or equal to the first provided number and less than the second? |
| Within(objects )         | Is the incoming object in the array of provided objects?  |
| Without(objects )        | Is the incoming object not in the array of the provided objects?                                    |
| TextP.startsWith(string) | Does the incoming String start with the provided String?  |
| TextP.endsWith(string)   | Does the incoming String end with the provided String?  |
| TextP.contains(string)   | Does the incoming String contain the provided String?   |

| Predicate                     | Description  |
|-------------------------------|--|
| TextP.notStartingWith(string) | Does the incoming String not start with the provided String? |
| TextP.notEndingWith(string)   | Does the incoming String not end with the provided String?   |
| TextP.notContaining(string)   | Does the incoming String not contain the provided String?    |

```
//Traverse the vertices that are of the person type.
gremlin> g.V().hasLabel('person')
==>v[1151560]
==>v[1195696]
==>v[1465088]
==>v[1421128]
//Traverse the vertices with the age property.
gremlin> g.V().properties().hasKey('age').value()
==>27
==>32
==>30
==>29
//Traverse the vertices where the values of the age properties are within the range from 20 to 30.
gremlin> g.V().has('age',inside(20,30)).values('age')
==>29
==>27
//Traverse the veretxes where the value of the name property is josh or marko.
gremlin> g.V().has('name',within('josh','marko')).valueMap()
==>{name=[josh], age=[32]}
==>{name=[marko], age=[29]}
```

### 2.5.3.9.6 Sorting the Result

```
//Sort the result based on the value of the age property in ascending order.
gremlin> g.V().hasLabel('person').order().by('age', incr).valueMap('name','age')
==>{name=[vadas], age=[27]}
==>{name=[marko], age=[29]}
==>{name=[blame], age=[30]}
==>{name=[josh], age=[32]}
//Sort the result based on the value of the age property in descending order.
gremlin> g.V().hasLabel('person').order().by('age', decr).valueMap('name','age')
==>{name=[josh], age=[32]}
==>{name=[blame], age=[30]}
==>{name=[marko], age=[29]}
==>{name=[vadas], age=[27]}
```

### 2.5.3.9.7 Displaying the Result in Multiple Pages

```
//Sort the result based on the value of the age property in ascending order and only obtain the first two data records.
gremlin> g.V().hasLabel('person').order().by('age', incr).limit(2).valueMap('name','age')
==>{name=[vadas], age=[27]}
==>{name=[marko], age=[29]}
//Sort the result based on the value of the age property in ascending order and obtain data from the third to the fourth records.
gremlin> g.V().hasLabel('person').order().by('age', incr).range(2,4).valueMap('name','age')
==>{name=[blame], age=[30]}
==>{name=[josh], age=[32]}
```

### 2.5.3.9.8 Calculating the Intersection Between Two Data Sets

```
//During the traversal, the concept of the intersection between two data sets must be differentiated from the join operation in the SQL syntax.
```

```
gremlin> g.V().hasLabel('person').match(
    __.as('c').values('name').as('name'),//Data set 1. The column alias is name.
    __.as('c').out('friend','knows').values('name').as('friend_or_knows') //Data set 2. The column alias is friend_or_knows.
    ).select('name', 'friend_or_knows')//Calculate the intersection between the two data sets.
==>{name=josh, friend_or_knows=blame}
==>{name=marko, friend_or_knows=josh}
==>{name=marko, friend_or_knows=vadas}
```

### 2.5.3.9.9 Calculating the Union of Two Data Sets

```
//During the traversal, calculate the union of the following two data sets.
gremlin> g.V().has('name','josh').union(
    __.in('friend').values('name'),//Data set 1
    __.out('knows').values('name'))//Data set 2
==>marko
==>blame
```

### 2.5.3.9.10 Using Statistical Functions

| Function | Description                   |
|----------|-------------------------------|
| count()  | Collects data.                |
| max()    | Calculates the maximum value. |
| min()    | Calculates the minimum value. |
| mean()   | Calculates the average value. |
| sum()    | Sums up data.                 |

```
//Calculate the number of vertices whose type is person
gremlin> g.V().hasLabel('person').count()
==>4
//Traverse all vertices to obtain the maximum value of the age property.
gremlin> g.V().values('age').max()
==>32
```

## 2.5.4 Application Commissioning

### 2.5.4.1 Commissioning an Application in Windows

#### 2.5.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.



If IBM JDK is used in the Windows environment, the application cannot be run in the Windows environment.

## Procedure

In the development environment (for example, running the REST API on the IntelliJ IDEA environment), right-click **GraphBaseRestExample.java** and choose **Run > GraphBaseRestExample.main()**.

### 2.5.4.1.2 Viewing Windows Commissioning Results

#### Scenario

After the running of GraphBase application completes, you can view the running results in the IntelliJ IDEA running result.

#### Procedure

The following information is displayed in the running results:

```
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph HTTP/1.1
REQ BODY: {"graphName":"graphbase"}
RSP BODY: {
    "msg": "create graph success.",
    "code": "0"
}
[SUCCESS]: create graph[graphbase] successfully.
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
REQ BODY: {"name":"person"}
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "person"
    }
}
[SUCCESS]: create vertex label[person] successfully.
REQ HEADER: POST https://10.154.4.90:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
REQ BODY: {"name":"phone"}
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "phone"
    }
}
[SUCCESS]: create vertex label[phone] successfully.
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label/person
HTTP/1.1
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "person"
    }
}
[SUCCESS]: query vertex label[person] successfully.
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {"vertexLabelList": [
        {
            "name": "person"
        },
        {
            "name": "phone"
        }
    ]}
}
```

```
}
```

[SUCCESS]: query all vertex label successfully.

...

## 2.5.4.2 Commissioning an Application in Linux

### 2.5.4.2.1 Compiling and Running an Application

#### Scenario

GraphBase applications can run in a Linux OS. After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

#### Prerequisites

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by the JAR package exported by IntelliJIDEA.
- If the Linux host is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node. The host name and IP address must be in one-to-one mapping.

#### Procedure

##### Step 1 Export a JAR package.

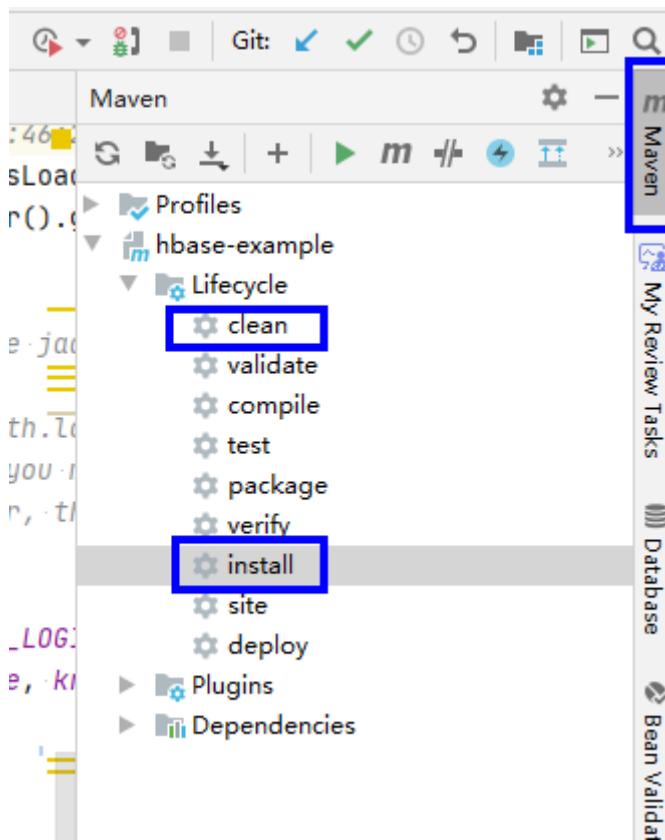
You can build a JAR package in either of the following ways:

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean**. Double-click **clean** to run the clean command of Maven.

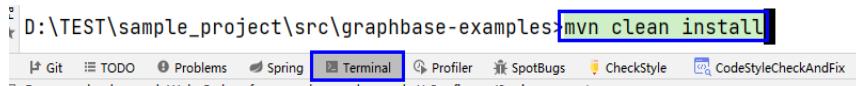
Choose **Maven** > *Sample project name* > **Lifecycle** > **install**. Double-click **install** to run the install command of Maven.

Figure 2-107 clean and install



- Method 2: Access the directory where the **pom.xml** file is located in the **Terminal** window at the bottom of the IDEA. Run the **mvn clean install** command to compile the file.

Figure 2-108 mvn clean install



After the compilation is complete, "Build Success" is printed, and the target directory is generated. The generated JAR package is stored in the target directory.

### Step 2 Export the JAR package on which the sample project depends.

Access the directory where the pom.xml file is located in the Terminal window or other command line tools at the bottom of the IDEA, and then run the following command:

```
mvn dependency:copy-dependencies -DoutputDirectory=lib
```

The lib folder is generated in the pom.xml directory, which contains the JAR package on which the sample project depends.

### Step 3 Prepare for the required JAR packages and configuration files.

- In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories**conf** and **lib**. Upload the JAR packages in **Step 1** and **Step 2**.

- 2 to the **lib** directory in Linux. Upload the conf configuration files in the sample project to the **conf** directory in Linux.
2. Import the user authentication credential files.  
Import the authentication credential files (**user.keytab** and **krb5.conf**) of the current user to the **conf** directory. On the FusionInsight Manager page, choose **System > Permission > User** to obtain the authentication credential files. Locate the row that contains the user whose authentication credentials need to be exported, and choose **More > Download Authentication Credential**.
3. In **/opt/test**, create the**GraphBase.sh** script, modify the following content, and save the file:

```
function Startup()
{
    SCRIPT_PATH=$(cd $(dirname $(readlink -f "${BASH_SOURCE[0]}")) && pwd )
    for f in ${SCRIPT_PATH}/lib/*.jar;
    do
        CLASSPATH=${CLASSPATH}:$f
    done
    JAVA_EXE=java
    JAVA_OPTS="$JAVA_OPTS -Xms2G -Xmx4G"
    ${JAVA_EXE} ${JAVA_OPTS} -classpath ${CLASSPATH}
    com.huawei.graphbase.example.GraphBaseRestExample
}
Startup
```

In the preceding content, "com.huawei.graphbase.example.GraphBaseRestExample" is used as an example.

**Step 4** Go to **/opt/test** and run the following commands to run the JAR packages:

**./graphbase.sh**

----End

### 2.5.4.2.2 Viewing Linux Commissioning Results

#### Scenario

After an GraphBase application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Viewing GraphBase logs.

#### Procedure

The following information is displayed in the running results:

```
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph HTTP/1.1
REQ BODY: {"graphName":"graphbase"}
RSP BODY: {
    "msg": "create graph success.",
    "code": "0"
}
[SUCCESS]: create graph[graphbase] successfully.
REQ HEADER: POST https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
REQ BODY: {"name":"person"}
RSP BODY: {
    "msg": "Success.",
```

```
"code": "0",
"data": {
    "name": "person",
    "isPartitioned": "false"
}
}
[SUCCESS]: create vertex label[person] successfully.
REQ HEADER: POST https://10.154.4.90:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
REQ BODY: {"name":"phone"}
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "phone",
        "isPartitioned": "false"
    }
}
[SUCCESS]: create vertex label[phone] successfully.
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label/person
HTTP/1.1
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {
        "name": "person",
        "isPartitioned": false
    }
}
[SUCCESS]: query vertex label[person] successfully.
REQ HEADER: GET https://10.4.64.110:22380/graphbase/graph/graphbase/schema/vertex-label HTTP/1.1
RSP BODY: {
    "msg": "Success.",
    "code": "0",
    "data": {"vertexLabelList": [
        {
            "name": "person",
            "isPartitioned": false
        },
        {
            "name": "phone",
            "isPartitioned": false
        }
    ]}
}
[SUCCESS]: query all vertex label successfully.
```

## 2.5.5 More Information

### 2.5.5.1 Common APIs

#### 2.5.5.1.1 REST API

For details about the complete and detailed description of GraphBase REST APIs, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

#### 2.5.5.1.2 Gremlin API

For details about the GraphBase Gremlin API, see [Gremlin Console](#), [Gremlin Java](#) and [Gremlin Application Scenarios](#).

## 2.6 HBase Development Guide

### 2.6.1 Overview

#### 2.6.1.1 Application Development Overview

##### HBase Introduction

HBase is a column-oriented scalable distributed storage system featuring high reliability and high performance. HBase is designed to break through the limitation when relational databases are used to process massive data.

HBase applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require efficient random read of massive data.
- Scenarios that require good scalability.
- Structured and unstructured data is concurrently processed.
- The Atomicity, Consistency, Isolation, Durability (ACID) feature supported by traditional relational databases is not required.
- HBase tables provide the following features:
  - Large: One table contains a hundred million rows and one million columns.
  - Column-oriented: Storage and rights control is implemented based on columns (families), and columns (families) are independently retrieved.
  - Sparse: Null columns do not occupy storage space, so a table is sparse.

##### Interface Type Introduction

The Java language is recommended for HBase application development because HBase is developed based on Java and Java is a concise, universal, and easy-to-understand language.

HBase adopts the same interfaces as those of Apache HBase. For details, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

**Table 2-60** describes the functions that HBase can provide by invoking interfaces.

**Table 2-60** Functions provided by HBase interfaces

| Function           | Description                                       |
|--------------------|---|
| Data CRUD function | Data creating, retrieving, updating, and deleting |

| Function            | Description                              |
|---------------------|--|
| Advanced feature    | Filter, secondary index, and coprocessor |
| Management function | Table management and cluster management  |

### 2.6.1.2 Common Concepts

- **Filter**

Filters provide powerful features to help users improve the table data processing efficiency of HBase. Users can use the filters predefined in HBase and customized filters.

- **Coprocessor**

Coprocessors enable users to perform region-level operations and provide functions similar to those of triggers in relational database management systems (RDBMSs).

- **Client**

Users can access the server from the client through the Java API, HBase Shell or WebUI to read and write HBase tables. The HBase client in this document indicates the HBase client installation package, see [External Interfaces](#).

### 2.6.1.3 Development Process

This document describes HBase application development based on the Java API.

For information about each phase in the development process, see [Figure 2-109](#) and [Table 2-61](#).

Figure 2-109 HBase application development process

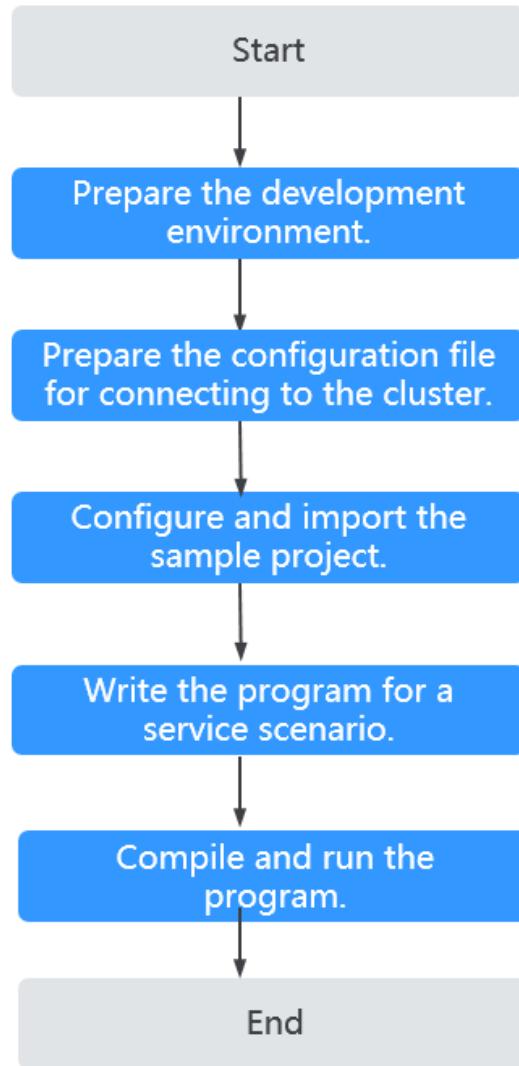


Table 2-61 HBase application development process description

| Phase                                | Description   | Reference Document  |
|--------------------------------------|---|---|
| Prepare the development environment. | Before developing an application, you need to prepare the development environment. You are advised to use the Java language and IntelliJ IDEA tool for development. In addition, you need to complete the initial configuration of the JDK and Maven. | <a href="#">Preparing for Development and Operating Environment</a> |

| Phase   | Description   | Reference Document  |
|---|---|---|
| Prepare the Configuration File for Connecting to the Cluster. | During application development or running, you need to connect to the MRS cluster through cluster configuration files. Configuration files include cluster component information files. You can obtain related content from the created MRS cluster.<br><br>Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured. | <a href="#">Preparing the Connection Cluster Configuration File</a> |
| Configure and import the sample project.                      | HBase provides multiple sample programs in different scenarios. You can obtain sample projects and import them to the local development environment for program learning.   | <a href="#">Configuring and Importing Sample Projects</a>           |
| Develop programs based on business scenarios.                 | Develop programs based on actual service scenarios and invoke component interfaces to implement corresponding functions.  | <a href="#">Developing an Application</a>                           |
| Compile and run the application.                              | You can commission and run the program in the local Windows development environment, or compile the program into a JAR package and submit it to the Linux node for running.   | <a href="#">Application Commissioning</a>                           |

#### 2.6.1.4 HBase Sample Project Introduction

To obtain the MRS sample project, visit <https://github.com/huaweicloud/huaweicloud-mrs-example>. Switch to the version branch that matches the MRS

cluster, download the package to the local PC, and decompress the package to obtain the sample code project of each component.

Currently, MRS provides the following HBase-related sample projects. You can select an example based on the actual service scenario:

**Table 2-62** HBase-related sample projects

| Sample Project Location             | Function Description  |
|-------------------------------------|---|
| hbase-examples/hbase-example        | <p>Application development example of HBase data read/write operations and global secondary indexes. The following functions can be implemented by calling HBase APIs:</p> <ul style="list-style-type: none"><li>• You can call HBase APIs to create user tables, import user data, add user information, query user information, and create secondary indexes for user tables. For details, see <a href="#">Service Scenario Description</a>.</li><li>• Creates and deletes global secondary indexes, modifies the status of global secondary indexes, and queries data based on global secondary indexes. For details about related service scenarios, see <a href="#">Service Scenario Description</a>.</li><li>• When multiple threads concurrently read data from or write data into an HBase table, region information can be preloaded. For details about related service scenarios, see <a href="#">Sample Program for Preloading Meta Tables When Request Concurrency Is High</a>.</li></ul> |
| hbase-examples/hbase-rest-example   | <p>HBase REST API development sample. Using REST APIs to query clusters, obtain tables, operate Namespaces, and operate tables. For details, see <a href="#">HBase Rest API Invoking Sample Program</a>.</p>  |
| hbase-examples/hbase-thrift-example | <p>Access the HBase ThriftServer development sample. Accessing ThriftServer to operate tables, writing data to tables, and reading data from tables. For details, see <a href="#">Accessing the HBase ThriftServer Sample Program</a>.</p>  |

| Sample Project Location         | Function Description   |
|---------------------------------|--|
| hbase-examples/hbase-zk-example | Application development example for HBase to access ZooKeeper.<br>The same client process accesses both the FusionInsight ZooKeeper and third-party ZooKeeper. The HBase client accesses the FusionInsight ZooKeeper, and the customer application accesses the third-party ZooKeeper. For details, see <a href="#">Sample Program for HBase to Access Multiple ZooKeepers</a> . |
| springboot/hbase-examples       | Provide an example of connecting Phoenix to SpringBoot.<br>This example describes how to interconnect HBase with Phoenix with Spring Boot. For details, see <a href="#">Example Configuration for Interconnecting HBase/Phoenix with SpringBoot</a> .  |

## 2.6.2 Environment Preparation

### 2.6.2.1 Preparing for Development and Operating Environment

[Table 2-63](#) describes the environment required for secondary development.

**Table 2-63** Development environment

| Item | Description   |
|------|---|
| OS   | <ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul> |

| Item   | Description   |
|--|---|
| Installing JDK                               | <p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul> |
| IntelliJ IDEA installation and configuration | <p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>  |
| JUnit plug-in installation                   | Basic configuration for the development environment   |

| Item                  | Description  |
|-----------------------|--|
| Installation of Maven | Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.<br><br>Huawei provides an open-source mirror site, Huawei Mirrors. You can download the supportive JAR packages of the sample projects from this site. You can download the rest open-source JAR packages from the Maven central repository or other user-defined repositories. For details, see <a href="#">Configuring Huawei Open-Source Mirrors</a> . |
| 7-zip                 | Used to decompress .zip and .rar packages.<br>7-Zip 16.04 is supported.  |

## 2.6.2.2 Preparing the Connection Cluster Configuration File

### Preparing an Operating Environment

During application development or running, you need to connect to the MRS cluster through cluster configuration files. Configuration files include cluster component information files. You can obtain related content from the created MRS cluster.

Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured.

- Scenario 1: Prepare the configuration files required by the commissioning program in the local Windows development environment.
  - a. Log in to the FusionInsight Manager, click **Download Client** in the upper right corner of the Homepage, set **Select Client Type** to **Configuration Files Only** and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client configuration file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC.
  - b. Go to the client configuration file decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\HBase\config**, obtain related configuration files from the [Table 2-64](#).

**Table 2-64** Configuration files

| file           | Function                           |
|----------------|------------------------------------|
| core-site.xml  | Configures Hadoop Core parameters. |
| hbase-site.xml | Configures HBase parameters.       |
| hdfs-site.xml  | Configures HDFS parameters.        |

 NOTE

To obtain the configuration file required by the HBase ThriftServer sample project, see [Preparing the ThriftServer Instance Configuration File](#).

- c. Copy the content in the **hosts** file in the decompression directory to the local **hosts** file.

 NOTE

- During application development, if you need to commission applications in the local Windows system, ensure that the local node can communicate with the hosts listed in the hosts file.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- When configuring IPv6 hosts on the local PC running Windows, add % and InterfaceIndex of the network interface card (NIC) to the end of the IPv6 address. The InterfaceIndex can be obtained by running the **ipconfig** command.

Example:

fec1:0:0:e505:8:99:5:1%10

- Scenario 2: Prepare the configuration file required for running programs in the Linux environment.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.
  - b. To obtain the configuration file.
    - i. Log in to the FusionInsight Manager, click **Download Client** in the upper right corner of the Homepage, set **Select Client Type** to **Configuration Files Only** and click **OK**. Download the client configuration file to the active OMS node in the cluster.
    - ii. Log in to the active OMS node as user **root**, go to the directory where the client configuration file is stored (the default directory is **/tmp/FusionInsight-Client/**), and decompress the software package to obtain the configuration files in the **Table 2-64** in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/HBase/config** directory.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, the download path is **/tmp/FusionInsight-Client** on the active management node.

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
```

```
tar -xvf  
FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles
```

 NOTE

HBase ThriftServer sample project, see [Preparing the ThriftServer Instance Configuration File](#).

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## Preparing the ThriftServer Instance Configuration File

To access HBase ThriftServer and perform table-related operations, perform the following steps:

- Step 1** Log in to FusionInsight Manager, choose **Cluster > Service > HBase > Configuration** and click **All Configurations**, search for and modify the parameter **hbase.thrift.security.qop** of the ThriftServer instance. The value of this parameter must be the same as that of **hbase.rpc.protection**. Save the configuration and restart the node service for the configuration to take effect.

 NOTE

The mapping between **hbase.rpc.protection** and **hbase.thrift.security.qop** is as follows:

- "privacy" - "auth-conf"
- "authentication" - "auth"
- "integrity" - "auth-int"

- Step 2** Obtain the configuration file of the ThriftServer instance in the cluster.

- Method 1: Choose **Cluster > Service > HBase > Instance**, click the ThriftServer instance to go to the details page. In the **Configuration File** area of the **Dashboard** page, click the names of the configuration files **hdfs-site.xml**, **core-site.xml**, and **hbase-site.xml** to obtain the configuration files.
- Method 2: Obtain the configuration files by decompressing the client file in [Preparing an Operating Environment](#). Manually add the following configuration to **hbase-site.xml**. The value of **hbase.thrift.security.qop** must be the same as that in [Step 1](#).

```
<property>  
<name>hbase.thrift.security.qop</name>  
<value>auth</value>  
</property>  
<property>  
<name>hbase.thrift.kerberos.principal</name>  
<value>thrift/hadoop.hadoop.com@HADOOP.COM</value>  
</property>  
<property>  
<name>hbase.thrift.keytab.file</name><value>/opt/huawei/Bigdata/FusionInsight_HD_XXX/install/  
FusionInsight-HBase-2.4.14/keytabs/HBase/thrift.keytab</value>  
</property>
```

----End

### 2.6.2.3 Configuring and Importing Sample Projects

#### Background

Obtain the HBase development example project . Import the project to IntelliJ IDEA for learning.

#### Prerequisites

Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the MRS cluster time in the bottom-right corner on FusionInsight Manager.

- Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the MRS cluster time in the bottom-right corner on FusionInsight Manager.
- The development environment and MRS cluster configuration files have been prepared. For details, see [Preparing the Connection Cluster Configuration File](#).

#### Procedure

- Step 1** Obtain the sample project from the `src` directory in the directory where the sample code is decompressed by referring to [Obtaining Sample Projects from Huawei Mirrors](#). You can select an example based on the actual service scenario. For details about the example, see [HBase Sample Project Introduction](#).
- Step 2** If you need to debug the sample code on the local Windows PC, place the configuration files required by each sample project as required in [Table 2-65](#).

**Table 2-65** Configuration files required by each sample project

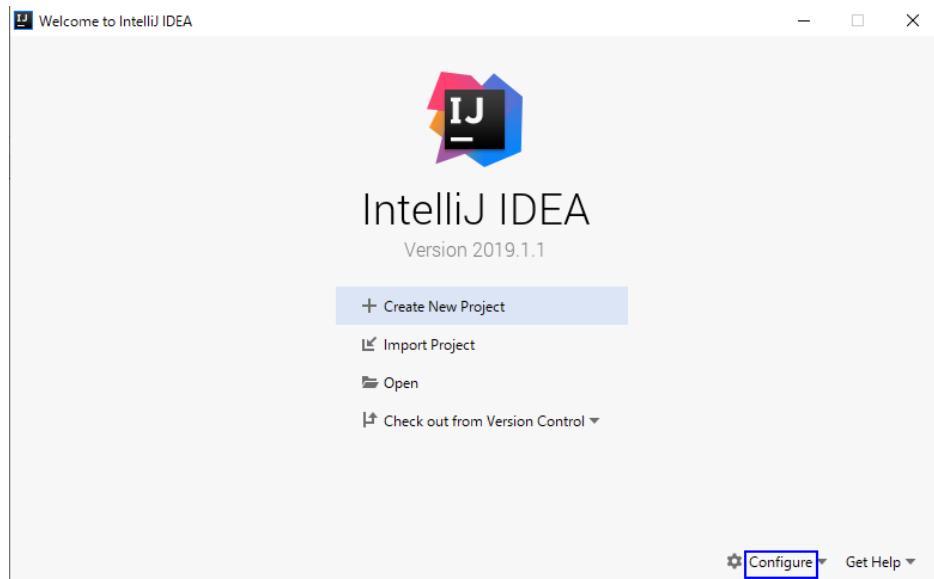
Sample Project Location	Configuration/authentication files to be placed
hbase-examples/hbase-example (single cluster)	Save the <code>core-site.xml</code> , <code>hbase-site.xml</code> , and <code>hdfs-site.xml</code> files obtained in the <a href="#">Preparing an Operating Environment</a> to the <code>../src/main/resources/conf</code> directory of the sample project.

Sample Project Location	Configuration/authentication files to be placed
hbase-examples/hbase-example (multi-cluster)	<p>Place the configuration file of a cluster with the same user name in the mutual trust scenario to the <code>..src/main/resources/hadoopDomain</code> directory, and place the configuration file of the other cluster to the <code>..src/main/resources/hadoop1Domain</code> directory.</p> <p>The configuration files are <code>core-site.xml</code>, <code>hbase-site.xml</code>, and <code>hdfs-site.xml</code> obtained in section <a href="#">Preparing an Operating Environment</a>.</p>
hbase-examples/hbase-rest-example	-
hbase-examples/hbase-thrift-example	<p>Save the <code>core-site.xml</code>, <code>hbase-site.xml</code>, and <code>hdfs-site.xml</code> files obtained in the <a href="#">Preparing the ThriftServer Instance Configuration File</a> to the <code>..src/main/resources/conf</code> directory of the sample project.</p>
hbase-examples/hbase-zk-example	<p>Place the following files in the <code>..src/main/resources/conf</code> directory of the sample project:</p> <ul style="list-style-type: none"> <li>The configuration files are <code>core-site.xml</code>, <code>hbase-site.xml</code>, and <code>hdfs-site.xml</code> obtained in <a href="#">Preparing an Operating Environment</a>.</li> <li>Ensure that the <code>zoo.cfg</code> and <code>jaas.conf</code> files required for authentication in the <a href="#">Accessing Multiple ZooKeepers</a> exist in the directory.</li> </ul>
springboot/hbase-examples	<p>Save the <code>core-site.xml</code>, <code>hbase-site.xml</code>, and <code>hdfs-site.xml</code> files obtained in <a href="#">Preparing an Operating Environment</a> to the local Windows operating system.</p>

**Step 3** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.

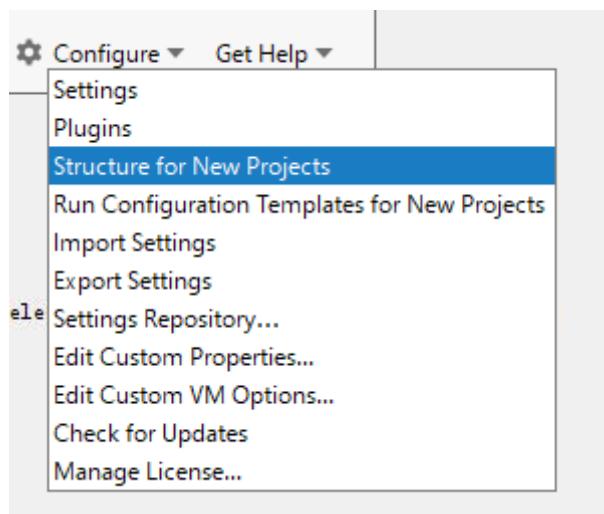
1. Start the IntelliJ IDEA and click **Configure**.

Figure 2-110 Quick Start



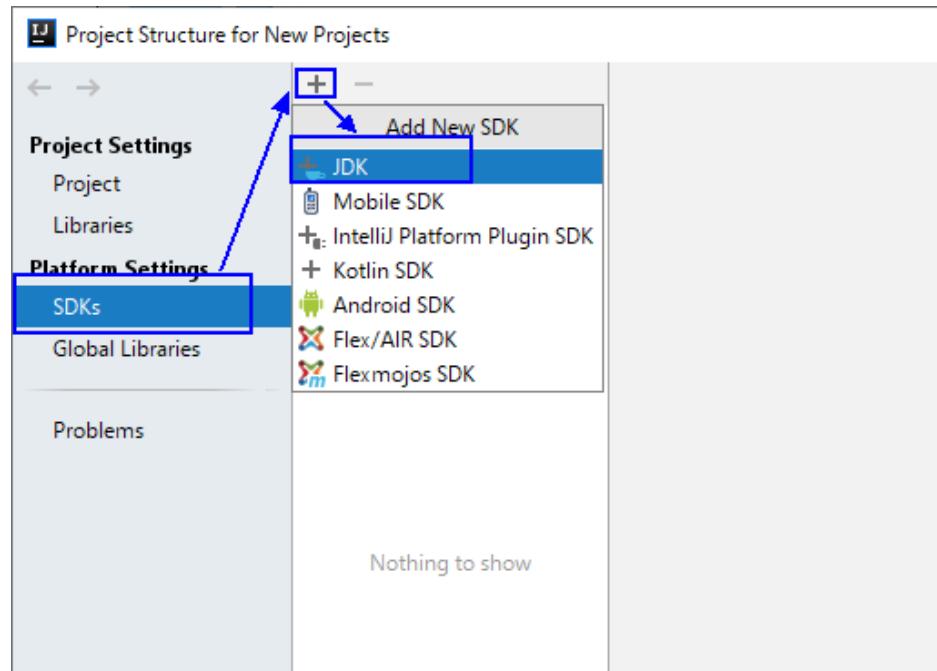
2. Click **Structure for New Projects** from the drop-down list.

Figure 2-111 Configure



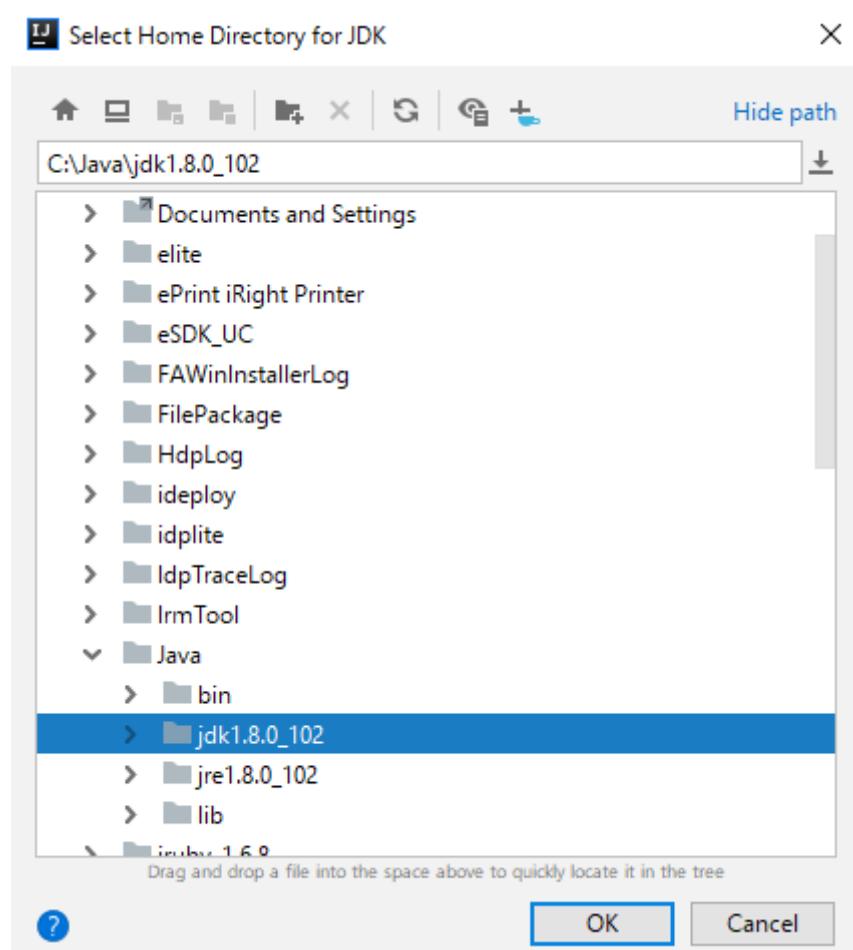
3. On the displayed **Project Structure for New Projects** page, select **SDKs** and click the plus sign (+) to add the **JDK**.

Figure 2-112 Project Structure for New Projects



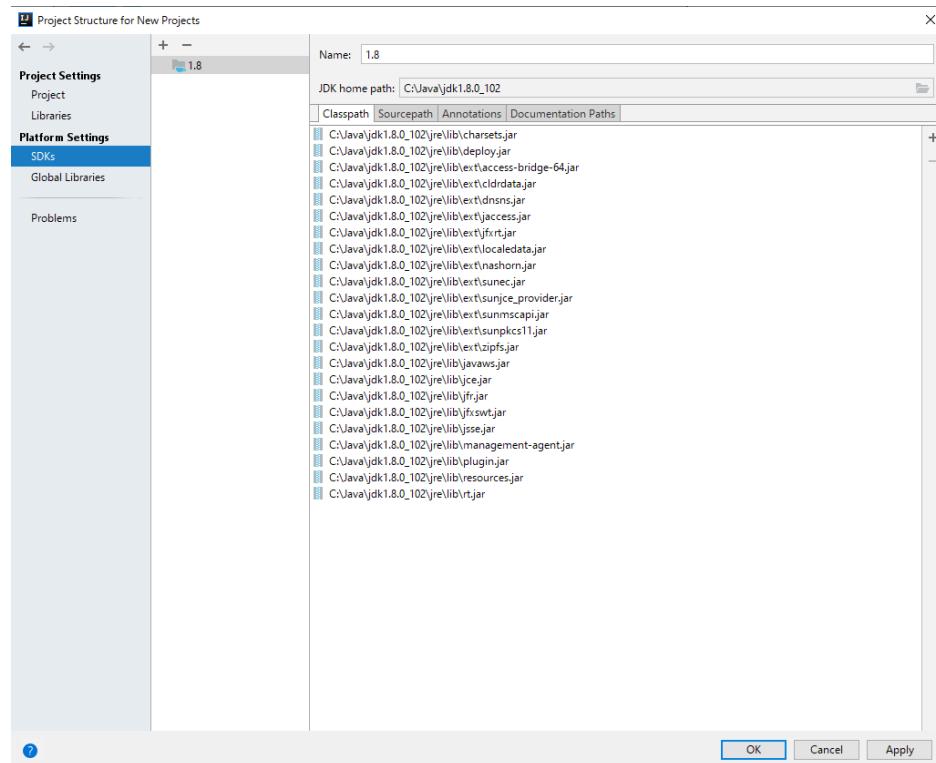
4. In the **Select Home Directory for JDK** window that is displayed, select the JDK directory, and click **OK**.

Figure 2-113 Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.

Figure 2-114 Complete JDK configuration



**NOTE**

The operations vary by the IDEA version. The operation procedure varies according to the actual version.

**Step 4** Import the example project to the IntelliJ IDEA development environment.

1. Start the IntelliJ IDEA, and click **Import Project** on the Open or Import.

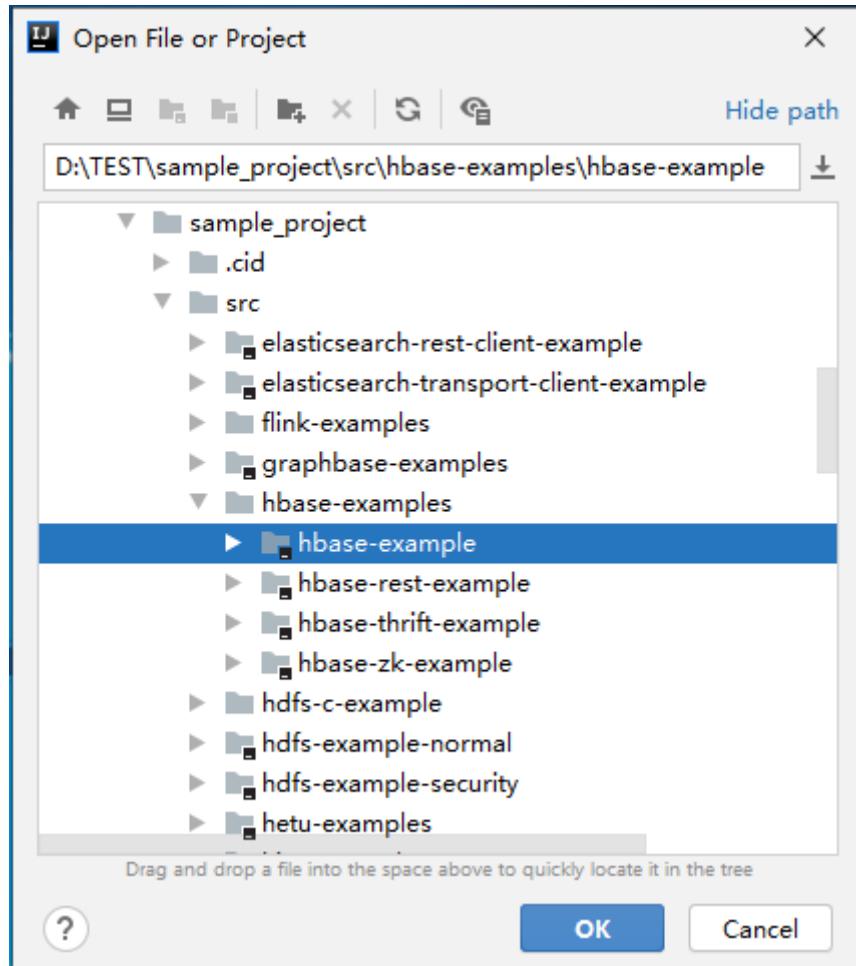
For the IDEA tool that has been used, you can choose **File > Import project...** on the main interface to import the sample project.

Figure 2-115 Open or Import (Quick Start page)



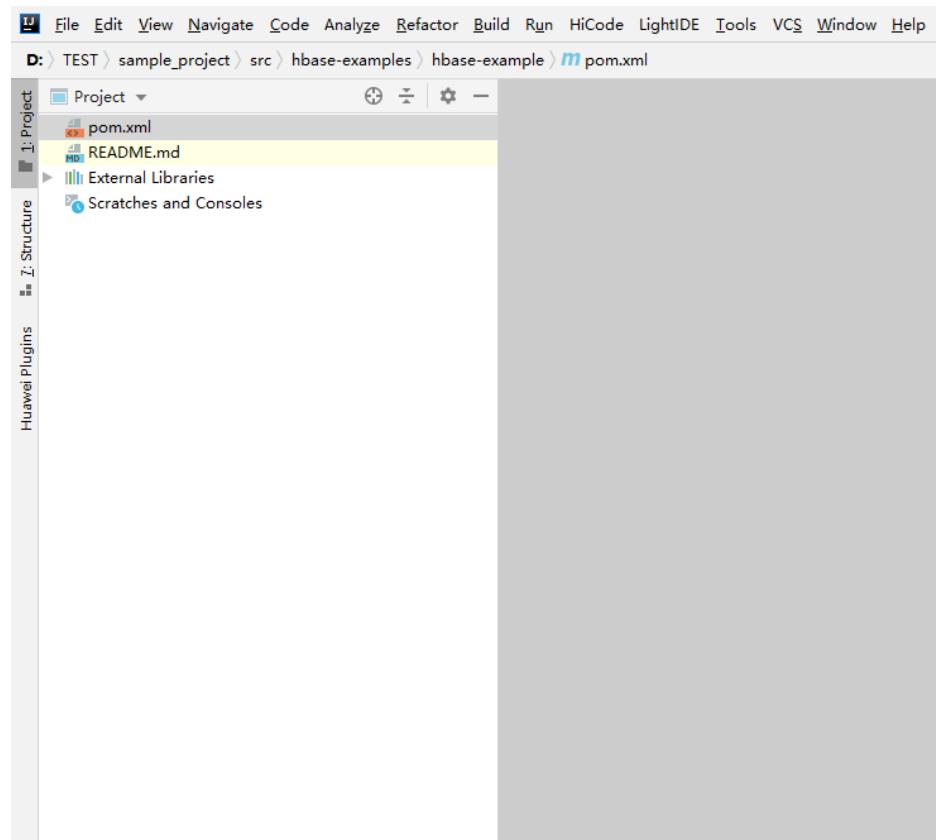
2. Select the example project folder **hbase-example**, and click **OK**.

**Figure 2-116** Select File or Directory to Import



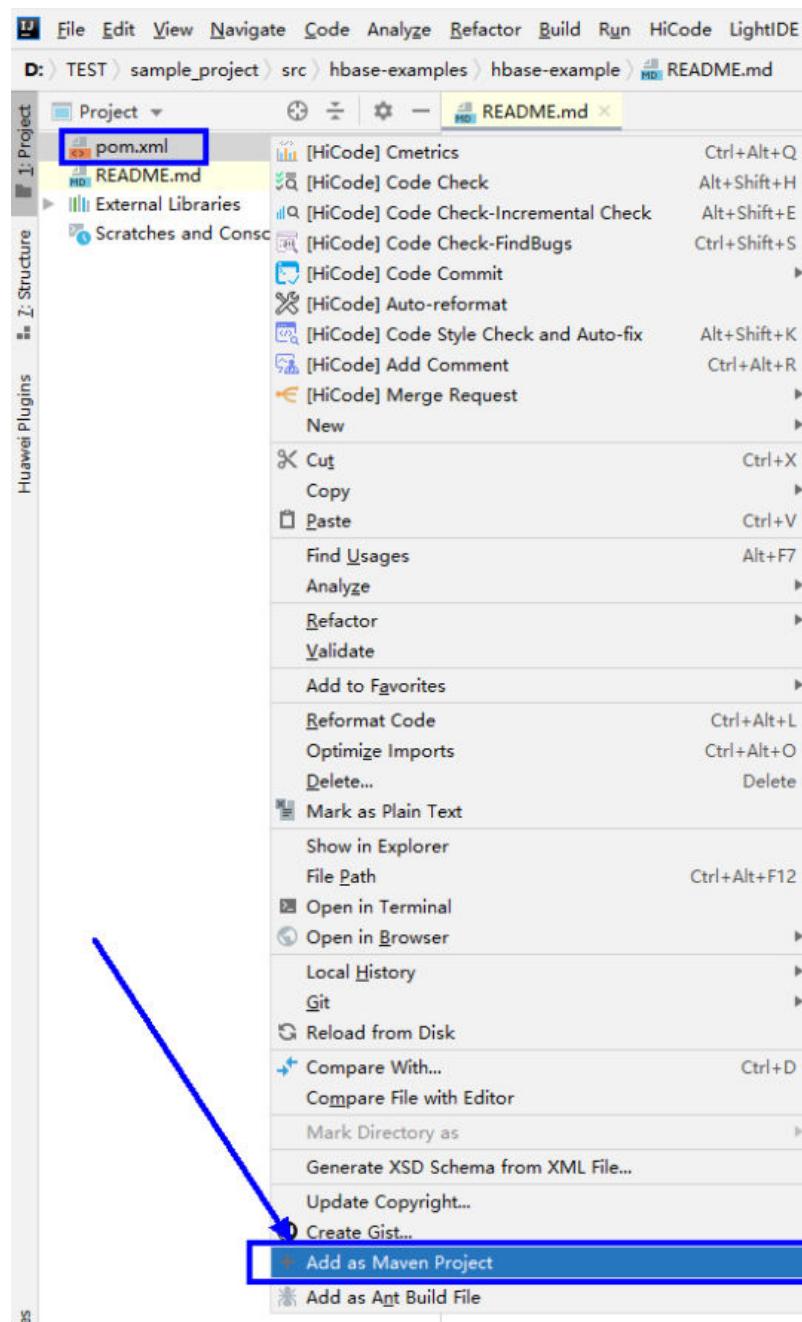
3. After the import, the imported sample project is displayed on the IDEA home page.

Figure 2-117 Successfully importing the sample project



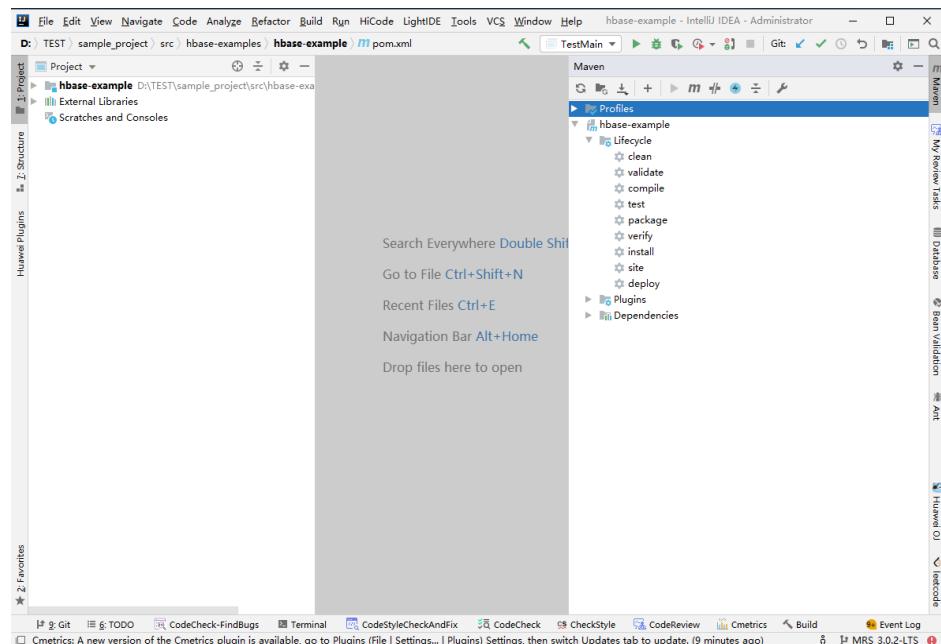
4. Right-click **pom.xml** and choose **Add as Maven Project** from the shortcut menu to add the project as a Maven Project. If the **pom.xml** icon is as shown in , go to the next step.

Figure 2-118 Add as Maven Project



In this case, the IDEA can identify the project as a Maven project.

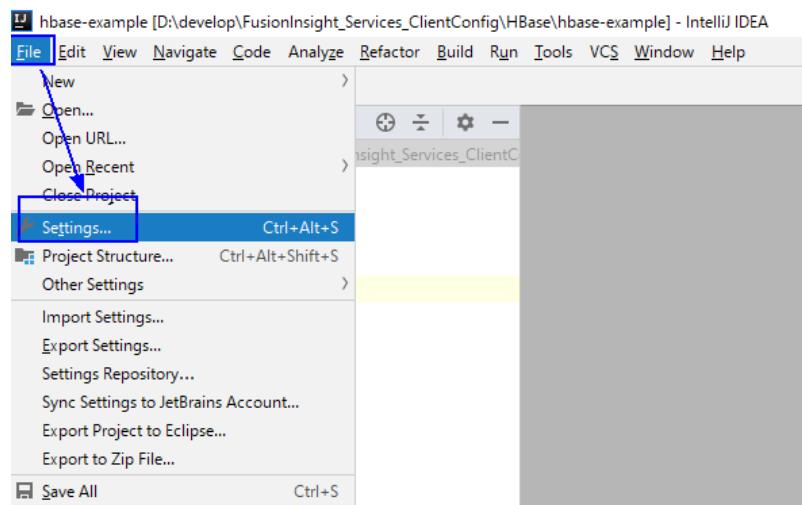
Figure 2-119 Sample project displayed in IDEA as a Maven project



### Step 5 Set the Maven version used by the project.

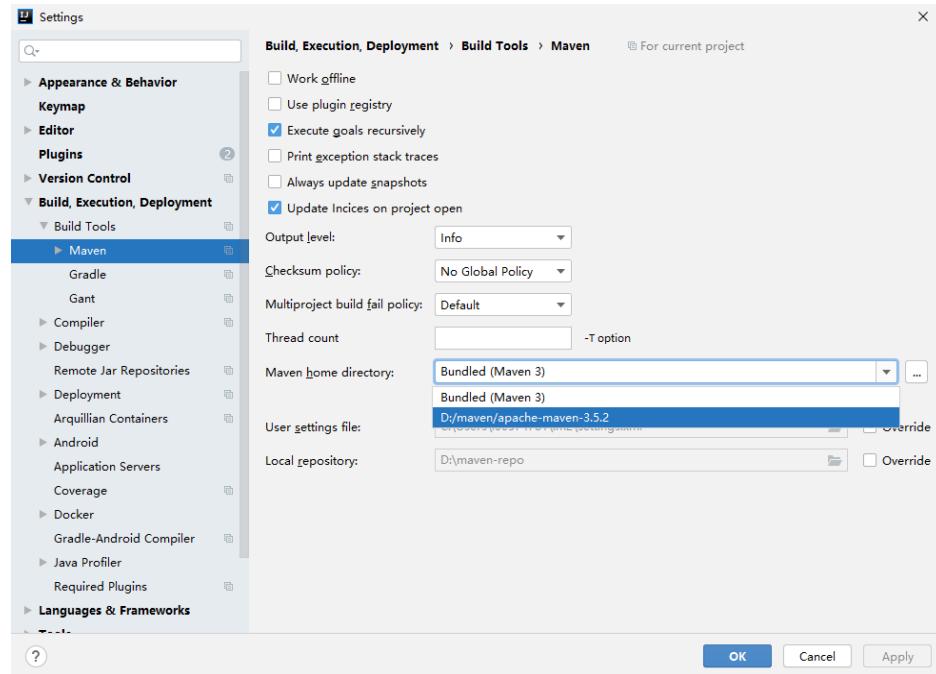
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

Figure 2-120 Settings



2. Choose **Build,Execution,Deployment > Maven** and set **Maven home directory** to the Maven version installed on the local host.  
Set **User settings file** and **Local repository** parameters based on the site requirements, and choose **Apply > OK**.

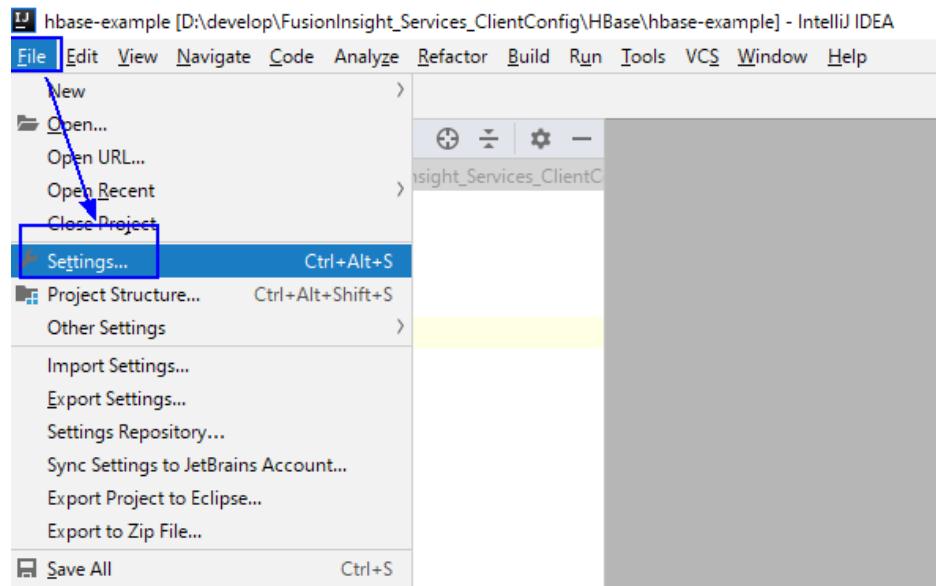
Figure 2-121 Selecting the local Maven installation directory



**Step 6** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

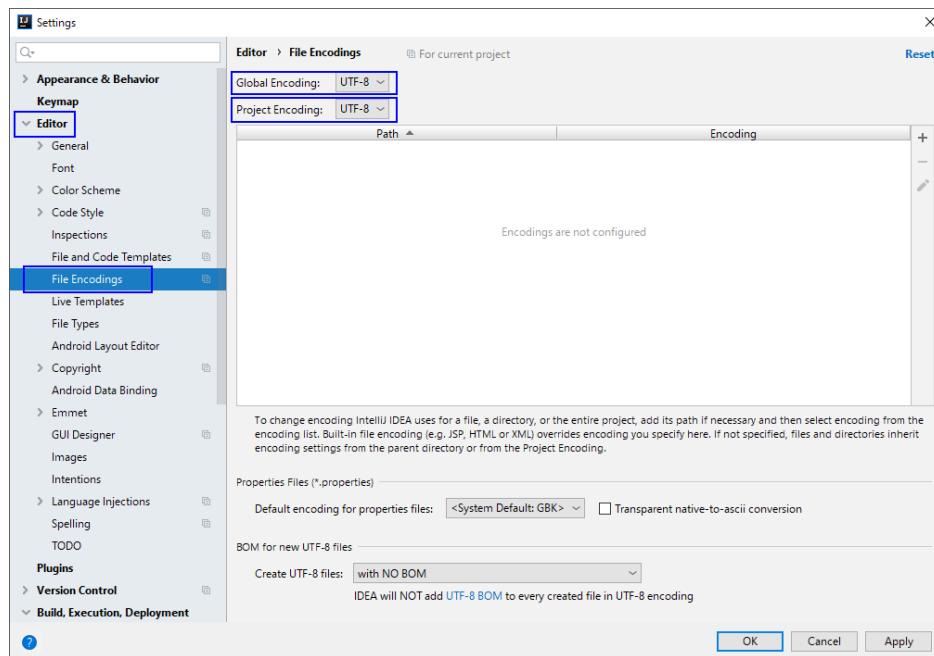
1. On the IntelliJ IDEA menu bar, choose **File > Settings....**

Figure 2-122 Settings



2. In the navigation tree of the **Settings** page, choose **Editor > File Encodings**, and select **UTF-8** for **Global Encoding** and **Project Encoding**.

Figure 2-123 File Encodings



- Click **Apply** and **OK** to complete the encoding configuration.

----End

## 2.6.3 Developing an Application

### 2.6.3.1 HBase Data Read/Write Sample Program

#### 2.6.3.1.1 Service Scenario Description

Based on typical scenarios, users can quickly learn and master the HBase development process and know important interface functions.

#### Scenario

Develop an application to manage information about users who use service A in an enterprise. **Table 2-66** provides the user information. The operation process of service A is described as follows:

- Create a user information table.
- Add diplomas and titles to the user information table.
- Query user names and addresses by user ID.
- Query information by user name.
- Query information about users whose ages range from 20 to 29.
- Collect statistics on the number and the maximum, minimum, and average ages of users.
- Deregister users, and delete user data.
- Delete the user information table after service A ends.

**Table 2-66** User information

ID	Name	Gender	Age	Address
12005000201	Zhang San	Male	19	Shenzhen, Guangdong
12005000202	Li Wanting	Female	23	Shijiazhuang, Hebei
12005000203	Wang Ming	Male	26	Ningbo, Zhejiang
12005000204	Li Gang	Male	18	Xiangyang, Hubei
12005000205	Zhao Enru	Female	21	Shangrao, Jiangxi
12005000206	Chen Long	Male	32	Zhuzhou, Hunan
12005000207	Zhou Wei	Female	29	Nanyang, Henan
12005000208	Yang Yiwen	Female	30	Kaixian, Chongqing
12005000209	Xu Bing	Male	26	Weinan, Shaanxi
12005000210	Xiao Kai	Male	25	Dalian, Liaoning

## Data Planning

Proper design of the table structure, RowKeys, and column names can give full play to the advantages of HBase. In this example, the unique ID is used as the RowKey, and columns are stored in the **info** column family.

---

**⚠ CAUTION**

HBase tables are stored in *Namespace:Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table namespace.

---

### 2.6.3.1.2 Application Development Approach

#### Function Decomposition

Functions are decomposed based on the preceding service scenario. **Table 2-67** provides the functions to be developed.

**Table 2-67** Functions to be developed in HBase

No.	Procedure	Code Implementation
1	Create a table based on <a href="#">Table 2-66</a> .	For details, see <a href="#">Creating a Table</a> .
2	Import user data.	For details, see <a href="#">Inserting Data</a> .
3	Add the <b>Education</b> column family, and add diplomas and titles to the user information table.	For details, see <a href="#">Modifying a Table</a> .
4	Query user names and addresses by user ID.	For details, see <a href="#">Reading Data Using Get</a> .
5	Query information by user name.	For details, see <a href="#">Filtering Data</a> .
6	To improve query performance, create or delete secondary indexes.	For details, see <a href="#">Creating a Secondary Index</a> and <a href="#">Secondary Index-based Query</a> .
7	Deregister users, and delete user data.	For details, see <a href="#">Deleting Data</a> .
8	Delete the user information table after service A ends.	For details, see <a href="#">Deleting a Table</a> .

## Key Design Principle

HBase is a distributed database system based on the lexicographic order of RowKeys. The RowKey design has great impact on performance, so the RowKeys must be designed based on specific services.

### 2.6.3.1.3 Configuring Log4j Log Output

#### Function Description

This section describes how to output HBase client logs to a specified log file separately from service logs to facilitate HBase problem analyzing and locating. If the **log4j** configuration exists in the process, copy the RFA and RFAS configurations in **hbase-example\src\main\resources\log4j.properties** to the existing **log4j** configuration.

#### Sample Code

```
hbase.root.logger=INFO,console,RFA      //HBase client log output configuration. console: outputs to  
the console; RFA: outputs to the log files.  
hbase.security.logger=DEBUG,console,RFAS    //HBase client security logs output configuration. console:  
outputs to the console; RFAS: outputs the log files.  
hbase.log.dir=/var/log/Bigdata/hbase/client/ //Log directory. Modify based on the actual directory. Ensure  
that the directory has the write permission.  
hbase.log.file=hbase-client.log          //Log file name
```

```
hbase.log.level=INFO          //Log level. If detailed logs are required for fault locating, change it  
to DEBUG. The modification takes effect after restart.  
hbase.log.maxbackupindex=20    //Maximum number of log files that can be saved.  
# Security audit appender  
hbase.security.log.file=hbase-client-audit.log //Command of the audit log file
```

### 2.6.3.1.4 Creating Configuration

#### Function

HBase obtain configuration items by using the login method. The configuration items include user login information and security authentication information.

#### Example Codes

The following code snippet belongs to the **createClientConf** method in the **createClientConf** class of the **com.huawei.bigdata.hadoop.security.examples** package.

```
public static Configuration createClientConf() {  
    // In Windows environment  
    String userDir = Utils.class.getClassLoader().getResource(CONF_DIRECTORY).getPath() + File.separator;  
    // In Linux environment  
    // String userDir = System.getProperty("user.dir") + File.separator + CONF_DIRECTORY + File.separator;  
    return createConfByUserDir(userDir);  
}  
public static Configuration createConfByUserDir(String userDir) {  
    // Default load from conf directory  
    Configuration conf = HBaseConfiguration.create();  
    if (userDir == null || userDir.isEmpty()) {  
        return conf;  
    }  
    conf.addResource(new Path(userDir + CLIENT_CORE_FILE), false);  
    conf.addResource(new Path(userDir + CLIENT_HDFS_FILE), false);  
    conf.addResource(new Path(userDir + CLIENT_HBASE_FILE), false);  
    return conf;  
}
```

### 2.6.3.1.5 Creating Connection

#### Function

HBase creates a Connection object using the **ConnectionFactory.createConnection(configuration)** method. The transferred parameter is the Configuration created in the last step.

Connection encapsulates the connections between underlying applications and servers and ZooKeeper. Connection is instantiated using the **ConnectionFactory** class. Creating Connection is a heavyweight operation. Connection is thread-safe. Therefore, multiple client threads can share one Connection.

In a typical scenario, a client program uses a Connection, and each thread obtains its own Admin or Table instance and invokes the operation interface provided by the Admin or Table object. You are not advised to cache or pool Table and Admin. The lifecycle of Connection is maintained by invokers that frees up resources by invoking **close()**.

## Example Code

The following code snippet exemplifies login, creating Connection, and creating a table. It belongs to the **HBaseSample** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
private TableName tableName = null;  
private Connection conn = null;  
  
public HBaseSample(Configuration conf) throws IOException {  
    this.tableName = TableName.valueOf("hbase_sample_table");  
    this.conn = ConnectionFactory.createConnection(conf);  
}
```



Avoid invoking login code repeatedly.

### 2.6.3.1.6 Creating a Table

## Function

Create a table using the **createTable** method of the **org.apache.hadoop.hbase.client.Admin** object and specify the table name and column family name. Tables can be created in two modes. (The mode of creating a table using preassigned regions is strongly recommended.)

- Quickly create a table. A newly created table contains only one region which will be split into multiple new regions as data increases.
- Create a table using preassigned regions. Preassign multiple regions before creating a table. This mode accelerates data write at the beginning of massive data write.



The column name and column family name of an HBase table consists of letters, digits, and underscores and cannot contain any special characters.

## Example Code

The following code snippet belongs to the **testCreateTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testCreateTable() {  
    LOG.info("Entering testCreateTable.");  
    // Specify the table descriptor.  
    TableDescriptorBuilder htd = TableDescriptorBuilder.newBuilder(tableName);(1)  
  
    // Set the column family name to info.  
    ColumnFamilyDescriptorBuilder hcd =  
    ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info"));(2)  
  
    // Set data encoding methods, HBase provides DIFF,FAST_DIFF,PREFIX  
  
    hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);  
    // Set compression methods, HBase provides two default compression  
    // methods:GZ and SNAPPY  
    // GZ has the highest compression rate, but low compression and  
    // decompression efficiency, fit for cold data  
    // SNAPPY has low compression rate, but high compression and  
    // decompression efficiency, fit for hot data.
```

```
// it is advised to use SNAANPPY
hcd.setCompressionType(Compression.Algorithm.SNAPPY); //Note [1]
htd.setColumnFamily(hcd.build()); (3)
Admin admin = null;
try {
    // Instantiate an Admin object.
    admin = conn.getAdmin(); (4)
    if (!admin.tableExists(tableName)) {
        LOG.info("Creating table...");
        admin.createTable(htd.build()); //Note [2] (5)
        LOG.info(admin.getClusterMetrics().toString());
        LOG.info(admin.listNamespaceDescriptors().toString());
        LOG.info("Table created successfully.");
    } else {
        LOG.warn("table already exists");
    }
} catch (IOException e) {
    LOG.error("Create table failed ", e);
} finally {
    if (admin != null) {
        try {
            // Close the Admin object.
            admin.close();
        } catch (IOException e) {
            LOG.error("Failed to close admin ", e);
        }
    }
}
LOG.info("Exiting testCreateTable.");
}
```

## Explanation

1. Create a table descriptor.
2. Create a column family descriptor.
3. Add the column family descriptor to the table descriptor.
4. Obtain an Admin object. The Admin object provides functions for creating a table, creating a column family, checking whether a table exists, modifying the table structure, modifying the column family structure, and deleting a table.
5. Invoke the table creation method of Admin.

## Precautions

- 1. The compression mode of a column family can be set. The code snippets are as follows:

```
// Set the encoding algorithm. HBase supports the DIFF, FAST_DIFF, PREFIX encoding algorithms.
hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);

// Set the file compression mode. HBase provides the GZ and SNAPPY compression algorithms by
default.
// GZ provides a high compression rate but low compression and decompression performance. GZ is
suitable for cold data.
// SNAPPY provides a low compression rate but high compression and decompression performance.
SNAPPY is suitable for hot data.
// It is recommended that SNAPPY be enabled by default.
hcd.setCompressionType(Compression.Algorithm.SNAPPY);
```

- 2. A table can be created by specifying the start and end RowKeys or preassigning regions using RowKey arrays. The code snippets are as follows:

```
// Create a table whose regions are preassigned.
byte[][] splits = new byte[4][];
splits[0] = Bytes.toBytes("A");
```

```
splits[1] = Bytes.toBytes("H");
splits[2] = Bytes.toBytes("O");
splits[3] = Bytes.toBytes("U");
admin.createTable(htd, splits);
```

### 2.6.3.1.7 Deleting a Table

#### Function

Delete a table using the **deleteTable** method of **org.apache.hadoop.hbase.client.Admin**.

#### Example Code

The following code snippet belongs to the **dropTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropTable() {
    LOG.info("Entering dropTable.");
    Admin admin = null;
    try {
        admin = conn.getAdmin();
        if (admin.tableExists(tableName)) {
            // Disable the table before deleting it.
            admin.disableTable(tableName);
            // Delete table.
            admin.deleteTable(tableName); // Note[1]
        }
        LOG.info("Drop table successfully.");
    } catch (IOException e) {
        LOG.error("Drop table failed ",e);
    } finally {
        if (admin != null) {
            try {
                // Close the Admin object.
                admin.close();
            } catch (IOException e) {
                LOG.error("Close admin failed ",e);
            }
        }
    }
    LOG.info("Exiting dropTable.");
}
```

#### Precautions

A table can be deleted only when the table is disabled. Therefore, **deleteTable** is used together with **disableTable**, **enableTable**, **tableExists**, **isTableEnabled**, and **isTableDisabled**.

### 2.6.3.1.8 Modifying a Table

#### Function

Modify table information using the **modifyTable** method of **org.apache.hadoop.hbase.client.Admin**.

#### Example Code

The following code snippet belongs to the **testModifyTable** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testModifyTable() {
    LOG.info("Entering testModifyTable.");

    // Specify the column family name.
    byte[] familyName = Bytes.toBytes("education");
    Admin admin = null;
    try {
        // Instantiate an Admin object.
        admin = conn.getAdmin();
        // Obtain the table descriptor.
        TableDescriptor htd = admin.getDescriptor(tableName);
        // Check whether the column family is specified before modification.
        if (!htd.hasFamily(familyName)) {
            // Create the column descriptor.

TableDescriptor tableBuilder = TableDescriptorBuilder.newBuilder(htd)
    .setColumnFamily(ColumnFamilyDescriptorBuilder.newBuilder(familyName).build()).build();

        // Disable the table to get the table offline before modifying
        // the table.
        admin.disableTable(tableName); // Note[1]
        // Submit a modifyTable request.
        admin.modifyTable(tableBuilder);
        // Enable the table to get the table online after modifying the
        // table.
        admin.enableTable(tableName);
    }
    LOG.info("Modify table successfully.");
} catch (IOException e) {
    LOG.error("Modify table failed " ,e);
} finally {
    if (admin != null) {
        try {
            // Close the Admin object.
            admin.close();
        } catch (IOException e) {
            LOG.error("Close admin failed " ,e);
        }
    }
}
LOG.info("Exiting testModifyTable.");
}
```

## Precautions

**modifyTable** takes effect only when a table is disabled.

### 2.6.3.1.9 Inserting Data

#### Function

HBase is a column-oriented database. A row of data may have multiple column families, and a column family may contain multiple columns. When writing data, you must specify the columns (including the column family names and column names) to which data is written. In HBase, data (a row of data or data sets) is inserted using the **put** method of HTable.

#### Example Code

The following code snippet belongs to the **testPut** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testPut() {
    LOG.info("Entering testPut.");
```

```
// Specify the column family name.  
byte[] familyName = Bytes.toBytes("info");  
// Specify the column name.  
byte[][] qualifiers = {  
    Bytes.toBytes("name"), Bytes.toBytes("gender"), Bytes.toBytes("age"), Bytes.toBytes("address")  
};  
  
Table table = null;  
try {  
    // Instantiate an HTable object.  
    table = conn.getTable(tableName);  
    List<Put> puts = new ArrayList<Put>();  
  
    // Instantiate a Put object.  
    Put put = putData(familyName, qualifiers,  
        Arrays.asList("012005000201", "Zhang San", "Male", "19", "Shenzhen, Guangdong"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000202", "Li Wanting", "Female", "23", "Shijiazhuang, Hebei"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000203", "Wang Ming", "Male", "26", "Ningbo, Zhejiang"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000204", "Li Gang", "Male", "18", "Xiangyang, Hubei"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000205", "Zhao Enru", "Female", "21", "Shangrao, Jiangxi"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000206", "Chen Long", "Male", "32", "Zhuzhou, Hunan"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000207", "Zhou Wei", "Female", "29", "Nanyang, Henan"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000208", "Yang Yiwen", "Female", "30", "Kaixian, Chongqing"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000209", "Xu Bing", "Male", "26", "Weinan, Shaanxi"));  
    puts.add(put);  
  
    put = putData(familyName, qualifiers,  
        Arrays.asList("012005000210", "Xiao Kai", "Male", "25", "Dalian, Liaoning"));  
    puts.add(put);  
  
    // Submit a put request.  
    table.put(puts);  
  
    LOG.info("Put successfully.");  
} catch (IOException e) {  
    LOG.error("Put failed ", e);  
} finally {  
    if (table != null) {  
        try {  
            // Close the HTable object.  
            table.close();  
        } catch (IOException e) {  
            LOG.error("Close table failed ", e);  
        }  
    }  
}
```

```
        }
    }
    LOG.info("Exiting testPut.");
}
```

## Precautions

Multiple threads are not allowed to use the same HTable instance at the same time. HTable is a non-thread safe class. If an HTable instance is used by multiple threads at the same time, concurrency problems will occur.

### 2.6.3.1.10 Deleting Data

#### Function

Delete data (a row of data or data sets) using the **delete** method of a Table instance.

#### Example Code

The following code snippet belongs to the **testDelete** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testDelete() {
    LOG.info("Entering testDelete.");
    byte[] rowKey = Bytes.toBytes("012005000201");

    Table table = null;
    try {
        // Instantiate an HTable object.
        table = conn.getTable(tableName);

        // Instantiate a Delete object.
        Delete delete = new Delete(rowKey);

        // Submit a delete request.
        table.delete(delete);

        LOG.info("Delete table successfully.");
    } catch (IOException e) {
        LOG.error("Delete table failed " ,e);
    } finally {
        if (table != null) {
            try {
                // Close the HTable object.
                table.close();
            } catch (IOException e) {
                LOG.error("Close table failed " ,e);
            }
        }
    }
    LOG.info("Exiting testDelete.");
}
```

#### NOTE

If secondary index is created in the family of the column where the deleted cell is, the index data is synchronously deleted.

### 2.6.3.1.11 Reading Data Using Get

#### Function

Before reading data from a table, instantiate the Table instance of the table, and then create a Get object. You can also set parameters for the Get object, such as the column family name and column name. Query results are stored in the Result object that stores multiple Cells.

#### Example Code

The following code snippet belongs to the **testGet** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testGet() {
    LOG.info("Entering testGet.");
    // Specify the column family name.
    byte[] familyName = Bytes.toBytes("info");
    // Specify the column name.
    byte[][] qualifier = { Bytes.toBytes("name"), Bytes.toBytes("address") };
    // Specify RowKey.
    byte[] rowKey = Bytes.toBytes("012005000201");
    Table table = null;
    try {
        // Create the Table instance.
        table = conn.getTable(tableName);
        // Instantiate a Get object.
        Get get = new Get(rowKey);
        // Set the column family name and column name.
        get.addColumn(familyName, qualifier[0]);
        get.addColumn(familyName, qualifier[1]);
        // Submit a get request.
        Result result = table.get(get);
        // Print query results.
        for (Cell cell : result.rawCells()) {
            LOG.info("{}:{}:{}",
                    Bytes.toString(CellUtil.cloneRow(cell)),
                    Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
                    Bytes.toString(CellUtil.cloneValue(cell)));
        }
        LOG.info("Get data successfully.");
    } catch (IOException e) {
        LOG.error("Get data failed ",e);
    } finally {
        if (table != null) {
            try {
                // Close the HTable object.
                table.close();
            } catch (IOException e) {
                LOG.error("Close table failed ",e);
            }
        }
    }
    LOG.info("Exiting testGet.");
}
```

### 2.6.3.1.12 Reading Data Using Scan

#### Function

Before reading data from a table, instantiate the Table instance of the table, create a Scan object, and set parameters for the Scan object based on search criteria. To improve query efficiency, you are advised to specify **StartRow** and

**StopRow.** Query results are stored in the ResultScanner object where each row of data is stored as a Result object that stores multiple Cells.

## Example Code

The following code snippet belongs to the **testScanData** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testScanData() {
    LOG.info("Entering testScanData.");
    Table table = null;
    // Instantiate a ResultScanner object.
    ResultScanner rScanner = null;
    try {
        // Create the Configuration instance.
        table = conn.getTable(tableName);
        // Instantiate a Get object.
        Scan scan = new Scan();
        scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
        // Set the cache size.
        scan.setCaching(1000);

        // Submit a scan request.
        rScanner = table.getScanner(scan);
        // Print query results.
        for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
            for (Cell cell : r.rawCells()) {
                LOG.info("{}:{}:{}",
                    Bytes.toString(CellUtil.cloneRow(cell)),
                    Bytes.toString(CellUtil.cloneFamily(cell)),
                    Bytes.toString(CellUtil.cloneQualifier(cell)),
                    Bytes.toString(CellUtil.cloneValue(cell)));
            }
        }
        LOG.info("Scan data successfully.");
    } catch (IOException e) {
        LOG.error("Scan data failed " ,e);
    } finally {
        if (rScanner != null) {
            // Close the scanner object.
            rScanner.close();
        }
        if (table != null) {
            try {
                // Close the HTable object.
                table.close();
            } catch (IOException e) {
                LOG.error("Close table failed " ,e);
            }
        }
    }
    LOG.info("Exiting testScanData.");
}
```

## Precautions

1. You are advised to specify **StartRow** and **StopRow** to ensure good performance with a specified Scan scope.
2. You can set **Batch** and **Caching**.
  - **Batch**  
Indicates the maximum number of records returned each time when the **next** interface is invoked using Scan. This parameter is related to the number of columns read each time.
  - **Caching**

Indicates the maximum number of **next** records returned for a remote procedure call (RPC) request. This parameter is related to the number of rows read by an RPC.

### 2.6.3.1.13 Filtering Data

#### Function

HBase Filter is used to filter data during Scan and Get. You can specify the filter criteria, such as filtering by RowKey, column name, or column value.

#### Example Code

The following code snippet belongs to the **testSingleColumnValueFilter** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testSingleColumnValueFilter() {
    LOG.info("Entering testSingleColumnValueFilter.");
    Table table = null;

    ResultScanner rScanner = null;

    try {
        table = conn.getTable(tableName);

        Scan scan = new Scan();
        scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
        // Set the filter criteria.
        SingleColumnValueFilter filter = new SingleColumnValueFilter(
            Bytes.toBytes("info"), Bytes.toBytes("name"), CompareOperator.EQUAL,
            Bytes.toBytes("Xu Bing"));
        scan.setFilter(filter);
        // Submit a scan request.
        rScanner = table.getScanner(scan);
        // Print query results.
        for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
            for (Cell cell : r.rawCells()) {
                LOG.info("{}:{}:{},{}", Bytes.toString(CellUtil.cloneRow(cell)),
                    Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
                    Bytes.toString(CellUtil.cloneValue(cell)));
            }
        }
        LOG.info("Single column value filter successfully.");
    } catch (IOException e) {
        LOG.error("Single column value filter failed ",e);
    } finally {
        if (rScanner != null) {
            // Close the scanner object.
            rScanner.close();
        }
        if (table != null) {
            try {
                // Close the HTable object.
                table.close();
            } catch (IOException e) {
                LOG.error("Close table failed ",e);
            }
        }
    }
    LOG.info("Exiting testSingleColumnValueFilter.");
}
```

## Precautions

Currently, secondary indexes do not support the comparators that use objects of the SubstringComparator class as filters.

For example, the following sample code is not supported:

```
Scan scan = new Scan();
filterList = new FilterList(FilterList.Operator.MUST_PASS_ALL);
filterList.addFilter(new SingleColumnValueFilter(Bytes
.toBytes(columnFamily), Bytes.toBytes(qualifier),
CompareOperator.EQUAL, new SubstringComparator(substring)));
scan.setFilter(filterList);
```

### 2.6.3.1.14 Creating a Secondary Index

#### Function

You can manage HBase secondary indexes using methods provided in **org.apache.hadoop.hbase.hindex.client.HIndexAdmin**. This class provides methods of creating an index.



#### NOTE

Secondary indexes cannot be modified. If you need to modify them, delete old indexes and create new ones.

#### Example Code

The following code snippet belongs to the **createIndex** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void createIndex() {
    LOG.info("Entering createIndex.");

    String indexName = "index_name";
    // Create hindex instance
    TableIndices tableIndices = new TableIndices();
    IndexSpecification iSpec = new IndexSpecification(indexName);
    iSpec.addIndexColumn(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("info")).build(),
        "name", ValueType.STRING);// Note[1]
    tableIndices.addIndex(iSpec);

    HIndexAdmin iAdmin = null;
    Admin admin = null;
    try {

        admin = conn.getAdmin();
        iAdmin = HIndexClient.newHIndexAdmin(admin);

        // add index to the table
        iAdmin.addIndices(tableName, tableIndices);

        LOG.info("Create index successfully.");
    } catch (IOException e) {
        LOG.error("Create index failed ",e);
    } finally {
        if (admin != null) {
            try {
                admin.close();
            } catch (IOException e) {
                LOG.error("Close admin failed ",e);
            }
        }
    }
}
```

```
if (iAdmin != null) {
    try {
        // Close IndexAdmin Object
        iAdmin.close();
    } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
    }
}
LOG.info("Exiting createIndex.");
```

By default, newly created level-2 indexes are disabled. To enable a specified level-2 index, see the following code snippet. The following code snippet belongs to the `enableIndex` method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
public void enableIndex() {
    LOG.info("Entering createIndex.");

    // Name of the index to be enabled
    String indexName = "index_name";

    List<String> indexNameList = new ArrayList<String>();
    indexNameList.add(indexName);

    HIndexAdmin iAdmin = null;
    Admin admin = null;
    try {
        admin = conn.getAdmin();
        iAdmin = HIndexClient.newHIndexAdmin(admin);

        // Alternately, enable the specified indices
        iAdmin.enableIndices(tableName, indexNameList);
        LOG.info("Successfully enable indices {} of the table {}", indexNameList, tableName);
    } catch (IOException e) {
        LOG.error("Failed to enable indices {} of the table {} . {}", indexNameList, tableName, e);
    } finally {
        if (admin != null) {
            try {
                admin.close();
            } catch (IOException e) {
                LOG.error("Close admin failed " , e);
            }
        }
        if (iAdmin != null) {
            try {
                iAdmin.close();
            } catch (IOException e) {
                LOG.error("Close admin failed " , e);
            }
        }
    }
}
```

## Precautions

Create a combination index.

HBase supports creation of secondary indexes on multiple fields, for example, the name and age columns.

```
HIndexSpecification iSpecUnite = new HIndexSpecification(indexName);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "name", ValueType.String, 10);
iSpecUnite.addIndexColumn(new HColumnDescriptor("info"), "age", ValueType.String, 3);
```

## Related Operations

Create an index table by running a command.

You can also use the TableIndexer tool to create an index in an existing user table.



The <table\_name> user table must exist.

```
hbase org.apache.hadoop.hbase.hindex.mapreduce.TableIndexer -Dtablename.to.index=<table_name> -  
Dindexspecs.to.add='IDX1=>cf1:[q1->datatype];cf2:[q2->datatype],[q3->datatype]#IDX2=>cf1:[q5->datatype]' -Dindexnames.to.build='IDX1'
```

"#" is used to separate indexes. ";" is used to separate column families. "," is used to separate columns.

**tablename.to.index**: indicates the name of the table where the index is created.

**indexspecs.to.add**: indicates the user table columns corresponding to the index.

The parameters in the command are described as follows:

- **IDX1**: indicates the index name.
- **cf1**: indicates the column family name.
- **q1**: indicates the column name.
- **datatype**: indicates the data type. Only the Integer, String, Double, Float, Long, Short, Byte and Char formats are supported.

### 2.6.3.1.15 Deleting an Index

## Function

You can manage HBase secondary indexes using methods provided in **org.apache.hadoop.hbase.hindex.client.HIndexAdmin**. This class provides methods of querying and deleting an index.

## Example Code

The following code snippet belongs to the **dropIndex** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void dropIndex() {  
    LOG.info("Entering dropIndex.");  
    String indexName = "index_name";  
    List<String> indexNameList = new ArrayList<String>();  
    indexNameList.add(indexName);  
  
    IndexAdmin iAdmin = null;  
    try {  
        // Instantiate HIndexAdmin Object  
        iAdmin = HIndexClient.newHIndexAdmin(conn.getAdmin());  
        // Delete Secondary Index  
        iAdmin.dropIndex(tableName, indexNameList);  
  
        LOG.info("Drop index successfully.");  
    } catch (IOException e) {  
        LOG.error("Drop index failed.");  
    } finally {  
        if (iAdmin != null) {
```

```
try {
    // Close Secondary Index
    iAdmin.close();
} catch (IOException e) {
    LOG.error("Close admin failed.");
}
}
LOG.info("Exiting dropIndex.");
}
```

## Precautions

- Use the dropIndex and dropIndexes interfaces to delete HBase level-2 indexes. Do not directly drop the index table, because it is an invalid operation and cannot update table information, leading to query failures at the time of index rebuilding.
- If a user table is deleted, the corresponding index table is also deleted.

### 2.6.3.1.16 Secondary Index-based Query

#### Function

In user tables with secondary indexes, you can use Filter to query data. The data query performance is higher than that in user tables without secondary indexes.



#### NOTE

- For details about the secondary index principles, see "HIndex" in the "Product Introduction > Components > HBase > HBase Enhanced Open Source Features".
- HIndex supports three Filter types: SingleColumnValueFilter, SingleColumnValueExcludeFilter, and SingleColumnValuePartitionFilter.
- HIndex supports the following Comparator types: BinaryComparator, BitComparator, LongComparator, DecimalComparator, DoubleComparator, FloatComparator, IntComparator, and NullComparator.

The secondary index usage rules are as follows:

- For scenarios in which a single index is created for one or multiple columns:
  - When you use this column for AND or OR query filtering, the index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND/OR Filter\_Condition(IndexCol2).
  - When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.  
For example, Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND Filter\_Condition(NonIndexCol1).
  - When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.  
For example, Filter\_Condition(IndexCol1) AND/OR Filter\_Condition(IndexCol2) OR Filter\_Condition(NonIndexCol1).
- For scenarios in which a combination index is created for multiple columns:
  - When the columns used for query are all or part of the columns of the combination index and are in the same sequence with the combination index, the index is used to improve the query performance.

For example, a combination index is created for C1, C2, and C3. The index takes effect in the following scenarios:

Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND  
Filter\_Condition(IndexCol3)

Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2)

Filter\_Condition(IndexCol1)

The index does not take effect in the following scenarios:

Filter\_Condition(IndexCol2) AND Filter\_Condition(IndexCol3)

Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol3)

Filter\_Condition(IndexCol2)

Filter\_Condition(IndexCol3)

- When you use "Index Column AND Non-Index Column" for filtering in the query, this index is used to improve the query performance.

For example:

Filter\_Condition(IndexCol1) AND Filter\_Condition(NonIndexCol1)

Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2) AND  
Filter\_Condition(NonIndexCol1)

- When you use "Index Column OR Non-Index Column" for filtering in the query, the index is not used and the query performance is not improved.

For example:

Filter\_Condition(IndexCol1) OR Filter\_Condition(NonIndexCol1)

(Filter\_Condition(IndexCol1) AND Filter\_Condition(IndexCol2))OR  
( Filter\_Condition(NonIndexCol1))

- When multiple columns are used for query, a value range can be specified only for the last column in the combination index and the other columns can only be set to a specified value.

For example, a combination index is created for C1, C2, and C3. In range query, a value range can be set only for C3 and the filter criterion is "C1 = XXX, C2 = XXX, and C3 = value range".

- For scenarios in which secondary index is created in a user table, you can use Filter to query data. The query results of the single and combination index with filter are the same as those in the table without secondary index. The data query performance is higher than that in user tables without secondary indexes.

## Example Code

The following code snippet belongs to the **testScanDataByIndex** method in the **HBaseSample** class of the **com.huawei.hadoop.hbase.example** package.

### Example: Query data using secondary indexes.

```
public void testScanDataByIndex() {  
    LOG.info("Entering testScanDataByIndex.");  
    Table table = null;  
    ResultScanner scanner = null;  
    try {  
        table = conn.getTable(tableName);  
  
        // Create a filter for indexed column.  
    } catch (IOException e) {  
        LOG.error("Error occurred while creating filter for indexed column.", e);  
    } finally {  
        if (scanner != null) {  
            scanner.close();  
        }  
        if (table != null) {  
            try {  
                table.close();  
            } catch (IOException e) {  
                LOG.error("Error occurred while closing table.", e);  
            }  
        }  
    }  
}
```

```
Filter filter = new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("name"),
    CompareOperator.EQUAL, "Li Gang".getBytes());
Scan scan = new Scan();
scan.setFilter(filter);
scanner = table.getScanner(scan);
LOG.info("Scan indexed data.");

for (Result result : scanner) {
    for (Cell cell : result.rawCells()) {
        LOG.info("{{:},{},{}]", Bytes.toString(CellUtil.cloneRow(cell)),
            Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
            Bytes.toString(CellUtil.cloneValue(cell)));
    }
}
LOG.info("Scan data by index successfully.");
} catch (IOException e) {
    LOG.error("Scan data by index failed.");
} finally {
    if (scanner != null) {
        // Close the scanner object.
        scanner.close();
    }
    try {
        if (table != null) {
            table.close();
        }
    } catch (IOException e) {
        LOG.error("Close table failed.");
    }
}

LOG.info("Exiting testScanDataByIndex.");
}
```

## Precaution

Create secondary indexes for the **name** field first.

## Related Operations

Query a table using a secondary index.

The following provides an example:

Add an index to the **name** column of the **info** column family in **hbase\_sample\_table**. Run the following command on the client:

```
hbase org.apache.hadoop.hbase.hindex.mapreduce.TableIndexer -Dtablename.to.index=hbase_sample_table -
Dindexspecs.to.add='IDX1=>info:[name->String]' -Dindexnames.to.build='IDX1'
```

Query **info:name**. Run the following command on the HBase shell client:

```
>scan 'hbase_sample_table',
{FILTER=>"SingleColumnValueFilter(family,qualifier,compareOp,comparator,
!IterIfMissing,latestVersionOnly)"}
```

### NOTE

Use APIs to perform complex query on the HBase shell client.

The parameters are described as follows:

- **family**: indicates the column family where the column to be queried locates, such as **info**.
- **qualifier**: indicates the column to be queried, such as **name**.
- **compareOp**: indicates the comparison operator, such as = and >.
- **comparator**: indicates the target value to be queried, such as **binary:Zhang San**.
- **filterIfMissing**: indicates whether a row is filtered if the column does not exist in this row. The default value is **false**.
- **latestVersionOnly**: indicates whether only values of the latest version are to be queried. The default value is **false**.

For example:

```
>scan 'hbase_sample_table',{FILTER=>"SingleColumnValueFilter('info','name',=,'binary:Zhang San',true,true)"}
```

#### 2.6.3.1.17 Multi-Point Region Division

### Function

You can perform multi-point division by using **org.apache.hadoop.hbase.client.HBaseAdmin**. Note that the division operations take effect on empty regions only.

In this example, the multi-point division is performed on an HBase table by using **multiSplit**. The table will be split into 5 parts: "-∞~A", "A~D", "D~F", "F~H", and "H~+∞".

### Example Code

The following code snippet belongs to the **testMultiSplit** method in the **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
public void testMultiSplit() {  
    LOG.info("Entering testMultiSplit.");  
  
    Table table = null;  
    Admin admin = null;  
    try {  
        admin = conn.getAdmin();  
  
        // initialize a HTable object  
        table = conn.getTable(tableName);  
        Set<HRegionInfo> regionSet = new HashSet<HRegionInfo>();  
        List<HRegionLocation> regionList = conn.getRegionLocator(tableName).getAllRegionLocations();  
        for(HRegionLocation hrl : regionList){  
            regionSet.add(hrl.getRegionInfo());  
        }  
        byte[][] sk = new byte[4][];  
        sk[0] = "A".getBytes();  
        sk[1] = "D".getBytes();  
        sk[2] = "F".getBytes();  
        sk[3] = "H".getBytes();  
        for (RegionInfo regionInfo : regionSet) {  
            admin.multiSplitSync(regionInfo.getRegionName(), sk);  
        }  
        LOG.info("MultiSplit successfully.");  
    } catch (Exception e) {
```

```
LOG.error("MultiSplit failed.");
} finally {
    if (table != null) {
        try {
            // Close table object
            table.close();
        } catch (IOException e) {
            LOG.error("Close table failed " ,e);
        }
    }
    if (admin != null) {
        try {
            // Close the Admin object.
            admin.close();
        } catch (IOException e) {
            LOG.error("Close admin failed " ,e);
        }
    }
}
LOG.info("Exiting testMultiSplit.");
```

Note that the division operations take effect on empty regions only.

### 2.6.3.1.18 Creating a Phoenix Table

#### Function

Phoenix can be installed on HBase to enable it to support SQL and JDBC APIs. This way, SQL users can access the HBase cluster.

#### Example Code

The following code snippet belongs to the **testCreateTable** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
/*
 * Create Table
 */
public void testCreateTable() {
    LOG.info("Entering testCreateTable.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Create table
    String createTableSQL =
        "CREATE TABLE IF NOT EXISTS TEST (id integer not null primary key, name varchar, "
        + "account char(6), birth date)";
    try (Connection conn = DriverManager.getConnection(url, props)) {
        Statement stat = conn.createStatement();
        // Execute Create SQL
        stat.executeUpdate(createTableSQL);
        LOG.info("Create table successfully.");
    } catch (Exception e) {
        LOG.error("Create table failed.", e);
    }
    LOG.info("Exiting testCreateTable.");
}
/*
 * Drop Table
 */
public void testDrop() {
    LOG.info("Entering testDrop.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Delete table
    String dropTableSQL = "DROP TABLE TEST";

    try (Connection conn = DriverManager.getConnection(url, props));
```

```
Statement stat = conn.createStatement();
stat.executeUpdate(dropTableSQL);
LOG.info("Drop successfully.");
} catch (Exception e) {
    LOG.error("Drop failed.", e);
}
LOG.info("Exiting testDrop.");
```

### 2.6.3.1.19 Writing Data to the PhoenixTable

#### Function

The Phoenix table enables data writing in HBase.

#### Example Code

The following code snippet belongs to the **testPut** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
/*
 * Put data
 */
public void testPut() {
    LOG.info("Entering testPut.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Insert
    String upsertSQL =
        "UPSERT INTO TEST VALUES(1,'John','100000', TO_DATE('1980-01-01','yyyy-MM-dd'))";
    try (Connection conn = DriverManager.getConnection(url, props);
        Statement stat = conn.createStatement()){
        // Execute Update SQL
        stat.executeUpdate(upsertSQL);
        conn.commit();
        LOG.info("Put successfully.");
    } catch (Exception e) {
        LOG.error("Put failed.", e);
    }
    LOG.info("Exiting testPut.");
}
```

### 2.6.3.1.20 Reading the PhoenixTable

#### Function

The Phoenix table enables data reading.

#### Example Code

The following code snippet belongs to the **testSelect** method in the **PhoenixSample** class of the **com.huawei.bigdata.hbase.examples** package.

```
/*
 * Select Data
 */
public void testSelect() {
    LOG.info("Entering testSelect.");
    String URL = "jdbc:phoenix:" + conf.get("hbase.zookeeper.quorum");
    // Query
    String querySQL = "SELECT * FROM TEST WHERE id = ?";
    Connection conn = null;
    PreparedStatement preStat = null;
    Statement stat = null;
```

```
ResultSet result = null;
try {
    // Create Connection
    conn = DriverManager.getConnection(url, props);
    // Create Statement
    stat = conn.createStatement();
    // Create PrepareStatement
    preStat = conn.prepareStatement(querySQL);
    // Execute query
    preStat.setInt(1, 1);
    result = preStat.executeQuery();
    // Get result
    while (result.next()) {
        int id = result.getInt("id");
        String name = result.getString(1);
        System.out.println("id: " + id);
        System.out.println("name: " + name);
    }
    LOG.info("Select successfully.");
} catch (Exception e) {
    LOG.error("Select failed.", e);
} finally {
    if (null != result) {
        try {
            result.close();
        } catch (Exception e2) {
            LOG.error("Result close failed.", e2);
        }
    }
    if (null != stat) {
        try {
            stat.close();
        } catch (Exception e2) {
            LOG.error("Stat close failed.", e2);
        }
    }
    if (null != conn) {
        try {
            conn.close();
        } catch (Exception e2) {
            LOG.error("Connection close failed.", e2);
        }
    }
}
LOG.info("Exiting testSelect.");
```

### 2.6.3.1.21 Using HBase Dual-Read Capacity

#### Scenario

The HBase client application loads the configuration items of the active and standby clusters by customization to implement the dual-read capability. HBase dual-read is a key feature that improves the high availability of the HBase cluster system. It applies to four query scenarios: reading data using **Get**, reading data in batches using **Get**, reading data using **Scan**, and querying data using a secondary index. HBase can read data from the active and standby clusters at the same time, reducing the query glitch time. The advantages are as follows:

- **High success rate:** The concurrent dual-read mechanism ensures a high success rate of read requests.
- **High availability:** When a single cluster is faulty, the query service is not interrupted. A short network jitter does not prolong the query time.

- High generality: The dual-read feature does not support dual-write, but does not affect the original real-time write scenario.
- Ease-of-use: Client encapsulation is performed, which is not sensed by services.

 NOTE

Restrictions on HBase dual-read:

- The HBase dual-read feature is implemented based on replication. Data read from the standby cluster may be different from that from the active cluster. Therefore, only eventual consistency can be achieved.
- Currently, the HBase dual-read feature is used only for query. When the active cluster breaks down, the latest data cannot be synchronized. As a result, the latest data cannot be queried in the standby cluster.
- A **Scan** operation of HBase may be split into multiple RPC operations. Data may not be completely the same because related session information is not synchronized between different clusters. Therefore, the dual-read feature takes effect only when an RPC operation is performed for the first time. Requests before ResultScanner close access the cluster used for the first RPC operation.
- The HBase Admin API and real-time write API access only the active cluster. Therefore, after the active cluster breaks down, the Admin API and real-time write API are unavailable, and only the **Get** and **Scan** query services are available.

HBase dual-read supports the following two methods to configure the active/standby cluster:

- Add active/standby cluster configurations to the **hbase-dual.xml** file.
- Add the active/standby cluster configuration to **HBaseMultiClusterConnection**.

## Add the Active/Standby Cluster Configuration to the **hbase-dual.xml** File

**Step 1** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the HBase active cluster and save them to the **src/main/resources/conf/active** directory. This directory needs to be created by yourself. For details, see [Preparing the Connection Cluster Configuration File](#).

**Step 2** Obtain the client configuration files **core-site.xml**, **hbase-site.xml**, and **hdfs-site.xml** of the standby cluster and save them to the **src/main/resources/conf/standby** directory. This directory needs to be created by yourself. For details, see [Preparing the Connection Cluster Configuration File](#).

**Step 3** Create the **hbase-dual.xml** configuration file and save it to the **src/main/resources/conf/** directory. For details about the configuration items in the configuration file, see [Table 2-70](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!--Configuration file directory of the active cluster-->
    <property>
        <name>hbase.dualclient.active.cluster.configuration.path</name>
        <value>{Sample code directory}\src\main\resources\conf\active</value>
    </property>
    <!--Configuration file directory of the standby cluster-->
    <property>
        <name>hbase.dualclient.standby.cluster.configuration.path</name>
        <value>{Sample code directory}\src\main\resources\conf\standby</value>
    </property>
    <!--Connection implementation of the dual-read mode-->
```

```
<property>
    <name>hbase.client.connection.impl</name>
    <value>org.apache.hadoop.hbase.client.HBaseMultiClusterConnectionImpl</value>
</property>
<!--Normal mode-->
<property>
    <name>hbase.security.authentication</name>
    <value>Simple</value>
</property>
<!--Normal mode-->
<property>
    <name>hadoop.security.authentication</name>
    <value>Simple</value>
</property>
</configuration>
```

**Step 4** Create a dual-read Configuration and delete the comment of **testHBaseDualReadSample** from the **main** method of the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package. Ensure that the value of **IS\_CREATE\_CONNECTION\_BY\_XML** in the **HBaseDualReadSample** class in the **com.huawei.bigdata.hbase.examples** package is **true**.

**Step 5** Determining the data source cluster

- GET request. The following code snippet belongs to the **testGet** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
Result result = table.get(get);
if (result instanceof DualResult) {
    LOG.info(((DualResult)result).getClusterId());
}
```

- Scan request. The following code snippet belongs to the **testScanData** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
ResultScanner rScanner = table.getScanner(scan);
if (rScanner instanceof HBaseMultiScanner) {
    LOG.info(((HBaseMultiScanner)rScanner).getClusterId());
}
```

**Step 6** The client can print metric information.

Add the following content to the **log4j.properties** file so that the client can export metric information to the specified file: For details about the metrics, see [HBase Dual-Read Configuration Items](#).

```
log4j.logger.DUAL=debug,DUAL
log4j.appenders.DUAL=org.apache.log4j.RollingFileAppender
log4j.appenders.DUAL.File=/var/log/dual.log //Local dual-read log path on the client. Change the value to the actual directory, but ensure that the directory has the write permission.
log4j.additivity.DUAL=false
log4j.appenders.DUAL.MaxFileSize=${hbase.log.maxfilesize}
log4j.appenders.DUAL.MaxBackupIndex=${hbase.log.maxbackupindex}
log4j.appenders.DUAL.layout=org.apache.log4j.PatternLayout
log4j.appenders.DUAL.layout.ConversionPattern=%d{ISO8601} %-5p [%t] %c{2}: %m%n
```

----End

## Configure the Active/Standy Cluster Configuration in HBaseMultiClusterConnection

**Step 1** Create a dual-read Configuration and delete the comment of **testHBaseDualReadSample** from the **main** method of the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package. Ensure that the value of

**IS\_CREATE\_CONNECTION\_BY\_XML** in the **HBaseDualReadSample** class in the **com.huawei.bigdata.hbase.examples** package is **false**.

- Step 2** Add related configurations to the **addHbaseDualXmlParam** method of the **HBaseDualReadSample** class. For details about related configuration items, see [HBase Dual-Read Configuration Items](#).

```
private void addHbaseDualXmlParam(Configuration conf) {  
    // We need to set the optional parameters contained in hbase-dual.xml to conf  
    // when we use configuration transfer solution  
    conf.set(CONNECTION_IMPL_KEY, DUAL_READ_CONNECTION);  
    // conf.set("", "");  
}
```

- Step 3** Add configurations related to the Active cluster client to the **initActiveConf** method of the **HBaseDualReadSample** class.

```
private void initActiveConf() {  
    // The hbase-dual.xml configuration scheme is used to generate the client configuration of the active  
    // cluster.  
    // In actual application development, you need to generate the client configuration of the active cluster.  
    String activeDir =  
        HBaseDualReadSample.class.getClassLoader().getResource(Utils.CONF_DIRECTORY).getPath()  
            + File.separator + ACTIVE_DIRECTORY + File.separator;  
    Configuration activeConf = Utils.createConfByUserDir(activeDir);  
    HBaseMultiClusterConnection.setActiveConf(activeConf);  
}
```

- Step 4** Add configurations related to the Standby cluster client to the **initStandbyConf** method of the **HBaseDualReadSample** class.

```
private void initStandbyConf() {  
    // The hbase-dual.xml configuration scheme is used to generate the client configuration of the standby  
    // cluster.  
    // In actual application development, you need to generate the client configuration of the standby cluster.  
    String standbyDir =  
        HBaseDualReadSample.class.getClassLoader().getResource(Utils.CONF_DIRECTORY).getPath()  
            + File.separator + STANDBY_DIRECTORY + File.separator;  
    Configuration standbyConf = Utils.createConfByUserDir(standbyDir);  
    HBaseMultiClusterConnection.setStandbyConf(standbyConf);  
}
```

- Step 5** Determining the data source cluster.

- GET request. The following code snippet belongs to the **testGet** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
Result result = table.get(get);  
if (result instanceof DualResult) {  
    LOG.info(((DualResult)result).getClusterId());  
}
```

- Scan request. The following code snippet belongs to the **testScanData** method in **HBaseSample** class of the **com.huawei.bigdata.hbase.examples** packet.

```
ResultScanner rScanner = table.getScanner(scan);  
if (rScanner instanceof HBaseMultiScanner) {  
    LOG.info(((HBaseMultiScanner)rScanner).getClusterId());  
}
```

- Step 6** The client can print metric information.

Add the following content to the **log4j.properties** file so that the client can export metric information to the specified file: For details about the metrics, see [HBase Dual-Read Configuration Items](#)

```
log4j.logger.DUAL=debug,DUAL  
log4j.appenders.DUAL=org.apache.log4j.RollingFileAppender  
log4j.appenders.DUAL.File=/var/log/dual.log //Local dual-read log path on the client. Change the value to
```

```
the actual directory, but ensure that the directory has the write permission.  
log4j.additivity.DUAL=false  
log4j.appendер.DUAL.MaxFileSize=${hbase.log.maxfilesize}  
log4j.appendер.DUAL.MaxBackupIndex=${hbase.log.maxbackupindex}  
log4j.appendер.DUAL.layout=org.apache.log4j.PatternLayout  
log4j.appendер.DUAL.layout.ConversionPattern=%d{ISO8601} %-5p [%t] %c{2}: %m%n
```

----End

## 2.6.3.2 HBase Global Secondary Index Sample Program

### 2.6.3.2.1 Service Scenario Description

HBase allows you to use global secondary indexes to accelerate conditional queries. This sample shows you how to manage and use global secondary indexes.

#### Scenario

Assume that you are developing an application that records user information and addresses. The following table lists the data to be recorded.

**Table 2-68** User information

<b>id</b>	<b>name</b>	<b>age</b>	<b>address</b>
1	Zhang	20	CityA
2	Li	30	CityB
3	Wang	35	CityC

#### Data Preparation

Proper design of a table structure, RowKeys, and column names enable you to make full use of HBase advantages. If you use a global secondary index in a scan condition query, the query is performed on an index table. You do not need to pay attention to the rowkeys of the user table. In this example, the rowkeys are in "r1", "r2", "r3"... format. All columns are stored in the **info** column family.

#### Sample Description

The examples in following content describe how to create and delete global secondary indexes, modify statuses, , and use them to accelerate queries.

### 2.6.3.2.2 Creating HBase Global Secondary Indexes

#### Function

Manage HBase global secondary indexes by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin**. **addIndices** is used for creating global secondary indexes.

To create a global secondary index, you need to configure the index column, covering column (optional), and pre-created index table regions (recommended).

### NOTE

To create global secondary indexes on a table with existing data, you need to pre-create regions for the index table to prevent hot-spotting. The rowkeys of the index table data consists of index columns and separators. The format is `\x01/index value\x00`. Specify the format when you pre-create a region, for example, when the **id** and **age** columns are used as the index columns, both columns need to be integers. Use the **id** column to pre-create regions. You can specify the pre-created index table region as follows:

`\x010,\x011,\x012....`

## Code Sample

The following code snippets are in the **addIndices** method in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, an index named **index\_id\_age** is created for the **user\_table** data table. The **id** and **age** columns in the data are used as index columns and the **name** column is overwritten. (The query condition is not used, but the query result needs to return to this column).

```
/**  
 * createIndex  
 */  
public void testCreateIndex() {  
    LOG.info("Entering createIndex.");  
    // Create index instance  
    TableIndices tableIndices = new TableIndices();  
    // Create index spec  
    // idx_id_age covered info:name  
    HIndexSpecification indexSpec = new HIndexSpecification("idx_id_age");  
  
    // Set index column  
    indexSpec.addIndexColumn(Bytes.toBytes("info"), Bytes.toBytes("id"), ValueType.STRING);  
    indexSpec.addIndexColumn(Bytes.toBytes("info"), Bytes.toBytes("age"), ValueType.STRING);  
  
    // Set covered column  
    // If you want cover one column, use addCoveredColumn  
    // If you want cover all column in one column family, use addCoveredFamilies  
    // If you want cover all column of all column family, use setCoveredAllColumns  
    indexSpec.addCoveredColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));  
  
    // Need specify index table split keys, it should specify by index column.  
    // Note: index data's row key has a same prefix "\x01"  
    // For example:  
    // Our index column include "id" and "age", "id" is a number, we  
    // could specify split key like \x010 \x011 \x012...  
    byte[][] splitKeys = new byte[10][];  
    for (int i = 0; i < 10; i++) {  
        splitKeys[i] = Bytes.toBytesBinary("\x01" + i);  
    }  
    indexSpec.setSplitKeys(splitKeys);  
    tableIndices.addIndex(indexSpec);  
  
    // iAdmin will close the inner admin instance  
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient newIndexAdmin(conn.getAdmin())) {  
        // add index to the table  
        iAdmin.addIndices(tableName, tableIndices);  
        LOG.info("Create index successfully.");  
    } catch (IOException e) {  
        LOG.error("Create index failed.", e);  
    }  
    LOG.info("Exiting createIndex.");  
}
```

### 2.6.3.2.3 Querying Global Secondary Indexes

#### Function

Manage HBase global secondary index by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin.listIndices** is used for querying indexes, including the definitions and status of all indexes related to the current user table.

#### Code Sample

The following code snippets are in the **listIndices** method in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, all indexes corresponding to the user table **user\_table** are queried.

```
/*
 * List indexes
 */
public void testListIndexes() {
    LOG.info("Entering testListIndexes.");
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient.newIndexAdmin(conn.getAdmin())) {
        for (Pair<HIndexSpecification, IndexState> indexPair : iAdmin.listIndices(tableName)) {
            LOG.info("index spec:{} , index state:{}" , indexPair.getFirst(), indexPair.getSecond());
        }
        LOG.info("List indexes successfully.");
    } catch (IOException e) {
        LOG.error("List indexes failed." , e);
    }
    LOG.info("Exiting testListIndexes.");
}
```

### 2.6.3.2.4 Querying Based on Global Secondary Indexes

#### Function

When a user table with a global secondary index is used for query, the query can be converted to a range query of the index table. The query performance is higher than that of a user table without a secondary index. For details about global secondary indexes, see

"Introduction to Global Secondary Indexes" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).

#### Code Sample

The following code snippets are in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, the values of **id**, **age**, and **name** of the specified **id** are queried, and the **idx\_id\_age** index is selected. The query results are completely overwritten. As a result, you do not need to query in the original table to achieve optimal query performance.

```
/*
 * Scan data by secondary index.
```

```

/*
public void testScanDataByIndex() {
    LOG.info("Entering testScanDataByIndex.");

    Scan scan = new Scan();
    // Create a filter for indexed column.
    SingleColumnValueFilter filter = new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes.toBytes("id"),
        CompareOperator.EQUAL, Bytes.toBytes("3"));
    filter.setFilterIfMissing(true);
    scan.setFilter(filter);

    // Specify returned columns
    // If returned columns not included in index table, will query back user table,
    // it's not the fast way to get data, suggest to cover all needed columns.
    // If you want to confirm whether using index for scanning, please set hbase client log level to DEBUG.
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("id"));
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("age"));
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));

    LOG.info("Scan indexed data.");
    try (Table table = conn.getTable(tableName); ResultScanner scanner = table.getScanner(scan)) {
        for (Result result : scanner) {
            for (Cell cell : result.rawCells()) {
                LOG.info("{}:{}:{},{}:{}:{}",
                    Bytes.toString(CellUtil.cloneRow(cell)),
                    Bytes.toString(CellUtil.cloneFamily(cell)), Bytes.toString(CellUtil.cloneQualifier(cell)),
                    Bytes.toString(CellUtil.cloneValue(cell)));
            }
        }
        LOG.info("Scan data by index successfully.");
    } catch (IOException e) {
        LOG.error("Scan data by index failed ", e);
    }
    LOG.info("Exiting testScanDataByIndex.");
}

```

### 2.6.3.2.5 Disabling Global Secondary Indexes

#### Function

The status of a global secondary index determines whether the index is valid. By modifying the index status, you can disable, enable, or discard an index (no index data will be generated). You can modify the index status by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin**. For details, see "Introduction to Global Secondary Index APIs" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).

#### Code Sample

The following code snippets are in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, the **idx\_id\_age** index is disabled. That is, the index is not used during query, but index data is generated.

```

/**
 * alter index to UNUSABLE state.
 */
public void testAlterIndex() {
    LOG.info("Entering testAlterIndex.");
    List<String> indexNameList = Lists.newArrayList("idx_id_age");
    // Instantiate HIndexAdmin Object
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient newIndexAdmin(conn.getAdmin())) {
        iAdmin.alterGlobalIndicesUnusable(tableName, indexNameList);
    }
}

```

```
        LOG.info("Alter indices to UNUSABLE successfully.");
    } catch (IOException e) {
        LOG.error("Alter indices to UNUSABLE failed.", e);
    }
    LOG.info("Exiting testAlterIndex.");
}
```

### 2.6.3.2.6 Deleting Global Secondary Indexes

#### Function

Manage HBase global secondary indexes by calling the method in **org.apache.hadoop.hbase.hindex.global.GlobalIndexAdmin**.

**dropIndices** is used to delete indexes.

#### Code Sample

The following code snippets are in the **dropIndices** method in the **GlobalSecondaryIndexSample** class of the **com.huawei.bigdata.hbase.examples** package.

In this case, the **idx\_id\_age** index is deleted from the **user\_table** table.

```
/*
 * dropIndex
 */
public void testDropIndex() {
    LOG.info("Entering testDropIndex.");
    List<String> indexNameList = Lists.newArrayList("idx_id_age");
    // Instantiate HIndexAdmin Object
    try (GlobalIndexAdmin iAdmin = GlobalIndexClient newIndexAdmin(conn.getAdmin())) {
        // Delete Secondary Index
        iAdmin.dropIndices(tableName, indexNameList);
        LOG.info("Drop index successfully.");
    } catch (IOException e) {
        LOG.error("Drop index failed ", e);
    }
    LOG.info("Exiting testDropIndex.");
}
```

### 2.6.3.3 Sample Program for Preloading Meta Tables When Request Concurrency Is High

#### 2.6.3.3.1 Service Scenario Description

When there is a large number of concurrent access requests to a data table, the client caches the required region location information again after an application is restarted. Each thread waits until the cache is complete before executing the requests. If the client timeout is too short and the target table contains a large number of regions, a large number of requests will be retried due to timeout. To prevent this issue, you need to cache the region information in the table before the application threads run so that subsequent requests will not time out.

This sample contains HBase multi-thread read/write tasks. The meta table will be preloaded before the thread tasks are started.

#### NOTE

In this sample, multiple threads concurrently access HBase, where:

- The number of regions of the HBase table is greater than or equal to 100.
- The data table is concurrently accessed by a single application.
- The HBase request timeout on the application side is less than 2s.

### 2.6.3.3.2 Preloading Meta Tables When Request Concurrency Is High

#### Function

Preload region location information in the scenario where multiple threads concurrently read and write HBase tables.

#### Code Sample

The following code snippets are in the **MultiThreadSample** class of the **com.huawei.bigdata.hbase.examples** package.

- Create a user table, pre-split 300 regions, and compress the table using SNAPPY.

```
private void createUserTable() {
    // Use number id as the first part of rowKey and pre-split 300 regions by the decimal string split
    // algorithm.
    ColumnFamilyDescriptor cf = ColumnFamilyDescriptorBuilder
        .newBuilder(Bytes.toBytes("f"))
        .setCompressionType(Compression.Algorithm.SNAPPY)
        .build();
    TableDescriptor td = TableDescriptorBuilder
        .newBuilder(tableName)
        .setColumnFamily(cf)
        .build();
    try (Admin admin = conn.getAdmin()) {
        if (!admin.tableExists(tableName)) {
            RegionSplitter.SplitAlgorithm splitAlgo = RegionSplitter
                .newSplitAlgoInstance(this.conn.getConfiguration(),
                    RegionSplitter.DecimalStringSplit.class.getName());
            admin.createTable(td, splitAlgo.split(REGION_NUMS));
            LOG.info("Create table success.");
        } else {
            LOG.warn("Table already exists.");
        }
    } catch (IOException e) {
        LOG.error("Create table failed!", e);
    }
}
```

- Preload the meta table and obtain the location information of all regions in the user table.

```
private void preLoadTableRegionLocations() {
    try (RegionLocator rl = conn.getRegionLocator(tableName)) {
        long startTime = System.currentTimeMillis();
        rl.getAllRegionLocations();
        LOG.info("Cache table region location success, cost {} ms.", System.currentTimeMillis() -
            startTime);
    } catch (IOException e) {
        LOG.error("Cache table region location failed!", e);
    }
}
```

- Perform multi-thread read and write operations.

```
private void doOperations() {
    int cores = Math.max(32, Runtime.getRuntime().availableProcessors());
```

```
// Pool for user service including HBase data read/write.
ThreadPoolExecutor pool = new ThreadPoolExecutor(cores, cores,
    60,
    TimeUnit.SECONDS,
    new LinkedBlockingQueue<>(cores * 100),
    new ThreadFactoryBuilder()
        .setDaemon(true)
        .setUncaughtExceptionHandler((t, e) ->
            LOG.warn("Thread:{} exited with Exception:{}", t, StringUtils.stringifyException(e)))
        .setNameFormat("Application-Pool-%d").build());

for (int i = 0; i < OPERATIONS_COUNT; i++) {
    pool.execute(() -> {
        try (Table table = conn.getTable(tableName)) {
            Put put = new Put(Bytes.toBytes(
                ThreadLocalRandom.current().nextLong(0, 100000L) + "#"
                    + RandomStringUtils.randomAlphabetic(5)));
            put.addColumn(Bytes.toBytes("f"), Bytes.toBytes("name"),
                Bytes.toBytes(RandomStringUtils.randomAlphabetic(10)));
            put.addColumn(Bytes.toBytes("f"), Bytes.toBytes("ts"),
                Bytes.toBytes(Long.toString(System.currentTimeMillis())));
            table.put(put);
        } catch (IOException e) {
            LOG.error("Put data failed ", e);
        }
    });
    pool.execute(() -> {
        Scan scan = new Scan()
            .withStartRow(Bytes.toBytes(
                ThreadLocalRandom.current().nextLong(0, 100000L) + "#"))
            .setLimit(1);
        try (Table table = conn.getTable(tableName);
            ResultScanner scanner = table.getScanner(scan)) {
            Result result;
            while ((result = scanner.next()) != null) {
                for (Cell cell : result.rawCells()) {
                    LOG.info("ROW:{};CF:{};CQ:{};VALUE:{}",
                        Bytes.toString(CellUtil.cloneRow(cell)),
                        Bytes.toString(CellUtil.cloneFamily(cell)),
                        Bytes.toString(CellUtil.cloneQualifier(cell)),
                        Bytes.toString(CellUtil.cloneValue(cell)));
                }
            }
        } catch (IOException e) {
            LOG.error("Scan data failed ", e);
        }
    });
}
shutDownPool(pool);
LOG.info("Finish do operations.");
}

private void shutDownPool(ThreadPoolExecutor pool) {
    pool.shutdown();
    try {
        if (!pool.awaitTermination(SHUT_DOWN_POOL_WAIT_TIMEOUT_SEC, TimeUnit.SECONDS)) {
            pool.shutdownNow();
        }
    } catch (InterruptedException e) {
        pool.shutdownNow();
    }
}
```

- Clear the user table.

```
private void dropUserTable() {
    try (Admin admin = conn.getAdmin()) {
        admin.disableTable(tableName);
        admin.deleteTable(tableName);
        LOG.info("Drop table success.");
    } catch (IOException e) {
        LOG.error("Drop table failed ", e);
    }
}
```

```
}
```

## 2.6.3.4 HBase Rest API Invoking Sample Program

### 2.6.3.4.1 Querying Cluster Information Using REST

#### Function

Use the REST service to transfer the URL consisting of the host and port to obtain the cluster version and status information through HTTP.

#### Example Code

- Connecting to the RestServer Service

In normal mode, users can connect to the RestServer service without logging in. Therefore, comment out the following code statements related to login in the **main** method of the **HBaseRestTest** class in the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** package:

```
//In Windows environment
//String userdir = HBaseRestTest.class.getClassLoader().getResource("conf").getPath() +
File.separator;
//In Linux environment
//String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
//userKeytabFile = userdir + "user.keytab";
//krb5File = userdir + "krb5.conf";
//String principal = "hbaseuser1";
//login(principal, userKeytabFile, krb5File);
```

The following code snippets are in the **main** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
// RESTServer's hostname.
String restHostName = "xxx.xxx.xxx.xxx";[1]
String securityModeUrl = new
StringBuilder("https://").append(restHostName).append(":21309").toString();
String nonSecurityModeUrl = new
StringBuilder("http://").append(restHostName).append(":21309").toString();
HBaseRestTest test = new HBaseRestTest();

//If cluster is non-security mode,use nonSecurityModeUrl as parameter.
test.test(nonSecurityModeUrl);[2]
```

[1] Change the value of **restHostName** to the IP address of the node where the RestServer instance to be accessed is located, and configure the node IP address in the hosts file on the local host where the sample code is run.

[2] In non-security mode, access the HBase REST service in HTTP mode and use **nonSecurityModeUrl** as the **test.test()** parameter.

- Obtaining the cluster version information

The following code snippets are in the **getClusterVersion** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterVersion(String url) {
    String endpoint = "/version/cluster";
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
    handleNormalResult((Optional<ResultModel>) result);
}
```

- Obtaining the cluster status information

The following code snippets are in the **getClusterStatus** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getClusterStatus(String url) {  
    String endpoint = "/status/cluster";  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);  
    handleNormalResult(result);  
}
```

#### 2.6.3.4.2 Obtaining All Tables Using REST

### Function

Use the REST service and transfer the URL consisting of the host and port to obtain all tables using HTTP.

### Example Code

The following code snippets are in the **getAllUserTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllUserTables(String url) {  
    String endpoint = "/";  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);  
    handleNormalResult(result);  
}
```

#### 2.6.3.4.3 Operate Namespaces Using REST

### Function

Use the REST service to import the URL consisting of the host and port and the specified namespace, Use HTTP to create, query, and delete namespaces, and obtain tables in the specified namespace.

#### ⚠ CAUTION

HBase tables are stored in *Namespace*.*Table name* format. If no namespace is specified when a table is created, the table is stored in **default**. The **hbase** namespace is the system table namespace. Do not create service tables or read or write data in the system table namespace.

### Example Code

- Invoking methods

```
// Namespace operations.  
createNamespace(url, "testNs");  
getAllNamespace(url);  
deleteNamespace(url, "testNs");  
getAllNamespaceTables(url, "default");
```

- Creating a namespace

The following code snippets are in the **createNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createNamespace(String url, String namespace) {  
    String endpoint = "/namespaces/" + namespace;  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.POST, null);  
    if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_CREATED) {  
        LOG.info("Create namespace '{}' success.", namespace);  
    } else {  
        LOG.error("Create namespace '{}' failed.", namespace);  
    }  
}
```

- Querying all namespaces

The following code snippets are in the **getAllNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllNamespace(String url) {  
    String endpoint = "/namespaces";  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);  
    handleNormalResult(result);  
}
```

- Deleting a specified namespace

The following code snippets are in the **deleteNamespace** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteNamespace(String url, String namespace) {  
    String endpoint = "/namespaces/" + namespace;  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.DELETE, null);  
    if (result.orElse(new ResultModel()).getStatusCode() == HttpStatus.SC_OK) {  
        LOG.info("Delete namespace '{}' success.", namespace);  
    } else {  
        LOG.error("Delete namespace '{}' failed.", namespace);  
    }  
}
```

- Obtain tables in a specified namespace.

The following code snippets are in the **getAllNamespaceTables** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getAllNamespaceTables(String url, String namespace) {  
    String endpoint = "/namespaces/" + namespace + "/tables";  
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);  
    handleNormalResult(result);  
}
```

## 2.6.3.4.4 Operate Tables Using REST

### Function

Use the REST service to transfer the URL consisting of the host and port as well as the specified **tableName** and **jsonHTD** to query, modify, create, and delete table information through HTTP.

### Example Code

- Invoking methods

```
// Add a table with specified info.  
createTable(url, "testRest",  
    "{\"name\":\"\\\"default:testRest\\\", \"ColumnSchema\":{\"name\":\"cf1\"}, \"name\":\"cf2\"}}");
```

```
// Add column family 'testCF1' if not exist, else update the 'VERSIONS' to 3.
// Notes: The unspecified property of this column family will be updated to default value.
modifyTable(url, "testRest",
    "{\"name\":\"testRest\",\"ColumnSchema\":[{\"name\":\"testCF1\",\"VERSIONS\":\"3\"}]}");

// Describe specific Table.
descTable(url, "default:testRest");

// delete a table with specified info.
deleteTable(url, "default:testRest",
    "{\"name\":\"default:testRest\",\"ColumnSchema\":[{\"name\":\"testCF\",\"VERSIONS\":\"3\"}]}");

```

- **Querying table information**

The following code snippets are in the **descTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void descTable(String url, String tableName) {
    String endpoint = "/" + tableName + "/schema";
    Optional<ResultModel> result = sendAction(url + endpoint, MethodType.GET, null);
    handleNormalResult((Optional<ResultModel>) result);
}
```

- **Modifying table information**

The following code snippets are in the **modifyTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void modifyTable(String url, String tableName, String jsonHTD) {
    LOG.info("Start modify table.");
    String endpoint = "/" + tableName + "/schema";
    JsonElement tableDesc = new JsonParser().parse(jsonHTD);

    // Add a new column family or modify it.
    handleNormalResult(sendAction(url + endpoint, MethodType.POST, tableDesc));
}
```

- **Creating a table**

The following code snippets are in the **createTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(String url, String tableName, String jsonHTD) {
    LOG.info("Start create table.");
    String endpoint = "/" + tableName + "/schema";
    JsonElement tableDesc = new JsonParser().parse(jsonHTD);

    // Add a table.
    handleCreateTableResult(sendAction(url + endpoint, MethodType.PUT, tableDesc));
}
```

- **Deleting a table**

The following code snippets are in the **deleteTable** method in the **HBaseRestTest** class of the **hbase-rest-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void deleteTable(String url, String tableName, String jsonHTD) {
    LOG.info("Start delete table.");
    String endpoint = "/" + tableName + "/schema";
    JsonElement tableDesc = new JsonParser().parse(jsonHTD);

    // delete a table.
    handleNormalResult(sendAction(url + endpoint, MethodType.DELETE, tableDesc));
}
```

## 2.6.3.5 Accessing the HBase ThriftServer Sample Program

### 2.6.3.5.1 Accessing the ThriftServer Operation Table

#### Scenario

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access ThriftServer, and obtain table names, create a table, and delete a table based on the specified namespace.

#### Procedure

- Step 1** Log in to FusionInsight Manager, choose **Cluster > Service > HBase > Configuration** and click **All Configurations**, search for and modify the parameter **hbase.thrift.security.qop** of the ThriftServer instance. The value of this parameter must be the same as that of **hbase.rpc.protection**. Save the configuration and restart the node service for the configuration to take effect.

##### NOTE

The mapping between **hbase.rpc.protection** and **hbase.thrift.security.qop** is as follows:

- "privacy" - "auth-conf"
- "authentication" - "auth"
- "integrity" - "auth-int"

- Step 2** Obtain the configuration file of the ThriftServer instance in the cluster.

- Method 1: Choose **Cluster > Service > HBase > Instance**, click the ThriftServer instance to go to the details page, and obtain the configuration files **hdfs-site.xml**, **core-site.xml**, and **hbase-site.xml**.
- Method 2: Obtain the configuration files by decompressing the client file in [Preparing for Development and Operating Environment](#). Manually add the following configuration to **hbase-site.xml**. The value of **hbase.thrift.security.qop** must be the same as that in [Step 1](#).

```
<property>
<name>hbase.thrift.security.qop</name>
<value>auth</value>
</property>
<property>
<name>hbase.thrift.kerberos.principal</name>
<value>thrift/hadoop.hadoop.com@HADOOP.COM</value>
</property>
<property>
<name>hbase.thrift.keytab.file</name>
<value>/opt/huawei/Bigdata/FusionInsight_HD_XXX/install/FusionInsight-HBase-2.4.14/keytabs/HBase/
thrift.keytab</value>
</property>
```

----End

#### Example Code

- Initializing configuration

The following code snippets belong to the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-thrift-example** sample project.

```
private static void init() throws IOException {
    // Default load from conf directory
    conf = HBaseConfiguration.create();

    String userdir = TestMain.class.getClassLoader().getResource("conf").getPath() + File.separator;
[1]    //In Linux environment
    //String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
    conf.addResource(new Path(userdir + "core-site.xml"), false);
    conf.addResource(new Path(userdir + "hdfs-site.xml"), false);
    conf.addResource(new Path(userdir + "hbase-site.xml"), false);
}
```

[1] **userdir** obtains the **conf** directory in the resource path after compilation. The **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** files used for initial configuration must be stored in the **src/main/resources/conf** directory.

- Connecting to a ThriftServer instance

The following code snippets belong to the **TestMain** class in the **com.huawei.bigdata.hbase.examples** package of the **hbase-thrift-example** sample project.

```
try {
    test = new ThriftSample();
    test.test("xxx.xxx.xxx.xxx", THRIFT_PORT, conf);[2]
} catch (TException | IOException e) {
    LOG.error("Test thrift error", e);
}
```

[2] The value of the input parameter **test.test()** is the IP address of the node where the ThriftServer instance to be accessed is located. Change the IP address to the actual one. The IP address of the node must be configured in the hosts file of the local host where the sample code is run.

**THRIFT\_PORT** is the value of **hbase.regionserver.thrift.port** configured for the ThriftServer instance.

- Invoking methods

```
// Get table of specified namespace.
getTableNamesByNamespace(client, "default");
// Create table.
createTable(client, TABLE_NAME);
// Delete specified table.
deleteTable(client, TABLE_NAME);
```

- Obtaining table names based on the specified namespace

The following code snippets are in the **getTableNamesByNamespace** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getTableNamesByNamespace(THBaseService.Iface client, String namespace) throws
TException {
    client.getTableNamesByNamespace(namespace)
        .forEach(
            tTableName -> LOGGER.info("{}: {}, TableName.valueOf(tTableName.getNs(),",
            tTableName.getQualifier())));
}
```

- Creating a table

The following code snippets are in the **createTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void createTable(THBaseService.Iface client, String tableName) throws TException,
IOException {
    TTableName table = getTableName(tableName);
    TTableDescriptor descriptor = new TTableDescriptor(table);
    descriptor.setColumns(
```

```

        Collections.singletonList(new
TColumnFamilyDescriptor(). setName(COLUMN_FAMILY.getBytes())));
if (client.tableExists(table)) {
    LOGGER.warn("Table {} is exists, delete it.", tableName);
    client.disableTable(table);
    client.deleteTable(table);
}
client.createTable(descriptor, null);
if (client.tableExists(table)) {
    LOGGER.info("Created {}.", tableName);
} else {
    LOGGER.error("Create {} failed.", tableName);
}
}

```

- Deleting a table

The following code snippets are in the **deleteTable** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```

private void deleteTable(THBaseService.Iface client, String tableName) throws TException,
IOException {
    TTableName table = getTableName(tableName);
    if (client.tableExists(table)) {
        client.disableTable(table);
        client.deleteTable(table);
        LOGGER.info("Deleted {}.", tableName);
    } else {
        LOGGER.warn("{} not exist.", tableName);
    }
}

```

### 2.6.3.5.2 Accessing ThriftServer to Write Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use Put and putMultiple to write data.

#### Example Code

- Invoking methods

```

// Write data with put.
putData(client, TABLE_NAME);

// Write data with putlist.
putDataList(client, TABLE_NAME);

```

- Using Put to write data.

The following code snippets are in the **putData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```

private void putData(THBaseService.Iface client, String tableName) throws TException {
    LOGGER.info("Test putData.");
    TPut put = new TPut();
    put.setRow("row1".getBytes());

    TColumnValue columnValue = new TColumnValue();
    columnValue.setFamily(COLUMN_FAMILY.getBytes());
    columnValue.setQualifier("q1".getBytes());
    columnValue.setValue("test value".getBytes());
    List<TColumnValue> columnValues = new ArrayList<>(1);
}

```

```
columnValues.add(columnValue);
put.setColumnValues(columnValues);

ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
client.put(table, put);
LOGGER.info("Test putData done.");
}
```

- Using `putMultiple` to write data.

The following code snippets are in the `putDataList` method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void putDataList(THBaseService.Iface client, String tableName) throws TException {
    LOGGER.info("Test putDataList.");
    TPut put1 = new TPut();
    put1.setRow("row2".getBytes());
    List<TPut> putList = new ArrayList<>();

    TColumnValue q1Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
        ByteBuffer.wrap("q1".getBytes()), ByteBuffer.wrap("test value".getBytes()));
    TColumnValue q2Value = new TColumnValue(ByteBuffer.wrap(COLUMN_FAMILY.getBytes()),
        ByteBuffer.wrap("q2".getBytes()), ByteBuffer.wrap("test q2 value".getBytes()));
    List<TColumnValue> columnValues = new ArrayList<>(2);
    columnValues.add(q1Value);
    columnValues.add(q2Value);
    put1.setColumnValues(columnValues);
    putList.add(put1);

    TPut put2 = new TPut();
    put2.setRow("row3".getBytes());

    TColumnValue columnValue = new TColumnValue();
    columnValue.setFamily(COLUMN_FAMILY.getBytes());
    columnValue.setQualifier("q1".getBytes());
    columnValue.setValue("test q1 value".getBytes());
    put2.setColumnValues(Collections.singletonList(columnValue));
    putList.add(put2);

    ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
    client.putMultiple(table, putList);
    LOGGER.info("Test putDataList done.");
}
```

### 2.6.3.5.3 Accessing ThriftServer to Read Data

#### Function

After importing the host where the ThriftServer instances are located and the port that provides services, you can create a Thrift client using the authentication credential and configuration file, access the ThriftServer, and use **get** and **scan** methods to read data.

#### Example Code

- Invoking methods

```
// Get data with single get.
getData(client, TABLE_NAME);

// Get data with getlist.
getDataList(client, TABLE_NAME);

// Scan data.
scanData(client, TABLE_NAME);
```

- Using the **get** method to write data.

The following code snippets are in the **getData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getData(THBaseService.Iface client, String tableName) throws TException {
    LOGGER.info("Test getData.");
    TGet get = new TGet();
    get.setRow("row1".getBytes());

    ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
    TResult result = client.get(table, get);
    printResult(result);
    LOGGER.info("Test getData done.");
}
```

- Using the **getlist** method to write data.

The following code snippets are in the **getDataList** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void getDataList(THBaseService.Iface client, String tableName) throws TException {
    LOGGER.info("Test getDataList.");
    List<TGet> getList = new ArrayList<>();
    TGet get1 = new TGet();
    get1.setRow("row1".getBytes());
    getList.add(get1);

    TGet get2 = new TGet();
    get2.setRow("row2".getBytes());
    getList.add(get2);

    ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
    List<TResult> resultList = client.getMultiple(table, getList);
    for (TResult tResult : resultList) {
        printResult(tResult);
    }
    LOGGER.info("Test getDataList done.");
}
```

- Using the **scan** method to write data.

The following code snippets are in the **scanData** method in the **ThriftSample** class of the **hbase-thrift-example\src\main\java\com\huawei\hadoop\hbase\examples** packet.

```
private void scanData(THBaseService.Iface client, String tableName) throws TException {
    LOGGER.info("Test scanData.");
    int scannerId = -1;
    try {
        ByteBuffer table = ByteBuffer.wrap(tableName.getBytes());
        TScan scan = new TScan();
        scan.setLimit(500);
        scannerId = client.openScanner(table, scan);
        List<TResult> resultList = client.getScannerRows(scannerId, 100);
        while (resultList != null && !resultList.isEmpty()) {
            for (TResult tResult : resultList) {
                printResult(tResult);
            }
            resultList = client.getScannerRows(scannerId, 100);
        }
    } finally {
        if (scannerId != -1) {
            client.closeScanner(scannerId);
            LOGGER.info("Closed scanner {}.", scannerId);
        }
    }
    LOGGER.info("Test scanData done.");
}
```

## 2.6.3.6 Sample Program for HBase to Access Multiple ZooKeepers

### 2.6.3.6.1 Accessing Multiple ZooKeepers

#### Function

This function allows simultaneous access to FusionInsight ZooKeeper from the HBase client and third-party ZooKeeper from the customer application in the same client process.

#### Example code

The following code snippet is in the **TestZKSample** class of the **hbase-zk-example** **\src\main\java\com\huawei\hadoop\hbase\example**. You need to pay attention to the **login** and **connectApacheZK** methods.

```
private static void login(String keytabFile, String principal) throws IOException {
    conf = HBaseConfiguration.create();
    //In Windows environment
    String confDirPath = TestZKSample.class.getClassLoader().getResource("").getPath() + File.separator;
[1]    //In Linux environment
    //String confDirPath = System.getProperty("user.dir") + File.separator + "conf" + File.separator;

    // Set zoo.cfg for hbase to connect to fi zookeeper.
    conf.set("hbase.client.zookeeper.config.path", confDirPath + "zoo.cfg");
    if (User.isHBaseSecurityEnabled(conf)) {
        // jaas.conf file, it is included in the client package file
        System.setProperty("java.security.auth.login.config", confDirPath + "jaas.conf");
        // set the kerberos server info, point to the kerberosclient
        System.setProperty("java.security.krb5.conf", confDirPath + "krb5.conf");
        // set the keytab file name
        conf.set("username.client.keytab.file", confDirPath + keytabFile);
        // set the user's principal
        try {
            conf.set("username.client.kerberos.principal", principal);
            User.login(conf, "username.client.keytab.file", "username.client.kerberos.principal",
                      InetAddress.getLocalHost().getCanonicalHostName());
        } catch (IOException e) {
            throw new IOException("Login failed.", e);
        }
    }
}
private void connectApacheZK() throws IOException, org.apache.zookeeper.KeeperException {
    try {
        // Create apache zookeeper connection.
        ZooKeeper digestZk = new ZooKeeper("127.0.0.1:24002", 60000, null);
        LOG.info("digest directory: {}", digestZk.getChildren("/", null));
        LOG.info("Successfully connect to apache zookeeper.");
    } catch (InterruptedException e) {
        LOG.error("Found error when connect apache zookeeper ", e);
    }
}
```

- [1]The value of **userdir** is the path of the **conf** directory in the resource path after compilation. Place the **core-site.xml**, **hdfs-site.xml**, and **hbase-site.xml** configuration files to the **src/main/resources/conf** directory.
- The **jaas.conf** file specified by the **java.security.auth.login.config** parameter in the **login** method is used to set the authentication information for accessing ZooKeeper. The example code contains the Client\_new and Client configurations. The Client\_new configuration is used to access FusionInsight ZooKeeper and the Client configuration is used to access Apache ZooKeeper.

- The **hbase.client.zookeeper.config.path** parameter in the **login** method controls the access to the FusionInsight ZooKeeper client. The following parameters are involved:
  - **zookeeper.sasl.clientconfig**: Specifies the configuration in the **jaas.conf** file used for accessing FusionInsight ZooKeeper.
  - **zookeeper.server.principal**: Specifies the principle of the ZooKeeper server. The format is **zookeeper/hadoop.System domain name**, for example, **zookeeper/hadoop.HADOOP.COM**. To obtain the system domain name, log in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and view the value of Local Domain.
  - **zookeeper.sasl.client**: If the cluster works in the security mode, set this parameter to **true**. Otherwise, set this parameter to **false**. If this parameter is set to **false**, the **zookeeper.sasl.clientconfig** and **zookeeper.server.principal** parameters do not take effect.

### 2.6.3.7 Example Configuration for Interconnecting HBase/Phoenix with SpringBoot

#### Scenario

Run Spring Boot interface sample code of MRS HBase/Phoenix.

#### Prerequisites

You have obtained the configuration file required for running the sample project. For details, see [Preparing the Connection Cluster Configuration File](#).

#### Procedure

**Step 1** In the IntelliJ IDEA development environment, click the **springclient.properties** file in the **src/springboot/hbase-examples/src/main/resources** directory and modify the parameters listed in [Table 2-69](#) as needed:

**Table 2-69** Configuration parameters

Parameter	Definition
conf.path	Directory where the HBase configuration file is stored, that is, <b>hbase-example\src\main\resources\conf</b>

----End

### 2.6.4 Application Commissioning

#### 2.6.4.1 Commissioning an Application in Windows

### 2.6.4.1.1 Compiling and Running an Application

#### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

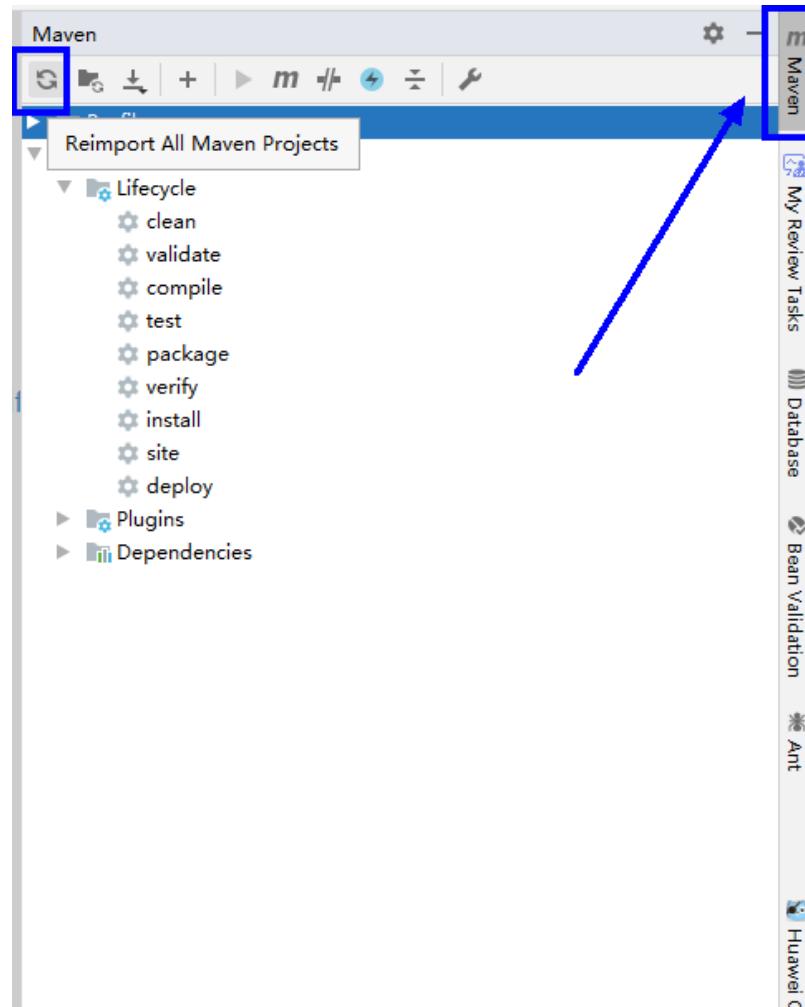
#### NOTE

- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

#### Procedure

**Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

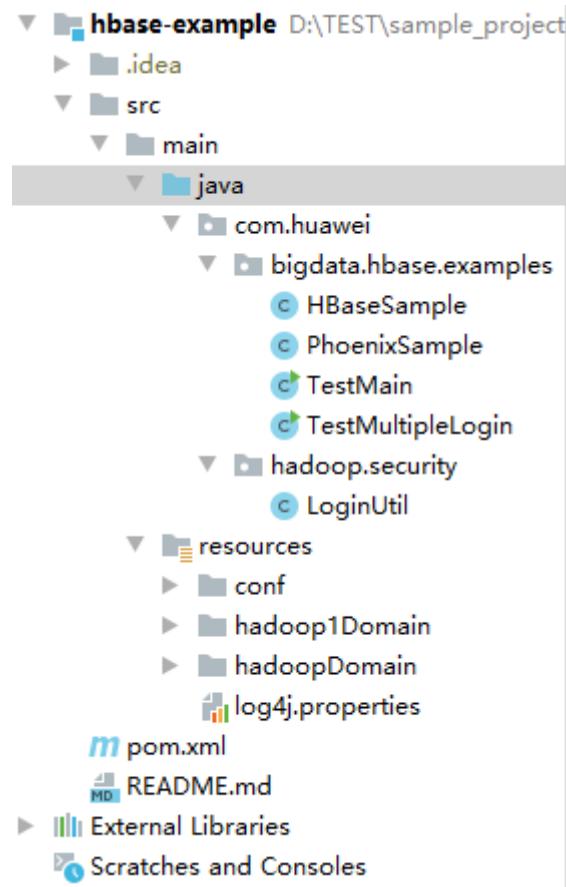
**Figure 2-124** reimport projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user. See [Figure 2-125](#).

**Figure 2-125** Directory list of **hbase-example** to be compiled



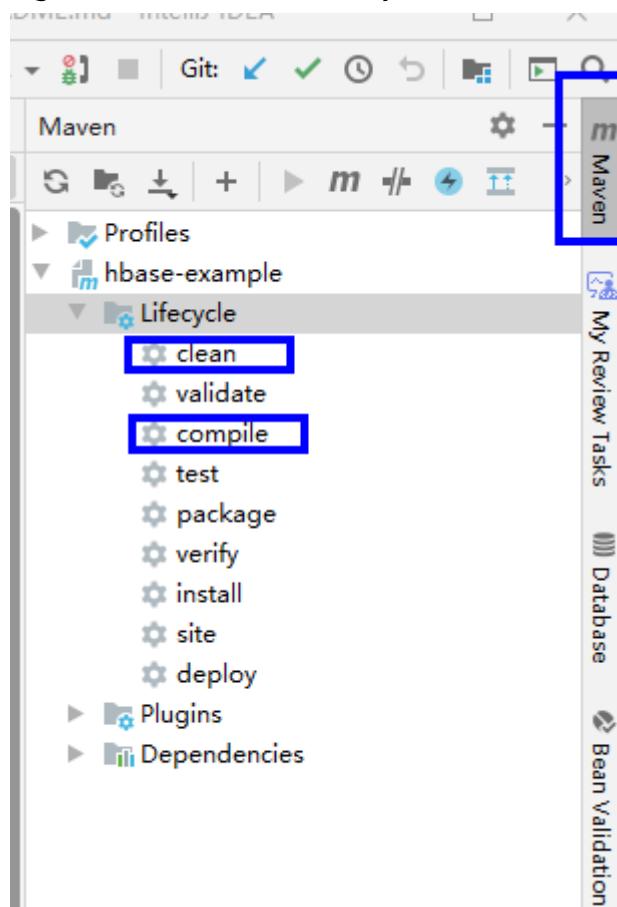
1. Compile an application.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

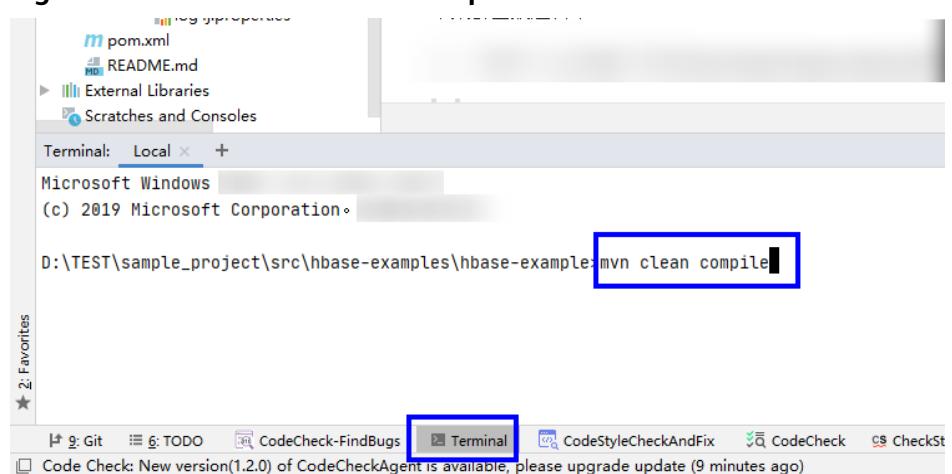
Figure 2-126 clean and compile tools of Maven



- Method 2:

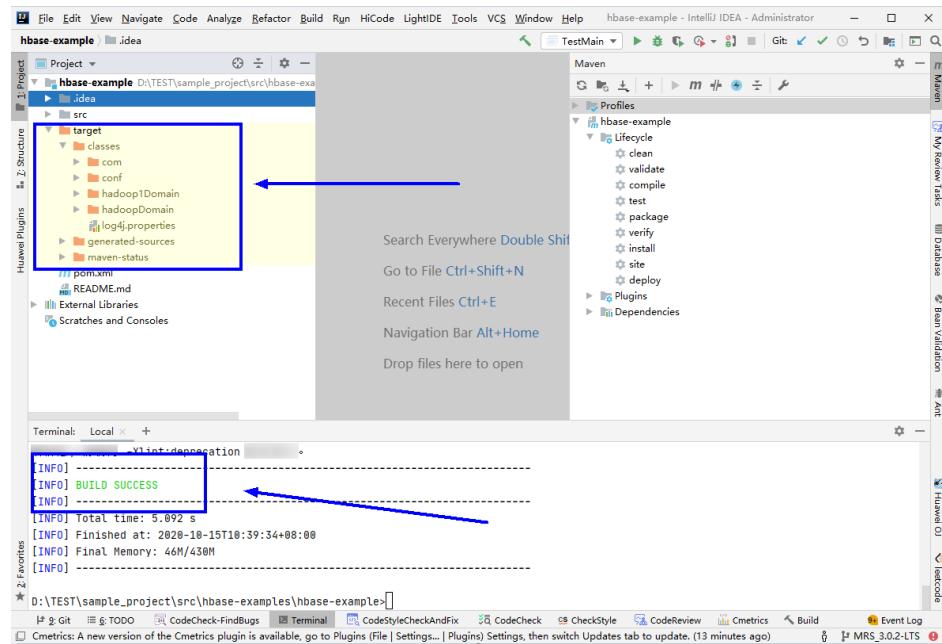
Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 2-127 Enter **mvn clean compile** in the IDEA Terminal text box.



2. After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

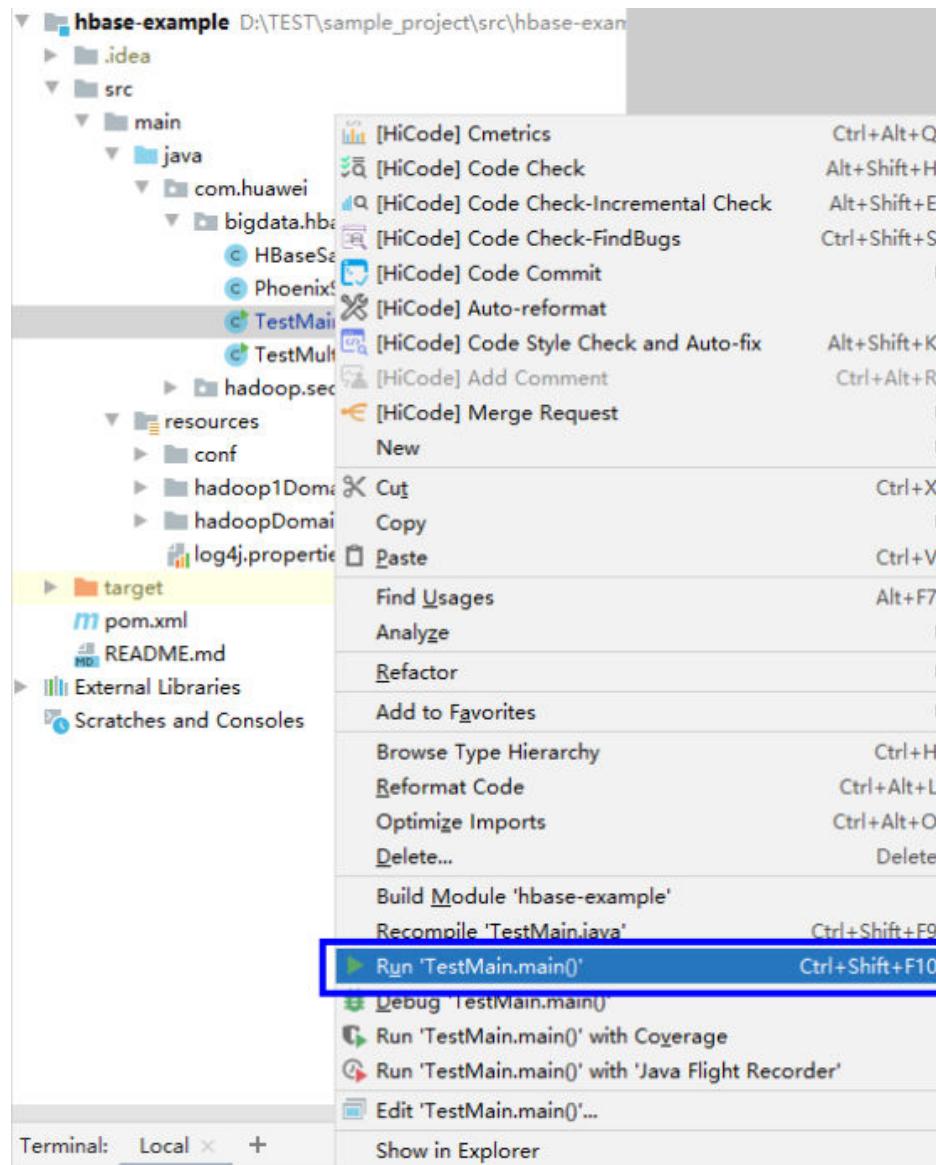
Figure 2-128 Compilation completed



3. Run the program.

Right-click the **TestMain.java** file and choose **Run'TestMain.main()** from the shortcut menu.

Figure 2-129 Run the application.



----End

#### 2.6.4.1.2 Viewing Windows Commissioning Results

##### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the IntelliJ IDEA running result.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see **More Information > External Interfaces > WebUI**.
- Using HBase Shell command, see **More Information > External Interfaces > Shell**.

## Procedure

- After the **hbase-sample** is successfully executed, the following information is displayed:

```
2016-07-13 14:36:12,736 INFO [main] basic.CreateTableSample: Create table sampleNameSpace:sampleTable successful!
2016-07-13 14:36:15,426 INFO [main] basic.ModifyTableSample: Modify table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:16,708 INFO [main] basic.MultiSplitSample: Mmulti split table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:17,299 INFO [main] basic.PutDataSample: Successfully put 9 items data into sampleNameSpace:sampleTable.
2016-07-13 14:36:18,992 INFO [main] basic.ScanSample: Scan data successfully.
2016-07-13 14:36:20,532 INFO [main] basic.DeleteDataSample: Successfully delete data from table sampleNameSpace:sampleTable.
2016-07-13 14:36:21,006 INFO [main] acl.AclSample: Grant ACL for table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:27,836 INFO [main] index.CreateIndexSample: Successfully add index for table sampleNameSpace:sampleTable.
```

### NOTE

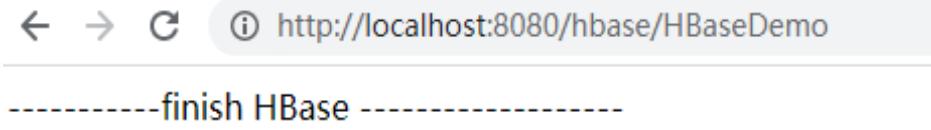
In the Windows environment, the following exception occurs but does not affect services.

`java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.`

- Result of interconnecting HBase/Phoenix with SpringBoot:

Open a browser and access **http://IP address of the sample running node:8080/hbase/HBaseDemo** and **http://IP address of the sample running node:8080/hbase/PhoenixDemo**. IDEA prints logs normally, and "finish HBase" and "finish Phoenix" are returned.

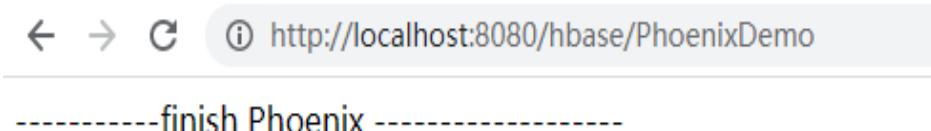
**Figure 2-130** "finish HBase" displayed



← → ⌂ ⓘ <http://localhost:8080/hbase/HBaseDemo>

-----finish HBase-----

**Figure 2-131** "finish Phoenix" returned



← → ⌂ ⓘ <http://localhost:8080/hbase/PhoenixDemo>

-----finish Phoenix-----

- Log description

The log level is **INFO** by default and more detailed information can be viewed by changing the log level (**DEBUG**, **INFO**, **WARN**, **ERROR**, and **FATL**). You can modify the **log4j.properties** file to change log levels, for example:

```
hbase.root.logger=INFO,console
...
log4j.logger.org.apache.zookeeper=INFO
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG
log4j.logger.org.apache.hadoop.hbase=INFO
# Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO
```

```
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO  
...
```

## 2.6.4.2 Commissioning an Application in Linux

### 2.6.4.2.1 Compiling and Running an Application When a Client Is Installed

#### Scenario

In a Linux environment where an HBase client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

#### Prerequisites

- You have installed the HBase client.
- If the host where the client is installed is not a node in the cluster, set the mapping between the host name and the IP address in the **hosts** file on the node where the client locates. The host name and IP address must be in one-to-one mapping.
- The JDK has been installed and Java environment variables have been correctly configured before Windows commissioning and compilation.

#### Procedure

##### Step 1 Export a JAR package.

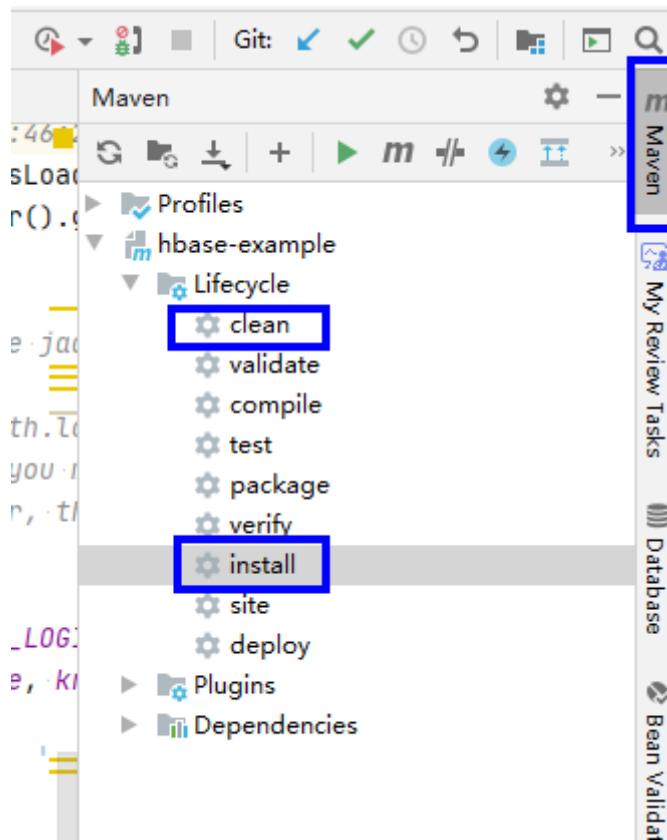
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

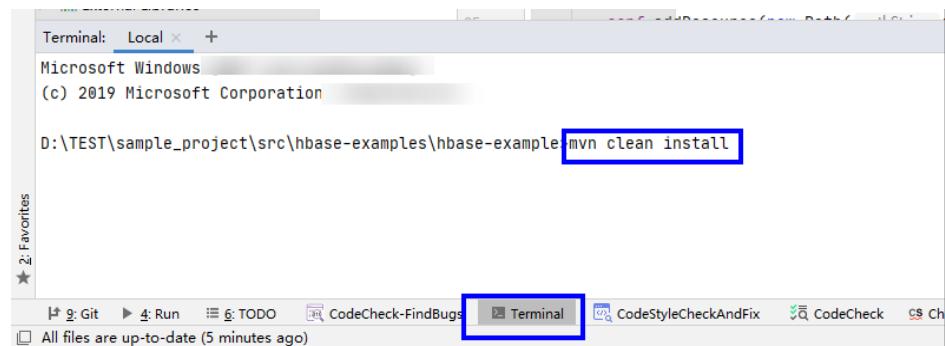
Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 2-132 Maven tools: clean and install



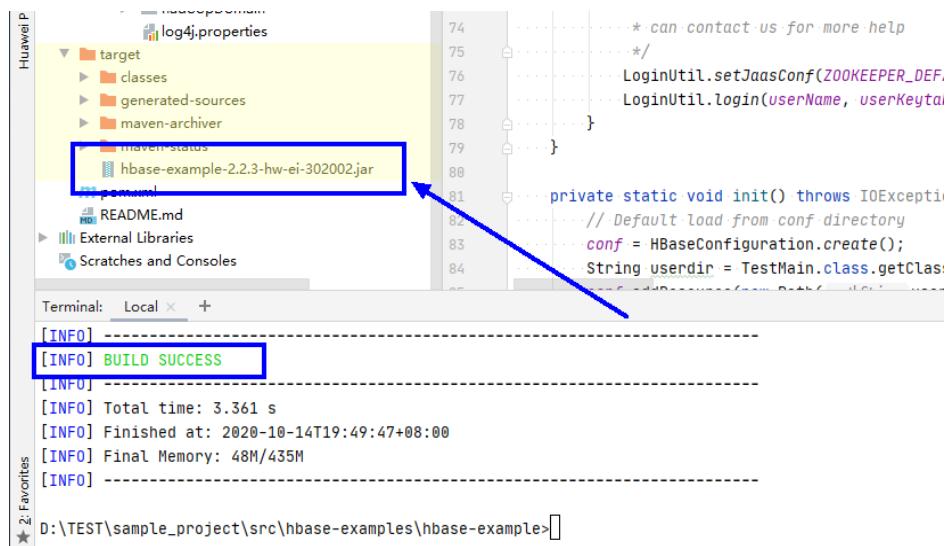
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 2-133 Enter mvn clean compile in the IDEA Terminal text box.



After the compilation is completed, the message "Build Success" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

Figure 2-134 When the compilation is completed, the JAR file is generated.



**Step 2** Example of whether to run HBase/Phoenix to interconnect with SpringBoot:

- If yes, perform the following steps to run the sample:
  - a. Create a running directory in the Linux environment and place **hbase-springboot-\*.jar** in the target directory to the directory. Upload the configuration file to the corresponding path specified in [Step 1](#).
  - b. Switch to the running directory and run the following command to run the JAR package:  
**java -jar hbase-springboot-\*.jar**
- If no, go to [Step 3](#).

**Step 3** Export the JAR file on which the sample project depends.

Access the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of IDEA or by using other command line tools.

Run the command **mvn dependency:copy-dependencies -DoutputDirectory=lib**.

The **lib** folder is generated in the directory where the **pom.xml** file is located. The **lib** folder contains the JAR files on which the sample project depends.

**Step 4** Run the JAR package.

1. Before running the JAR package on the Linux client, run the following command to go to the client directory:

**cd \$BIGDATA\_CLIENT\_HOME**

**NOTE**

**\$BIGDATA\_CLIENT\_HOME** is the installation directory of the HBase client.

2. Then run the following command:

**source bigdata\_env**

 NOTE

After the multi-instance function is enabled, run the following command to switch to the client of the specified service instance before performing application development for the HBase service instance.

For example, for HBase2, run the `source /opt/hadoopclient/HBase2/component_env` command.

3. Upload the JAR file (non-dependent JAR file) of the sample project generated in the application development environment to the *Client installation directory/HBase/hbase/lib* directory in the client running environment. If the HBase dual-read sample needs to be executed, Upload the hbase-dualclient JAR package exported in [Step 3](#) to the directory. In addition, you need to copy the configuration file required by the example project obtained from [Preparing the Connection Cluster Configuration File](#) to the *client installation directory/HBase/hbase/conf* directory.
4. Go to the `$BIGDATA_CLIENT_HOME/HBase/hbase` directory and run the following command to run the JAR package. The task is complete.  
`hbase com.huawei.bigdata.hbase.examples.TestMain`  
In the preceding command, `hbase com.huawei.bigdata.hbase.examples.TestMain` is used as an example.

----End

#### 2.6.4.2.2 Compiling and Running an Application When No Client Is Installed

##### Scenario

In a Linux environment where no HBase client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

##### Prerequisites

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by the JAR package exported by IntelliJ IDEA.
- If the Linux host is not a node in the cluster, set the mapping between the host name and the IP address in the `hosts` file on the node. The host name and IP address must be in one-to-one mapping.

##### Procedure

**Step 1** Export a JAR package. For details, see [Step 1](#) in section [Compiling and Running an Application When a Client Is Installed](#).

**Step 2** Example of whether to run HBase/Phoenix to interconnect with SpringBoot:

- If yes, perform the following steps to run the sample:
  - a. Create a running directory in the Linux environment and place `hbase-springboot-*jar` in the target directory to the directory. Upload the configuration file to the corresponding path specified in [Step 1](#).
  - b. Switch to the running directory and run the following command to run the JAR package:

**java -jar hbase-springboot-\*jar**

- If no, go to [Step 3](#).

**Step 3** Prepare for the required JAR packages and configuration files.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories **lib** and **conf**. Export the JAR package that the sample project depends on. For details about how to export the JAR package, see [Step 3 in 1.4.2.1 Compiling and Running an Application When a Client Is Installed](#). Upload this JAR file and that exported in [Step 1](#) to the **lib** directory on the Linux server. Upload the configuration file obtained in section [Preparing the Connection Cluster Configuration File](#) to the **conf** directory on Linux.
2. In **/opt/test**, create the **run.sh** script, modify the following content, and save the file:

```
#!/bin/sh  
BASEDIR=`cd $(dirname $0);pwd`  
cd ${BASEDIR}  
for file in ${BASEDIR}/lib/*.jar  
do  
i_cp=$i_cp:$file  
echo "$file"  
done  
for file in ${BASEDIR}/conf/*  
do  
i_cp=$i_cp:$file  
done  
  
java -cp .${i_cp} com.huawei.bigdata.hbase.examples.TestMain
```

**Step 4** Go to **/opt/test** and run the following commands to run the JAR packages:

**sh run.sh**

----End

### 2.6.4.2.3 Viewing Linux Commissioning Results

#### Scenario

After an HBase application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Viewing HBase logs.
- Logging in to the HBase WebUI, see [More Information > External Interfaces > WebUI](#).
- Using HBase Shell command, see [More Information > External Interfaces > Shell](#).

#### Procedure

- You can view the execution details of the submitted application in the run logs. For example, after the **hbase-sample** is successfully executed, the following information is displayed:

```
2020-07-13 14:36:12,736 INFO [main] basic.CreateTableSample: Create table
```

```
sampleNameSpace:sampleTable successful!
```

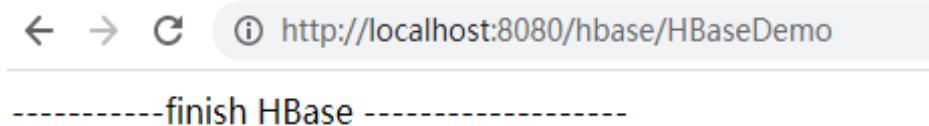
```
2020-07-13 14:36:15,426 INFO [main] basic.ModifyTableSample: Modify table
```

```
sampleNameSpace:sampleTable successfully.  
2020-07-13 14:36:16,708 INFO [main] basic.MultiSplitSample: Mmulti split table  
sampleNameSpace:sampleTable successfully.  
2020-07-13 14:36:17,299 INFO [main] basic.PutDataSample: Successfully put 9 items data into  
sampleNameSpace:sampleTable.  
2020-07-13 14:36:18,992 INFO [main] basic.ScanSample: Scan data successfully.  
2020-07-13 14:36:20,532 INFO [main] basic.DeleteDataSample: Successfully delete data from table  
sampleNameSpace:sampleTable.  
2020-07-13 14:36:21,006 INFO [main] acl.AclSample: Grant ACL for table  
sampleNameSpace:sampleTable successfully.  
2020-07-13 14:36:27,836 INFO [main] index.CreateIndexSample: Successfully add index for table  
sampleNameSpace:sampleTable.
```

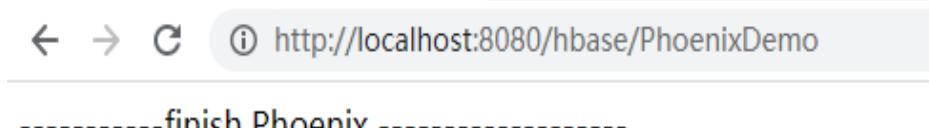
- Result of interconnecting HBase/Phoenix with SpringBoot:

Open a browser and access **http://IP address of the sample running node:8080/hbase/HBaseDemo** and **http://IP address of the sample running node:8080/hbase/PhoenixDemo**. IDEA prints logs normally, and "finish HBase" and "finish Phoenix" are returned.

**Figure 2-135 "finish HBase" displayed**



**Figure 2-136 "finish Phoenix" returned**



## 2.6.5 More Information

### 2.6.5.1 SQL Query

#### Function

Phoenix is an intermediate structured query language (SQL) layer built on HBase. Phoenix provides a JDBC driver that can be embedded in a client. The Phoenix query engine converts input SQL statements to one or multiple HBase scans, and compiles and executes the scan tasks to generate a standard JDBC result set.

#### Example Code

- The **hbase-example/conf/hbase-site.xml** file on the client is used to configure the temporary directory for storing query results. If the client program configures the temporary directory in a Linux environment, configure a Linux path. If the client program configures the temporary directory in a Windows environment, configure a Windows path.

```
<property>  
  <name>phoenix.spool.directory</name>
```

```
<value>[1] Temporary directory for storing intermediate query results</value>
</property>
```

- JAVA Example: Use the JDBC interface to access HBase.

```
public String getURL(Configuration conf)
{
    String phoenix_jdbc = "jdbc:phoenix";
    String zkQuorum = conf.get("hbase.zookeeper.quorum");
    return phoenix_jdbc + ":" + zkQuorum;
}

public void testSQL()
{
    String tableName = "TEST";
    // Create table
    String createTableSQL = "CREATE TABLE IF NOT EXISTS TEST(id integer not null primary key,
name varchar, account char(6), birth date)";

    // Delete table
    String dropTableSQL = "DROP TABLE TEST";

    // Insert
    String upsertSQL = "UPSERT INTO TEST VALUES(1,'John','100000',
TO_DATE('1980-01-01','yyyy-MM-dd'))";

    // Query
    String querySQL = "SELECT * FROM TEST WHERE id = ?";

    // Create the Configuration instance
    Configuration conf = getConfiguration();

    // Get URL
    String URL = getURL(conf);

    Connection conn = null;
    PreparedStatement preStat = null;
    Statement stat = null;
    ResultSet result = null;

    try
    {
        // Create Connection
        conn = DriverManager.getConnection(URL);
        // Create Statement
        stat = conn.createStatement();
        // Execute Create SQL
        stat.executeUpdate(createTableSQL);
        // Execute Update SQL
        stat.executeUpdate(upsertSQL);
        // Create PrepareStatement
        preStat = conn.prepareStatement(querySQL);
        conn.commit();
        // Execute query
        preStat.setInt(1,1);
        result = preStat.executeQuery();
        // Get result
        while (result.next())
        {
            int id = result.getInt("id");
            String name = result.getString(1);
        }
    }
    catch (Exception e)
    {
        // handler exception
    }
    finally
    {
        if(null != result){
            try {
```

```
        result.close();
    } catch (Exception e2) {
        // handler exception
    }
}
if(null != stat){
    try {
        stat.close();
    } catch (Exception e2) {
        // handler exception
    }
}
if(null != conn){
    try {
        conn.close();
    } catch (Exception e2) {
        // handler exception
    }
}
}
```

## Precaution

- You need to configure a temporary directory for storing intermediate query results in **hbase-site.xml**. The size of the query result set is restricted by the directory size.
- Phoenix provides most **java.sql** interfaces and follows the ANSI SQL standard.

### 2.6.5.2 HBase Dual-Read Configuration Items

This section provides the details of all the configurations required for the HBase dual-read feature.

## HBase Dual-Read Operations

**Table 2-70** Configuration items in hbase-dual.xml

Configuration Item	Description	Default Value	Level
hbase.dualclient.active.cluster.configuration.path	HBase client configuration directory of the active cluster	None	Mandatory
hbase.dualclient.standby.cluster.configuration.path	HBase client configuration directory of the standby cluster	None	Mandatory
dual.client.schedule.update.table.delay.second	DR table update interval	5	Optional

Configuration Item	Description	Default Value	Level
hbase.dualclient.gitchtimeout.ms	Maximum glitch time can be tolerated in the active cluster	50	Optional
hbase.dualclient.slow.query.timeout.ms	Slow query alarm log	180000	Optional
hbase.dualclient.active.cluster.id	Active cluster ID	ACTIVE	Optional
hbase.dualclient.standby.cluster.id	Standby cluster ID	STANDBY	Optional
hbase.dualclient.active.executor.threads.max	Maximum size of the thread pool for processing requests to the active cluster	100	Optional
hbase.dualclient.active.executor.threads.core	Core size of the thread pool for processing requests to the active cluster	100	Optional
hbase.dualclient.active.executor.queue	Queue size of the thread pool for processing requests to the active cluster	256	Optional
hbase.dualclient.standby.executor.threads.max	Maximum size of the thread pool for processing requests to the standby cluster	100	Optional
hbase.dualclient.standby.executor.threads.core	Core size of the thread pool for processing requests to the standby cluster	100	Optional
hbase.dualclient.standby.executor.queue	Queue size of the thread pool for processing requests to the standby cluster	256	Optional

Configuration Item	Description	Default Value	Level
hbase.dualclient.clear.executor.threads.max	Maximum size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.threads.core	Core size of the thread pool for clearing resources	30	Optional
hbase.dualclient.clear.executor.queue	Queue size of the thread pool for clearing resources	Integer.MAX_VALUE	Optional
dual.client.metrics.enable	Whether to print client metric information	true	Optional
dual.client.schedule.metrics.second	Interval for printing client metric information	300	Optional
dual.client.asynchronous.enable	Whether to asynchronously request the active and standby clusters	false	Optional

## Printing Metric Information

**Table 2-71** Basic specifications

Metric Name	Description	Log level
total_request_count	Total number of queries in a period	INFO
active_success_count	Number of successful queries in the active cluster in a period	INFO
active_error_count	Number of failed queries in the active cluster in a period	INFO
active_timeout_count	Number of query timeouts in the active cluster in a period	INFO

Metric Name	Description	Log level
standby_success_count	Number of successful queries in the standby cluster in a period	INFO
standby_error_count	Number of failed queries in the standby cluster in a period	INFO
Active Thread pool	Periodically printed information about the thread pool for processing requests to the active cluster	DEBUG
Standby Thread pool	Periodically printed information about the thread pool for processing requests to the standby cluster	DEBUG
Clear Thread pool	Periodically printed information about the thread pool for releasing resources	DEBUG

**Table 2-72 Histogram indicators for GET, BatchGET, and SCAN requests**

Metric Name	Description	Log level
averageLatency(ms)	Average latency	INFO
minLatency(ms)	Minimum latency	INFO
maxLatency(ms)	Maximum latency	INFO
95thPercentileLatency(ms)	Maximum latency of 95% requests	INFO
99thPercentileLatency(ms)	Maximum latency of 99% requests	INFO
99.9PercentileLatency(ms)	Maximum latency of 99.9% requests	INFO
99.99PercentileLatency(ms)	Maximum latency of 99.99% requests	INFO

### 2.6.5.3 External Interfaces

### 2.6.5.3.1 Shell

You can directly perform operations on HBase using Shell on the server. Versions of Shell interfaces of HBase need to be consistent with those in the open-source community. For details, see <http://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/>.

Methods of running the Shell command:

Go to any directory on the HBase client and run the following command:

**hbase shell**

Go to the running mode of the HBase command (also called CLI client connection).

```
hbase(main):001:0>
```

Run the **help** command to obtain the help information of the HBase command parameters.

## Precautions

The **count** command does not support conditional statistics. It supports only full table statistics.

## Command to retrieve HBase replication metrics

All the required metrics will be added for the shell command "status".

- Command to view replication source metrics.

```
hbase(main):019:0> status 'replication', 'source'
```

The output is as follows:

```
version 2.4.14
1 live servers
BLR1000006595:
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0 PeerID=3,
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,
TimeStampsOfLastShippedOp=Wed May 25 20:44:42 CST 2016, Replication Lag=0,
FailedReplicationAttempts=0
```

- Command to view replication sink metrics.

```
hbase(main):020:0> status 'replication', 'sink'
```

The output is as follows:

```
version 2.4.14
1 live servers
BLR1000006595:
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

- Command to view both replication source and replication sink metrics.

```
hbase(main):018:0> status 'replication'
```

The output is as follows:

```
version 2.4.14
1 live servers
```

```
BLR1000006595:  
SOURCE: PeerID=1, SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0,  
LogReadInBytes=1389, LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,  
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,  
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0 PeerID=3,  
SizeOfLogQueue=0, ShippedBatches=0, ShippedOps=0, ShippedBytes=0, LogReadInBytes=1389,  
LogEditsRead=4, LogEditsFiltered=4, SizeOfLogToReplicate=0,  
TimeForLogToReplicate=0, ShippedHFiles=0, SizeOfHFileRefsQueue=0, AgeOfLastShippedOp=0,  
TimeStampsOfLastShippedOp=Wed May 25 20:43:24 CST 2016, Replication Lag=0,  
FailedReplicationAttempts=0  
SINK : AppliedBatches=0, AppliedOps=0, AppliedHFiles=0, AgeOfLastAppliedOp=0,  
TimeStampsOfLastAppliedOp=Wed May 25 17:55:21 CST 2016
```

### 2.6.5.3.2 Java APIs

For details about HBase APIs, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

### API Usage Suggestions

- org.apache.hadoop.hbase.Cell rather than org.apache.hadoop.hbase.KeyValue is recommended as the KV data object.
- It is recommended that Connection connection = ConnectionFactory.createConnection(conf) be used to create a connection. The HTablePool is abandoned.
- org.apache.hadoop.hbase.mapreduce rather than org.apache.hadoop.hbase.mapred is recommended.
- You are advised to obtain the HBase client operation object using the getAdmin() method of the constructed Connection object.

### Common HBase APIs

Common HBase Java classes are as follows:

- Interface class Admin: It is the core class of HBase client applications, which encapsulates APIs for HBase management, such as table creation and table deletion. For details, see [Table 2-73](#).
- Interface class Table: It is a class for HBase read/write, which encapsulates APIs for reading and writing HBase tables. For details, see [Table 2-74](#).

**Table 2-73** org.apache.hadoop.hbase.client.Admin

Method	Description
boolean tableExists(final TableName tableName)	This method is used to check whether a specified table exists. If the table exists in the <b>hbase:meta</b> table, <b>true</b> is returned. Otherwise, <b>false</b> is returned.

Method	Description
HTableDescriptor[] listTables(String regex)	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default.
HTableDescriptor[] listTables(final Pattern pattern, final boolean includeSysTables)	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.
TableName[] listTableNames(String regex)	This method is used to view the user table that complies with the specified regular expression format. There are two other overload methods, one with the input parameter type of Pattern, and the other with the empty input parameter. When the input parameter is empty, all user tables are queried by default. The function of this method is similar to that of listTables. The only difference is that this method returns TableName[].
TableName[] listTableNames(final Pattern pattern, final boolean includeSysTables)	The function of this method is similar to that of the previous method. Users can use this method to specify whether the returned result contains the system table. The previous method returns only the user table.
void createTable(HTableDescriptor desc)	This method is used to create a table with only one region.
void createTable(HTableDescriptor desc, byte[] startKey, byte[] endKey, int numRegions)	This method is used to create a table with a specified number of regions. The endKey of the first region is the StartKey, and the StartKey of the last region is the endKey. If there are a large number of regions, the invoking of this method may time out.

Method	Description
void createTable(final HTableDescriptor desc, byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the StartKey of each region are determined by splitKeys. If there are a large number of regions, the invoking of this method may time out.
void createTable(final HTableDescriptor desc, final byte[][] splitKeys)	This method is used to create a table. The number of regions in the table and the StartKey of each region are determined by splitKeys. This method uses asynchronous invoking, and does not wait for the created table to be online.
void deleteTable(final TableName tableName)	This method is used to delete a specified table.
public void truncateTable(final TableName tableName, final boolean preserveSplits)	This method is used to re-create a specified table. If the second parameter is set to <b>true</b> , the region of the reconstructed table is the same as that of the previous table. Otherwise, there is only one region.
void enableTable(final TableName tableName)	This method is used to enable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void enableTableAsync(final TableName tableName)	This method is used to enable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be online.
void disableTable(final TableName tableName)	This method is used to disable a specified table. If there are a large number of regions in a table, the invoking of this method may time out.
void disableTableAsync(final TableName tableName)	This method is used to disable a specified table. This method uses asynchronous invoking, and does not wait for all regions to be offline.
boolean isTableEnabled(TableName tableName)	This method is used to check whether a table is enabled. This method can be used with the enableTableAsync method to check whether the operation of enabling a table is complete.

Method	Description
boolean isTableDisabled(TableName tableName)	This method is used to check whether a table is disabled. This method can be used with the disableTableAsync method to check whether the operation of disabling a table is complete.
void addColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to add a column family to a specified table.
void deleteColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to delete a specified column family from a specified table.
void modifyColumn(final TableName tableName, final HColumnDescriptor column)	This method is used to modify a specified column family.

**Table 2-74** org.apache.hadoop.hbase.client.Table

Method	Description
boolean exists(Get get)	This method is used to check whether the specified rowkey exists in the table.
boolean[] existsAll(List<Get> gets)	This method is used to check whether the specified rowkey exists in the table. The returned Boolean array result corresponds to the input parameter position.
Result get(Get get)	This method is used to read data based on the specified RowKey.
Result[] get(List<Get> gets)	This method is used to read data in batches by specifying a batch of RowKeys.
ResultScanner getScanner(Scan scan)	This method is used to obtain a scanner object in the table. The query parameters can be specified by the input parameter scan, including <b>StartRow</b> , <b>StopRow</b> , and <b>caching</b> .
void put(Put put)	This method is used to write a data record to the table.
void put(List<Put> puts)	This method is used to write a batch of data records to the table.

Method	Description
void close()	This method is used to release the resources held by the table object.

 NOTE

[Table 2-73](#) and [Table 2-74](#) list only some common methods. For more methods, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

### 2.6.5.3.3 SQLLine

You can run `sqlline.py` on the client to perform SQL operations on HBase. The SQLLine APIs of Phoenix are the same as those of the open-source community. For details, see <http://phoenix.apache.org/>.

[Table 2-75](#) lists common SQLline syntax, [Table 2-76](#) lists common functions, and [Phoenix Command Line](#) describes how to use commands.

**Table 2-75** Common syntax

Command	Description	Example
<b>CREATE TABLE</b>	Creates a table.	CREATE TABLE MY_TABLE(id BIGINT not null primary key, name VARCHAR);
<b>ALTER</b>	Alters a table or a view.	ALTER TABLE MY_TABLE DROP COLUMN name;
<b>DROP TABLE</b>	Deletes a table.	DROP TABLE MY_TABLE;
<b>UPSERT VALUES</b>	Inserts or changes data.	UPSERT INTO MY_TABLE VALUES(1,'abc');
<b>SELECT</b>	Queries data.	SELECT * FROM MY_TABLE;
<b>CREATE INDEX</b>	Creates a global index.	CREATE INDEX MY_IDX ON MY_TABLE(name);
<b>CREATE LOCAL INDEX</b>	Creates a local index.	CREATE LOCAL INDEX MY_LOCAL_IDX ON MY_TABLE(id,name);
<b>ALTER INDEX</b>	Changes the index status.	ALTER INDEX MY_IDX ON MY_TABLE DISABLE;
<b>DROP INDEX</b>	Deletes an index.	DROP INDEX MY_IDX ON MY_TABLE;
<b>EXPLAIN</b>	Displays an execution plan.	EXPLAIN SELECT name FROM MY_TABLE;
<b>CREATE SEQUENCE</b>	Creates a sequence.	CREATE SEQUENCE MY_SEQUENCE;

Command	Description	Example
<b>DROP SEQUENCE</b>	Deletes a sequence.	DROP SEQUENCE MY_SEQUENCE;
<b>CREATE VIEW</b>	Creates a view.	CREATE VIEW MY_VIEW AS SELECT * FROM MY_TABLE;
<b>DROP VIEW</b>	Deletes a view.	DROP VIEW MY_VIEW;
<b>CREATE SCHEMA</b>	Creates a schema.	CREATE SCHEMA MY_SCHEMA;
<b>USE</b>	Modifies the default schema.	USE MY_SCHEMA;
<b>DROP SCHEMA</b>	Deletes a schema.	DROP SCHEMA MY_SCHEMA;

**Table 2-76** Common functions

Function Type	Function	Description	Example
String functions	<b>SUBSTR</b>	Extracts a part of a string.	SUBSTR('[Hello]', 2, 5)
	<b>INSTR</b>	Queries the position of the first occurrence of a string in another string.	INSTR('Hello World', 'World')
	<b>TRIM</b>	Removes whitespaces from both ends of a string.	TRIM(' Hello ')
	<b>LTRIM</b>	Removes whitespaces found on the left-hand side of the string.	LTRIM(' Hello')
	<b>RTRIM</b>	Removes whitespaces found on the right-hand side of the string.	RTRIM('Hello ')
	<b>LPAD</b>	Left-pads a string with another string.	LPAD('John',30,'*')

Function Type	Function	Description	Example
	<b>LENGTH</b>	Gets the length of a given string.	LENGTH('Hello')
	<b>UPPER</b>	Converts all the characters in a string to uppercase characters.	UPPER('Hello')
	<b>LOWER</b>	Converts all the characters in a string to lowercase characters.	LOWER('HELLO')
	<b>REVERSE</b>	Reverses a string.	REVERSE('Hello')
	<b>REGEXP_SPLIT</b>	Splits a string using a regular expression.	REGEXP_SPLIT('ONE,TWO,THREE', ',')
	<b>REGEXP_REPLACE</b>	Returns a string by applying a regular expression and replacing the matches with the replacement string.	REGEXP_REPLACE('abc123ABC', '[0-9]+', '#')
	<b>REGEXP_SUBSTR</b>	Returns a substring from a string based on a regular expression pattern.	REGEXP_SUBSTR('na1-appsrv35-sj35', '[-]+')
Aggregate functions	<b>AVG</b>	Gets the average value.	AVG(X)
	<b>COUNT</b>	Gets the number of data records.	COUNT(*)
	<b>MAX</b>	Gets the maximum value.	MAX(NAME)
	<b>MIN</b>	Gets the minimum value.	MIN(NAME)
	<b>SUM</b>	Gets the sum of all values.	SUM(X)

Function Type	Function	Description	Example
	<b>STDDEV_POP</b>	Gets the population standard deviation of all values.	STDDEV_POP( X )
	<b>STDDEV_SAMP</b>	Gets the sample standard deviation of all values.	STDDEV_SAMP( X )
	<b>NTH_VALUE</b>	Gets the Nth value in each distinct group ordered according to the <b>ORDER BY</b> specification.	NTH_VALUE( name, 2 ) WITHIN GROUP ( ORDER BY salary DESC )
Time and date functions	<b>NOW</b>	Returns the current date.	NOW()
	<b>CURRENT_TIME</b>	Returns the current time.	CURRENT_TIME()
	<b>CURRENT_DATE</b>	Returns the current date.	CURRENT_DATE()
	<b>TO_DATE</b>	Parses a string and returns a date.	TO_DATE('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIME</b>	Converts the given string into a <b>TIME</b> instance.	TO_TIME('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>TO_TIMESTAMP</b>	Converts the given string into a <b>TIMESTAMP</b> instance.	TO_TIMESTAMP('1970-01-01', 'yyyy-MM-dd', 'GMT+1')
	<b>YEAR</b>	Returns the year of the specified date.	YEAR(TO_DATE('2015-6-05'))
	<b>MONTH</b>	Returns the month of the specified date.	MONTH(TO_TIMESTAMP('2015-6-05'))
	<b>WEEK</b>	Returns the week of the specified date.	WEEK(TO_TIME('2010-6-15'))

Function Type	Function	Description	Example
	<b>HOUR</b>	Returns the hour of the specified date.	HOUR(TO_TIMESTAMP('2015-6-05'))
	<b>MINUTE</b>	Returns the minute of the specified date.	MINUTE(TO_TIME('2015-6-05'))
	<b>SECOND</b>	Returns the second of the specified date.	SECOND(TO_DATE('2015-6-05'))
Numeric functions	<b>ROUND</b>	Rounds the numeric or timestamp expression to the nearest scale or time unit specified.	ROUND(2.56)
	<b>CEIL</b>	Rounds any fractional value up to the next even multiple.	CEIL(2.34)
	<b>FLOOR</b>	Rounds any fractional value down to the previous even multiple.	FLOOR(2.34)
	<b>TRUNC</b>	Rounds any fractional value down to the previous even multiple (same as <b>FLOOR</b> ).	TRUNC(2.34)
	<b>TO_NUMBER</b>	Formats a string as a number.	TO_NUMBER('-123.33')
	<b>RAND</b>	Returns a random number.	RAND()
	<b>ABS</b>	Returns the absolute value of the given numeric expression.	ABS(-1)

Function Type	Function	Description	Example
	<b>SQRT</b>	$\sqrt{x}$	SQRT(1.1)
	<b>EXP</b>	$e^x$	EXP(-1)
	<b>POWER</b>	$x^y$	POWER(2, 3)
	<b>LN</b>	$\ln(x)$	LN(3)
	<b>LOG</b>	$\log_x(y)$	LOG(2, 3)
Array functions	<b>ARRAY_ELEM</b>	Uses the array subscript notation to access an array element.	ARRAY_ELEM(ARRAY[1,2,3], 1)
	<b>ARRAY_APPEND</b>	Appends the given element to the beginning of the array.	ARRAY_APPEND(ARRAY[1,2,3], 4)
	<b>ARRAY_CAT</b>	Concatenates the input arrays and returns the result.	ARRAY_CAT(ARRAY[1,2], ARRAY[3,4])
	<b>ARRAY_FILL</b>	Fills an array with values.	ARRAY_FILL(1, 3)
	<b>ARRAY_TO_STRING</b>	Converts the elements of an array to a string and joins them using the specified delimiter.	ARRAY_TO_STRING(ARRAY['a','b','c'], ',')

Function Type	Function	Description	Example
	<b>ANY</b>	Used on the right-hand side of a comparison expression to test that any array element satisfies the comparison expression against the left-hand side.	10 > ANY(my_array)
	<b>ALL</b>	Used on the right-hand side of a comparison expression to test that all array elements satisfy the comparison expression against the left-hand side.	10 > ALL(my_array)
Other functions	<b>MD5</b>	Computes the MD5 hash of the argument.	MD5(my_column)
	<b>ENCODE</b>	Encodes the expression according to the encoding format provided.	ENCODE(myNumber, 'BASE62')
	<b>DECODE</b>	Decodes the expression according to the encoding format provided.	DECODE('000000008512af277fff fff8', 'HEX')

#### 2.6.5.3.4 JDBC APIs

Phoenix implements most **java.sql** interfaces. The SQL syntax follows the ANSI SQL standard.

For details about the functions that are supported, visit:

<http://phoenix.apache.org/language/functions.html>

For details about the syntax that is supported, visit:

<http://phoenix.apache.org/language/index.html>

### 2.6.5.3.5 WebUI

#### Scenario

The web user interface (WebUI) displays the HBase cluster status, including summary of a cluster and information about RegionServer, Master, snapshots, and running processes. You can determine the HBase cluster status based on the information displayed on the WebUI.

##### NOTE

Contact the administrator to obtain a service account that has the permission to access the WebUI and obtain its password.

#### Procedure

1. **Accessing FusionInsight Manager of an MRS Cluster.** Choose **Cluster > Name of the desired cluster > Services > HBase > HMaster(Active)** to open the HBase WebUI.
2. On the home page of the HBase WebUI, view the HBase summary. The following information is included.
  - a. On the **Region Servers** page, view basic information about the RegionServer, as shown in [Figure 2-137](#).

**Figure 2-137** Region Servers

Region Servers			
Base Stats	Memory	Requests	Storefiles
ServerName	Start time	Requests Per Second	Num. Regions
VM-3,21302,1415117599935	Wed Nov 05 00:13:19 CST 2014	0	1
VM-4,21302,1415117602775	Wed Nov 05 00:13:22 CST 2014	0	3
VM-5,21302,1415117607877	Wed Nov 05 00:13:27 CST 2014	0	0
Total:3		0	4

- b. On the **Backup Masters** page, view information about the Backup Master, as shown in [Figure 2-138](#).

**Figure 2-138** Backup Masters

Backup Masters		
ServerName	Port	Start Time
VM-4	21300	Wed Nov 05 00:13:08 CST 2014
Total:1		

- c. On the **Tables** page, view HBase table information, including **User Tables**, **Catalog Tables**, and **Snapshots**, as shown in [Figure 2-139](#).

**Figure 2-139 Tables**

Tables		
User Tables	Catalog Tables	Snapshots
1 table(s) in set. [Details]		
Table Name	Online Regions	Description
t	1	't', {NAME => 'd'}

- d. On the **Tasks** page, view information about tasks running on HBase, including the start time and status, as shown in [Figure 2-140](#).

**Figure 2-140 Tasks**

Tasks				
Show All Monitored Tasks	Show non-RPC Tasks	Show All RPC Handler Tasks	Show Active RPC Calls	Show Client Operations
<a href="#">View as JSON</a>				
Start Time	Description	State	Status	
Wed Nov 05 03:06:35 CST 2014	Closing region availabilityCheck_VM-5_1415127994683.1d82d088fbf4ba672ee2235fd98ae695.	COMPLETE (since 44sec ago)	Closed (since 44sec ago)	
Wed Nov 05 00:20:13 CST 2014	RpcServer.reader=0,port=21300	WAITING (since 2mins, 36sec ago)	Waiting for a call (since 2mins, 36sec ago)	
Wed Nov 05 00:19:49 CST 2014	RpcServer.reader=9,port=21300	WAITING (since 2mins, 53sec ago)	Waiting for a call (since 2mins, 53sec ago)	
Wed Nov 05 00:19:17 CST 2014	RpcServer.reader=8,port=21300	WAITING (since 3mins, 8sec ago)	Waiting for a call (since 3mins, 8sec ago)	
Wed Nov 05 00:15:10 CST 2014	RpcServer.reader=7,port=21300	WAITING (since 3mins, 46sec ago)	Waiting for a call (since 3mins, 46sec ago)	
Wed Nov 05 00:14:52 CST 2014	RpcServer.reader=6,port=21300	WAITING (since 5mins, 2sec ago)	Waiting for a call (since 5mins, 2sec ago)	
Wed Nov 05 00:14:36 CST 2014	RpcServer.reader=5,port=21300	WAITING (since 10sec ago)	Waiting for a call (since 10sec ago)	
Wed Nov 05 00:14:01 CST 2014	RpcServer.reader=4,port=21300	WAITING (since 0sec ago)	Waiting for a call (since 0sec ago)	

3. On the **Table Details** page of the HBase WebUI, view the summary of HBase storage tables, as shown in [Figure 2-141](#).

**Figure 2-141 Table Details**

User Tables	
1 table(s) in set.	
Table	Description
t	't', {NAME => 'd', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}

4. On the **Debug dump** page of the HBase WebUI, view the HBase debug information, as shown in [Figure 2-142](#).

**Figure 2-142** Debug dump

- On the **HBase Configuration** page of the HBase WebUI, view the HBase control information, as shown in [Figure 2-143](#).

**Figure 2-143 HBase Configuration**

```
<?xml version="1.0" encoding="UTF-8"?>
- <configuration>
  - <property>
    <name>dfs.journalnode.rpc-address</name>
    <value>0.0.0.0:8485</value>
    <source>hdfs-default.xml</source>
  </property>
  - <property>
    <name>io.storefile.bloom.block.size</name>
    <value>131072</value>
    <source>hbase-default.xml</source>
  </property>
  - <property>
    <name>hbase.sessioncontrol.maxSessions</name>
    <value>65535</value>
    <source>hbase-site.xml</source>
  </property>
  - <property>
    <name>yarn.ipc.rpc.class</name>
    <value>org.apache.hadoop.yarn.ipc.HadoopYarnProtoRPC</value>
    <source>yarn-default.xml</source>
  </property>
  - <property>
    <name>mapreduce.job.maxtaskfailures.per.tracker</name>
    <value>3</value>
    <source>mapred-default.xml</source>
  </property>
  - <property>
    <name>hbase.rest.threads.min</name>
    <value>2</value>
    <source>hbase-default.xml</source>
  </property>
  - <property>
    <name>hbase.rs.cacheblocksonwrite</name>
    <value>false</value>
    <source>hbase-default.xml</source>
  </property>
  - <property>
    <name>ha.health-monitor.connect-retry-interval.ms</name>
    <value>1000</value>
    <source>core-default.xml</source>
  </property>
  - <property>
    <name>yarn.resourcemanager.work-preserving-recovery.enabled</name>
    <value>false</value>
    <source>yarn-default.xml</source>
  </property>
  - <property>
    <name>dfs.client.mmap.cache.size</name>
    <value>256</value>
    <source>hdfs-default.xml</source>
  </property>
```

#### 2.6.5.4 Phoenix Command Line

Phoenix supports SQL statements to operate HBase. The following describes how to use SQL statements to create tables, insert data, query data, and delete tables.

#### Prerequisites

The HBase client has been installed. For details, see [Installing a Client](#). For example, the client has been installed in the `/opt/hadoopclient` directory. The client directory in the following operations is only an example. Change it based on the actual installation directory onsite. Before using the client, download and update the client configuration file, and ensure that the active management node of Manager is available.

## Procedure

**Step 1 Optional:** Log in to the node where the client is installed as the client installation user.

Access the HBase client installation directory:

```
cd /opt/hadoopclient
```

**Step 2** Run the following command to configure environment variables:

```
source bigdata_env
```

**Step 3** If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission to create HBase tables. If Kerberos authentication is disabled for the current cluster, skip this step:

```
kinit MRS cluster user
```

For example, **kinit hbaseuser**.

**Step 4** Running Commands on the Phoenix Client.

```
sqlline.py
```

**Step 5** Create a table:

```
CREATE TABLE TEST (id VARCHAR PRIMARY KEY, name VARCHAR);
```

**Step 6** Insert data:

```
UPSERT INTO TEST(id,name) VALUES ('1','jamee');
```

**Step 7** Query data:

```
SELECT * FROM TEST;
```

**Step 8** Deleting a table:

```
DROP TABLE TEST;
```

**Step 9** Exit the Phoenix CLI:

```
!quit
```

```
----End
```

### 2.6.5.5 FAQs

#### 2.6.5.5.1 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory" Is Displayed in the Error Information?

**Step 1** Check whether the **hbase.rpc.controllerfactory.class** configuration item is included in the **hbase-site.xml** configuration file.

```
<name>hbase.rpc.controllerfactory.class</name>
<value>org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory</value>
```

**Step 2** If this configuration item is included in the current application development project, you need to import the **phoenix-core-\*jar** package. You can obtain this package from **HBase/hbase/lib** in the HBase client installation directory.

**Step 3** If you do not import this JAR package, you need to delete **hbase.rpc.controllerfactory.class** from the **hbase-site.xml** configuration file.

----End

### 2.6.5.5.2 What Are the Application Scenarios of the bulkload and put Data-loading Modes?

#### Question

Both the bulkload and put data-loading modes can be used to load data to HBase. Though the bulkload mode loads data faster than the put mode, the bulkload mode has its own disadvantages. The following describes the application scenarios of these two data-loading modes.

#### Answer

The bulkload starts MapReduce tasks to generate HFile files, and then registers HFile files with HBase. Incorrect use of the bulkload mode will consume more cluster memory and CPU resources due to started MapReduce tasks. The large number of HFile files may frequently trigger Compaction, decreasing the query speed drastically.

Incorrect use of the put mode may cause a slow data loading rate. If the memory allocated to RegionServer is not sufficient, the process may exit.

The application scenarios of the bulkload and put modes are as follows:

- bulkload:
  - Load a large amount of data to HBase in the one-off manner.
  - Load data to HBase with low reliability requirements and without generating WAL files.
  - Low loading and query speed if the put mode is used.
  - The size of the HFile generated after data loading is similar to the size of HDFS block.
- put:
  - The size of the data loaded to one Region at a time is smaller than half the size of an HDFS block.
  - Load data to HBase in real time.
  - The query speed does not decrease wildly during data loading.

### 2.6.5.5.3 An Error Occurred When Building a JAR Package

#### Question

When the sample code is compiled using Maven to build a JAR package, the error message "Could not transfer artifact org.apache.commons:commons-crypto:pom:\${commons-crypto.version}" is displayed and the package building fails.

#### Answer

The hbase-common module depends on commons-crypto. In the **pom.xml** file of hbase-common, the **\${commons-crypto.version}** variable is used to introduce

commons-crypto. The parsing logic of this variable is as follows: If the OS is AArch64, the value is **1.0.0-hw-aarch64**. If the OS is x86\_64, the value is **1.0.0**. If Maven fails to parse the variable using the OS due to incorrect configuration in the compilation environment, you can manually modify the **pom.xml** file to prevent correct compilation.

In the **pom.xml** file, manually change the dependency of the hbase-common module to the following to exclude the commons-crypto dependency:

```
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-common</artifactId>
    <version>${hbase.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-crypto</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Manually add the commons-crypto dependency of the specified version. Enter a correct version based on the OS architecture (x86\_64 or AArch64).

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-crypto</artifactId>
    <version>1.0.0</version>
</dependency>
```

#### 2.6.5.5.4 How to Rectify the Fault When an Exception Occurs During the Running of an HBase-developed Application and "java.lang.OutOfMemoryError: Direct buffer memory" Is Displayed in the Error Information?

**Step 1** Run the **java -version** command to check whether the JDK version is JDK1.8u102 or later.

- If yes, go to [Step 2](#).
- If no, the client for secondary development supports only the built-in OpenJDK 1.8 (jdk-8u181-linux-x64). If the JDK version is not JDK1.8u102 or later, upgrade it to JDK1.8u102 or later, and then go to [Step 2](#).

**Step 2** When running the program, set the JVM parameter - **Djdk.nio.maxCachedBufferSize=262144** to limit the size of the direct memory used by each thread.

----End

## 2.7 HDFS Development Guide

### 2.7.1 Overview

### 2.7.1.1 Introduction to HDFS

#### Introduction to HDFS

Hadoop distribute file system (HDFS) is a distributed file system with high fault tolerance. HDFS supports data access with high throughput and applies to processing of large data sets.

HDFS applies to the following application scenarios:

- Massive data processing (higher than the TB or PB level).
- Scenarios that require high throughput.
- Scenarios that require high reliability.
- Scenarios that require good scalability.

#### Introduction to HDFS Interface

HDFS can be developed by using Java language. For details of API reference, see [Java API](#).

### 2.7.1.2 Basic Concepts

#### DataNode

A DataNode is used to store data blocks of each file and periodically report the storage status to the NameNode.

#### NameNode

A NameNode is used to manage the namespace, directory structure, and metadata information of a file system and provide the backup mechanism. NameNodes are classified into the following two types:

- Active NameNode: manages the file system namespace, maintains the directory structure tree and metadata information of a file system, and records the relationship between each data block and the file to which the data block belongs.
- Standby NameNode: Data in a standby NameNode is synchronous with those in an active NameNode. A standby NameNode takes over services from the active NameNode if the active NameNode is exception.

#### JournalNode

A JournalNode synchronizes metadata between the active and standby NameNodes in the High Availability (HA) cluster.

#### ZKFC

ZKFC must be deployed for each NameNode. It is responsible for monitoring NameNode status and writing status information to the ZooKeeper. ZKFC also has permission to select the active NameNode.

## Colocation

Colocation is used to store associated data or the data to be associated on the same storage node. The HDFS Colocation stores files to be associated on a same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

## Federation

Federation is adopted to solve most problems in the HDFS architecture that allows only a single NameNode.

Federation allows multiple NameServices. The NameServices can be configured in the HA mode. In HA mode, a hot standby NameNode is provided for the active NameNode to address the single node failure. The standby NameNode is able to quickly take over services from the active NameNode when the active NameNode breaks down. The administrator can start the failover during maintenance.



This function applies only to MRS physical machine clusters.

## Client

The HDFS can be accessed from the Java application programming interface (API), C API, Shell, HTTP REST API and web user interface (WebUI).

For details, see [Common API Introduction](#) and [Shell Command Introduce](#).

- JAVA API  
Provides an application interface for the HDFS. This guide describes how to use the Java API to develop HDFS applications.
- C API  
Provides an application interface for the HDFS. This guide describes how to use the C API to develop HDFS applications.
- Shell  
Provides shell commands to perform operations on the HDFS.
- HTTP REST API  
Additional interfaces except Shell, Java API and C API. You can use the interfaces to monitor HDFS status.
- WEB UI  
Provides a visualized management web page.

### 2.7.1.3 Development Process

All stages of the development process are shown and described in [Figure 2-144](#) and [Table 2-77](#).

Figure 2-144 HDFS development process

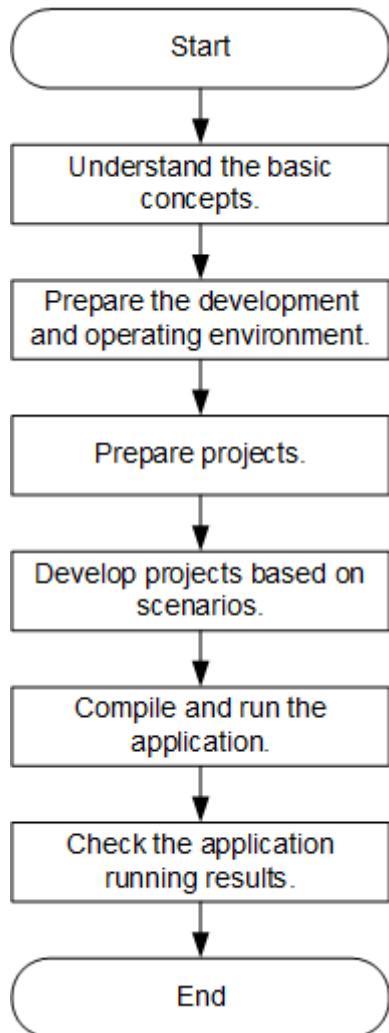


Table 2-77 Description of HDFS development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of HDFS are required to be understood.	<a href="#">Basic Concepts</a>
Prepare the development and operating environment.	Use IntelliJ IDEA and configure the development environment based on the reference. The running environment of HDFS is the HDFS client. Install and configure the client based on the reference.	<a href="#">Development and Operating Environment</a>
Prepare projects.	HDFS provides sample projects in various scenarios. Sample projects can be imported for studying.	<a href="#">Configuring and Importing Sample Projects</a>

Stage	Description	Reference
Develop projects based on scenarios.	Provide the sample project. This helps users to learn about the programming interfaces of all HDFS components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Commissioning the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Commissioning the Application</a>

## 2.7.2 Environment Preparation

### 2.7.2.1 Development and Operating Environment

#### Preparing Development Environment

[Table 2-78](#) describes the environment required for application development.

**Table 2-78** Development Environment

Preparation	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS.</li></ul> <p>If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</p>

Preparation	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <code>com.ibm.jsse2.overrideDefaultTLS</code> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. **Accessing FusionInsight Manager of an MRS Cluster.** In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HDFS\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the HDFS sample project.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster.** Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/HDFS/config** directory and save them to the **conf** folder of the project code.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path

is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp HDFS/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

**Table 2-79** Configuration files

File	Function
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

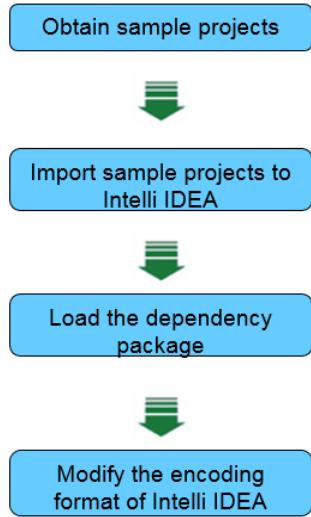
### 2.7.2.2 Configuring and Importing Sample Projects

#### Scenario

HDFS provides sample projects for multiple scenarios to help you quickly learn HDFS projects.

The procedure for importing HDFS example codes is described as follows: [Figure 2-145](#) shows the procedure.

Figure 2-145 Sample project importing procedure



## Procedure

**Step 1** Obtain the sample project folder **hdfs-example-normal** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

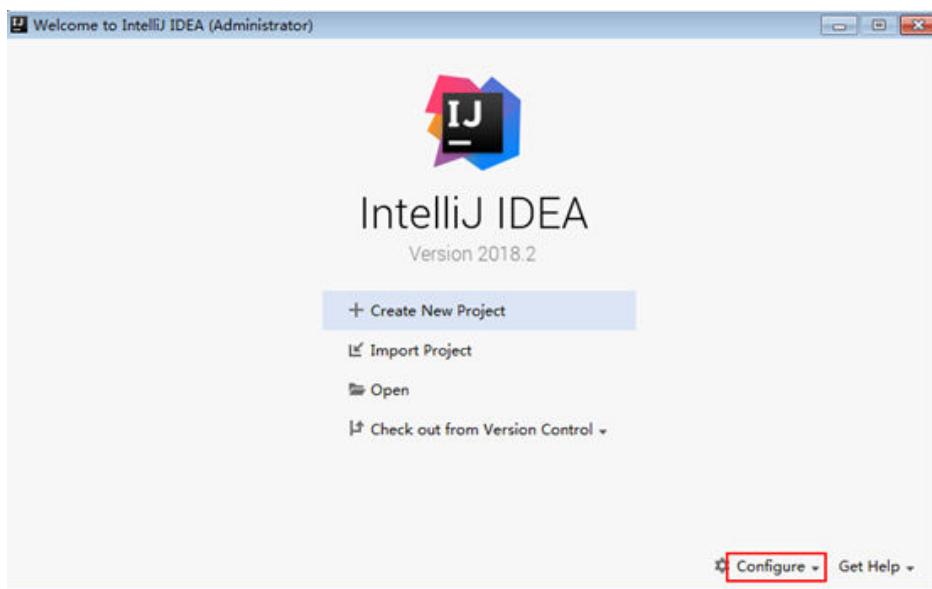
**Step 2** Import the example project to the IntelliJ IDEA development environment.

1. Open IntelliJ IDEA and choose **File > Open**.
2. Choose the directory of the example project **hdfs-example-normal**. Click **OK**.

**Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.

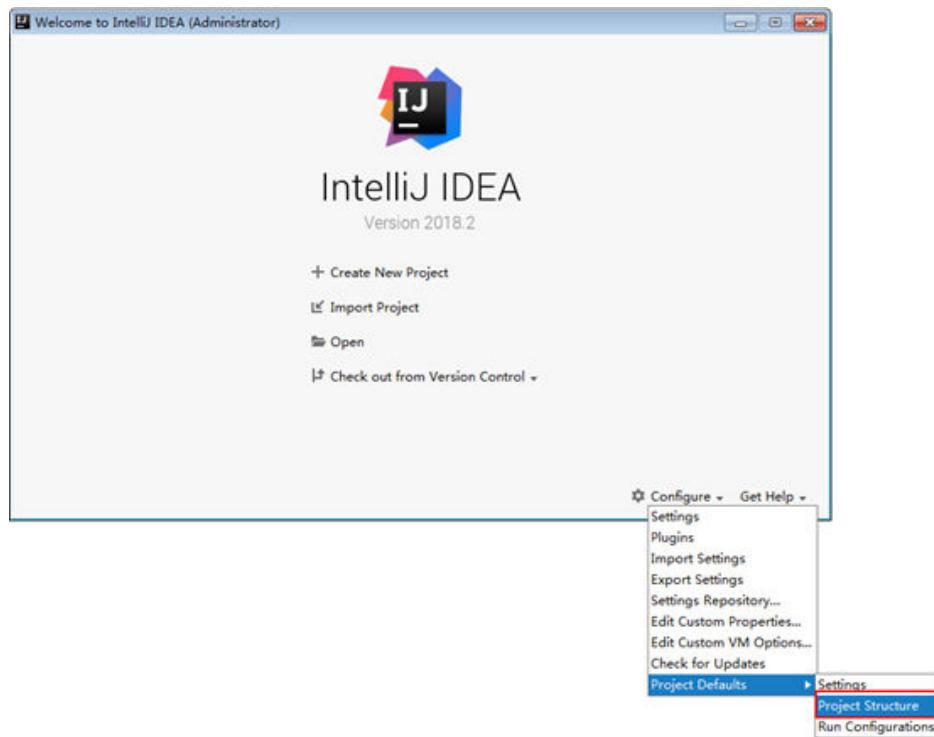
1. Open IntelliJ IDEA and click **Configure**.

Figure 2-146 Quick Start



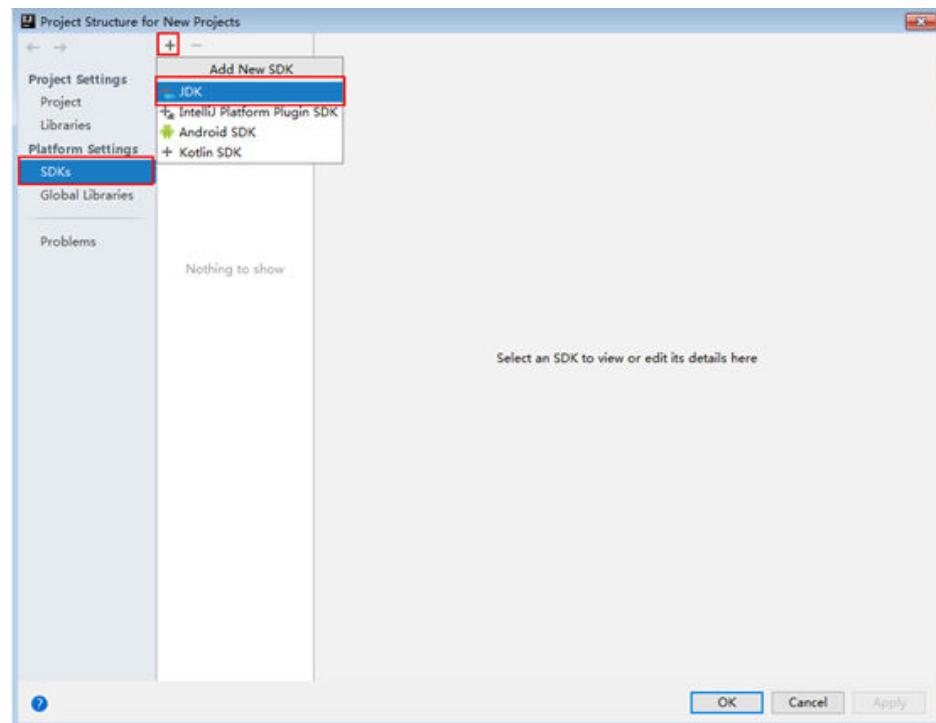
2. Choose **Project Defaults > Project Structure**.

**Figure 2-147** Configure



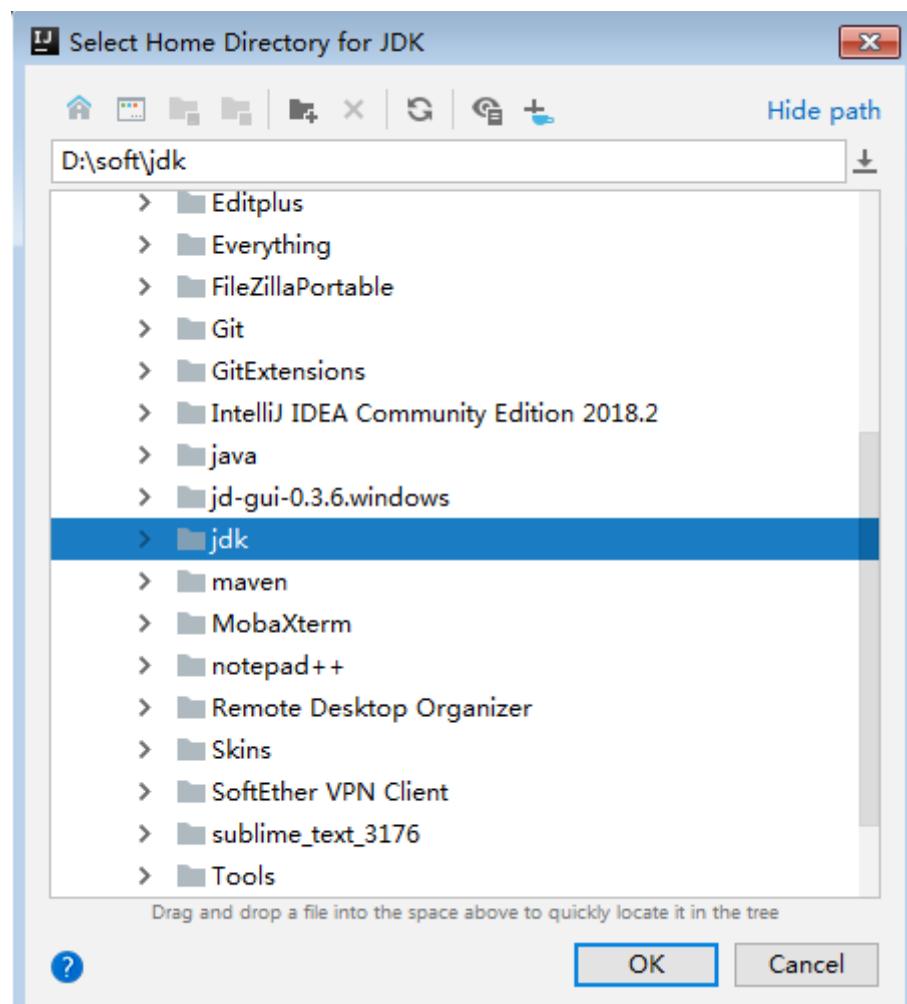
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 2-148 Project Structure for New Projects



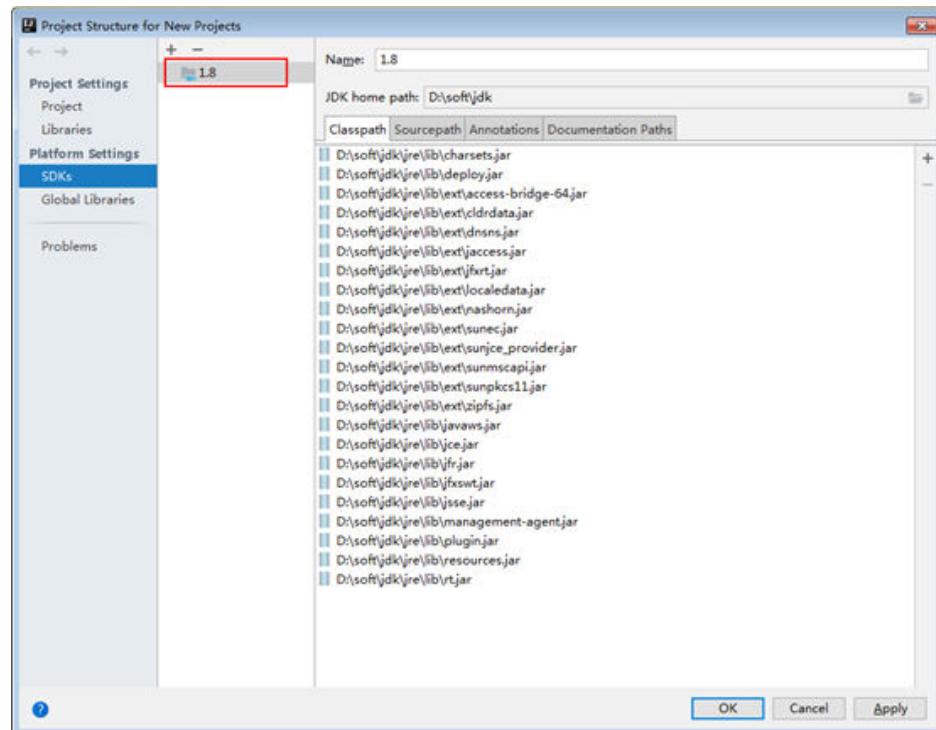
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 2-149 Select Home Directory for JDK



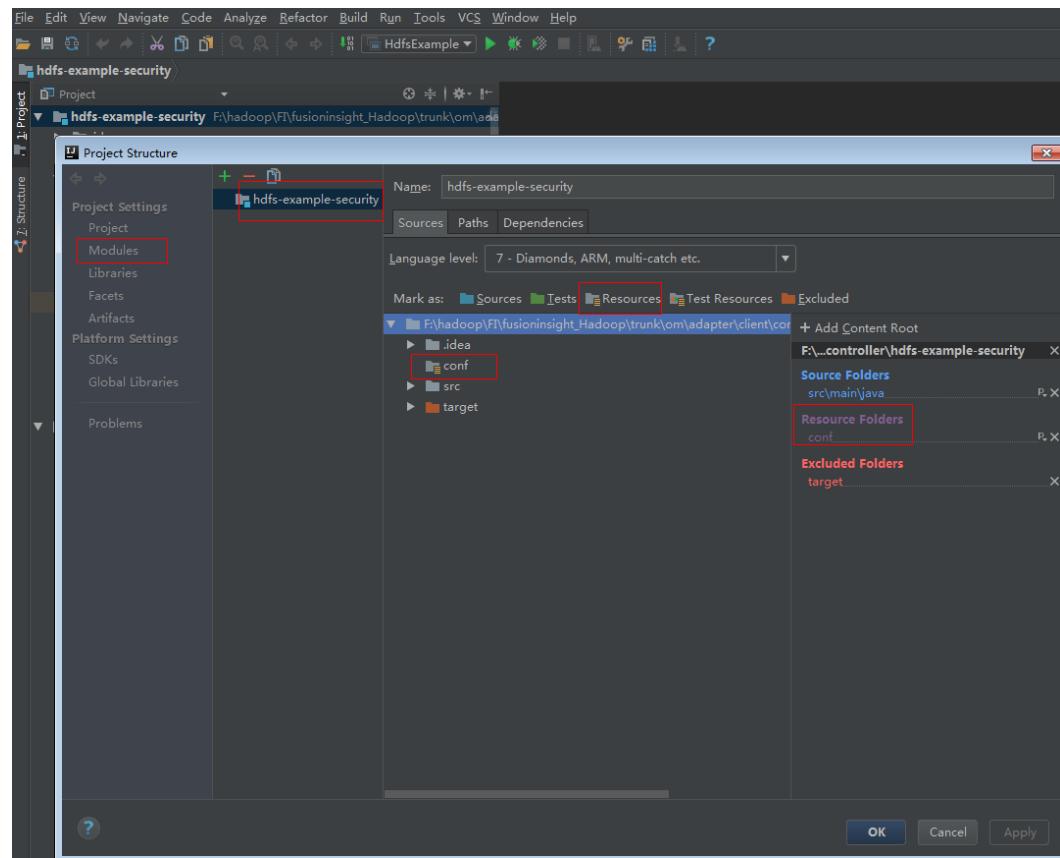
5. Click **OK**.

Figure 2-150 Completing the JDK configuration



**Step 4** Add the conf directory in the project to the resource path. On the menu bar of the IntelliJ IDEA, choose **File > Project Structure**. In the dialog box that is displayed, select the current project and click **Resources > conf > OK** to set the resource directory. The figure shows the directory for setting the project resource.

Figure 2-151 Setting the Project Resource Directory



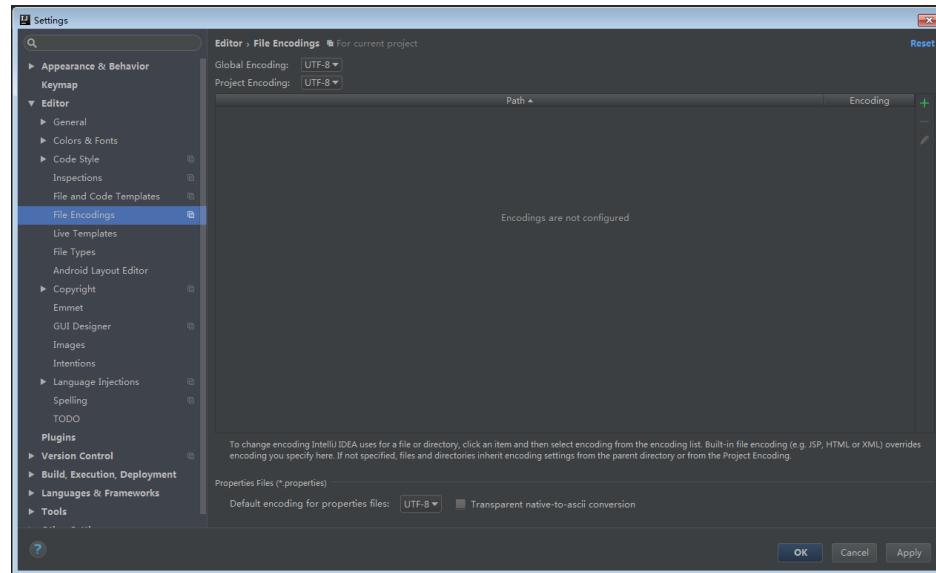
**Step 5** Add the related jar files the example project depending on into classpath.

If the sample project code is obtained from an open-source image site, dependency JAR files are automatically downloaded after Maven is configured (for details, see [Configuring Huawei Open-Source Mirrors](#)).

**Step 6** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.
2. Choose **Editor > File Encodings** from the navigation tree of the **Settings** window. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 2-152](#)

Figure 2-152 Setting the IntelliJ IDEA coding format



----End

## 2.7.3 Developing the Project

### 2.7.3.1 Scenario

#### Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the HDFS development process and know important interface functions.

#### Scenario

Service operation objects of HDFS are files. File operations in example codes include creating a folder, writing data into a file, appending data to a file, reading data from a file, and deleting a file or folder. HDFS also supports other services including setting file permission. You can learn how to perform other operations on HDFS after learning the example codes in this chapter.

The example codes are described in the following order:

1. Initializing the HDFS.
2. Creating directories.
3. Writing data into a file.
4. Appending data to a file.
5. Reading data from a file.
6. Deleting a file.
7. Deleting directories.
8. Multi-thread tasks
9. Setting storage policies.

10. Colocation.

### 2.7.3.2 Development Idea

#### Development Idea

According to the previous scenario description, the following provides the basic operations for HDFS files with read, write, and delete operations on the **/user/hdfs-examples/test.txt** file as an example:

1. Create a FileSystem object: fSystem.
2. Call the mkdir interface in fSystem to create a directory.
3. Call the create interface in fSystem to create an FSDataOutputStream object: out. Use the write method to write data into the object out.
4. Call the append interface in fSystem to create an FSDataOutputStream object: out. Use the write method to append data into the object out.
5. Call the open interface in fSystem to create an FSDataInputStream object: in. Use the read method to read files of the object in.
6. Call the delete interface in fSystem to delete the file.
7. Call the delete interface in fSystem to delete the folder.

### 2.7.3.3 Declare the Example Codes

#### 2.7.3.3.1 Initializing the HDFS

##### Function

Hadoop distributed file system (HDFS) initialization is a prerequisite for using application programming interfaces (APIs) provided by the HDFS. The process of initializing the HDFS is

1. Load the HDFS service configuration file.
2. Instantiate a FileSystem.

#### Configuration File Description

**Table 2-80** lists the configuration files to be used during the login to the HDFS. These files already imported to the **conf** directory of the **hadoop-examples** project.

**Table 2-80** Configuration files

File	Function
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.

#### NOTE

The **log4j.properties** file under the **conf** directory can be configured based on your needs.

### Example Codes

The following is code snippets. For complete codes, see the HdfsExample class in com.huawei.bigdata.hdfs.examples.

The initialization codes used when applications are run in Linux and the codes used when applications are run in Windows are the same. The example codes are as follows:

```
//initialization
/**confLoad();

// Creating a sample project
HdfsExample hdfs_examples = new HdfsExample("/user/hdfs-examples", "test.txt");
/**
*
* If the application is running in the Linux OS, the path of core-site.xml, hdfs-site.xml must be modified to
the absolute path of the client file in the Linux OS.
*
*/
private static void confLoad() throws IOException {
    conf = new Configuration();
    // conf file
    conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));
    conf.addResource(new Path(PATH_TO_CORE_SITE_XML));
}

/**
*Create a sample project.
*/
public HdfsExample(String path, String fileName) throws IOException {
    this.DEST_PATH = path;
    this.FILE_NAME = fileName;
    instanceBuild();
}
private void instanceBuild() throws IOException {
    fSystem = FileSystem.get(conf);
}
```

#### NOTE

- (Optional) Specify a user to run the example code. To run the example code related to the Colocation operation, the user must be a member of the supergroup group. The following describes two ways to specify the user who runs the example code:

Add the environment variable **HADOOP\_USER\_NAME**: For operation details in a Windows-based environment, see [Step 1](#) in section **Compiling and Running an Application** For operation details in a Linux-based environment, see [Step 4](#) in section **Compiling and Running an Application with the Client Installed** or [Step 4](#).in section **Compiling and Running an Application with the Client Not Installed**.

Modify the code: If **HADOOP\_USER\_NAME** is not specified, modify "USER" in the code to the actual user name:

```
System.setProperty("HADOOP_USER_NAME", USER);
```

### 2.7.3.3.2 Creating Directories

#### Function

Process of creating a directory:

1. Call the exists method of the FileSystem instance to check whether the directory exists.
2. If yes, the method stops.
3. If no, call the mkdirs method in the FileSystem instance to create a directory.

## Example Codes

The following is a code snippet. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
/**  
 * Create a directory.  
 *  
 * @throws java.io.IOException  
 */  
private void mkdir() throws IOException {  
    Path destPath = new Path(DEST_PATH);  
    if (!createPath(destPath)) {  
        LOG.error("failed to create destPath " + DEST_PATH);  
        return;  
    }  
  
    LOG.info("success to create path " + DEST_PATH);  
}  
  
/**  
 * create file path  
 *  
 * @param filePath  
 * @return  
 * @throws java.io.IOException  
 */  
private boolean createPath(final Path filePath) throws IOException {  
    if (!fSystem.exists(filePath)) {  
        fSystem.mkdirs(filePath);  
    }  
    return true;  
}
```

### 2.7.3.3.3 Writing Data into a File

## Function

The process of writing data into a file is

1. Use the create method in the FileSystem instance to obtain the output stream of writing files.
2. Uses this data stream to write content into a specified file in the HDFS.



Close all requested resources after writing files.

## Example Codes

The following is code snippets. For complete codes, see HdfsMain and HdfsWriter classes in **com.huawei.bigdata.hdfs.examples**.

```
/**  
 * Write a file, write the data  
 *
```

```
* @throws IOException
* @throws ParameterException
*/
private void write() throws IOException {
    final String content = "hi, I am bigdata. It is successful if you can see me.";
    FSDatOutputStream out = null;
    try {
        out = fSystem.create(new Path(DEST_PATH + File.separator + FILE_NAME));
        out.write(content.getBytes());
        out.hsync();
        LOG.info("success to write.");
    } finally {
        // make sure the stream is closed finally.
        IOUtils.closeStream(out);
    }
}
```

### 2.7.3.3.4 Appending Data to a File

## Function

Append data to a specified file in the Hadoop distributed file system (HDFS). The process of appending data to a file is

1. Use the append method in the FileSystem instance to obtain the output stream of the appended data.
2. Use the output stream to add the content to be appended behind the specified file in the HDFS.



Close all requested resources after appending data.

## Example Codes

The following is code snippets. For complete codes, see HdfsMain and HdfsWriter classes in **com.huawei.bigdata.hdfs.examples**.

```
 /**
 * Append data to a file.
 *
 * @throws IOException
 */
private void append() throws IOException {
    final String content = "I append this content.";
    FSDatOutputStream out = null;
    try {
        out = fSystem.append(new Path(DEST_PATH + File.separator + FILE_NAME));
        out.write(content.getBytes());
        out.hsync();
        LOG.info("success to append.");
    } finally {
        // make sure the stream is closed finally.
        IOUtils.closeStream(out);
    }
}
```

### 2.7.3.3.5 Reading Data from a File

## Function

Read data from a specified file in the Hadoop distributed file system (HDFS). The process is:

1. Use the open method in the FileSystem instance to obtain the input stream of writing files.
2. Use the input stream to read the content of the specified file in the HDFS.

 NOTE

Close all requested resources after reading files.

## Example Codes

The following is code snippets. For complete codes, see the `HdfsMain` class in `com.huawei.bigdata.hdfs.examples`.

```
/*
 * Read a file.
 *
 * @throws IOException
 */
private void read() throws IOException {
    String strPath = DEST_PATH + File.separator + FILE_NAME;
    Path path = new Path(strPath);
    FSDataInputStream in = null;
    BufferedReader reader = null;
    StringBuffer strBuffer = new StringBuffer();
    try {
        in = fSystem.open(path);
        reader = new BufferedReader(new InputStreamReader(in));
        String sTempOneLine;
        // write file
        while ((sTempOneLine = reader.readLine()) != null) {
            strBuffer.append(sTempOneLine);
        }
        LOG.info("result is : " + strBuffer.toString());
        LOG.info("success to read.");
    } finally {
        // make sure the streams are closed finally.
        IOUtils.closeStream(reader);
        IOUtils.closeStream(in);
    }
}
```

### 2.7.3.3.6 Deleting a File

## Function

Delete a file from the Hadoop distributed file system (HDFS).

 NOTE

Exercise caution when you delete files because the deletion is irreversible.

## Example Codes

The following is code snippets. For complete codes, see the `HdfsMain` class in `com.huawei.bigdata.hdfs.examples`.

```
/*
 * Delete a file.
 *
 * @throws IOException
 */
private void delete() throws IOException {
    Path beDeletedPath = new Path(DEST_PATH + File.separator + FILE_NAME);
```

```
if (fSystem.delete(beDeletedPath, true)) {
    LOG.info("success to delete the file " + DEST_PATH + File.separator + FILE_NAME);
} else {
    LOG.warn("failed to delete the file " + DEST_PATH + File.separator + FILE_NAME);
}
```

### 2.7.3.3.7 Deleting Directories

## Function

Delete a specified directory from the HDFS.



Files in the deleted directory will be stored in the **Trash/Current** folder of the directory of the current user. In the event of mis-deletion, you can restore the folder from the directory.

## Example Codes

The following is a code snippet of file deletion. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
/*
 * delete the directory
 *
 * @throws java.io.IOException
 */
private void rmdir() throws IOException {
    Path destPath = new Path(DEST_PATH);
    if (!deletePath(destPath)) {
        LOG.error("failed to delete destPath " + DEST_PATH);
        return;
    }
    LOG.info("success to delete path " + DEST_PATH);
}

/*
 * delete file path
 *
 * @param filePath
 * @return
 * @throws java.io.IOException
 */
private boolean deletePath(final Path filePath) throws IOException {
    if (!fSystem.exists(filePath)) {
        return false;
    }
    // fSystem.delete(filePath, true);
    return fSystem.delete(filePath, true);
}
```

### 2.7.3.3.8 Multi-Thread Tasks

## Function

Create multi-threaded tasks and initiate multiple instances to perform file operations.

## Example Codes

The following is a code snippet of file deletion. For complete codes, see the **HdfsExample** class in **com.huawei.bigdata.hdfs.examples**.

```
// Service example 2: multi-thread tasks
final int THREAD_COUNT = 2;
for (int threadNum = 0; threadNum < THREAD_COUNT; threadNum++) {
    HdfsExampleThread example_thread = new HdfsExampleThread("hdfs_example_" + threadNum);
    example_thread.start();
}

class HdfsExampleThread extends Thread {
    private final static Log LOG = LogFactory.getLog(HdfsExampleThread.class.getName());
    /**
     *
     * @param threadName
     */
    public HdfsExampleThread(String threadName) {
        super(threadName);
    }
    public void run() {
        HdfsExample example;
        try {
            example = new HdfsExample("/user/hdfs-examples/" + getName(), "test.txt");
            example.test();
        } catch (IOException e) {
            LOG.error(e);
        }
    }
}
```

The **example.test()** method is the operation on files. The code is as follows:

```
/*
 * HDFS operation instance
 *
 * @throws IOException
 * @throws ParameterException
 *
 * @throws Exception
 */
public void test() throws IOException {
    // Create a directory.
    mkdir();

    // Write data into a file.
    write();

    // Append data to a file.
    append();

    // Read a file.
    read();

    // Delete a file.
    delete();

    // Delete a directory.
    rmdir();
}
```

### 2.7.3.3.9 Setting Storage Policies

#### Function

Specify storage policies for a file or folder in the HDFS.

## Example Code

1. **Accessing FusionInsight Manager of an MRS Cluster**, choose **Cluster > Name of the desired cluster > Services > HDFS > Configurations > All Configurations**.
2. Check whether the value of **dfs.storage.policy.enabled** is the default value **true**. If not, modify the value to **true**, click **Save**, and restart HDFS.
3. Check the code.

The following code segment is only an example. For details, see the **HdfsMain** class in **com.huawei.bigdata.hdfs.examples**.

```
/*
 * set storage policy to path
 * @param policyName
 * Policy Name can be accepted:
 * <li>HOT
 * <li>WARM
 * <li>COLD
 * <li>LAZY_PERSIST
 * <li>ALL_SSD
 * <li>ONE_SSD
 * @throws IOException
 */

private void setStoragePolicy(String policyName) throws IOException {
    if (fSystem instanceof DistributedFileSystem) {
        DistributedFileSystem dfs = (DistributedFileSystem) fSystem;
        Path destPath = new Path(DEST_PATH);
        Boolean flag = false;
        mkdir();
        BlockStoragePolicySpi[] storage = dfs.getStoragePolicies();
        for (BlockStoragePolicySpi bs : storage) {
            if (bs.getName().equals(policyName)) {
                flag = true;
            }
            LOG.info("StoragePolicy:" + bs.getName());
        }
        if (!flag) {
            policyName = storage[0].getName();
        }
        dfs.setStoragePolicy(destPath, policyName);
        LOG.info("success to set Storage Policy path " + DEST_PATH);
        rmdir();
    } else {
        LOG.info("SmallFile not support to set Storage Policy !!!");
    }
}
```

### 2.7.3.3.10 Colocation

#### Function Description

Colocation is to store associated data or data on which associated operations are performed on the same storage node. The HDFS Colocation feature stores files on which associated operations are performed on the same data node so that data can be obtained from the same data node during associated operations. This greatly reduces network bandwidth consumption.

Before using the Colocation function, you are advised to be familiar with the internal mechanisms of Colocation, including:

#### Colocation node allocation principle

## Capacity expansion and Colocation allocation

### Colocation and data node capacity

- **Colocation node allocation principle**

Colocation allocates data nodes to locators evenly according to the allocation node quantity.

 NOTE

The allocation algorithm principle is as follows: Colocation queries all locators, reads the data nodes allocated to the locators, and records the number of times. Based on the number of times, Colocation sorts the data nodes. The data nodes that are rarely used are placed at the beginning and selected first. The count increase by 1 each time after a node is selected. The nodes are shorted again, and the subsequent node will be selected.

- **Capacity expansion and Colocation allocation**

After cluster capacity expansion, you can select one of the following two policies shown in [Table 2-81](#) to balance the usage of data nodes and ensure that the allocation frequency of the newly added nodes is consistent with that of the old data nodes.

**Table 2-81** Allocation policies

No.	Policy	Description
1	Delete the original locators and create new locators for all data nodes in the cluster.	<ol style="list-style-type: none"><li>1. The original locator before the capacity expansion evenly uses all data nodes. After the capacity expansion, the newly added nodes are not allocated to existing locators, so Colocation stores data only to the old data nodes.</li><li>2. Data nodes are allocated to specific locators. Therefore, after capacity expansion, Colocation needs to reallocated data nodes to locators.</li></ol>
2	Create new locators and plan the data storage mode again.	The old locators use the old data nodes, while the newly created locators mainly use the new data nodes. Therefore, locators need to be planned again based on the actual service requirements on data.

 NOTE

Generally, you are advised to use the policy to reallocate data nodes to locators after capacity expansion to prevent data from being stored only to the new data nodes.

- **Colocation and data node capacity**

When Colocation is used to store data, the data is stored to the data node of a specified locator. If no locator planning is performed, the data node capacity will be uneven. [Table 2-82](#) summarizes the two usage principles to ensure even data node capacity.

**Table 2-82 Usage Principle**

No.	Usage Principle	Description
1	All the data nodes are used in the same frequency in locators.	Assume that there are N data nodes, the number of locators must be an integral multiple of N (N, 2 N, ...).
2	A proper data storage plan must be made for all locators to ensure that data is evenly stored in the locators.	None

During HDFS secondary development, you can obtain the DFSColocatinAdmin and DFSColocatinClient instances to create groups, delete groups, write files, and delete files in or from the location.

**BOOK NOTE**

- When the Colocation function is enabled and users specify DataNodes, the data volume will be large on some nodes. Serious data skew will result in HDFS data write failures.
- Because of data skew, MapReduce accesses only several nodes. In this case, the load is heavy on these nodes, while other nodes are idle.
- For a single application task, the DFSColocatinAdmin and DFSColocatinClient instances can be used only once. If the instances are used for many times, excessive HDFS links will be created and use up HDFS resources.
- If you need to perform the balance operation for a file uploaded by colocation, you can set the `oi.dfs.colocation.file.pattern` parameter on FusionInsight Manager to the file path to avoid invalid colocation. If there are multiple files, use commas (,) to separate the file paths, for example, `/test1,/test2`.
- Colocation stores associated data or data on which associated operations are performed on the same storage. When Balancer- or Mover-related operations are performed, data blocks will be moved. As a result, the Colocation function becomes unavailable. Therefore, when using the Colocation function, you are advised to set the HDFS configuration item `dfs.datanode.block-pinning.enabled` to `true`. In this case, files written by Colocation will not be moved when Balancer- or Mover-related operations are performed in the cluster, ensuring file colocation.

## Example Codes

For complete example codes, see  
`com.huawei.bigdata.hdfs.examples.ColocationExample`.

**BOOK NOTE**

- Specify a user to run a Colocation project. This user must be a member of the supergroup group.
- When the Colocation project is run, the HDFS parameter `fs.defaultFS` cannot be set to `viewfs://ClusterX`.

### 1. Initialization

The user to run the Colocation project must be specified before the running.

```
private static void init() throws IOException {
    // Set the user. If HADOOP_USER_NAME is not set for the use, use USER.
    if (System.getenv("HADOOP_USER_NAME") == null &&
```

```
System.getProperty("HADOOP_USER_NAME") == null) {  
    System.setProperty("HADOOP_USER_NAME", USER);  
}  
}
```

2. Obtain instances.

Example: Colocation operations require the DFSColocatinAdmin and DFSColocatinClient instances. Therefore, the instances must be obtained before operations, such as creating a group.

```
dfsAdmin = new DFSColocatinAdmin(conf);  
dfs = new DFSColocatinClient();  
dfs.initialize(URI.create(conf.get("fs.defaultFS")), conf);
```

3. Create a group.

Example: Create a gid01 group, which contains three locators.

```
/**  
 * create group  
 *  
 * @throws java.io.IOException  
 */  
private static void createGroup() throws IOException {  
    dfsAdmin.createColocationGroup(COLOCATION_GROUP_GROUP01,  
        Arrays.asList(new String[] { "lid01", "lid02", "lid03" }));  
}
```

4. Write data into a file. The related group must be created before writing data into the file.

Example: Write data into the testfile.txt file.

```
/**  
 * create and write file  
 *  
 * @throws java.io.IOException  
 */  
private static void put() throws IOException {  
    FSDatOutputStream out = dfs.create(new Path(TESTFILE_TXT), true,  
    COLOCATION_GROUP_GROUP01, "lid01");  
    // the data to be written to the hdfs.  
    byte[] readBuf = "Hello World".getBytes("UTF-8");  
    out.write(readBuf, 0, readBuf.length);  
    out.close();  
}
```

5. Delete a file.

Example: Delete the testfile.txt file.

```
/**  
 * delete file  
 *  
 * @throws java.io.IOException  
 */  
@SuppressWarnings("deprecation")  
private static void delete() throws IOException {  
    dfs.delete(new Path(TESTFILE_TXT));  
}
```

6. Delete a group.

Example: Delete gid01.

```
/**  
 * delete group  
 *  
 * @throws java.io.IOException  
 */  
private static void deleteGroup() throws IOException {  
    dfsAdmin.deleteColocationGroup(COLOCATION_GROUP_GROUP01);  
}
```

## 2.7.4 Commissioning the Application

### 2.7.4.1 Commissioning an Application in the Windows Environment

#### 2.7.4.1.1 Compiling and Running an Application

##### Scenario

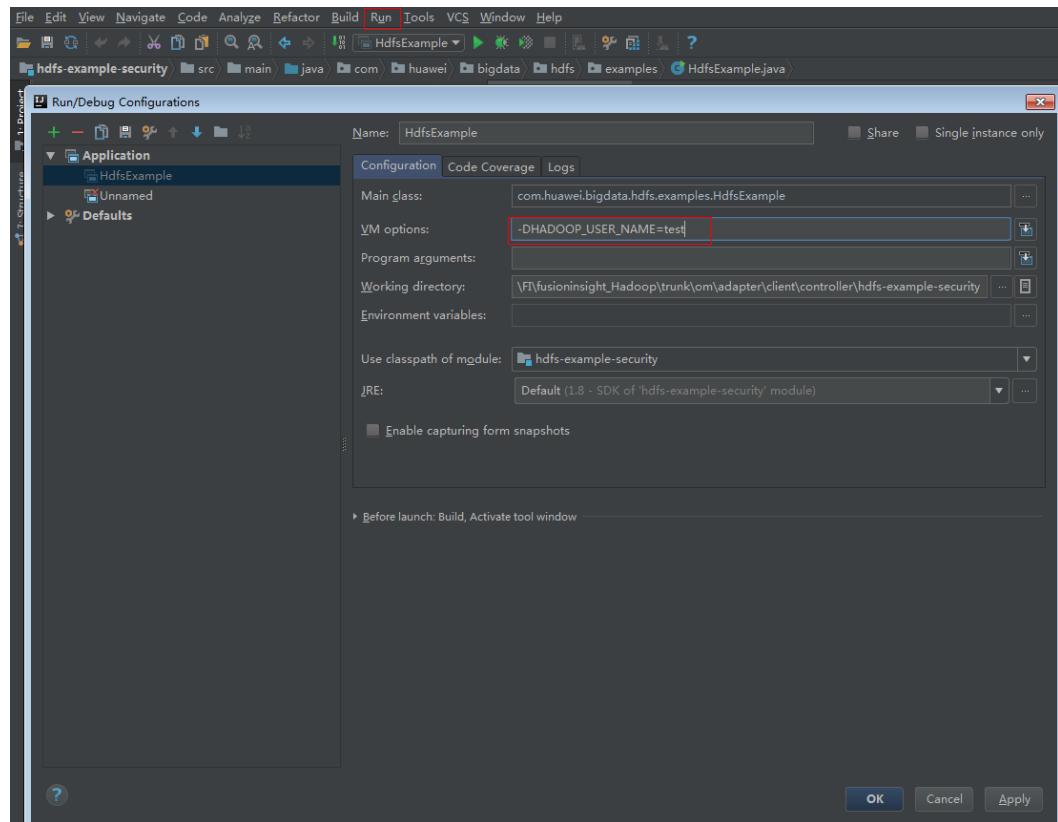
After the code development is complete, you can run an application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

##### Procedure

**Step 1** (Optional) In a development environment (for example, IntelliJ IDEA), a user must be specified to run the example code. There are two ways to specify the user:

Select the sample program HdfsExample.java or ColocationExample.java to be run, right-click the project, and choose Run Configurations from the shortcut menu. In the dialog box that is displayed, select JavaApplication > HdfsExample to set the running parameters. On the menu bar of the IntelliJ IDEA, choose Run > Edit Configurations. In the dialog box that is displayed, set the running user.

-DHADOOP\_USER\_NAME=test



 NOTE

The **test** user here is an example. To run the example code related to the Colocation operation, the user must be a member of the supergroup group.

**Step 2** If the environment variable is configured according to [Step 1](#) click **Run** to run the application. If not, choose the following two projects separately and run the projects:

- Choose HdfsExample.java, right-click the project and choose **Run 'HdfsExample.main()'** from the shortcut menu to run the project.
- Choose ColocationExample.java, right-click the project and choose **Run 'ColocationExample.main()'** from the shortcut menu to run the project.

 NOTE

- It is forbidden to restart HDFS service while HDFS application is in running status, otherwise the application will fail.
- When the Colocation project is run, the HDFS parameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

#### 2.7.4.1.2 Checking the Commissioning Result

##### Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

##### Procedure

- **Learn the application running conditions by viewing the running result.**
  - The running result of the HDFS windows example application is shown as follows:

```
1654 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-circuit local reads feature cannot be used because UNIX Domain sockets are not available on Windows.
2013 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples
2137 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2590 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
3245 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
4447 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I am bigdata. It is successful if you can see me.I append this content.
4447 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to read.
4509 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete the file /user/hdfs-examples\test.txt
4618 [main] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to delete path /user/hdfs-examples
4743 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples/hdfs_example_1
4743 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to create path /user/hdfs-examples/hdfs_example_0
5087 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
5087 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to write.
6507 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to append.
6553 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to
```

```
append.  
7505 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I  
am bigdata. It is successful if you can see me.I append this content.  
7505 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
read.  
7568 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete the file /user/hdfs-examples/hdfs_example_1\test.txt  
7583 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - result is : hi, I  
am bigdata. It is successful if you can see me.I append this content.  
7583 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
read.  
7630 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete the file /user/hdfs-examples/hdfs_example_0\test.txt  
7677 [hdfs_example_1] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete path /user/hdfs-examples/hdfs_example_1  
7739 [hdfs_example_0] INFO com.huawei.bigdata.hdfs.examples.HdfsExample - success to  
delete path /user/hdfs-examples/hdfs_example_0
```

#### NOTE

In the Windows environment, the following exception occurs but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the  
Hadoop binaries.
```

- The running result of the Colocation windows example application is shown as follows:

```
1623 [main] WARN org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory - The short-  
circuit local reads feature cannot be used because UNIX Domain sockets are not available on  
Windows.  
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client  
environment:zookeeper.version=***, built on 10/19/2017 04:21 GMT  
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client  
environment:host.name=siyay7user1.china.huawei.com  
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:java.version=***  
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:java.vendor=Oracle  
Corporation  
1670 [main] INFO org.apache.zookeeper.ZooKeeper - Client  
environment:java.home=D:\Program Files\Java\jre1.8.0_131  
.....  
Create Group has finished.  
Put file is running...  
5930 [main] WARN org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop  
library for your platform... using builtin-java classes where applicable  
Put file has finished.  
Delete file is running...  
Delete file has finished.  
Delete Group is running...  
Delete Group has finished.  
6866 [main] INFO org.apache.zookeeper.ZooKeeper - Session: 0x13000074b7e464b7 closed  
6866 [main-EventThread] INFO org.apache.zookeeper.ClientCnxn - EventThread shut down for  
session: 0x13000074b7e464b7  
6928 [main-EventThread] INFO org.apache.zookeeper.ClientCnxn - EventThread shut down for  
session: 0x14000073f13b657b  
6928 [main] INFO org.apache.zookeeper.ZooKeeper - Session: 0x14000073f13b657b closed
```

- **Learn the application running conditions by viewing HDFS logs.**

The NameNode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

### 2.7.4.2 Commissioning an Application in the Linux Environment

### 2.7.4.2.1 Compiling and Running an Application with the Client Installed

#### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client installed. After the application code has been developed, you can upload the jar packages to the prepared Linux client operating environment and run the application.

#### Prerequisite

- The HDFS client has been installed.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the client is installed. The host name must be correctly mapped to the IP address.

#### Procedure

**Step 1** Go to the local root directory of the sample project, copy the required configuration file to the conf folder of the local project, and run the following command in Windows CLI to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```



##### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package, for example, *HDFSTest-XXX.jar*, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

**Step 2** Upload the exported jar packages to any directory in the running environment of the client, for example, **/opt/hadoopclient**.

**Step 3** Configure the environment variables:

```
cd /opt/hadoopclient  
source bigdata_env
```

**Step 4** Specify a user to run the example. There are two ways to specify the user: add the environment variable **HADOOP\_USER\_NAME** and modify the code. If the code cannot be modified, run the following statement to add the environment variable:  
`export HADOOP_USER_NAME=test`



The **test** user here is an example. To run the example code related to the Colocation operation, the user must be a member of the supergroup group.

**Step 5** Run the following commands to execute the jar packages.

```
hadoop jar HDFSTest-XXX.jar com.huawei.bigdata.hdfs.examples.HdfsExample
```

```
hadoop jar HDFSTest-XXX.jar  
com.huawei.bigdata.hdfs.examples.ColocationExample
```

 NOTE

When `com.huawei.bigdata.hdfs.examples.ColocationExample` is run, the HDFS parameter `fs.defaultFS` cannot be set to `viewfs://ClusterX`.

----End

### 2.7.4.2.2 Compiling and Running an Application with the Client Not Installed

#### Scenario

The Hadoop distributed file system (HDFS) application can run in the Linux operating system (OS) with the HDFS client not installed. After the application code has been developed, you can upload the jar packages to the Linux OS and run the application.

#### Prerequisite

- A JDK has been installed in the Linux environment. The version of the JDK must be consistent with that of the JDK used by IDEA to export the JAR package.
- When the host where the Linux OS runs is not a node of the cluster, you are required to set the mapping between the host name and IP address in the **hosts** file of the node where the Linux OS runs. The host name must be correctly mapped to the IP address.

#### Procedure

**Step 1** Go to the local root directory of the project, copy the required configuration file to the conf folder of the local project and run the following command in Windows CLI to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```

 NOTE

- In the preceding command, {maven\_setting\_path} is the path of the **settings.xml** file of the local Maven.
- After the package is successfully packed, obtain the JAR package from the target subdirectory in the root directory of the project.

**Step 2** Upload the exported jar packages to any directory in the running environment of the Linux OS, for example, **/opt/hadoopclient**.

**Step 3** Create a **lib** folder in the Linux operating environment directory (for example, **/opt/hadoopclient**) and upload the required **JAR** packages. The **lib** folder contains all the **JAR** packages that the project depends on. For details, see section [Preparing an Operating Environment](#).

**Step 4** Specify a user to run the example. There are two ways to specify the user: add the environment variable **HADOOP\_USER\_NAME** and modify the code. If the code cannot be modified, run the following statement to add the environment variable:

```
export HADOOP_USER_NAME=test
```

 NOTE

The **test** user here is an example. To run the example code related to the Colocation operation, the user must be a member of the supergroup group.

**Step 5** Run the following commands to execute the jar packages.

```
java -cp HDFSTest-XXX.jar.lib/* com.huawei.bigdata.hdfs.examples.HdfsExample
```

```
java -cp HDFSTest-XXX.jar.lib/*  
com.huawei.bigdata.hdfs.examples.ColocationExample
```

 NOTE

When **com.huawei.bigdata.hdfs.examples.ColocationExample** is run, the HDFS parameter **fs.defaultFS** cannot be set to **viewfs://ClusterX**.

----End

### 2.7.4.2.3 Checking the Commissioning Result

#### Scenario

After an HDFS application is run, you can learn the application running conditions by viewing the running result or HDFS logs.

#### Procedure

- **Learn the application running conditions by viewing the running result.**

- The running result of the HDFS example application is shown as follows:

```
[root@192-168-32-144 client]#hadoop jar HDFSTest-XXX.jar  
com.huawei.bigdata.hdfs.examples.HdfsExample  
WARNING: Use "yarn jar" to launch YARN applications.  
17/10/26 19:11:44 INFO examples.HdfsExample: success to create path /user/hdfs-examples  
17/10/26 19:11:44 INFO examples.HdfsExample: success to write.  
17/10/26 19:11:45 INFO examples.HdfsExample: success to append.  
17/10/26 19:11:45 INFO examples.HdfsExample: result is : hi, I am bigdata. It is successful if you  
can see me.l append this content.  
17/10/26 19:11:45 INFO examples.HdfsExample: success to read.  
17/10/26 19:11:45 INFO examples.HdfsExample: success to delete the file /user/hdfs-examples/  
test.txt  
17/10/26 19:11:45 INFO examples.HdfsExample: success to delete path /user/hdfs-examples/  
17/10/26 19:11:45 INFO examples.HdfsExample: success to create path /user/hdfs-examples/  
hdfs_example_1  
17/10/26 19:11:45 INFO examples.HdfsExample: success to create path /user/hdfs-examples/  
hdfs_example_0  
17/10/26 19:11:45 INFO examples.HdfsExample: success to write.  
17/10/26 19:11:45 INFO examples.HdfsExample: success to write.  
17/10/26 19:11:46 INFO examples.HdfsExample: success to append.  
17/10/26 19:11:46 INFO examples.HdfsExample: result is : hi, I am bigdata. It is successful if you  
can see me.l append this content.  
17/10/26 19:11:46 INFO examples.HdfsExample: success to read.  
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete the file /user/hdfs-examples/  
hdfs_example_1/test.txt  
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete path /user/hdfs-examples/  
hdfs_example_1  
17/10/26 19:11:46 INFO examples.HdfsExample: success to append.  
17/10/26 19:11:46 INFO examples.HdfsExample: result is : hi, I am bigdata. It is successful if you  
can see me.l append this content.  
17/10/26 19:11:46 INFO examples.HdfsExample: success to read.  
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete the file /user/hdfs-examples/  
hdfs_example_0/test.txt  
17/10/26 19:11:46 INFO examples.HdfsExample: success to delete path /user/hdfs-examples/  
hdfs_example_0
```

- The running result of the Colocation example application is shown as follows:

```
[root@192-168-32-144 client]#hadoop jar HDFSTest-XXX.jar  
com.huawei.bigdata.hdfs.examples.ColocationExample  
WARNING: Use "yarn jar" to launch YARN applications.  
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:zookeeper.version=xxx, built  
on 10/19/2017 04:21 GMT  
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:host.name=192-168-32-144  
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.version=1.8.0_144  
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.vendor=Oracle  
Corporation  
17/10/26 19:12:38 INFO zookeeper.ZooKeeper: Client environment:java.home=/opt/  
hadoopclient/JDK/jdk1.8.0_144/jre  
.....  
Create Group has finished.  
Put file is running...  
Put file has finished.  
Delete file is running...  
Delete file has finished.  
Delete Group is running...  
Delete Group has finished.  
17/10/26 19:12:39 INFO zookeeper.ZooKeeper: Session: 0x13000074b7e4687f closed  
17/10/26 19:12:39 INFO zookeeper.ClientCnxn: EventThread shut down for session:  
0x13000074b7e4687f  
17/10/26 19:12:39 INFO zookeeper.ZooKeeper: Session: 0x12000059699f69e1 closed  
17/10/26 19:12:39 INFO zookeeper.ClientCnxn: EventThread shut down for session:  
0x12000059699f69e1
```

- **Learn the application running conditions by viewing HDFS logs.**

The NameNode logs of HDFS offer immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 2.7.5 More Information

### 2.7.5.1 Common API Introduction

#### 2.7.5.1.1 Java API

For details about Hadoop distributed file system (HDFS) APIs, see:

<http://hadoop.apache.org/docs/r3.3.1/api/index.html>

#### HDFS Common API

Common HDFS Java classes are as follows:

- FileSystem: the core class of client applications. For details about common APIs, see [Table 2-83](#).
- FileStatus: record the status of files and directories. For details about common APIs, see [Table 2-84](#).
- DFSColocatinAdmin: API used to manage colocatiion group information. For details about common APIs, see [Table 2-85](#).
- DFSColocatinClient: API used to manage colocatiion files. For details about common APIs, see [Table 2-86](#).

 NOTE

- The system reserves only the mapping between nodes and locator IDs, but does not reserve the mapping between files and locator IDs. When a file is created using a Colocation interface, the file is created on the node that corresponds to a locator ID. File creation and writing must be performed using Colocation interfaces.
- After the file is written, subsequent operations on the file can use other open-source interfaces in addition to Colocation interfaces.
- The DFSColocatiionClient class inherits from the open-source DistributedFileSystem class and contains common file operation functions. If a user uses the DFSColocatiionClient class to create a Colocation file, the user is advanced to use the functions of this class in file operations.

**Table 2-83** Common FileSystem APIs

API	Description
public static FileSystem get(Configuration conf)	FileSystem is the API class provided for users in the Hadoop class library. FileSystem is an abstract class. Concrete classes can be obtained only using the get method. The get method has multiple overload versions and is commonly used.
public FSDataOutputStream create(Path f)	This API is used to create files in the HDFS. <i>f</i> indicates a complete file path.
public void copyFromLocalFile(Pat h src, Path dst)	This API is used to upload local files to a specified directory in the HDFS. <i>src</i> and <i>dst</i> indicate complete file paths.
public boolean mkdirs(Path f)	This API is used to create folders in the HDFS. <i>f</i> indicates a complete folder path.
public abstract boolean rename(Path src, Path dst)	This API is used to rename a specified HDFS file. <i>src</i> and <i>dst</i> indicate complete file paths.
public abstract boolean delete(Path f, boolean recursive)	This API is used to delete a specified HDFS file. <i>f</i> indicates the complete path of the file to be deleted, and <b>recursive</b> specifies recursive deletion.
public boolean exists(Path f)	This API is used to query a specified HDFS file. <i>f</i> indicates a complete file path.
public FileStatus getFileStatus(Path f)	This API is used to obtain the FileStatus object of a file or folder. The FsStatus object records status information of the file or folder, including the modification time and file directory.
public BlockLocation[] getFileBlockLoca tions(FileStatus file, long start, long len)	This API is used to query the block location of a specified file in an HDFS cluster. <i>file</i> indicates a complete file path, and <i>start</i> and <i>len</i> specify the block scope.

API	Description
public FSDataInputStream open(Path f)	This API is used to open the output stream of a specified file in the HDFS and read the file using the API provided by the FSDataInputStream class. <i>f</i> indicates a complete file path.
public FSDataOutputStream create(Path f, boolean overwrite)	This API is used to create the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class. <i>f</i> indicates a complete file path. If <b>overwrite</b> is <b>true</b> , the file is rewritten if it exists; if <b>overwrite</b> is <b>false</b> , an error is reported if the file exists.
public FSDataOutputStream append(Path f)	This API is used to open the input stream of a specified file in the HDFS and write the file using the API provided by the FSDataOutputStream class. <i>f</i> indicates a complete file path.

**Table 2-84** Common FileStatus APIs

API	Description
public long getModificationTime()	This API is used to query the modification time of a specified HDFS file.
public Path getPath()	This API is used to query all files in an HDFS directory.

**Table 2-85** Common DFSColocationAdmin APIs

API	Description
public Map<String, List<DatanodeInfo>> createColocationGroup(String groupId,String file)	This API is used to create a group based on the locatorIds information in the file. <i>file</i> indicates the file path.
public Map<String, List<DatanodeInfo>> createColocationGroup(String groupId,List<String> locators)	This API is used to create a group based on the locatorIds information in the list in the memory.
public void deleteColocationGroup(String groupId)	This API is used to delete a group.
public List<String> listColocationGroups()	This API is used to return all group information of Colocation. The returned group ID arrays are sorted by the creation time.

API	Description
public List<DatanodeInfo> getNodesForLocator(String groupId, String locatorId)	This API is used to obtain the list of all nodes in the locator.

**Table 2-86** Common DFSColocationAdmin APIs

API	Description
public FSDataOutputStream create(Path f, boolean overwrite, String groupId, String locatorId)	This API is used to create a FSDataOutputStream in colocation mode to allow users to write files in f. f is the HDFS path. overwrite indicates whether an existing file can be overwritten. groupId and locatorId of the file specified by a user must exist.
public FSDataOutputStream create(final Path f, final FsPermission permission, final EnumSet<CreateFlag> cflags, final int bufferSize, final short replication, final long blockSize, final Progressable progress, final ChecksumOpt checksumOpt, final String groupId, final String locatorId)	The function of this API is the same as that of FSDataOutputStream create(Path f, boolean overwrite, String groupId, String locatorId), except that users are allowed to customize checksum.
public void close()	This API is used to close the connection.

**Table 2-87** HDFS client WebHdfsFileSystem API

API	Description
public RemoteIterator<FileStatus> listStatusIterator(final Path)	This API will help in fetching the child files and folders information through multiple request using remote iterator, thus avoiding the user interface from becoming slow when there are millions of child information to be fetched.

## Glob path pattern based API to get LocatedFileStatus and Open file from FileStatus

Following APIs are added in DistributedFileSystem to get the FileStatus with block location and open file from FileStatus object. These APIs will reduce the number of RPC calls from client to Namenodes.

**Table 2-88** FileSystem APIs

Interface	Description
public LocatedFileStatus[] globLocatedStatus(Path, PathFilter, boolean) throws IOException	Return an array of LocatedFileStatus objects whose path names match pathPattern and pass the in path filter.
public FSDataInputStream open(FileStatus stat) throws IOException	If the stat is an instance of LocatedFileStatusHdfs that already have the location information, the InputStream is created without contacting NameNode.

### 2.7.5.1.2 C API

#### Function Description

Users can use the C application programming interface (API) to create, read and write, append, and delete files. For details of the C API, see the following official guidelines:

<https://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/LibHdfs.html>

#### Code Sample

The following code snippets are used as an example. For complete code, see the HDFS C sample code **HDFS/hdfs-c-example/hdfs\_test.c** in the HDFS sample code decompression directory.

1. Configure the HDFS NameNode parameter and create the link connecting to the HDFS files.

```
hdfsFS fs = hdfsConnect("default", 0);
fprintf(stderr, "hdfsConnect- SUCCESS!\n");
```

2. Create the HDFS directory.

```
const char* dir = "/tmp/nativeTest";
int exitCode = hdfsCreateDirectory(fs, dir);
if( exitCode == -1 ){
    fprintf(stderr, "Failed to create directory %s \n", dir );
    exit(-1);
}
fprintf(stderr, "hdfsCreateDirectory- SUCCESS! : %s\n", dir);
```

3. Write files.

```
const char* file = "/tmp/nativeTest/testfile.txt";
hdfsFile writeFile = openFile(fs, (char*)file, O_WRONLY |O_CREAT, 0, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for write : %s\n", file);

if(!hdfsFileIsOpenForWrite(writeFile)){
    fprintf(stderr, "Failed to open %s for writing.\n", file);
    exit(-1);
}

char* buffer = "Hadoop HDFS Native file write!";

hdfsWrite(fs, writeFile, (void*)buffer, strlen(buffer)+1);
fprintf(stderr, "hdfsWrite- SUCCESS! : %s\n", file);
```

```
printf("Flushing file data ....\n");
if (hdfsFlush(fs, writeFile)) {
    fprintf(stderr, "Failed to 'flush' %s\n", file);
    exit(-1);
}
hdfsCloseFile(fs, writeFile);
fprintf(stderr, "hdfsCloseFile- SUCCESS! : %s\n", file);
```

4. Read files.

```
hdfsFile readFile = openFile(fs, (char*)file, O_RDONLY, 100, 0, 0);
fprintf(stderr, "hdfsOpenFile- SUCCESS! for read : %s\n", file);

if(!hdfsFileisOpenForRead(readFile)){
    fprintf(stderr, "Failed to open %s for reading.\n", file);
    exit(-1);
}

buffer = (char *) malloc(100);
tSize num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsRead- SUCCESS!, Byte read : %d, File content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);
```

5. From the specified location to read the file.

```
buffer = (char *) malloc(100);
readFile = openFile(fs, file, O_RDONLY, 100, 0, 0);
if (hdfsSeek(fs, readFile, 10)) {
    fprintf(stderr, "Failed to 'seek' %s\n", file);
    exit(-1);
}
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsSeek- SUCCESS!, Byte read : %d, File seek content : %s \n", num_read ,buffer);
hdfsCloseFile(fs, readFile);
```

6. Copy the file.

```
const char* destfile = "/tmp/nativeTest/testfile1.txt";
if (hdfsCopy(fs, file, fs, destfile)) {
    fprintf(stderr, "File copy failed, src : %s, des : %s \n", file, destfile);
    exit(-1);
}
fprintf(stderr, "hdfsCopy- SUCCESS!, File copied, src : %s, des : %s \n", file, destfile);
```

7. Move the file.

```
const char* mvfile = "/tmp/nativeTest/testfile2.txt";
if (hdfsMove(fs, destfile, fs, mvfile )) {
    fprintf(stderr, "File move failed, src : %s, des : %s \n", destfile , mvfile);
    exit(-1);
}
fprintf(stderr, "hdfsMove- SUCCESS!, File moved, src : %s, des : %s \n", destfile , mvfile);
```

8. Rename the file.

```
const char* renamefile = "/tmp/nativeTest/testfile3.txt";
if (hdfsRename(fs, mvfile, renamefile)) {
    fprintf(stderr, "File rename failed, Old name : %s, New name : %s \n", mvfile, renamefile);
    exit(-1);
}
fprintf(stderr, "hdfsRename- SUCCESS!, File renamed, Old name : %s, New name : %s \n", mvfile, renamefile);
```

9. Delete Files.

```
if (hdfsDelete(fs, renamefile, 0)) {
    fprintf(stderr, "File delete failed : %s \n", renamefile);
    exit(-1);
}
fprintf(stderr, "hdfsDelete- SUCCESS!, File deleted : %s\n",renamefile);
```

10. Set the number of replications.

```
if (hdfsSetReplication(fs, file, 10)) {
    fprintf(stderr, "Failed to set replication : %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsSetReplication- SUCCESS!, Set replication 10 for %s\n",file);
```

11. Set users, user groups.

```
if (hdfsChown(fs, file, "root", "root")) {
    fprintf(stderr, "Failed to set chown : %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsChown- SUCCESS!, Chown success for %s\n",file);
```

12. Set permissions.

```
if (hdfsChmod(fs, file, S_IRWXU | S_IRWXG | S_IROTH)) {
    fprintf(stderr, "Failed to set chmod: %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsChmod- SUCCESS!, Chmod success for %s\n",file);
```

13. Set the file time.

```
struct timeval now;
gettimeofday(&now, NULL);
if (hdfsUtime(fs, file, now.tv_sec, now.tv_sec)) {
    fprintf(stderr, "Failed to set time: %s \n", file );
    exit(-1);
}
fprintf(stderr, "hdfsUtime- SUCCESS!, Set time success for %s\n",file);
```

14. Get file information.

```
hdfsFileInfo * fileInfo = NULL;
if((fileInfo = hdfsGetPathInfo(fs, file)) != NULL) {
    printFileInfo(fileInfo);
    hdfsFreeFileInfo(fileInfo, 1);
    fprintf(stderr, "hdfsGetPathInfo - SUCCESS!\n");
}
```

15. Variable directory.

```
hdfsFileInfo * fileList = 0;
int numEntries = 0;
if((fileList = hdfsListDirectory(fs, dir, &numEntries)) != NULL) {
    int i = 0;
    for(i=0; i < numEntries; ++i) {
        printFileInfo(fileList+i);
    }
    hdfsFreeFileInfo(fileList, numEntries);
}
fprintf(stderr, "hdfsListDirectory- SUCCESS!, %s\n", dir);
```

16. Stream builder interfaces.

```
buffer = (char *) malloc(100);
struct hdfsStreamBuilder *builder= hdfsStreamBuilderAlloc(fs, (char*)file, O_RDONLY);
hdfsStreamBuilderSetBufferSize(builder,100);
hdfsStreamBuilderSetReplication(builder,20);
hdfsStreamBuilderSetDefaultBlockSize(builder,10485760);
readFile = hdfsStreamBuilderBuild(builder);
num_read = hdfsRead(fs, readFile, (void*)buffer, 100);
fprintf(stderr, "hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : %d, File content : %s
\n", num_read ,buffer);
free(buffer);

struct hdfsReadStatistics *stats = NULL;
hdfsFileGetReadStatistics(readFile, &stats);
fprintf(stderr, "hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : %" PRId64 " ,
totalLocalBytesRead : %" PRId64 " , totalShortCircuitBytesRead : %" PRId64 " ,
totalZeroCopyBytesRead : %" PRId64 "\n", stats->totalBytesRead , stats->totalLocalBytesRead, stats-
>totalShortCircuitBytesRead, stats->totalZeroCopyBytesRead);
hdfsFileFreeReadStatistics(stats);
```

17. Disconnect the HDFS links.

```
hdfsDisconnect(fs);
```

## Preparing Running Environment

Install a client on the node. For example, install a client in the **/opt/hadoopclient** directory.

## Compiling and Running Applications in Linux

1. Go to the **/opt/hadoopclient** directory and run the following command to import the environment variables of the C client:

```
cd /opt/hadoopclient  
source bigdata_env
```

2. Go to the **/opt/hadoopclient/HDFS/hadoop/hdfs-c-example** directory and run the following command to import the environment variables of the C client:

```
cd /opt/hadoopclient/HDFS/hadoop/hdfs-c-example  
source component_env_C_example
```

3. Run the following command to clean the object files and executable files that are generated before:

```
make clean
```

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make clean  
rm -f hdfs_test.o  
rm -f hdfs_test
```

4. Run the following command to compile the new object files and executable files:

```
make ( or make all )
```

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make  
cc -c -I/opt/hadoopclient/HDFS/hadoop/include -Wall -o hdfs_test.o hdfs_test.c  
cc -o hdfs_test hdfs_test.o -lhdfs
```

5. Run the following command to create, write and read, append, and delete files:

```
make run
```

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make run  
.hdfs_test  
hdfsConnect- SUCCESS!  
hdfsCreateDirectory- SUCCESS! : /tmp/nativeTest  
hdfsOpenFile- SUCCESS! for write : /tmp/nativeTest/testfile.txt  
hdfsWrite- SUCCESS! : /tmp/nativeTest/testfile.txt  
Flushing file data ....  
hdfsCloseFile- SUCCESS! : /tmp/nativeTest/testfile.txt  
hdfsOpenFile- SUCCESS! for read : /tmp/nativeTest/testfile.txt  
hdfsRead- SUCCESS!, Byte read : 31, File content : Hadoop HDFS Native file write!  
hdfsSeek- SUCCESS!, Byte read : 21, File seek content : S Native file write!  
hdfsPread- SUCCESS!, Byte read : 10, File phead content : S Native f  
hdfsCopy- SUCCESS!, File copied, src : /tmp/nativeTest/testfile.txt, des : /tmp/nativeTest/testfile1.txt  
hdfsMove- SUCCESS!, File moved, src : /tmp/nativeTest/testfile1.txt, des : /tmp/nativeTest/testfile2.txt  
hdfsRename- SUCCESS!, File renamed, Old name : /tmp/nativeTest/testfile2.txt, New name : /tmp/  
nativeTest/testfile3.txt  
hdfsDelete- SUCCESS!, File deleted : /tmp/nativeTest/testfile3.txt  
hdfsSetReplication- SUCCESS!, Set replication 10 for /tmp/nativeTest/testfile.txt  
hdfsChown- SUCCESS!, Chown success for /tmp/nativeTest/testfile.txt  
hdfsChmod- SUCCESS!, Chmod success for /tmp/nativeTest/testfile.txt  
hdfsUtime- SUCCESS!, Set time success for /tmp/nativeTest/testfile.txt  
  
Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:  
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)  
hdfsGetPathInfo - SUCCESS!
```

```
Name: hdfs://hacluster/tmp/nativeTest/testfile.txt, Type: F, Replication: 10, BlockSize: 134217728, Size:
```

```
31, LastMod: 1500345260, Owner: root, Group: root, Permissions: 511 (rwxrwxrwx)
hdfsListDirectory- SUCCESS!, /tmp/nativeTest
hdfsTruncateFile- SUCCESS!, /tmp/nativeTest/testfile.txt
Block Size : 134217728
hdfsGetDefaultBlockSize- SUCCESS!
Block Size : 134217728 for file /tmp/nativeTest/testfile.txt
hdfsGetDefaultBlockSizeAtPath- SUCCESS!
HDFS Capacity : 102726873909
hdfsGetCapacity- SUCCESS!
HDFS Used : 4767076324
hdfsGetCapacity- SUCCESS!
hdfsExists- SUCCESS! /tmp/nativeTest/testfile.txt
hdfsConfGetStr- SUCCESS : hdfs://hacluster
hdfsStreamBuilderBuild- SUCCESS! File read success. Byte read : 31, File content : Hadoop HDFS
Native file write!
hdfsFileGetReadStatistics- SUCCESS! totalBytesRead : 31, totalLocalBytesRead : 0,
totalShortCircuitBytesRead : 0, totalZeroCopyBytesRead : 0
```

6. Enter the debug mode. (Optional)

**make gdb**

The running result is displayed as follows:

```
[root@10-120-85-2 hdfs-c-example]# make gdb
gdb hdfs_test
GNU gdb (GDB) SUSE (7.5.1-0.7.29)
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-suse-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /opt/hadoopclient/HDFS/hadoop/hdfs-c-example/hdfs_test...done.
(gdb)
```

### 2.7.5.1.3 HTTP REST API

#### Function Description

Users can use the application programming interface (API) of Representational State Transfer (REST) to create, read and write, append, and delete files. For details of the REST API, see the following official guidelines:

<http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html>.

#### Preparing Running Environment

**Step 1** Install the client. Install the client on the node. For example, install the client in the **/opt/hadoopclient** directory. See details in "Installing the Client."

1. Prepare files **testFile** and **testFileAppend** and write content 'Hello, webhdfs user!' and 'Welcome back to webhdfs!'. Run the following command to prepare **testFile** and **testFileAppend** files:

**touch testFile**

**vi testFile**

Write 'Hello, webhdfs user!', save the files, and exit.

**touch testFileAppend**

**vi testFileAppend**

Write 'Welcome back to webhdfs!', save the files, and exit.

- Step 2** In normal mode, only the HTTP service is supported. [Accessing FusionInsight Manager of an MRS Cluster](#), choose **Cluster > Name of the desired cluster > Services > HDFS > Configurations > All Configurations**. Type **dfs.http.policy** in the research box, select **HTTP\_ONLY**, click **Save Configuration**, and select **Restart the affected services or instances**. Click **OK** to restart the HDFS service.



**HTTP\_ONLY** is selected by default.

----End

## Procedure

- Step 1** [Accessing FusionInsight Manager of an MRS Cluster](#), click **Cluster > Name of the desired cluster > Services**, and then select **HDFS**. The HDFS page is displayed.



Because webhdfs is accessed through HTTP, you need to obtain the IP address of the active NameNode and the HTTP port.

1. Click **Instances**, view the host name and IP address of the active NameNode.
2. Click **Configurations**, search **namenode.http.port** in the search box (25002).

- Step 2** Create a directory by referring to the following link:

[http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make\\_a\\_Directory](http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/WebHDFS.html#Make_a_Directory).

Click the link, **Figure 2-153** is displayed:

**Figure 2-153** Example code of creating a directory

The screenshot shows a browser interface for creating a directory. It includes a URL input field with the value "curl -i -X PUT \"http://:8033/webhdfs/v1/25002/tmp/testdir?op=MKDIRS&permissions=0777\"". Below it, a text area displays the response: "HTTP/1.1 200 OK Content-Type: application/json Transfer-Encoding: chunked {"boolean": true}".

Go to the **/opt/hadoopclient** directory, the installation directory of the client, and create the **huawei** directory.

1. Run the following command to check whether the **huawei** directory exists in the current path.

**hdfs dfs -ls /**

The running results are as follows:

```
linux1:/opt/hadoopclient # hdfs dfs -ls /
16/04/22 16:10:02 INFO hdfs.PeerCache: SocketCache disabled.
Found 7 items
-rw-r--r-- 3 hdfs supergroup          0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS
drwxr-x---  - flume hadoop          0 2016-04-20 18:02 /flume
drwx-----  - hbase hadoop          0 2016-04-22 15:19 /hbase
drwxrwxrwx  - mapred hadoop         0 2016-04-20 18:02 /mr-history
```

```
drwxrwxrwx - spark supergroup 0 2016-04-22 15:19 /sparkJobHistory  
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:51 /tmp  
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:50 /user
```

The **huawei** directory does not exist in the current path.

2. Run the command in [Figure 2-153](#) that is named with **huawei**. Replace the <HOST> and <PORT>in the command with the host name or IP address and port number that are obtained in [Step 1](#). Type the **huawei** as the directory in the <PATH>.

 NOTE

<HOST> can be replaced by the host name or IP address. It is noted that the port of HTTP is different from the port of HTTPS.

- Run the following command to access HTTP:  
`curl -i -X PUT --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei?  
user.name=test&op=MKDIRS"`

In the command, <HOST> is replaced by **linux1** and <PORT> is replaced by **25002**.

The test in the preceding command is the user who performs the operation. The user must confirm with the administrator for the permission.

- The running result is displayed as follows:

```
HTTP/1.1 200 OK  
Cache-Control: no-cache  
Expires: Thu, 14 Jul 2016 08:04:39 GMT  
Date: Thu, 14 Jul 2016 08:04:39 GMT  
Pragma: no-cache  
Expires: Thu, 14 Jul 2016 08:04:39 GMT  
Date: Thu, 14 Jul 2016 08:04:39 GMT  
Pragma: no-cache  
Content-Type: application/json  
X-FRAME-OPTIONS: SAMEORIGIN  
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468519479514&s=/j/J  
+ZnVrN7NSz1yKnB2JViwkj0="; Path=/; Expires=Thu, 14-Jul-2016 18:04:39 GMT; HttpOnly  
Transfer-Encoding: chunked  
{"boolean":true}
```

If {"boolean":true} returns, the **huawei** directory is successfully created.

3. Run the following command to check the **huawei** directory in the path.

```
linux1:/opt/hadoopclient # hdfs dfs -ls /  
16/04/22 16:14:25 INFO hdfs.PeerCache: SocketCache disabled.  
Found 8 items  
-rw-r--r-- 3 hdfs supergroup 0 2016-04-20 18:03 /PRE_CREATE_DIR.SUCCESS  
drwxr-x--- - flume hadoop 0 2016-04-20 18:02 /flume  
drwx----- - hbase hadoop 0 2016-04-22 15:19 /hbase  
drwxr-xr-x - hdfs supergroup 0 2016-04-22 16:13 /huawei  
drwxrwxrwx - mapred hadoop 0 2016-04-20 18:02 /mr-history  
drwxrwxrwx - spark supergroup 0 2016-04-22 16:12 /sparkJobHistory  
drwxrwxrwx - hdfs hadoop 0 2016-04-22 14:51 /tmp  
drwxrwxrwx - hdfs hadoop 0 2016-04-22 16:10 /user
```

**Step 3** Create a command of the upload request to obtain the information about Location where the DataNode IP address is written in.

- Run the following command to access HTTP:  
`linux1:/opt/hadoopclient # curl -i -X PUT --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/  
testHdfs?user.name=test&op=CREATE"`

- The running result is displayed as follows:

```
HTTP/1.1 307 TEMPORARY_REDIRECT  
Cache-Control: no-cache  
Expires: Thu, 14 Jul 2016 08:53:07 GMT  
Date: Thu, 14 Jul 2016 08:53:07 GMT
```

```
Pragma: no-cache
Expires: Thu, 14 Jul 2016 08:53:07 GMT
Date: Thu, 14 Jul 2016 08:53:07 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-FRAME-OPTIONS: SAMEORIGIN
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468522387880&s=OlksfRJvEkh/
Out9y2Ot2FvrxWk="; Path=/; Expires=Thu, 14-Jul-2016 18:53:07 GMT; HttpOnly
Location:
http://10-120-180-170:25010/webhdfs/v1/testHdfs?
op=CREATE&user.name=hdfs&namenoderpcaddress=hacluster&createflag=&createparent=true&overwr
ite=false
Content-Length: 0
```

**Step 4** According to the Location information, create the **testHdfs** file in the **/huawei/testHdfs** file on the HDFS and upload the content in the local **testFile** file into the **testHdfs** file.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X PUT -T testFile --negotiate -u: "http://10-120-180-170:25010/
webhdfs/v1/testHdfs?
op=CREATE&user.name=test&namenoderpcaddress=hacluster&createflag=&createparent=true&overwr
ite=false"
```

- The running result is displayed as follows:

```
HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Location: hdfs://hacluster/testHdfs
Content-Length: 0
Connection: close
```

**Step 5** Go to the **/huawei/testHdfs** directory and read the content of **testHdfs** file.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -L --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs??
user.name=test&op=OPEN"
```

- The running result is displayed as follows:

```
Hello, webhdfs user!
```

**Step 6** Create a command of the upload request to obtain the information about Location where the DataNode IP address of **testHdfs** file is written in.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X POST --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/
testHdfs??user.name=test&op=APPEND"
```

- The running result is displayed as follows:

```
HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Thu, 14 Jul 2016 09:18:30 GMT
Date: Thu, 14 Jul 2016 09:18:30 GMT
Pragma: no-cache
Expires: Thu, 14 Jul 2016 09:18:30 GMT
Date: Thu, 14 Jul 2016 09:18:30 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-FRAME-OPTIONS: SAMEORIGIN
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468523910234&s=JGK
+6M6PsVMFdAw2cg!HaKU1kBm="; Path=/; Expires=Thu, 14-Jul-2016 19:18:30 GMT; HttpOnly
Location:
http://10-120-180-170:25010/webhdfs/v1/testHdfs?
op=APPEND&user.name=hdfs&namenoderpcaddress=hacluster
Content-Length: 0
```

**Step 7** According to the Location information, add the content in the local **testFileAppend** file to the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X POST -T testFileAppend --negotiate -u: "http://linux1:25010/webhdfs/v1/huawei/testHdfs?user.name=test&op=APPEND&namenoderpcaddress=hacluster"
```

- The running result is displayed as follows:

```
HTTP/1.1 100 Continue  
HTTP/1.1 200 OK  
Content-Length: 0  
Connection: close
```

**Step 8** Go to the **/huawei/testHdfs** directory and read all content in the **testHdfs** file.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -L --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?user.name=test&op=OPEN"
```

- The running result is displayed as follows:

```
Hello, webhdfs user!  
Welcome back to webhdfs!
```

**Step 9** List details of all directory and file information in the **huawei** directory of the HDFS.

LISTSTATUS will return all child files and folders information in a single request.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?user.name=test&op=LISTSTATUS"
```

- The result is displayed as follows:

```
{"FileStatuses":[{"FileStatus":{  
"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr  
oup","length":70,"modificationTime":1462426678379,"owner":"test","pathSuffix":"","permission":"755",  
"replication":3,"storagePolicy":0,"type":"FILE"}  
}]}
```

LISTSTATUS along with size and startafter param will help in fetching the child files and folders information through multiple request, thus avoiding the user interface from becoming slow when there are millions of child information to be fetched.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/?user.name=test&op=LISTSTATUS&startafter=sparkJobHistory&size=1"
```

- The result is displayed as follows:

```
{"FileStatuses":[{"FileStatus":{  
"accessTime":1462425245595,"blockSize":134217728,"childrenNum":0,"fileId":17680,"group":"supergr  
oup","length":70,"modificationTime":1462426678379,"owner":"test","pathSuffix":"testHdfs","permmissio  
n":"755","replication":3,"storagePolicy":0,"type":"FILE"}  
}]}
```

**Step 10** Delete the **testHdfs** file that is in the **/huawei/testHdfs** directory of HDFS.

- Run the following command to access HTTP:

```
linux1:/opt/hadoopclient # curl -i -X DELETE --negotiate -u: "http://linux1:25002/webhdfs/v1/huawei/testHdfs?user.name=test&op=DELETE"
```

- The running result is displayed as follows:

```
HTTP/1.1 200 OK  
Cache-Control: no-cache  
Expires: Thu, 14 Jul 2016 10:27:44 GMT  
Date: Thu, 14 Jul 2016 10:27:44 GMT  
Pragma: no-cache  
Expires: Thu, 14 Jul 2016 10:27:44 GMT  
Date: Thu, 14 Jul 2016 10:27:44 GMT  
Pragma: no-cache  
Content-Type: application/json  
X-FRAME-OPTIONS: SAMEORIGIN  
Set-Cookie: hadoop.auth="u=hdfs&p=hdfs&t=simple&e=1468528064220&s=HrvUEd72+V5L4GwCLC/
```

```
SG3xTl0o="; Path=/; Expires=Thu, 14-Jul-2016 20:27:44 GMT; HttpOnly  
Transfer-Encoding: chunked  
{\"boolean\":true}
```

----End

The Key Management Server (KMS) uses the HTTP REST API to provide key management services for external systems. For details about the API, see <https://hadoop.apache.org/docs/r3.3.1/hadoop-kms/index.html>.

#### NOTE

As REST API reference has done security hardening to prevent script injection attack. Through REST API reference, it cannot create directory and file name which contain those key words "<script ", "<iframe", "<frame", "javascript:".

### 2.7.5.2 Shell Command Introduce

#### HDFS Shell

You can use the Hadoop Distributed File System (HDFS) Shell command to perform operations on the HDFS, such as reading and writing files.

To run the HDFS Shell:

Go to the directory of HDFS client and enter the command. An example is shown as follows:

```
cd /opt/hadoopclient/HDFS/hadoop/bin
```

```
hdfs dfs -mkdir /tmp/input
```

You can run the following command to seek help about HDFS commands:

```
hdfs --help
```

For details about the shell, see

<http://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-common/FileSystemShell.html>

**Table 2-89** Transparent encryption-related commands

Scenario	Operation	Command	Description
hadoop shell command management key	Create keys.	<b><i>hadoop key create &lt;keyname&gt; [-cipher &lt;cipher&gt;] [-size &lt;size&gt;] [-description &lt;description&gt;] [-attr &lt;attribute=value&gt;] [-provider &lt;provider&gt;] [-help]</i></b>	The create subcommand creates a key for the name specified by the <keyname> argument within the provider specified by the -provider argument. You may specify a cipher with the -cipher argument. The default keysize is 128. You may specify the requested key length using the -size argument. Arbitrary attribute=value style attributes may be specified using the -attr argument. The -attr may be specified for multiple times, once per attribute.
	Rollback	<b><i>hadoop key roll &lt;keyname&gt; [-provider &lt;provider&gt;] [-help]</i></b>	The roll subcommand creates a new version for the specified key within the provider indicated using the -provider argument.
	Delete keys	<b><i>hadoop key delete &lt;keyname&gt; [-provider &lt;provider&gt;] [-f] [-help]</i></b>	The delete subcommand deletes all versions of the key specified by the <keyname> argument within the provider specified by the -provider argument. The command asks for user confirmation unless -f is specified.
	View keys	<b><i>hadoop key list [-provider &lt;provider&gt;] [-metadata] [-help]</i></b>	The list subcommand displays the keynames contained in a particular provider as configured in core-site.xml or specified with the -provider argument. The -metadata argument displays the metadata.

**Table 2-90** Shell commands of Colocation client

Operation	Command	Description
Group creation	hdfs colocationadmin -createGroup -groupId <groupId> -locatorIDs <comma separated locatorIDs> or -file <path of the file contains all of locatorIDs>	Used to create a group. In the command, groupId is the group name and locatorID is the locator name. You can enter comma-separated locator IDs using command lines. You can also write locator IDs into a file so that the system can obtain the locator IDs by reading the file.
Group deletion	hdfs colocationadmin -deleteGroup <groupId>	Used to delete the specified group.
Group query	hdfs colocationadmin -queryGroup <groupId>	Used to query details about a specified group, including locators in the group and information about each locator and its corresponding DataNode.
Viewing all groups	hdfs colocationadmin -listGroups	Used to list all groups and their creation time.
Setting ACL permissions on Colocation directories	hdfs colocationadmin -setAcl	Used to set ACL permissions on Colocation directories in ZooKeeper. The default root directory of Colocation in ZooKeeper is <b>/hadoop/colocationDetails</b> .

## 2.8 HetuEngine Development Guide

### 2.8.1 Overview

#### 2.8.1.1 Introduction to HetuEngine

##### Introduction to HetuEngine

HetuEngine is a high-performance interactive SQL analysis and data virtualization engine developed by Huawei. It seamlessly integrates with the big data ecosystem

to implement interactive query of massive amounts of data within seconds, and supports cross-source and cross-domain unified data access to enable one-stop SQL convergence analysis in the data lake, between lakes, and between lakehouses.

### 2.8.1.2 Concepts

#### Basic Concepts

- HSFabric: unified entry for the HetuEngine services. It receives external requests and supports cross-network access to the HetuEngine services.
- HSBroker: service management of the HetuEngine services. It is used for resource management verification, health monitoring, and automatic maintenance of compute instances.
- Coordinator: coordinator of the HetuEngine services. It is responsible for SQL parsing and optimization.
- Worker: compute node of the HetuEngine compute instances. It executes tasks and processes data.
- Connector: an API for HetuEngine to access the database. Through the connector driver, HetuEngine connects to the data sources, reads data source metadata, and operates data (adding, deleting, modification, and query).
- Catalog: the catalog configuration file corresponds to a data source in HetuEngine. A data source can have multiple catalog configurations, which can be configured in the **properties** file of a data source.
- Schema: schema name of the database.
- Table: table name of the database.

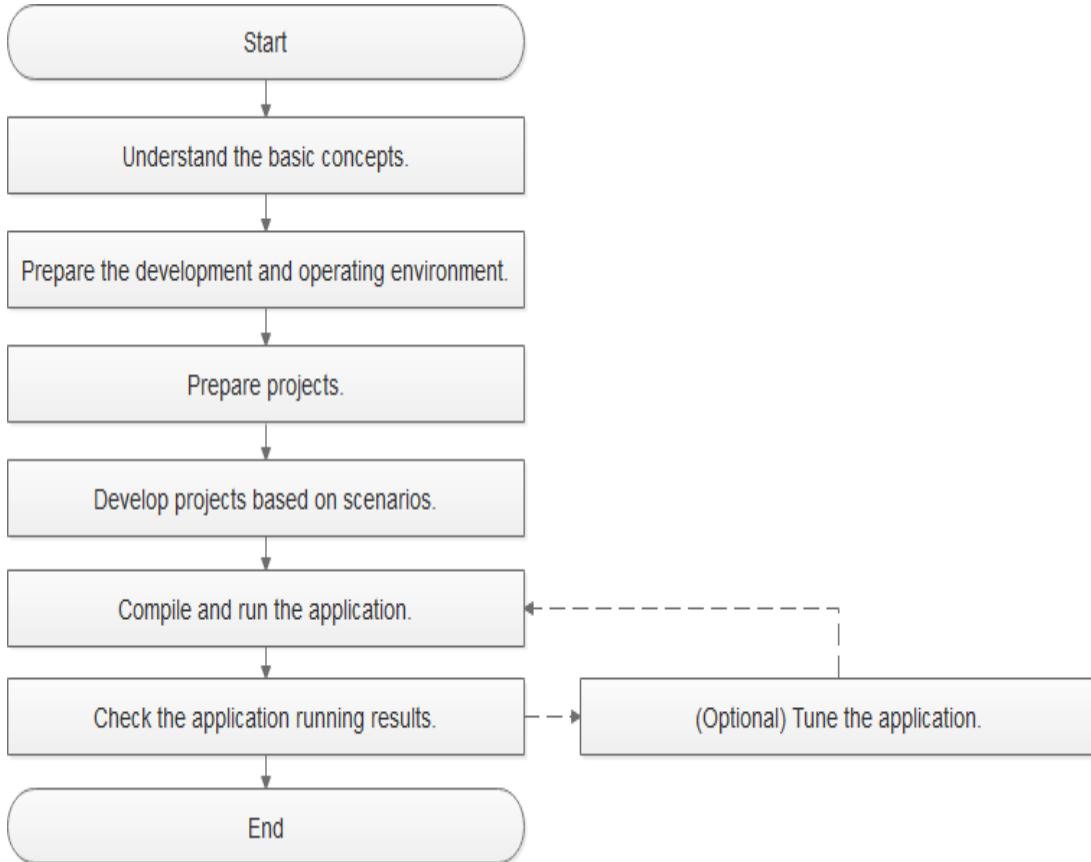
### 2.8.1.3 Connection Modes

Connection Mode	Support for Cross - Network-Segment Client Access	Prerequisite
HSFabric	Yes	<ul style="list-style-type: none"><li>The node running user services can communicate with the service nodes where HSFabric resides on the HetuEngine server side.</li><li>Dual-plane network scenarios are supported.</li><li>Only fixed IP addresses and ports need to be opened for HSFabric.</li><li>Supported version: MRS 3.1.3 or later.</li></ul>
HSBroker	No	<ul style="list-style-type: none"><li>The node running user services can communicate with the service nodes where HSBroker and Coordinator (randomly distributed in Yarn NodeManager) reside on the HetuEngine server side.</li><li>A large number of IP addresses and ports need to be opened for coordinators.</li><li>Supported version: MRS 3.1.0 or later</li></ul>

### 2.8.1.4 Development Process

This section describes the development process, as shown in [Figure 2-154](#).

**Figure 2-154 HetuEngine application development process**



**Table 2-91 Description of the HetuEngine application development process**

Phase	Description	Reference Documents
Understand basic concepts.	Before application development, learn basic concepts of HetuEngine, and understand the scenario requirements.	<a href="#">Concepts</a>
Prepare the development and running environment.	The HetuEngine application can invoke the JDBC interface in any language for development. The current example uses the Java language. You are advised to use the IDEA tool to configure development environments in different languages according to the guide. The HetuEngine running environment is a client. Install and configure the client according to the guide.	<a href="#">Preparing Development and Operating Environment</a>
Prepare a project.	HetuEngine provides sample projects for different scenarios. You can import a sample project to learn the application. You can also create a HetuEngine project according to the guide.	<a href="#">Configuring and Importing Sample Projects</a>

Phase	Description	Reference Documents
Develop a project based on the scenario.	A sample project in the Java language is provided, including an example project that connects the HetuEngine, SQL statement execution, result parsing, and disconnection.	<a href="#">Application Development</a>
Compile and run applications .	You can compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
Check the application running results.	The program running result is displayed in the expected display according to the implementation of the result parsing part.	<a href="#">Application Commissioning</a>

## 2.8.2 Preparing Environment

### 2.8.2.1 Preparing Development and Operating Environment

#### Preparing the Development Environment

This section describes the development and running environment to be prepared for application development, as listed in [Table 2-92](#).

**Table 2-92** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows 7 or later</li><li>• Running environment: Windows or Linux If the program needs to be commissioned locally, the operating environment must be able to communicate with the cluster service plane.</li></ul>

Item	Description
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Basic configuration of the development environment. The version must be 2019.1 or other compatible versions.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If you use the IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If you are using an Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If you are using an OpenJDK, ensure that the JDK configured in IntelliJ IDEA is the OpenJDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA projects.</li></ul>
Maven installation	Basic configuration of the development environment. It can be used for project management throughout the lifecycle of software development.

Item	Description
7-zip	Tool used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

**Table 2-93** Python3 environment (required when the Python sample project is used)

Item	Description
Python3	Tool used to develop HetuEngine Python applications. The version must range from 3.6 to 3.9.
Setuptools	Basic configuration of the Python3 development environment. Version: 47.3.1
jaydebeapi	Basic configuration of the Python3 development environment. You can use this module to connect to the database using Java JDBC.

## Preparing the Operating Environment

During application development, prepare the environment for running and commissioning code to verify that the application can run properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required for commissioning; configure the network connection, and commission the application in Windows.
  - a. Log in to FusionInsight Manager. In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**, select the correct platform type (**x86\_64** for x86 and **aarch64** for ARM) based on the platform type of the node where the client is to be installed, and click **OK**. After the client file package is generated, download it to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\HetuEngine\config** and manually import the configuration file to the configuration file directory (for example, **D:\hetuclient\conf**) of the HetuEngine sample project.  
**Table 2-94** describes the main configuration files. (Obtain the required files you need.)

**Table 2-94** Configuration files

File	Description
hdfs-site.xml	HDFS parameters
hetuserver-client.properties	HetuEngine client connection parameters
hetuserver-client-logging.properties	HetuEngine client log parameters

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, install the client in the **/opt/hadoopclient** directory.  
The difference between the client time and the cluster time must be less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then log in to the active management node as user **root**. Go to the decompression directory of the cluster client, and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/HetuEngine/config** directory to the client node to the **conf** directory where the compiled JAR package is stored, for example, **/opt/hadoopclient/conf**, for subsequent commissioning.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active OMS node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig
```

```
scp HetuEngine/config/* root@IP address of the client node:/opt/
hadoopclient/conf
```

**Table 2-95** describes the main configuration files. (Obtain the required files you need.)

**Table 2-95** Configuration files

File	Description
hdfs-site.xml	HDFS parameters
hetuserver-client.properties	HetuEngine client connection parameters
hetuserver-client-logging.properties	HetuEngine client log parameters

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

### 2.8.2.2 Configuring and Importing Sample Projects

#### Scenario

The HetuEngine client installation program directory contains a HetuEngine development sample project. You can start the sample project learning by importing the project. This document uses IntelliJ IDEA 2020.1.3 (Community Edition) as an example.

#### Prerequisites

Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the time of the MRS cluster in the upper-right corner on the FusionInsight Manager page.

#### Procedure

- Step 1** Obtain the sample project folder **hetu-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. Open IntelliJ IDEA and choose **File > New > General > Project from Existing Sources > Select File or Directory to Import** to go to the **Browse Folder** dialog box is displayed.
2. Select the sample project folder, choose **Import project from external model > Maven** during import, and click **Next** and then **Finish**.

 **NOTE**

The sample code is a Maven project. You can adjust the project configuration based on the site requirements.

----End

### 2.8.2.3 Setting Up a Python3 Project

#### Scenario

The following content describes the operations you need to do to run the python3 sample code of HetuEngine on FusionInsight MRS.

#### Procedure

- Step 1** Install Python 3.6 or a later version (before 3.9) on the client node.

The Python version can be viewed by running the **python3** command on the CLI of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

- Step 2** Setuptools 47.3.1 must be installed on the client.

Download the software from the official website <https://pypi.org/project/setuptools/#files>.

Copy the downloaded setuptools package to the client, decompress the package, go to the setuptools project directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```

 **NOTE**

If the system displays a message indicating that setuptools 47.3.1 fails to be installed, check whether the environment is faulty or Python is faulty.

- Step 3** The JayDeBeApi module must be installed on the client. You can use the Java JDBC to connect to the database through this module.

You can install it either of the following ways:

- pip:  
Run the **pip install JayDeBeApi** command on the client node.
- Script:
  - a. Download the **JayDeBeApi project** file from the official website at <https://pypi.org/project/JayDeBeApi/>

- b. Go to the **JayDeBeApi** project directory and run the **python3 setup.py install** command. During the installation, if the system displays a message indicating that the python3 module or package is missing, you need to add the module or package.

Take JayDeBeApi-1.2.3 as an example. If **Successfully installed JayDeBeApi-1.2.3** is displayed, the installation is successful.

**Step 4** Install Java on the client. For details about the supported Java versions, see "JDK Installation" in the [Table 1-105](#).

**Step 5** Obtain the Python3 sample code.

1. Obtain the sample project folder **python3-examples** in the **src\hetu-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
  - **normal** folder: Python3 sample code for interconnecting with HetuEngine in common mode
  - **security** folder: Python3 sample code for interconnecting with HetuEngine in security mode

**Step 6** Obtain the **hetu-jdbc** JAR package.

- Download the client file to the local PC through Manager.
  - a. Log in to FusionInsight Manager and choose **Cluster > Services > HetuEngine > More > Download Client**.
  - b. Select **Complete Client**, select the correct platform type (**x86\_64** for x86 and **aarch64** for ARM) for the node where the client is to be installed, deselect **Save to Path**, and click **OK**. Wait until the client file package is generated and download it.
  - c. Decompress the downloaded software package to obtain the **hetu-jdbc** package and decompress it to obtain the JDBC package.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress the package to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then decompress the package to obtain the **FusionInsight\_Cluster\_1\_Services\_ClientConfig** file. (The path cannot contain spaces.) Decompress **\FusionInsight\_Cluster\_1\_Services\_ClientConfig\HetuEngine\x86\hetu-jdbc.tar.gz** to obtain **hetu-jdbc-XXX.jar** and copy it to the user-defined path on the host where the sample code will run.

- Obtain the cluster client node.  
Log in to the node where the HetuEngine client has been installed. For example, if the client installation path is **/opt/hadoopclient**, obtain **hetu-jdbc-XXX.jar** from **/opt/hadoopclient/HetuEngine/hetuserver/jars/**, and copy the file to the user-defined path on the node where the sample code will run.

----End

## 2.8.2.4 Setting Up a Spring Boot Sample Project

### Scenario

This topic describes how to run the Spring Boot sample code of the HetuEngine component in MRS.

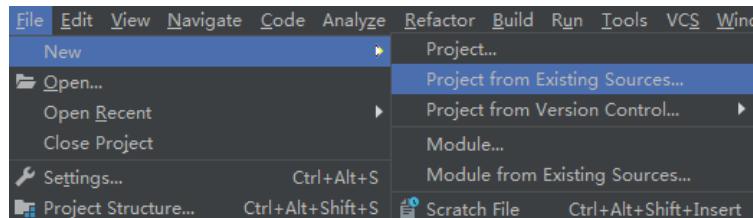
This example develops an application for connecting to the HetuEngine service using Spring Boot in a Windows environment.

### Procedure

**Step 1** Obtain the **src/springboot/hetu-examples** directory in the directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** In the application development environment, import the sample project to IntelliJ IDEA.

1. Choose **File > New > Project from Existing Sources....**



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **hetu-examples** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.

Select the recommended JDK version and click **Finish**.

----End

## 2.8.3 Application Development

### 2.8.3.1 Typical Application Scenario

You can quickly learn and master the HetuEngine development process and know key interface functions in a typical application scenario.

### Scenario

Assume that a user develops an application and needs to perform the join operation on table A of the Hive data source and table B of the MPPDB data source. In this case, HetuEngine can be used to implement data query of the Hive data source, the process is as follows:

1. Connect to a HetuEngine JDBC server.
2. Assemble SQL statements.
3. Execute SQL statements.

4. Parse the returned result.
5. Disconnect HetuEngine JDBC server.

### 2.8.3.2 Java Sample Code

#### 2.8.3.2.1 Accessing Hive Data Sources Using HSFabric

##### Description

This section describes how to use HSFabric to connect to HetuEngine, assemble SQL statements, and send the SQL statements to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
public class JDBCExampleFabric {  
    private static Properties properties = new Properties();  
    private static void init() throws ClassNotFoundException {  
        properties.setProperty("User", "YourUserName");  
        properties.setProperty("SSL", "false");  
        Class.forName("io.trino.jdbc.TrinoDriver");  
    }  
    /**  
     * Program entry  
     *  
     * @param args no need program parameter  
     */  
    public static void main(String[] args) {  
        Connection connection = null;  
        ResultSet result = null;  
        PreparedStatement statement = null;  
        String url = "jdbc:trino://192.168.1.130:29903,192.168.1.131:29903/hive/default?  
serviceDiscoveryMode=hsfabric";  
        try {  
            init();  
            String sql = "show tables";  
            connection = DriverManager.getConnection(url, properties);  
            statement = connection.prepareStatement(sql.trim());  
            result = statement.executeQuery();  
            ResultSetMetaData resultMetaData = result.getMetaData();  
            Integer colNum = resultMetaData.getColumnCount();  
            for (int j = 1; j <= colNum; j++) {  
                System.out.print(resultMetaData.getColumnLabel(j) + "\t");  
            }  
            System.out.println();  
            while (result.next()) {  
                for (int j = 1; j <= colNum; j++) {  
                    System.out.print(result.getString(j) + "\t");  
                }  
                System.out.println();  
            }  
        } catch (SQLException | ClassNotFoundException e) {  
            e.printStackTrace();  
        } finally {  
            if (result != null) {  
                try {  
                    result.close();  
                } catch (SQLException e) {  
                    e.printStackTrace();  
                }  
            }  
            if (statement != null) {  
                try {  
                    statement.close();  
                } catch (SQLException e) {  
                    e.printStackTrace();  
                }  
            }  
        }  
    }  
}
```

```
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

**Table 2-96** describes the parameters in the preceding code.

**Table 2-96** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSFabric1_IP.HSFabric1_Port,HSFabric2_IP.HSFabric2_Port,</i> <i>HSFabric3_IP.HSFabric3_Port/catalog/schema?</i> serviceDiscoveryMode=hsfabric</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <i>catalog</i> and <i>schema</i> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <i>HSFabric_IP:HSFabric_Port</i>: indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29903,192.168.195.232:29903,192.168.169.84:29903. On FusionInsight Manager, choose <b>Cluster &gt; Services &gt; HetuEngine</b> and click <b>Instance</b> to obtain the service IP addresses of all HSFabric instances. On the <b>Configurations</b> page, search for <b>gateway.port</b> to obtain the port number of HSFabric.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the machine-machine user created in the cluster.
SSL	Indicates whether to use the HTTPS connection. The default value is <b>false</b> .

### 2.8.3.2.2 Accessing Hive Data Sources Using HSBroker

#### Description

This section describes how to use HSBroker to connect to HetuEngine, assemble SQL statements, and send the SQL statements to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
public class JDBCExampleBroker {
    private static Properties properties = new Properties();
```

```
private static void init() throws ClassNotFoundException {
    properties.setProperty("user", "YourUserName");
    properties.setProperty("SSL", "false");
    Class.forName("io.trino.jdbc.TrinoDriver");
}
/**
 * Program entry
 *
 * @param args no need program parameter
 */
public static void main(String[] args) {
    Connection connection = null;
    ResultSet result = null;
    PreparedStatement statement = null;
    String url = "jdbc:trino://192.168.1.130:29861,192.168.1.131:29861/hive/default?
serviceDiscoveryMode=hsbroker";
    try {
        init();
        String sql = "show tables";
        connection = DriverManager.getConnection(url, properties);
        statement = connection.prepareStatement(sql.trim());
        result = statement.executeQuery();
        ResultSetMetaData metaData = result.getMetaData();
        Integer colNum = metaData.getColumnCount();
        for (int j = 1; j <= colNum; j++) {
            System.out.print(resultMetaData.getColumnLabel(j) + "\t");
        }
        System.out.println();
        while (result.next()) {
            for (int j = 1; j <= colNum; j++) {
                System.out.print(result.getString(j) + "\t");
            }
            System.out.println();
        }
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        if (result != null) {
            try {
                result.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

**Table 2-97** describes the parameters in the preceding code.

**Table 2-97** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSBroker1_IP.HSBroker1_Port,HSBroker2_IP.HSBroker2_Port,HSBroker3_IP.HSBroker3_Port/catalog/schema?</i> serviceDiscoveryMode=hsbroker</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <i>catalog</i> and <i>schema</i> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <i>HSBroker_IP:HSBroker_Port</i>: indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29861,192.168.195.232:29861,192.168.169.84:29861.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the machine-machine user created in the cluster.
SSL	Indicates whether to use the HTTPS connection. The default value is <b>false</b> .

### 2.8.3.2.3 Querying the Execution Progress and Status of an SQL Statement Using JDBC

#### Description

This section describes how to use JDBC to connect to HetuEngine, and assemble and send the SQL statements to HetuEngine for execution.

```
import io.trino.jdbc.TrinoResultSet;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Properties;
import java.util.Timer;
import java.util.TimerTask;

public class JDBCExampleStatementProgressPercentage{

    private static Properties properties = new Properties();
    public static Connection connection = null;
    public static ResultSet result = null;
    public static PreparedStatement statement = null;
    private static void init() throws ClassNotFoundException {
        properties.setProperty("user", "YourUserName");
        properties.setProperty("SSL", "false");
        Class.forName("io.trino.jdbc.TrinoDriver");
```

```
}

/**
 * Program entry
 *
 * @param args no need program parameter
 */
public static void main(String[] args) {
    String url = "jdbc:trino://192.168.1.130:29861,192.168.1.131:29861/hive/default?
serviceDiscoveryMode=hsbroker";
    try {
        init();
        String sql = "show tables";
        connection = DriverManager.getConnection(url, properties);
        statement = connection.prepareStatement(sql.trim());
        result = statement.executeQuery();

        TrinoResultSet rs = (TrinoResultSet) result;
        new Thread() {
            public void run() {
                Timer timer = new Timer();
                //The SQL statement is executed after 3 seconds and is executed every 2 seconds
                timer.schedule(new TimerTask() {
                    @Override
                    public void run() {
                        double statementProgressPercentage = rs.getProgressPercentage().orElse(0.0);
                        System.out.println("The Current Query Progress Percentage is " +
statementProgressPercentage*100 + "%");
                        if("FINISHED".equals(rs.getStatementStatus().orElse("")))) {
                            System.out.println("The Current Query Progress Percentage is 100%");
                            timer.cancel();
                            Thread.currentThread().interrupt();
                        }
                    }
                }, 3000, 2000);
            }
        }.start();

        ResultSetMetaData resultMetaData = result.getMetaData();
        int colNum = resultMetaData.getColumnCount();
        for (int j = 1; j <= colNum; j++) {
            try {
                System.out.print(resultMetaData.getColumnLabel(j) + "\t");
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        }

        while (result.next()) {
            for (int j = 1; j <= colNum; j++) {
                System.out.print(result.getString(j) + "\t");
            }
            System.out.println();
        }
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        if (result != null) {
            try {
                result.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
        }
    }
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

**Table 2-98** describes the parameters in the preceding code.

**Table 2-98** Parameter description

Parameter	Description
url	<code>jdbc:trino:// HSBroker1_IP.HSBroker1_Port,HSBroker2_IP.HSBroker2_Port,HSBroker3_IP.HSBroker3_Port/catalog/schema? serviceDiscoveryMode=hsbroker</code> <b>NOTE</b> <ul style="list-style-type: none"><li>• <i>catalog</i> and <i>schema</i> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <i>HSBroker_IP:HSBroker_Port</i>: indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29861,192.168.195.232:29861,192.168.169.84:29861.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the machine-machine user created in the cluster.
SSL	Indicates whether to use the HTTPS connection. The default value is <b>false</b> .
getStatementStatus()	Execution status of an SQL statement, which can be <b>RUNNING</b> , <b>FAILED</b> , <b>FINISHED</b> , <b>QUEUEED</b> , <b>WAITING_FOR_RESOURCES</b> , <b>DISPATCHING</b> , <b>PLANNING</b> , <b>STARTING</b> , <b>RESCHEDULING</b> , <b>RESUMING</b> , or <b>FINISHING</b> .
getProgressPercentage()	Execution progress of an SQL statement. The value range is 0 to 1.

### 2.8.3.3 Python3 Sample Code

### 2.8.3.3.1 Accessing Hive Data Sources Using HSBroker

This section describes how to use HSBroker to connect to HetuEngine, assemble SQL statements, and send them to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
import jaydebeapi  
  
driver = "io.trino.jdbc.TrinoDriver"  
  
# need to change the value based on the cluster information  
url = "jdbc:trino://192.168.37.61:29861,192.168.37.62:29861/hive/default?serviceDiscoveryMode=hsbroker"  
user = "YourUserName"  
tenant = "YourTenant"  
jdbc_location = "Your file path of the jdbc jar"  
  
sql = "show catalogs"  
  
if __name__ == '__main__':  
    conn = jaydebeapi.connect(driver, url, {"user": user,  
                                         "SSL": "false",  
                                         "tenant": tenant},  
                           [jdbc_location])  
    curs = conn.cursor()  
    curs.execute(sql)  
    result = curs.fetchall()  
    print(result)  
    curs.close()  
    conn.close()
```

The following table describes the parameters in the preceding code.

**Table 2-99** Parameter description

Parameter	Description
url	<p><code>jdbc:trino:// HSBroker1_IP.HSBroker1_Port,HSBroker2_IP.HSBroker2_Port,HSB roker3_IP.HSBroker3_Port/catalog/schema?serviceDiscovery- Mode=hsbroker</code></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <i>catalog</i> and <i>schema</i> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <i>HSBroker_IP:HSBroker_Port</i> indicates the HSBroker URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29860,192.168.195.232:29860,192.168.169.84:29860.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances

Parameter	Description
jdbc_location	Full path of the <b>hetu-jdbc-XXX.jar</b> package obtained in <a href="#">Setting Up a Python3 Project</a> . <ul style="list-style-type: none"><li>• Example path in Windows: <b>D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar</b></li><li>• Example path in Linux: <b>/opt/hetu-examples-python3/hetu-jdbc-XXX.jar</b></li></ul>

### 2.8.3.3.2 Accessing Hive Data Sources Using HSFabric

This section describes how to use HSFabric to connect to HetuEngine, assemble SQL statements, and send them to HetuEngine for execution to add, delete, modify, and query Hive data sources.

```
import jaydebeapi
driver = "io.trino.jdbc.TrinoDriver"

# need to change the value based on the cluster information
url = "jdbc:trino://192.168.37.61:29903,192.168.37.61:29903/hive/default?serviceDiscoveryMode=hsfabric"
user = "YourUserName"
tenant = "YourTenant"
jdbc_location = "Your file path of the jdbc jar"

sql = "show catalogs"

if __name__ == '__main__':
    conn = jaydebeapi.connect(driver, url, {"user": user,
                                             "SSL": "false",
                                             "tenant": tenant},
                             [jdbc_location])
    curs = conn.cursor()
    curs.execute(sql)
    result = curs.fetchall()
    print(result)
    curs.close()
    conn.close()
```

The following table describes the parameters in the preceding code.

**Table 2-100** Parameter description

Parameter	Description
url	<p>jdbc:trino:// <i>HSFabric1_IP.HSFabric1_Port,HSFabric2_IP.HSFabric2_Port,HSFabric3_IP.HSFabric3_Port/catalog/schema?serviceDiscovery-Mode=hsfabric</i></p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <b>catalog</b> and <b>schema</b> indicate the names of the catalog and schema to be connected to the JDBC client, respectively.</li><li>• <b>HSFabric_IP:HSFabric_Port</b> indicates the HSFabric URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:29902,192.168.195.232:29902,192.168.169.84:29902. On FusionInsight Manager, choose <b>Cluster &gt; Services &gt; HetuEngine</b> and click <b>Instance</b> to obtain the service IP addresses of all HSFabric instances. On the <b>Configurations</b> page, search for <b>gateway.port</b> to obtain the port number of HSFabric.</li><li>• (Optional) <b>auditAddition</b>: custom information about the audit log. This field is recorded in the audit log. This parameter is used to supplement information for user audit.<ul style="list-style-type: none"><li>• Name of the image. The name can contain 1 to 255 characters.</li><li>• Only letters, digits, underscores (_), commas (,), and colons (:) are allowed.</li></ul></li></ul>
user	Username for accessing HetuEngine, that is, the username of the human-machine user created in the cluster.
tenant	Tenant resource queue for accessing HetuEngine compute instances
jdbc_location	<p>Full path of the <b>hetu-jdbc-XXX.jar</b> package obtained in <a href="#">Setting Up a Python3 Project</a>.</p> <ul style="list-style-type: none"><li>• Example path in Windows: <b>D:\\hetu-examples-python3\\hetu-jdbc-XXX.jar</b></li><li>• Example path in Linux: <b>/opt/hetu-examples-python3/hetu-jdbc-XXX.jar</b></li></ul>

## 2.8.4 Application Commissioning

### 2.8.4.1 Commissioning Applications on Windows

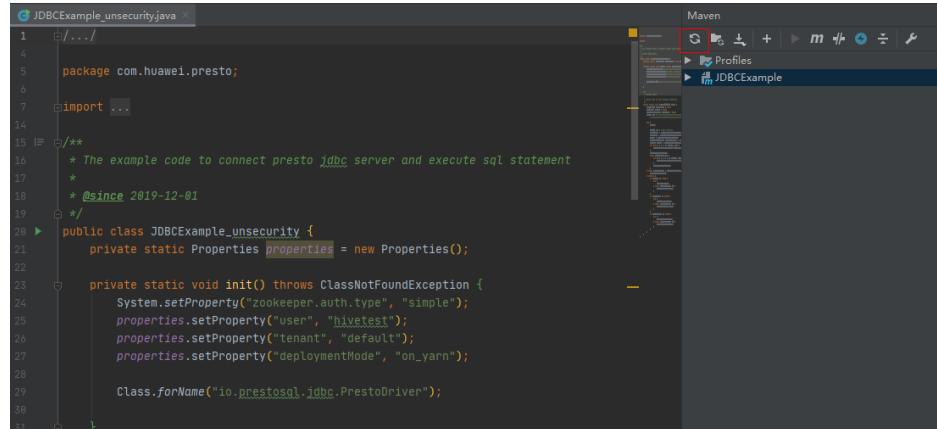
#### Scenario

After the program code is developed, you can compile the program in the Windows environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

## Procedure

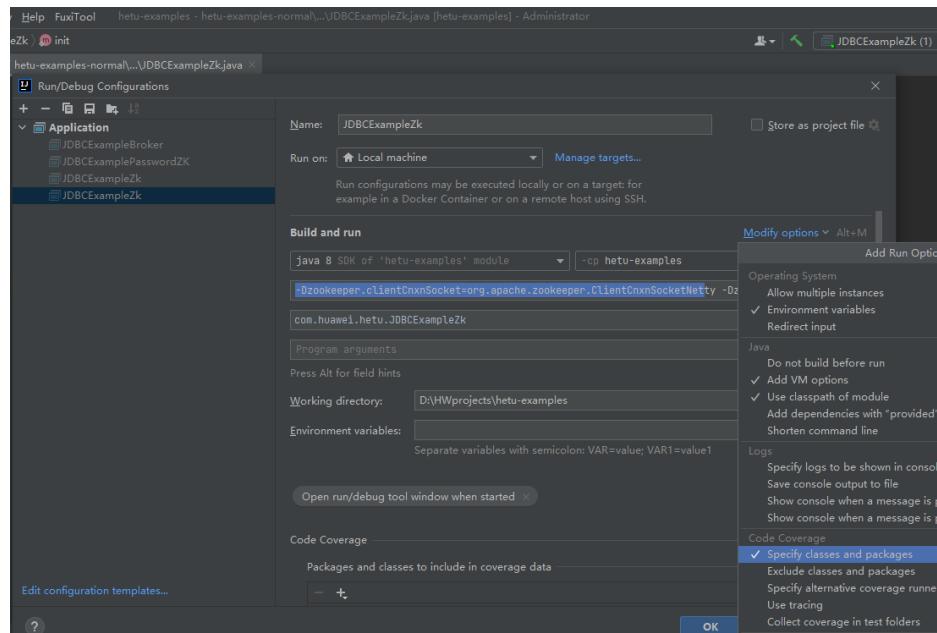
**Step 1** In the IntelliJ IDEA development environment on Windows, click **Maven** on the right of IDEA to import the dependency.

**Figure 2-155** Importing the dependency



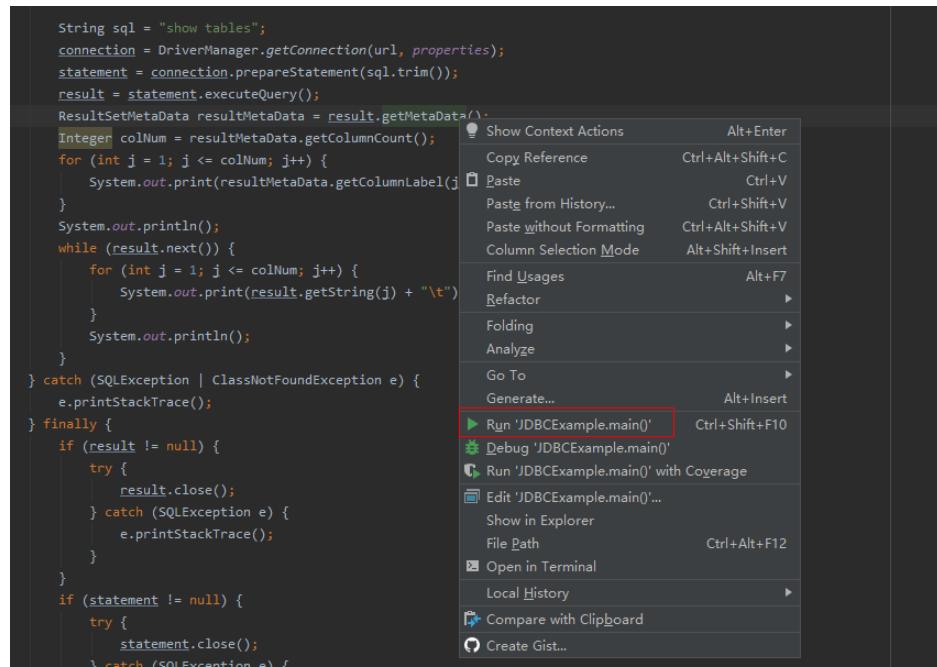
**Step 2** (Optional) If the SSL authentication communication function of ZooKeeper is enabled for the interconnected cluster, add the JVM configuration

**-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true.**



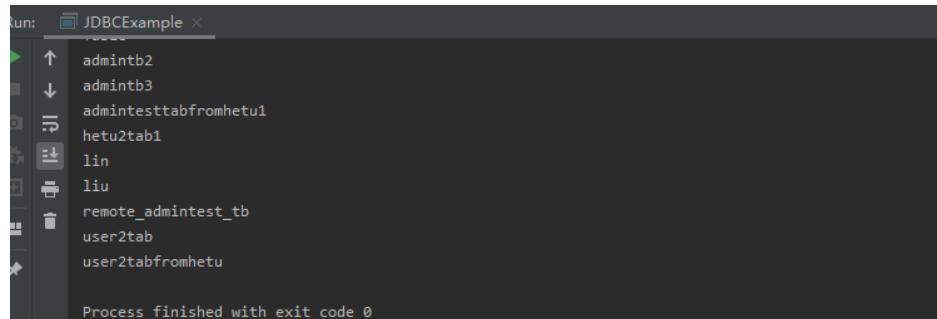
**Step 3** Right-click the **JDBCExampleZK.java** file and choose **Run'JDBCExampleZK.main()'** from the shortcut menu.

Figure 2-156 Running the program



You can view the output result on the console of the IDEA.

Figure 2-157 Outputs



----End

## 2.8.4.2 Commissioning Applications on Linux

### Scenario

After the code is developed, you can compile the code into a JAR file and upload it to a Linux environment for application function commissioning.

#### NOTE

Before commissioning applications on Linux, you need to pre-install a client on the Linux node.

## Procedure

- Step 1** In the Windows development environment IntelliJ IDEA, choose **Maven Projects > JDBCExample > Lifecycle** and perform the **clean** and **package** operations. After the compilation is complete, the **JDBCExample-1.0-SNAPSHOT.jar** file is generated in the target directory.
- Step 2** Upload the **JDBCExample-1.0-SNAPSHOT.jar** package to the Linux client installation directory, for example, **/opt/hadoopclient**.
- Step 3** Download and decompress the client file **FusionInsight\_Cluster\_Cluster ID\_HetuEngine\_Client.tar** by referring to [Preparing the Operating Environment](#), obtain the JDBC driver package, and upload it to the **/opt/hadoopclient** directory in the Linux environment.

### NOTE

To obtain the JDBC driver package:

Obtain the **hetu-jdbc-\*.jar** file from the **FusionInsight\_Cluster\_Cluster ID\_HetuEngine\_ClientConfig\HetuEngine\xxx\** directory.

Note: **xxx** can be **arm** or **x86**.

- Step 4** Run the following command to go to the client installation directory:

```
cd /opt/hadoopclient;
```

- Step 5** Run the following command to configure environment variables:

```
source bigdata_env;
```

- Step 6** Run the following command to debug the development program:

```
java -classpath JDBCExample-*.jar:hetu-jdbc-*.jar com.huawei.hetu.className
```

### NOTE

Replace the JDBC driver package name and class name with the actual ones, for example,  
**java -classpath JDBCExample-\*.jar:hetu-jdbc-\*.jar com.huawei.hetu.JDBCExampleZk**.

- Step 7** Check whether the command output is normal.

.....

Table  
testtable

----End

## 2.8.4.3 Commissioning the Python3 Sample Project

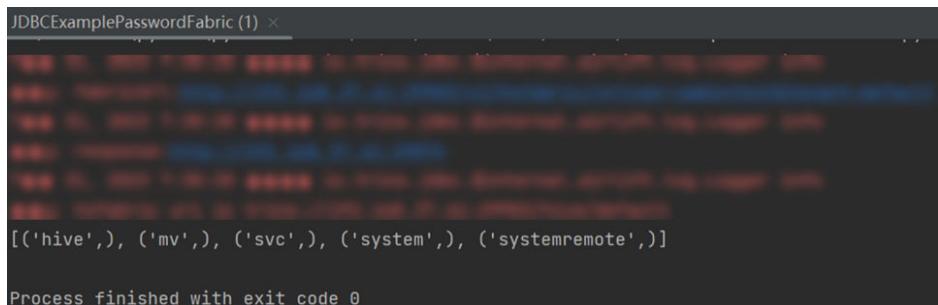
### Scenario

After the program code is written, you can commission the code in the Windows environment or upload the code to the Linux environment. If the local environment can communicate with the cluster service plane network, you can commission the program locally.

## Procedure

1. Refer to [Setting Up a Python3 Project](#) to obtain the sample code, obtain the **hetu-jdbc-XXX.jar** file, and copy it to the user-defined directory.

2. Edit the sample code, modify URLs and user information based on the cluster requirements, and modify **jdbc\_location** based on the actual path.
  - Example path in Windows: **D:\hetu-examples-python3\hetu-jdbc-XXX.jar**
  - Example path in Linux: **/opt/hetu-examples-python3/hetu-jdbc-XXX.jar**
3. Run the python3 sample code.
  - In Windows, run the **py** script through pycharm or Python IDLE.
  - To run the sample code on Linux, install Java first.  
Go to the sample code path and run the **py** script. An example of the sample code path is **/opt/hetu-examples-python3**.  
**cd /opt/hetu-examples-python3  
python3 JDBCExampleBroker.py**
4. View the command output.
  - In Windows, view the running result on the console.

**Figure 2-158** Running result

```
JDBCExamplePasswordFabric (1) ×
[('hive',), ('mv',), ('svc',), ('system',), ('systemremote',)]
Process finished with exit code 0
```

- The following is an example of the running result in Linux:  
...  
[ ('hive', ('mv'), ('svc'), ('system'), ('systemremote'))]

#### 2.8.4.4 Commissioning a Spring Boot Application

**Step 1** Configure the properties required for connecting the Spring Boot sample program to HetuEngine. Set the following parameters in the **springboot\hetu-examples\src\main\resources\application.yml** configuration file:

**Table 2-101** Configuring HetuEngine

Parameter	Mandatory	Description
host	Yes	HS Fabric address list of the HetuEngine service. The format is <i>HSFabric_IP1:HSFabric_Port,HSFabric_IP2:HSFabric_Port,HSFabric_IP3:HSFabric_Port</i> . Log in to FusionInsight Manager, choose <b>Cluster &gt; Services &gt; HetuEngine &gt; Instances</b> , and obtain the service IP addresses of all HSFabric instances. In the <b>Configurations</b> tab, search for <b>gateway.port</b> to obtain the HSFabric port.

Parameter	Mandatory	Description
catalog	Yes	Catalog name of the HetuEngine compute instance
schema	Yes	Schema name of the HetuEngine compute instance
user	Yes	Username for connecting to the HetuEngine compute instance
password	No	Password of the user for connecting to the HetuEngine compute instance in a cluster in security mode
ssl	Yes	Whether to use SSL to connect to the HetuEngine compute instance <ul style="list-style-type: none"> <li>● For a cluster in security mode, set this parameter to <b>true</b>.</li> <li>● For a cluster in normal mode, set this parameter to <b>false</b>.</li> </ul>

**Step 2** Click **Terminal** on the lower left part of the IDEA page to access the terminal. Run the following command to compile the file:

**mvn clean package**

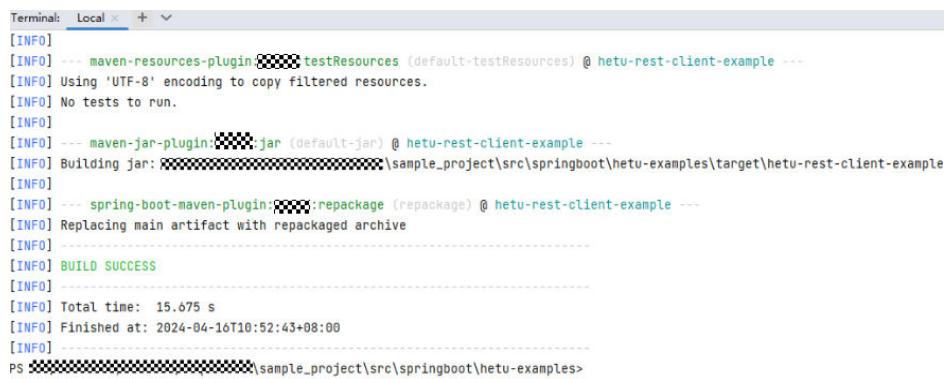
**Figure 2-159** Compile command



```
Terminal: Local + 
PowerShell
PS C:\sample_project\src\springboot\hetu-examples> mvn clean package
```

If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR package containing the **hetu-rest-client-example** field is generated in the target directory of the sample project.

**Figure 2-160** Successful compilation



```
[INFO]
[INFO] --- maven-resources-plugin:testResources (default-testResources) @ hetu-rest-client-example ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:jar (default-jar) @ hetu-rest-client-example ---
[INFO] Building jar: C:\sample_project\src\springboot\hetu-examples\target\hetu-rest-client-example.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:repackage (repackage) @ hetu-rest-client-example ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 15.675 s
[INFO] Finished at: 2024-04-16T10:52:43+08:00
[INFO]
PS C:\sample_project\src\springboot\hetu-examples>
```

**Step 3** Create a directory in the system as the run directory to save the JAR package that contains the **hetu-rest-client-example** field generated in **Step 2**.

- Create a directory on Windows, for example, D:\hetu-rest-client-example.
- Create a directory on Linux, for example, /opt/hetu-rest-client-example.

**Step 4** Run the following command to start the SpringBoot service:

- Commands for Windows:

```
cd /d d:\hetu-rest-client-example  
java -jar hetu-rest-client-example-*-.SNAPSHOT.jar
```

- Commands for Linux:

```
chmod +x /opt/hetu-rest-client-example -R  
cd /opt/hetu-rest-client-example  
java -jar hetu-rest-client-example-*-.SNAPSHOT.jar
```

#### NOTE

The preceding JAR package names are for reference only. Use the actual names.

**Step 5** Call a Spring Boot sample API of HetuEngine to run the sample code.

- For the Windows environment:

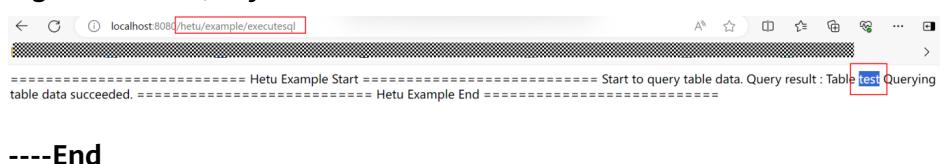
Open a browser and enter **http://localhost:8080/hetu/example/executesql** in the address box.

- For the Linux environment:

Run the **curl http://localhost:8080/hetu/example/executesql** command on the node where the JAR package is stored in **Step 3**.

**Step 6** View the query result of the SQL statement used in the sample code.

**Figure 2-161** Query results



## 2.9 Hive Development Guide

### 2.9.1 Overview

#### 2.9.1.1 Application Development Overview

##### Hive Introduction

Hive is an open-source data warehouse built on Hadoop. It stores structured data and provides basic data analysis services using the Hive query language (HQL), a language like the SQL. Hive converts HQL statements to MapReduce or Spark jobs for querying and analyzing massive data stored in Hadoop clusters.

Hive provides the following features:

- Extracts, transforms, and loads (ETL) data using HQL.
- Analyzes massive structured data using HQL.
- Supports flexible data storage formats, including JavaScript object notation (JSON), comma separated values (CSV), TextFile, RCFile, ORCFILE, and SequenceFile, and supports custom extensions.
- Multiple client connection modes. Interfaces, such as JDBC and Thrift interfaces are supported.

Hive applies to offline massive data analysis (such as log and cluster status analysis), large-scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

To ensure Hive high availability (HA), user data security, and service access security, MRS incorporates the following features based on Hive 3.1.0:

- Data file encryption

For Hive features in the Open Source Community, see <https://cwiki.apache.org/confluence/display/hive/designdocs>.

### 2.9.1.2 Common Concepts

- **Client**

Users can access the server from the client through the Java API and Thrift API to perform Hive-related operations.

- **HQL**

Similar to SQL

- **HCatalog**

HCatalog is a table information management layer created based on Hive metadata and absorbs DDL commands of Hive. HCatalog provides read/write interfaces for MapReduce and provides Hive command line interfaces (CLIs) for defining data and querying metadata. Hive and MapReduce development personnel can share metadata based on the HCatalog component of MRS, preventing intermediate conversion and adjustment and improving the data processing efficiency.

- **WebHCat**

WebHCat running users use Rest APIs to perform operations, such as running Hive DDL commands, submitting MapReduce tasks, and querying MapReduce task execution results.

### 2.9.1.3 Development Process

[Figure 2-162](#) and [Table 2-102](#) illustrates each phase of the development process.

Figure 2-162 Hive application development process

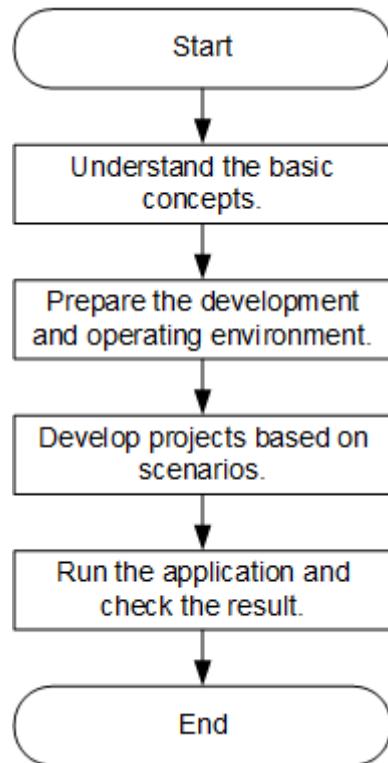


Table 2-102 Description of the Hive application development process

Phase	Description	Reference Document
Understand the basic concepts.	Before application development, understand common concepts about Hive.	<a href="#">Common Concepts</a>
Prepare the development and operating environment.	Hive applications support Java and Python development languages. You are advised to use the IntelliJ IDEA tool to configure the development environment based on the language.	<a href="#">Preparing the Environment</a>
Develop projects based on scenarios.	Provides example projects of the Java and Python languages as well as example projects covering table creation, data load, and data query.	<a href="#">Developing an Application</a>

Phase	Description	Reference Document
Run the application and view results.	Describes how to submit a developed application for compiling and view the result.	<a href="#">Commissioning Applications</a>

## 2.9.2 Preparing the Environment

### 2.9.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

Hive applications can be developed by using the JDBC/Metastore/Python/Python3 API. The following table describes the development and operating environment to be prepared.

**Table 2-103** JDBC/Metastore development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>

Item	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>Tool used for developing Hive applications. Requirements are as follows:</p> <p>JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li></ul>
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. 7-Zip 16.04 is supported.

**Table 2-104** Python development environment

Item	Description
OS	Development and operating environment: Linux OS
Python installation	Python is a tool used to develop Hive applications. Its version must be 2.6.6 or later but should not be later than 2.7.13.
setuptools installation	Basic configuration for the Python development environment. The version must be 5.0 or later.

 NOTE

For details about how to install and configure the Python development tool, see the [Configuring the Python Sample Project](#).

**Table 2-105** Python3 development environment

Item	Description
OS	Development and operating environment: Linux OS
Python3 installation	Python3 is a tool used to develop Hive applications. Its version must be 3.6 or later but should not be later than 3.9.
setuptools installation	Basic configuration for the Python3 development environment. The version must be 47.3.1.

 NOTE

For details about how to install and configure the Python3 development tool, see the [Configuring the Python3 Sample Project](#).

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. Log in to the FusionInsight Manager, click **Download Client** in the upper right corner of the Homepage, set **Select Client Type** to **Configuration Files Only** and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain

**FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file.

Then, decompress

**FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file to the D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path

**FusionInsight\_Cluster\_1\_Services\_ClientConfig\_Hive\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Hive sample project.

The keytab file obtained during the [Preparing a Developer Account](#) is also stored in this directory. **Table 2-106** describes the main configuration files.

**Table 2-106** Configuration file

Document Name	Function
hive metastore-site.xml	Configures Hive parameters.
hiveclient.properties	Configures Hive parameters.
user.keytab	Provides Hive user information for Kerberos security authentication.
krb5.conf	Contains Kerberos server configuration information.
core-site.xml	Configures Hive parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

#### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
  - The local **hosts** file in a Windows environment is stored in, for example, C:\WINDOWS\system32\drivers\etc\hosts.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is /opt/hadoopclient.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. [Accessing FusionInsight Manager of an MRS Cluster](#). Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**.

Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Hive/config** directory to the **src/main/resources** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/src/main/resources**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Hive/config/* root@IP address of the client node:/opt/  
hadoopclient/src/main/resources
```

**Table 2-107** Configuration files

File	Function
hive metastore-site.xml	Configures Hive parameters.
hive-site.xml	Configures Hive parameters.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

### 2.9.2.2 Configuring the JDBC Sample Project

#### Scenario

To run the JDBC API example codes of the Hive component of MRS, perform the following operations.



The following uses the development of an application for connecting the Hive service in JDBC mode on Windows as an example.

#### Procedure

- Step 1** Obtain the sample project folder **hive-jdbc-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** Go to the client decompression path

**FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\Hive\config** and manually import the **core-site.xml** and **hiveclient.properties** files to the **hive-jdbc-example\src\main\resources** directory of the sample project.

 NOTE

Hive allows you to add extension identifiers to JDBC connection strings. These extension identifiers are printed in HiveServer audit logs to distinguish SQL sources. If you need to configure an identifier, add the following content to the **hive-jdbc-example\src\main\resources\hiveclient.properties** file:

**auditAddition=xxx**

xxx is the custom identifier. The identifier can contain a maximum of 256 bytes and only letters, digits, underscores (\_), commas (,), and colons (:) are allowed.

After the SQL statement is executed, the configured extension identifier is printed in the HiveServer audit log **/var/log/Bigdata/audit/hive/hiveserver/hive-audit.log**. The following is an example.

```
2024-02-27 11:55:56,501 | INFO  | HiveServer2-Handler-Pool: Thread-402 | UserName=test  UserID=192.168.64.230  Time=2024/02/27 11:55:56      Operation=OpenSession Result= Data
11:55:56,501 | INFO  | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:56,501 | INFO  | HiveServer2-Handler-Pool: Thread-402 | UserName=test  UserID=192.168.64.230  Time=2024/02/27 11:55:56      Operation=OpenSession Result=SUCCESS
Detail= Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:59,212 | INFO  | HiveServer2-Handler-Pool: Thread-402 | UserName=test  UserID=192.168.64.230  Time=2024/02/27 11:55:59      Operation=ExecuteStatement  Stmt
=(show tables)  Result= Detail= Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:59,336 | INFO  | HiveServer2-Handler-Pool: Thread-402 | OperationId=7187e0c-b8d3-42f0-80d3-760babf5282ef  UserName=test  UserID=192.168.64.230  Time=2024/02/27 11:55:59      Operation=ExecuteStatement  Stmt=(show tables)  Result=SUCCESS  Detail= Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
```

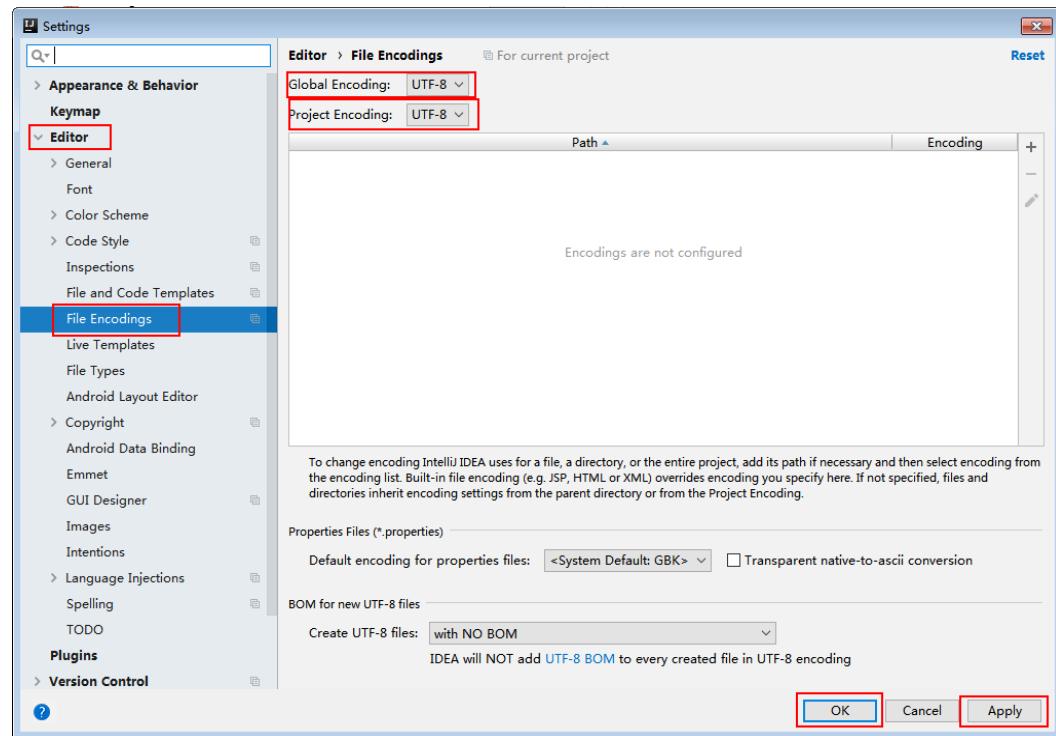
**Step 3** Import the example project to the IntelliJ IDEA development environment.

1. On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
2. In the displayed window, select the folder **hive-jdbc-example**, and click **OK**. On Windows, the path cannot contain any space.

**Step 4** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

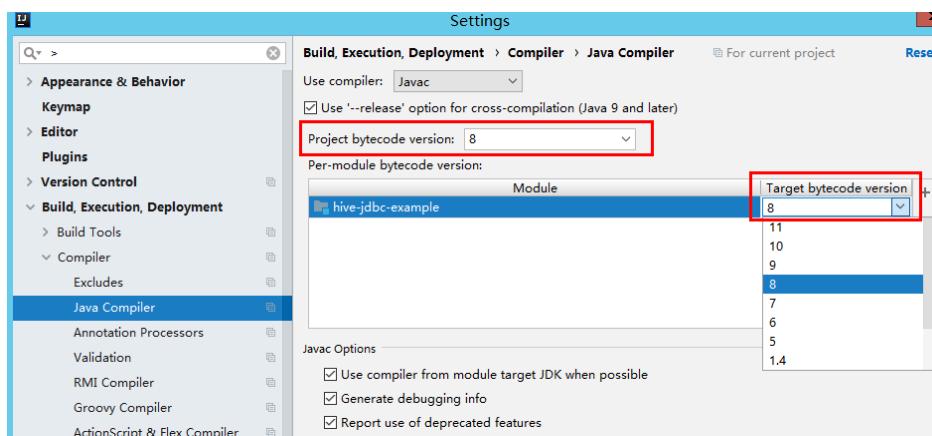
1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.
2. In the navigating tree on the left, choose **Editor > File Encodings**. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, and click **Apply** and **OK**, as shown in [Figure 2-163](#).

Figure 2-163 Setting the IntelliJ IDEA coding format

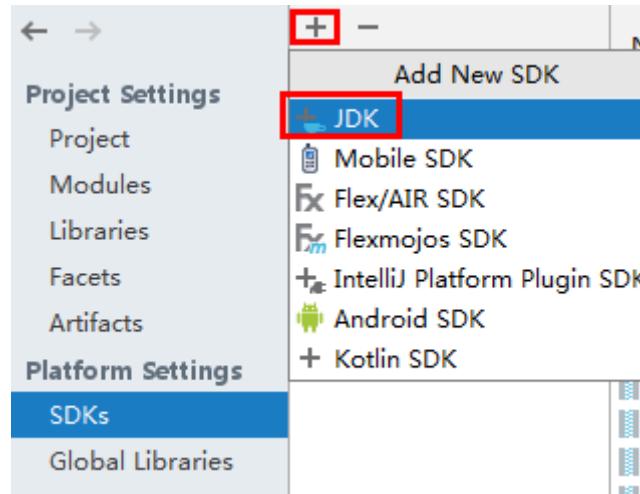


**Step 5** Set the JDK of the project.

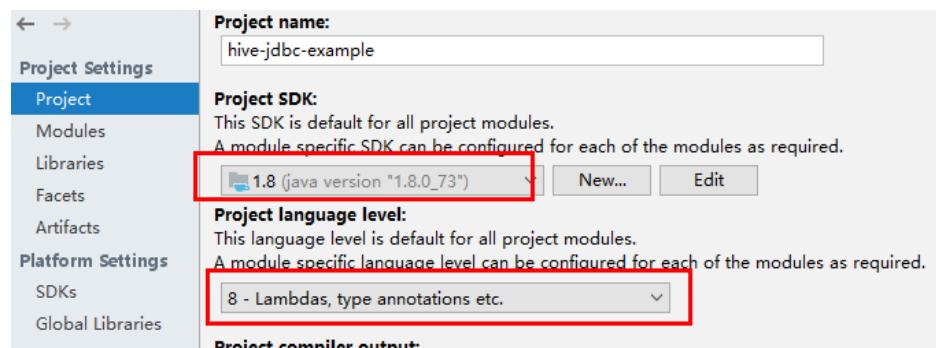
1. On the IntelliJ IDEA menu bar, choose **File > Settings....** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



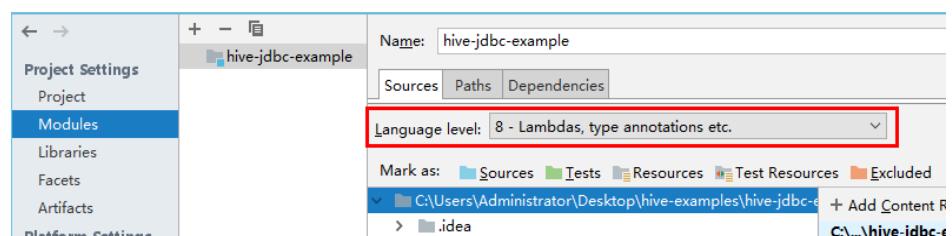
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



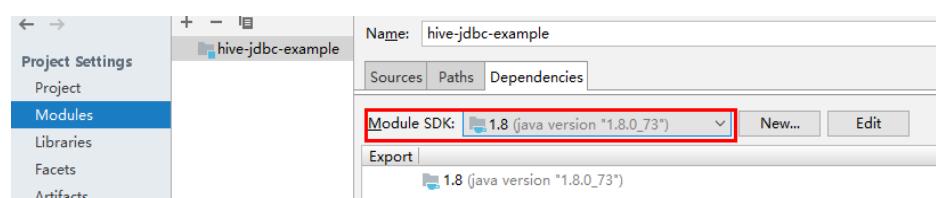
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

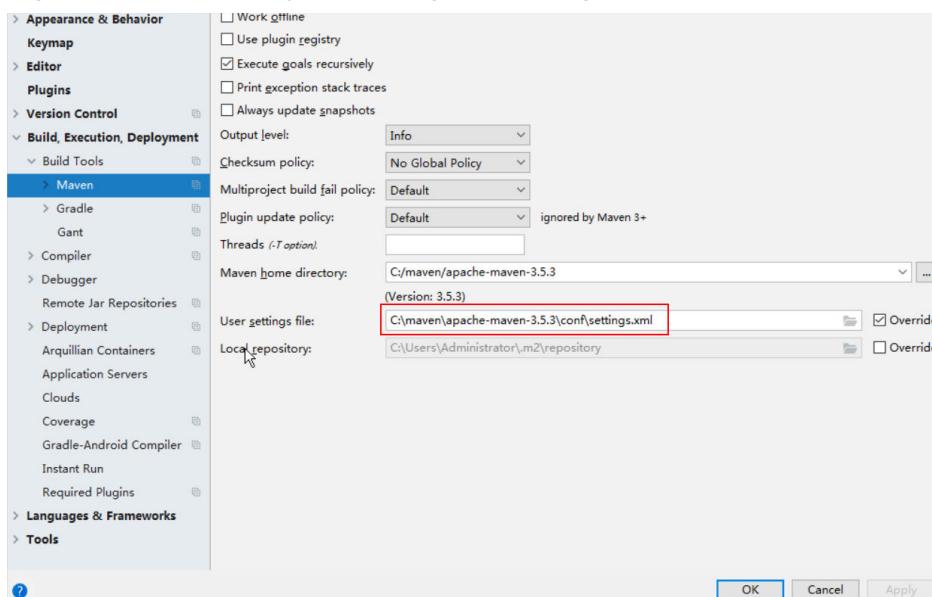


11. Click **Apply** and **OK**.

**Step 6** Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is *<Local Maven installation directory>\conf\settings.xml*.

**Figure 2-164** Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

### 2.9.2.3 Configuring the Hcatalog Sample Project

#### Scenario

To run the HCatalog interface example codes of the Hive component of MRS, perform the following operations.

#### NOTE

The following uses the development of an application for connecting the Hive service in HCatalog mode on Windows as an example.

## Procedure

**Step 1** Obtain the sample project folder **hcatalog-example** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

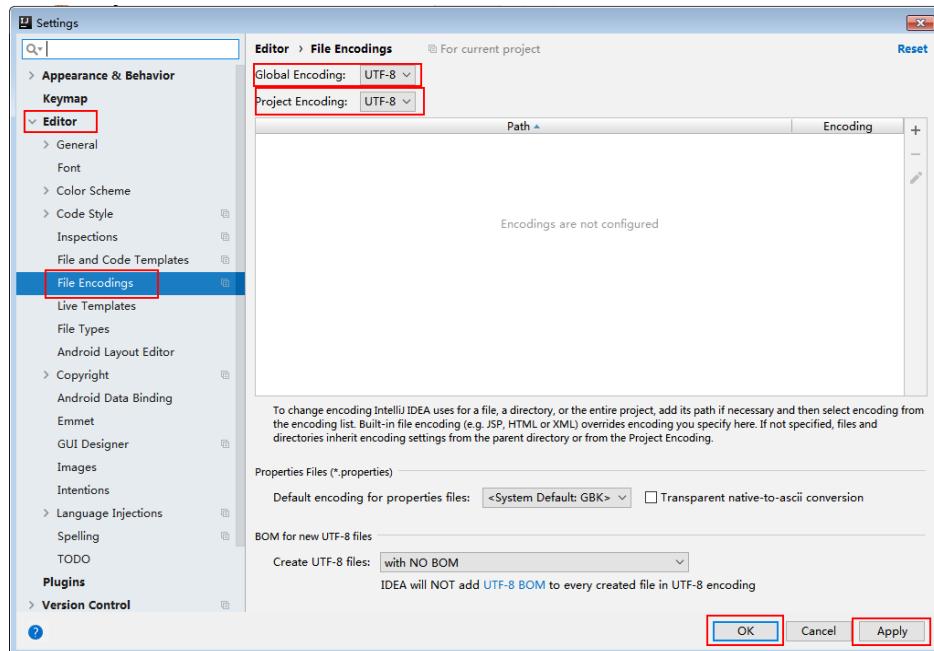
**Step 2** Import the example project to the IntelliJ IDEA development environment.

1. On the IntelliJ IDEA menu bar, choose **File > Open....** The **Open File or Project** window is displayed.
2. In the displayed window, select the folder **hcatalog-example**, and click **OK**. On Windows, the path cannot contain any space.

**Step 3** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

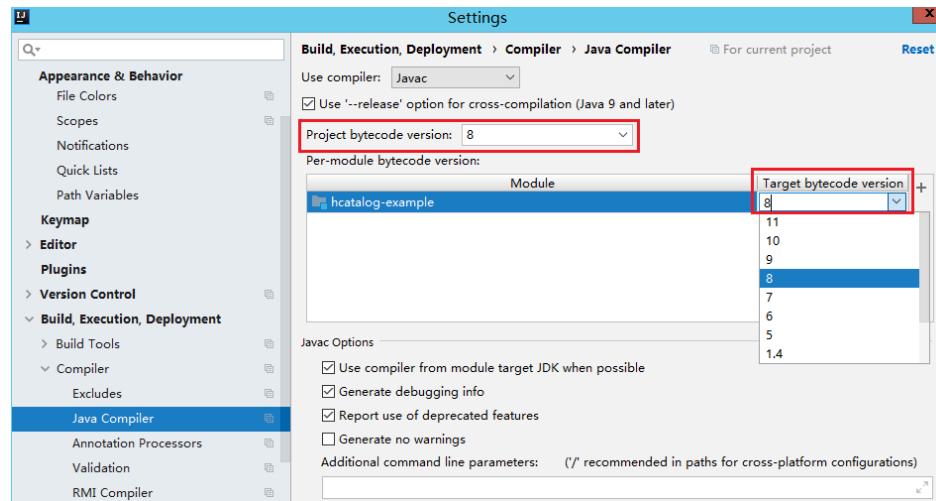
1. On the IntelliJ IDEA menu bar, choose **File > Settings**. The **Settings** window is displayed.
2. Choose **Editor > File Encodings** from the navigation tree. In the **Project Encoding** area and **Global Encoding** area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 2-165](#).

**Figure 2-165** Setting the IntelliJ IDEA coding format

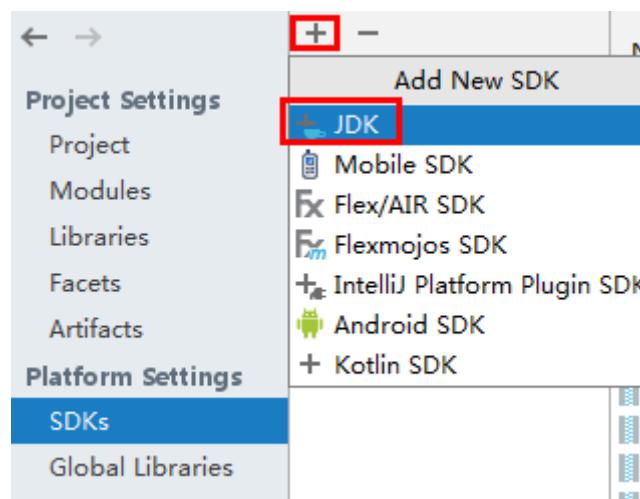


**Step 4** Set the JDK of the project.

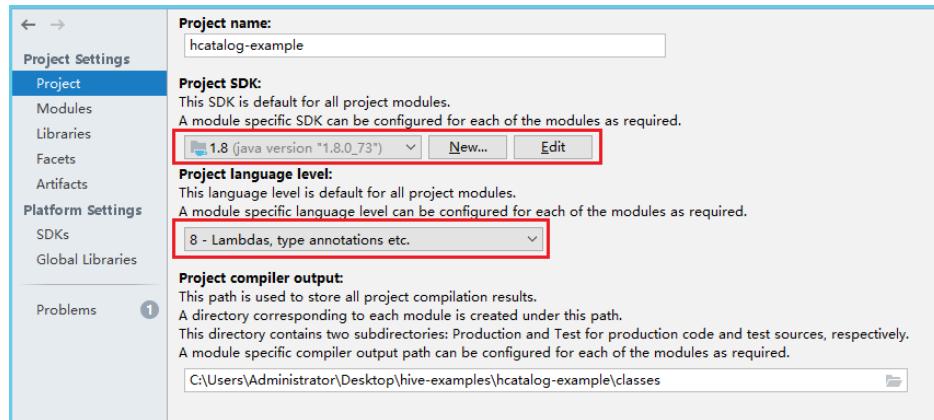
1. On the IntelliJ IDEA menu bar, choose **File > Settings....** The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler**, select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** to **8** for **hive-jdbc-example**.



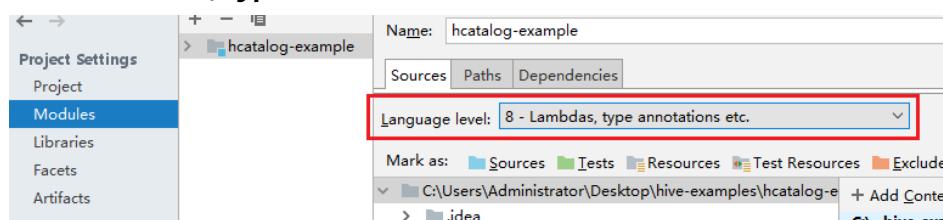
3. Click **Apply** and **OK**.
4. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Select **SDKs**, click the plus sign (+), and select **JDK**.



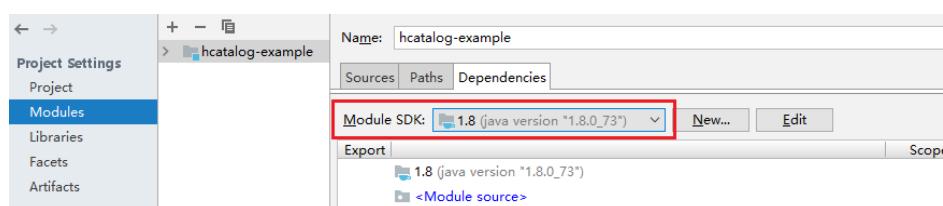
6. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
7. After selecting the JDK, click **Apply**.
8. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



11. On the **Dependencies** page, change the value of **Module SDK** to the JDK added in **SDKs**.

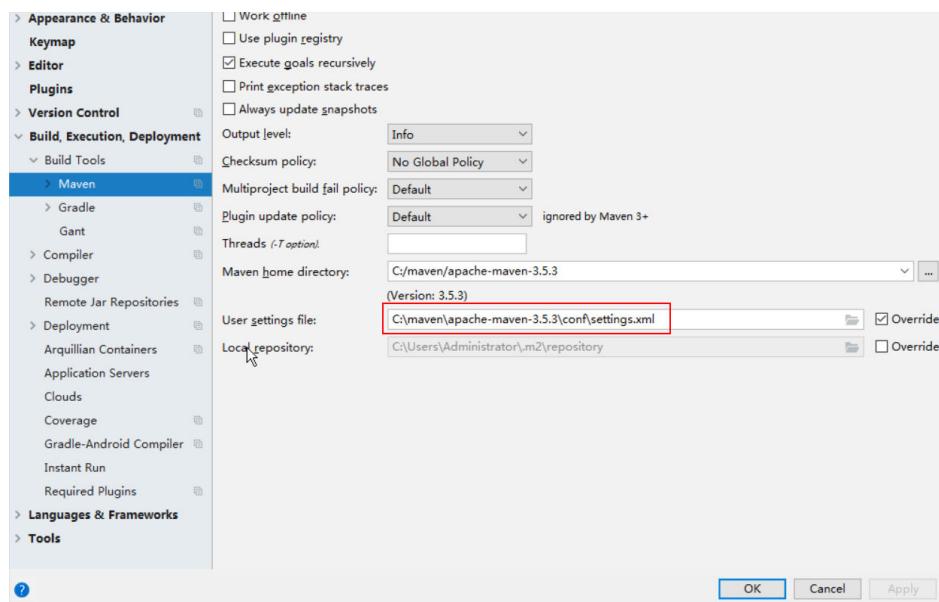


12. Click **Apply** and **OK**.

#### Step 5 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

Figure 2-166 Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply**, and then click **OK**.

----End

#### 2.9.2.4 Configuring the Python Sample Project

##### Scenario

To run the Python interface example codes of the Hive component of MRS, perform the following operations.

##### Procedure

**Step 1** Python of 2.6.6 or a higher version has been installed on a client. The Python version cannot be higher than 2.7.13.

The Python version can be viewed by running the **python** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 2.6.6.

```
Python 2.6.6 (r266:84292, Oct 12 2012, 14:23:48)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

**Step 2** Setuptools of 5.0 or a higher version has been installed on a client. The setuptools version cannot be higher than 36.8.0.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python setup.py install** command in the CLI of the client.

The following information indicates that setuptools 5.7 is installed successfully.

Finished processing dependencies for setuptools==5.7

**Step 3** Install Python on the client.

1. Obtain the sample project folder **python-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python-examples** folder.
3. Run the **python setup.py install** command in the CLI.

The following information indicates that Python is installed successfully.

Finished processing dependencies for pyhs2==0.5.0

**Step 4** After the installation is successful, the following files are generated. **python-examples/pyCLI\_nosec.py** is the Python client example codes. **python-examples/pyhs2/haconnection.py** is the Python client API. Run **hive\_python\_client** scripts to execute the SQL functions, for example, **sh hive\_python\_client 'show tables'**. This function applies to only simple SQL statements and depends on the ZooKeeper client.

----End

### 2.9.2.5 Configuring the Python3 Sample Project

#### Scenario

To run the Python3 interface example codes of the Hive component of FusionInsight MRS, perform the following operations.

#### Procedure

**Step 1** Python3 of 3.6 or a higher version has been installed on a client. The Python3 version cannot be higher than 3.9.

The Python version can be viewed by running the **python3** command on the command-line interface (CLI) of the client. The following information indicates that the Python version is 3.8.2.

Python 3.8.2 (default, Jun 23 2020, 10:26:03)  
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux  
Type "help", "copyright", "credits" or "license" for more information.

**Step 2** Setuptools of 47.3.1 version has been installed on a client.

To obtain the software, visit the official website.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

Finished processing dependencies for setuptools==47.3.1

 NOTE

If the system displays a message indicating that the installation of setuptools of 47.3.1 fails, check whether the environment is faulty or whether the problem is caused by Python.

**Step 3** Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency\_python3.6**, **dependency\_python3.7**, or **dependency\_python3.8**, or **dependency\_python3.9** folder based on the Python3 version.
4. Run the **whereis easy\_install** command to find the path of the **easy\_install** program. If there are multiple paths, run the **easy\_install --version** command to select the **easy\_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy\_install**.
5. Run the **easy\_install** command to install the egg files in the **dependency\_python3.x** folder. If the egg file has dependencies, you can use wildcards to install the egg file. For example:
  - **dependency\_python3.6** directory:  
**/usr/local/bin/easy\_install future\*egg six\*egg python\*egg sasl-\*linux-\$uname -p.egg thrift-\*egg thrift\_sasl\*egg**
  - **dependency\_python3.7** directory:  
**/usr/local/bin/easy\_install future\*egg six\*egg sasl-\*linux-\$uname -p.egg thrift-\*egg thrift\_sasl\*egg**
  - **dependency\_python3.8** directory:  
**/usr/local/bin/easy\_install future\*egg six\*egg python\*egg sasl-\*linux-\$uname -p.egg thrift-\*linux-\$uname -p.egg thrift\_sasl\*egg**
  - **dependency\_python3.9** directory:  
**/usr/local/bin/easy\_install future\*egg six\*egg sasl-\*linux-\$uname -p.egg six-\*egg thrift-\*linux-\$uname -p.egg thrift\_sasl\*egg**

If the following information is displayed for each egg file, the installation is successful:

Finished processing dependencies for \*\*\*

**Step 4** After the installation is successful, the following files are generated. **python3-examples/pyCLI\_sec.py** is the Python client example codes. **python3-examples/pyhive/hive.py** is the Python client API.

----End

### 2.9.2.6 Configuring a Spring Boot Sample Project

#### Scenario

Run the Spring Boot interface sample code of MRS Hive.

In the following example, an application is developed by compiling a Spring Boot project to connect to Hive in Windows.

## Procedure

- Step 1** Obtain the sample project folder **hive-rest-client-example** in the `src\springboot\hive-examples` directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Go to the client decompression path `FusionInsight_Cluster_1_Services_ClientConfig\Hive\config` and copy the `core-site.xml` and `hiveclient.properties` files to the `hive-jdbc-example\src\main\resources` directory of the sample project.



Hive allows you to add extension identifiers to JDBC connection strings. These extension identifiers are printed in HiveServer audit logs to distinguish SQL sources. If you need to configure an identifier, add the following content to the `hive-jdbc-example\src\main\resources\hiveclient.properties` file:

```
auditAddition=xxx
```

`xxx` is the custom identifier. The identifier can contain a maximum of 256 bytes and only letters, digits, underscores (\_), commas (,), and colons (:) are allowed.

After the SQL statement is executed, the configured extension identifier is printed in the HiveServer audit log `/var/log/Bigdata/audit/hive/hiveserver/hive-audit.log`. The following is an example.

```
2024-02-27 11:55:56,501 | INFO | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserIP=192.168.64.230 Time=2024/02/27 11:55:56 Operation=OpenSession Result= Data
11=Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:56,501 | INFO | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserIP=192.168.64.230 Time=2024/02/27 11:55:56 Operation=OpenSession Result=SUCCE
SS
2024-02-27 11:55:59,336 | INFO | HiveServer2-Handler-Pool: Thread-402 | UserName=test UserIP=192.168.64.230 Time=2024/02/27 11:55:59 Operation=ExecuteStatement stat
=(show tables) Result= Details=Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
2024-02-27 11:55:59,336 | INFO | HiveServer2-Handler-Pool: Thread-402 | OperationId=7187e0c-b8d1-42f0-80d3-76babf5282ef UserName=test UserIP=192.168.64.230 Time=2024/02
/27 11:55:59 Operation=ExecuteStatement: stat=(show tables) Result=SUCCESS Detail= Addition=xxx | org.apache.hive.service.cli.thrift.ThriftCLIService.logAuditEvent(ThriftCLIService.java:512)
```

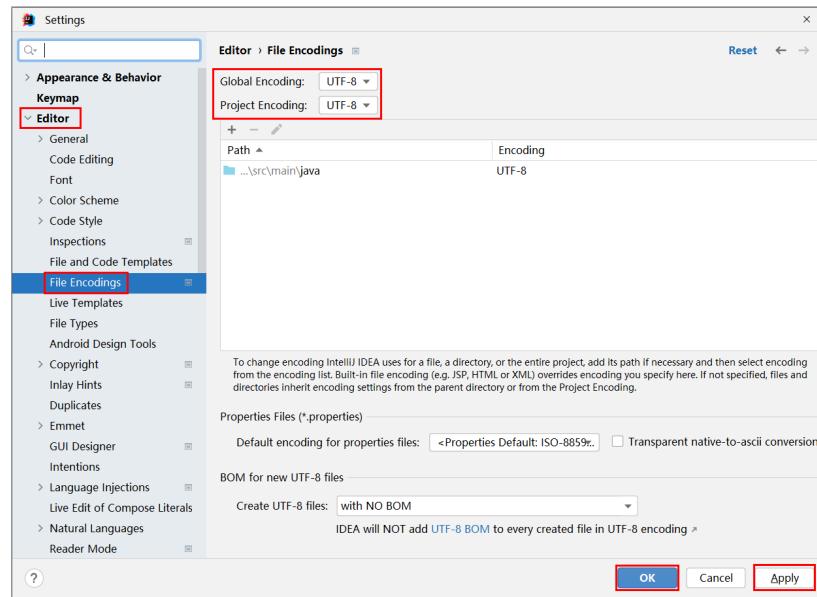
- Step 3** Import the sample project to the IntelliJ IDEA development environment.

- On the menu bar of IntelliJ IDEA, choose **File > Open....**
- In the **Open File or Project** dialog box that is displayed, select the **hive-rest-client-example** folder and click **OK**. In Windows, the folder path cannot contain any space.

- Step 4** Set the IntelliJ IDEA text file coding format to prevent garbled characters.

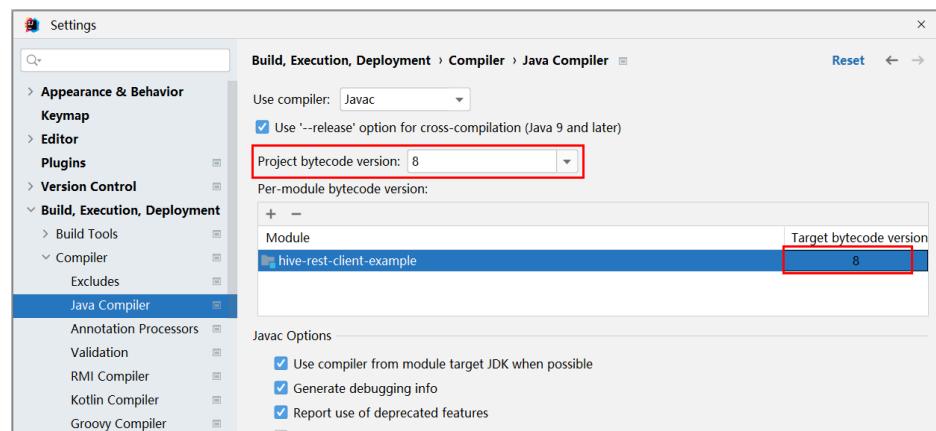
- On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
- In the left navigation pane, choose **Editor > File Encodings**. In the **Project Encoding** and **Global Encoding** areas, set the parameter to **UTF-8**. Click **Apply** and **OK**.

Figure 2-167 Setting the IntelliJ IDEA coding format

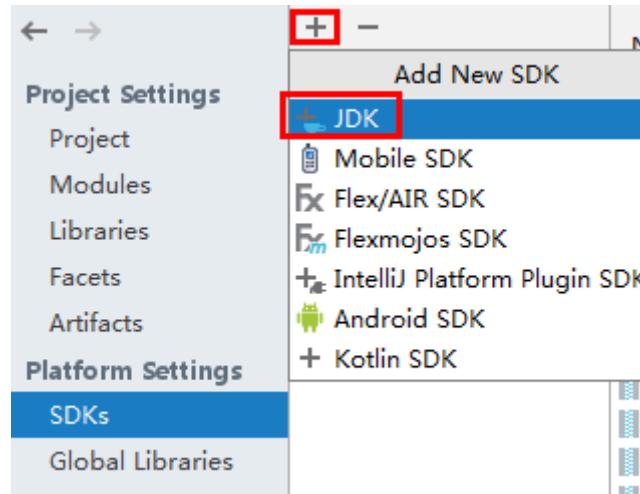


**Step 5** Set the JDK of the project.

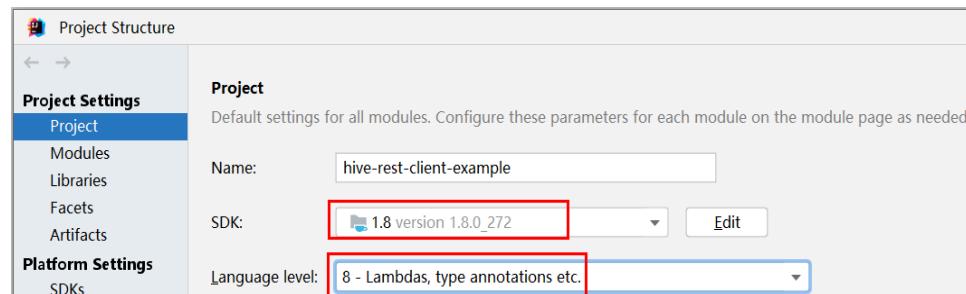
1. On the menu bar of IntelliJ IDEA, choose **File > Settings**. The **Settings** window is displayed.
2. Choose **Build, Execution, Deployment > Compiler > Java Compiler** and select **8** from the **Project bytecode version** drop-down list box. Change the value of **Target bytecode version** in **hive-rest-client-example** to **8**.



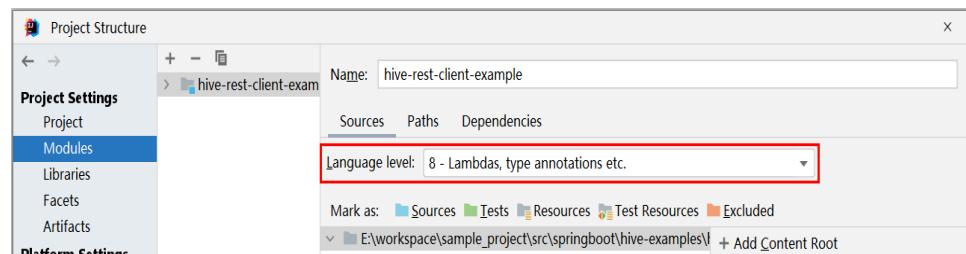
3. Click **Apply** then **OK**.
4. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
5. Choose **SDKs**, click the plus sign (+), and select **JDK**.



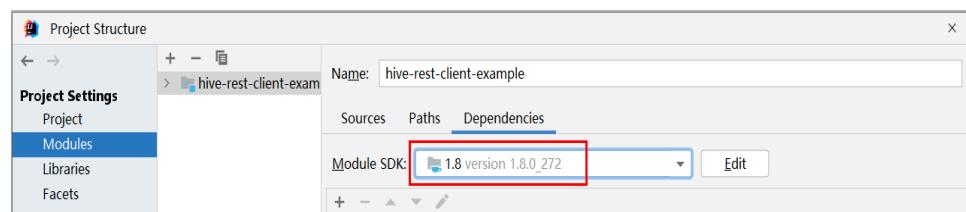
6. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
7. Click **Apply**.
8. Select **Project**, select the JDK added in **SDKs** from the **SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Language level** drop-down list box.



9. Click **Apply**.
10. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc..**



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.

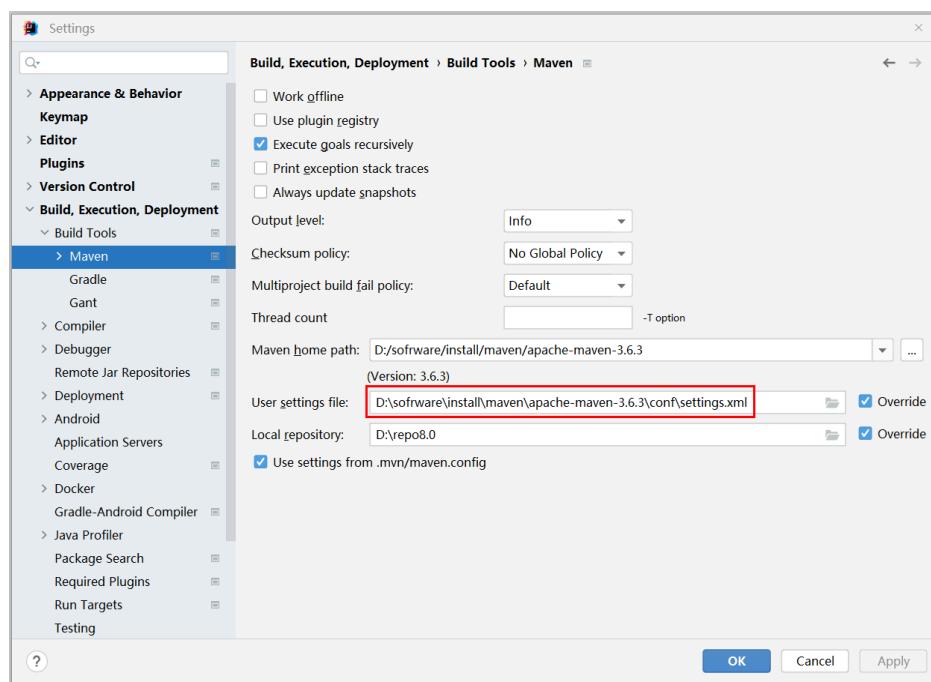


11. Click **Apply** then **OK**.

### Step 6 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is *<Local Maven installation directory>\conf\settings.xml*.

Figure 2-168 Directory for storing the settings.xml file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply** then **OK**.

----End

## 2.9.3 Developing an Application

### 2.9.3.1 Typical Scenario Description

#### Scenarios

A user develops a Hive data analysis application for managing employee information described in [Table 2-108](#) and [Table 2-109](#).

#### Procedure

##### Step 1 Prepare data.

1. Create three tables: employee information table **employees\_info**, contact table **employees\_contact**, and extended employee information table **employees\_info\_extended**.
  - Fields in the **employees\_info** table include the employee ID, name, salary currency, salary, tax category, work place, and hiring date. **R** indicates RMB, and **D** indicates USD.
  - Fields in the **employees\_contact** table include the employee ID, mobile phone number, and e-mail address.
  - Fields in the **employees\_info\_extended** table include the employee ID, name, mobile phone number, e-mail address, salary currency, salary, tax category, and work place. The partition field is the hiring date.

For table creation codes, see [Creating a Table](#).

2. Load employee information to **employees\_info**.

For data loading codes, see [Loading Data](#).

**Table 2-108** describes employee information.

**Table 2-108** Employee information

ID	Name	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wang	R	8000.01	personal income tax&0.05	Country 1:City1	2014
3	Tom	D	12000.02	personal income tax&0.09	Country 2:City2	2014
4	Jack	D	24000.03	personal income tax&0.09	Country 3:City3	2014
6	Linda	D	36000.04	personal income tax&0.09	Country 4:City4	2014
8	Zhang	R	9000.05	personal income tax&0.05	Country 5:City5	2014

3. Load employee contact information to **employees\_contact**.

**Table 2-109** describes employee contact information.

**Table 2-109** Employee contact information

ID	Mobile Phone Number	E-mail Address
1	135 XXXX XXXX	xxxx@xx.com

ID	Mobile Phone Number	E-mail Address
3	159 XXXX XXXX	xxxxx@xx.com.cn
4	186 XXXX XXXX	xxxx@xx.org
6	189 XXXX XXXX	xxxx@xxx.cn
8	134 XXXX XXXX	xxxx@xxxx.cn

4. Load extended employee information to **employees\_info\_extended**.

**Table 2-110** describes the extended employee information.

**Table 2-110** Extended employee information

ID	Name	Mobile Phone Number	E-mail Address	Salary Currency	Salary	Tax Category	Work Place	Hiring Date
1	Wa ng	135 XXXX XXXX	xxxx@x x.com	R	8000 .01	personal income tax&0.0 5	Country1 :City1	201 4
3	To m	159 XXXX XXXX	xxxxx@ xx.com. cn	D	1200 0.02	personal income tax&0.0 9	Country2 :City2	201 4
4	Jack	186 XXXX XXXX	xxxx@x x.org	D	2400 0.03	personal income tax&0.0 9	Country3 :City3	201 4
6	Lin da	189 XXXX XXXX	xxxx@x x.cn	D	3600 0.04	personal income tax&0.0 9	Country4 :City4	201 4
8	Zha ng	134 XXXX XXXX	xxxx@x xx.cn	R	9000 .05	personal income tax&0.0 5	Country5 :City5	201 4

## Step 2 Analyze data.

For data analysis codes, see [Querying Data](#).

- Query contact information of employees whose salaries are paid in USD.
- Query the IDs and names of employees who were hired in 2014, and load query results to the partition with the hiring time of 2014 in **employees\_info\_extended**.

- Collect statistics for the number of records in the employees\_info table.
- Query information about employees whose email addresses end with "cn".

**Step 3** Submit a data analysis task to collect statistics for the number of records in the employees\_info table.

For details about the implementation, see [Example Program Guide](#).

----End

### 2.9.3.2 Example Codes

#### 2.9.3.2.1 Creating a Table

##### Function

This topic describes how to use Hive Query Language (HQL) to create internal and external tables. You can create a table in three modes:

- Define the table structure, and use the key word EXTERNAL to differentiate between internal and external tables.
  - If all data is to be processed by Hive, create an internal table. When an internal table is deleted, the metadata and data in the table are deleted.
  - If data is to be processed by multiple tools, create an external table. When an external table is deleted, only the metadata is deleted.
- Create a table based on existing tables. Use CREATE LIKE to fully copy the original table structure, including the storage format.
- Create a table based on query results using CREATE AS SELECT.  
In this mode, you can specify which fields are to be copied when copying the original table structure. The storage format is not copied.



Both the table name and field name can contain a maximum of 128 bytes. Both the field comment and value can contain a maximum of 4000 bytes. The key in **WITH SERDEPROPERTIES** can contain a maximum of 256 bytes.

### Example Codes

```
-- Create an external table employees_info.  
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info  
(  
id INT,  
name STRING,  
usd_flag STRING,  
salary DOUBLE,  
deductions MAP<STRING, DOUBLE>,  
address STRING,  
entrytime STRING  
)  
-- Specify the field delimiter. Use delimited fields terminated by to specify the delimiter between columns to a comma (,).  
-- Use MAP KEYS TERMINATED BY to specify the delimiter between map keys to &  
ROW FORMAT delimited fields terminated by '' MAP KEYS TERMINATED BY '&'  
-- Set the storage format to TEXTFILE.  
STORED AS TEXTFILE;  
  
-- Create an external table employees_contact.
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS employees_contact
(
    id INT,
    tel_phone STRING,
    email STRING
)
ROW FORMAT delimited fields terminated by ','
STORED AS TEXTFILE;

-- Create an external table employees_info_extended.
CREATE EXTERNAL TABLE IF NOT EXISTS employees_info_extended(id INT, name STRING, usd_flag STRING,
salary DOUBLE, deductions MAP<STRING, DOUBLE>, address STRING)
-- A table may have one or multiple partitions. Each partition is saved as an independent folder in the table
directory. Partitions help minimize the query scope, accelerate data query, and allow users to manage data
based on certain criteria.
-- Use PARTITIONED BY to specify the column name and data type of the partition.
PARTITIONED BY(entrytime STRING)
ROW FORMAT delimited fields terminated by ',' MAP KEYS TERMINATED BY '&
STORED AS TEXTFILE;
-- After a table is created, you can use ALTER TABLE to add or delete fields to or from the table, modify
table attributes, and add partitions.
-- Add the tel_phone and email fields to the employees_info_extended table.
ALTER TABLE employees_info_extended ADD COLUMNS (tel_phone STRING, email STRING);
```

### 2.9.3.2.2 Loading Data

#### Function

This topic describes how to use Hive Query Language (HQL) to load data to the existing **employees\_info** table. You can learn how to load data from a local file system and MRS cluster. LOCAL is used to differentiate between local and non-local data sources.

#### Example Codes

```
-- Load the employee_info.txt file from the /opt/hive_examples_data/ directory of the local file system to
the employees_info table.
---- Overwrite the original data using the new data
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
---- Retain the original data and add the new data to the table.
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt' INTO TABLE employees_info;

-- Load /user/hive_examples_data/employee_info.txt from HDFS to the employees_info table.
---- Overwrite the original data using the new data
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' OVERWRITE INTO TABLE
employees_info;
---- Retain the original data and add the new data to the table.
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt' INTO TABLE employees_info;
```



Loading data is to copy data to a specified table in the Hadoop distributed file system (HDFS).

#### Sample Data

Data in **employees\_info** is as follows:

```
1,Wang,R,8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014
3,Tom,D,12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014
4,Jack,D,24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014
6,Linda,D,36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014
8,Zhang,R,9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

Data in **employees\_contact** is as follows:

```
1,135 XXXX XXXX,xxxx@xx.com  
3,159 XXXX XXXX,xxxx@xx.com.cn  
4,186 XXXX XXXX,xxxx@xx.org  
6,189 XXXX XXXX,xxxx@xxx.cn  
8,134 XXXX XXXX,xxxx@xxxx.cn
```

Data in **employees\_info\_extended** is as follows:

```
1,Wang,135 XXXX  
XXXX,xxxx@xx.com,R,8000.01,person&personal^Btype&income^Btax&0.05,Country1:City1,2014  
3,Tom,159 XXXX  
XXXX,xxxx@xx.com.cn,D,12000.02,person&personal^Btype&income^Btax&0.09,Country2:City2,2014  
4,Jack,186 XXXX  
XXXX,xxxx@xx.org,D,24000.03,person&personal^Btype&income^Btax&0.05,Country3:City3,2014  
6,Linda,189 XXXX  
XXXX,xxxx@xxx.cn,D,36000.04,person&personal^Btype&income^Btax&0.05,Country4:City4,2014  
8,Zhang,134 XXXX  
XXXX,xxxx@xxxx.cn,R,9000.05,person&personal^Btype&income^Btax&0.05,Country5:City5,2014
```

### 2.9.3.2.3 Querying Data

#### Function

This topic describes how to use Hive Query Language (HQL) to query and analyze data. You can query and analyze data using the following methods:

- Use common features for SELECT query, such as JOIN.
- Load data to a specified partition.
- Use Hive-provided functions.
- Query and analyze data using user-defined functions (UDFs). For details about how to create and define UDFs, see [UDF](#).

#### Example Codes

```
-- Query the contact information of employees whose salaries are paid in USD.  
SELECT  
a.name,  
b.tel_phone,  
b.email  
FROM employees_info a JOIN employees_contact b ON(a.id = b.id) WHERE usd_flag='D';  
  
-- Query the IDs and names of employees who were hired in 2014, and load query results to the partition  
with the hiring time of 2014 in the employees_info_extended table.  
INSERT OVERWRITE TABLE employees_info_extended PARTITION (entrytime = '2014')  
SELECT  
a.id,  
a.name,  
a.usd_flag,  
a.salary,  
a.deductions,  
a.address,  
b.tel_phone,  
b.email  
FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE a.entrytime = '2014';  
  
-- Use the existing Hive function COUNT() to count the number of records in the employees_info table.  
SELECT COUNT(*) FROM employees_info;  
  
-- Query information about employees whose email addresses end with "cn".  
SELECT a.name, b.tel_phone FROM employees_info a JOIN employees_contact b ON (a.id = b.id) WHERE  
b.email like '%cn';
```

## Extensions

- Configure intermediate Hive data encryption.  
Set the table format to RCFile (recommended) or SequenceFile, and the encryption algorithm to ARC4Codec. SequenceFile is a unique Hadoop file format, and RCFile is a Hive file format with optimized column storage. When a big table is queried, RCFile provides higher performance than SequenceFile.

```
set hive.exec.compress.output=true;
set hive.exec.compress.intermediate=true;
set hive.intermediate.compression.codec=org.apache.hadoop.io.encryption.arc4.ARC4Codec;
```
- For details about UDFs, see [UDF](#).

### 2.9.3.2.4 UDF

When internal functions of Hive cannot meet requirements, you can compile user-defined functions (UDFs) and use them for query.

According to implementation methods, UDFs are classified as follows:

- Common UDFs: used to perform operations on a single data row and export a single data row.
- User-defined aggregating functions (UDAFs): used to input multiple data rows and export a single data row.
- User-defined table-generating functions (UDTFs): used to perform operations on a single data row and export multiple data rows.

According to usage methods, UDFs are classified as follows:

- Temporary functions: used only in the current session and must be recreated after a session restarts.
- Permanent function: used in multiple sessions and do not have to be created every time a session restarts.

The following uses AddDoublesUDF as an example to describe how to compile and use UDFs.

## Function

AddDoublesUDF is used to add two or multiple floating point values. The following example describes how to compile and use UDFs.

### NOTE

- The normal UDF must be originated from `org.apache.hadoop.hive.ql.exec.UDF`.
- The normal UDF must implement at least one `evaluate()`. The `evaluate` function supports overloading.
- To develop a customized function, you need to add the `hive-exec-3.1.0.jar` dependency package to the project. The package can be obtained from the Hive installation directory.

## Example Codes

The following is a UDF example:

```
package com.huawei.bigdata.hive.example.udf;
import org.apache.hadoop.hive.ql.exec.UDF;
```

```
public class AddDoublesUDF extends UDF {  
    public Double evaluate(Double... a) {  
        Double total = 0.0;  
        // Processing logic  
        for (int i = 0; i < a.length; i++)  
            if (a[i] != null)  
                total += a[i];  
        return total;  
    }  
}
```

## How to Use

- Step 1** Package the preceding program into **AddDoublesUDF.jar**, and upload it to a directory on the HDFS (such as `/user/hive_examples_jars/`). The user who creates the UDF and the user who uses the UDF function must have read right on this JAR file. Example statements:

```
hdfs dfs -put ./hive_examples_jars /user/hive_examples_jars
```

```
hdfs dfs -chmod 777 /user/hive_examples_jars
```

- Step 2** Run the following command:

```
beeline -n Hive service user
```

- Step 3** Define the function in Hive Server. Run the following SQL statement to create a permanent function:

```
CREATE FUNCTION addDoubles AS  
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://  
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

*addDoubles* indicates the function alias that is used for SELECT query.

Run the following statement to create a temporary function:

```
CREATE TEMPORARY FUNCTION addDoubles AS  
'com.huawei.bigdata.hive.example.udf.AddDoublesUDF' using jar 'hdfs://  
hacluster/user/hive_examples_jars/AddDoublesUDF.jar';
```

- *addDoubles* indicates the function alias that is used for SELECT query.
- TEMPORARY indicates that the function is used only in the current session with the Hive server.

- Step 4** Run the following SQL statement to use the function in the Hive server:

```
SELECT addDoubles(1,2,3);
```



If an **[Error 10011]** error is displayed when you log in to the client again, run the **reload function;** command and then use this function.

- Step 5** Run the following SQL statement to delete the function from the Hive server:

```
DROP FUNCTION addDoubles;
```

----End

## Extensions

None

### 2.9.3.2.5 Example Program Guide

## Function

This section describes how to use an example program to complete an analysis task. An example program can submit a task by using the following methods:

- Submitting a data analysis task by using JDBC interfaces
- Submitting a data analysis task by using Python

## Example Codes

- Submit a data analysis task using the Hive Java database connectivity (JDBC) interface, that is, JDBCExample.java.

- Read the **property** file of the HiveServer client. The **hiveclient.properties** file is saved in the **resources** directory of the JDBC example program provided by Hive.

```
Properties clientInfo = null;
String userdir = System.getProperty("user.dir") + File.separator
+ "conf" + File.separator;
InputStream fileInputStream = null;
try{
    clientInfo = new Properties();
    //hiveclient.properties indicates the client configuration file. If the multiple instances feature is used, the file must be replaced with the hiveclient.properties file on the instance client.
    //hiveclient.properties is located under the config directory of the directory where the instance client installation package is decompressed.
    String hiveclientProp = userdir + "hiveclient.properties" ;
    File propertiesFile = new File(hiveclientProp);
    fileInputStream = new FileInputStream(propertiesFile);
    clientInfo.load(fileInputStream);
} catch (Exception e) {
    throw new IOException(e);
} finally{
    if(fileInputStream != null){
        fileInputStream.close();
        fileInputStream = null;
    }
}
```

- Obtain the IP address and port list of ZooKeeper, the cluster authentication mode, the SASL configuration of HiveServers, node names of HiveServers in ZooKeeper, the discovery mode from the client to the server, and the principal server process for user authentication. You can read all these configurations from the **hiveclient.properties** file.

```
//The format of zkQuorum is
xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx:24002,xxx.xxx.xxx:24002;
//xxx.xxx.xxx.xxx is the IP address of the node where ZooKeeper resides. The default port is
24002.
zkQuorum = clientInfo.getProperty("zk.quorum");
auth = clientInfo.getProperty("auth");
sasl_qop = clientInfo.getProperty("sasl.qop");
zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
principal = clientInfo.getProperty("principal");
auditAddition = clientInfo.getProperty("auditAddition");
```

- In security mode, the kerberos user and keytab file path are required for login authentication. For details about how to obtain **USER\_NAME**,

**USER\_KEYTAB\_FILE**, and **KRB5\_FILE**, see [Running JDBC and Viewing Results](#).

```
// Set the userName of new user.  
USER_NAME = "xxx";  
// Set the keytab and krb5 files location of client.  
String userdir = System.getProperty("user.dir") + File.separator  
+ "conf" + File.separator;  
USER_KEYTAB_FILE = userdir + "user.keytab";  
KRB5_FILE = userdir + "krb5.conf";
```

- d. Define HQL. HQL must be a single statement and cannot contain semicolons (;).

```
// Define HQL. HQL cannot contain ";"  
String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",  
"SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};
```

- e. Build JDBC URL.

 **NOTE**

You can also implement pre-authentication without the need of providing the account and keytab file path. For details, see JDBC code example 2 in [Examples](#). If IBM JDK is used to run Hive applications, pre-authentication in JDBC sample code 2 must be implemented.

The following is an example of the JDBC URL composed of code snippets:

```
jdbc:hive2://  
xxx.xxx.xxx:24002,xxx.xxx.xxx:24002,xxx.xxx.xxx:24002;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2;sasl.qop=auth-conf;auth=KERBEROS;principal=hive/hadoop.<system domain name>@<system domain name>;
```

You can login in to FusionInsight Manager, choose **System > Permission > Domain and Mutual Trust**, and check the value of **Local Domain**, which is the current system domain name.

**hive/hadoop.<system domain name>** is the user name. All letters in the system domain name contained in the user name of the system are lowercase letters. For example, if **Local domain** is set to **9427068F-6EFA-4833-B43E-60CB641E5B6C.COM**, the user is **hive/hadoop.9427068f-6efa-4833-b43e-60cb641e5b6c.com**.

```
// Concat JDBC URL  
StringBuilder sBuilder = new StringBuilder(  
"jdbc:hive2://"").append(zkQuorum).append("/");  
  
if ("KERBEROS".equalsIgnoreCase(auth)) {  
    sBuilder.append(";serviceDiscoveryMode=")  
        .append(serviceDiscoveryMode)  
        .append(";zooKeeperNamespace=")  
        .append(zooKeeperNamespace)  
        .append(";sasl.qop=")  
        .append(sasl_qop)  
        .append(";auth=")  
        .append(auth)  
        .append(";principal=")  
        .append(principal)  
        .append(";user.principal=")  
        .append(USER_NAME)  
        .append(";user.keytab=")  
        .append(USER_KEYTAB_FILE);  
  
} else {  
    // Normal mode  
    sBuilder.append(";serviceDiscoveryMode=")  
        .append(serviceDiscoveryMode)  
        .append(";zooKeeperNamespace=")  
        .append(zooKeeperNamespace)  
        .append(";auth=none;");
```

```
        }
        if (auditAddition != null && !auditAddition.isEmpty()) {
            strBuilder.append(";auditAddition=").append(auditAddition);
        }
        String url = sBuilder.toString();
```

- f. Load the Hive JDBC driver.

```
// Load the Hive JDBC driver.  
Class.forName(HIVE_DRIVER);
```

- g. Obtain the JDBC connection, confirm the HQL type (DDL/DML), call ports to run the HQL statement, return the queried column name and results to the console, and close the JDBC connection.

```
Connection connection = null;  
try {  
    // Obtain the JDBC connection.  
    // If the normal mode is used, the second parameter needs to be set to a correct username.  
    // Otherwise, the anonymous user will be used for login.  
    connection = DriverManager.getConnection(url, "", "");  
  
    // Create a table  
    // To import data to a table after the table is created, you can use the LOAD statement. For  
    // example, import data from the HDFS to the table.  
    //load data inpath '/tmp/employees.txt' overwrite into table employees_info;  
    execDDL(connection,sqls[0]);  
    System.out.println("Create table success!");  
  
    // Query  
    execDML(connection,sqls[1]);  
  
    // Delete the table  
    execDDL(connection,sqls[2]);  
    System.out.println("Delete table success!");  
}  
finally {  
    // Close the JDBC connection.  
    if (null != connection) {  
        connection.close();  
    }  
  
public static void execDDL(Connection connection, String sql)  
throws SQLException {  
    PreparedStatement statement = null;  
    try {  
        statement = connection.prepareStatement(sql);  
        statement.execute();  
    }  
    finally {  
        if (null != statement) {  
            statement.close();  
        }  
    }  
}  
  
public static void execDML(Connection connection, String sql) throws SQLException {  
    PreparedStatement statement = null;  
    ResultSet resultSet = null;  
    ResultSetMetaData resultMetaData = null;  
  
    try {  
        // Run the HQL statement.  
        statement = connection.prepareStatement(sql);  
        resultSet = statement.executeQuery();  
  
        // Return the queried column name to the console.  
        resultMetaData = resultSet.getMetaData();  
        int columnCount = resultMetaData.getColumnCount();  
    }
```

```
for (int i = 1; i <= columnCount; i++) {  
    System.out.print(resultMetaData.getColumnLabel(i) + '\t');  
}  
System.out.println();  
  
// Return the results to the console.  
while (resultSet.next()) {  
    for (int i = 1; i <= columnCount; i++) {  
        System.out.print(resultSet.getString(i) + '\t');  
    }  
    System.out.println();  
}  
finally {  
    if (null != resultSet) {  
        resultSet.close();  
    }  
  
    if (null != statement) {  
        statement.close();  
    }  
}
```

- Submit a data analysis task using the Python interface, that is, **python-examples/pyCLI\_nosec.py**.

- Import the HAConnection class.

```
from pyhs2.haconnection import HAConnection
```

- Declare the HiveServer IP address list. In this example, **hosts** indicate the nodes of HiveServer, and **xxx.xxx.xxx.xxx** indicates the business IP address.

```
hosts = ["xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"]
```

#### NOTE

- If the HiveServer instance is migrated, the original sample program is invalid. You need to update the IP address of the HiveServer after the migration of the HiveServer instance used in the sample program.
- If multiple Hive instances are used, update the IP address based on the address of the instance that is actually connected.
- Set the third parameter of the **HAConnection** constructor to a correct username. The password can be left empty. Create a connection, run the HQL statement, and return the queried column name and results to the console.

```
try:  
    with HAConnection(hosts = hosts,  
                      port = 21066,  
                      authMechanism = "PLAIN",  
                      user='user1',  
                      password='*****') as haConn:  
        with haConn.getConnection() as conn:  
            with conn.cursor() as cur:  
                # Show databases  
                print cur.getDatabases()  
                # Execute query  
                cur.execute("show tables")  
                # Return column info from query  
                print cur.getSchema()  
                # Fetch table results  
                for i in cur.fetch():  
                    print i
```

```
except Exception, e:  
    print e
```

#### NOTE

If multiple Hive instances are used, you need to modify hosts according to the description in **b** and change the port number based on to the actual port number. The default ports of Hive to Hive4 are 21066 to 21070, respectively.

### 2.9.3.2.6 Accessing Multiple ZooKeepers

#### Description

This section describes how to access both FusionInsight ZooKeeper and the third-party ZooKeeper in the same client process using the `testConnectHive` and `testConnectApacheZK` methods respectively.

In the `JDBCExample` class of the `hive-jdbc-example-multizk` package, the code structure of the main method is as follows:

```
public static void main(String[] args) throws InstantiationException, IllegalAccessException,  
ClassNotFoundException, SQLException, IOException{  
    testConnectHive(); // Method of accessing FusionInsight ZooKeeper  
    testConnectApacheZk(); // Method of accessing the open-source ZooKeeper  
}
```

#### Accessing FusionInsight ZooKeeper

If only the method of accessing FusionInsight ZooKeeper needs to be executed, comment out the `testConnectApacheZk` method in the `main` function.

Before using the `testConnectHive` method to access FusionInsight ZooKeeper, perform the following operations:

- Step 1** Change the value of `USER_NAME` in the `init` method in `JDBCExample`.  
`USER_NAME` indicates the user that is used to access FusionInsight ZooKeeper and has permissions of the FusionInsight Hive and Hadoop common user groups.
  - Step 2** Go to the client decompression path  
`FusionInsight_Cluster_1_Services_ClientConfig_ConfigFiles\Hive\config` and manually import the `core-site.xml` and `hiveclient.properties` files to the `hive-jdbc-example-multizk\src\main\resources` directory of the sample project.
  - Step 3** Check and change the values of `zk.port` and `zk.quorum` in the `hiveclient.properties` file in the `resources` directory.
    - `zk.port`: indicates the port for accessing FusionInsight ZooKeeper. Generally, the default value is used. Change the value as required.
    - `zk.quorum`: indicates the IP address for accessing ZooKeeper quorumpeer. Set it to the IP address of the cluster deployed with the FusionInsight ZooKeeper service.
- End

#### Accessing Open-Source ZooKeeper

To use `testConnectApacheZk` to connect to the open-source ZooKeeper code, change `xxx.xxx.xxx.xxx` in the following code to the IP address of the open-source

ZooKeeper to be connected. Change the port number as required. After the connection is used, run the try statement in try-with-resources mode to automatically close the ZooKeeper connection. If only the sample for accessing the third-party ZooKeeper needs to be executed, comment out the **testConnectHive** method in the **main** function.

```
private static void testConnectApacheZk() {  
    //After the try statement is executed, the ZooKeeper connection is automatically disabled to prevent  
    //connection leakage.  
    try (org.apache.zookeeper.ZooKeeper digestZk =  
        new org.apache.zookeeper.ZooKeeper("xxx.xxx.xxx:port", 600000, null)) {  
        while (true) {  
            if (digestZk.getState().isConnected()) {  
                List<String> nodes = digestZk.getChildren("/", null);  
                logger.info("digest root path:{}", nodes);  
                for (String node : nodes) {  
                    List<String> subNodes = digestZk.getChildren="/" + node, null);  
                    logger.info("{} child path:{}", node, subNodes);  
                    for (String subNode : subNodes) {  
                        logger.info(  
                            "child patch:{}", subNode, digestZk.getChildren="/" + node + "/" + subNode, null));  
                    }  
                }  
                break;  
            }  
            Thread.sleep(1000);  
        }  
    } catch (IOException | KeeperException | InterruptedException e) {  
        logger.error("failed to connect zookeeper", e);  
    }  
}
```

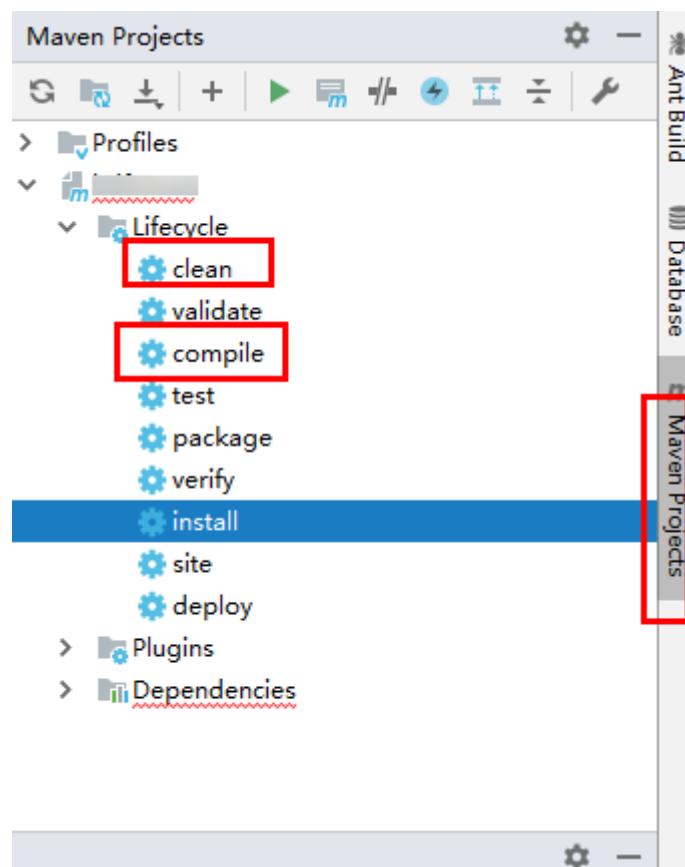
## 2.9.4 Commissioning Applications

### 2.9.4.1 Running JDBC and Viewing Results

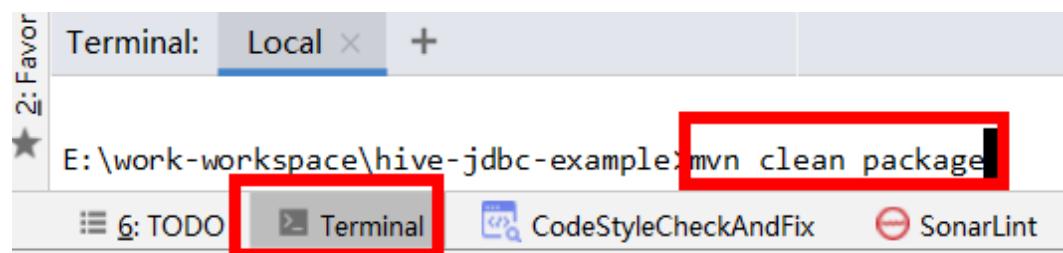
#### Running JDBC in CLI Mode

**Step 1** On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 2-169 Maven Projects page



**Step 2** In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean package** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. A JAR file containing the **-with-dependencies** field is generated in the **target** directory of the sample project.

```
Terminal: Local +  
[INFO] com/ already added, skipping  
[INFO] com/huawei/ already added, skipping  
[INFO] com/huawei/bigdata/ already added, skipping  
[INFO] META-INF/maven/ already added, skipping  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 32.933 s  
[INFO] Finished at: 2020-11-23T16:18:08+08:00  
[INFO] -----
```

**Step 3** Create a directory on Windows or Linux as the running directory, for example, **D:\jdbc\_example** (Windows) or **/opt/jdbc\_example** (Linux). Place the JAR file whose name contains **-with-dependencies** in the **target** directory generated in **Step 2** to this directory. Create the **src/main/resources** subdirectory in the directory. Copy all files in the **resources** directory of the **jdbc-examples** project to the **resources** directory.

**Step 4** In Windows, run the following command:

```
cd /d d:\jdbc_example  
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```

In Linux, run the following command:

```
chmod +x /opt/jdbc_example -R  
cd /opt/jdbc_example  
java -jar hive-jdbc-example-1.0-SNAPSHOT-jar-with-dependencies.jar
```

#### NOTE

The preceding JAR file names are for reference only. The actual names may vary.

**Step 5** In the CLI, view the HQL query results in the example codes.

The following information is displayed if the sample project is successful in Windows:

```
Create table success!  
_c0  
0  
Delete table success!
```

The following information is displayed if the sample project is successful in Linux:

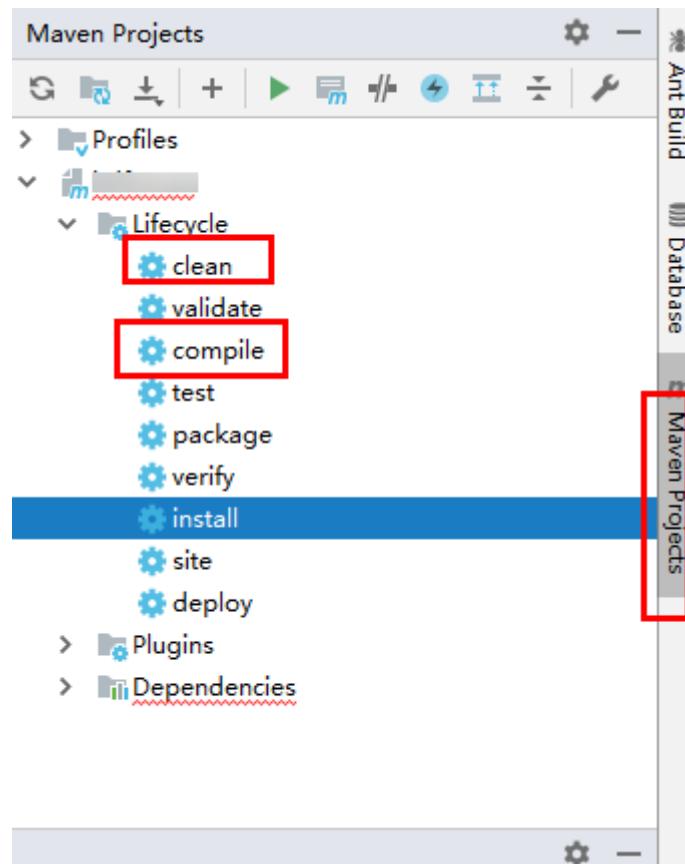
```
Create table success!  
_c0  
0  
Delete table success!
```

----End

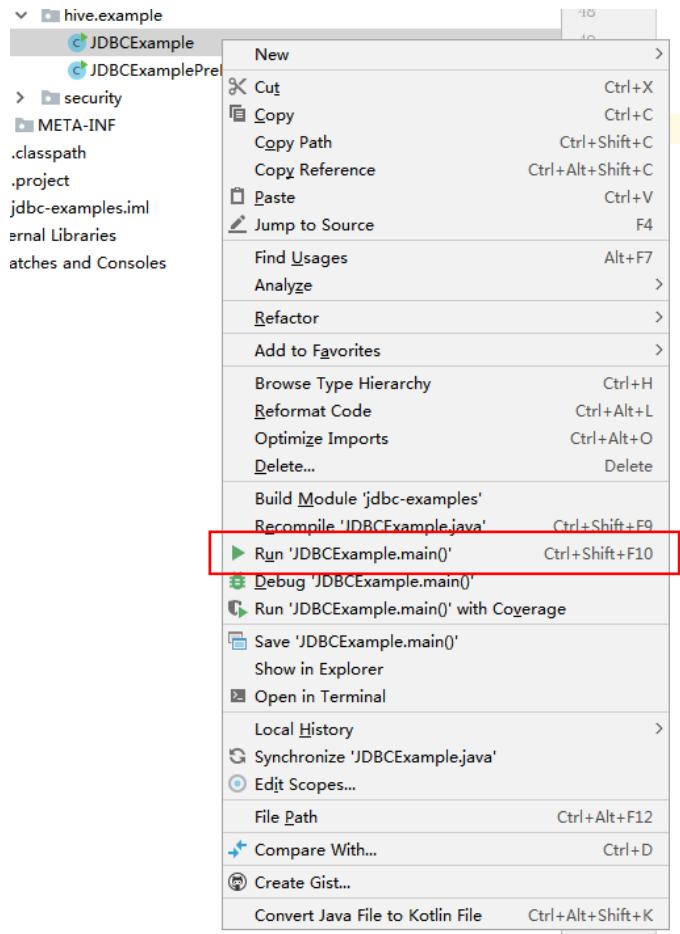
## Running JDBC in IntelliJ IDEA Mode

**Step 1** On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 2-170 Maven Projects page



**Step 2** Right-click the JDBCExample class in the IntelliJ IDEA jdbc-examples project, and choose **Run JDBCExample.main()** from the shortcut menu. As shown in the following figure.



**Step 3** In the IntelliJ IDEA output window, view the HQL query results in the example codes.

```
Create table success!
_c0
0
Delete table success!
```

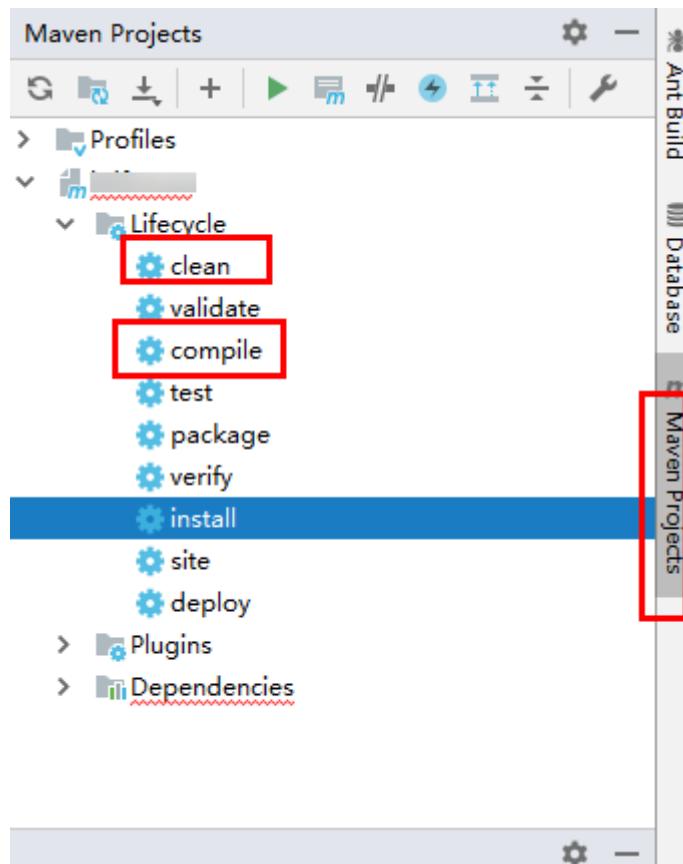
----End

#### 2.9.4.2 Running HCatalog and Viewing Results

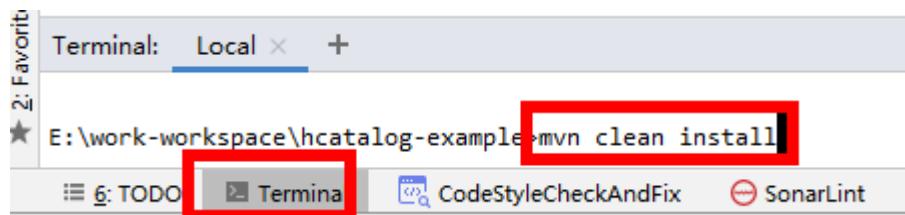
##### Running HCatalog Example Projects

**Step 1** On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name > Lifecycle* and run the **clean** and **compile** scripts.

Figure 2-171 Maven Projects page



**Step 2** In the lower left corner of the IDEA page, click **Terminal** to access the terminal. Run the **mvn clean install** command to compile the package.



If **BUILD SUCCESS** is displayed, the compilation is successful, as shown in the following figure. The **hcatalog-example-\*jar** package is generated in the **target** directory of the sample project.

```
Terminal: Local × +  
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ hcatalog-example ---  
[INFO] Building jar: E:\other-workspase\sample_project\src\hive-examples\hcatalog-example\target\hcatalog-example-1.0-SNAPSHOT.jar  
[INFO]  
[INFO] --- maven-install-plugin:2.4:install (default-install) @ hcatalog-example ---  
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcatalog-example\target\hcatalog-example-1.0-SNAPSHOT.jar  
[INFO] Installing E:\other-workspase\sample_project\src\hive-examples\hcatalog-example\target\hcatalog-example-1.0-SNAPSHOT.pom  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 4.916 s  
[INFO] Finished at: 2020-11-23T16:25:57+08:00  
[INFO] -----
```

#### NOTE

The preceding JAR file names are for reference only. The actual names may vary.

- Step 3** Upload the **hcatalog-example-\*.jar** file generated in the **target** directory in the previous step to the specified directory on Linux, for example, **/opt/hive\_client**, marked as **\$HCAT\_CLIENT**, and ensure that the Hive and YARN clients have been installed. Execute environment variables for the HCAT\_CLIENT to take effect.

```
export HCAT_CLIENT=/opt/hive_client
```

- Step 4** Run the following command to configure environment parameters (client installation path **/opt/hadoopclient** is used as an example):

```
export HADOOP_HOME=/opt/hadoopclient/HDFS/hadoop  
export HIVE_HOME=/opt/hadoopclient/Hive/Beeline  
export HCAT_HOME=$HIVE_HOME/../HCatalog  
export LIB_JARS=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar,$HCAT_HOME/lib/hive-metastore-xxx.jar,$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar,$HIVE_HOME/lib/hive-exec-xxx.jar,$HCAT_HOME/lib/libfb303-xxx.jar,$HCAT_HOME/lib/slf4j-api-xxx.jar,$HCAT_HOME/lib/jdo-api-xxx.jar,$HCAT_HOME/lib/antlr-runtime-xxx.jar,$HCAT_HOME/lib/datanucleus-api-jdo-xxx.jar,$HCAT_HOME/lib/datanucleus-core-xxx.jar,$HCAT_HOME/lib/datanucleus-rdbms-fic-xxx.jar,$HCAT_HOME/lib/log4j-api-xxx.jar,$HCAT_HOME/lib/log4j-core-xxx.jar,$HIVE_HOME/lib/commons-lang-xxx.jar,$HIVE_HOME/lib/hive-exec-xxx.jar  
export HADOOP_CLASSPATH=$HCAT_HOME/lib/hive-hcatalog-core-xxx.jar:$HCAT_HOME/lib/hive-metastore-xxx.jar:$HCAT_HOME/lib/hive-standalone-metastore-xxx.jar:$HIVE_HOME/lib/hive-exec-xxx.jar:$HCAT_HOME/lib/libfb303-xxx.jar:$HADOOP_HOME/etc/hadoop:$HCAT_HOME/conf:$HCAT_HOME/lib/slf4j-api-xxx.jar:$HCAT_HOME/lib/jdo-api-xxx.jar:$HCAT_HOME/lib/antlr-runtime-xxx.jar:$HCAT_HOME/lib/datanucleus-api-jdo-xxx.jar:$HCAT_HOME/lib/datanucleus-core-xxx.jar:$HCAT_HOME/lib/datanucleus-rdbms-fic-xxx.jar:$HCAT_HOME/lib/log4j-api-xxx.jar:$HCAT_HOME/lib/log4j-core-xxx.jar:$HIVE_HOME/lib/commons-lang-xxx.jar:$HIVE_HOME/lib/hive-exec-xxx.jar
```

#### NOTE

- xxx: Indicates the version number of the JAR package. **Change the version numbers of the JAR files specified in LIB\_JARS and HADOOP\_CLASSPATH based on the actual environment.**
- If the multi-instance function is enabled for Hive, perform the configuration in **export HIVE\_HOME=/opt/hadoopclient/Hive/Beeline**. For example, if Hive 1 is used, ensure that the Hive 1 client has been installed before using it. Change the value of **export HIVE\_HOME** to **/opt/hadoopclient/Hive1/Beeline**.

- Step 5** Prepare for the running:

1. Use the Hive client to create source table t1 in beeline: **create table t1(col1 int);**

Insert the following data into t1:

t1.col1
1
1
1
2
2
3

2. Create destination table t2: **create table t2(col1 int,col2 int);**

**Step 6** Use the Yarn client to submit tasks:

```
yarn --config $SHADOOP_HOME/etc/hadoop jar $HCAT_CLIENT/hcatalog-example-1.0-SNAPSHOT.jar com.huawei.bigdata.HCatalogExample -libjars $LIB_JARS t1 t2
```

**Step 7** View the running result. The data in t2 is as follows:

```
0: jdbc:hive2://192.168.1.18:24002,192.168.1.18:24002> select * from t2;
+-----+-----+
| t2.col1 | t2.col2 |
+-----+-----+
| 1       | 3       |
| 2       | 2       |
| 3       | 1       |
+-----+-----+
```

----End

### 2.9.4.3 Running Python and Viewing Results

#### Running Python in CLI Mode

**Step 1** Grant the execution permission for the script of **python-examples** folder. Run the following command in the CLI:

```
chmod +x python-examples -R
```

**Step 2** Enter the service plane IP address of the node where HiveServer is installed in the hosts array of **python-examples/pyCLI\_nosec.py**.



When the multi-instance is enabled: In **python-examples/pyCLI\_nosec.py** the hosts array must be modified. In addition, the port must be set according to the used instance. The port (`hive.server2.thrift.port`) is used for Hive to provide the Thrift service. For example, if the Hive 1 instance is used, port must be set to 21067. (The default ports of Hive to Hive 4 is 21066 to 21070, respectively).

**Step 3** Run the following command:

```
cd python-examples
```

```
python pyCLI_nosec.py
```

**Step 4** In the CLI, view the HQL query results in the example codes.

For example:

```
[['default', '']]  
[{'comment': 'from deserializer', 'columnName': 'tab_name', 'type': 'STRING_TYPE'}]  
['xx']
```



If the following exception occurs:

```
importError: libsasl2.so.2: cannot open shared object file: No such file or directory
```

You can handle the problem as follows:

1. Run the following command to check the LibSASL version in the installed operating system.

```
ldconfig -p|grep sasl
```

If the following is displayed, the current operating system only has the 3.x version.

```
libsasl2.so.3 (libc6,x86-64) => /usr/lib64/libsasl2.so.3  
libsasl2.so.3 (libc6) => /usr/lib/libsasl2.so.3
```

2. If only the 3.x version exists, run the following command to create a soft link.  

```
ln -s /usr/lib64/libsasl2.so.3.0.0 /usr/lib64/libsasl2.so.2
```

----End

#### 2.9.4.4 Running Python3 and Viewing Results

##### Running Python in CLI Mode

**Step 1** Grant the execution permission for the script of **python3-examples** folder. Run the following command in the CLI:

```
chmod +x python3-examples -R.
```

**Step 2** In **python3-examples/pyCLI\_nosec.py**, change the value of host to the service plane IP address of the node where HiveServer is installed, and change the value of port to the port (**hive.server2.thrift.port**) used by Hive to provide the Thrift service. The default value is 21066.



When the multi-instance is enabled: In **python3-examples/pyCLI\_nosec.py**, the hosts must be modified. In addition, the port must be set according to the used instance. The port (**hive.server2.thrift.port**) is used for Hive to provide the Thrift service. For example, if the Hive 1 instance is used, port must be set to 21067. (The default ports of Hive to Hive 4 is 21066 to 21070, respectively).

**Step 3** Run the following commands to start the Python3 client:

```
cd python3-examples
```

```
python pyCLI_nosec.py
```

**Step 4** In the CLI, view the HQL query results in the example codes. For example:

```
('table_name1',)  
(('table_name2',)  
(('table_name3',)  
(('table_name4',)  
(('table_name5',)
```

In the preceding command, **table\_nameX** indicates the actual table name.

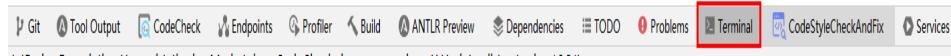
----End

## 2.9.4.5 Running a Spring Boot Sample Project and Viewing Results

### Running the Spring Boot Sample Project in CLI

- Step 1** Click **Terminal** in the lower left corner of the IDEA page to access the terminal. Run the **mvn clean package** command to perform compilation.

```
PS E:\workspace\sample_project\src\springboot\hive-examples\hive-rest-client-example> mvn clean package
```



If "BUILD SUCCESS" is displayed, the compilation is successful. A JAR file containing the **-with-dependencies** field is generated in the **target** directory of the sample project.

```
Terminal: Local +  
[INFO] com/ already added, skipping  
[INFO] com/huawei/ already added, skipping  
[INFO] com/huawei/bigdata/ already added, skipping  
[INFO] META-INF/maven/ already added, skipping  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 32.933 s  
[INFO] Finished at: 2020-11-23T16:18:08+08:00  
[INFO] -----
```

- Step 2** Create a directory on Windows or Linux as the running directory, for example, **D:\hive-rest-client-example** (Windows) or **/opt/hive-rest-client-example** (Linux). Place the JAR file in **Step 1** to this directory, and create the **src/main/resources** subdirectory in the directory. Copy all files in the **resources** directory of the **hive-rest-client-example** project to **resources**.

- Step 3** Run the following commands to start the Spring Boot service:

- Commands for Windows:

```
cd /d D:\hive-rest-client-example
```

```
java -jar hive-rest-client-example-8.3.1-3.3.1-SNAPSHOT-jar-with-dependencies.jar
```

- Commands for Linux:

```
chmod +x /opt/hive-rest-client-example -R
```

```
cd /opt/hive-rest-client-example
```

```
java -jar hive-rest-client-example-8.3.1-3.3.1-SNAPSHOT-jar-with-dependencies.jar
```

#### NOTE

The preceding JAR file name is for reference only. Replace it with the actual one.

**Step 4** Call the Spring Boot sample API of Hive to trigger running the sample code.

- Running method in Windows:

Open the browser and enter <http://localhost:8080/hive/example/executesql> in the address box.

- Running method in Linux:

Run the `curl http://localhost:8080/hive/example/executesql` command on the node where the JAR file is stored in [Step 2](#).

 **NOTE**

The following information may be printed in the log when the sample code is executed. Although the log level is ERROR, the execution result is not affected.

```
ERROR 51320 --- [c-8-EventThread] o.a.c.framework.imps.EnsembleTracker : Invalid config event received: {version=100000000, server.48=IP address of the ZooKeeper node.ZooKeeper port number.ZooKeeper port number:participant...}
```

**Step 5** View the HQL query results in the sample code.

- The following information is displayed if the sample project is successful in Windows:

```
===== Hive Example Start =====
Start create table.
Table created successfully.
Start to insert data into the table.
Inserting data to the table succeeded.
Start to query table data.
Query result :
employees_infoa.id    employees_infoa.age    employees_infoa.name
1      31      SJK
2      25      HS
3      28      HT
```

```
Querying table data succeeded.
Start to delete the table.
Table deleted successfully.
===== Hive Example End =====
```

- The following information is displayed if the sample project is successful in Linux:

```
===== Hive Example Start =====
Start create table.
Table created successfully.
Start to insert data into the table.
Inserting data to the table succeeded.
Start to query table data.
Query result :
employees_infoa.id    employees_infoa.age    employees_infoa.name
1      31      SJK
2      25      HS
3      28      HT
```

```
Querying table data succeeded.
Start to delete the table.
Table deleted successfully.
===== Hive Example End =====
```

----End

## 2.9.5 More Information

### 2.9.5.1 Interface Reference

### 2.9.5.1.1 JDBC

The Hive Java database connectivity (JDBC) interface complies with the Java JDBC driver standard.

#### NOTE

As a data warehouse, Hive does not support all JDBC APIs. For example, if transactional operations, such as rollback and setAutoCommit, are performed, SQL exceptions like **Method not supported** will occur.

### 2.9.5.1.2 Hive SQL

Hive SQL supports all features in Hive-3.1.0. For details, see <https://cwiki.apache.org/confluence/display/hive/languagemanual>.

**Table 2-111** describes the extended Hive statements provided by FusionInsight.

**Table 2-111** Extended Hive statements

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format]   STORED BY 'storage.handler.cl ass.name' [WITH SERDEPROPERTIE S (...) ] ..... [TBLPROPERTIES ("groupId"=" group1 ","locatorId"="loc ator1")] ...;</pre>	<p>The statement is used to create a Hive table and specify locators on which table data files locate. For details, see "Component Operation Guide" &gt; "Using Hive" &gt; "Using HDFS Colocation to Store Hive Tables" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i>.</p>	<pre>CREATE TABLE tab1 (id INT, name STRING) row format delimited fields terminated by '\t' stored as RCFILE TBLPROPERTIES("group1 ","locatorId"="locator1");</pre>	The statement is used to create table <b>tab1</b> and specify locator1 on which the table data of <b>tab1</b> locates.

Extended Syntax	Syntax Description	Syntax Example	Example Description
<pre>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] [STORED AS file_format]   STORED BY 'storage.handler.cl ass.name' [WITH SERDEPROPERTIE S (...) ] ... [TBLPROPERTIES ('column.encode. columns'=col_na me1,col_name2')  'column.encode.i ndices'=col_id1,c ol_id2', 'column.encode.c lassname'=encod e_classname')]...;</pre>	<p>The statement is used to create a hive table and specify the table encryption column and encryption algorithm. For details, see "Component Operation Guide" &gt; "Using Hive" &gt; "Using the Hive Column Encryption Function" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i>.</p>	<pre>create table encode_test(id INT, name STRING, phone STRING, address STRING) ROW FORMAT SERDE 'org.apache.hadoo p.hive.serde2.lazy. LazySimpleSerDe' WITH SERDEPROPERTIE S ('column.encode.i ndices'=2,3', 'column.encode.cl assname='org.apa che.hadoop.hive.s erde2.SMS4Rewrit er') STORED AS TEXTFILE;</pre>	<p>The statement is used to create table <b>encode_test</b> and specify that column 2 and column 3 will be encrypted using the <b>org.apache.hadoop.hive.serde2.SMS4Rewriter</b> encryption algorithm class during data insertion.</p>

Extended Syntax	Syntax Description	Syntax Example	Example Description
<code>REMOVE TABLE hbase_tablename [WHERE where_condition];</code>	The statement is used to delete data that meets criteria from the Hive on HBase table. For details, see "Component Operation Guide" > "Using Hive" > "Deleting Single-Row Records from Hive on HBase" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i> .	<code>remove table hbase_table1 where id = 1;</code>	The statement is used to delete data that meets the criterion of "id = 1" from the table.
<code>CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_ name (col_name data_type [COMMENT col_comment], ...) [ROW FORMAT row_format] STORED AS inputformat 'org.apache.hadoop.hive.contrib.format.SpecifiedDelimiterInputFormat' outputformat 'org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat';</code>	The statement is used to create a hive table and specify that the table supports customized row delimiters. For details, see "Component Operation Guide" > "Using Hive" > "Customizing Row Separators" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)</i> in the <i>MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i> .	<code>create table blu(time string, num string, msg string) row format delimited fields terminated by ',' stored as inputformat 'org.apache.hadoop.hive.contrib.format.SpecifiedDelimiterInputFormat' outputformat 'org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat';</code>	The statement is used to create table <b>blu</b> and set <b>inputformat</b> to <b>SpecifiedDelimiterInputFormat</b> so that the query row delimiter can be specified during the query.

### 2.9.5.1.3 WebHCat

#### NOTE

- The following uses the service IP address of WebHCat and the WebHCat HTTP port configured during the installation as an example.
- The **user.name** parameter needs to be added to APIs except **:version**, **status**, **version**, **version/hive**, and **version/hadoop**.

#### 1. :version(GET)

##### - Description

Obtain the list of the response types supported by WebHCat.

##### - URL

`http://www.myserver.com/templeton/:version`

##### - Parameter

Parameter	Description
<code>:version</code>	WebHCat version number (Currently this must be v1.)

##### - Returned result

Parameter	Description
<code>responseTypes</code>	A list of all supported response types

##### - Example

```
curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1'
```

#### 2. status (GET)

##### - Description

Obtain the status of the current server.

##### - URL

`http://www.myserver.com/templeton/v1/status`

##### - Parameter

None

##### - Returned result

Parameter	Description
<code>status</code>	<b>OK</b> is returned if the WebHCat server was contacted.
<code>version</code>	String containing the version number similar to "v1"

- Example  
`curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/status'`

### 3. version (GET)

- Description  
Obtain the WebHCat version of the server.
- URL  
`http://www.myserver.com/templeton/v1/version`
- Parameter  
None
- Returned result

Parameter	Description
supportedVersions	A list of all supported versions
version	The current version

- Example  
`curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/version'`

### 4. version/hive (GET)

- Description  
Obtain the Hive version of the server.
- URL  
`http://www.myserver.com/templeton/v1/version/hive`
- Parameter  
None
- Returned result

Parameter	Description
module	hive
version	Hive version

- Example  
`curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/version/hive'`

### 5. version/hadoop (GET)

- Description  
Obtain the Hadoop version of the server.
- URL  
`http://www.myserver.com/templeton/v1/version/hadoop`
- Parameter  
None
- Returned result

Parameter	Description
module	hadoop
version	Hadoop version

- Example  

```
curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/version/hadoop'
```
6. **ddl (POST)**
- Description  
 Execute a DDL statement.
  - URL  
<http://www.myserver.com/templeton/v1/ddl>
  - Parameter
- | Parameter   | Description   |
|-------------|---|
| exec        | The HCatalog ddl string to execute  |
| group       | The user group to use when creating a table                                     |
| permissions | The permissions string to use when creating a table. The format is "rwxrw-r-x". |
- Returned result
- | Parameter | Description   |
|-----------|---|
| stdout    | A string containing the result HCatalog sent to standard out (possibly empty)   |
| stderr    | A string containing the result HCatalog sent to standard error (possibly empty) |
| exitcode  | The exitcode HCatalog returned  |
- Example  

```
curl -i -u : --negotiate -d exec="show tables" 'http://10.64.35.144:21055/templeton/v1/ddl?user.name=user1'
```

7. **ddl/database (GET)**
- Description  
 List all databases.
  - URL  
<http://www.myserver.com/templeton/v1/ddl/database>
  - Parameter

Parameter	Description
like	List only databases whose names match the specified pattern.

- Returned result

Parameter	Description
databases	A list of database names

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/ddl/database?user.name=user1'
```

## 8. ddl/database/:db (GET)

- Description

Obtain details about a specified database.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db>

- Parameter

Parameter	Description
:db	The database name

- Returned result

Parameter	Description
location	The database location
comment	The database comment. If there is no database comment, the value is null.
database	The database name
owner	The database owner
owertype	The type of the database owner

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/ddl/database/default?  
user.name=user1'
```

## 9. ddl/database/:db (PUT)

- Description

Create a database.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db>

- Parameter

Parameter	Description
:db	The database name
group	The user group to use
permission	The permissions string to use
location	The database location
comment	A comment for the database, like a description
properties	The database properties

- Returned result

Parameter	Description
database	The database name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"location": "/tmp/a", "comment": "my db", "properties": {"a": "b"}}' http://10.64.35.144:21055/templeton/v1/ddl/database/db2?user.name=user1'
```

## 10. ddl/database/:db (DELETE)

- Description  
Delete a database.
- URL  
<http://www.myserver.com/templeton/v1/ddl/database/:db>
- Parameter

Parameter	Description
:db	The database name
IfExists	Hive returns an error if the database specified does not exist, unless <b>IfExists</b> is set to true.
option	Parameter set to either <b>cascade</b> or <b>restrict</b> . <b>restrict</b> will remove the schema if all the tables are empty. <b>cascade</b> removes everything including data and definitions.

- Returned result

Parameter	Description
database	The database name

- Example

```
curl -i -u : --negotiate -X DELETE 'http://10.64.35.144:21055/templeton/v1/ddl/database/db3?ifExists=true&user.name=user1'
```

11. `ddl/database/:db/table` (GET)

- Description

List all tables in a database.

- URL

`http://www.myserver.com/templeton/v1/ddl/database/:db/table`

- Parameter

Parameter	Description
<code>:db</code>	The database name
<code>like</code>	List only tables whose names match the specified pattern.

- Returned result

Parameter	Description
<code>database</code>	A list of table names
<code>tables</code>	The database name

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/ddl/database/default/table?user.name=user1'
```

12. `ddl/database/:db/table/:table` (GET)

- Description

Obtain details of a table.

- URL

`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table`

- Parameter

Parameter	Description
<code>:db</code>	The database name
<code>:table</code>	The table name
<code>format</code>	Set "format=extended" to see additional information (using "show table extended like").

- Returned result

Parameter	Description
<code>columns</code>	A list of column names and types

Parameter	Description
database	The database name
table	The table name
partitioned	Indicates whether a table is a partition table. This parameter is available only when the table format is <b>extended</b> .
location	Location of a table. This parameter is available only when the table format is <b>extended</b> .
outputformat	Output format. This parameter is available only when the table format is <b>extended</b> .
inputformat	Input format. This parameter is available only when the table format is extended.
owner	Owner of a table. This parameter is available only when the table format is extended.
partitionColumns	Partition column. This parameter is available only when the table format is extended.

- Example

```
curl -i -u : --negotiate 'http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1?format=extended&user.name=user1'
```

### 13. `ddl/database/:db/table/:table` (PUT)

- Description

Create a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table>

- Parameter

Parameter	Description
:db	The database name
:table	The new table name
group	The user group to use when creating a table
permissions	The permissions string to use when creating a table

Parameter	Description
external	Allows you to specify a location so that Hive does not use the default location for this table.
ifNotExists	If the value <b>true</b> , you will not receive an error if the table already exists.
comment	Comment for the table
columns	A list of column descriptions, including name, type, and an optional comment
partitionedBy	A list of column descriptions used to partition the table. Like the <b>columns</b> parameter this is a list of name, type, and comment file.
clusteredBy	Bucket column description, including the <b>columnNames</b> , <b>sortedBy</b> , and <b>numberOfBuckets</b> parameters. The <b>columnNames</b> parameter includes <b>columnName</b> and sorting sequence ( <b>ASC</b> for ascending or <b>DESC</b> for descending).
format	Storage format description including parameters for <b>rowFormat</b> , <b>storedAs</b> , and <b>storedBy</b> .
location	The HDFS path
tableProperties	A list of table property names and values (key/value pairs).

- Returned result

Parameter	Description
database	The new table name
table	The database name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"columns": [{"name": "id", "type": "int"}, {"name": "name", "type": "string"}], "comment": "hello", "format": {"storedAs": "orc"} }' 'http://10.64.35.144:21055/templeton/v1/ddl/database/db3/table/tbl1?user.name=user1'
```

#### 14. `ddl/database/:db/table/:table (POST)`

- Description

Rename a table.

- URL  
`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table`
- Parameter

Parameter	Description
:db	The database name
:table	The existing (old) table name
rename	The new table name

- Returned result

Parameter	Description
database	The database name
table	The new table name

- Example

```
curl -i -u : --negotiate -d rename=table1 'http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/tbl1?user.name=user1'
```

## 15. ddl/database/:db/table/:table (DELETE)

- Description  
Delete a table.
- URL  
`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table`
- Parameter

Parameter	Description
:db	The database name
:table	The table name
IfExists	If the value is <b>true</b> , you will not receive an error.

- Returned result

Parameter	Description
database	The database name
table	The table name

- Example

```
curl -i -u : --negotiate -X DELETE 'http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/table2?IfExists=true&user.name=user1'
```

## 16. ddl/database/:db/table/:existingtable/like/:newtable (PUT)

- Description  
Create a table that is the same as an existing one.
- URL  
<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:existingtable/like/:newtable>
- Parameter

Parameter	Description
:db	The database name
:existingtable	The existing table name
:newtable	The new table name
group	The user group to use when creating a table
permissions	The permissions string to use when creating a table
external	Allows you to specify a location so that Hive does not use the default location for this table.
ifNotExists	If the value is <b>true</b> , you will not receive an error if the table already exists.
location	The HDFS path

- Returned result

Parameter	Description
database	The database name
table	The new table name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"ifNotExists": "true"}' 'http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/like/tt1?user.name=user1'
```

## 17. ddl/database/:db/table/:table/partition(GET)

- Description  
List partition information of a table.
- URL  
<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition>
- Parameter

Parameter	Description
:db	The database name
:table	The table name

- Returned result

Parameter	Description
database	The database name
table	The table name
partitions	A list of partition values and of partition names

- Example

```
curl -i -u : --negotiate http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition?user.name=user1
```

#### 18. ddl/database/:db/table/:table/partition/:partition(GET)

- Description

List information about a partition of a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition>

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:partition	The partition name, col_name='value' list. Be careful to properly encode the quote for http, for example, "country=%27algeria%27".

- Returned result

Parameter	Description
database	The database name
table	The table name
partition	The partition name
partitioned	True if the table is partitioned
location	Location of table

Parameter	Description
outputFormat	Output format
columns	List of column names, types, and comments
owner	The owner's user name
partitionColumns	List of the partition columns
inputFormat	Input format
totalNumberFiles	Number of files in a partition
totalFileSize	Total file size in a partition
maxFileSize	Maximum file size
minFileSize	Minimum file size
lastAccessTime	Last access time
lastUpdateTime	Last update time

- Example

```
curl -i -u : --negotiate http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition/dt=1?user.name=user1
```

#### 19. `ddl/database/:db/table/:table/partition/:partition(PUT)`

- Description  
Add a table partition.
- URL  
<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition>
- Parameter

Parameter	Description
:db	The database name
:table	The table name
group	The user group to use
permissions	The permissions string to use
location	The location for partition creation
ifNotExists	If the value is <b>true</b> , return an error if the partition already exists.

- Returned result

Parameter	Description
database	The database name
table	The table name
partitions	The partition name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{}' http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition/dt=10?user.name=user1
```

20. `ddl/database/:db/table/:table/partition/:partition(DELETE)`

- Description

Delete a table partition.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/partition/:partition>

- Parameter

Parameter	Description
:db	The database name
:table	The table name
group	The user group to use
permissions	The permissions string to use. The format is "rwxrw-r-x".
IfExists	Hive returns an error if the partition specified does not exist, unless ifExists is set to true.

- Returned result

Parameter	Description
database	The database name
table	The table name
partitions	The partition name

- Example

```
curl -i -u : --negotiate -X DELETE -HContent-type:application/json -d '{}' http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/x1/partition/dt=10?user.name=user1
```

21. `ddl/database/:db/table/:table/column(GET)`

- Description

Obtain the column list of a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column>

- Parameter

Parameter	Description
:db	The database name
:table	The table name

- Returned result

Parameter	Description
database	The database name
table	The table name
columns	A list of column names and types

- Example

```
curl -i -u : --negotiate http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/column?user.name=user1
```

## 22. ddl/database/:db/table/:table/column/:column(GET)

- Description

Obtain details about a column in a table.

- URL

<http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column>

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:column	The column name

- Returned result

Parameter	Description
database	The database name
table	The table name
column	Column name and type

- Example

```
curl -i -u : --negotiate http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/column/id?user.name=user1
```

23. `ddl/database/:db/table/:table/column/:column(PUT)`

- Description  
Add a column to a table.
- URL  
`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/column/:column`
- Parameter

Parameter	Description
<code>:db</code>	The database name
<code>:table</code>	The table name
<code>:column</code>	The column name
<code>type</code>	The type of column to add, like "string" or "int"
<code>comment</code>	The column comment, like a description

- Returned result

Parameter	Description
<code>database</code>	The database name
<code>table</code>	The table name
<code>column</code>	The column name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"type": "string", "comment": "new column"}' http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/column/name?user.name=user1
```

24. `ddl/database/:db/table/:table/property(GET)`

- Description  
Obtain table properties.
- URL  
`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/property`
- Parameter

Parameter	Description
<code>:db</code>	The database name
<code>:table</code>	The table name

- Returned result

Parameter	Description
database	The database name
table	The table name
properties	Property list

- Example

```
curl -i -u : --negotiate http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/
property?user.name=user1
```

25. `ddl/database/:db/table/:table/property/:property(GET)`

- Description

Obtain a specified property value of a table.

- URL

`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/
property/:property`

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:property	The property name

- Returned result

Parameter	Description
database	The database name
table	The table name
property	Property list

- Example

```
curl -i -u : --negotiate http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/
property/last_modified_by?user.name=user1
```

26. `ddl/database/:db/table/:table/property/:property(PUT)`

- Description

Add a property value for a table.

- URL

`http://www.myserver.com/templeton/v1/ddl/database/:db/table/:table/
property/:property`

- Parameter

Parameter	Description
:db	The database name
:table	The table name
:property	The property name
value	The property value

- Returned result

Parameter	Description
database	The database name
table	The table name
property	The property name

- Example

```
curl -i -u : --negotiate -X PUT -HContent-type:application/json -d '{"value": "my value"}' http://10.64.35.144:21055/templeton/v1/ddl/database/default/table/t1/property/mykey?  
user.name=user1
```

## 27. mapreduce/jar(POST)

- Description

Execute an MR task. Before the execution, upload MR JAR packages to HDFS.

- URL

<http://www.myserver.com/templeton/v1/mapreduce/jar>

- Parameter

Parameter	Description
jar	Name of the jar file for Map Reduce to use
class	Name of the class for Map Reduce to use
libjars	Comma-separated jar files to include in the classpath
files	Comma-separated files to be copied to the map reduce cluster
arg	Main class input parameter
define	Set a Hadoop configuration variable using the syntax define=NAME=VALUE.

Parameter	Description
statusdir	A directory where WebHCat will write the status of the Map Reduce job. If provided, it is the caller's responsibility to remove this directory when done.
enablelog	If <b>statusdir</b> is set and <b>enablelog</b> is <b>true</b> , collect Hadoop job configuration and logs in to a directory named <b>\$statusdir/logs</b> after the job finishes. Both completed and failed attempts are logged. The layout of subdirectories in <b>\$statusdir/logs</b> is:  logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog This parameter supports only Hadoop 1.X.
callback	Define a URL to be called upon job completion. You may embed a specific job ID into this URL using \$jobId. This tag will be replaced in the callback URL with this job's job ID.

- Returned result

Parameter	Description
id	A string containing the job ID similar to "job_201110132141_0001"

- Example

```
curl -i -u : --negotiate -d jar="/tmp/word.count-0.0.1-SNAPSHOT.jar" -d
class=com.huawei.word.count.WD -d statusdir="/output" "http://10.64.35.144:210559111/
templeton/v1/mapreduce/jar?user.name=user1"
```

## 28. mapreduce/streaming(POST)

- Description

Submit an MR task in Streaming mode.

- URL  
`http://www.myserver.com/templeton/v1/mapreduce/streaming`
- Parameter

Parameter	Description
input	Location of the input data in Hadoop
output	Location in which to store the output data. If not specified, WebHCat will store the output in a location that can be discovered using the queue resource.
mapper	Location of the mapper program in Hadoop
reducer	Location of the reducer program in Hadoop
files	Add an HDFS file to the distributed cache.
arg	Set a program argument.
define	Set a Hadoop configuration variable using the syntax <code>define=NAME=VALUE</code> .
cmdenv	Set an environment variable using the syntax <code>cmdenv=NAME=VALUE</code> .
statusdir	A directory where WebHCat will write the status of the Map Reduce job. If provided, it is the caller's responsibility to remove this directory when done.

Parameter	Description
enablelog	If <b>statusdir</b> is set and <b>enablelog</b> is <b>true</b> , collect Hadoop job configuration and logs in to a directory named <b>\$statusdir/logs</b> after the job finishes. Both completed and failed attempts are logged. The layout of subdirectories in <b>\$statusdir/logs</b> is:  logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog This parameter supports only Hadoop 1.X.
callback	Define a URL to be called upon job completion. You may embed a specific job ID into this URL using \$jobId. This tag will be replaced in the callback URL with this job's job ID.

- Returned result

Parameter	Description
id	A string containing the job ID similar to "job_201110132141_0001"

- Example

```
curl -i -u : --negotiate -d input=/input -d output=/oooo -d mapper=/bin/cat -d reducer="/usr/bin/wc -w" -d statusdir="/output" 'http://10.64.35.144:21055/templeton/v1/mapreduce/streaming?user.name=user1'
```

#### NOTE

For details about prerequisites for using this interface, see [Rules](#).

### 29. /hive(POST)

- Description  
Run Hive commands.
- URL  
<http://www.myserver.com/templeton/v1/hive>

## - Parameter

Parameter	Description
execute	String containing an entire, short Hive program to run
file	HDFS file name of a Hive program to run
files	Comma-separated files to be copied to the map reduce cluster
arg	Set a program argument.
define	Hive configuration. The format is define=key=value. If multiple instances are used, you need to configure the scratch dir for them. For example, the WebHCat instance uses define=hive.exec.scratchdir=/tmp/ <b>hive-scratch</b> . The WebHCat1 instance uses define=hive.exec.scratchdir=/tmp/ <b>hive1-scratch</b> . The same rule applies to other instances.
statusdir	A directory where WebHCat will write the status of the Hive job. If provided, it is the caller's responsibility to remove this directory when done.
enablelog	If <b>statusdir</b> is set and <b>enablelog</b> is <b>true</b> , collect Hadoop job configuration and logs in to a directory named <b>\$statusdir/logs</b> after the job finishes. Both completed and failed attempts are logged. The layout of subdirectories in <b>\$statusdir/logs</b> is:  logs/\$job_id (directory for \$job_id) logs/\$job_id/job.xml.html logs/\$job_id/\$attempt_id (directory for \$attempt_id) logs/\$job_id/\$attempt_id/stderr logs/\$job_id/\$attempt_id/stdout logs/\$job_id/\$attempt_id/syslog This parameter supports only Hadoop 1.X.

Parameter	Description
callback	Define a URL to be called upon job completion. You may embed a specific job ID into this URL using \$jobId. This tag will be replaced in the callback URL with this job's job ID.

- Returned result

Parameter	Description
id	A string containing the job ID similar to "job_201110132141_0001"

- Example

```
curl -i -u : --negotiate -d execute="select count(*) from t1" -d statusdir="/output" -d
define=hive.exec.scratchdir=/tmp/hive-scratch "http://10.64.35.144:21055/templeton/v1/hive?
user.name=user1"
```

### 30. jobs(GET)

- Indicates IDs of all obtained jobs.
- URL <http://www.myserver.com/templeton/v1/jobs>
- Parameter

Parameter	Description
fields	If <b>fields</b> set to *, the request will return full details of the job. If <b>fields</b> is missing, will only return the job ID. Currently the value can only be *, other values are not allowed and will throw exception.
jobid	If <b>jobid</b> is present, only the records whose job ID is lexicographically greater than <b>jobid</b> are returned. For example, if <b>jobid</b> = "job_201312091733_0001", the jobs whose job ID is greater than "job_201312091733_0001" are returned. The number of records returned depends on the value of <b>numrecords</b> .

Parameter	Description
numrecords	If the <b>jobid</b> and <b>numrecords</b> parameters are present, the top numrecords records appearing after <b>jobid</b> will be returned after sorting the job ID list lexicographically. If the <b>jobid</b> parameter is missing and <b>numrecords</b> is present, the top <b>numrecords</b> will be returned after lexicographically sorting the job ID list. If the <b>jobid</b> parameter is present and <b>numrecords</b> is missing, all the records whose job ID is greater than <b>jobid</b> are returned.
showall	If this parameter is set to <b>true</b> , all jobs can be obtained. If this parameter is set to <b>false</b> , only jobs submitted by the current user can be obtained. This parameter is set to <b>false</b> by default.

- Returned result

Parameter	Description
id	Job id
detail	Job details if <b>showall</b> is set to <b>true</b> ; otherwise <b>null</b> .

- Example

```
curl -i -u : --negotiate "http://10.64.35.144:21055/templeton/v1/jobs?user.name=user1"
```

### 31. jobs/:jobid(GET)

- Indicates the information of the specified job.
- URL  
<http://www.myserver.com/templeton/v1/jobs/:jobid>
- Parameter

Parameter	Description
jobid	The job ID to check. This is the ID received when the job was created.

- Returned result

Parameter	Description
status	A JSON object containing the job status information
profile	A JSON object containing the job profile information. WebHCat passes along the information in the JobProfile object, which is subject to change from one Hadoop version to another.
id	The job ID
percentComplete	The job completion percentage, for example "75% complete". If the job is completed, the value is null.
user	User name of the job creator
callback	The callback URL, if any
userargs	A JSON object representing the argument names and values for the job submission request
exitValue	The job's exit value

- Example

```
curl -i -u : --negotiate "http://10.64.35.144:21055/templeton/v1/jobs/job_1440386556001_0255?  
user.name=user1"
```

### 32. jobs/:jobid(DELETE)

- Indicates kill jobs.
- URL  
<http://www.myserver.com/templeton/v1/jobs/:jobid>
- Parameter

Parameter	Description
jobid	The job ID to delete. This is the ID received when the job was created.

- Returned result

Parameter	Description
user	Indicates the user that submits jobs.

Parameter	Description
status	A JSON object containing the job status information
profile	A JSON object containing the job profile information. WebHCat passes along the information in the JobProfile object, which is subject to change from one Hadoop version to another.
id	The job ID
callback	The callback URL, if any

- Example

```
curl -i -u : --negotiate -X DELETE "http://10.64.35.143:21055/templeton/v1/jobs/job_1440386556001_0265?user.name=user1"
```

## 2.9.5.2 FAQ

### 2.9.5.2.1 Problem performing GSS wrap Message Is Displayed Due to IBM JDK Exceptions

#### Question

IBM JDK is abnormal, and the **Problem performing GSS wrap** message is displayed.

#### Answer

##### Possible Causes

The Hive connection duration exceeds the user authentication timeout duration (one day by default), causing authentication failure.

##### NOTE

The IBM JDK mechanism is different from the Oracle JDK mechanism. IBM JDK executes time check after authentication and login, but does not check external time update. This results in refreshing failure even Hive relogin is invoked explicitly.

##### Solution

When one Hive connection fails, disable this connection, and create a connection to continue performing previous operations.

### 2.9.5.2.2 "version 'GLIBCXX\_3.4.21' not found" Exception Occurs When the Python 3 Secondary Development Program Is Used to Access Hive

#### Question

"version 'GLIBCXX\_3.4.21' not found" exception occurs when the Python 3 secondary development program is used to access Hive.

## Answer

- **Possible Causes**

The default GCC version of the OS used by the client is too early, and the dynamic link library (DLL) does not contain **GLIBCXX\_3.4.21**.

- **Solution**

Upgrade the GCC version of the client OS to 5.4.0.

# 2.10 IoTDB Development Guide

## 2.10.1 Overview

### 2.10.1.1 Application Development Overview

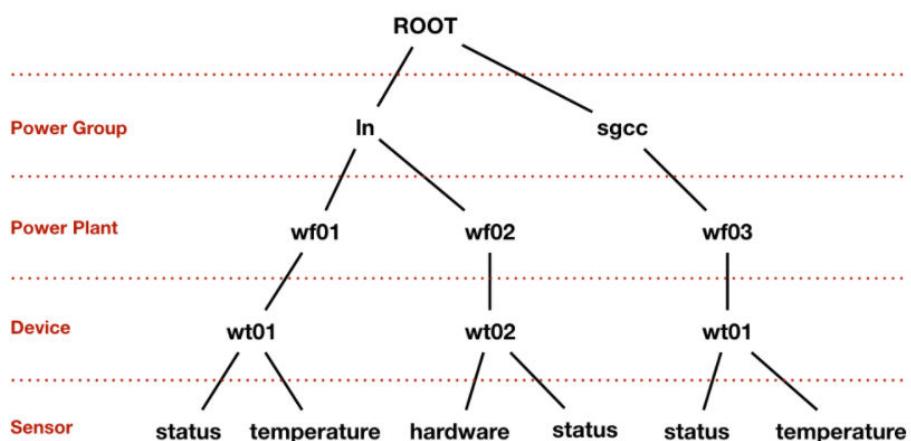
#### Introduction to IoTDB

IoTDB is a data management engine that integrates collection, storage, and analysis of time series data. It features lightweight, high performance, and ease of use. It perfectly interconnects with the Hadoop and Spark ecosystems and meets the requirements of high-speed write and complex analysis and query on massive time series data in industrial IoT applications.

#### 2.10.1.2 Basic Concepts

The following uses an electric power scenario as an example to describe how to create a correct data model in IoTDB.

**Figure 2-172** Hierarchical structure of attributes in an electric power scenario



**Figure 2-172** shows the power group layer, power plant layer, device layer, and sensor layer. **ROOT** is a root node, and each node at the sensor layer is a leaf node. According to the IoTDB syntax, the path from **ROOT** to a leaf node is separated by a dot (.). The complete path is used to name a time series in the

IoTDB. For example, the time series name corresponding to the path on the left in the figure is **ROOT.ln.wf01.wt01.status**.

## Basic Concepts

- **Device**

A device is a machine equipped with sensors in actual scenarios. In IoTDB, all sensors must have their corresponding devices.

- **Sensor**

A sensor is a detection machine in actual scenarios. It can sense the information to be measured, and can transform the sensed information into an electrical signal or other desired form of information output and send it to IoTDB. In IoTDB, all data and paths stored are organized in units of sensors.

- **Database**

A database is also called a storage group. You can set any prefixed path as a database. If there are four time series, for example, **root.vehicle.d1.s1**, **root.vehicle.d1.s2**, **root.vehicle.d2.s1**, and **root.vehicle.d2.s2**, two devices **d1** and **d2** in the path **root.vehicle** may belong to the same owner or manufacturer, so **d1** and **d2** are closely related. In this case, the prefixed path **root.vehicle** can be designated as a database, which will enable IoTDB to store the data of all devices under it in the same folder. Newly added devices in **root.vehicle** will also belong to this storage group.

 **NOTE**

- A proper number of storage groups can improve performance. There is neither the slowdown of the system due to frequent I/O switching (which will also take up excessive memory and result in frequent memory-file switching) caused by too many storage files or folders, nor the block of write commands caused by too few storage files or folders (which reduces concurrency).
- You need to balance the storage group settings of storage files according to their own data size and usage scenarios to achieve better system performance.

- **Time series**

Time series is a core concept in IoTDB. The time series can be considered as the complete path of the sensor that generates the time series data. In IoTDB, all time series must start with root and end with the sensor.

### 2.10.1.3 Development Process

This section describes how to use Java APIs to develop IoTDB applications.

[Figure 2-173](#) and [Table 2-112](#) describe the phases in the development process.

Figure 2-173 IoTDB application development process

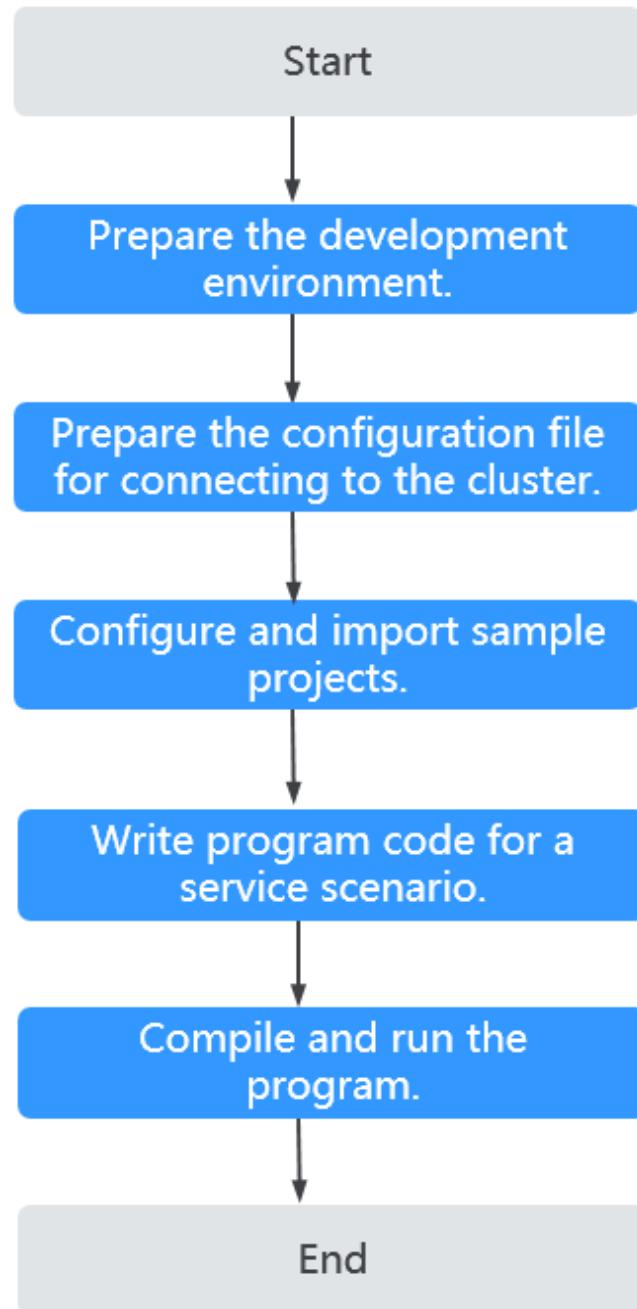


Table 2-112 Description of the IoTDB application development process

Phase	Description	Reference
Prepare the development environment.	Before developing an application, you need to prepare the development environment. You are advised to use the Java language and IntelliJ IDEA tool for development. In addition, you need to complete the initial configuration of the JDK and Maven.	<a href="#">Prepare the Development Environment</a>

Phase	Description	Reference
Prepare the Configuration File for Connecting to the Cluster.	<p>During application development or running, you need to connect to the MRS cluster through cluster configuration files. Configuration files include cluster component information files and user files used for security authentication. You can obtain related content from the created MRS cluster.</p> <p>Nodes used for program commissioning or running must be able to communicate with nodes in the MRS cluster and the hosts domain name must be configured.</p>	<a href="#">Preparing the Configuration Files for Connecting to the Cluster</a>
Configure and import the sample project.	IoTDB provides multiple sample programs in different scenarios. You can obtain sample projects and import them to the local development environment for program learning.	<a href="#">Configuring and Importing Sample Projects</a>
Develop programs based on business scenarios.	Sample projects of the Java language are provided, including JDBC and Session connection modes. A sample project covers the entire process from creating a storage group, creating a time sequence, inserting data, to deleting a storage group.	<a href="#">Application Development</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>

#### 2.10.1.4 IoTDB Sample Project

To obtain an MRS sample project, visit <https://github.com/huaweicloud/huaweicloud-mrs-example> and switch to the branch that matches the MRS cluster version. Download the package to the local PC and decompress it to obtain the sample project of each component.

MRS provides the following IoTDB sample projects.

**Table 2-113** IoTDB sample projects

Sample Project Location	Description
iotdb-examples/iotdb-flink-example	<p>Program for using Flink to access IoTDB data, including FlinkIoTDBSink and FlinkIoTDBSource data.</p> <p>FlinkIoTDBSink can use Flink jobs to write time series data to IoTDB. FlinkIoTDBSource reads time series data from IoTDB through Flink jobs and prints the data. For details, see <a href="#">IoTDB Flink</a>.</p>

Sample Project Location	Description
iotdb-examples/iotdb-jdbc-example	Java sample program for IoTDB JDBC to process data  This example demonstrates how to use JDBC APIs to connect to IoTDB and run IoTDB SQL statements. For details, see <a href="#">IoTDB JDBC</a> .
iotdb-examples/iotdb-kafka-example	Sample program for accessing IoTDB data through Kafka  This program demonstrates how to send time series data to Kafka and then use multiple threads to write the data to IoTDB. For details, see <a href="#">IoTDB Kafka</a> .
iotdb-examples/iotdb-session-example	Java sample program for IoTDB Session to process data  This example demonstrates how to use Session to connect to IoTDB and run IoTDB SQL statements. For details, see <a href="#">IoTDB Session</a> .
iotdb-examples/iotdb-udf-exmaple	This sample program describes how to implement a simple IoTDB user-defined function (UDF). For details, see <a href="#">IoTDB UDF Program</a> .

## 2.10.2 Environment Preparations

### 2.10.2.1 Prepare the Development Environment

[Table 2-114](#) describes the environment required for application development.

**Table 2-114** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>Development environment: Windows 7 or later versions</li><li>Operating environment: Windows or Linux If the program needs to be commissioned locally, the operating environment must be able to communicate with network on the cluster service plane.</li></ul>

Item	Description
JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>● For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>● For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● For security purposes, the server supports only TLS V1.2 or later.</li><li>● By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextget-stancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextget-stancetls-oracle#matchsslcontext_tls</a>.</li><li>● For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA	<p>Tool used for developing IoTDB applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● If you use IBM JDK, ensure that the JDK configured in IntelliJ IDEA is IBM JDK.</li><li>● If you use Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is Oracle JDK.</li><li>● If you use OpenJDK, ensure that the JDK configured in IntelliJ IDEA is OpenJDK.</li><li>● Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Maven	<p>Basic configurations of the development environment. Maven is used for project management throughout the lifecycle of software development.</p> <p>Huawei provides an open-source mirror site, Huawei Mirrors. You can download the dependent JAR packages of the sample projects from this site. You can download the rest open-source JAR packages from the Maven central repository or other user-defined repositories. For details, see <a href="#">Configuring Huawei Open-Source Mirrors</a>.</p>

Item	Description
7-zip	This tool is used to decompress *.zip and *.rar files. <b>7-Zip 16.04</b> is supported.

## 2.10.2.2 Preparing the Configuration Files for Connecting to the Cluster

### Preparing the Configuration Files of the Running Environment

During the development or a test run of the program, you need to use cluster configuration files to connect to an MRS cluster. The configuration files usually contain the cluster component information file and user files used for security authentication. You can obtain the required information from the created MRS cluster.

Nodes used for program debugging or running must be able to communicate with nodes within the MRS cluster, and the hosts domain name must be configured.

- Scenario 1: Preparing the configuration files required for debugging in the local Windows development environment
  - a. Log in to FusionInsight Manager. In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Configuration Files Only**, and click **OK**. After the client file package is generated, download it to the local PC as prompted and decompress it.  
For example, if the client configuration file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar**. Decompress the package to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC.
  - b. Copy all items from the **hosts** file in the decompression directory to the **hosts** file on the node where the client is installed. Ensure that the network communication between the local PC and hosts listed in the **hosts** file is normal.

#### NOTE

- If the client host is outside the cluster, configure network connections to the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored, for example, in **C:\WINDOWS\system32\drivers\etc\hosts**.
- Scenario 2: Preparing the configuration files required for running the program in a Linux environment
  - a. Install the client. For example, install the client in the **/opt/hadoopclient** directory.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.

- b. Log in to FusionInsight Manager, click **Download Client** in the upper right corner of **Homepage**. Set **Select Client Type** to **Configuration Files Only**, select **Save to Path**, and click **OK** to download the client configuration file to the active OMS node of the cluster.
- c. Log in to the active OMS node as user **root**, go to the directory where the client configuration files are stored (**/tmp/FusionInsight-Client** by default), decompress the software package, and obtain all configuration files in the **IoTDB/config** directory. Place the files in the **conf** directory where the compiled JAR package is stored on the client node, for example, **/opt/hadoopclient/conf**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and it is downloaded to the **/tmp/FusionInsight-Client** directory on the active management node, run the following commands:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp IoTDB/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

- d. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If there is no required information, copy the content of the **hosts** file in the decompression directory to the **hosts** file on the node where the client is deployed, to ensure that the local host can communicate with each host in the cluster.

### 2.10.2.3 Configuring and Importing Sample Projects

#### Context

Obtain the IoTDB development sample project and import the project to IntelliJ IDEA to learn the sample project.

#### Procedure

- Step 1** Obtain the sample project from the **src/iotdb-examples** directory in the directory where the sample code is decompressed by referring to [Obtaining Sample Projects from Huawei Mirrors](#). You can select a sample based on the actual service scenario. For details about the samples, see [IoTDB Sample Project](#).
- Step 2** To debug **iotdb-flink-example**, **iotdb-jdbc-example**, **iotdb-kafka-example**, or **iotdb-session-example** sample code on the local Windows environment, perform the following operations:  
Configure the **com.huawei.bigdata.iotdb.IoTDBProperties** class in the ..\src\main\resources directory of each sample project, change the value of **proPath** in the **IoTDBProperties()** method to the absolute path of the **iotdb-example.properties** file.

Figure 2-174 Configuring the proPath parameter

```
private IoTDBProperties() {
    // If Windows environment, proPath is "iotdb-example.properties" absolute file path.
    // eg: "D:\\sample_project\\sample_project\\src\\iotdb-examples\\iotdb-session-example\\src\\main\\resources\\iotdb-example.properties"
    String proPath = "D:\\sample_project\\sample_project\\src\\iotdb-examples\\iotdb-session-example\\src\\main\\resources\\iotdb-example.properties";
    // String proPath = System.getProperty("user.dir") + File.separator + "src" + File.separator + "main"
    // + File.separator + "resources" + File.separator + "iotdb-example.properties";
    try {
        iotdbProps.load(new FileInputStream(new File(proPath)));
    } catch (IOException e) {
        LOG.info("The Exception occurred.", e);
    }
}
```

- Step 3** Modify the **iotdb-example.properties** file in the **..\src\main\resources** directory of each sample project.

```
jdbc_url=jdbc:iotdb://IP address of the IoTDBServer instance:IoTDBServer RPC port
# Username for authentication
username=developuser
# User password for authentication. It is recommended that the password be stored in ciphertext and decrypted when being used.
password=xxx
```

- Step 4** After the IntelliJ IDEA and JDK tools are installed, configure the JDK in IntelliJ IDEA.

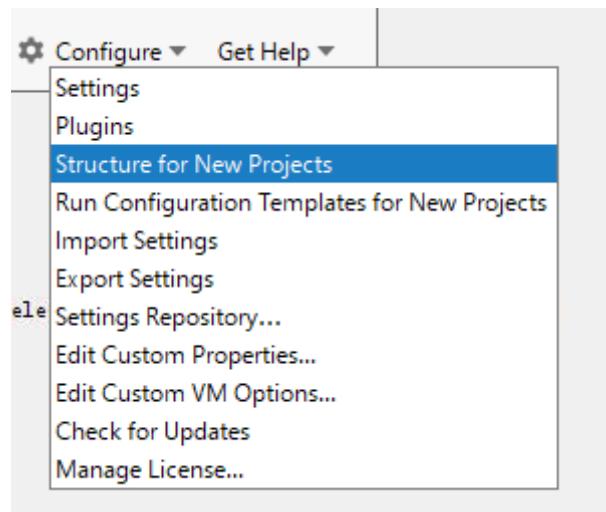
1. Start IntelliJ IDEA and choose **Configure**.

Figure 2-175 Quick Start



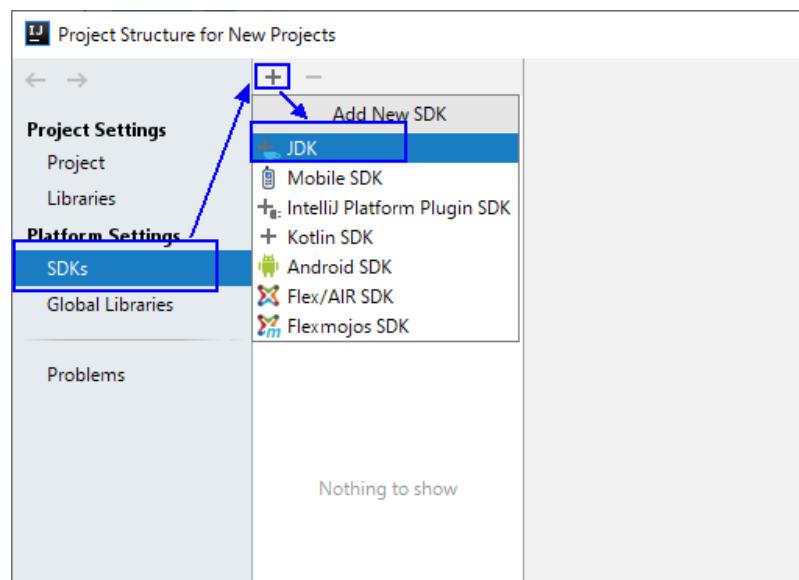
2. Select **Structure for New Projects** from the drop-down list.

Figure 2-176 Configure



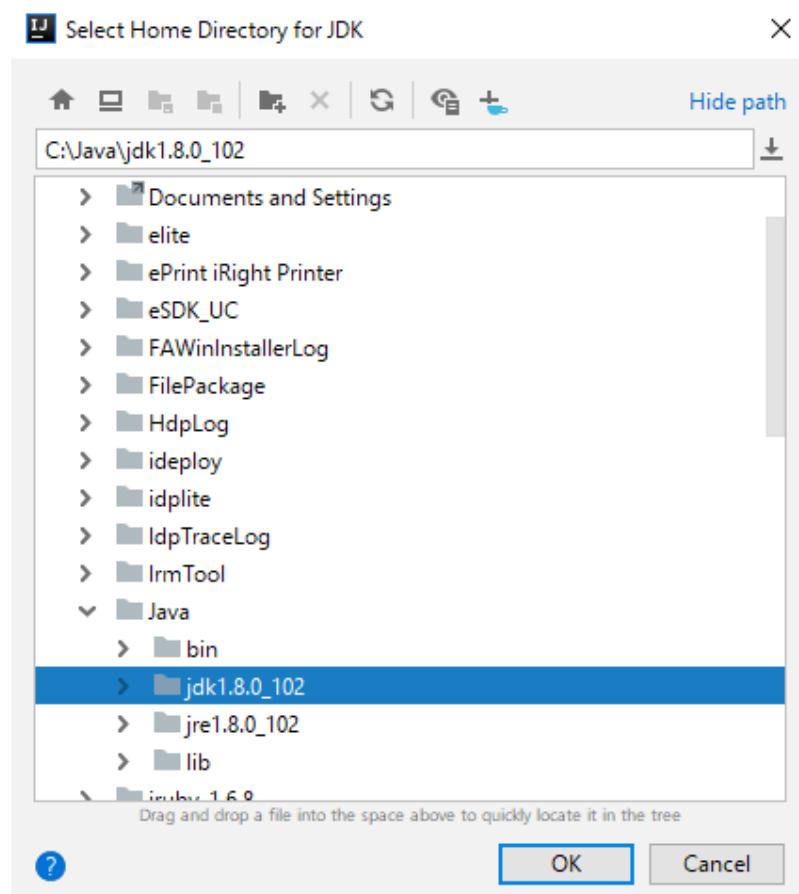
3. On the displayed **Project Structure for New Projects** page, select **SDKs**, click the plus sign (+), and click **JDK**.

Figure 2-177 Project Structure for New Projects



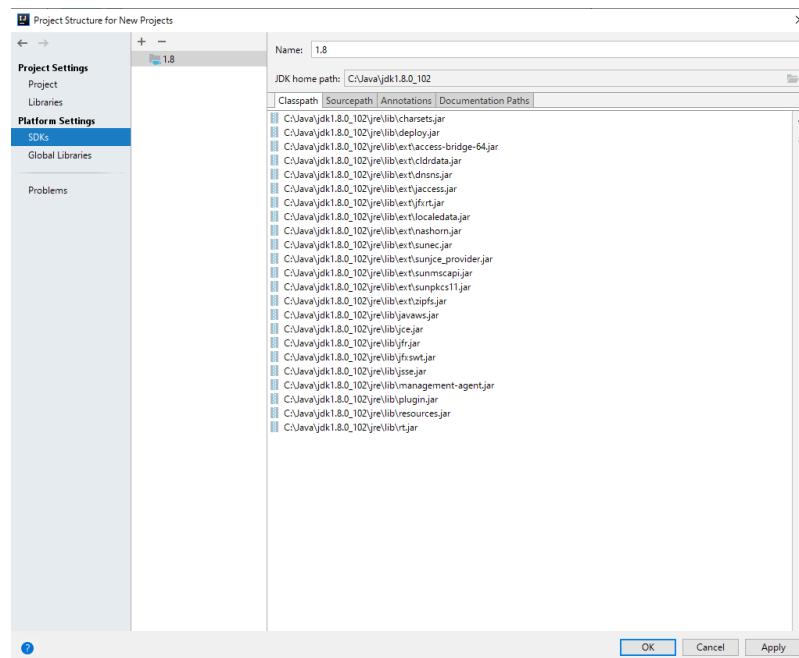
4. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.

Figure 2-178 Select Home Directory for JDK



5. After selecting the JDK, click **OK** to complete the configuration.

Figure 2-179 Completing the JDK configuration



 NOTE

The operation procedure may vary according to the IDEA version.

**Step 5** Import the sample project to the IntelliJ IDEA development environment.

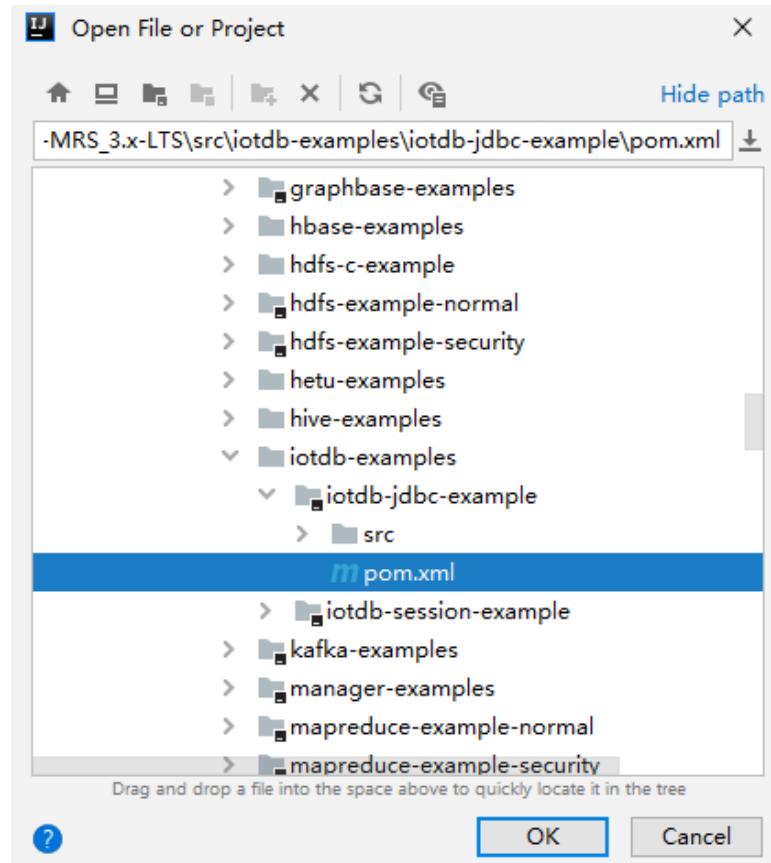
1. Start the IntelliJ IDEA, and click **Open or Import** on the quick start page.

For the IDEA tool that has been used, you can choose **File > Import project...** on the main interface to import the sample project.

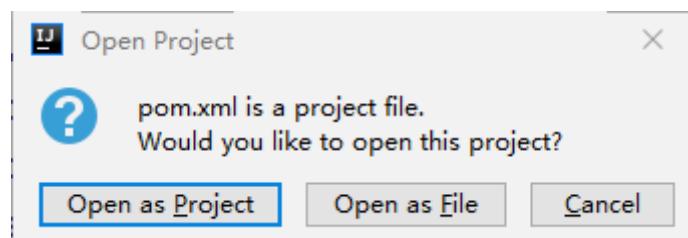
**Figure 2-180** Open or Import (quick start page)



2. Select the **pom.xml** file of **iotdb-jdbc-example** in the sample project folder **iotdb-examples**, and click **OK**.

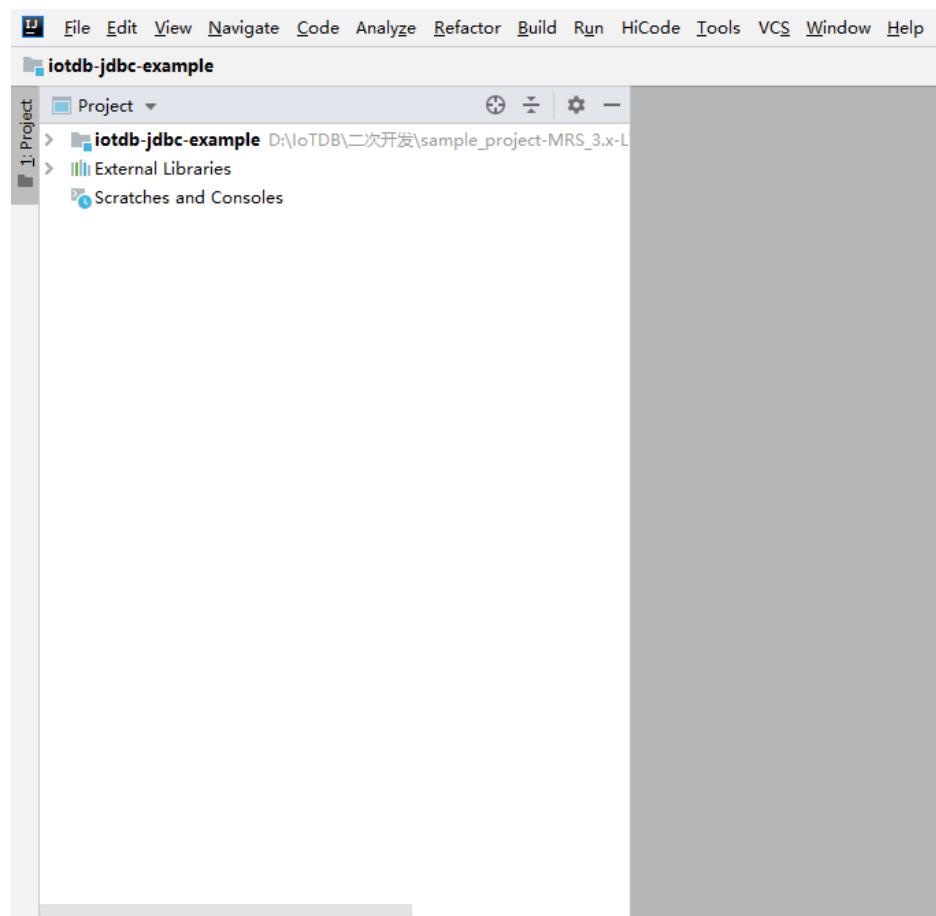


Select Open as Project.



3. After the import is complete, check that the imported sample projects are displayed on the IDEA home page.

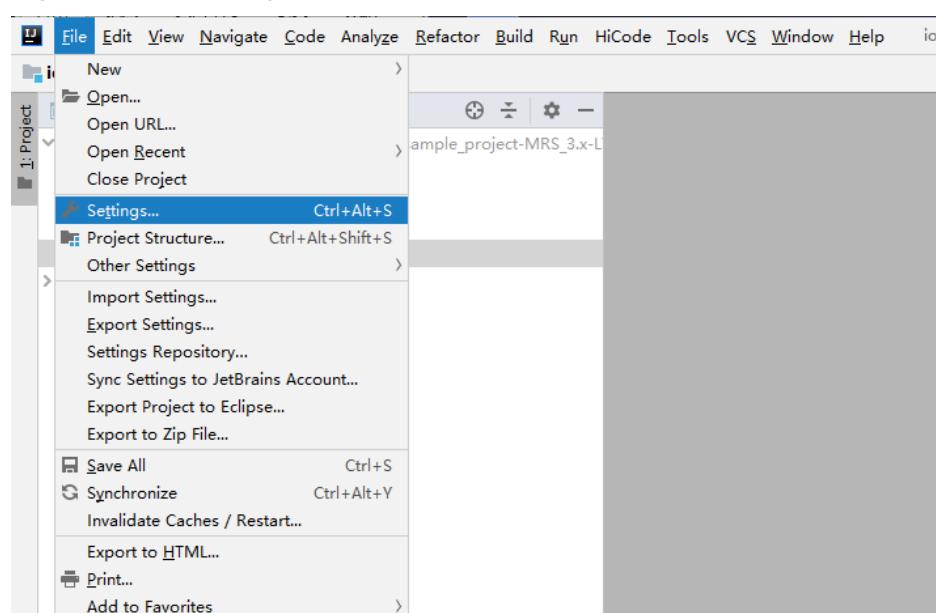
Figure 2-181 Successfully importing sample projects



**Step 6** Set the Maven version used by the project.

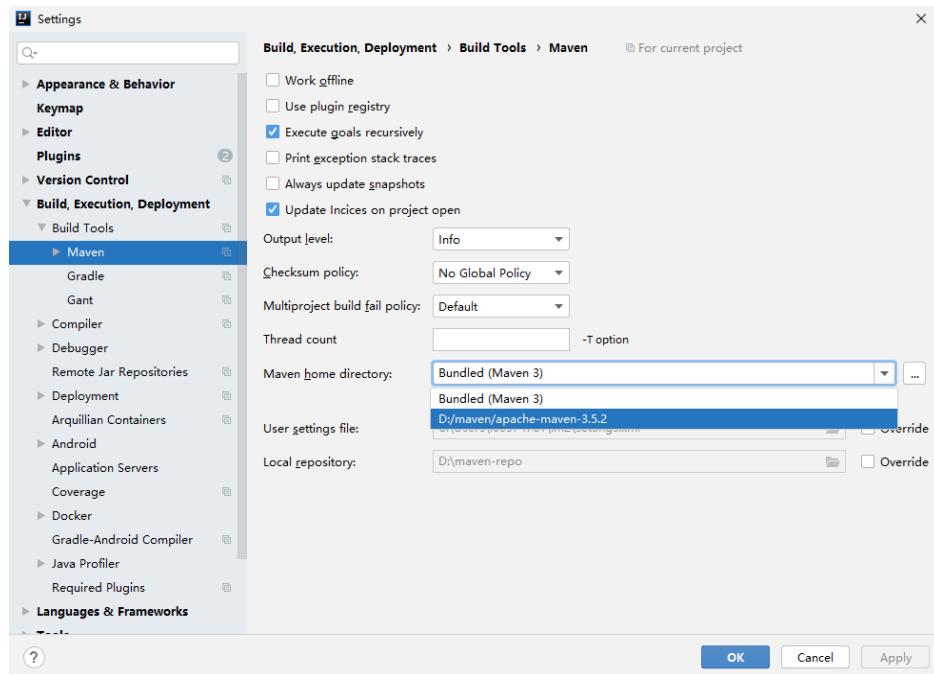
1. Choose **File > Settings...** from the main menu of IntelliJ IDEA.

Figure 2-182 Settings



2. Choose **Build, Execution, Deployment > Maven** and set **Maven home directory** to the Maven version installed on the local host.  
Configure **User settings file** and **Local repository** based on the site requirements.

**Figure 2-183** Selecting the local Maven installation directory



3. Click **Apply** and **OK** to complete the configuration.

----End

## 2.10.3 Application Development

### 2.10.3.1 IoTDB JDBC

#### 2.10.3.1.1 Java Example Code

##### Description

Run the IoTDB SQL statement in JDBC connection mode.

##### Sample Code

The following code snippets are used as an example. For complete codes, see the **com.huawei.bigdata.iotdb.JDBCExample** class.

**jdbc url** includes the IP address, port number, username, and password of the node where the IoTDBServer to be connected is located.

### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instances** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation. To obtain the default password, see [Huawei Cloud Stack 8.3.1 Account List](#) or contact the system administrator. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
private static final String IOTDB_SSL_ENABLE = "false"; //To obtain the value, log in to FusionInsight Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and view the value of SSL_ENABLE.
public static void main(String[] args) throws ClassNotFoundException, SQLException {
    Class.forName("org.apache.iotdb.jdbc.IoTDBDriver");
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
        // set truststore.jks path
        System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    try (Connection connection =
        DriverManager.getConnection ("jdbc:iotdb://127.0.0.1:22260/", "Authentication
username", "Password");
        Statement statement = connection.createStatement()) {
        // set JDBC fetchSize
        statement.setFetchSize(10000);
        for (int i = 0; i <= 100; i++) {
            statement.addBatch(prepareInsertStatement(i));
        }
        statement.executeBatch();
        statement.clearBatch();

        ResultSet resultSet = statement.executeQuery("select ** from root where time <= 10");
        outputResult(resultSet);
        resultSet = statement.executeQuery("select count(**) from root");
        outputResult(resultSet);
        resultSet =
            statement.executeQuery(
                "select count(**) from root where time >= 1 and time <= 100 group by ([0, 100), 20ms, 20ms)");
        outputResult(resultSet);
    } catch (IoTDBSQLException e) {
        System.out.println(e.getMessage());
    }
}
```

## 2.10.3.2 IoTDB Session

### 2.10.3.2.1 Java Example Code

#### Description

Run the IoTDB SQL statement in Session connection mode.

## Sample Code

The following code snippets are used as an example. For complete code, see [com.huawei.bigdata.SessionExample](#).

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instances** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation. To obtain the default password, see [Huawei Cloud Stack 8.3.1 Account List](#) or contact the system administrator. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
private static final String IOTDB_SSL_ENABLE = "true"; //To obtain the value, log in to FusionInsight Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and view the value of SSL_ENABLE.
public static void main(String[] args)
    throws IoTDBConnectionException, StatementExecutionException {
    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
        // set truststore.jks path
        System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    session = new Session(nodeUrls, "Authentication username", "password");
    session.open(false);

    // set session fetchSize
    session.setFetchSize(10000);

    try {
        session.setStorageGroup("root.sg1");
    } catch (StatementExecutionException e) {
        if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {
            throw e;
        }
    }

    createTimeseries();
    createMultiTimeseries();
    insertRecord();
    insertTablet();
    insertTablets();
    insertRecords();
    nonQuery();
    query();
    queryWithTimeout();
    rawDataQuery();
    queryByIterator();
    deleteData();
    deleteTimeseries();
    setTimeout();
```

```
sessionEnableRedirect = new Session(nodeUrls, Authentication.username, Password);
sessionEnableRedirect.setEnableQueryRedirection(true);
sessionEnableRedirect.open(false);

// set session fetchSize
sessionEnableRedirect.setFetchSize(10000);

insertRecord4Redirect();
query4Redirect();
sessionEnableRedirect.close();
session.close();
}
```

## 2.10.3.3 IoTDB Flink

### 2.10.3.3.1 FlinkIoTDBSink

#### Description

It is an integration of IoTDB and Flink. This module contains IoTDBSink and writes time series data into IoTDB using a Flink job.

#### Sample Code

This example shows how to send data from a Flink job to an IoTDB server.

- A simulated source SensorSource generates a data point every second.
- Flink uses IoTDBSink to consume produced data and write the data into IoTDB.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

#### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instances** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation. To obtain the default password, see [Huawei Cloud Stack 8.3.1 Account List](#) or contact the system administrator. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication.username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
public class FlinkIoTDBSink {
    public static void main(String[] args) throws Exception {
        // run the flink job on local mini cluster
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        IoTDBSinkOptions options = new IoTDBSinkOptions();
        options.setHost("127.0.0.1");
        options.setPort(22260);
        options.setUser ("Authentication.username");
```

```
options.setPassword ("Password");

// If the server enables auto_create_schema, then we do not need to register all timeseries
// here.
options.setTimeseriesOptionList(
    Lists.newArrayList(
        new IoTDBSinkOptions.TimeseriesOption(
            "root.sg.d1.s1", TSDDataType.DOUBLE, TSEncoding.GORILLA, CompressionType.SNAPPY)));

IoTSerializationSchema serializationSchema = new DefaultIoTSerializationSchema();
IoTDBSink ioTDBSink =
    new IoTDBSink(options, serializationSchema)
        // enable batching
        .withBatchSize(10)
        // how many connections to the server will be created for each parallelism
        .withSessionPoolSize(3);

env.addSource(new SensorSource()
    .name("sensor-source")
    .setParallelism(1)
    .addSink(ioTDBSink)
    .name("iotdb-sink"));

env.execute("iotdb-flink-example");
}

private static class SensorSource implements SourceFunction<Map<String, String>> {
    boolean running = true;
    Random random = new SecureRandom();

    @Override
    public void run(SourceContext context) throws Exception {
        while (running) {
            Map<String, String> tuple = new HashMap();
            tuple.put("device", "root.sg.d1");
            tuple.put("timestamp", String.valueOf(System.currentTimeMillis()));
            tuple.put("measurements", "s1");
            tuple.put("types", "DOUBLE");
            tuple.put("values", String.valueOf(random.nextDouble()));

            context.collect(tuple);
            Thread.sleep(1000);
        }
    }

    @Override
    public void cancel() {
        running = false;
    }
}
```

### 2.10.3.3.2 FlinkIoTDBSource

#### Description

It is an integration of IoTDB and Flink. This module contains IoTDBSource and reads time series data from IoTDB and prints the data using a Flink job.

#### Sample Code

This example shows how a Flink job reads time series data from a IoTDB server.

- Flink uses IoTDBSource to read data from a IoTDB server.
- To use IoTDBSource, you need to construct an IoTDBSource instance by specifying **IoTDBSourceOptions** and implementing the abstract method

**convert()** in IoTDBSource. The **convert()** method defines how you want to convert row data.

Session object parameters include the IP address, port number, username, and password of the node where the IoTDBServer is located.

#### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB > Instances** to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB**, click **Configurations** then **All Configurations**, and search for **IOTDB\_SERVER\_RPC\_PORT**.
- In normal mode, IoTDB has a default user **root** after initial installation. To obtain the default password, see [Huawei Cloud Stack 8.3.1 Account List](#) or contact the system administrator. This user is an administrator and has all permissions, which cannot be assigned, revoked, or deleted.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - *Authentication username* is the username for accessing IoTDB.
  - *Password* is the password for accessing IoTDB.

```
public class FlinkIoTDBSource {  
    private static final String IOTDB_SSL_ENABLE = "true"; // To obtain the value, log in to FusionInsight  
    Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and  
    view the value of SSL_ENABLE.  
    static final String LOCAL_HOST = "127.0.0.1";  
    static final String ROOT_SG1_D1_S1 = "root.sg1.d1.s1";  
    static final String ROOT_SG1_D1 = "root.sg1.d1";  
  
    public static void main(String[] args) throws Exception {  
        // use session api to create data in IoTDB  
        prepareData();  
  
        // run the flink job on local mini cluster  
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
  
        IoTDBSourceOptions ioTDBSourceOptions =  
            new IoTDBSourceOptions()  
                .LOCAL_HOST(22260, "Authentication username", "Password", "select s1 from "+ ROOT_SG1_D1 +"  
align by device");  
  
        IoTDBSource<RowRecord> source =  
            new IoTDBSource<RowRecord>(ioTDBSourceOptions) {  
                @Override  
                public RowRecord convert(RowRecord rowRecord) {  
                    return rowRecord;  
                }  
            };  
        env.addSource(source).name("sensor-source").print().setParallelism(2);  
        env.execute();  
    }  
  
    private static void prepareData()  
        throws IoTDBConnectionException, StatementExecutionException, TTransportException {  
        // set iotdb_ssl_enable  
        System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);  
        if ("true".equals(IOTDB_SSL_ENABLE)) {  
            // set truststore.jks path  
            System.setProperty("iotdb_ssl_truststore", "truststore file path");  
        }  
        Session session = new Session(LOCAL_HOST, 22260, "Authentication username", "Password");  
        session.open(false);  
        try {
```

```
session.setStorageGroup("root.sg1");
if (!session.checkTimeseriesExists(ROOT_SG1_D1_S1)) {
    session.createTimeseries(
        ROOT_SG1_D1_S1, TSDataType.INT64, TSEncoding.RLE, CompressionType.SNAPPY);
    List<String> measurements = new ArrayList<>();
    measurements.add("s1");
    measurements.add("s2");
    measurements.add("s3");
    List<TSDataType> types = new ArrayList<>();
    types.add(TSDataType.INT64);
    types.add(TSDataType.INT64);
    types.add(TSDataType.INT64);
    types.add(TSDataType.INT64);

    for (long time = 0; time < 1000; time++) {
        List<Object> values = new ArrayList<>();
        values.add(1L);
        values.add(2L);
        values.add(3L);
        session.insertRecord(ROOT_SG1_D1, time, measurements, types, values);
    }
}
} catch (StatementExecutionException e) {
    if (e.getStatusCode() != TSStatusCode.PATH_ALREADY_EXIST_ERROR.getStatusCode()) {
        throw e;
    }
}
}
```

## 2.10.3.4 IoTDB Kafka

### 2.10.3.4.1 Java Sample Code

#### Description

This example shows how to send data to IoTDB using Kafka.

#### Sample Code

- Producer.java:

This example shows how to send time series data to a Kafka cluster.

- Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.
- The default time series data format of this sample is *Device name, Timestamp, Value*, for example, **sensor\_1,1642215835758,1.0**. You can change the value of **IOTDB\_DATA\_SAMPLE\_TEMPLATE** in the **Constant.java** file based on the site requirements.

```
public static void main(String[] args) {
    // whether use security mode
    final boolean isSecurityMode = false;
    ...
    // whether to use the asynchronous sending mode
    final boolean asyncEnable = false;
    Producer producerThread = new Producer(KafkaProperties.TOPIC, asyncEnable);
}
```



For details about the Kafka producer code, see [Producer API Usage Sample](#).

- KafkaConsumerMultThread.java:

This example shows how to write data from a Kafka cluster to IoTDB using multiple threads. Kafka cluster data is generated by **Producer.java**.

- Change the value of the **public final static String TOPIC** variable in the **KafkaProperties.java** file based on the site requirements, for example, **kafka-topic**.
- Change the value of **CONCURRENCY\_THREAD\_NUM** in the **KafkaConsumerMultThread.java** file to adjust the number of consumer threads.

#### NOTICE

- If multiple threads are used to consume Kafka cluster data, ensure that the number of consumed topic partitions is greater than 1.
  - The Kafka client configuration files **client.properties** and **log4j.properties** must be stored in the configuration file directory of the program.
- c. Set the IP address, port number, username, and password of the node where the IoTDBServer is located in the IoTDBSessionPool object parameters.

#### NOTE

- On FusionInsight Manager, choose **Cluster > Services > IoTDB** and click the **Instances** tab to view the IP address of the node where the IoTDBServer to be connected is located.
- The default RPC port is **22260**. To obtain the port number, choose **Cluster > Services > IoTDB** and click the **Configurations** tab and then **All Configurations**. On the displayed page, search for **rpc\_port**.
- You need to set the username and password for authentication in the local environment variables. You are advised to store the username and password in ciphertext and decrypt them upon using.
  - Authentication username* is the username for accessing IoTDB.
  - Password* is the password for accessing IoTDB.

```
private static final String IOTDB_SSL_ENABLE = "true"; // To obtain the value, log in to FusionInsight Manager, choose Cluster > Services > IoTDB, click Configurations, search for SSL in the search box, and view the value of SSL_ENABLE.
public static void main(String[] args) {
    // whether use security mode
    final boolean isSecurityModel = false;
    ...

    // set iotdb_ssl_enable
    System.setProperty("iotdb_ssl_enable", IOTDB_SSL_ENABLE);
    if ("true".equals(IOTDB_SSL_ENABLE)) {
        // set truststore.jks path
        System.setProperty("iotdb_ssl_truststore", "truststore file path");
    }
    // create IoTDB session connection pool
    IoTDBSessionPool sessionPool = new IoTDBSessionPool("127.0.0.1", 22260, "Authentication username", "Password", 3);

    // start consumer thread
    KafkaConsumerMultThread consumerThread = new KafkaConsumerMultThread(KafkaProperties.TOPIC, sessionPool);
```

```
        consumerThread.start();
    }

    /**
     * consumer thread
     */
    private class ConsumerThread extends ShutdownableThread {
        private int threadNum;
        private String topic;
        private KafkaConsumer<String, String> consumer;
        private IoTDBSessionPool sessionPool;

        public ConsumerThread(int threadNum, String topic, Properties props, IoTDBSessionPool sessionPool) {
            super("ConsumerThread" + threadNum, true);
            this.threadNum = threadNum;
            this.topic = topic;
            this.sessionPool = sessionPool;
            this.consumer = new KafkaConsumer<>(props);
            consumer.subscribe(Collections.singleton(this.topic));
        }

        public void doWork() {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(waitTime));
            for (ConsumerRecord<String, String> record : records) {
                LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + " record: " +
                        + record.value() + " offsets: " + record.offset());
                // insert kafka consumer data to iotdb
                sessionPool.insertRecord(record.value());
            }
        }
    }
}
```

#### NOTE

For details about the Kafka consumer code, see [Multi-thread Producer Sample](#).

### 2.10.3.5 IoTDB UDF Program

#### 2.10.3.5.1 IoTDB UDF Sample Code

##### Description

This section describes how to implement a simple IoTDB user-defined function (UDF). For details, see "UDFs" in the *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

##### Sample Code

```
package com.huawei.bigdata.iotdb;
import org.apache.iotdb.udf.api.UDTF;
import org.apache.iotdb.udf.api.access.Row;
import org.apache.iotdb.udf.api.collector.PointCollector;
import org.apache.iotdb.udf.api.customizer.config.UDTFConfigurations;
import org.apache.iotdb.udf.api.customizer.parameter.UDFParameters;
import org.apache.iotdb.udf.api.customizer.strategy.RowByRowAccessStrategy;
import org.apache.iotdb.udf.api.type.Type;
import java.io.IOException;

public class UDTFExample implements UDTF {
    @Override
    public void beforeStart(UDFParameters parameters, UDTFConfigurations configurations) {
```

```
        configurations.setAccessStrategy(new RowByRowAccessStrategy()).setOutputDataType(Type.INT32);  
    }  
  
    @Override  
    public void transform(Row row, PointCollector collector) throws IOException {  
        collector.putInt(row.getTime(), -row.getInt(0));  
    }  
}
```

## 2.10.4 Application Commissioning

### 2.10.4.1 Commissioning Applications on Windows

#### 2.10.4.1.1 Compiling and Running Applications

##### Scenario

Run applications in a Windows development environment after application code is developed. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

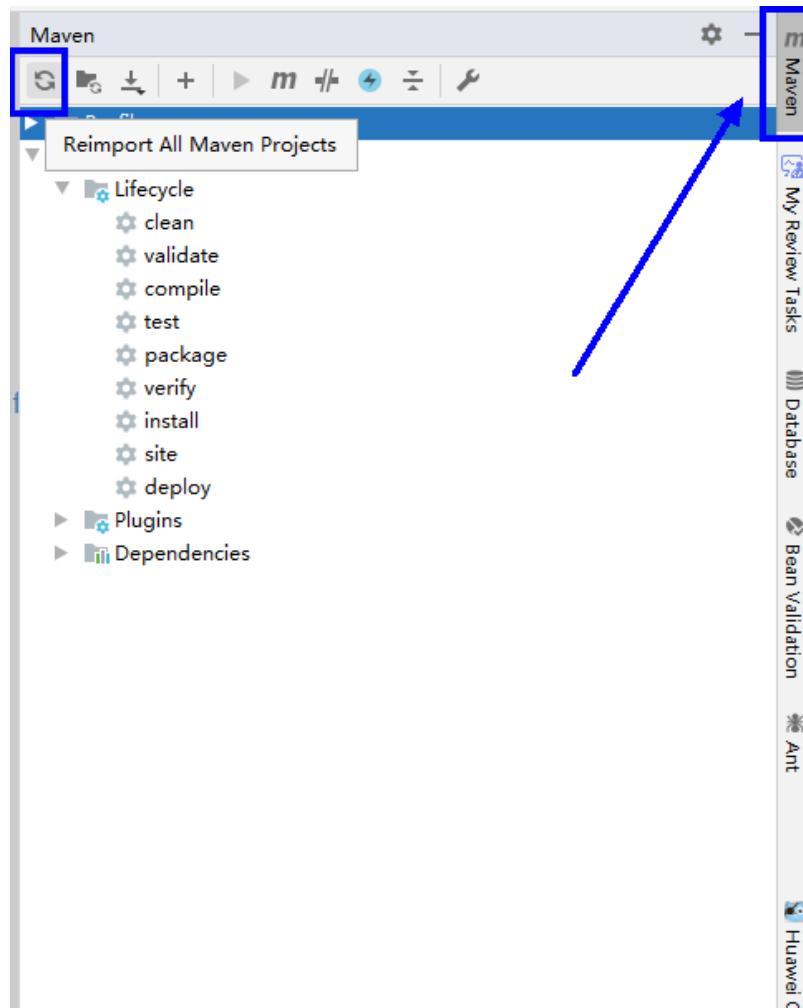
##### NOTE

- If the IBM JDK is used in the Windows environment, applications cannot be directly run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the **hosts** file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

##### Procedure

- Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.

Figure 2-184 reimport projects



**Step 2** Compile and run the application.

Modify the IP address, port number, login username, and password of the IoTDBServer node that matches the code.

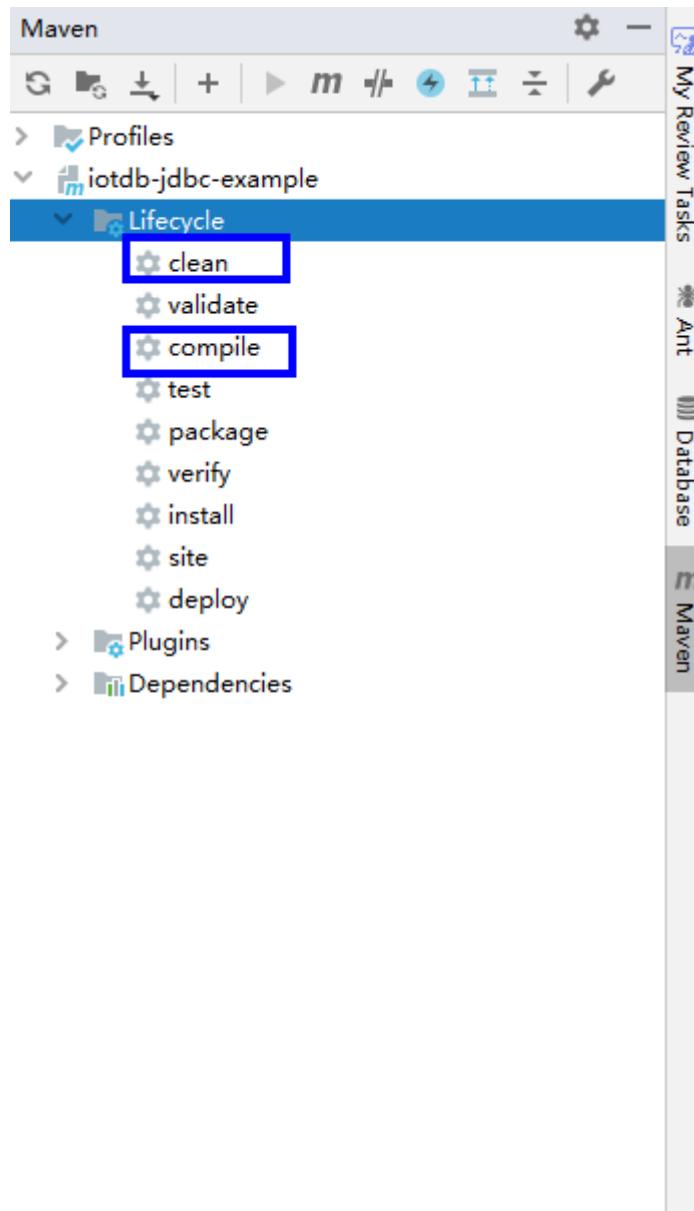
1. Use either of the following two methods:

- Method 1

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile** and double-click **compile** to run the **compile** command of Maven.

Figure 2-185 Maven tools clean and compile



- Method 2

Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 2-186 Enter **mvn clean compile** in the IDEA **Terminal** text box.

The screenshot shows the IntelliJ IDEA terminal window. The command `mvn clean compile` is typed into the terminal. The terminal output shows the command was run successfully.

```
Terminal: Local +  
Microsoft Windows [Version 10.0.18362.592]  
(c) 2019 Microsoft Corporation. All rights reserved.  
D:\IoTDB\develop\sample_project-MRS_3.x-LTS\src\iotdb-examples\iotdb-jdbc-example>mvn clean compile
```

After the compilation is complete, the message "BUILD SUCCESS" is displayed.

**Figure 2-187** Compilation completed



The screenshot shows a terminal window with the following text output:

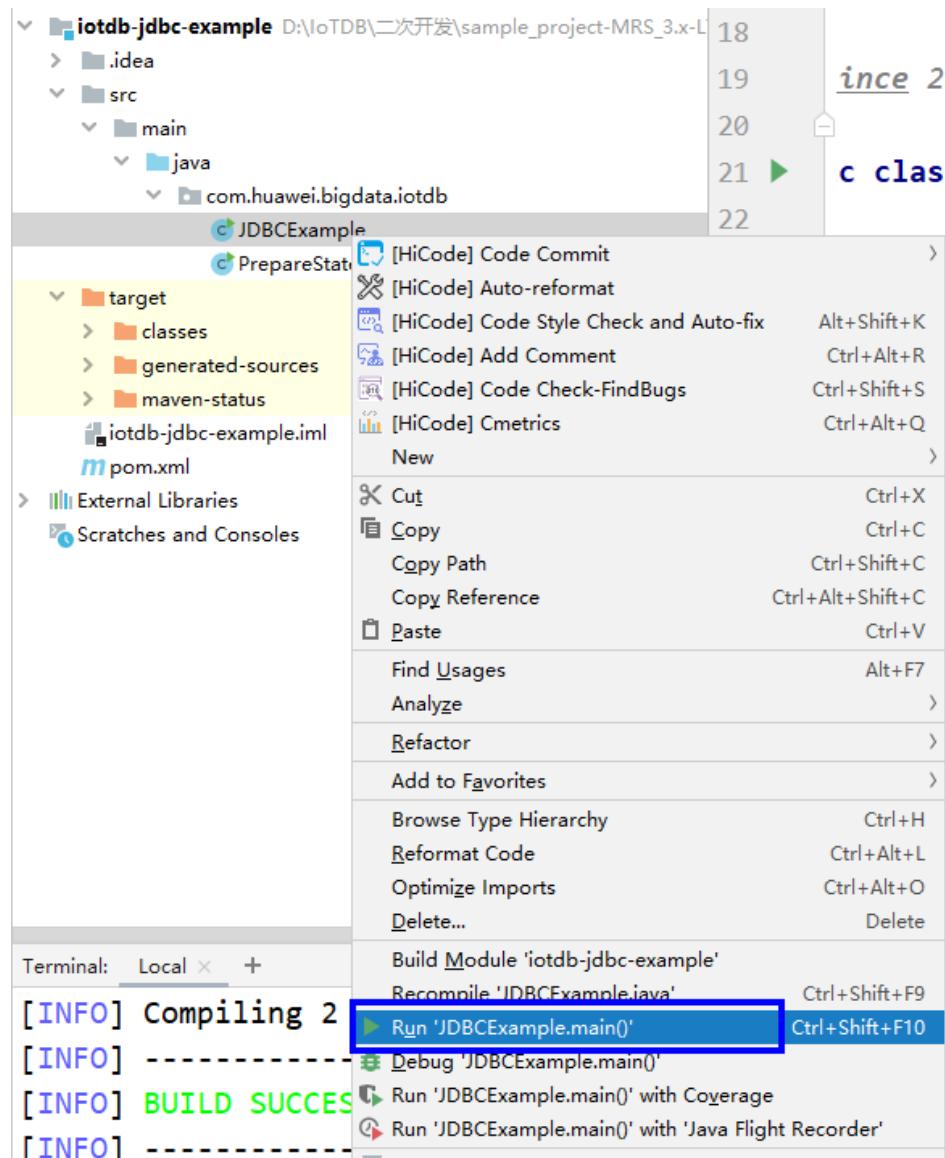
```
[INFO] Compiling 2 source files to D:\IoTDB\develop\sample_project-MRS_3.x-LTS\src\io  
[INFO]  
[INFO] BUILD SUCCESS  
[INFO]  
[INFO] Total time: 1.156 s  
[INFO] Finished at: 2021-06-16T14:29:36+08:00  
[INFO]
```

The terminal window has tabs at the bottom: Cmetrics, CodeCheck-FindBugs, CodeReview, CodeStyleCheckAndFix, Terminal, Build, Messages, Run, and TODO. The Terminal tab is selected.

2. Run the application.

Right-click the **JDBCExample.java** file and choose **Run 'JDBCExample.main()'** from the shortcut menu.

Figure 2-188 Running the application



----End

#### 2.10.4.1.2 Viewing Commissioning Results

After the IoTDB application is run, you can view the running results in IntelliJ IDEA.

#### **2.10.4.2 Commissioning JDBC and Session Applications on Linux**

#### **2.10.4.2.1 Compiling and Running Applications**

## Scenario

IoTDB applications can run in a Linux environment where an IoTDB client is installed. After the application code is developed, you can upload the JAR file to the prepared Linux environment. A Session application is used as an example. Operations for JDBC applications are the same as those for Session applications.

## Prerequisites

- You have installed the IoTDB client.
  - If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

## Procedure

## **Step 1 Export a JAR file.**

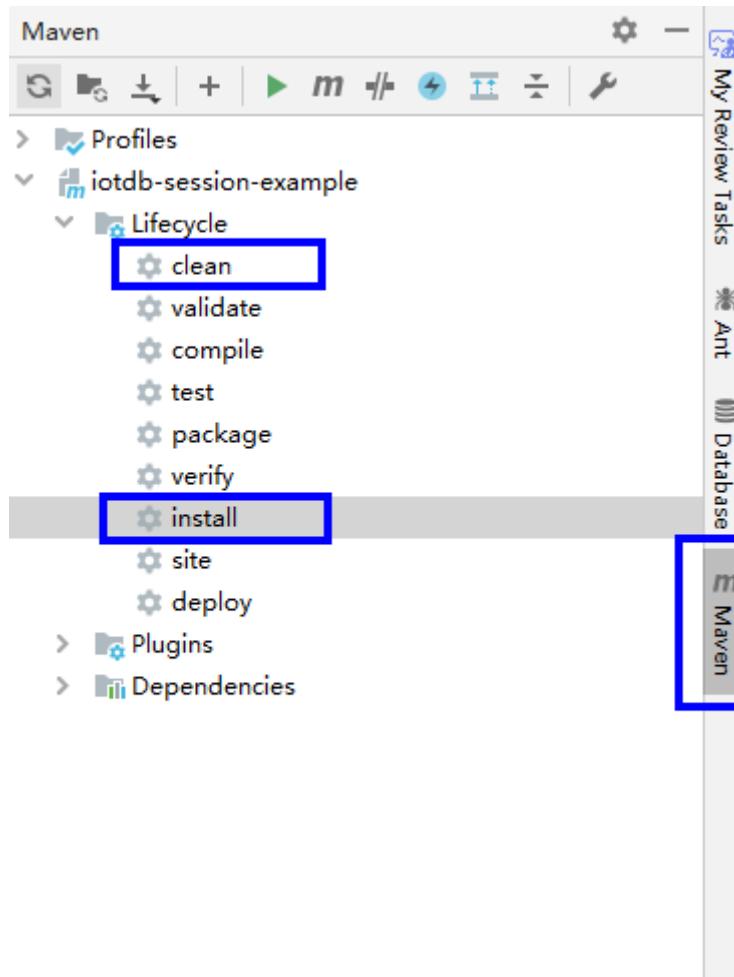
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

Figure 2-189 Maven tools clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the file.

Figure 2-190 Entering mvn clean install in the IDEA Terminal text box

```
Terminal: Local +  
1006\iotdb-session-example-0.12.0-hw-ei-311006.pom  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.401 s  
[INFO] Finished at: 2021-07-31T11:22:12+08:00  
[INFO] -----  
D:\IoTDB\二次开发\v9\sample_project\src\iotdb-examples\iotdb-session-example mvn clean install
```

After the compilation is completed, the message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

## Step 2 Prepare the dependent JAR file.

- Log in to the IoTDB client and import the JAR file generated in **Step 1** to the **lib** directory of the IoTDB client, for example, **/opt/hadoopclient/IoTDB/iotdb/lib**.

2. In the root directory of the IoTDB client, for example, **/opt/hadoopclient/IoTDB/iotdb**, create the **run.sh** script, modify the content as follows, and save the modification.

```
#!/bin/sh  
BASEDIR=`cd $(dirname $0);pwd`  
cd ${BASEDIR}  
for file in ${BASEDIR}/lib/*.jar  
do  
i_cp=$i_cp:$file  
echo "$file"  
done  
for file in ${BASEDIR}/conf/*  
do  
i_cp=$i_cp:$file  
done
```

```
java -cp .${i_cp} com.huawei.bigdata.iotdb.JDBCExample
```

*com.huawei.bigdata.iotdb.JDBCExample* is an example. Use the actual code instead.

3. Run the **run.sh** script to run the JAR file.

```
sh /opt/hadoopclient/IoTDB/iotdb/run.sh
```

----End

## 2.10.4.2.2 Viewing Commissioning Results

If the application running is successful, the following information is displayed.

```
11:53:47.586 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect2.d1 where time >= 1 and time < 10 and root.redirect2.d1.s1 > 1  
[Time, root.redirect2.d1.s3, root.redirect2.d1.s1, root.redirect2.d1.s2]  
1 2 5 3 4  
2 5 3 4  
3 6 4 5  
4 7 5 6  
11:53:47.590 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect3.d1 where time >= 1 and time < 10 and root.redirect3.d1.s1 > 1  
[Time, root.redirect3.d1.s3, root.redirect3.d1.s1, root.redirect3.d1.s2]  
1 2 5 3 4  
2 5 3 4  
3 6 4 5  
4 7 5 6  
11:53:47.596 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect4.d1 where time >= 1 and time < 10 and root.redirect4.d1.s1 > 1  
[Time, root.redirect4.d1.s3, root.redirect4.d1.s1, root.redirect4.d1.s2]  
1 2 5 3 4  
2 5 3 4  
3 6 4 5  
4 7 5 6  
11:53:47.601 [main] INFO org.apache.iotdb.session.Session - EndPoint(ip:8.5.248.2, port:22260) execute sql select * from root.redirect5.d1 where time >= 1 and time < 10 and root.redirect5.d1.s1 > 1  
[Time, root.redirect5.d1.s3, root.redirect5.d1.s1, root.redirect5.d1.s2]  
1 2 5 3 4  
2 5 3 4  
3 6 4 5  
4 7 5 6  
[root@8-5-248-2 iotdb]#
```

## 2.10.4.3 Commissioning Flink Applications on Flink Web UI and Linux

### 2.10.4.3.1 Compiling and Running Applications

#### Scenario

IoTDB applications can run in a Linux environment where the Flink client is installed and in an environment where the Flink web UI is installed. After the application code is developed, you can upload the JAR file to the prepared environment.

#### Prerequisites

- The Flink component has been installed in the cluster and the FlinkServer instance has been added.
- The cluster client that contains the Flink service has been installed, for example, in the **/opt/hadoopclient** directory.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts**

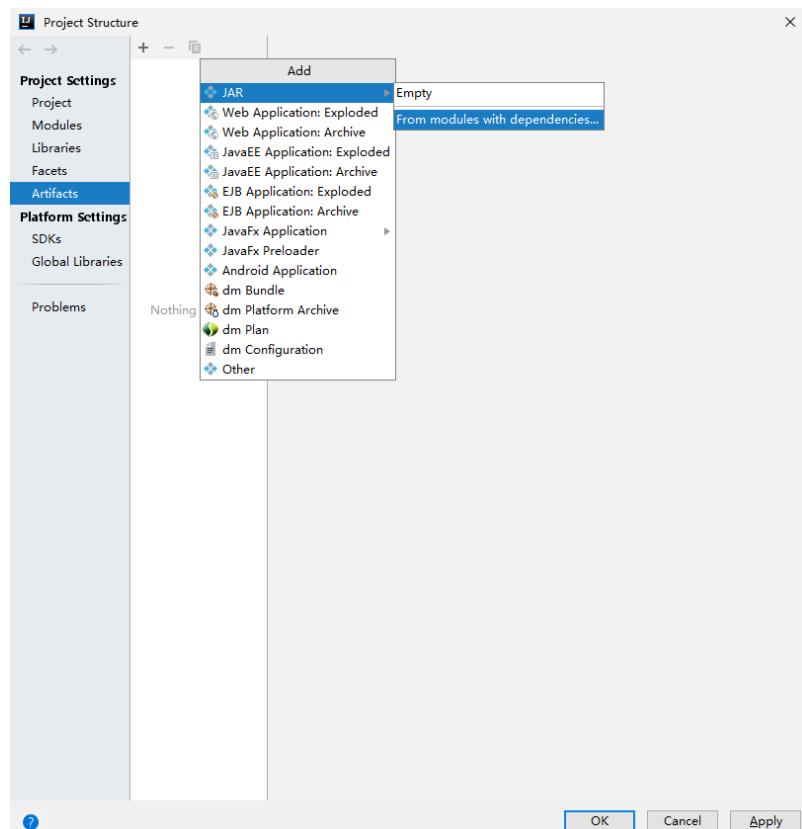
file on the node where the client is located. The host names and IP addresses must be mapped one by one.

## Procedure

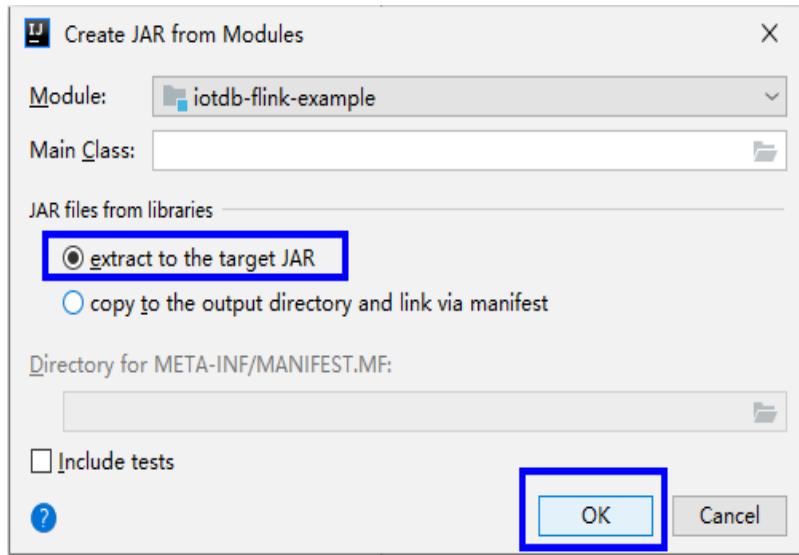
### Step 1 Build a JAR file.

- In IntelliJ IDEA, configure **Artifacts** of the project before generating a JAR file.
  - a. On the IDEA homepage, choose **File > Project Structures...** to go to the **Project Structure** page.
  - b. On the **Project Structure** page, select **Artifacts**, click **+**, and select **From modules with dependencies....**

**Figure 2-191 Adding Artifacts**



- c. Select **extract to the target JAR** and click **OK**.



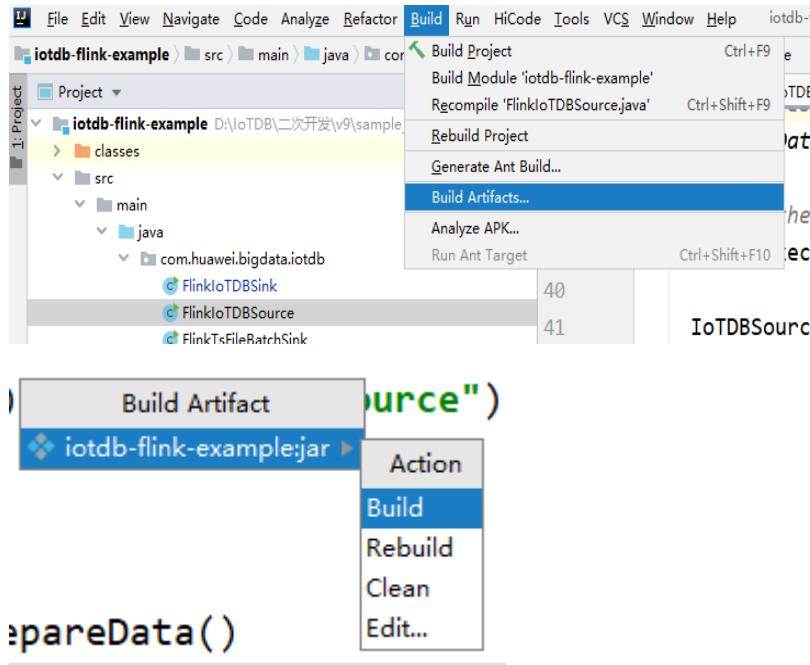
- d. Set the name, type, and output path of the JAR file based on the site requirements.

To avoid JAR file conflicts caused by unnecessary JAR files, you only need to load the following basic JAR files related to IoTDB:

- flink-iotdb-connector-\*
- flink-tsfile-connector-\*
- hdoop-tsfile-\*
- influxdb-thrift-\*
- iotdb-antlr-
- iotdb-session-\*
- iotdb-thrift-\*
- iotdb-thrift-commons-\*
- isession-\*
- libthrift-\*
- iotdb-session-\*
- iotdb-thrift-\*
- service-rpc-\*
- tsfile-\*

Click **OK**.

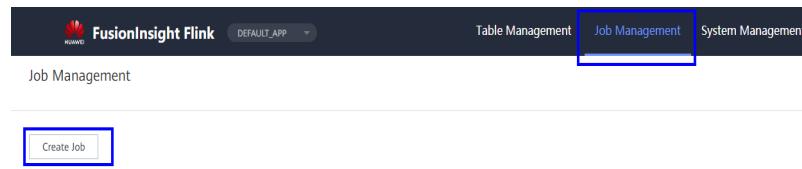
- e. On the IDEA home page, choose **Build > Build Artifacts...**. On the **Build Artifact** page that is displayed, choose **Action > Build**.



- f. After the build is successful, the message "Build completed successfully" is displayed in the lower right corner, and the corresponding JAR file is generated in the **Output Directory** directory.

**Step 2** (Scenario 1) Run a Flink job on the Flink web UI.

1. Log in to FusionInsight Manager of the cluster as a user who has the FlinkServer web UI management permission, choose **Cluster > Services > Flink**, and click the hyperlink next to **Flink WebUI** on the dashboard page to go to the FlinkServer web UI.
2. On the FusionInsight Flink web UI, choose **Job Management > Create Job** to create a job.



3. Set **Type** to **Flink Jar**, enter the name of the job to be created, select a task type, and click **OK**.

Create Job

\*Type Flink SQL Flink Jar

\*Name iotdb\_flink

\*Task Type Stream job Batch Job

Description Enter the description.

OK Cancel

4. Upload the JAR file generated in **Step 1**, set **Main Class** to **Specify**, enter the class to be executed in **Class Parameter**, and click **Submit**.

For example, set **Type** to **com.huawei.bigdata.iotdb.FlinkIoTDBSink** (development program that executes **FlinkIoTDBSink**) or **com.huawei.bigdata.iotdb.FlinkIoTDBSource** (development program that executes **FlinkIoTDBSource**).

Job Management > iotdb\_flink > Back

Upload .jar File

Local jar File Select flink-example.jar

Main Class Specify Default com.huawei.bigdata.iotdb.FlinkIoTDB

Configure Parameters

Parallelism EnterParallelism

JobManager Memory(MB) Enter the memory

Save Submit

### Step 3 (Scenario 2) Submit a Flink job on the Flink client in the Linux environment.

- Save the JAR file generated in **Step 1** to the Flink running environment (Flink client) in the Linux environment, for example, **/opt/hadoopclient**.
- Start the Flink cluster before running the Flink applications on Linux. Run the **yarn session** command on the Flink client to start the Flink cluster. The following is a command example:

```
bin/yarn-session.sh -jm 1024 -tm 1024
```

- Run the **flink-example.jar** sample application.

Open another window on the terminal. Go to the Flink client directory and invoke the **bin/flink run** script to run code.

```
bin/flink flink run --class com.huawei.bigdata.iotdb.FlinkIoTDBSink /opt/hadoopclient/Flink/flink/flink-example.jar
```

**com.huawei.bigdata.iotdb.FlinkIoTDBSink** is the application in **FlinkIoTDBSink**.

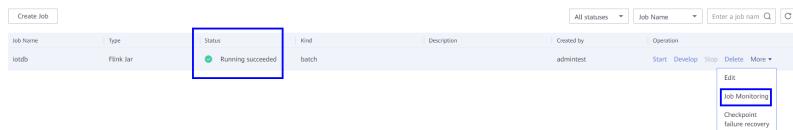
----End

#### 2.10.4.3.2 Viewing Commissioning Results

- Check whether the job is executed.

- Using the Flink web UI

If a success message is returned on the Flink web UI, the execution is successful. You can choose **More > Job Monitoring** in the **Operation** column to view detailed logs.



- Using a Flink Client

Log in to FusionInsight Manager as a running user, go to the native page of the Yarn service, find the application of the corresponding job, and click the application name to go to the job details page.

- If the job is not complete, click **Tracking URL** to go to the native Flink service page and view the job running information.
- If the job submitted in a session has been completed, you can click **Tracking URL** to log in to the native Flink service page to view job information.

- Verify the job execution result.

- FlinkIoTDBSink execution result:

Run the following command on the IoTDB client and check whether the data has been written from Flink to IoTDB:

**SQLselect \* from root.sg.d1**



- FlinkIoTDBSource execution result:
    - i. Log in to FusionInsight Manager as a running user and choose **Cluster > Services > HDFS**. Click the hyperlink next to **NameNode WebUI** to access the HDFS web UI.
    - ii. Choose **Utilities > Browse the file system**.

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities + Logout

Browse the file system

Logs Log Level Metrics Configuration Process Thread Dump Network Topology

## Startup Progress

Elapsed Time: 2 sec, Percent Complete: 100%

Phase	Completion	Elapsed Time
Loading fsimage /usr/libexec/hadoop/current/fsimage_00000000000000000000 399 B	100%	1 sec
erasure coding policies (0/0)	100%	
inodes (1/1)	100%	
delegation tokens (0/0)	100%	
cache pools (0/0)	100%	
Loading edits	100%	0 sec
Saving checkpoint	100%	0 sec
Safe mode	100%	0 sec

- iii. Go to the `/tmp/logs/Execution user name/bucket-logs-tfile/Task ID/Flink task ID` directory and download all files in the directory to the local PC.

## Browse Directory

Execution User Name		Flink JAR task ID							
/tmp/logs/bucket-logs-tfile/0020/application_1663988224590_0020				Go!					
Show	25							Search:	
□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	⋮
□	-rw-r-----	poc	hadoop	408.52 KB	Sep 24 18:15	3	128 MB	192-168-43-111_26009	
□	-rw-r-----	poc	hadoop	313.45 KB	Sep 24 18:15	3	128 MB	192-168-43-116_26009	
□	-rw-r-----	poc	hadoop	318.76 KB	Sep 24 18:15	3	128 MB	192-168-43-77_26009	

- iv. Search for the **root.sg.d1** file from the files downloaded in 2.iii. If the following information is displayed in the file, data is read from IoTDB.

## 2.10.4.4 Commissioning Kafka Applications on Linux

### 2.10.4.4.1 Compiling and Running Applications

#### Scenario

After code development is complete, you can run a IoTDB-Kafka sample application in the Linux environment.

#### Prerequisite

- You have installed the IoTDB and Kafka clients.
- If the host where the client is installed is not a node in the cluster, the mapping between the host name and the IP address must be set in the **hosts** file on the node where the client is located. The host names and IP addresses must be mapped one by one.

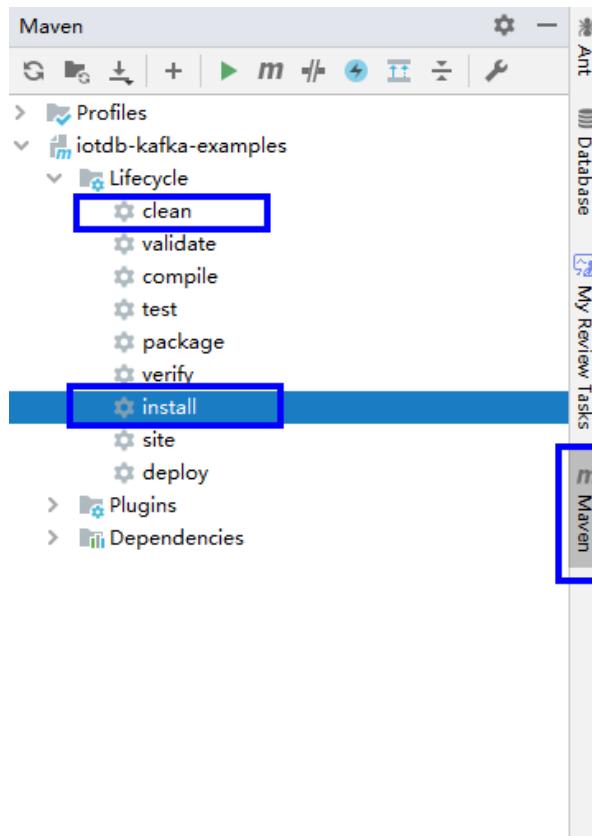
#### Procedure

##### Step 1 Export a JAR file.

You can build a JAR file in either of the following ways:

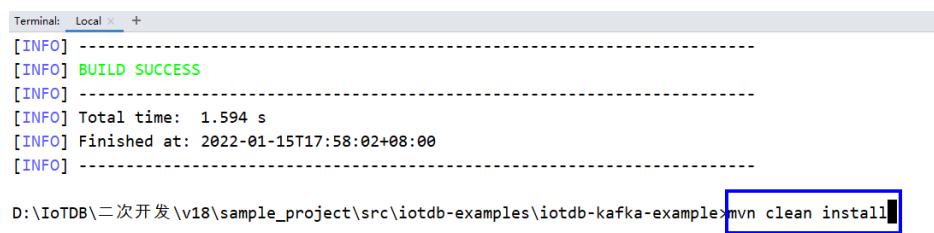
- Method 1:  
Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.  
Choose **Maven** > *Sample project name* > **Lifecycle** > **install** and double-click **install** to run the **install** command of Maven.

Figure 2-192 Maven tools clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the file.

Figure 2-193 Entering **mvn clean install** in the IDEA Terminal text box



After the compilation is completed, the message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

## Step 2 Prepare the dependent JAR file.

- Go to the client installation directory, create the **lib** directory, and import the JAR package generated in **Step 1** to the **lib** directory, for example, **/opt/hadoopclient/lib**.
- Go to the Kafka client and copy the JAR package on which Kafka depends to the **lib** directory in **Step 2.1**. The following is an example directory:

```
cp /opt/hadoopclient/Kafka/kafka/libs/*.jar /opt/hadoopclient/lib
```

3. Go to the IoTDB client and copy the JAR package on which IoTDB depends to the **lib** directory in **Step 2.1**. The following is an example directory:

```
cp /opt/hadoopclient/IoTDB/iotdb/lib/*.jar /opt/hadoopclient/lib
```

4. Copy all files in the **src/main/resources** directory of the IntelliJ IDEA project to the **src/main/resources** directory at the same level as the **lib** folder, that is, **/opt/hadoopclient/src/main/resources**.

**Step 3** Go to the **/opt/hadoopclient** directory and ensure that the current user has the read permission on all files in the **src/main/resources** directory and the dependency library folder. In addition, ensure that JDK has been installed and Java environment variables have been set. Then, run the following command to run the sample project:

```
java -cp /opt/hadoopclient/lib/*:/opt/hadoopclient/src/main/resources  
com.huawei.bigdata.iotdb.KafkaConsumerMultThread
```

----End

#### 2.10.4.4.2 Viewing Commissioning Results

The following information is displayed if the application is running properly:

```
[2022-01-17 14:43:57,511] INFO Consumer Thread-1 partitions:1 record:sensor_89,1642401919971,1.000000  
offsets:6951 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,511] INFO Consumer Thread-0 partitions:0 record:sensor_97,1642401919971,1.000000  
offsets:7129 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,512] INFO Consumer Thread-0 partitions:0 record:sensor_98,1642401919971,1.000000  
offsets:7130 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,512] INFO Consumer Thread-1 partitions:1 record:sensor_92,1642401919971,1.000000  
offsets:6952 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,513] INFO Consumer Thread-0 partitions:0 record:sensor_99,1642401919971,1.000000  
offsets:7131 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,513] INFO Consumer Thread-1 partitions:1 record:sensor_93,1642401919971,1.000000  
offsets:6953 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)  
[2022-01-17 14:43:57,513] INFO Consumer Thread-1 partitions:1 record:sensor_95,1642401919971,1.000000  
offsets:6954 (com.huawei.bigdata.iotdb.KafkaConsumerMultThread)
```

#### 2.10.4.5 Using a UDF

##### 2.10.4.5.1 Registering a UDF

1. Build a JAR file.

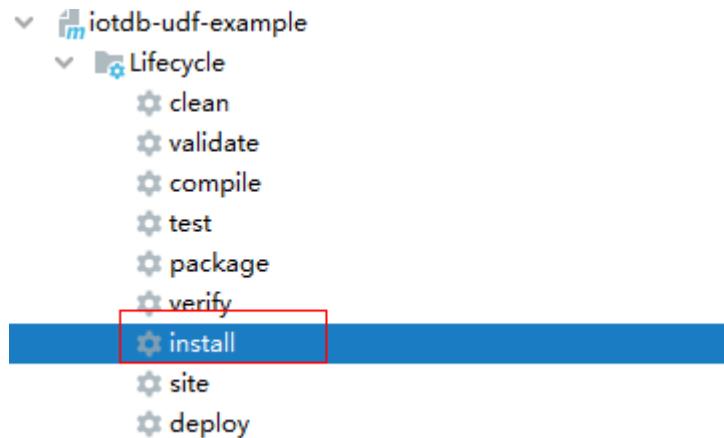
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.

Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 2-194 Maven clean and install



- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 2-195 Building result after mvn clean install is entered

```
[INFO] --- maven-install-plugin:2.4:install (default-install) @ iotdb-udf-example ---
[INFO] Installing D:\code\lib\sample_project\src\iotdb-examples\iotdb-udf-example\target\iotdb-udf-example-0.12.0-hw-ei-312005.jar to D:\repo1\com\huawei\mrs\iotdb-udf-example\0.12.0-hw-ei-312005\iot
[INFO] Installing D:\code\lib\sample_project\src\iotdb-examples\iotdb-udf-example\pom.xml to D:\repo1\com\huawei\mrs\iotdb-udf-example\0.12.0-hw-ei-312005\iotdb-udf-example-0.12.0-hw-ei-312005.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:19 min
[INFO] Finished at: 2022-02-14T15:51:35+08:00
[INFO] -----
```

After the build is complete, message "BUILD SUCCESS" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

2. Import the dependent JAR file.

Log in to the node where IoTDBServer is located as user **root**, run **su - omm** to switch to user **omm**, and import the JAR file generated in 1 to the **\$BIGDATA\_HOME/FusionInsight\_IoTDB\_\*/install/FusionInsight-IoTDB-\*/iotdb/ext/udf** directory.

**NOTICE**

During cluster deployment, ensure that a corresponding JAR package exists in the UDF JAR package path of each IoTDBServer node. You can modify IoTDB configuration **udf\_root\_dir** to specify the root path for the UDF to load JAR files.

3. Run the following SQL statement to register the UDF:

**CREATE FUNCTION <UDF-NAME> AS '<UDF-CLASS-FULL-PATHNAME>'**

For example, you can run the following statement to register a UDF named **example**:

```
CREATE FUNCTION example AS 'com.huawei.bigdata.iotdb.UDTFExample'
```

### 2.10.4.5.2 Querying a UDF

#### Basic SQL Syntax Supported

- SLIMIT / SOFFSET
- LIMIT / OFFSET
- NON ALIGN
- Queries with value filters
- Queries with time filters



Currently, aligned time series are not supported in UDF queries. An error message is reported if you use UDF queries with aligned time series selected.

#### Queries with an Asterisk (\*)

Assume that there are two time series (`root.sg.d1.s1` and `root.sg.d1.s2`).

- Execute the **SELECT example(s1) from root.sg.d1** statement.  
The result set contains the result of **example(root.sg.d1.s1)**.
- Execute the **SELECT example(s1, s2) from root.sg.d1** statement.  
The result set contains the result of **example(root.sg.d1.s1, root.sg.d1.s2)**.

#### Queries with key-value pair attributes in UDF parameters

You can pass any number of key-value pair parameters to the UDF when constructing a UDF query. The key and value in a key-value pair must be enclosed in single or double quotation marks.

---

#### NOTICE

The key-value pair parameters can be passed in only after the time series have been passed in.

---

For example:

```
SELECT example(s1, 'key1'='value1', 'key2'='value2'), example(*, 'key3'='value3') FROM root.sg.d1;  
SELECT example(s1, s2, 'key1'='value1', 'key2'='value2') FROM root.sg.d1;
```

#### Showing All Registered UDFs

Run the following SQL statement on the IoTDB client to view the registered UDFs:

```
SHOW FUNCTIONS
```

### 2.10.4.5.3 Deregistering a UDF

#### Syntax

```
DROP FUNCTION <UDF-NAME>
```

## Example

Run the following command on the IoTDB client to deregister the UDF named **example**:

```
DROP FUNCTION example
```

## 2.10.5 More Information

### 2.10.5.1 Common APIs

#### 2.10.5.1.1 Java API

IoTDB provides a connection pool (SessionPool) for native APIs. When using the APIs, you only need to specify the pool size to obtain connections from the pool. If you cannot get a connection in 60 seconds, a warning log will be printed, but the program continues to wait.

When a connection is used, it automatically returns to the pool and waits to be used next time. When a connection is damaged, it is deleted from the pool and a new connection is created to perform user operations again.

For query operations:

1. When SessionPool is used for query, the result set is SessionDataSetWrapper, the encapsulation class of SessionDataSet.
2. If a query result set is not traversed and you do not want to continue traversing, you need to call **closeResultSet** to release the connection.
3. If an exception is reported when you traverse a query result set, you need to call **closeResultSet** to release the connection.
4. You can call the **getColumnName()** method of SessionDataSetWrapper to obtain the column names in the result set.

**Table 2-115** Sessions APIs and corresponding parameters

Method	Description
<ul style="list-style-type: none"><li>• Session(String host, int rpcPort)</li><li>• Session(String host, String rpcPort, String username, String password)</li><li>• Session(String host, int rpcPort, String username, String password)</li></ul>	Initializes a session.
Session.open()	Opens a session.
Session.close()	Closes a session.
void createDatabase(String database)	Creates a database.

Method	Description
<ul style="list-style-type: none"> <li>• void deleteDatabase(String storageGroup)</li> <li>• void deleteDatabases(List&lt;String&gt; storageGroups)</li> </ul>	Deletes one or more databases.
<ul style="list-style-type: none"> <li>• void createTimeseries(String path, TSDataType dataType, TSEncoding encoding, CompressionType compressor, Map&lt;String, String&gt; props, Map&lt;String, String&gt; tags, Map&lt;String, String&gt; attributes, String measurementAlias)</li> <li>• void createMultiTimeseries(List&lt;String&gt; paths, List&lt;TSDataType&gt; dataTypes, List&lt;TSEncoding&gt; encodings, List&lt;CompressionType&gt; compressors, List&lt;Map&lt;String, String&gt;&gt; propsList, List&lt;Map&lt;String, String&gt;&gt; tagsList, List&lt;Map&lt;String, String&gt;&gt; attributesList, List&lt;String&gt; measurementAliasList)</li> </ul>	Creates one or more time series.
<ul style="list-style-type: none"> <li>• void deleteTimeseries(String path)</li> <li>• void deleteTimeseries(List&lt;String&gt; paths)</li> </ul>	Deletes one or more time series.
<ul style="list-style-type: none"> <li>• void deleteData(String path, long time)</li> <li>• void deleteData(List&lt;String&gt; paths, long time)</li> </ul>	Deletes the data of one or more time series before or at a specific time point.
void insertRecord(String deviceId, long time, List<String> measurements, List<String> values)	Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. Servers need to perform type inference, which may take extra time.
void insertTablet(Tablet tablet)	Inserts a tablet, which contains multiple rows of non-empty data blocks. The columns in each row are the same.
void insertTablets(Map<String, Tablet> tablet)	Inserts multiple tablets.
void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList, List<List<String>> valuesList)	Inserts multiple records. Servers need to perform type inference, which may take extra time.

Method	Description
void insertRecord(String deviceld, long time, List<String> measurements, List<TSDDataType> types, List<Object> values)	Inserts a record, which contains the data of multiple measurement points of a device at a timestamp. With the data type information, servers do not need to perform type inference, improving the performance.
void insertRecords(List<String> devicelds, List<Long> times, List<List<String>> measurementsList, List<List<TSDDataType>> typesList, List<List<Object>> valuesList)	Inserts multiple records. With the data type information, servers do not need to perform type inference, improving the performance.
submitApplication(SubmitApplicationRequest request)	Used by the client to submit a new application to ResourceManager.
void insertRecordsOfOneDevice(String deviceld, List<Long> times, List<List<String>> measurementsList, List<List<TSDDataType>> typesList, List<List<Object>> valuesList)	Inserts multiple records of the same device.
SessionDataSet executeRawDataQuery(List<String> paths, long startTime, long endTime)	Queries raw data. The interval includes the start time but does not include the end time.
SessionDataSet executeQueryStatement(String sql)	Executes query statements.
void executeNonQueryStatement(String sql)	Executes non-query statements.

**Table 2-116** Test APIs

Method	Description
<ul style="list-style-type: none"><li>• void testInsertRecords(List&lt;String&gt; devicelds, List&lt;Long&gt; times, List&lt;List&lt;String&gt;&gt; measurementsList, List&lt;List&lt;String&gt;&gt; valuesList)</li><li>• void testInsertRecords(List&lt;String&gt; devicelds, List&lt;Long&gt; times, List&lt;List&lt;String&gt;&gt; measurementsList, List&lt;List&lt;TSDDataType&gt;&gt; typesList, List&lt;List&lt;Object&gt;&gt; valuesList)</li></ul>	Tests testInsertRecords. A response is returned immediately after data is transmitted to the server. No data is written.

Method	Description
<ul style="list-style-type: none"><li>• void testInsertRecord(String deviceld, long time, List&lt;String&gt; measurements, List&lt;String&gt; values)</li><li>• void testInsertRecord(String deviceld, long time, List&lt;String&gt; measurements, List&lt;TSDataType&gt; types, List&lt;Object&gt; values)</li></ul>	Tests insertRecord. A response is returned immediately after data is transmitted to the server. No data is written.
void testInsertTablet(Tablet tablet)	Tests insertTablet. A response is returned immediately after data is transmitted to the server. No data is written.

## 2.11 Kafka Development Guide

### 2.11.1 Overview

#### 2.11.1.1 Development Environment Preparation

##### Kafka Introduction

Kafka is a distributed message release and subscription system. With features similar to JMS, Kafka processes active streaming data.

Kafka is applicable to message queuing, behavior tracing, operation & maintenance (O&M) data monitoring, log collection, streaming processing, event tracing, and log persistence.

Kafka features:

- High throughput
- Message persistence to disks
- Scalable distributed system
- Fault-tolerant
- Support for online and offline scenarios

##### Interface Type Introduction

APIs provided by Kafka can be divided into two types: Producer API and Consumer API. Both the types of APIs contain Java API. For details, see section [Java API](#).

#### 2.11.1.2 Common Concepts

- **Topic**

A same type of messages maintained by Kafka.

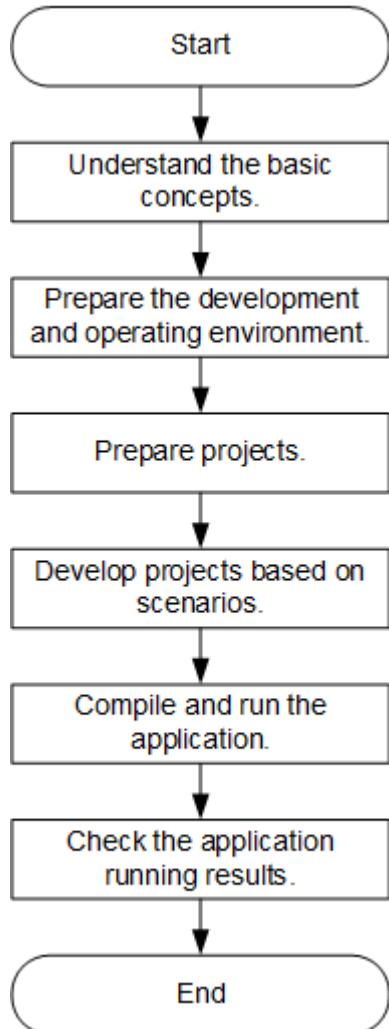
- **Partition**  
One topic can be divided into multiple partitions, and each partition corresponds to an appendant and log file whose sequence is fixed.
- **Producer**  
Role in a Kafka topic to which messages are sent.
- **Consumer**  
Role that obtains messages from Kafka topics.
- **Broker**  
A node server in a Kafka cluster.

### 2.11.1.3 Development Process

Kafka client roles include Producer and Consumer, which share the same application development process.

[Figure 2-196](#) and [Table 2-117](#) show each phase of the development process.

[Figure 2-196](#) Kafka client application development process



**Table 2-117 Kafka client application development process description**

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Kafka, and determine whether the desired role is Producer or Consumer based on the actual scenario.	<a href="#">Common Concepts</a>
Prepare the development and operating environment.	The Java language is recommended for the development of Java applications. The IntelliJ IDEA tool can be used. The Kafka running environment is the Kafka client. Install and configure the client according to the guide.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare projects	Kafka provides program samples for different scenarios. You can import the program samples for learning.	<a href="#">Configuring and Importing Sample Projects</a>
Develop projects based on scenarios.	Producer and Consumer API usage samples are provided and cover API, and multi-thread usage scenarios, helping you quickly know Kafka interfaces well.	<a href="#">Developing an Application</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
View application running results.	Program running results will be written to and printed on the console. You can use a Linux client to consume topic data to check whether data has been successfully written.	<a href="#">Application Commissioning</a>

## 2.11.2 Environment Preparation

## 2.11.2.1 Preparing for Development and Operating Environment

### Preparing Development Environment

[Table 2-118](#) describes the environment required for secondary development.

**Table 2-118** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. JDK 1.8 is used. IntelliJ IDEA 2019.1 or other compatible versions are used.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li></ul>

Item	Description
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <code>com.ibm.jsse2.overrideDefaultTLS</code> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
7-zip	It is a tool used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#) and choose **Homepage > Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Kafka\config**. Obtain the Kafka-related configuration file.

**Table 2-119** describes the main configuration files.

**Table 2-119** Configuration file

Document Name	Function
server.properties	Configures the Kafka server.
client.properties	Configures the Kafka client.
producer.properties	Configures producer of kafka parameters.
consumer.properties	Configures consumer of kafka parameters.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

#### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
  - The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all JAR package in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Kafka/install\_files/kafka/libs** directory to the **lib** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/Kafka/kafka/libs**.

For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Kafka/install_files/kafka/libs/* root@IP address of the client  
node:/opt/hadoopclient/Kafka/kafka/libs
```

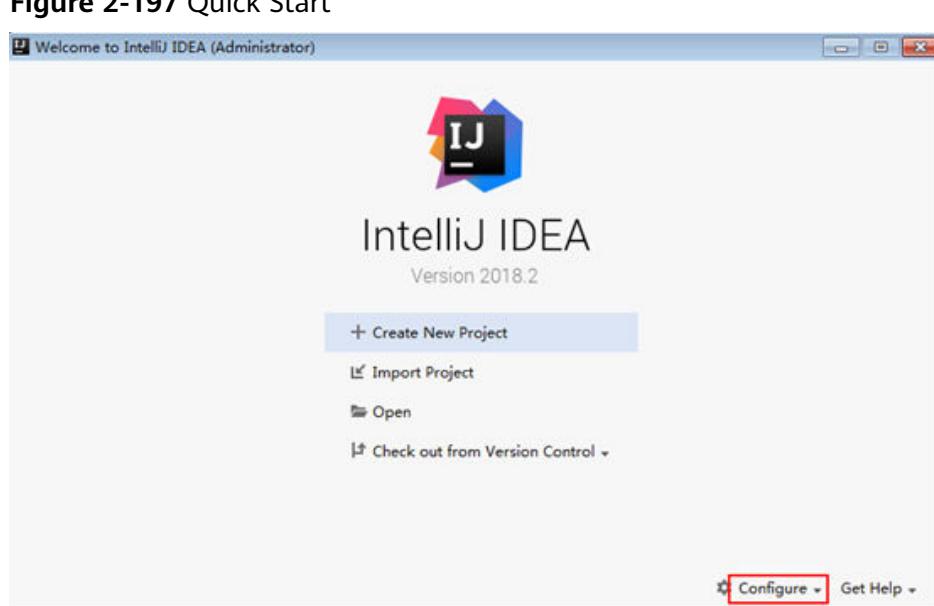
- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 2.11.2.2 Configuring and Importing Sample Projects

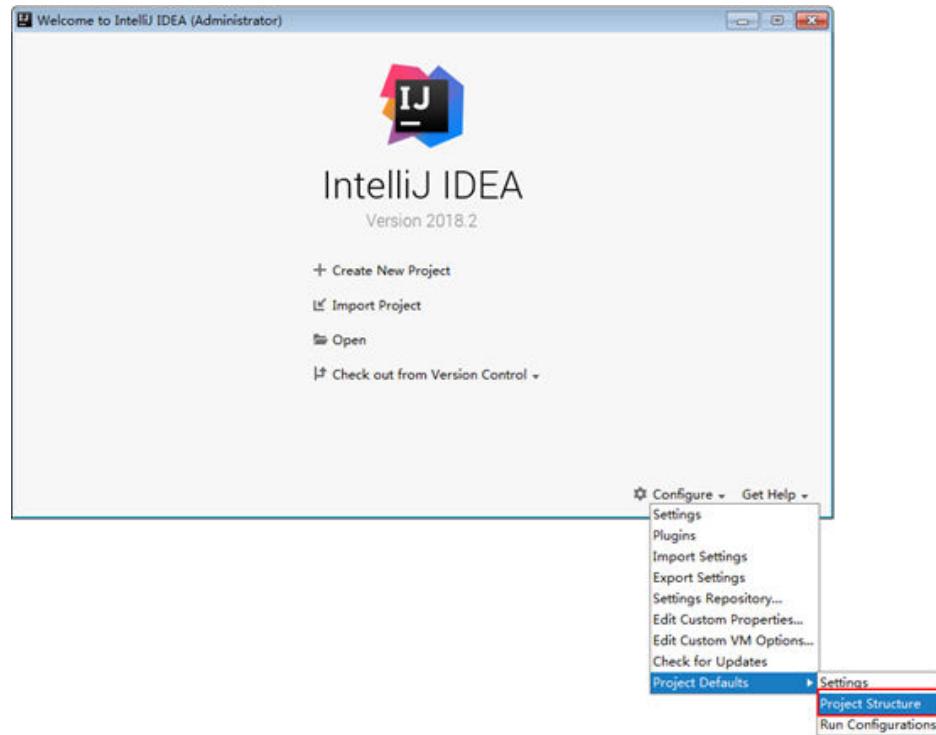
- Step 1** Obtain the sample project folder **kafka-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Copy the all the cluster configuration file obtained in section [Preparing an Operating Environment](#) to the **kafka-examples\src\main\resources** directory of the sample project.
- Step 3** After installing the IntelliJ IDEA and JDK tools, configure JDK in IntelliJ IDEA.
  1. Open IntelliJ IDEA and click **Configure**.

Figure 2-197 Quick Start



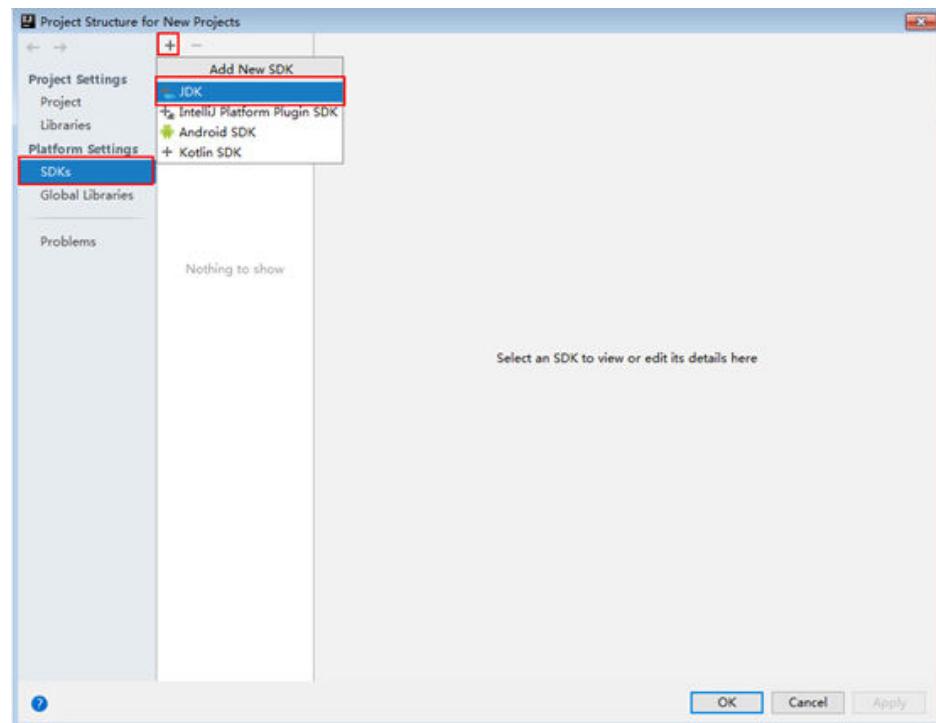
2. Choose **Project Defaults > Project Structure**.

**Figure 2-198** Configure



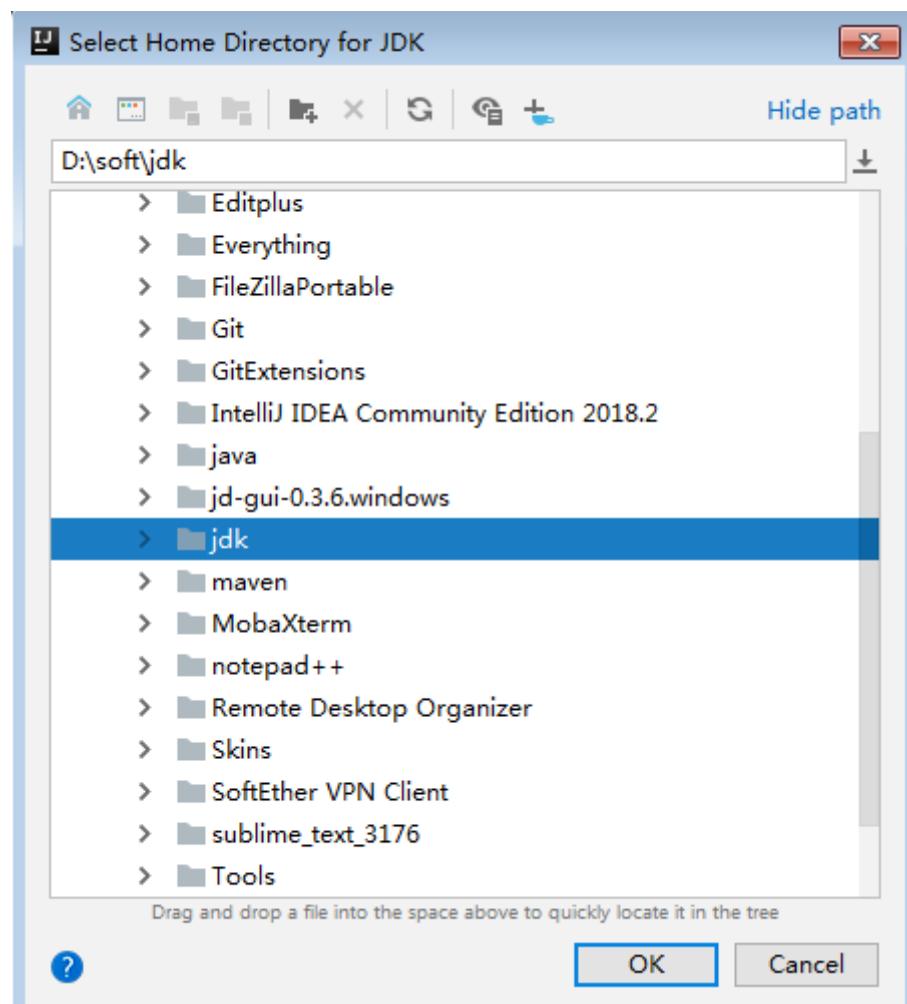
3. In the displayed **Project Structure for New Projects** interface, click **SDKs**. Then click the plus sign (+) and choose **JDK**.

Figure 2-199 Project Structure for New Projects



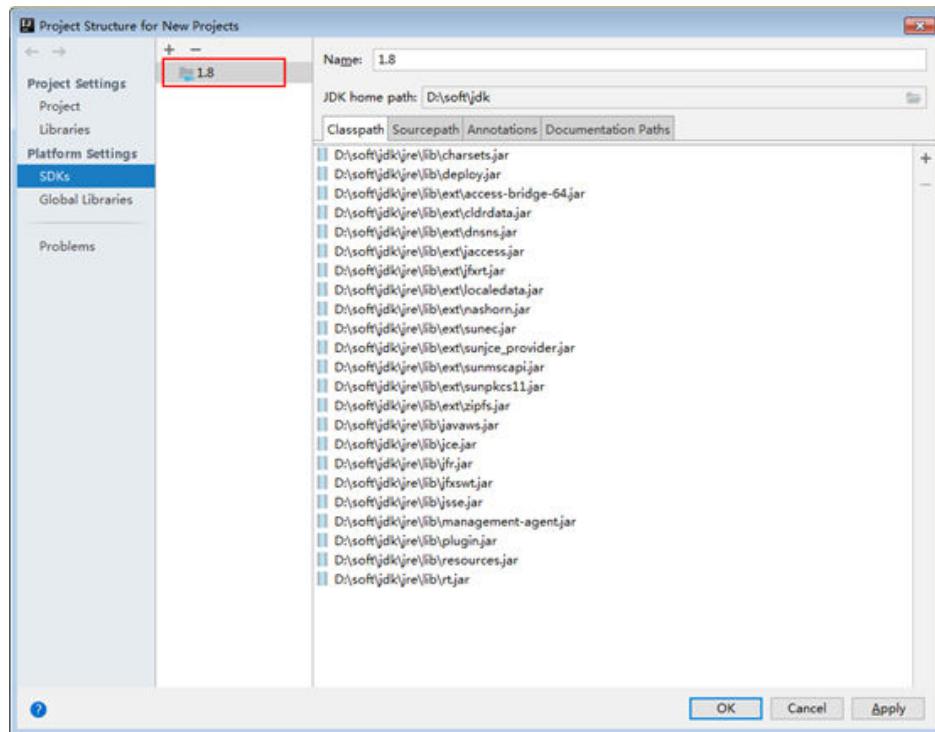
4. In the displayed **Select Home Directory for JDK** interface, select the JDK directory, and click **OK**.

Figure 2-200 Select Home Directory for JDK



5. Click **OK**.

Figure 2-201 Completing the JDK configuration



**Step 4** Import example projects into the IntelliJ IDEA development environment.

1. Choose **Open**.

The dialog box for browsing directories is displayed.

2. Select the sample project folder and click **OK**.

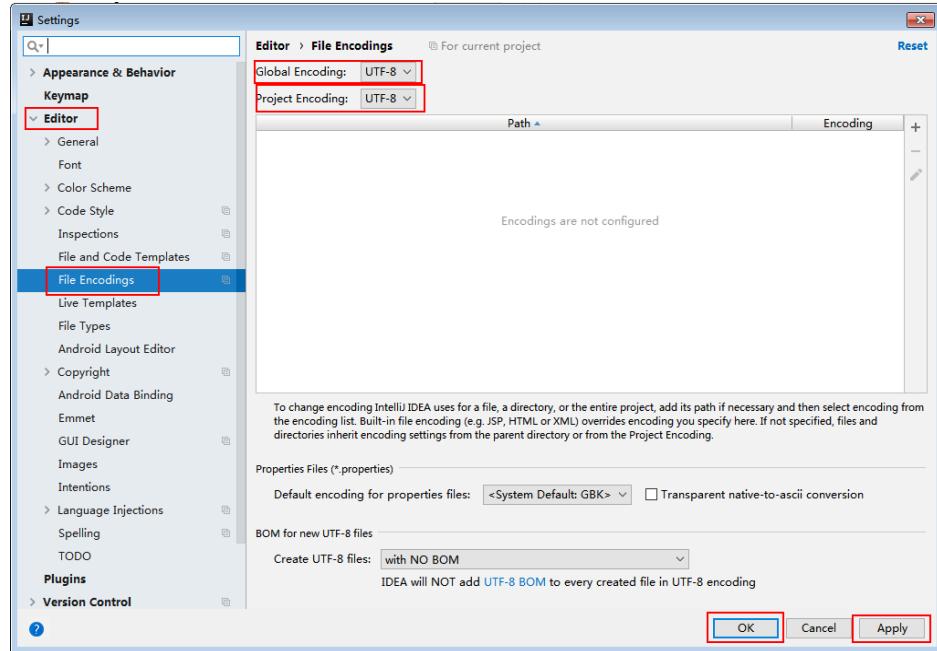
**Step 5** Set the IntelliJ IDEA text file coding format to prevent invalid characters.

1. On the IntelliJ IDEA menu bar, choose **File > Settings**.

The **Settings** window is displayed.

2. Choose **Editor > File Encodings** in the **Project Encoding** area and **Global Encoding** area, set the parameter to **UTF-8**, click **Apply**, and then click **OK**, as shown in [Figure 2-202](#).

Figure 2-202 Setting the IntelliJ IDEA coding format



----End

## 2.11.3 Developing an Application

### 2.11.3.1 Typical Scenario Description

#### Scenario

Kafka is a distributed message system, in which messages can be publicized or subscribed. A Producer is to be developed to send a message to a topic of a Kafka cluster every second, and a Consumer is to be implemented to ensure that the topic is subscribed and that messages of the topic are consumed in real time.

#### Development Idea

1. Use a Linux client to create a topic.
2. Develop a Producer to produce data to the topic.
3. Develop a Consumer to consume the data of the topic.

#### Suggestions on Performance Tuning

1. Create a topic and plan its partitions based on service requirements. The number of partitions limits the number of concurrent consumers.
2. The key value of a message must be variable to ensure even message distribution.
3. Consumers are advised to proactively submit the offset to avoid repeated consumption.

4. For details about other tuning suggestions, see section "Kafka" in *MapReduce Service (MRS) 3.3.1-LTS Performance Tuning Guide (for Huawei Cloud Stack 8.3.1)*.

## 2.11.3.2 Example Code Description

### 2.11.3.2.1 Producer API Usage Sample

#### Function Description

The following code snippet belongs to the **run** method of the **com.huawei.bigdata.kafka.example.Producer** class. It is used by the Producer APIs to produce messages for the security topic.

#### Code Sample

```
/*
 * The Producer thread executes a function to send messages periodically.
 */
public void run() {
    LOG.info("New Producer: start.");
    int messageNo = 1;

    while (messageNo <= MESSAGE_NUM) {
        String messageStr = "Message_" + messageNo;
        long startTime = System.currentTimeMillis();

        // Construct message records.
        ProducerRecord<Integer, String> record = new ProducerRecord<Integer, String>(topic, messageNo,
messageStr);

        if (isAsync) {
            // Sending in asynchronous mode
            producer.send(record, new DemoCallBack(startTime, messageNo, messageStr));
        } else {
            try {
                // Sending in synchronous mode
                producer.send(record).get();
                long elapsedTime = System.currentTimeMillis() - startTime;
                LOG.info("message(" + messageNo + ", " + messageStr + ") sent to topic(" + topic + ") in " +
elapsedTime + " ms.");
            } catch (InterruptedException ie) {
                LOG.info("The InterruptedException occurred : {}.", ie);
            } catch (ExecutionException ee) {
                LOG.info("The ExecutionException occurred : {}.", ee);
            }
        }
        messageNo++;
    }
}
```

### 2.11.3.2.2 Consumer API Usage Sample

#### Function Description

The following code sample belongs to the **com.huawei.bigdata.kafka.example.Consumer** class. It is used to enable the Consumer API to subscribe a secure topic and consume messages.

## Code Sample

```
/*
 * Consumer constructor.
 * @param topic Name of the subscribed topic
 */

public Consumer(String topic) {
    super("KafkaConsumerExample", false);
    // Initializes the configuration parameters required for starting the consumer. For details, see the code.
    Properties props = initProperties();
    consumer = new KafkaConsumer<Integer, String>(props);
    this.topic = topic;
}

public void doWork() {
    // Subscribe
    consumer.subscribe(Collections.singletonList(this.topic));
    // Message consumption request
    ConsumerRecords<Integer, String> records = consumer.poll(waitTime);
    // Message Processing
    for (ConsumerRecord<Integer, String> record : records) {
        LOG.info("[ConsumerExample], Received message: (" + record.key() + ", " + record.value() + ") at
offset " + record.offset());
    }
}
```

### 2.11.3.2.3 Multi-thread Producer Sample

#### Function Description

The multi-thread producer function is implemented based on the code sample described in section [Producer API Usage Sample](#). Multiple producer threads can be started. Each thread sends messages to the partition whose key is the same as the thread ID.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ProducerMultThread** class, and these code snippets are used to enable multiple threads to produce data.

## Code Sample

```
/*
 * Specify the current ThreadID as the key value and send data.
 */
public void run()
{
LOG.info("Producer: start.");

    // Record the number of messages.
int messageCount = 1;

    // Specify the number of messages sent by each thread.
int messagesPerThread = 5;
while (messageCount <= messagesPerThread)
{
    // Specify the content of messages to be sent.
    String messageStr = new String("Message_" + sendThreadId + "_" + messageCount);

    // Specify a key value for each thread to enable the thread to send messages to only a specified
partition.
    String key = String.valueOf(sendThreadId);

    // Send messages.
    producer.send(new KeyedMessage<String, String>(sendTopic, key, messageStr));
}
```

```
LOG.info("Producer: send " + messageStr + " to " + sendTopic + " with key: " + key);
    messageCount++;
}
}
```

#### 2.11.3.2.4 Multi-thread Consumer Sample

#### Function Description

The multi-thread consumer function is implemented based on the code sample described in section [Consumer API Usage Sample](#). The number of consumer threads that can be started to consume the messages in partitions is the same as the number of partitions in the topic.

The following code snippet belongs to the run method in the **com.huawei.bigdata.kafka.example.ConsumerMultThread** class, and these code snippets are used to implement concurrent consumption of messages in a specified topic.

#### Code Sample

```
/*
 * Start the concurrent multi-thread consumer.
 */
public void run() {
    LOG.info("Consumer: start.");
    Properties props = Consumer.initProperties();
    // Start a specified number of consumer threads to consume.
    // Note: When this parameter is larger than the number of partitions of the topic to be consumed, the
    extra threads fails to consume data.
    for (int threadNum = 0; threadNum < CONCURRENCY_THREAD_NUM; threadNum++) {
        new ConsumerThread(threadNum, topic, props).start();
        LOG.info("Consumer Thread " + threadNum + " Start.");
    }
}

private class ConsumerThread extends ShutdownableThread {
    private int threadNum = 0;
    private String topic;
    private Properties props;
    private KafkaConsumer<String, String> consumer = null;

    /**
     * Construction method of the consumer thread class
     *
     * @param threadNum Thread number
     * @param topic      topic
     */
    public ConsumerThread(int threadNum, String topic, Properties props) {
        super("ConsumerThread" + threadNum, true);
        this.threadNum = threadNum;
        this.topic = topic;
        this.props = props;
        this.consumer = new KafkaConsumer<String, String>(props);
    }

    public void doWork() {
        consumer.subscribe(Collections.singleton(this.topic));
        ConsumerRecords<String, String> records = consumer.poll(waitTime);
        for (ConsumerRecord<String, String> record : records) {
            LOG.info("Consumer Thread-" + this.threadNum + " partitions:" + record.partition() + " record: "
                + record.value() + " offsets: " + record.offset());
        }
    }
}
```

### 2.11.3.3 Kafka Streams Sample Code Description

#### Scenario Description

Kafka Streams is a lightweight stream processing framework provided by Apache Kafka. The input and output of Kafka Streams are stored in the Kafka cluster.

The following describes the most common WordCount samples.

#### Development Guideline

1. Create two topics on the Linux client to serve as the input and output topics.
2. Develop a Kafka Streams to implement the word count function. The system collects statistics on the number of words in each message by reading the message in the input topic, consumes data from the output topic, and outputs the statistical result in the form of a key-value pair.

### 2.11.3.4 Kafka Streams Sample Code Description

#### 2.11.3.4.1 High level KafkaStreams API Usage Sample

#### Function Description

The following code snippets are used in the `createWordCountStream` method of the `com.huawei.bigdata.kafka.example.WordCountDemo` class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

#### Code Sample

```
static void createWordCountStream(final StreamsBuilder builder) {  
    // Receive the input records from input-topic.  
    final KStream<String, String> source = builder.stream(INPUT_TOPIC_NAME);  
  
    // Aggregate the calculation result of the key-value pair.  
    final KTable<String, Long> counts = source  
        // Process the received records and split according to the regular expression REGEX_STRING.  
        .flatMapValues(value ->  
            Arrays.asList(value.toLowerCase(Locale.getDefault()).split(REGEX_STRING)))  
        // Aggregate the calculation result of the key-value pair.  
        .groupBy((key, value) -> value)  
        // The final calculation result  
        .count();  
  
    // Output the key-value pair of the calculation result from the output topic.  
    counts.toStream().to(OUTPUT_TOPIC_NAME, Produced.with(Serdes.String(), Serdes.Long()));  
}
```

### 2.11.3.4.2 Low level KafkaStreams API Usage Sample

#### Function Description

The following code snippets are used in the **com.huawei.bigdata.kafka.example.WordCountProcessorDemo** class to implement the following function:

Collects statistics on input records. Same words are divided into a group, which is used as a key value. The occurrence times of each word are calculated as a value and are output in the form of a key-value pair.

#### Code Sample

```
private static class MyProcessorSupplier implements ProcessorSupplier<String, String> {
    @Override
    public Processor<String, String> get() {
        return new Processor<String, String>() {
            // ProcessorContext instance, which provides the access of the metadata of the records being
            processed
            private ProcessorContext context;
            private KeyValueStore<String, Integer> kvStore;

            @Override
            @SuppressWarnings("unchecked")
            public void init(ProcessorContext context) {
                // Save processor context in the local host, because it will be used for punctuate() and commit().
                this.context = context;
                // Execute punctuate() once every second.
                this.context.schedule(Duration.ofSeconds(1), PunctuationType.STREAM_TIME, timestamp -> {
                    try (final KeyValueIterator<String, Integer> iter = kvStore.all()) {
                        System.out.println("----- " + timestamp + " ----- ");
                        while (iter.hasNext()) {
                            final KeyValue<String, Integer> entry = iter.next();
                            System.out.println("[ " + entry.key + ", " + entry.value + " ]");
                            // Send the new records to the downstream processor as key-value pairs.
                            context.forward(entry.key, entry.value.toString());
                        }
                    }
                });
                // Search for the key-value states storage area named KEY_VALUE_STATE_STORE_NAME to
                memorize the recently received input records.
                this.kvStore = (KeyValueStore<String, Integer>)
                context.getStateStore(KEY_VALUE_STATE_STORE_NAME);
            }

            // Process the receiving records of input topic. Split the records into words, and count the words.
            @Override
            public void process(String dummy, String line) {
                String[] words = line.toLowerCase(Locale.getDefault()).split(REGEX_STRING);

                for (String word : words) {
                    Integer oldValue = this.kvStore.get(word);

                    if (oldValue == null) {
                        this.kvStore.put(word, 1);
                    } else {
                        this.kvStore.put(word, oldValue + 1);
                    }
                }
            }

            @Override
            public void close() {
            }
        };
    }
}
```

```
    }
```

### 2.11.3.5 Sample Code for Connecting Kafka to Spring Boot

#### Function

Spring Boot is used to produce and consume Kafka clusters.

#### Sample Code

Use Spring Boot to implement Kafka production and consumption.

```
@RestController
public class MessageController {
    private final static Logger LOG = LoggerFactory.getLogger(MessageController.class);
    @Autowired
    private KafkaProperties kafkaProperties;
    @GetMapping("/produce")
    public String produce() {
        Producer producerThread = new Producer();
        producerThread.init(this.kafkaProperties);
        producerThread.start();
        String message = "Start to produce messages";
        LOG.info(message);
        return message;
    }
    @GetMapping("/consume")
    public String consume() {
        Consumer consumerThread = new Consumer();
        consumerThread.init(this.kafkaProperties);
        consumerThread.start();
        LOG.info("Start to consume messages");
        // Wait for 180s and close the consumer. Modification can be made during actual execution.
        try {
            Thread.sleep(consumerThread.getThreadAliveTime());
        } catch (InterruptedException e) {
            LOG.info("Occurred InterruptedException: ", e);
        } finally {
            consumerThread.close();
        }
        return String.format("Finished consume messages");
    }
}
```

**Produce** indicates the message production interface, **consume** indicates the message consumption interface, and **KafkaProperties** indicates the client parameter. The parameters need to be modified based on the actual service.

### NOTE

The **KafkaProperties** parameter in the sample code can be configured in **springboot > kafka-examples > src > main > resources > application.properties** or manually compiled in the **application.properties** file in the sample running environment. If no default value is specified, the configuration is mandatory.

- **bootstrap.servers**: Broker address list of the Kafka cluster. The format is **IP address:Port, IP address:Port, IP address:Port**. In the IPv6 environment, add **[]** to the IP address, for example, **[1:2:3:4:5:6:7:8]:21007**.
- **security.protocol**: authentication protocol used by the Kafka client. The default value is **PLAINTEXT**.
- **topic**: name of the production and consumption topic. The default value is **example-metric1**.
- **isAsync**: whether to use asynchronous production. The default value is **false**.
- **consumer.alive.time**: lifetime of the consumer thread. The default value is **180000**, in milliseconds.
- **server.port**: port for accessing the Spring Boot server. The default value is **8080**, which can be customized.
- **server.address**: IP address bound to the Spring Boot server during the startup. The default value is **0.0.0.0**, The default value is 0.0.0.0, which needs to be changed to the IP address of the node where Spring Boot is deployed.
- **is.security.mode**: whether the client connects to the cluster in security mode. The default value is **false**.

## Sample Running

### Step 1 Obtain the **huawei-spring-boot-kafka-examples-\*jar** package.

Find the POM file in the **springboot/kafka-examples** directory of the sample code, and use the Maven install tool to compile the Spring Boot sample in the same directory as the POM file. A target folder is generated, and **huawei-spring-boot-kafka-examples-\*jar** is obtained from the **target** folder.

### Step 2 Create a directory on Windows or Linux as the running directory.

- Create the **D:\Spring** directory in the Windows OS and upload the **huawei-spring-boot-kafka-examples-\*jar** and **application.properties** files to the current directory.
- Create the **/opt/spring** directory on the Linux OS and upload the **huawei-spring-boot-kafka-examples-\*jar** and **application.properties** files to the current directory.

### Step 3 Run the corresponding command to start Spring Boot:

- In Windows, open the CLI in the **D:\Spring** directory and run the following command:  
**java -jar huawei-spring-boot-kafka-examples-\*jar**
- In Linux, run the following command in the **/opt/spring** directory:  
**java -jar huawei-spring-boot-kafka-examples-\*jar**

### Step 4 Produce data.

- For a Windows OS, open a browser and enter **http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/produce** to generate data to Broker, as shown in the following figure.

Figure 2-203 Producing data



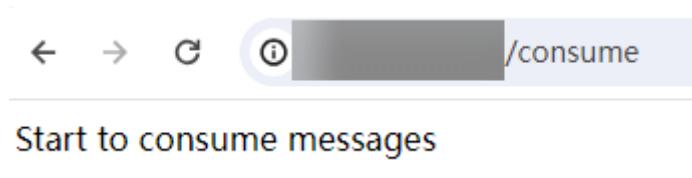
- For a Linux OS, run the **curl** command to access Spring Boot.

**curl http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/produce**

**Step 5** Consume data.

- For a Windows OS, open a browser and enter **http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/consume** to consume data from broker, as shown in the following figure.

Figure 2-204 Consuming data



- For a Linux OS, run the **curl** command to access Spring Boot.

**curl http://IP address bound during Spring Boot startup:Port bound during Spring Boot startup/consume**

----End

## 2.11.4 Application Commissioning

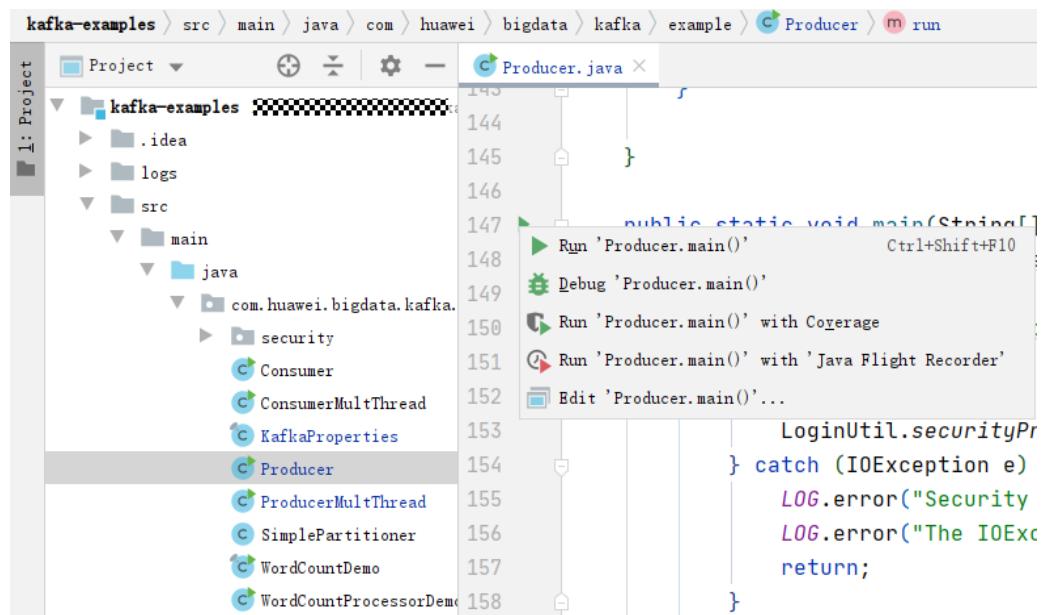
### 2.11.4.1 Commissioning an Application in Windows

#### Starting Producer

**Step 1** Ensure that the mappings between the host names and service IP addresses of all the hosts in the remote cluster are configured in the local **hosts** file.

**Step 2** Run **Producer.java** on IntelliJ IDEA, as shown in [Figure 2-205](#).

Figure 2-205 Run Producer.java on IntelliJ IDEA



**Step 3** The console window appears. You can find that Producer is sending messages to the default topic (example-metric1). One piece of log is printed when every 10 messages are sent.

Figure 2-206 Procedurer running window

```

[2019-06-12 10:31:37,865] INFO Updated cluster metadata version 2 to Cluster(id = 1cPugjn8QAernMH8RS_~JA, nodes = [187,
[2019-06-12 10:31:51,729] INFO Updated cluster metadata version 3 to Cluster(id = 1cPugjn8QAernMH8RS_~JA, nodes = [187,
[2019-06-12 10:31:53,140] INFO The Producer have send 10 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:54,516] INFO The Producer have send 20 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:55,906] INFO The Producer have send 30 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:57,299] INFO The Producer have send 40 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:31:58,686] INFO The Producer have send 50 messages. (com.huawei.bigdata.kafka.example.NewProducer)
[2019-06-12 10:32:00,070] INFO The Producer have send 60 messages. (com.huawei.bigdata.kafka.example.NewProducer)

```

----End

## Starting Consumer

**Step 1** Run the `Consumer.java` file.

**Step 2** Click **Run**. In the console window that appears, you can find that Producer starts after Consumer successfully starts, and then you can view messages received in real time.

Figure 2-207 Consumer.java running window

```

[2019-06-23 17:49:48,609] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] (Re-)joining group (org.apache.kafka.clients.consumer.internals.AbstractCoordinator)
[2019-06-23 17:49:51,865] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Successfully joined group with generation 5 (org.apache.kafka.clients.consumer.internals.ConsumerCoordinator)
[2019-06-23 17:49:51,866] INFO [Consumer clientId=consumer-1, groupId=DemoConsumer] Setting newly assigned partitions [example-metric1-0, example-metric1-1] (org.apache.kafka.clients.consumer.internals.ConsumerCoordinator)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (1, Message_1) at offset 180 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (5, Message_5) at offset 189 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (8, Message_8) at offset 190 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (9, Message_9) at offset 191 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (15, Message_15) at offset 192 (com.huawei.bigdata.kafka.example.NewConsumer)
[2019-06-23 17:49:56,670] INFO [NewConsumerExample], Received message: (18, Message_18) at offset 193 (com.huawei.bigdata.kafka.example.NewConsumer)

```

----End

## Starting Other Code Samples

The procedures for starting other code samples are similar to the procedures for starting Producer and Consumer in this section.

### 2.11.4.2 Commissioning an Application in Linux

#### Scenario

Run a sample program in Linux after code development is complete.

#### Procedure

##### Step 1 Export a JAR package.

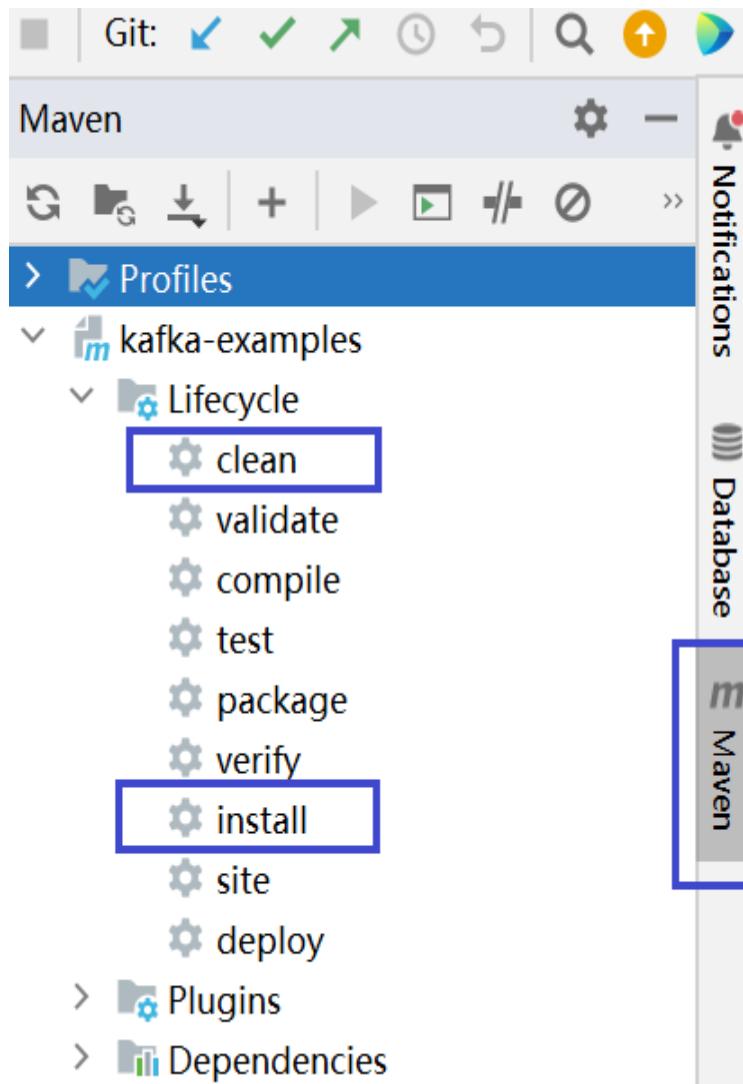
You can build a JAR file in either of the following ways:

- Method 1:

Choose **Maven** > *Sample projectname* > **Lifecycle** > **clean**, double click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **install**, double click **install** to run the **install** command of Maven.

Figure 2-208 Maven tools: clean and install



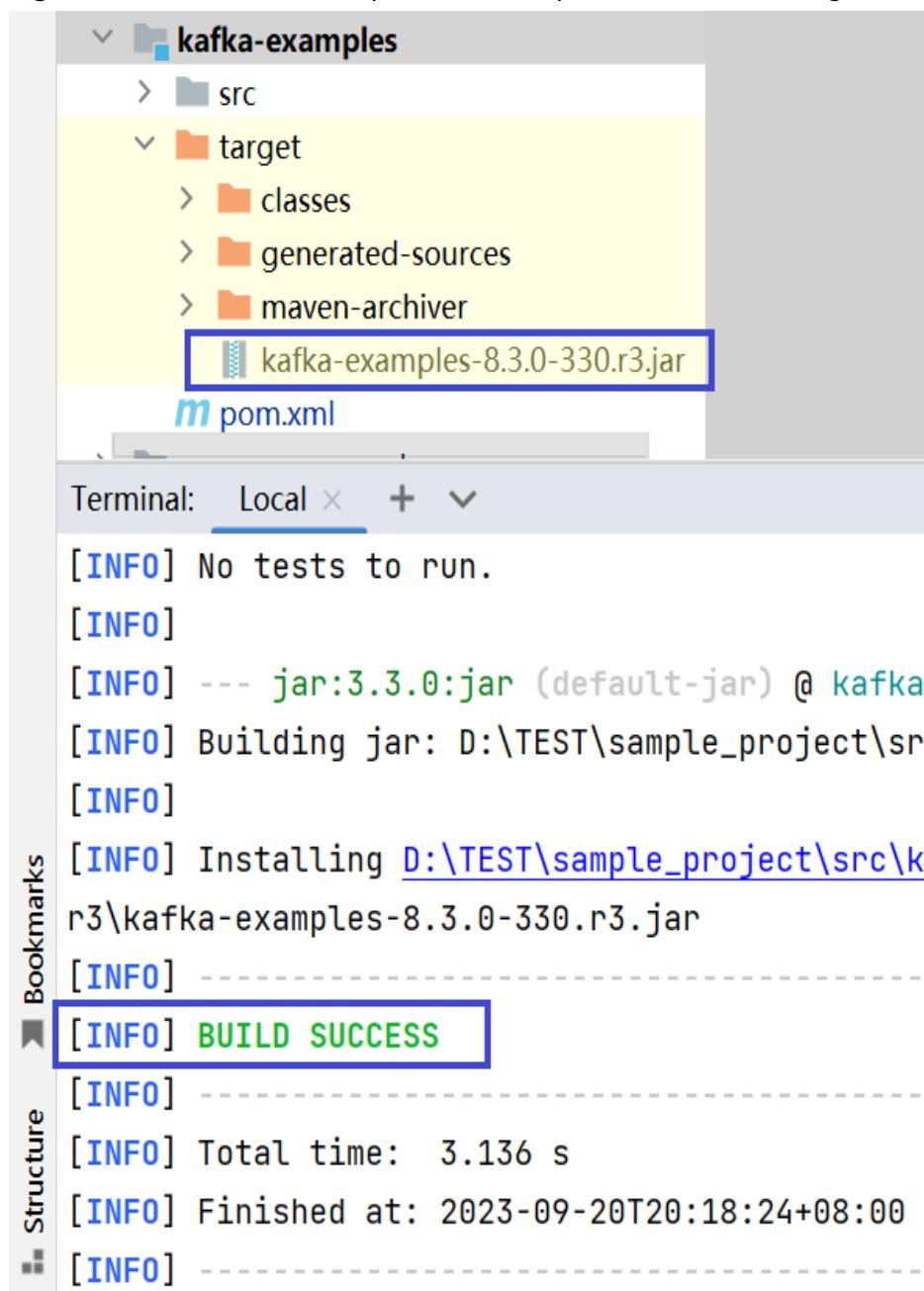
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean install** command to compile the **pom.xml** file.

Figure 2-209 Enter mvn clean compile in the IDEA Terminal text box.

The screenshot shows the IntelliJ IDEA terminal window with the command `mvn clean install` entered. The terminal output area is currently empty.

After the compilation is completed, the message "Build Success" is displayed, the **target** directory is generated, and the generated JAR file is stored in the **target** directory.

Figure 2-210 When the compilation is completed, the JAR file is generated.



- Step 2** Copy the JAR file generated during project compilation to the `/opt/hadoopclient/Kafka/kafka/libs` directory.
- Step 3** Copy all files in the `src/main/resources` directory of the IntelliJ IDEA project to the `src/main/resources` directory at the same level as the `lib` folder, that is, `/opt/hadoopclient/src/kafka-examples/src/main/resources`.
- Step 4** Ensure that the current user has read permission of all the files in the `src/main/resources` and `libs` folders in `/opt/hadoopclient/src/kafka-examples`, jdk has been installed, and java environment variables are set. Then, run the command, for example, `java -cp /opt/hadoopclient/Kafka/kafka/libs/*:src/main/resources com.huawei.bigdata.kafka.example.Producer` to run the example project.

----End

## 2.11.4.3 Kafka Streams Sample Running Guide

### 2.11.4.3.1 High Level Kafka Streams API Sample Usage Guide

1. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic  
private static final String INPUT_TOPIC_NAME = "streams-wordcount-input";  
// sink-topic name created by the user, that is, output topic  
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-output";
```

- Create the input topic:

```
kafka-topics.sh --create --bootstrap-server  
192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --  
command-config config/client.properties --replication-factor 1 --  
partitions 1 --topic streams-wordcount-input
```

- Create the output topic:

```
kafka-topics.sh --create --bootstrap-server  
192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --  
command-config config/client.properties --replication-factor 1 --  
partitions 1 --topic streams-wordcount-output --config  
cleanup.policy=compact
```

- Run sample code **WordCountDemo.java**. For details, see[Commissioning an Application in Windows](#)and[Commissioning an Application in Linux](#).

2. Use the Linux client to write messages and view the statistics result.

Run the **kafka-console-producer.sh** command to write messages to the input topic.

```
# kafka-console-producer.sh --broker-list 192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --  
topic streams-wordcount-input --producer.config /opt/hadoopclient/Kafka/kafka/config/  
producer.properties  
>This is Kafka Streams test  
>test starting  
>now Kafka Streams is running  
>test end  
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-output --bootstrap-server  
192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --consumer.config /opt/hadoopclient/  
Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property  
print.value=true --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer --  
property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer --formatter  
kafka.tools.DefaultMessageFormatter  
this 1  
is 6  
kafka 12  
streams 8  
test 8  
test 9  
starting 1  
now 1  
kafka 13  
streams 9  
is 7  
running 1  
test 10  
end 1
```

### 2.11.4.3.2 Low level Kafka Streams API Sample Usage Guide

1. Use the Linux client to create the input and output topics. Ensure that the topic names are the same as those in the sample code, set the policy for clearing the output topic to compact, and run the sample code.

```
// source-topic name created by the user, that is, input topic  
private static final String INPUT_TOPIC_NAME = "streams-wordcount-processor-input";  
  
// sink-topic name created by the user, that is, output topic  
private static final String OUTPUT_TOPIC_NAME = "streams-wordcount-processor-output";
```

- Create the input topic:

```
kafka-topics.sh --create --bootstrap-server  
192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --  
command-config config/client.properties --replication-factor 1 --  
partitions 1 --topic streams-wordcount-processor-input
```

- Create the output topic:

```
kafka-topics.sh --create --bootstrap-server  
192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --  
command-config config/client.properties --replication-factor 1 --  
partitions 1 --topic streams-wordcount-processor-output --config  
cleanup.policy=compact
```

- Run sample code **WordCountProcessorDemo.java**. For details, see[Commissioning an Application in Windows](#) and[Commissioning an Application in Linux](#).

2. Use the Linux client to write messages and view the statistics result.

Run the **kafka-console-producer.sh** command to write messages to the input topic.

```
# kafka-console-producer.sh --broker-list 192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --  
topic streams-wordcount-processor-input --producer.config /opt/hadoopclient/Kafka/kafka/config/  
producer.properties  
>This is Kafka Streams test  
>now Kafka Streams is running  
>test end  
>
```

Run the **kafka-console-consumer.sh** command to consume data from the output topic and view the statistics result.

```
# kafka-console-consumer.sh --topic streams-wordcount-processor-output --bootstrap-server  
192.168.0.11:21005,192.168.0.12:21005,192.168.0.13:21005 --consumer.config /opt/hadoopclient/  
Kafka/kafka/config/consumer.properties --from-beginning --property print.key=true --property  
print.value=true  
is 1  
kafka 1  
streams 1  
test 1  
this 1  
end 1  
is 2  
kafka 2  
now 1  
running 1  
streams 2  
test 2  
this 1
```

## 2.11.5 More Information

## 2.11.5.1 External Interfaces

### 2.11.5.1.1 Shell

1. Query the list of topics in current clusters.

```
bin/kafka-topics.sh --list --bootstrap-server <Kafka cluster IP address:21005> --command-config config/client.properties
```

2. Query the details of a topic.

```
bin/kafka-topics.sh --describe --bootstrap-server <Kafka cluster IP address:21005> --command-config config/client.properties --topic <Topic name>
```

3. Delete a topic (this operation can be performed by an administrator).

```
bin/kafka-topics.sh --delete --bootstrap-server <Kafka cluster IP address:21005> --command-config config/client.properties --topic <Topic name>
```

4. Create a topic (this operation can be performed by an administrator).

```
bin/kafka-topics.sh --create --bootstrap-server <Kafka cluster IP address:21005> --command-config config/client.properties --partitions 6 --replication-factor 2 --topic <Topic name>
```

5. Assign permissions to the Consumer (this operation is performed by an administrator).

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:24002/kafka> --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
```

```
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21005> --command-config config/client.properties --add --allow-principal User:<User name> --consumer --topic <Topic name> --group <Consumer group name>
```

6. Assign permissions to the Producer (this operation is performed by an administrator).

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=<ZooKeeper cluster IP address:24002/kafka> --add --allow-principal User:<User name> --producer --topic <Topic name>
```

```
bin/kafka-acls.sh --bootstrap-server <Kafka cluster IP address:21005> --command-config config/client.properties --add --allow-principal User:<User name> --producer --topic <Topic name>
```

7. Produce messages (this operation requires the producer permission of the desired topic).

```
bin/kafka-console-producer.sh --broker-list <KafkaCluster IP address:21005> --topic <Topic name> --producer.config config/producer.properties
```

8. Consume data (the producer permission of the topic is required).

```
bin/kafka-console-consumer.sh --topic <Topic name> --bootstrap-server <KafkaCluster IP address:21005> --consumer.config config/consumer.properties
```

For details, see the *MapReduce Service (MRS) 3.3.1-LTS Shell O&M Commands (for Huawei Cloud Stack 8.3.1)*.

### 2.11.5.1.2 Java API

Versions of interfaces adopted by Kafka are consistent with those in Open Source Community. For details, see <https://kafka.apache.org/24/documentation.html>.

#### Major Interfaces of a Producer

**Table 2-120** Major parameters of a Producer

Parameter	Description	Remarks
bootstrap.servers	Broker address list	The Producer creates connections with the Broker based on this parameter.
security.protocol	Security protocol type	The Producer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as <b>SASL_PLAINTEXT</b> .
sasl.kerberos.service.name	Service name	This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as <b>kafka</b> .
key.serializer	Serialization of key values in messages	This parameter specifies how to serialize the key values in messages.
value.serializer	Serialization of messages	This parameter specifies how to serialize transmitted messages.

**Table 2-121** Major interface functions of a Producer

Return Value	Interface Function	Description
java.util.concurrent.Future<RecordMetadata>	send(ProducerRecord<K, V> record)	Indicates a TX interface without a callback function. Generally, the get() function of Future is used for synchronous transmission.

Return Value	Interface Function	Description
java.util.concurrent.Future<RecordMetadata>	send(ProducerRecord<K, V> record, Callback callback)	Indicates a TX interface with a callback function. Generally, this interface uses the callback function to process transmission results after asynchronous transmission.
void	onCompletion(RecordMetadata metadata, Exception exception);	Indicates the interface method for a callback function. This method is used to process asynchronous transmission results.

## Major Interfaces of a Consumer

Table 2-122 Major parameters of a Consumer

Parameter	Description	Remarks
bootstrap.servers	Broker address list	The Consumer creates connections with the Broker based on this parameter.
security.protocol	Security protocol type	The Consumer uses the security protocol of the type specified by this parameter. In security mode, only the SASL protocol is supported, so this parameter needs to be configured as <b>SASL_PLAINTEXT</b> .
sasl.kerberos.service.name	Service name	This parameter specifies the Kerberos user name used by the Kafka cluster for running. This parameter needs to be configured as <b>kafka</b> .
key.deserializer	Deserialization of key values in messages	This parameter specifies how to deserialize the key values in messages.

Parameter	Description	Remarks
value.deserializer	Deserialization of messages	This parameter specifies how to deserialize received messages.

**Table 2-123** Major interface functions of aConsumer

Return Value	Interface Function	Description
void	close()	Indicates the interface method for closing the Consumer.
void	subscribe(java.util.Collection<java.lang.String> topics)	Indicates the interface method for subscribing topics.
ConsumerRecords<K,V>	poll(final Duration timeout)	Indicates the interface method for requesting messages.

## 2.11.5.2 FAQ

### 2.11.5.2.1 Running the Producer.java Sample to Obtain Metadata Fails and "ERROR fetching topic metadata for topics..." Is Displayed, Even the Access Permission for the Related Topic

#### Troubleshooting Procedure

- Step 1** Find `bootstrap.servers` in `producer.properties` under the `conf` directory of the project, and check whether the IP address and port ID are configured correctly.
- If the IP address is inconsistent with the service IP address of the Kafka cluster, change the IP address to the correct one.
  - If the port ID is 21007 (security mode port), change it to 21005 (normal mode port).

- Step 2** Check whether network connections are correct to ensure that the current device can access the Kafka cluster normally.

----End

## 2.12 MapReduce Development Guide

### 2.12.1 Overview

### 2.12.1.1 MapReduce Overview

#### MapReduce Introduction

Hadoop MapReduce is an easy-to-use parallel computing software framework. Applications developed based on MapReduce can run on large clusters consisting of thousands of servers and process data sets larger than 1 TB in fault tolerance (FT) mode.

A MapReduce job (application or job) splits an input data set into several data blocks which then are processed by Map tasks in parallel mode. The framework sorts output results of the Map task, sends the results to Reduce tasks, and returns a result to the client. Input and output information is stored in the Hadoop Distributed File System (HDFS). The framework schedules and monitors tasks and re-executes failed tasks.

MapReduce supports the following features:

- Large-scale parallel computing
- Large data set processing
- High FT and reliability
- Reasonable resource scheduling

#### 2.12.1.2 Basic Concepts

##### Hadoop shell command

Basic hadoop shell commands include commands that are used to submit MapReduce jobs, kill MapReduce jobs, and perform operations on the HDFS.

##### MapReduce InputFormat and OutputFormat

Based on the specified InputFormat, the MapReduce framework splits data sets, reads data, provides key-value pairs for Map tasks, and determines the number of Map tasks that are started in parallel mode. Based on the OutputFormat, the MapReduce framework outputs the generated key-value pairs to data in a specific format.

Map and Reduce tasks are running based on <key,value> pairs. In other words, the framework regards the input information of a job as a group of key-value pairs and outputs a group of key-value pairs. Two groups of key-value pairs may be of different types. For a single Map or Reduce task, key-value pairs are processed in single-thread serial mode.

The framework needs to perform serialized operations on key and value classes. Therefore, the classes must support the Writable interface. To facilitate sorting operations, key classes must support the WritableComparable interface.

The input and output types of a MapReduce job are as follows:

(input) <k1,v1> -> Map -> <k2,v2> -> Summary data -> <k2, List(v2)> -> Reduce -> <k3,v3> (output)

## Job Core

In normal cases, an application only needs to inherit Mapper and Reducer classes and rewrite map and reduce methods to implement service logic. The map and reduce methods constitute the core of jobs.

## MapReduce WebUI

Allows users to monitor running or historical MapReduce jobs, view logs, and implement fine-grained job development, configuration, and optimization.

## Reduce

A processing model function that merges all intermediate values associated with the same intermediate key.

## Shuffle

A process of outputting data from a Map task to a Reduce task.

## Map

A method used to map a group of key-value pairs into a new group of key-value pairs.

### 2.12.1.3 Development Process

All stages of the development process are shown and described in [Figure 2-211](#) and [Table 2-124](#).

Figure 2-211 MapReduce development process

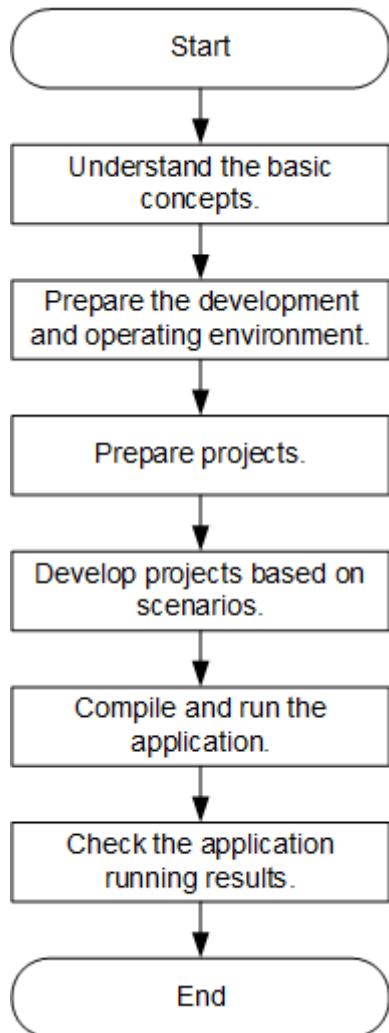


Table 2-124 Description of MapReduce development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of MapReduce are required to be understood.	<a href="#">Basic Concepts</a>
Prepare the development and operating environment.	Use IntelliJ IDEA and configure the development environment based on the reference. The running environment of MapReduce is the MapReduce client. Install and configure the client based on the reference.	<a href="#">Preparing Development and Operating Environment</a>

Stage	Description	Reference
Prepare projects	MapReduce provides sample projects in various scenarios. Sample projects can be imported for studying. Or you can create a new MapReduce project based on the reference.	<a href="#">Configuring and Importing Sample Projects</a> <a href="#">Creating a New Project (Optional)</a>
Develop projects based on scenarios.	Provide the example project. This helps users to learn about the programming interfaces of all Spark components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Commissioning the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Commissioning the Application</a>

## 2.12.2 Environment Preparation

### 2.12.2.1 Preparing Development and Operating Environment

#### Preparing Development Environment

[Table 2-125](#) describes the environment required for application development.

**Table 2-125** Development environment

Preparation Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>

Preparation Item	Description
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– Oracle JDK 1.8</li><li>– IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>– OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>– BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstanceTLS-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-jav 技术/8?topic=customization-matching-behavior-sslcontextgetinstanceTLS-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. 7-Zip 16.04 is supported.

Preparation Item	Description
checkJarsCrossPlatform.sh	In a cluster with both the x86 and TaiShan platforms, if a third-party JAR package not supported on the x86 and TaiShan platforms is required in the MapReduce program, you need to check whether this JAR package supports cross-platform deployment. If it does not support, resolve this problem following instructions provided in <a href="#">Cross-Platform Compatibility of JAR Packages</a> .

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. [Accessing FusionInsight Manager of an MRS Cluster](#). In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** and obtain the cluster configuration file. The configuration file will be imported to the configuration file directory (usually the **conf** folder) of the MapReduce sample project.
  - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Yarn\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:  

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Yarn/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```
  - c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

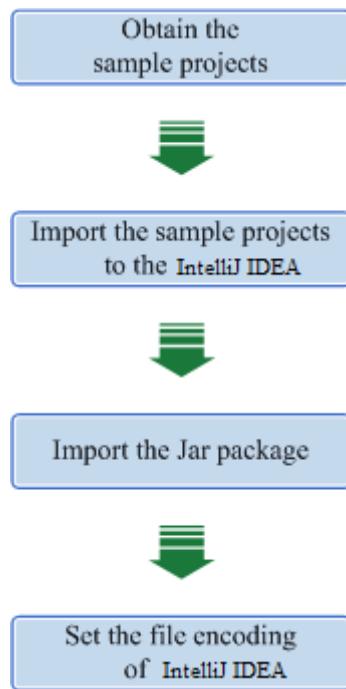
### 2.12.2.2 Configuring and Importing Sample Projects

#### Scenario

MapReduce provides sample projects for multiple scenarios to help you quickly learn MapReduce projects.

The procedure of importing MapReduce example codes is described as follows: [Figure 2-212](#) shows the procedure.

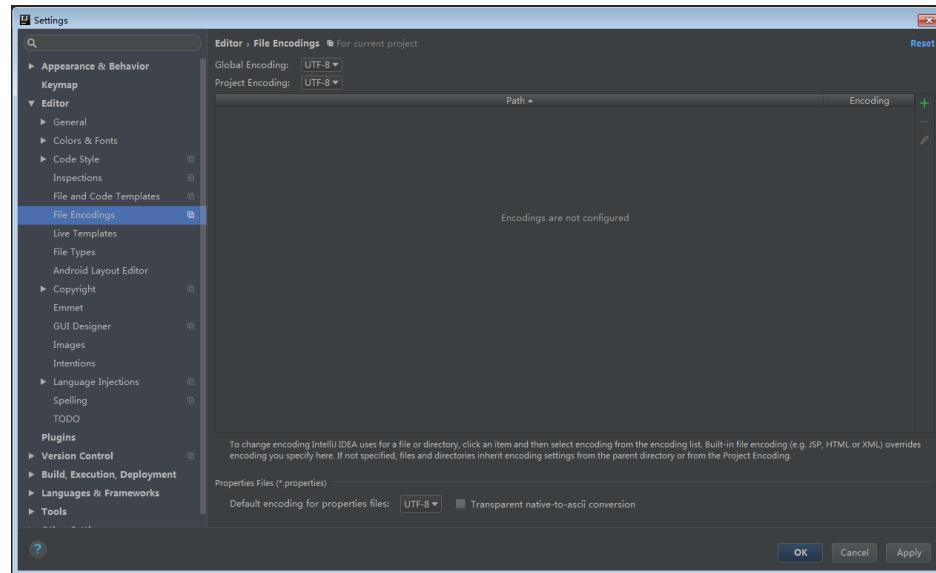
**Figure 2-212** Procedure of importing sample projects



## Procedure

- Step 1** Obtain the sample project **mapreduce-example-normal** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the example project to the IntelliJ IDEA development environment.
  1. Open IntelliJ IDEA and choose **File > Open**.
  2. Choose the directory of the example project **mapreduce-example-normal**. Click **OK**.
- Step 3** Set the IntelliJ IDEA text file coding format to prevent garbled characters.
  1. On the IntelliJ IDEA menu bar, choose **File > Settings**.  
The **Settings** window is displayed.
  2. Choose **Editor > File Encodings** from the navigation tree. In the "Global Encoding" and "Project Encodings" area, set the value to **UTF-8**, click **Apply**, and click **OK**, as shown in [Figure 2-213](#)

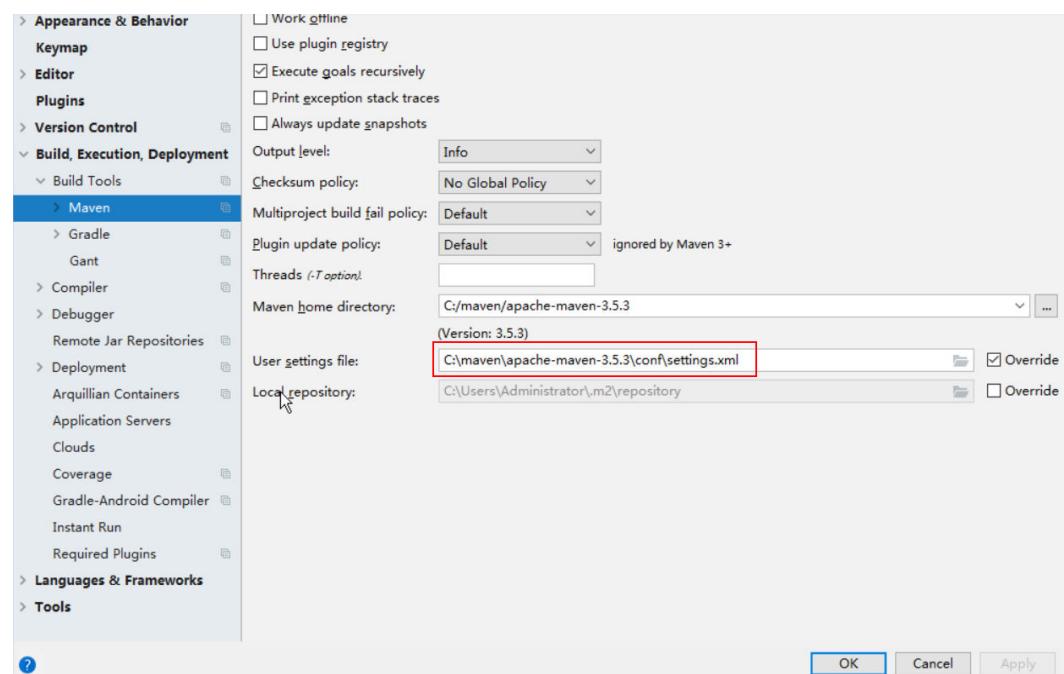
Figure 2-213 Setting the IntelliJ IDEA coding format



**Step 4** Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).

On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

Figure 2-214 Directory for storing the **settings.xml** file



----End

## Reference

The dependency packages mapped to the sample projects of MapReduce are as follows:

- MapReduce statistics sample project  
No additional jars.
- MapReduce accessing multi-components sample project

 NOTE

- If you want to use the multi-component accessing sample project after importing a sample project, ensure that the HBase service have been installed in the cluster.
- If you do not use the multi-components accessing sample project, you can ignore errors about the multi-components accessing sample project as long as the compilation of the statistics sample project is not affected. Otherwise, delete the files about the multi-components accessing sample project after importing the sample projects.

### 2.12.2.3 Creating a New Project (Optional)

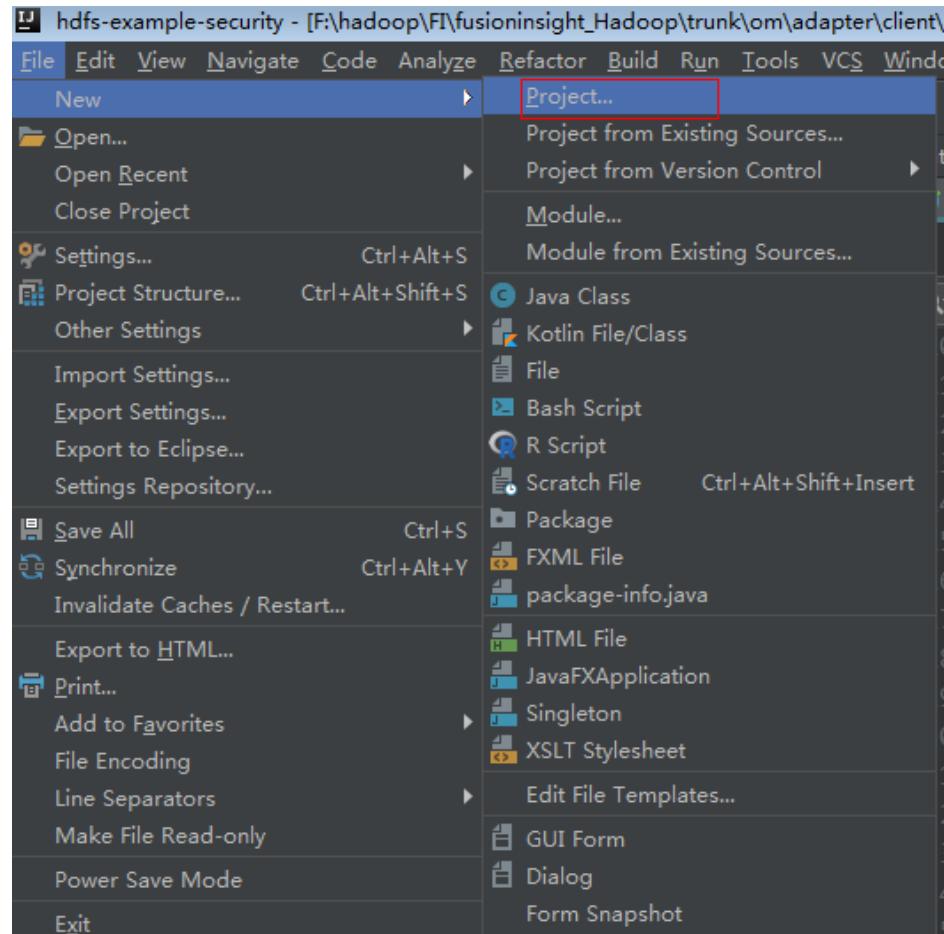
#### Scenario

Apart from importing MapReduce sample projects, you can also create a new MapReduce project using IntelliJ IDEA.

#### Procedure

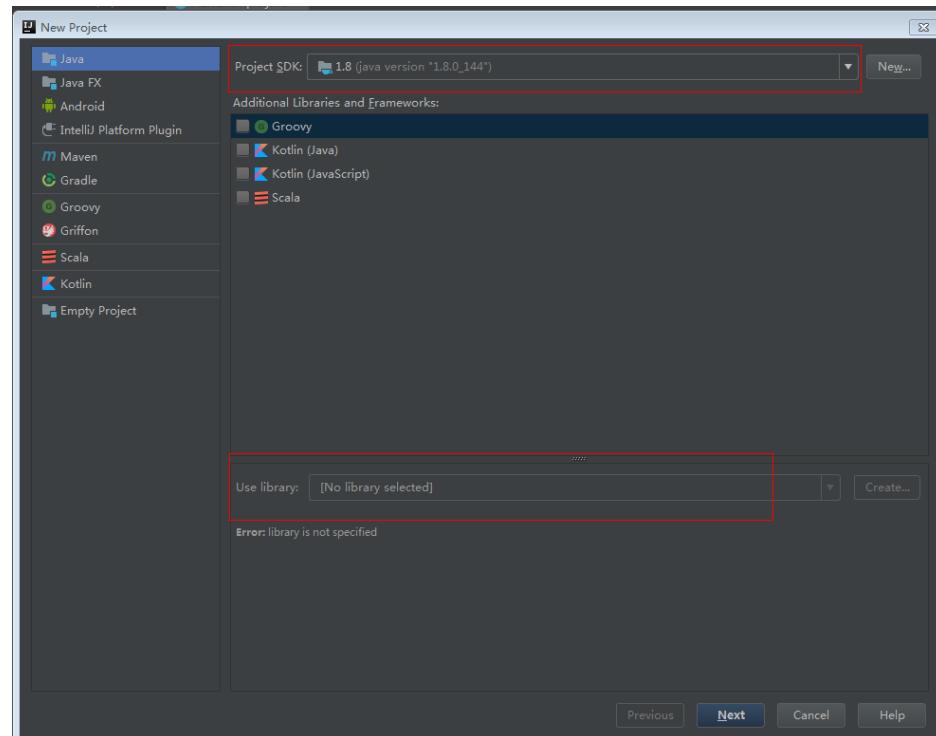
**Step 1** Open IntelliJ IDEA and choose **File > New > Project**, as shown in [Figure 2-215](#).

Figure 2-215 Creating a project



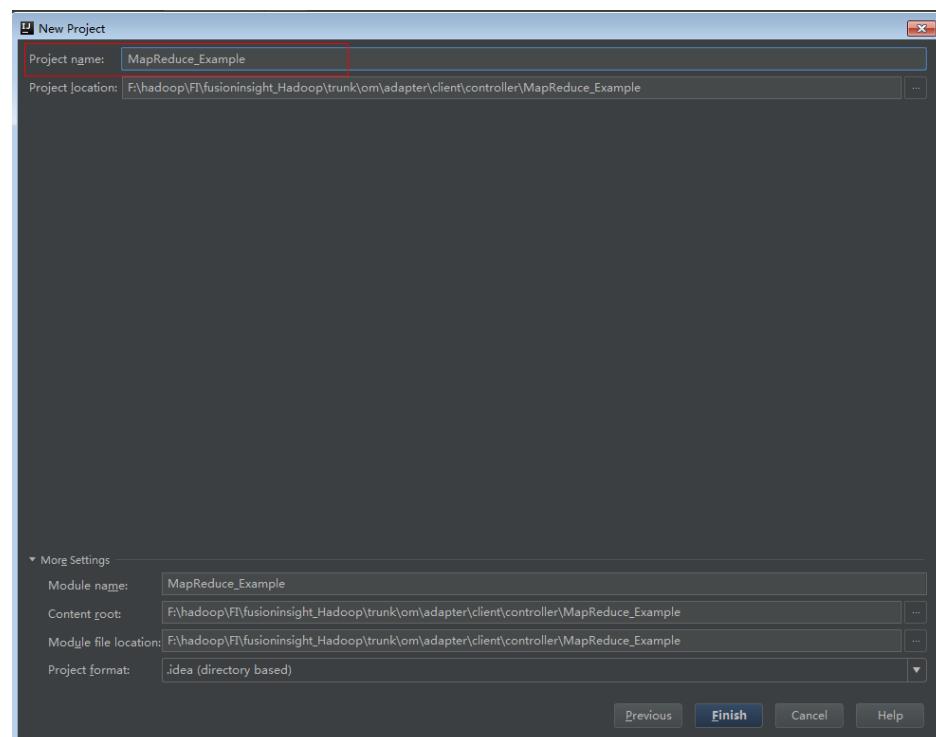
**Step 2** On the **New Project** page, select **Java**, and then configure the JDK and other Java libraries required by the project. [Figure 2-216](#) shows the SDK information required for configuring a project. After the configuration is complete, click **Next**.

**Figure 2-216** Configuring SDK Information Required by the Project



**Step 3** Enter the name of the new project in the dialog box. Click **Finish**.

**Figure 2-217** Enter the project name



----End

## 2.12.3 Developing the Project

### 2.12.3.1 MapReduce Statistics Sample Project

#### 2.12.3.1.1 Typical Scenarios

##### Scenario

Develop a MapReduce application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

**log1.txt:** logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

**log2.txt:** logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

##### Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files input\_data1.txt and input\_data2.txt on the Linux operating system, and copy log1.txt to input\_data1.txt and log2.txt to input\_data2.txt.
2. Create **/tmp/input** on the HDFS, and run the following commands to upload input\_data1.txt and input\_data2.txt to **/tmp/input**:
  - a. On the Linux client, run **hdfs dfs -mkdir /tmp/input**.

- b. On the Linux client, run **hdfs dfs -put local\_file\_path /tmp/input**.

## Development Idea

Collects the information about female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information about the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information about female netizens who spend more than 2 hours online.

### 2.12.3.1.2 Example Codes

#### Function

Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.

The operation is performed in three steps:

- Filter the online time of female netizens in original files using the CollectionMapper class inherited from the Mapper abstract class.
- Count the online time of each female netizen, and output information about female netizens who dwell online for more than 2 hours using the CollectionReducer class inherited from the Reducer abstract class.
- The main method creates a MapReduce job and submits the MapReduce job to the Hadoop cluster.

#### Example Codes

The following code snippets are used as an example. For complete codes, see the com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector class.

Example 1: The CollectionMapper class defines the map() and setup() methods of the Mapper abstract class.

```
public static class CollectionMapper extends  
    Mapper<Object, Text, Text, IntWritable> {  
  
    // Delimiter.  
    String delim;  
    // Filter sex.  
    String sexFilter;  
  
    // Name.  
    private Text nameInfo = new Text();  
  
    // Output <key,value> must be serialized.  
    private IntWritable timeInfo = new IntWritable(1);  
  
    /**  
     * Distributed computing  
     *  
     * @param key Object: location offset of the source file.
```

```

* @param value Text: a row of characters in the source file.
* @param context Context: output parameter.
* @throws IOException , InterruptedException
*/
public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException
{
    String line = value.toString();

    if (line.contains(sexFilter))
    {

        // A character string that has been read.
        String name = line.substring(0, line.indexOf(delim));
        nameInfo.set(name);
        // Obtain the dwell duration.
        String time = line.substring(line.lastIndexOf(delim) + 1,
            line.length());
        timeInfo.set(Integer.parseInt(time));

        // The Map task outputs a key-value pair.
        context.write(nameInfo, timeInfo);
    }
}
/***
 * map use to init.
 *
 * @param context Context.
 */
public void setup(Context context) throws IOException,
    InterruptedException
{
    // Obtain configuration information using Context.
    delim = context.getConfiguration().get("log.delimiter", ",");
}

sexFilter = delim
    + context.getConfiguration()
        .get("log.sex.filter", "female") + delim;
}

```

Example 2: The CollectionReducer class defines the reduce() method of the Reducer abstract class.

```

public static class CollectionReducer extends
    Reducer<Text, IntWritable, Text, IntWritable>
{

    // Statistical results.
    private IntWritable result = new IntWritable();

    // Total time threshold.
    private int timeThreshold;

    /**
     * @param key Text : key after Mapper.
     * @param values Iterable : all statistical results with the same key.
     * @param context Context
     * @throws IOException , InterruptedException
     */
    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException
    {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
    }
}

```

```
// No results are output if the time is less than the threshold.  
if (sum < timeThreshold)  
{  
    return;  
}  
result.set(sum);  
  
// In the output information, key indicates netizen information, and value indicates the total online  
time of the netizen.  
context.write(key, result);  
}  
  
/*  
 * The setup() method is invoked for only once before the map() method or reduce() method.  
 */  
* @param context Context  
* @throws IOException , InterruptedException  
*/  
public void setup(Context context) throws IOException,  
    InterruptedException  
{  
  
    // Context obtains configuration information.  
    timeThreshold = context.getConfiguration().getInt(  
        "log.time.threshold", 120);  
}
```

Example 3: Use the main() method to create a job, set parameters, and submit the job to the hadoop cluster.

```
public static void main(String[] args) throws Exception {  
    // Initialize environment variables.  
    Configuration conf = new Configuration();  
  
    // Obtain input parameters.  
    String[] otherArgs = new GenericOptionsParser(conf, args)  
        .getRemainingArgs();  
    if (otherArgs.length != 2) {  
        System.err.println("Usage: collect female info <in> <out>");  
        System.exit(2);  
    }  
  
    // Initialize the job object.  
    @SuppressWarnings("deprecation")  
    Job job = Job.getInstance(conf, "Collect Female Info");  
    job.setJarByClass(FemaleInfoCollector.class);  
  
    // Set map and reduce classes to be executed, or specify the map and reduce classes using configuration  
    // files.  
    job.setMapperClass(CollectionMapper.class);  
    job.setReducerClass(CollectionReducer.class);  
  
    // Set the Combiner class. The combiner class is not used by default. Classes same as the reduce class are  
    // used.  
    // Exercise caution when using the Combiner class. You can specify it using configuration files.  
    job.setCombinerClass(CollectionCombiner.class);  
  
    // Set the output type of the job.  
    job.setOutputKeyClass(Text.class);  
    job.setOutputValueClass(IntWritable.class);  
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
  
    // Submit the job to a remote environment for execution.  
    System.exit(job.waitForCompletion(true) ? 0 : 1);  
}
```

Example 4: CollectionCombiner class combines the mapped data on the map side to reduce the amount of data transmitted from map to reduce.

```
/*
 * Combiner class
 */
public static class CollectionCombiner extends
Reducer<Text, IntWritable, Text, IntWritable> {

    // Intermediate statistical results
    private IntWritable intermediateResult = new IntWritable();

    /**
     * @param key Text : key after Mapper
     * @param values Iterable : all results with the same key in this map task
     * @param context Context
     * @throws IOException , InterruptedException
     */
    public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}

intermediateResult.set(sum);

// In the output information, key indicates netizen information,
// and value indicates the total online time of the netizen in this map task.
context.write(key, intermediateResult);
}
}
}
```

## 2.12.3.2 MapReduce Accessing Multi-Component Example Project

### 2.12.3.2.1 Instance

#### Scenario

The sample project illustrates how to compile MapReduce jobs to visit multiple service components in HDFS, HBase, and Hive, helping users to understand key actions such as certificating and configuration loading.

The logic of the sample project is as follows:

The input data is HDFS text file and the input file is **log1.txt**.

```
YuanJing,male,10
GuoYijun,male,5
```

Map:

1. Obtain one row of the input data and extract the user name.
2. Query one piece of data from HBase.
3. Query one piece of data from Hive.
4. Combine the data queried from HBase and that from Hive as the output of Map as the output of Map.

Reduce:

1. Obtain the last piece of data from Map output.

2. Import the data to HBase.
3. Save the data to HDFS.

## Data Planning

1. Create an HDFS data file.
  - a. Create a text file named **data.txt** in the Linux-based HDFS and copy the content of **log1.txt** to **data.txt**.
  - b. Run the following commands to create a directory **/tmp/examples/multi-components/mapreduce/input/** and copy the **data.txt** to the directory:
    - i. **hdfs dfs -mkdir -p /tmp/examples/multi-components/mapreduce/input/**
    - ii. **hdfs dfs -put data.txt /tmp/examples/multi-components/mapreduce/input/**
2. Create a HBase table and insert data into it.
  - a. Run the **source bigdata\_env** command on a Linux-based HBase client and run the **hbase shell** command.
  - b. Run the **create 'table1', 'cf'** command in the HBase shell to create table1 with column family cf.
  - c. Run the **put 'table1', '1', 'cf:cid', '123'** command to insert data whose rowkey is 1, column name is **cid**, and data value is **123**.
  - d. Run the **quit** command to exit the table.
3. Create a Hive table and load data to it.
  - a. Run the **beeline** command on a Linux-based Hive client.
  - b. In the Hive beeline interaction window, run the **CREATE TABLE person(name STRING, gender STRING, stayTime INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as textfile;** command to create data table **person** with three fields.
  - c. In the Hive beeline interaction window, run the **LOAD DATA INPATH '/tmp/examples/multi-components/mapreduce/input/' OVERWRITE INTO TABLE person;** command to load data files to the **person** table.
  - d. Run **!q** to exit the table.
4. The data of HDFS is cleared in the preceding step. Therefore, perform **1** again.

### 2.12.3.2.2 Example Code

## Function

The functions of the sample project are as follows:

- Collect the name information from HDFS source files, query and combine data of HBase and Hive using the MultiComponentMapper class inherited from the Mapper abstract class.
- Obtain the last piece of mapped data and output to HBase and HDFS, using the MultiComponentMapper class inherited from the Reducer abstract class.
- The main function creates a MapReduce job and submits the MapReduce job to Hadoop clusters.

## Example Code

For details about code, see the class  
`com.huawei.bigdata.mapreduce.examples.MultiComponentExample`.

Example code of the map function used by `MultiComponentMapper` class to define the Mapper abstract class.

```
private static class MultiComponentMapper extends Mapper<Object, Text, Text, Text> {  
  
    Configuration conf;  
  
    @Override protected void map(Object key, Text value, Context context) throws IOException,  
    InterruptedException {  
  
        conf = context.getConfiguration();  
  
        String name = "";  
        String line = value.toString();  
        if (line.contains("male")) {  
            // A character string that has been read  
            name = line.substring(0, line.indexOf(", "));  
        }  
        // 1. read from HBase  
        String hbaseData = readHBase();  
  
        // 2. read from Hive  
        String hiveData = readHive(name);  
  
        // The Map task outputs a key-value pair.  
        context.write(new Text(name), new Text("hbase:" + hbaseData + ", hive:" + hiveData));  
    }  
}
```

Example code of the `readHBase` function.

```
private String readHBase() {  
    String tableName = "table1";  
    String columnFamily = "cf";  
    String hbaseKey = "1";  
    String hbaseValue;  
  
    Configuration hbaseConfig = HBaseConfiguration.create(conf);  
    org.apache.hadoop.hbase.client.Connection conn = null;  
    try {  
        // Create a HBase connection  
        conn = ConnectionFactory.createConnection(hbaseConfig);  
        // get table  
        Table table = conn.getTable(tableName);  
        // Instantiate a Get object.  
        Get get = new Get(hbaseKey.getBytes());  
        // Submit a get request.  
        Result result = table.get(get);  
        hbaseValue = Bytes.toString(result.getValue(columnFamily.getBytes(), "cid".getBytes()));  
  
        return hbaseValue;  
    } catch (IOException e) {  
        LOG.warn("Exception occur ", e);  
    } finally {  
        if (conn != null) {  
            try {  
                conn.close();  
            } catch (Exception e1) {  
                LOG.error("Failed to close the connection ", e1);  
            }  
        }  
    }  
}
```

```
    return "";
}
```

Example of the readHive function.

```
private String readHive(String name) throws IOException {
    //Load the configuration file.
    Properties clientInfo = null;
    String userdir = System.getProperty("user.dir") + "/";
    InputStream fileInputStream = null;
    try {
        clientInfo = new Properties();
        String hiveclientProp = userdir + "hiveclient.properties";
        File propertiesFile = new File(hiveclientProp);
        fileInputStream = new FileInputStream(propertiesFile);
        clientInfo.load(fileInputStream);
    } catch (Exception e) {
        throw new IOException(e);
    } finally {
        if (fileInputStream != null) {
            fileInputStream.close();
        }
    }
    String zkQuorum = clientInfo.getProperty("zk.quorum");
    String zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
    String serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
    // Read this carefully:
    // MapReduce can only use Hive through JDBC.
    // Hive will submit another MapReduce Job to execute query.
    // So we run Hive in MapReduce is not recommended.
    final String driver = "org.apache.hive.jdbc.HiveDriver";

    String sql = "select name,sum(stayTime) as " + "stayTime from person where name = '" + name + "'"
    group by name";

    StringBuilder sBuilder = new StringBuilder("jdbc:hive2://").append(zkQuorum).append("/");
    sBuilder
        .append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
        .append(";");
    String url = sBuilder.toString();
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        Class.forName(driver);
        connection = DriverManager.getConnection(url, "", "");
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();

        if (resultSet.next()) {
            return resultSet.getString(1);
        }
    } catch (ClassNotFoundException e) {
        LOG.warn("Exception occur ", e);
    } catch (SQLException e) {
        LOG.warn("Exception occur ", e);
    } finally {
        if (null != resultSet) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                // handle exception
            }
        }
        if (null != statement) {
            try {
                statement.close();
            }
```

```
        } catch (SQLException e) {
            // handle exception
        }
    }
    if (null != connection) {
        try {
            connection.close();
        } catch (SQLException e) {
            // handle exception
        }
    }
}

return "";
}
```

### NOTE

Replace all the zkQuorum objects with the actual information about the ZooKeeper cluster nodes.

Example code of the reduce function used by MultiComponentReducer class to define the Reducer abstract class.

```
private static class MultiComponentReducer extends Reducer<Text, Text, Text, Text> {
    Configuration conf;

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
    InterruptedException {
        conf = context.getConfiguration();

        Text finalValue = new Text("");
        // just pick the last value as the data to save
        for (Text value : values) {
            finalValue = value;
        }

        // write data to HBase
        writeHBase(key.toString(), finalValue.toString());

        // save result to HDFS
        context.write(key, finalValue);
    }
}
```

Example of the writeHBase function.

```
private void writeHBase(String rowKey, String data) {
    String tableName = "table1";
    String columnFamily = "cf";

    Configuration hbaseConfig = HBaseConfiguration.create(conf);
    org.apache.hadoop.hbase.client.Connection conn = null;
    try {
        // Create a HBase connection
        conn = ConnectionFactory.createConnection(hbaseConfig);
        // get table
        Table table = conn.getTable(TableName.valueOf(tableName));

        // create a Put to HBase
        List<Put> list = new ArrayList<Put>();
        byte[] row = Bytes.toBytes("row" + rowKey);
        Put put = new Put(row);
        byte[] family = Bytes.toBytes(columnFamily);
        byte[] qualifier = Bytes.toBytes("value");
        byte[] value = Bytes.toBytes(data);
        put.addColumn(family, qualifier, value);
        list.add(put);
        // execute Put
    }
```

```
        table.put(list);
    } catch (IOException e) {
        LOG.warn("Exception occur ", e);
    } finally {
    if (conn != null) {
        try {
            conn.close();
        } catch (Exception e1) {
            LOG.error("Failed to close the connection ", e1);
        }
    }
}
}
```

Example code: the main() function creates a job, configures the dependency and permission, and submits the job to Hadoop clusters.

```
public static void main(String[] args) throws Exception {
    String hiveClientProperties =
MultiComponentExample.class.getClassLoader().getResource("hiveclient.properties").getPath();
    // A file that contains configuration information.
    String file = "file://" + hiveClientProperties;
    // In runtime, put the configuration information on HDFS.
    config.set("tmpfiles", file);
    // Clean up the desired directory before submitting the job.
    MultiComponentExample.cleanupBeforeRun();
    // Find dependency jars for hive.
    Class hiveDriverClass = Class.forName("org.apache.hive.jdbc.HiveDriver");
    Class thriftClass = Class.forName("org.apache.thrift.TException");
    Class serviceThriftCLIClass = Class.forName("org.apache.hive.service.rpc.thrift.TCLIService");
    Class hiveConfClass = Class.forName("org.apache.hadoop.hive.conf.HiveConf");
    Class hiveTransClass = Class.forName("org.apache.thrift.transport.HiveTSaslServerTransport");
    Class hiveMetaClass = Class.forName("org.apache.hadoop.hive.metastore.api.MetaException");
    Class hiveShimClass =
Class.forName("org.apache.hadoop.hive.metastore.security.HadoopThriftAuthBridge23");
    Class thriftCLIClass = Class.forName("org.apache.hive.service.cli.thrift.ThriftCLIService");
    Class thriftType = Class.forName("org.apache.hadoop.hive.serde2.thrift.Type");
    // Add dependency jars to Job

    JarFinderUtil.addDependencyJars(config, hiveDriverClass, serviceThriftCLIClass, thriftCLIClass, thriftClass,
        hiveConfClass, hiveTransClass, hiveMetaClass, hiveShimClass, thriftType);
    // Add hive config file.
    config.addResource("hive-site.xml");
    // Add hbase config file
    Configuration conf = HBaseConfiguration.create(config);
    // Initialize the job object.
    Job job = Job.getInstance(conf);
    job.setJarByClass(MultiComponentExample.class);
    // Set mapper&reducer class
    job.setMapperClass(MultiComponentMapper.class);
    job.setReducerClass(MultiComponentReducer.class);
    // Set the path of Job input&output
    FileInputFormat.addInputPath(job, new Path(baseDir, INPUT_DIR_NAME + File.separator + "data.txt"));
    FileOutputFormat.setOutputPath(job, new Path(baseDir, OUTPUT_DIR_NAME));
    // Sets the output key type
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    // HBase use this utility class to add dependency jars of hbase to MR job
    TableMapReduceUtil.addDependencyJars(job);
    // Submit the job to a remote environment for execution.
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

## 2.12.4 Commissioning the Application

## 2.12.4.1 Commissioning the Application in the Windows Environment

### 2.12.4.1.1 Compiling and Running the Application

#### Scenario

After the code development is complete, you can run an application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.



Do not restart HDFS service while MapReduce application is in running status, otherwise the application will fail.

#### Running MapReduce Statistics Sample Project

- Step 1** Ensure that all JAR packages on which the sample project depends have been obtained.
- Step 2** In the IntelliJ IDEA development environment, select the LocalRunner.java project. Right-click the project and choose **Run MultiComponentLocalRunner.main()** from the shortcut menu to run the project. Click to run the related application project.

----End

#### Running Sample Applications About Multi-Components

- Step 1** Save the **hive-site.xml**, **hbase-site.xml**, and **hiveclient.properties** files to the **conf** directory of the project.
- Step 2** Ensure that all Hive and HBase JAR packages on which the sample project depends have been obtained.
- Step 3** In the MultiComponentLocalRunner.java code, `System.setProperty("HADOOP_USER_NAME", "root");` specifies that user **root** is used. Ensure that user **root** is used for uploading data, or change user **root** in the code to the username used for uploading data.
- Step 4** In the IntelliJ IDEA development environment, click **MultiComponentLocalRunner.java** to run the application project. You can also right-click the project and choose **Run MultiComponentLocalRunner.main()** from the shortcut menu to run the project.

----End

### 2.12.4.1.2 Checking the Commissioning Result

#### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the program running status in IntelliJ IDEA.
- Viewing MapReduce logs.
- Logging in to the MapReduce WebUI.
- Logging in to the Yarn WebUI.

 NOTE

You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

## Procedure

- **Check the running result by obtaining the MapReduce log.**

As shown below, the console outputs the application running result.

```
3614 [main] INFO org.apache.hadoop.hdfs.PeerCache - SocketCache disabled.
10159 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 2
11378 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:2
12707 [main] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job: job_1468241424339_0002
16434 [main] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application application_1468241424339_0002
16656 [main] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: http://10-120-180-170:26000/proxy/application_1468241424339_0002/
16657 [main] INFO org.apache.hadoop.mapreduce.Job - Running job: job_1468241424339_0002
31177 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1468241424339_0002 running in uber mode : false
31200 [main] INFO org.apache.hadoop.mapreduce.Job - map 0% reduce 0%
45893 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 0%
57172 [main] INFO org.apache.hadoop.mapreduce.Job - map 100% reduce 100%
58554 [main] INFO org.apache.hadoop.mapreduce.Job - Job job_1468241424339_0002 completed successfully
58908 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 49
File System Counters
FILE: Number of bytes read=75
FILE: Number of bytes written=436979
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=674
HDFS: Number of bytes written=23
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=206088
Total time spent by all reduces in occupied slots (ms)=73824
Total time spent by all map tasks (ms)=25761
Total time spent by all reduce tasks (ms)=9228
Total vcore-seconds taken by all map tasks=25761
Total vcore-seconds taken by all reduce tasks=9228
Total megabyte-seconds taken by all map tasks=105517056
Total megabyte-seconds taken by all reduce tasks=37797888
Map-Reduce Framework
Map input records=26
Map output records=16
Map output bytes=186
Map output materialized bytes=114
Input split bytes=230
Combine input records=16
Combine output records=6
Reduce input groups=3
```

```

Reduce shuffle bytes=114
Reduce input records=6
Reduce output records=2
Spilled Records=12
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=356
CPU time spent (ms)=2860
Physical memory (bytes) snapshot=1601576960
Virtual memory (bytes) snapshot=12999819264
Total committed heap usage (bytes)=2403336192
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=444
File Output Format Counters
Bytes Written=23

```

#### NOTE

In the Windows environment, the following exception occurs but does not affect services.

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

- Check the running result by using MapReduce WebUI.

Log in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Mapreduce > JobHistoryServer**. On the web page that is displayed, view the task execution status.

Figure 2-218 JobHistory Web UI



The screenshot shows the JobHistory Web UI interface. At the top, there is a navigation bar with links for Application, About Jobs, and Tools. Below the navigation bar is a table titled "Retired Jobs". The table has columns for Submit Time, Start Time, Finish Time, Job ID, Name, User, Queue, State, Maps Total, Maps Completed, Reduces Total, and Reduces Completed. There are two rows of data in the table, both of which are highlighted with blue boxes around the "State" column, indicating they are in the "SUCCEEDED" state. The first row corresponds to the job ID "job\_1498241424339\_0002" and the second row corresponds to "job\_1498241424339\_0001".

Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
2016-07-11 21:34:24 CST	2016-07-11 21:40:37 CST	2016-07-11 21:44:15 CST	job_1498241424339_0002	Collect Female	x00108747	default	SUCCEEDED	2	2	1	1
2016-07-11 21:39:05 CST	2016-07-11 21:39:57 CST	2016-07-11 21:39:57 CST	job_1498241424339_0001	Collect Female	x00239503	default	SUCCEEDED	2	2	1	1

- Check the running result by using YARN WebUI.

Log in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. On the web page that is displayed, view the task execution status.

Figure 2-219 ResourceManager Web UI

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Containers	Allocated CPU Cores	Allocated Memory MB	Progress %	Tracking UI	Logs
application_1408241424138_0002	root109747	FemaleInfo	MAPREDUCE	default	Tue Jul 12 14:42:34 +0800 2016	Tue Jul 12 14:44:04 +0800 2016	FINISHED	FINISHED	N/A	N/A	N/A	100	History	N/A
application_1408241424138_0001	root1095065	FemaleInfo	MAPREDUCE	default	Mon Jul 11 21:30:43 +0800 2016	Mon Jul 11 21:33:21 +0800 2016	FINISHED	FINISHED	N/A	N/A	N/A	100	History	N/A

- View MapReduce logs to learn application running conditions.
- MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 2.12.4.2 Commissioning the Application in the Linux Environment

### 2.12.4.2.1 Compiling and Running the Application

#### Scenario

After the code development is complete, you can run an application in the Linux development environment.

#### Procedure

- Step 1** Go to the local root directory of the project and run the following command in Windows cmd to compress the package:

```
mvn -s "{maven_setting_path}" clean package
```



#### NOTE

- In the preceding command, {maven\_setting\_path} is the path of the setting.xml file of the local maven.
- After the package is successfully packed, obtain the JAR package, for example, *MRTTest-XXX.jar*, from the **target** subdirectory in the root directory of the project. The name of the JAR package varies according to the actual package.

- Step 2** Upload **MRTTest-XXX.jar** to the Linux client, such as **/opt/hadoopclient/conf**, the same directory where the configuration files are located in.

- Step 3** Submit the MapReduce job in the Linux environment and execute the sample project.

- Run the following command for MapReduce statistics sample project to configure parameters and submit jobs:

```
yarn jar MRTTest-XXX.jar
com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector
<inputPath> <outputPath>
```

<inputPath> indicates the input path and <outputPath> indicates the output path in HDFS.

 NOTE

- Before running the `yarn jar MRTTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>` command, upload the `log1.txt` and `log2.txt` files to the `<inputPath>` directory in HDFS. See [Typical Scenarios](#).
- Before running the `yarn jar MRTTest-XXX.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector <inputPath> <outputPath>` command, ensure that the `<outputPath>` directory is deleted. Otherwise, an error will occur.
- Do not restart the HDFS service during the running of MapReduce tasks. Otherwise, the tasks may fail.
- In MapReduce accessing multi-components sample project, perform the following operations:
  - a. Obtain the `hbase-site.xml`, `hiveclient.properties` and `hive-site.xml` configuration files, create a folder for example `/opt/hadoopclient/conf`, and save the configurations files.

 NOTE

The file `hbase-site.xml` is acquired from the HBase client, `hiveclient.properties` and `hive-site.xml` are acquired from the Hive client.

- b. Add the classpath required for sample projects in the Linux environment, Example of classpath:  
`export YARN_USER_CLASSPATH=/opt/hadoopclient/conf:/opt/hadoopclient/HBase/hbase/lib/*:/opt/hadoopclient/HBase/hbase/lib/client-facing-thirdparty/*:/opt/hadoopclient/Hive/Beeline/lib/*`
- c. Submit MapReduce jobs. Run the following command to run the sample project:  
`yarn jar MRTTest-XXX.jar com.huawei.bigdata.mapreduce.examples.MultiComponentExample`

----End

### 2.12.4.2.2 Checking the Commissioning Result

#### Scenario

After a MapReduce application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the MapReduce WebUI.
- Logging in to the Yarn WebUI.
- Viewing MapReduce logs.

 NOTE

You must have permission to access WebUI. If not, contact the admin to obtain an account and password.

## Procedure

- Check the running result by using MapReduce WebUI.

Log in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Mapreduce > JobHistoryServer**. On the web page that is displayed, view the task execution status.

**Figure 2-220 JobHistory Web UI**

The screenshot shows a table titled "Retired Jobs" with the following data:

Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
job_1468241424339_0002	Collect Female	x00100747	default	SUCCEEDED	2	2	1	1
job_1468241424339_0001	Collect Female	x00200603	default	SUCCEEDED	2	2	1	1

- Check the running result by using YARN WebUI.

Log in to FusionInsight Manager as a user who has the permission to view task and choose **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. On the web page that is displayed, view the task execution status.

**Figure 2-221 ResourceManager Web UI**

The screenshot shows a table titled "All Applications" with the following data:

ID	User	Name	Type	Application Queue	Start Time	Finish Time	State	Final Status	Running Containers	Allocated CPU	Allocated Memory	Progress	Tracking UI	Logs	History
application_1468241424339_0002	x00100747	Collect Female	MAPREDUCE	default	Tue Jul 12 14:09:24 +0800 2016	Tue Jul 12 14:44:04 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	History
application_1468241424339_0001	x00200603	Collect Female	MAPREDUCE	default	Mon Jul 11 21:30:43 +0800 2016	Mon Jul 11 21:31:22 +0800 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	History

- Check the running result of the MapReduce application.

- After running the **yarn jar MRTest-XXX.jar** command in the Linux environment, you can check the running status of the application by the returned information about the command.

```
yarn jar MRTest-XXX.jar /tmp/mapred/example/input/ /tmp/root/output/1
16/07/12 17:07:16 INFO hdfs.PeerCache: SocketCache disabled.
16/07/12 17:07:17 INFO input.FileInputFormat: Total input files to process : 2
16/07/12 17:07:18 INFO mapreduce.JobSubmitter: number of splits:2
16/07/12 17:07:18 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1468241424339_0006
16/07/12 17:07:18 INFO impl.YarnClientImpl: Submitted application
application_1468241424339_0006
16/07/12 17:07:18 INFO mapreduce.Job: The url to track the job: http://10-120-180-170:26000/
proxy/application_1468241424339_0006/
16/07/12 17:07:18 INFO mapreduce.Job: Running job: job_1468241424339_0006
16/07/12 17:07:31 INFO mapreduce.Job: Job job_1468241424339_0006 running in uber mode :
false
16/07/12 17:07:31 INFO mapreduce.Job: map 0% reduce 0%
16/07/12 17:07:41 INFO mapreduce.Job: map 50% reduce 0%
16/07/12 17:07:43 INFO mapreduce.Job: map 100% reduce 0%
16/07/12 17:07:51 INFO mapreduce.Job: map 100% reduce 100%
16/07/12 17:07:51 INFO mapreduce.Job: Job job_1468241424339_0006 completed successfully
16/07/12 17:07:51 INFO mapreduce.Job: Counters: 49
```

### File System Counters

FILE: Number of bytes read=75

```
FILE: Number of bytes written=435659
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=674
HDFS: Number of bytes written=23
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2

Job Counters
Launched map tasks=2
Launched reduce tasks=1
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=144984
Total time spent by all reduces in occupied slots (ms)=56280
Total time spent by all map tasks (ms)=18123
Total time spent by all reduce tasks (ms)=7035
Total vcore-milliseconds taken by all map tasks=18123
Total vcore-milliseconds taken by all reduce tasks=7035
Total megabyte-milliseconds taken by all map tasks=74231808
Total megabyte-milliseconds taken by all reduce tasks=28815360

Map-Reduce Framework
Map input records=26
Map output records=16
Map output bytes=186
Map output materialized bytes=114
Input split bytes=230
Combine input records=16
Combine output records=6
Reduce input groups=3
Reduce shuffle bytes=114
Reduce input records=6
Reduce output records=2
Spilled Records=12
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=202
CPU time spent (ms)=2720
Physical memory (bytes) snapshot=1595645952
Virtual memory (bytes) snapshot=12967759872
Total committed heap usage (bytes)=2403860480

Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters
Bytes Read=444
File Output Format Counters
Bytes Written=23
```

- In the Linux environment, run the **yarn application -status <ApplicationID>** command to check the running result of the current application. Example:

```
yarn application -status application_1468241424339_0006
```

```
Application Report :
```

```
Application-Id : application_1468241424339_0006
Application-Name : Collect Female Info
Application-Type : MAPREDUCE
User : root
Queue : default
Start-Time : 1468314438442
Finish-Time : 1468314470080
Progress : 100%
State : FINISHED
Final-State : SUCCEEDED
```

```
Tracking-URL : http://10-120-180-170:26012/jobhistory/job/job_1468241424339_0006
RPC Port : 27100
AM Host : 10-120-169-46
Aggregate Resource Allocation : 172153 MB-seconds, 64 vcore-seconds
Log Aggregation Status : SUCCEEDED
Diagnostics : Application finished execution.
Application Node Label Expression : <Not set>
AM container Node Label Expression : <DEFAULT_PARTITION>
```

- **View MapReduce logs to learn application running conditions.**

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

## 2.12.5 More Information

### 2.12.5.1 Cross-Platform Compatibility of JAR Packages

#### Scenarios

This section describes how to configure a third-party JAR package in the MapReduce program if the package is not supported by x86 and TaiShan platforms in the cluster.

#### Procedure

- Step 1** On the Linux client, use the checkJarsCrossPlatform.sh script to check the cross-platform compatibility of all used JAR packages.

Run the following command to configure the cross-platform compatibility:

```
sh /opt/hadoopClient/checkJarsCrossPlatform.sh {path}
```



*path* is the client directory that contains all third-party JAR packages in the program.

```
sh /opt/hadoopClient/checkJarsCrossPlatformCompatibility.sh /opt/hadoopClient/lib/
4]: Succeed to check cross-platform compatibility for jars in dir.
4]: Not Compatibility For Cross-Platform Jars List is
4]: /opt/hadoopClient/lib/jansi-1.4.jar
```

- Step 2** Based on the command output in **Step 1**, download the JAR packages not supported on the x86 and TaiShan platforms.

For example, as shown in the command output in **Step 1**, the **jansi-1.4.jar** package is not supported by the x86 and TaiShan platforms and you need to download this package on both platforms.

- Step 3** Classify all used JAR packages shown in **Step 1** into following three types and compress these three types of JAR packages into .zip packages.

- public: JAR packages belonging to this type can run across platforms. Compress all JAR packages of this type to a **job-public.zip** package.
- x86: JAR packages belonging to this type can run only on the x86 platform. Compress all JAR packages of this type to a **job-x86.zip** package.
- TaiShan: JAR packages belonging to this type can run only on the TaiShan platform. Compress all JAR packages of this type to a **job-arm.zip** package.

**Step 4 Accessing FusionInsight Manager of an MRS Cluster.**

**Step 5** Choose **Cluster > Name of the desired cluster > Services > HDFS.**

**Step 6** Click **NameNode (Active)** to enter the WebUI page.

**Step 7** On the WebUI page, choose **Utilities > Browse the file system.**

**Step 8** Upload the three .zip packages **job-public.zip**, **job-x86.zip**, and **job-arm.zip** generated in **Step 3** to the **/tmp** directory of HDFS, as shown in **Figure 2-222**.

**Figure 2-222** Uploading .zip packages

Browse Directory

Browse Directory										
/tmp										
Show 25 entries										
#	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name		
1	drwx-----	hdfs	hadoop	0 B	Jan 04 14:33	0	0 B	.testHDFS		
2	drwxrwxrwx	mapred	hadoop	0 B	Jan 02 10:12	0	0 B	hadoop-yarn		
3	drwxrwx---	hive	hadoop	0 B	Jan 02 16:26	0	0 B	hive		
4	drwxrwx---	hive	hive	0 B	Jan 02 16:26	0	0 B	hive-scratch		
5	-rw-r-----	super	hadoop	126.68 KB	Jan 04 14:33	3	128 MB	job-arm.zip		
6	-rw-r-----	super	hadoop	444.18 KB	Jan 04 14:33	3	128 MB	job-public.zip		
7	-rw-r-----	super	hadoop	147.26 KB	Jan 04 14:33	3	128 MB	job-x86.zip		
8	drwxrwxrwt	yarn	hadoop	0 B	Jan 04 10:16	0	0 B	logs		
9	drwxrwxrwx	spark	hadoop	0 B	Jan 02 16:28	0	0 B	spark		
10	drwxrwxrwx	spark	hadoop	0 B	Jan 02 16:27	0	0 B	sparkhive-scratch		

**Step 9** Add the **mapreduce.job.cache.archives** to the **mapred-site.xml** file in the **/conf** directory of the program.

```
<property>
<name>mapreduce.job.cache.archives</name>
<value>hdfs://hacluster/tmp/job-public.zip#public,hdfs://hacluster/tmp/job-x86.zip#x86,hdfs://
hacluster/tmp/job-arm.zip#arm</value>
</property>
```

**Step 10** Modify the **yarn.application.classpath** of the **yarn-site.xml** file in the **/conf** directory of the program. Specifically, add **\$PWD/public/\*,\$PWD/\$CPU\_TYPE/\*** to the value and separate them with a comma (,).

```
<property>
<name>yarn.application.classpath</name>
<value>$PWD/public/*,$PWD/$CPU_TYPE/*,$HADOOP_CONF_DIR,$HADOOP_COMMON_HOME/share/
hadoop/common/*,$HADOOP_COMMON_HOME/share/hadoop/common/lib/*,$HADOOP_HDFS_HOME/
share/hadoop/hdfs/*,$HADOOP_HDFS_HOME/share/hadoop/hdfs/lib/*,$HADOOP_YARN_HOME/share/
hadoop/mapreduce/*,$HADOOP_YARN_HOME/share/hadoop/mapreduce/lib/*,$HADOOP_YARN_HOME/
share/hadoop/yarn/*,$HADOOP_YARN_HOME/share/hadoop/yarn/lib/*,$HADOOP_YARN_HOME/share/
hadoop/tools/lib/*</value>
</property>
```

**Step 11** Compile and run the program.

----End

## 2.12.5.2 Common APIs

### 2.12.5.2.1 Java API

Directly consult official website for detailed API of MapReduce: **Common Interfaces**

<http://hadoop.apache.org/docs/r3.3.1/api/index.html>

Common classes in MapReduce are as follows:

- org.apache.hadoop.mapreduce.Job: an interface for users to submit MR jobs and used to set job parameters, submit jobs, control job executions, and query job status.
- org.apache.hadoop.mapred.JobConf: configuration class of a MapReduce job and major configuration interface for users to submit jobs to Hadoop.

**Table 2-126** Common interfaces of org.apache.hadoop.mapreduce.Job

Interface	Description
Job(Configuration conf, String jobName), Job(Configuration conf)	Creates a MapReduce client for configuring job attributes and submitting the job.
setMapperClass(Class<extends Mapper> cls)	A core interface used to specify the Mapper class of a MapReduce job. The Mapper class is empty by default. You can also configure <b>mapreduce.job.map.class</b> in <b>mapred-site.xml</b> .
setReducerClass(Class<extends Reducer> cls)	A core interface used to specify the Reducer class of a MapReduce job. The Reducer class is empty by default. You can also configure <b>mapreduce.job.reduce.class</b> in <b>mapred-site.xml</b> .
setCombinerClass(Class<extends Reducer> cls)	Specifies the Combiner class of a MapReduce job. The Combiner class is empty by default. You can also configure <b>mapreduce.job.combine.class</b> in <b>mapred-site.xml</b> . The Combiner class can be used only when the input and output key and value types of the reduce task are the same.
setInputFormatClass(Class <extends InputFormat> cls)	A core interface used to specify the InputFormat class of a MapReduce job. The default InputFormat class is <b>TextInputFormat</b> . You can also configure <b>mapreduce.job.inputformat.class</b> in <b>mapred-site.xml</b> . This interface can be used to specify the InputFormat class for processing data in different formats, reading data, and splitting data into data blocks.
setJarByClass(Class<> cls)	A core interface used to specify the local location of the JAR package of a class. Java locates the JAR package based on the class file and uploads the JAR package to the Hadoop distributed file system (HDFS).

Interface	Description
setJar(String jar)	Specifies the local location of the JAR package of a class. You can directly set the location of a JAR package and upload the JAR package to the HDFS. Use either setJar(String jar) or setJarByClass(Class<> cls). You can also configure <b>mapreduce.job.jar</b> in <b>mapred-site.xml</b> .
setOutputFormat-Class(Class<extends OutputFormat> theClass)	A core interface used to specify the OutputFormat class of a MapReduce job. The default OutputFormat class is <b>TextOutputFormat</b> . You can also configure <b>mapred.output.format.class</b> in <b>mapred-site.xml</b> . In the default <b>TextOutputFormat</b> , each key and value are recorded in text. <b>OutputFormat</b> is not specified usually.
setOutputKeyClass(Class<> theClass)	A core interface used to specify the output key type of a MapReduce job. You can also configure <b>mapreduce.job.output.key.class</b> in <b>mapred-site.xml</b> .
setOutputValueClass(Class <> theClass)	A core interface used to specify the output value type of a MapReduce job. You can also configure <b>mapreduce.job.output.value.class</b> in <b>mapred-site.xml</b> .
setPartitionerClass(Class<extends Partitioner> theClass)	Specifies the Partitioner class of a MapReduce job. You can also configure <b>mapred.partition.class</b> in <b>mapred-site.xml</b> . This method is used to allocate Map output results to reduce classes. <b>HashPartitioner</b> is used by default, which evenly allocates the key-value pairs of a map task. For example, in HBase applications, different key-value pairs belong to different regions. In this case, you must specify the Partitioner class to allocate map output results.
setSortComparator-Class(Class<extends RawComparator> cls)	Specifies the compression class for output results of a map task. Compression is not implemented by default. You can also configure <b>mapreduce.map.output.compress</b> and <b>mapreduce.map.output.compress.codec</b> in <b>mapred-site.xml</b> . You can compress data for transmission when the map task outputs a large amount of data.
setPriority(JobPriority priority)	Specifies the priority of a MapReduce job. Five priorities can be set: <b>VERY_HIGH</b> , <b>HIGH</b> , <b>NORMAL</b> , <b>LOW</b> , and <b>VERY_LOW</b> . The default priority is <b>NORMAL</b> . You can also configure <b>mapreduce.job.priority</b> in <b>mapred-site.xml</b> .

**Table 2-127** Common interfaces of org.apache.hadoop.mapred.JobConf

Interface	Description
setNumMapTasks(int n)	A core interface used to specify the number of map tasks in a MapReduce job. You can also configure <b>mapreduce.job.maps</b> in <b>mapred-site.xml</b> . <b>NOTE</b> The InputFormat class controls the number of map tasks. Ensure that the InputFormat class supports setting the number of map tasks on the client.
setNumReduceTasks(int n)	A core interface used to specify the number of reduce tasks in a MapReduce job. Only one reduce task is started by default. You can also configure <b>mapreduce.job.reduces</b> in <b>mapred-site.xml</b> . The number of reduce tasks is controlled by users. In most cases, the number of reduce tasks is one-fourth the number of map tasks.
setQueueName(String queueName)	Specifies the queue where a MapReduce job is submitted. The default queue is used by default. You can also configure <b>mapreduce.job.queuename</b> in <b>mapred-site.xml</b> .

### 2.12.5.2.2 REST API

#### Function Description

Use the HTTP REST API to view more information about MapReduce tasks. Currently, the REST API of MapReduce can be used to query the status of completed tasks. For details about the API, see the official website:

<http://hadoop.apache.org/docs/r3.3.1/hadoop-mapreduce-client/hadoop-mapreduce-client-hs/HistoryServerRest.html>

#### Preparing the Running Environment

1. Install the client, for example, to the **/opt/hadoopclient** directory on the node. For details, see section "Installing a Client."
2. Go to the client installation directory and run the following commands to configure the environment variables:

```
source bigdata_env
```

## Procedure

Obtain detailed information about tasks that have been completed on MapReduce.

- Commands for the operation:

```
curl -k -i --negotiate -u : "http://10.120.85.2:26012/ws/v1/history/mapreduce/jobs"
```

In the preceding command, **10.120.85.2** indicates the value of **JHS\_FLOAT\_IP** for MapReduce, and **26012** indicates the port ID of the JobHistoryServer node.

### NOTE

In RedHat 6.x and CentOS 6.x, a compatibility problem occurs when the curl command is used to access the JobHistoryServer. As a result, the correct result cannot be returned.

- You can view the status information about historical tasks, such as the task IDs, start time, end time, and task execution status.
- Execution result

```
{
  "jobs": [
    "job": [
      {
        "submitTime":1525693184360,
        "startTime":1525693194840,
        "finishTime":1525693215540,
        "id":"job_1525686535456_0001",
        "name":"QuasiMonteCarlo",
        "queue":"default",
        "user":"mapred",
        "state":"SUCCEEDED",
        "mapsTotal":1,
        "mapsCompleted":1,
        "reducesTotal":1,
        "reducesCompleted":1
      }
    ]
  }
}
```

- Result analysis:

Using this API, you can query the completed MapReduce tasks in the current cluster and obtain information listed in [Table 1](#).

**Table 2-128** Common information

Parameter	Description
submitTime	Time when a task is submitted
startTime	Start time
finishTime	End time
queue	Task queue
user	User who submits the task
state	Task state, success or failure

### 2.12.5.3 FAQ

#### 2.12.5.3.1 No Response from the Client When Submitting the MapReduce Application

##### Question

After the MapReduce task is submitted to the YARN server, the client is prompted without response for a long time.

##### Answer

For the above problem, ResourceManager provides the key diagnosis information of MapReduce operating status on the WebUI. For the MapReduce application that is submitted to the YARN, users can obtain the current application status and the reason to be in the status with the diagnosis information.

Procedures:

Log in to FusionInsight Manager, and select **Cluster > Name of the desired cluster > Services > Yarn > ResourceManager(Active)**. Enter the WebUI, click the submitted MapReduce application on the WebUI of ResourceManager (Active), check the diagnosis information on the WebUI, and take measures according to the diagnosis information.

MapReduce logs offers immediate visibility into application running conditions. You can adjust application programs based on the logs.

#### 2.12.5.3.2 How to Perform Remote Debugging During MapReduce Secondary Development?

##### Question

During the secondary development of MapReduce, how to perform remote debugging?

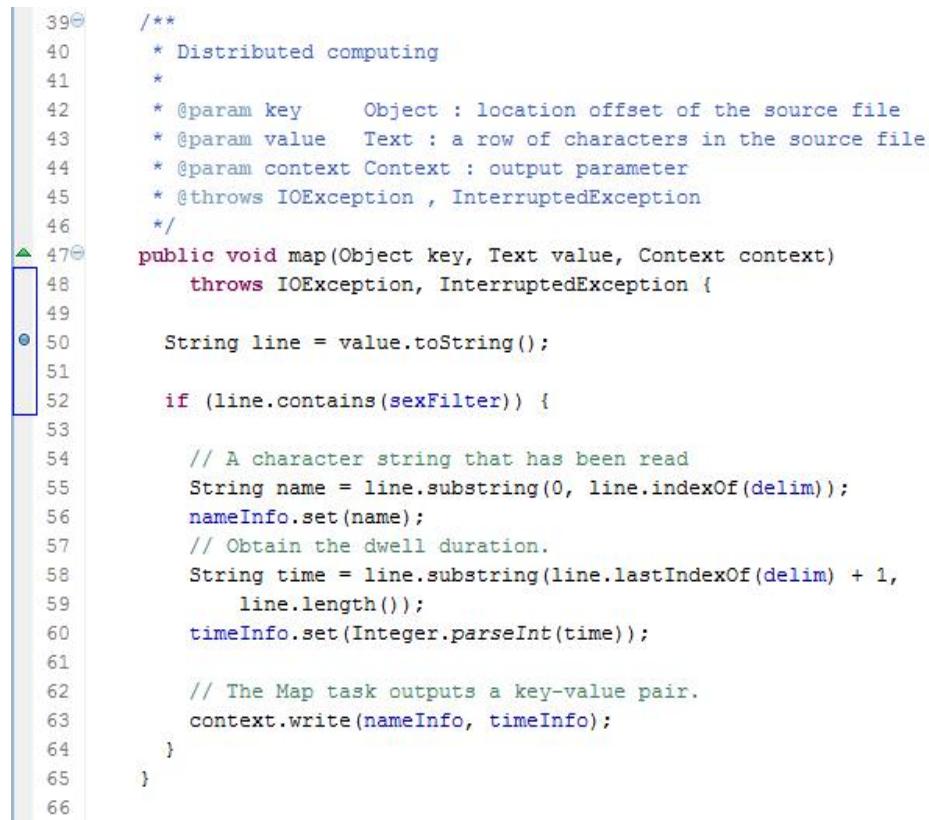
##### Answer

MapReduce adopts Java remote debugging mechanism. Run Java remote debugging commands when starting the Map or Reduce tasks.

- Step 1** The **mapreduce.map.java.opts** and **mapreduce.reduce.java.opts** parameters specify the JVM startup parameters of Map and Reduce tasks respectively. In the **mapred-site.xml** configuration file on the client, add the command **-agentlib:jdwp=transport=dt\_socket,server=y,suspend=y,address=8000** to the **mapreduce.map.java.opts** and **mapreduce.reduce.java.opts** parameters.
- Step 2** MapReduce is a distributed computing framework, and the node where Map or Reduce tasks are running are not fixed. It is advisable to keep only one NodeManager running in the cluster to ensure that the task is executed in the running NodeManager.
- Step 3** Submit MapReduce tasks on the client, then the Map or Reduce tasks will be suspended and listen to the port 8000 to wait for remote debugging.

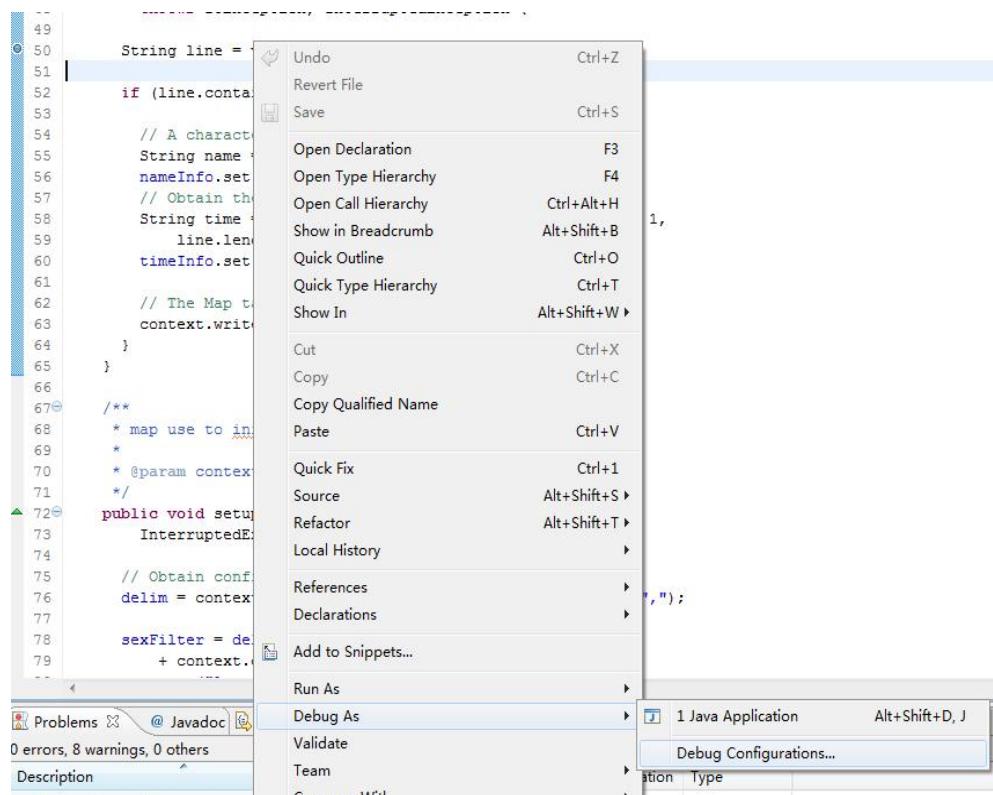
**Step 4** In IDE, select the implementation class of MapReduce tasks and configure the remote debugging information to perform debugging.

1. Double-click the area in the blue box to configure or cancel the break point.

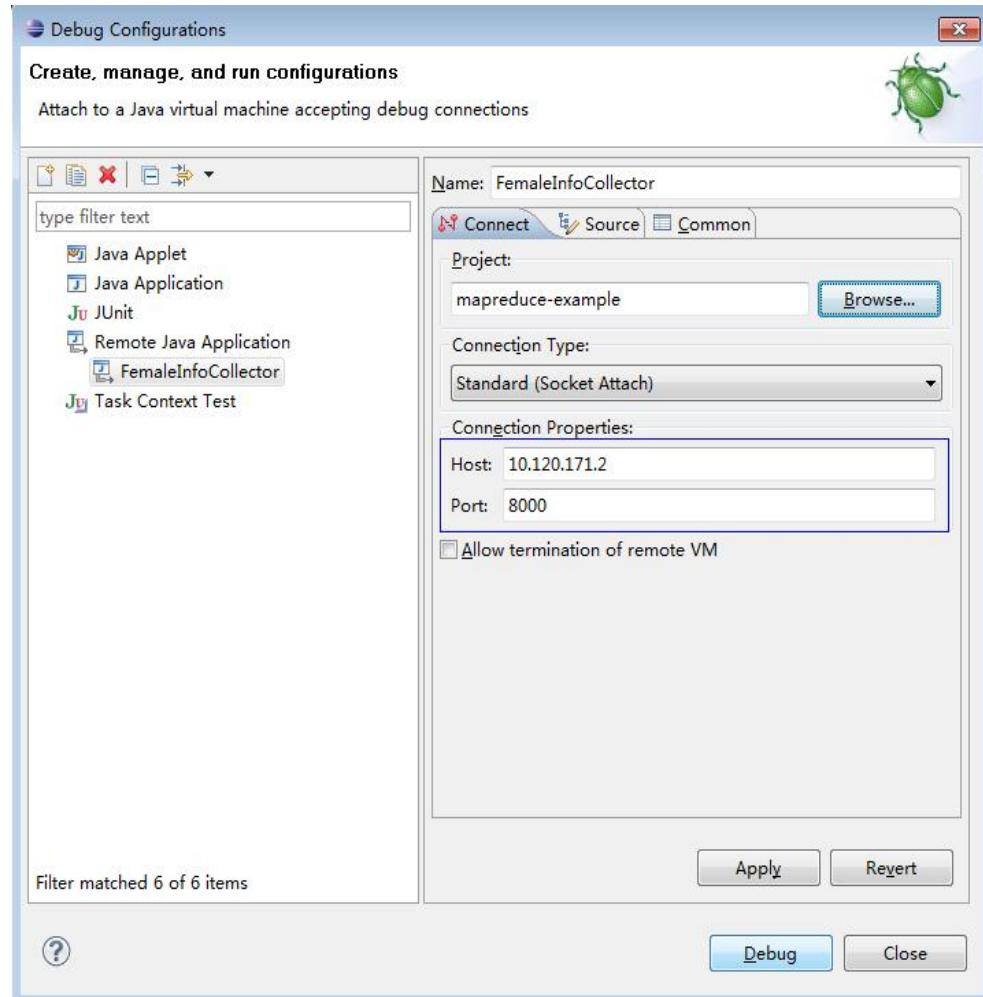


```
39    /**
40     * Distributed computing
41     *
42     * @param key Object : location offset of the source file
43     * @param value Text : a row of characters in the source file
44     * @param context Context : output parameter
45     * @throws IOException , InterruptedException
46     */
47    public void map(Object key, Text value, Context context)
48        throws IOException, InterruptedException {
49
50        String line = value.toString();
51
52        if (line.contains(sexFilter)) {
53
54            // A character string that has been read
55            String name = line.substring(0, line.indexOf(delim));
56            nameInfo.set(name);
57            // Obtain the dwell duration.
58            String time = line.substring(line.lastIndexOf(delim) + 1,
59                line.length());
60            timeInfo.set(Integer.parseInt(time));
61
62            // The Map task outputs a key-value pair.
63            context.write(nameInfo, timeInfo);
64        }
65    }
66}
```

2. Right-click the break point and choose **Debug As->Debug Configurations...** from the shortcut menu.



3. On the displayed page, double-click **Remote Java Application** and configure the **Connection Properties** area. Set **Host** to the IP address of the NodeManager and **Port** to **8000**, and then click **Debug**.



----End

NOTE

If you use IDE to submit MapReduce tasks, the IDE is the client. Modify the **mapred-site.xml** file of the secondary development project by referring to [Step 1](#).

## 2.13 Oozie Development Guide

### 2.13.1 Overview

#### 2.13.1.1 Application Development Overview

##### Oozie Introduction

Oozie is a workflow engine used to manage Hadoop jobs. Oozie workflows are defined and described based on the directed acyclic graph (DAG). Oozie supports multiple types of workflow modes and workflow scheduled triggering mechanisms. Oozie features easy extensibility, convenient maintenance, and high reliability and works closely with each component in the Hadoop ecosystem.

Oozie workflows are classified into three types:

- **Workflow**  
Provides a complete basic service workflow.
- **Coordinator**  
Built on workflows, triggers workflows in a scheduled manner or based on conditions.
- **Bundle**  
Built on coordinators, centrally schedules, controls, and manages coordinators.

Oozie provides the following features:

- Supports distribution, merging, and choice workflow modes.
- Works closely with each component in the Hadoop ecosystem.
- Supports parameterized workflow variables.
- Supports scheduled workflow triggering.
- Provides a web console that allows users to view and monitor workflows and view logs.

### 2.13.1.2 Common Concepts

- **Workflow definition file**

Workflow definition files refer to XML files that describe service logic, including **workflow.xml**, **coordinator.xml**, and **bundle.xml**. Workflow definition files are parsed and executed by the Oozie engine.

- **Workflow property file**

The workflow property file refers to the parameter configuration file named **job.properties** for workflow execution. Each workflow has only one property file.

- **Client**

Users can access the server from the client through the Java API, Shell API, REST API, or WebUI to access the Oozie server. The client in this document includes the example code used for accessing Oozie through the Java API.

- **Oozie WebUI**

Log in to the Oozie WebUI via <https://Oozie server IP address:21003/oozie>.

### 2.13.1.3 Development Process

This document describes Oozie application development based on the Java API.

For information about each phase in the development process, see [Figure 2-223](#) and [Table 2-129](#).

Figure 2-223 Oozie application development process

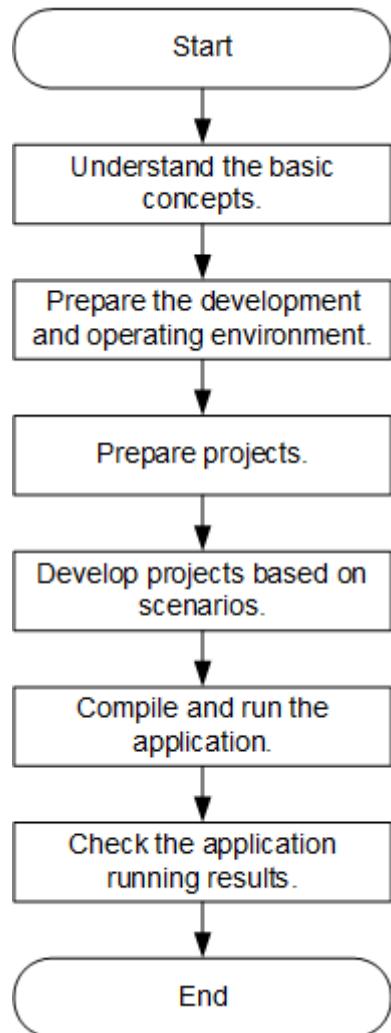


Table 2-129 Description of Oozie development process

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Oozie and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of Oozie applications. The IDEA tool can be used.	<a href="#">Preparing for Development and Operating Environment</a>
Prepare projects.	Oozie provides example projects for different scenarios. You can import an example project to learn the application.	<a href="#">Downloading and Importing Sample Projects</a>
Develop projects based on scenarios.	An example project based on the Java language is provided.	<a href="#">Developing the Project</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Commissioning the Application</a>
View application running results.	Application running results are written into a specified path. You can also view the application running status on the UI.	<a href="#">Commissioning the Application</a>

## 2.13.2 Environment Preparation

### 2.13.2.1 Preparing for Development and Operating Environment

[Table 2-130](#) describes the environment required for secondary development.

**Table 2-130** Development environment

Item	Description
OS	Windows OS. Windows 7 or later is supported. If the program needs to be commissioned locally, the development and running environment must be able to communicate with the cluster service plane network.
Installing JDK	Basic configuration of the development and running environment. The version requirements are as follows: The server and client support only the built-in OpenJDK. Other JDKs cannot be used. If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows: <ul style="list-style-type: none"><li>● For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>● For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● For security purposes, the server supports only TLS V1.2 or later.</li><li>● By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>● For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IDEA installation and configuration	configuration for the development environment. The version must be 2019.1 or other compatible version. <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>● If the IBM JDK is used, ensure that the JDK configured in IDEA is the IBM JDK.</li><li>● If the Oracle JDK is used, ensure that the JDK configured in IDEA is the Oracle JDK.</li><li>● If the Open JDK is used, ensure that the JDK configured in IDEA is the Open JDK.</li><li>● Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.

Item	Description
7-zip	Used to decompress .zip and .rar packages. The 7-zip 16.04 is supported.

## 2.13.2.2 Downloading and Importing Sample Projects

### Scenario

Download sample projects and import them to the IDEA on Windows for learning.

### Prerequisites

- If the active Oozie node and Solr are deployed on the same node, the Oozie IP address needs to be replaced with the host name to submit an Oozie task. Otherwise, the task fails to be submitted.
- A complete client has been installed in the Linux environment.
- The URL of a running Oozie server (any node) has been obtained. The URL is the target address to which the client submits a workflow job.

The URL format is `https://Oozie service IP address:21003/oozie`, for example, `https://10.10.10.176:21003/oozie`.

### Procedure

- Step 1** Obtain the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects from the sample project folder **oozienormal-examples** in the **src\oozie-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In an application development environment, import the sample projects to the IDEA development environment.
1. Choose **File > Open**.  
The **Browse** dialog box is displayed.
  2. Select a sample project folder, and click **OK**.
- Step 3** Modify the parameters in the sample project. For details, see [Table 2-131](#).

**Table 2-131** Parameters to be modified

File Name	Parameter	Value	Example Value
\src\main \resources \job.properties	userName	User who submits a job.	developuser

File Name	Parameter	Value	Example Value
\src\main\resources\application.properties	submit_user	User who submits a job.	developuser
	oozie_url_default	<code>https://Oozie service IP address:21003/oozie</code>	<code>https://10.10.10.176:21003/oozie</code>

**Step 4** Select the sample project to be run.

- For the **OozieMapReduceExample** sample project, go to [Step 5](#).
- For the **OozieSparkHBaseExample** and **OozieSparkHiveExample** sample projects, see [Scheduling Spark to Access HBase and Hive Using Oozie](#).

**Step 5** Use the client to upload the example files of Oozie to HDFS.

1. Log in to the node where the client is installed, and switch to the directory of the client, for example `/opt/hadoopclient`.

`cd /opt/hadoopclient`

2. Run the following command to configure environment variables:

`source bigdata_env`

3. Run the following commands to create a directory in HDFS and upload the sample project to the directory:

`hdfs dfs -mkdir /user/developuser`

`hdfs dfs -put -f /opt/hadoopclient/Oozie/oozie-client-*/*examples /user/developuser`



In the preceding command, `oozie-client-*` indicates the version number. Replace it with the actual version number.

----End

## 2.13.3 Developing the Project

### 2.13.3.1 Development of Configuration Files

#### 2.13.3.1.1 Description

#### Development Process

1. Configure the `workflow.xml` workflow configuration file (`coordinator.xml` schedules the workflow, and `bundle.xml` manages a pair of Coordinators) and `job.properties`.
2. If you want to implement codes, develop relevant jar packages, for example, Java Action. If you want to use Hive, develop SQL files.

3. Upload the configuration file and jar packages (including dependent jar packages) to the HDFS. The upload path is **oozie.wf.application.path** in **workflow.xml**.
4. The workflow can be implemented by using the following three methods. For details, see [More Information](#).
  - Shell command
  - Java API
  - Hue
5. The Oozie client provides examples for your reference, involving various Actions and how to use Coordinator and Bundle. For example, if the installation directory of the Oozie client is **/opt/hadoopclient**, the examples directory is **/opt/hadoopclient/Oozie/oozie-client-\*/examples**.

The following example shows you how to configure a configuration file by using the Mapreduce workflow and invoke the configuration file by running the Shell command.

## Description

Provides that a user needs to analyze website logs offline every day, and collect statistics on the access frequency of each module of the website. Log files are stored in the HDFS.

Jobs are submitted through templates and configuration files in the client.

### 2.13.3.1.2 Development Procedure

#### Step 1 Analyze the service.

1. Analyze and process logs using Mapreduce in the client example directory.
2. Move Mapreduce analysis results to the data analysis result directory, and set the data file access permission to **660**.
3. To analyze data every day, perform [Step 1.1](#) and [Step 1.2](#) every day.

#### Step 2 Implement the service.

1. Log in to the node where the client is located, and create the **dataLoad** directory, for example, **/opt/hadoopclient/Oozie/oozie-client-\*/examples/apps/dataLoad/**. This directory is used as a program running directory to store files that are edited subsequently.

#### NOTE

You can directly copy the content in the **map-reduce** directory of the example directory to the **dataLoad** directory and edit the content.

Replace **oozie-client-\*** in the directory with the actual version number.

2. Compile a workflow job property file **job.properties**.  
For details, see [job.properties](#).
3. Compile a workflow job using **workflow.xml**.

**Table 2-132 Actions in a Workflow**

No.	Procedure	Description
1	Define the startaction.	For details, see <a href="#">Start Action</a>
2	Define the MapReduceaction.	For details, see <a href="#">MapReduce Action</a>
3	Define the FS action.	For details, see <a href="#">FS Action</a>
4	Define the end action.	For details, see <a href="#">End Action</a>
5	Define the killaction.	For details, see <a href="#">Kill Action</a>

 **NOTE**

Dependent or newly developed JAR packages must be saved in **dataLoad/lib**.

The following provides an example workflow file:

```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">
  <start to="mr-dataLoad"/>
  <action name="mr-dataLoad">
    <map-reduce><resource-manager>${resourceManager}</resource-manager>
      <name-node>${nameNode}</name-node>
      <prepare>
        <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce"/>
      </prepare>
      <configuration>
        <property>
          <name>mapred.job.queue.name</name>
          <value>${queueName}</value>
        </property>
        <property>
          <name>mapred.mapper.class</name>
          <value>org.apache.oozie.example.SampleMapper</value>
        </property>
        <property>
          <name>mapred.reducer.class</name>
          <value>org.apache.oozie.example.SampleReducer</value>
        </property>
        <property>
          <name>mapred.map.tasks</name>
          <value>1</value>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="copyData"/>
    <error to="fail"/>
  </action>

  <action name="copyData">
    <fs>
```

```
<delete path='${nameNode}/user/oozie/${dataLoadRoot}/result'/>
<move source='${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce'
      target='${nameNode}/user/oozie/${dataLoadRoot}/result' />
<chmod path='${nameNode}/user/oozie/${dataLoadRoot}/result' permissions='-rwxrw-rw-'
dir-files='true'></chmod>
</fs>
<ok to="end"/>
<error to="fail"/>
</action>

<kill name="fail">
  <message>This workflow failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</
message>
</kill>
<end name="end"/>
</workflow-app>
```

#### 4. Compile a Coordinator job using [coordinator.xml](#).

The Coordinator job is used to analyze data every day. For details, see [coordinator.xml](#).

#### Step 3 Upload the workflow file.

1. Use or switch to the user account that is granted with rights to upload files to the HDFS.
2. Run the HDFS upload command to upload the **dataLoad** folder to a specified directory on the HDFS (user **oozie\_cli** must have the read/write permission for the directory).



The specified directory must be the same as **oozie.coord.application.path** and **workflowAppUri** defined in **job.properties**.

#### Step 4 Execute the workflow file.

Command:

```
oozie job -oozie https://oozie server hostname:port/oozie -config
job.propertiesfile path -run
```

Parameter list:

**Table 2-133** Parameters

Parameter	Description
job	Indicates that a job is to be executed.
-oozie	Indicates the (any instance) Oozie server address.
-config	Indicates the path of <b>job.properties</b> .
-run	Indicates the starts workflow.

For example:

```
oozie job -oozie https://10-1-130-10:21003/oozie -config job.properties -run
```

----End

## 2.13.3.2 Example Codes

### 2.13.3.2.1 job.properties

#### Function

**job.properties** is a workflow property file that defines external parameters used for workflow execution.

#### Parameter Description

[Table 2-134](#) describes parameters in **job.properties**.

**Table 2-134** Parameters

Parameter	Meaning
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode cluster address.
resourceManager	Indicates the Yarn ResourceManager address.
queueName	Identifies the MapReduce queue where a workflow job is executed.
dataLoadRoot	Identifies the folder where the workflow job resides.
oozie.coord.application.path	Indicates the storage path of a Coordinator job in the HDFS.
start	Indicates the time when a scheduled workflow job is started.
end	Indicates the time when a scheduled workflow job is stopped.
workflowAppUri	Indicates the storage path of a workflow job in the HDFS.

#### NOTE

You can define parameters in the key=value format based on service requirements.

#### Example Codes

```
nameNode=hdfs://hacluster  
resourceManager=10.1.130.10:26004  
queueName=QueueA  
dataLoadRoot=examples  
oozie.coord.application.path=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad
```

```
start=2013-04-02T00:00Z  
end=2014-04-02T00:00Z  
workflowAppUri=${nameNode}/user/oozie_cli/${dataLoadRoot}/apps/dataLoad
```

### 2.13.3.2.2 workflow.xml

#### Function

**workflow.xml** describes a complete service workflow. A workflow consists of a start node, an end node, and multiple action nodes.

#### Parameter Description

**Table 2-135** describes parameters in **workflow.xml**.

**Table 2-135** Parameters

Parameter	Meaning
name	Identifies a workflow file.
start	Indicates the workflow start node.
end	Indicates the workflow end node.
action	Indicates nodes (one or multiple) that are used to implement a service.

#### Example Codes

```
<workflow-app xmlns="uri:oozie:workflow:1.0" name="data_load">  
  <start to="copyData"/>  
  <action name="copyData">  
    </action>  
    <!--  
    <end name="end"/>  
</workflow-app>
```

### 2.13.3.2.3 Start Action

#### Function

The Start Action node indicates the start point of a workflow job. Each workflow job has only one Start Action node.

#### Parameter Description

**Table 2-136** describes the parameter used on the Start Action node.

**Table 2-136** Parameters

Parameter	Meaning
to	Identifies a subsequent action node.

## Example Codes

```
<start to="mr-dataLoad"/>
```

### 2.13.3.2.4 End Action

#### Function

The End Action node indicates the end point of a workflow job. Each workflow job has only one End Action node.

#### Parameter Description

[Table 2-137](#) describes the parameter used on the End Action node.

**Table 2-137** Parameters

Parameter	Meaning
name	Identifies an end action.

## Example Codes

```
<end name="end"/>
```

### 2.13.3.2.5 Kill Action

#### Function

The Kill Action node indicates the end point of a workflow job when an error occurs.

#### Parameter Description

[Table 2-138](#) describes parameters used on the Kill Action node.

**Table 2-138** Parameters

Parameter	Meaning
name	Identifies a kill action.
message	Provides error messages.
\$ {wf:errorMessage(wf:lastErrorNode())}	Indicates the internal error message function in the Oozie system.

## Example Codes

```
<kill name="fail">
<message>
    This workflow failed, error message[${wf:errorMessage(wf:lastErrorNode())}]
```

```
</message>  
</kill>
```

### 2.13.3.2.6 FS Action

## Function

The FS Action node is a Hadoop distributed file system (HDFS) operation node. You can create and delete HDFS files and folders and grant permissions for HDFS files and folders using this node.

## Parameter Description

**Table 2-139** describes parameters used on the FS Action node.

**Table 2-139** Parameters

Parameter	Meaning
name	Identifies an FS action.
delete	Deletes a specified file or folder.
move	Moves a file from the source directory to the target directory.
chmod	Modifies file or folder access rights.
path	Indicates the current file path.
source	Indicates the source file path.
target	Indicates the target file path.
permissions	Indicates permissions.

### NOTE

**\${variable name}** indicates the value defined in **job.properties**.

For example,  **\${nameNode}** indicates **hdfs://hacluster**. (See [job.properties](#).)

## Example Codes

```
<action name="copyData">  
  <fs>  
    <delete path='${nameNode}/user/oozie_cli/${dataLoadRoot}/result' />  
    <move source='${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce' target='${nameNode}/user/oozie_cli/${dataLoadRoot}/result' />  
    <chmod path='${nameNode}/user/oozie_cli/${dataLoadRoot}/reuslt' permissions='-rwxrw-rw-' dir-files='true' />  
  </fs>  
  <ok to="end" />  
  <error to="fail" />  
</action>
```

### 2.13.3.2.7 MapReduce Action

#### Function

The MapReduce Action node is used to execute a map-reduce job.

#### Parameter Description

**Table 2-140** describes parameters used on the MapReduce Action node.

**Table 2-140** Parameters

Parameter	Meaning
name	Identifies a map-reduce action.
resourceManager	Indicates the MapReduce ResourceManager address.
name-node	Indicates the Hadoop distributed file system (HDFS) NameNode address.
queueName	Identifies the MapReduce queue where a job is executed.
mapred.mapper.class	Identifies the Mapper class.
mapred.reducer.class	Identifies the Reducer class.
mapred.input.dir	Indicates the input directory of MapReduce processed data.
mapred.output.dir	Indicates the output directory of MapReduce processing results.
mapred.map.tasks	Indicates the number of map tasks.

#### NOTE

**\${variable name}** indicates the value defined in **job.properties**.

For example,  **\${nameNode}** indicates **hdfs://hacluster**. (See **job.properties**.)

#### Example Codes

Change **OOZIE\_URL\_DEFALUT** in the code example to the actual host name of any Oozie node, for example, <https://10-1-131-131:21003/oozie/>.

```
<action name="mr-dataLoad">
  <map-reduce>
    <resource-manager>${resourceManager}</resource-manager>
    <name-node>${nameNode}</name-node>
    <prepare>
      <delete path="${nameNode}/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce"/>
    </prepare>
    <configuration>
      <property>
```

```
<name>mapred.job.queue.name</name>
<value>${queueName}</value>
</property>
<property>
    <name>mapred.mapper.class</name>
    <value>org.apache.oozie.example.SampleMapper</value>
</property>
<property>
    <name>mapred.reducer.class</name>
    <value>org.apache.oozie.example.SampleReducer</value>
</property>
<property>
    <name>mapred.map.tasks</name>
    <value>1</value>
</property>
<property>
    <name>mapred.input.dir</name>
    <value>/user/oozie/${dataLoadRoot}/input-data/text</value>
</property>
<property>
    <name>mapred.output.dir</name>
    <value>/user/${wf:user()}/${dataLoadRoot}/output-data/map-reduce</value>
</property>
</configuration>
</map-reduce>
<ok to="copyData"/>
<error to="fail"/>
</action>
```

### 2.13.3.2.8 coordinator.xml

#### Function

coordinator.xml is used to periodically execute a workflow job.

#### Parameter Description

[Table 2-141](#) describes parameters in **coordinator.xml**.

**Table 2-141** Parameters

Parameter	Meaning
frequency	Indicates the workflow execution interval.
start	Indicates the time when a scheduled workflow job is started.
end	Indicates the time when a scheduled workflow job is stopped.
resourceManager	Indicates the MapReduce ResourceManager address.
queueName	Identifies the MapReduce queue where a job is executed.
nameNode	Indicates the Hadoop distributed file system (HDFS) NameNode address.

 NOTE

**`\${variable name}`** indicates the value defined in **job.properties**.

For example, **`\${nameNode}`** indicates **hdfs://hacluster**. (See [job.properties](#).)

## Example Codes

```
<coordinator-app name="cron-coord" frequency ="${coord:days(1)}" start ="${start}" end ="${end}">
  timezone="UTC" xmlns="uri:oozie:coordinator:0.2">
    <action>
      <workflow>
        <app-path>${workflowAppUri}</app-path>
        <configuration>
          <property>
            <name>resourceManager</name>
            <value>${resourceManager}</value>
          </property>
          <property>
            <name>nameNode</name>
            <value>${nameNode}</value>
          </property>
          <property>
            <name>queueName</name>
            <value>${queueName}</value>
          </property>
        </configuration>
      </workflow>
    </action>
  </coordinator-app>
```

### 2.13.3.3 Development of Java

#### 2.13.3.3.1 Description

These typical scenarios help you quickly understand the development procedure of Oozie and learn key API functions.

This example shows you how to submit a MapReduce job and query the job status by using the Java API. The sample code relates to the MapReduce job only, but the API invocation codes of other jobs are the same. The difference is that the configuration of **job.properties** in a job and that of **workflow.xml** in a workflow is different.

 NOTE

After the operations in [Downloading and Importing Sample Projects](#) are complete, you can submit MapReduce jobs and query job status through Java APIs.

#### 2.13.3.3.2 Sample Code

### Function

Oozie submits a job from the run method of **org.apache.oozie.client.OozieClient** and obtains job information from **getJobInfo**.

### Sample Code

```
public void test(String jobFilePath) {
  try {
    runJob(jobFilePath);
```

```
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }

private void runJob(String jobFilePath) throws OozieClientException, InterruptedException {

    Properties conf = getJobProperties(jobFilePath);
    String user = PropertiesCache.getInstance().getProperty("submit_user");
    conf.setProperty("user.name", user);

    // submit and start the workflow job
    String jobId = oozieClient.run(conf);

    logger.info("Workflow job submitted: {}" , jobId);

    // wait until the workflow job finishes printing the status every 10 seconds
    while (oozieClient.getJobInfo(jobId).getStatus() == WorkflowJob.Status.RUNNING) {
        logger.info("Workflow job running ... {}" , jobId);
        Thread.sleep(10 * 1000);
    }

    // print the final status of the workflow job
    logger.info("Workflow job completed ... {}" , jobId);
    logger.info(String.valueOf(oozieClient.getJobInfo(jobId)));
}

/**
 * Get job.properties File in filePath
 *
 * @param filePath file path
 * @return job.properties
 * @since 2020-09-30
 */
public Properties getJobProperties(String filePath) {
    File configFile = new File(filePath);
    if (!configFile.exists()) {
        logger.info(filePath , "{} is not exist.");
    }

    InputStream inputStream = null;

    // create a workflow job configuration
    Properties properties = oozieClient.createConfiguration();
    try {
        inputStream = new FileInputStream(filePath);
        properties.load(inputStream);

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (inputStream != null) {
            try {
                inputStream.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    return properties;
}
```

#### 2.13.3.4 Scheduling Spark to Access HBase and Hive Using Oozie

##### Prerequisites

Prerequisites in [Downloading and Importing Sample Projects](#) have been met.

## Preparing a Development Environment

- Step 1** Obtain the **OozieMapReduceExample**, **OozieSparkHBaseExample**, and **OozieSparkHiveExample** sample projects from the sample project folder **oozienormal-examples** in the **src\oozie-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

 **NOTE**

Currently, MRS Spark uses Scala 2.12.14.

- Step 2** Modify the parameters in each sample project. For details, see [Table 2-142](#).

**Table 2-142** Parameters to be modified

File Name	Parameter	Value	Example Value
src\main\resources\application.properties	submit_user	User who submits a job.	developuser
	oozie_url_default	https://Oozie service IP address:21003/oozie/	https://10.10.10.233:21003/oozie/
src\main\resources\job.properties	userName	User who submits a job.	developuser
	examplesRoot	Use the default value or change the value based on the site requirements.	myjobs
	oozie.wf.application.path	Use the default value or change the value based on the site requirements.	<p> \${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark <b>NOTICE</b> Ensure that the path is the same as the path with the &lt;jar&gt; and &lt;spark-opts&gt; tags in the src\main\resources\workflow.xml file.</p>
src\main\resources\workflow.xml	<jar> </jar>	Change <b>OoizeSparkHBase-1.0.jar</b> to the actual JAR package name.	<jar> \${nameNode}/user/\${userName}/\${examplesRoot}/apps/spark/lib/OoizeSparkHBase-1.0.jar</jar>

 NOTE

Go to the root directory of the project, for example, D:\sample\_project\src\oozie-examples\oozienormal-examples\OozieSparkHBaseExample, and run the **mvn clean package -DskipTests** command. After the operation is successful, the package is in the target directory.

**Step 3** Create the following folders on the HDFS client in the configured path:

/user/developuser/myjobs/apps/spark/lib  
/user/developuser/myjobs/apps/spark/hbase  
/user/developuser/myjobs/apps/spark/hive

**Step 4** Upload the files listed in [Table 2-143](#) to the corresponding path.

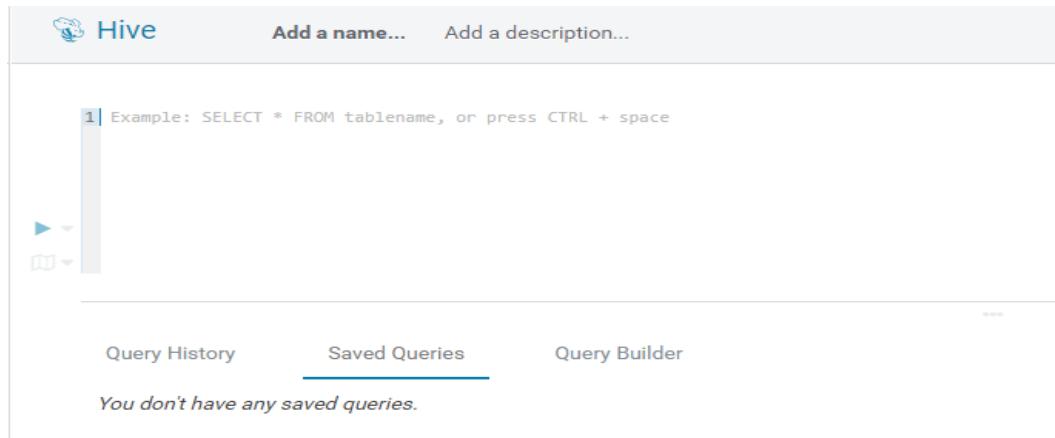
**Table 2-143** Files to be uploaded

Initial File Path	File	Destination Path
Spark client directory (for example, /opt/hadoopclient/ <b>Spark/spark/conf</b> )	hive-site.xml	<b>/user/developuser/myjobs/apps/spark</b> directory in the HDFS.
	hbase-site.xml	
Spark client directory (for example, /opt/hadoopclient/ <b>Spark/spark/jars</b> )	JAR package	Share HDFS <b>/user/oozie/share/lib/spark</b> directory of Oozie. <b>NOTE</b> This file must be uploaded as user <b>oozie</b> . Run the <b>su - oozie</b> command to switch to user <b>oozie</b> . After the file is uploaded, restart the Oozie service.
JAR package of the sample projects to be used	JAR package	<b>/user/developuser/myjobs/apps/spark/lib/</b> directory in the HDFS.
OozieSparkHiveExample sample project directory <b>src\main\resources</b>	workflow.xml	<b>/user/developuser/myjobs/apps/spark/hive</b> directory in the HDFS. <b>NOTE</b> Change the path of <b>spark-archive.zip</b> in <b>&lt;spark-opts&gt;</b> based on the actual HDFS file path.
OozieSparkHBaseExample sample project directory <b>src\main\resources</b>	workflow.xml	<b>/user/developuser/myjobs/apps/spark/hbase</b> directory in the HDFS. <b>NOTE</b> Change the path of <b>spark-archive.zip</b> in <b>&lt;spark-opts&gt;</b> based on the actual HDFS file path.

**Step 5** Change the value of `hive.security.authenticator.manager` in the `hive-site.xml` file in the `/user/developuser/myjobs/apps/spark` directory of HDFS from `org.apache.hadoop.hive.ql.security.SessionStateUserMSGGroupAuthenticator` to `org.apache.hadoop.hive.ql.security.SessionStateUserGroupAuthenticator`.

**Step 6** Run the following commands to create a Hive table:

Enter the following SQL statements in the Hive panel on the Hue UI:



```
CREATE DATABASE test;
```

```
CREATE TABLE IF NOT EXISTS `test`.`usr` (user_id int comment 'userID',user_name string comment 'userName',age int comment 'age')PARTITIONED BY (country string)STORED AS PARQUET;
```

```
CREATE TABLE IF NOT EXISTS `test`.`usr2` (user_id int comment 'userID',user_name string comment 'userName',age int comment 'age')PARTITIONED BY (country string)STORED AS PARQUET;
```

```
INSERT INTO TABLE test.usr partition(country='CN') VALUES(1,'maxwell',45),(2,'minwell',30),(3,'mike',22);
```

```
INSERT INTO TABLE test.usr partition(country='USA') VALUES(4,'minbin',35);
```

**Step 7** Use HBase Shell to run the following commands to create an HBase table:

```
create 'SparkHBase',{NAME=>'cf1'}  
put 'SparkHBase','01','cf1:name','Max'  
put 'SparkHBase','01','cf1:age','23'  
----End
```

## 2.13.4 Commissioning the Application

### 2.13.4.1 Commissioning an Application in the Windows Environment

### 2.13.4.1.1 Compiling and Running Applications

#### Scenario

After the code development is complete using Java APIs, you can run applications in the Windows development environment. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

#### Procedure

- The HTTPS SSL certificate is required for running applications in Windows.
  - a. Log in to any node in the cluster and go to the following directory to download the **ca.crt** file.

```
cd ${BIGDATA_HOME}/om-agent_8.3.1/nodeagent/security/cert/subcert/certFile/
```
  - b. Download the **ca.crt** file to a local directory and open the CLI as the administrator.

Run the following command:

```
keytool -import -v -trustcacerts -alias ca -file "D:\xx\ca.crt" -storepass JRE truststore password -keystore "%JAVA_HOME%\jre\lib\security\cacerts"
```

In the preceding command, **D:\xx\ca.crt** is the path for storing the **ca.crt** file. **%JAVA\_HOME %** indicates the JDK installation path. Obtain the JRE truststore password by following the instructions provided in [Huawei Cloud Stack 8.3.1 Account List](#).

- c. In the development environment (such as IDEA), right-click **OozieRestApiMain.java**, and choose **Run 'OozieRestApiMain.main()'** to run the application project.
- Run the following command on the Oozie client:  

```
oozie job -oozie https://Oozie service IP:21003/oozie -config job.properties -run
```

You need to copy the **job.properties** file in the **src\main\resources** directory of the sample project to the directory where the Oozie client is located in advance.

### 2.13.4.1.2 Checking the Commissioning Result

#### Scenario

The results can be viewed on the console after the Oozie sample project is completed.

#### Procedure

The following information is displayed if the sample project is successful:

```
cluset status is false
Warning: Could not get charToByteConverterClass!
Workflow job submitted: 0000067-160729120057089-oozie-omm-W
Workflow job running ...0000067-160729120057089-oozie-omm-W
```

```
Workflow job running ...0000067-160729120057089-oozie-omm-W  
Workflow job completed ...0000067-160729120057089-oozie-omm-W  
Workflow id[0000067-160729120057089-oozie-omm-W] status[SUCCEEDED]  
-----finish Oozie -----
```

Directory **/user/developuser/examples/output-data/map-reduce** is generated on the HDFS. The directory contains the following two files:

- `_SUCCESS`
- `part-00000`

You can view the files by using the file browser of the Hue or running the following commands on the HDFS:

**hdfs dfs -ls /user/developuser/examples/output-data/map-reduce**

 **NOTE**

In the Windows environment, the following exception may occur but does not affect services.

```
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
```

## 2.13.5 More Information

### 2.13.5.1 Common API Introduce

#### 2.13.5.1.1 Shell

**Table 2-144** Interface parameters

Command	Parameter	Meaning
oozie version	-	The version information
oozie job	-config <arg>	Indicates the path to the job configuration file <b>job.properties</b> .
	-oozie <arg>	Indicates the Oozie server address.
	-haconfig <arg>	Indicates the path to the Oozie service configuration file <b>oozie-site.xml</b> .
	-run	Runs a job.
	-start <arg>	Starts a specified job.
	-submit	Submits a job.

Command	Parameter	Meaning
	-kill <arg>	Deletes a specified job.
	-suspend <arg>	Suspends a specified job.
	-resume <arg>	Resumes a specified job.
	-D <property=value>	Sets a property.
oozie admin	-oozie <arg>	Indicates the Oozie server address.
	-status	Indicates the service status of the Oozie service.

The Oozie command and other parameters can be found in the following address:[https://oozie.apache.org/docs/5.1.0/DG\\_CommandLineTool.html](https://oozie.apache.org/docs/5.1.0/DG_CommandLineTool.html).

### 2.13.5.1.2 Java

Java APIs are provided by **org.apache.oozie.client.OozieClient**.

**Table 2-145 API**

Item	Description
public String run(Properties conf)	Runs a job.
public void start(String jobId)	Starts the specified job.
public String submit(Properties conf)	Submits a job.
public void kill(String jobId)	Delete the specified job.
public void suspend(String jobId)	Suspends the specified job.
public void resume(String jobId)	Resumes the specified job.
public WorkflowJob getJobInfo(String jobId)	Obtains job information.

### 2.13.5.1.3 REST

The common APIs of REST are the same as the APIs of Java. For details, see <http://oozie.apache.org/docs/5.1.0/WebServicesAPI.html>.

## 2.14 Redis Development Guide

## 2.14.1 Overview

### 2.14.1.1 Application Development Overview

#### Redis Introduction

Redis is an open-source, high-performance key-value distributed storage database. It supports a variety of data types, making up storage limitations of Memcached and meeting requirements for real-time and high-concurrency processing. In certain application scenarios, Redis is a good supplement to relational databases.

Redis is similar to Memcached. Besides, it supports data persistence and diverse data types. Redis supports union, intersection, and complement of mathematical sets on the server side and a wide range of sorting functions.

Redis applies to the following application scenarios:

- High performance.
- Low latency.
- Enriching data structure access.
- Supporting persistence.

#### Interface Type Introduction

Redis server is developed on the basis of C language. The community provides clients with common development languages and using Java language to develop Redis applications is recommended.

Redis employs the same interfaces as Jedis. For details, see <https://github.com/xetorthio/jedis>.

Redis can invoke the interfaces to provide diverse functions, as shown in **Table 2-146**.

**Table 2-146** Functions provided by Redis interfaces

Function	Description
String type access	Common operations of String in class java are: set, get, append, and strlen.
List type access	Common operations of List data structures are: lpush, rpush, lpop, rpop, llen, lindex, lrange, and lrem.
Hash type access	Common operations of Map data structures.
Set type access	Common operations of Set data structures.
Sorted Set type access	Sorted set operations

Function	Description
GEO (Geographical location)	Geographical location information can be added to the stored data

### 2.14.1.2 Common Concepts

- Client

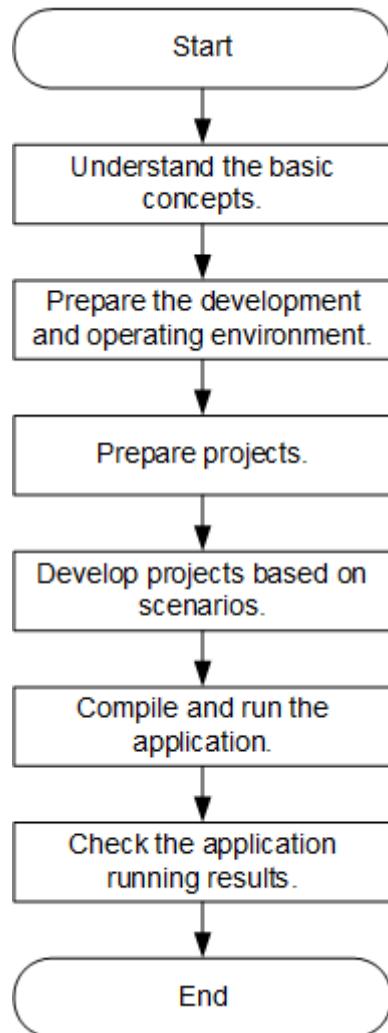
Users can access the server from the client through Java API and Redis-cli to read and write the data in Redis. The Redis client in this document includes the example code used for accessing Redis through the Java API.

### 2.14.1.3 Development Process

This document describes Redis application development based on the Java API.

For information about each phase in the development process, see [Figure 2-224](#) and [Table 2-147](#).

**Figure 2-224** Redis application development process



**Table 2-147** Redis application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Redis and understand the scenario requirements.	<a href="#">Common Concepts</a>
Prepare the development and operating environment.	The Java language is recommended for the development of Redis applications. The IntelliJ IDEA tool can be used.  The Redis running environment is the client. Install and configure the client according to the guide.	<a href="#">Development and Operating Environment</a>
Prepare projects.	Redis provides example projects for different scenarios. You can import an example project to learn the application.	<a href="#">Configuring and Importing Sample Projects</a>
Develop projects based on scenarios.	Providing the example projects of the Java language, covering Redis cluster initialization and a variety of data types access.	<a href="#">Developing an Application</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
View application running results.	Application running results will be written and output to the console.	<a href="#">Application Commissioning</a>

## 2.14.2 Environment Preparation

### 2.14.2.1 Development and Operating Environment

#### Preparing Development Environment

**Table 2-148** describes the environment required for secondary development.

**Table 2-148** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
IntelliJ IDEA installation and configuration	<p>It is the basic configuration for the development environment. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>

Item	Description
Maven installation	Basic configuration of the development environment for project management throughout the lifecycle of software development.
Redis cluster creation	<ol style="list-style-type: none"><li>1. Log in to FusionInsight Manager.</li><li>2. Choose <b>Cluster &gt; Name of the desired cluster &gt; Services &gt; Redis &gt; Redis Manager</b>.</li><li>3. Click <b>Create Redis Cluster</b>.</li><li>4. Enter a cluster name and click <b>Next</b>.</li><li>5. Select all the available Redis instances contained in the Redis cluster.</li><li>6. Click <b>Submit</b>. In the displayed <b>Create Redis Cluster</b> window, enter the administrator password and click <b>OK</b>.</li></ol> <p><b>NOTE</b> Skip these steps if such a cluster containing all the Redis instances already exists.</p>
7-zip	Used to decompress <b>.zip</b> and <b>.rar</b> packages. The 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. Log in to the FusionInsight Manager portal, in the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.  
For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.
  - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Redis\FusionInsight-redis-6.0.12.tar.gz\redis\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the Storm sample project.

**Table 2-149** describes the main configuration files.

**Table 2-149** Configuration file

Document Name	Function
redis.conf	Configures Redis parameters.
log4j.xml	Configures logs.
auth.conf	Provides authentication configuration when jedisclient is connected.
aof-backup.properties	Provides configuration for backing up Redis data.

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.
- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. Log in to FusionInsight Manager. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Redis/FusionInsight-redis-6.0.12.tar.gz/redis/config** directory to the **config** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/config**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:  
**cd /tmp/FusionInsight-Client**  
**tar -xvf FusionInsight\_Cluster\_1\_Services\_Client.tar**  
**tar -xvf FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar**  
**cd FusionInsight\_Cluster\_1\_Services\_ClientConfig**

```
scp Redis/FusionInsight-redis-6.0.12.tar.gz/redis/config/* root@IP  
address of the client node:/opt/hadoopclient/config
```

**Table 2-150 Configuration files**

File	Function
redis.conf	Configures Redis parameters.
log4j.xml	Configures logs.
auth.conf	Provides authentication configuration when jedisclient is connected.
aof-backup.properties	Provides configuration for backing up Redis data.

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 2.14.2.2 Configuring and Importing Sample Projects

### Background

The Redis development example project is included in the client installation directory. Import the project to IntelliJ IDEA for learning.

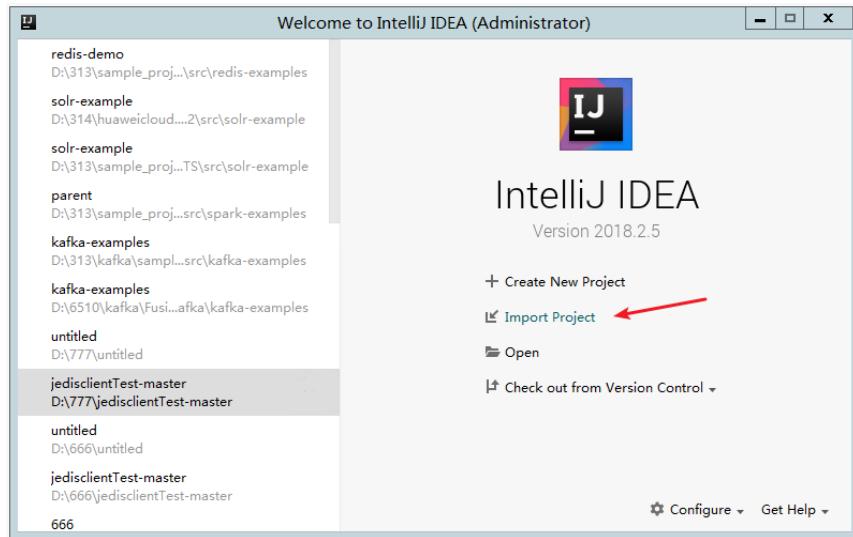
### Prerequisites

Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator.

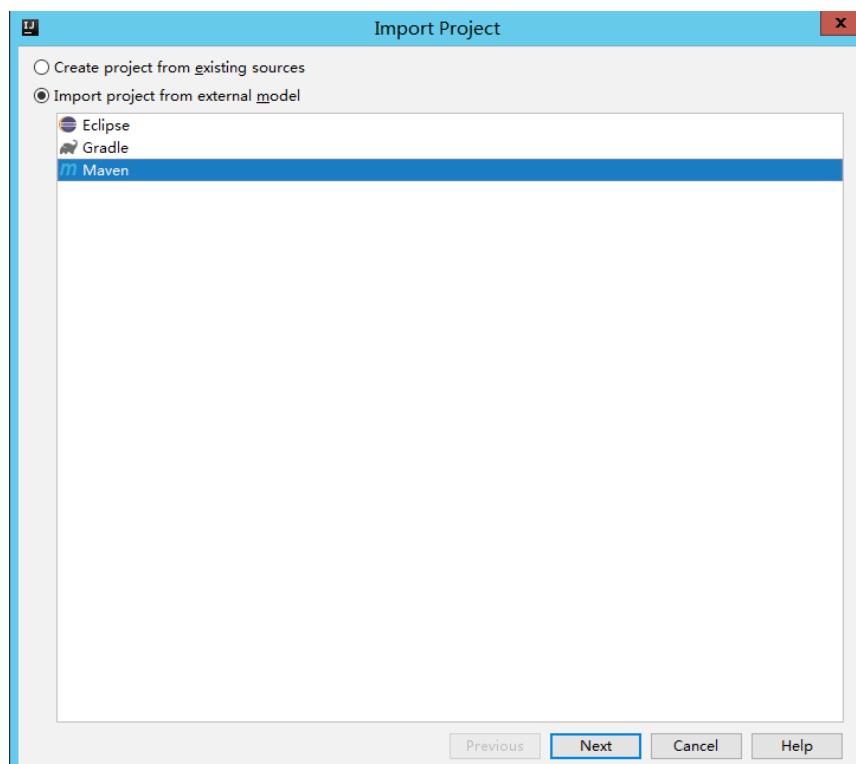
You can view the MRS cluster time in the lower-right corner on FusionInsight Manager.

### Procedure

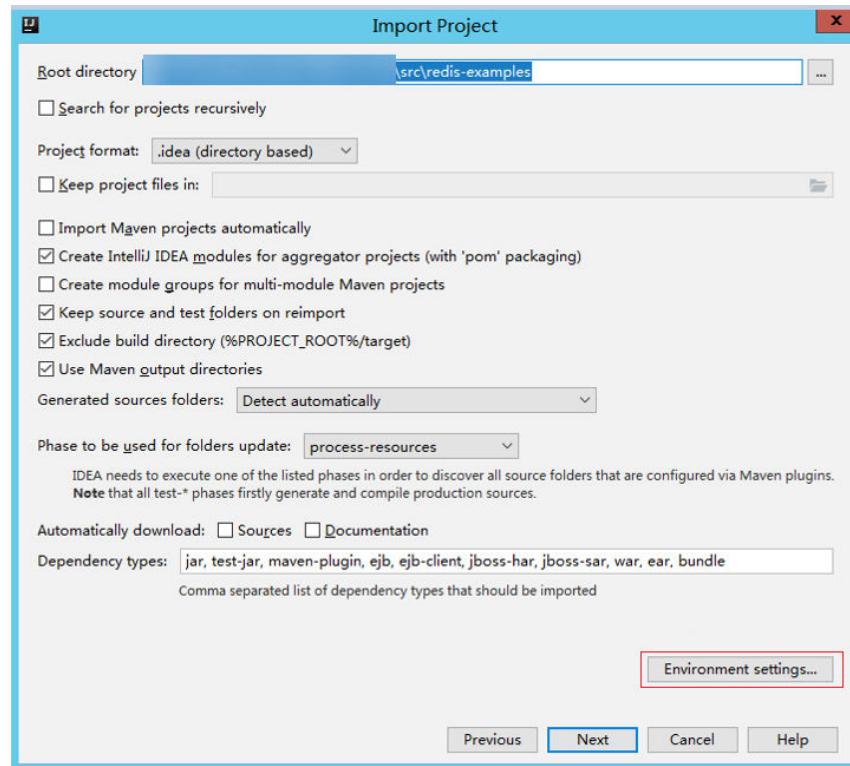
- Step 1** Obtain the sample project folder in the **/src/redis-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** Import the example project to the IntelliJ IDEA development environment.
- Start the IntelliJ IDEA, select **Import Project** on the **Quick Start** page.  
Or, for the used IDEA tool, add projects directly from the IDEA homepage. Select **File > Close Project** back to the **Quick Start** page and select **Import Project**.



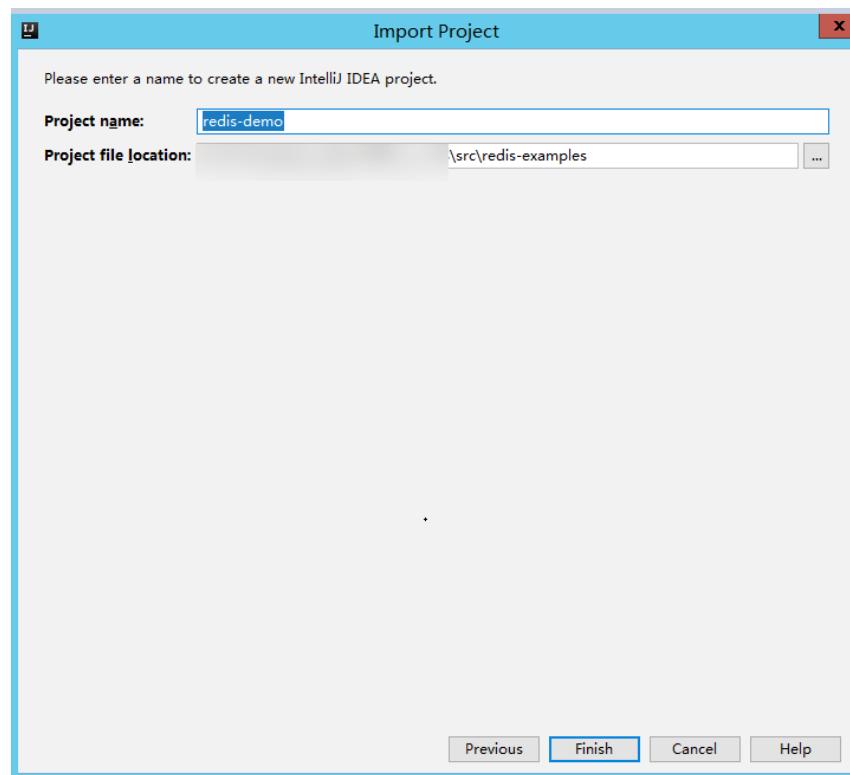
2. Select the example project folder, and click **OK**.
3. On the **Import Project** page that is displayed, select **Import project from external model** and **Maven**, and click **Next**.



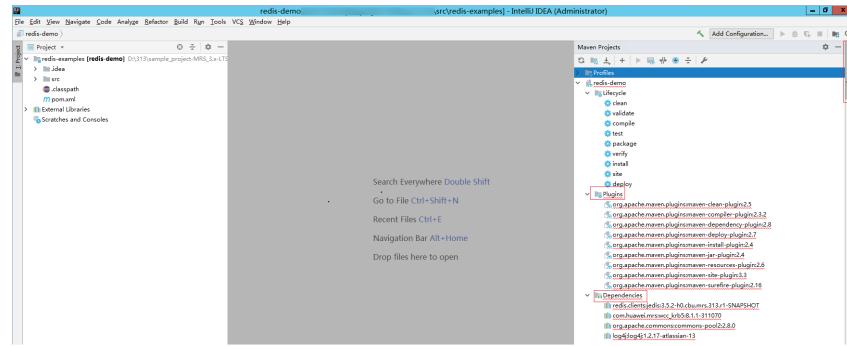
4. Click **Environment settings** to configure the environment information. Select the local Maven version. Set **User settings file** and **Local repository** based on the site requirements, click **OK**, and then click **Next**.



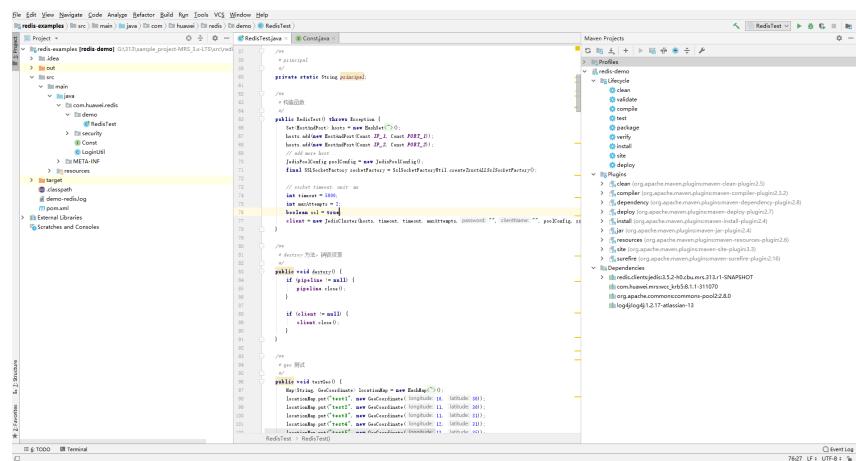
5. Confirm the subsequent configurations and click **Next**. Retain the default values unless otherwise required.
6. Enter the **project name**, select the **Project file location**, and click **Finish**.



**Step 3** In the menu bar on the right, click **Maven Project** and check whether the plug-in is loaded.



**Step 4** Click **Install** under the current project to execute the program.



----End

## 2.14.3 Developing an Application

### 2.14.3.1 Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the Redis development process and know important interface functions.

#### Scenarios

The service operation object of the Redis is key. Operations covered by example codes include access operations for keys of the String, List, Hash, Set, and Sorted Set types.

Example codes are described in the following sequence:

- Redis cluster initialization
- String type access
- List type access
- Hash type access
- Set type access
- Sorted Set type access

- Usage of Basic GEO APIs
- Use of Pipelines in the Single-thread Scenario
- Use of Pipelines in the Multi-thread Scenario
- Use of Pipelines in the bytes data Scenario

### 2.14.3.2 Example Code Description

#### 2.14.3.2.1 Redis Cluster Initialization

##### Function

Create JedisCluster instances by specifying the IP addresses and port IDs of one or more instances in a cluster.

##### Example Code

The following code snippet belongs to the **RedisTest** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public RedisTest() {  
  
    Set<HostAndPort> hosts = new HashSet<HostAndPort>();  
    hosts.add(new HostAndPort(Const.IP_1, Const.PORT_1));  
    hosts.add(new HostAndPort(Const.IP_2, Const.PORT_2));  
    // add more host...  
  
    // socket timeout(connect, read), unit: ms  
    JedisPoolConfig poolConfig = new JedisPoolConfig();  
    final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();  
    int timeout = 5000;  
    int maxAttempts = 2;  
    // Enabling channel encryption for Redis, the value of boolean SSL is changed to true  
    boolean ssl = false;  
    client = new JedisCluster(hosts, timeout, timeout, maxAttempts, "", "", poolConfig, ssl, socketFactory,  
    null, null, null);  
}
```

##### NOTE

- On FusionInsight Manager, users can view the Redis instances included in the Redis clusters.
- In the Redis cluster, the port ID corresponding to role Redis\_1 is 22400, and the port ID corresponding to role Redis\_2 is 22401. The others follow the same rule.
- Set the values, such as Const.IP\_1, Const.IP\_2, Const.PORT\_1, and Const.PORT\_2 in the **Const** class of the **com.huawei.redis** package, to the IP addresses and port IDs in the actual environment.
- Enabling channel encryption for Redis

Log in to FusionInsight Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for **REDIS\_SSL\_ON**, and set the parameter value to **true** to enable SSL channel encryption for Redis.

#### 2.14.3.2.2 String Type Access

##### Function

Implement setting and obtaining data of simple String type in the Redis, through the methods such as set and get in the JedisCluster instance.

## Example Code

The following code snippet belongs to the **testString** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testString() {
    String key = "sid-user01";

    // Save user's session ID, and set expire time
    client.setex(key, 5, "A0BC9869FBC92933255A37A1D21167B2");
    String sessionId = client.get(key);
    LOGGER.info("User " + key + ", session id: " + sessionId);
    try {
        Thread.sleep(10000);
    } catch (InterruptedException e) {
        LOGGER.warn("InterruptedException");
    }

    sessionId = client.get(key);
    LOGGER.info("User " + key + ", session id: " + sessionId);

    key = "message";

    client.set(key, "hello");
    String value = client.get(key);
    LOGGER.info("Value: " + value);

    client.append(key, " world");
    value = client.get(key);
    LOGGER.info("After append, value: " + value);

    client.del(key);
}
```

## Precautions

- All types of key in Redis are case-sensitive.
- **sessionId** is a variable name. In actual use, sensitive data may be stored in Redis. Therefore, you are not advised to directly print the data read from Redis.

### 2.14.3.2.3 List Type Access

## Function

Implement setting and obtaining data of List type in the Redis, through the methods such as rpush and lpop in the JedisCluster instance.

## Example Code

The following code snippet belongs to the **testList** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testList() {
    String key = "messages";

    // Right push
    client.rpush(key, "Hello how are you?");
    client.rpush(key, "Fine thanks. I'm having fun with redis.");
    client.rpush(key, "I should look into this NOSQL thing ASAP");

    // Fetch all data
    List<String> messages = client.lrange(key, 0, -1);
```

```
LOGGER.info("All messages: " + messages);

long len = client.llen(key);
LOGGER.info("Message count: " + len);

// Fetch the first element and delete it from list
String message = client.lpop(key);
LOGGER.info("First message: " + message);
len = client.llen(key);
LOGGER.info("After one pop, message count: " + len);

client.del(key);
}
```

#### 2.14.3.2.4 Hash Type Access

### Function

Implement setting and obtaining data of Hash type in the Redis, through the methods such as hset, hget, hmset and hgetall in the JedisCluster instance.

### Example Code

The following code snippet belongs to the **testHash** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testHash() {
    String key = "userinfo-001";

    // like Map.put()
    client.hset(key, "id", "J001");
    client.hset(key, "name", "John");
    client.hset(key, "gender", "male");
    client.hset(key, "age", "35");
    client.hset(key, "salary", "1000000");

    // like Map.get()
    String id = client.hget(key, "id");
    String name = client.hget(key, "name");
    LOGGER.info("User " + id + "'s name is " + name);

    Map<String, String> user = client.hgetAll(key);
    LOGGER.info(user);
    client.del(key);

    key = "userinfo-002";
    Map<String, String> user2 = new HashMap<String, String>();
    user2.put("id", "L002");
    user2.put("name", "Lucy");
    user2.put("gender", "female");
    user2.put("age", "25");
    user2.put("salary", "200000");
    client.hmset(key, user2);
    client.hincrBy(key, "salary", 50000);
    id = client.hget(key, "id");
    String salary = client.hget(key, "salary");
    LOGGER.info("User " + id + "'s salary is " + salary);

    // like Map.keySet()
    Set<String> keys = client.hkeys(key);
    LOGGER.info("all fields: " + keys);
    // like Map.values()
    List<String> values = client.hvals(key);
    LOGGER.info("all values: " + values);
```

```
// Fetch some fields  
values = client.hmget(key, "id", "name");  
LOGGER.info("partial field values: " + values);  
  
// like Map.containsKey();  
boolean exist = client.hexists(key, "gender");  
LOGGER.info("Exist field gender? " + exist);  
  
// like Map.remove();  
client.hdel(key, "age");  
keys = client.hkeys(key);  
LOGGER.info("after del field age, rest fields: " + keys);  
  
client.del(key);  
}
```

### 2.14.3.2.5 Set Type Access

#### Function

Implement setting and obtaining of Set type in the Redis, through the methods such as sadd and smember in the JedisCluster instance.

#### Example Code

The following code snippet belongs to the **testSet** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testSet() {  
    String key = "sets";  
  
    client.sadd(key, "HashSet");  
    client.sadd(key, "SortedSet");  
    client.sadd(key, "TreeSet");  
  
    // like Set.size()  
    long size = client.scard(key);  
    LOGGER.info("Set size: " + size);  
  
    client.sadd(key, "SortedSet");  
    size = client.scard(key);  
    LOGGER.info("Set size: " + size);  
  
    Set<String> sets = client.smembers(key);  
    LOGGER.info("Set: " + sets);  
  
    client.srem(key, "SortedSet");  
    sets = client.smembers(key);  
    LOGGER.info("Set: " + sets);  
  
    boolean ismember = client.sismember(key, "TreeSet");  
    LOGGER.info("TreeSet is set's member: " + ismember);  
  
    client.del(key);  
}
```

### 2.14.3.2.6 Sorted Set Type Access

#### Function

Implement setting and obtaining data of Sorted Set type in the Redis, through the methods such as zadd and zrange in the JedisCluster instance.

## Example Code

The following code snippet belongs to the **testSortedSet** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testSortedSet() {
    String key = "hackers";

    // Score: age
    client.zadd(key, 1940, "Alan Kay");
    client.zadd(key, 1953, "Richard Stallman");
    client.zadd(key, 1965, "Yukihiro Matsumoto");
    client.zadd(key, 1916, "Claude Shannon");
    client.zadd(key, 1969, "Linus Torvalds");
    client.zadd(key, 1912, "Alan Turing");

    // sort by score, ascending order
    Set<String> setValues = client.zrange(key, 0, -1);
    LOGGER.info("All hackers: " + setValues);

    long size = client.zcard(key);
    LOGGER.info("Size: " + size);

    Double score = client.zscore(key, "Linus Torvalds");
    LOGGER.info("Score: " + score);

    long count = client.zcount(key, 1960, 1969);
    LOGGER.info("Count: " + count);

    // sort by score, descending order
    Set<String> setValues2 = client.zrevrange(key, 0, -1);
    LOGGER.info("All hackers 2: " + setValues2);

    client.zrem(key, "Linus Torvalds");
    setValues = client.zrange(key, 0, -1);
    LOGGER.info("All hackers: " + setValues);

    client.del(key);
}
```

### 2.14.3.2.7 Usage of Basic GEO APIs

#### Function

This section describes how to use basic GEO APIs, such as the geoadd, geodist, and georadius methods of the **JedisCluster** instance.

## Example Code

The following code snippet belongs to the **testGeo** method in the **RedisTest** class of the **com.huawei.redis.demo** package.

```
public void testGeo() {
    Map<String,GeoCoordinate> locationMap = new HashMap<String, GeoCoordinate>();
    locationMap.put("test1",new GeoCoordinate(10,30));
    locationMap.put("test2",new GeoCoordinate(11,30));
    locationMap.put("test3",new GeoCoordinate(11,31));
    locationMap.put("test4",new GeoCoordinate(12,32));
    locationMap.put("test5",new GeoCoordinate(13,35));

    client.geoadd("location",locationMap);

    double dis = client.geodist("location","test1","test2", GeoUnit.KM);
    LOGGER.info("The distance between test1 and test2 is " + dis + " KM.");
}
```

```
List<GeoCoordinate> geoCoordinateList = client.geopos("location", "test1", "test2");
LOGGER.info("Geo location info is " + geoCoordinateList.toString());

List<String> hashInfo = client.geohash("location", "test1", "test2");
LOGGER.info("geohash info of test1 and test2 is " + hashInfo.toString());

List<GeoRadiusResponse> memberInfoByNumber = client.georadius("location", 11, 31, 500,
GeoUnit.KM, GeoRadiusParam.geoRadiusParam().withDist());
LOGGER.info("Get location info by longitude and latitude : ");
for(GeoRadiusResponse geoRadiusResponse : memberInfoByNumber){
    LOGGER.info(geoRadiusResponse.getMemberByString() + ":" + geoRadiusResponse.getDistance());
}

List<GeoRadiusResponse> memberInfoByLocation = client.georadiusByMember("location", "test3",
500, GeoUnit.KM, GeoRadiusParam.geoRadiusParam().sortAscending().withDist().count(3));
LOGGER.info("Get location info by member : ");
for(GeoRadiusResponse geoRadiusResponse : memberInfoByLocation){
    LOGGER.info(geoRadiusResponse.getMemberByString() + ":" + geoRadiusResponse.getDistance());
}
```

### 2.14.3.2.8 Use of Pipelines in the Single-thread Scenario

#### Function

The pipeline (pipeline) function of Redis can achieve the efficient operation in Redis, and JedisCluster can read and write cluster data.

#### Example Code

For scenarios that do not require return values and require all return values and some return values, use the corresponding interface in the sample code. The example code is applicable to single-thread scenarios.

```
public void testPipeline() {
    HostAndPort host1 = new HostAndPort(IP_1, PORT_1);
    HostAndPort host2 = new HostAndPort(IP_2, PORT_2);
    HashSet<HostAndPort> set = new HashSet<HostAndPort>();
    set.add(host1);
    set.add(host2);
    JedisCluster cluster = null;
    try{
        cluster = new JedisCluster(set);
        ClusterBatch pipeline = cluster.getPipeline();
        pipeline.set("key1", "value1");
        pipeline.set("key2", "value2");
        pipeline.set("key3", "value3");

        // none returned value
        pipeline.sync();

        // all returned value
        List<Object> syncAndReturnAll = pipeline.syncAndReturnAll();

        // part returned value
        List<Object> syncAndReturn = pipeline.syncAndReturn(new String[]{"key1"});
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        if(cluster != null){
            cluster.close();
        }
    }
}
```

### 2.14.3.2.9 Use of Pipelines in the Multi-thread Scenario

#### Function

The pipeline (pipeline) function of Redis can achieve the efficient operation in Redis, and JedisCluster can read and write cluster data.

#### Example Code

For scenarios that do not require return values and require all return values and some return values, use the corresponding interface in the sample code. The example code is applicable to multi-thread scenarios.

```
public void testPipeline() {
    public static void main(String[] args){
        HostAndPort host1 = new HostAndPort(IP_1,POR_1);
        HostAndPort host2 = new HostAndPort(IP_2,POR_2);
        HashSet<HostAndPort> set = new HashSet<HostAndPort>();
        set.add(host1);
        set.add(host2);

        myThread t1 = new myThread(set);
        myThread t2 = new myThread(set);
        myThread t3 = new myThread(set);

        t1.start();
        t2.start();
        t3.start();
    }

    public static class myThread extends Thread{
        private JedisCluster cluster;
        myThread(Set<HostAndPort> set){
            cluster = new JedisCluster(set);
        }

        public void run(){
            try{
                ClusterBatch pipeline = cluster.getPipeline();
                pipeline.set("key1","value1");
                pipeline.set("key2","value2");
                pipeline.set("key3","value3");

                // none returned value
                pipeline.sync();

                // all returned value
                List<Object> syncAndReturnAll = pipeline.syncAndReturnAll();

                //part returned value
                List<Object> syncAndReturn = pipeline.syncAndReturn(new String[]{"key1"});
            }catch (Exception e){
                e.printStackTrace();
            }finally {
                if(cluster != null){
                    cluster.close();
                }
            }
        }
    }
}
```

### 2.14.3.2.10 Use of Pipelines in the bytes data Scenario

#### Function

The pipeline function of Redis can achieve the efficient operation in Redis, and JedisCluster can read and write cluster data.

#### Example Code

For scenarios that do not require return values and require all return values and some return values, use the corresponding interface in the sample code. This example applies to scenarios where input and output parameters are bytes data.

```
public void testBinaryPipeline() {
    HostAndPort host1 = new HostAndPort(IP_1,POR_1);
    HostAndPort host2 = new HostAndPort(IP_2,POR_2);
    HashSet<HostAndPort> set = new HashSet<HostAndPort>();
    set.add(host1);
    set.add(host2);
    JedisCluster cluster = null;
    try{
        cluster = new JedisCluster(set);
        BinaryClusterBatch pipeline = cluster.getBinaryPipeline();
        pipeline.set("key1".getBytes(),"value1".getBytes());
        pipeline.set("key2".getBytes(),"value2".getBytes());
        pipeline.set("key3".getBytes(),"value3".getBytes());

        // none returned value
        pipeline.sync();

        // all returned value
        List<Object> syncAndReturnAll = pipeline.syncAndReturnAll();

        //part returned value
        byte[][] part = new byte[][]{"key1".getBytes(), "key2".getBytes()};
        List<Object> syncAndReturn = pipeline.syncAndReturn(part);
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        if(cluster != null){
            cluster.close();
        }
    }
}
```

### 2.14.3.2.11 Connection to a Redis Logical ClusterUsing Spring Boot

#### Function

Spring Boot is used to connect to a Redis logical cluster and read and write data in this cluster.

#### Example Code

The example code is used to connect to a Redis logical cluster to realize read and write in Redis.

```
@Repository
public class CustomerRedisTemplateUtil {
    protected static final Logger logger =
    LoggerFactory.getLogger(CustomerRedisTemplateUtil.class.getName());
    private static JedisCluster client;
```

```
/*
 * Default connection timeout
 */
private static final Integer TIMEOUT = 3000;
/*
 * Maximum retries
*/
private static final Integer MAX_ATTEMPTS = 1;
@Value("${spring.redis.cluster.nodes}")
private String nodes;
@Value("${redis.keytab}")
private String keytab;
@Value("${redis.user}")
private String user;
@Value("${redis.krb5}")
private String krb5;
@Value("${redis.realm}")
private String realm;
@Value("${redis.ssl}")
private boolean ssl;
@PostConstruct
private void init() throws NoSuchAlgorithmException, KeyManagementException {
    if (Objects.isNull(client)) {
        System.setProperty("redis.authentication.jaas", "true");
        AuthConfiguration authConfig = new AuthConfiguration(krb5, keytab, user);
        GlobalConfig.setAuthConfiguration(authConfig);
        authConfig.setServerRealm(realm);
        authConfig.setLocalRealm(realm);
        logger.info("user={}, realm={}, user, realm");
        //Configure the connection pool.
        JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
        jedisPoolConfig.setMaxTotal(5);
        jedisPoolConfig.setMaxIdle(3);
        jedisPoolConfig.setMinIdle(2);
        Set<HostAndPort> hosts = new HashSet<HostAndPort>();
        for (String node : nodes.split(",")) {
            HostAndPort hp = getIpAndPort(node);
            if (hp == null) {
                logger.warn("{} not valid", node);
                continue;
            }
            hosts.add(hp);
        }
        final SSLSocketFactory socketFactory = SslSocketFactoryUtil.createTrustAllSslSocketFactory();
        client = new JedisCluster(hosts, TIMEOUT, TIMEOUT, TIMEOUT, MAX_ATTEMPTS, jedisPoolConfig,
        ssl,
            socketFactory, authConfig);
    }
}
public static HostAndPort getIpAndPort(String hostAndPortStr) {
    if (StringUtils.isBlank(hostAndPortStr)) {
        return null;
    }
    String[] hostAndPort = hostAndPortStr.split(":");
    String ipAddr;
    String port;
    if (hostAndPort.length < 2) {
        return null;
    } else {
        port = hostAndPort[hostAndPort.length - 1];
        ipAddr = hostAndPortStr.substring(0, hostAndPortStr.lastIndexOf(":"));
    }
    return new HostAndPort(ipAddr, Integer.parseInt(port));
}
public void save(String key, String value) {
    client.set(key, value);
}
public String find(String id) {
    String s = client.get(id);
```

```
        System.out.println(s);
        return s;
    }
}
```

The values of **spring.redis.cluster.nodes** and **redis.realm** need to be read from the **/src/main/resources/application.properties** configuration file of the sample project. Modify the parameters based on the site requirements.

#### NOTE

- **spring.redis.cluster.nodes** indicates the IP address and port number of the logical cluster. Multiple IP addresses and ports are separated by commas (,), for example, **IP address 1:Port number 1,IP address 2:Port number 2,IP address 3:Port number 3**.
  - Obtaining the instance service IP address: Log in to FusionInsight Manager, choose **Cluster > Services > Redis**, and click **Instance** to view the IP addresses of instances running in the cluster.
  - Obtaining the port number: On Redis nodes, the port number used by role **Redis\_1** is **22400**, and that used by role **Redis\_2** is **22401**. The others follow the same rule.
- **redis.realm** indicates the current cluster domain name. You can log in to FusionInsight Manager and choose **System > Permission > Domain** and **Mutual Trust** to view the actual domain name of the cluster.
- **redis.ssl** indicates whether to enable the channel encryption. If the channel encryption is enabled, set this parameter to **true**. Otherwise, set this parameter to **false**. To check whether channel encryption has been enabled for a cluster, perform the following operations:

Log in to FusionInsight Manager, choose **Cluster > Services > Redis**. On the page that is displayed, click the **Configurations** tab then the **All Configurations** sub-tab. On this sub-tab page, search for **REDIS\_SSL\_ON**. If its value is **true**, SSL channel encryption has been enabled for Redis. Otherwise, channel encryption has been disabled for the Redis cluster.

- **server.port** indicates the port for accessing the Spring Boot server. The default value is **9093** and can be customized.

## 2.14.4 Application Commissioning

### 2.14.4.1 Commissioning an Application in Windows

#### 2.14.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment.

If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

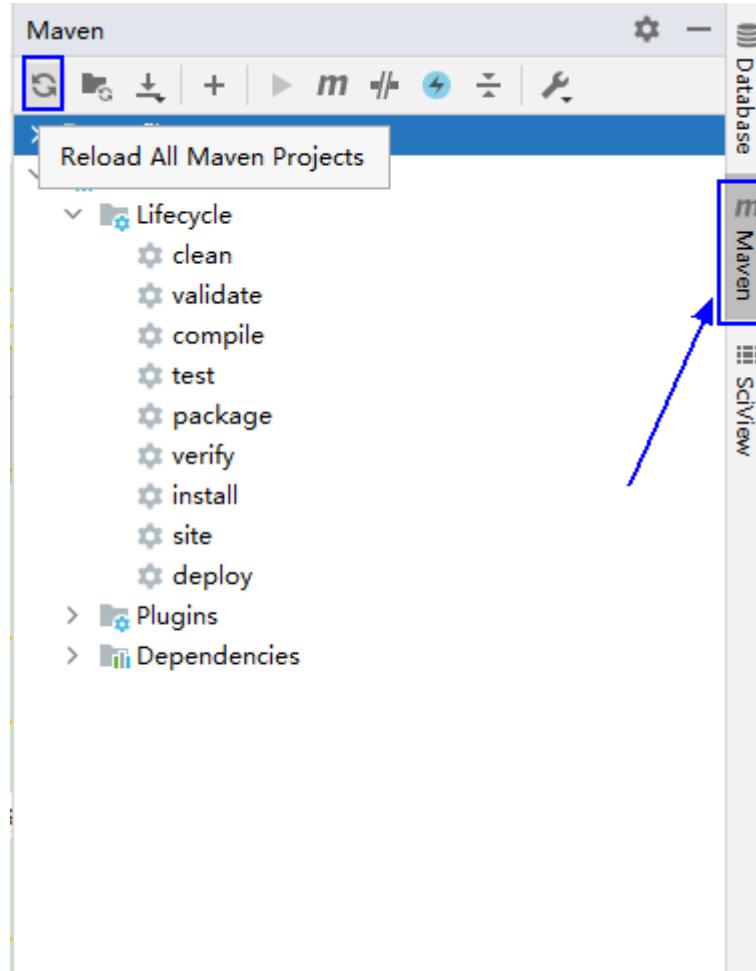
#### NOTE

- If IBM JDK is used in the Windows development environment, the application cannot be run in the Windows environment.
- You need to set the mapping between the host names and IP addresses of the access nodes in the hosts file on the local host where the sample code is run. The host names and IP addresses must be mapped one by one.

## Procedure

**Step 1** Click **Reload All Maven Projects** in the Maven window on the right of the IDEA to reload the Maven project dependency.

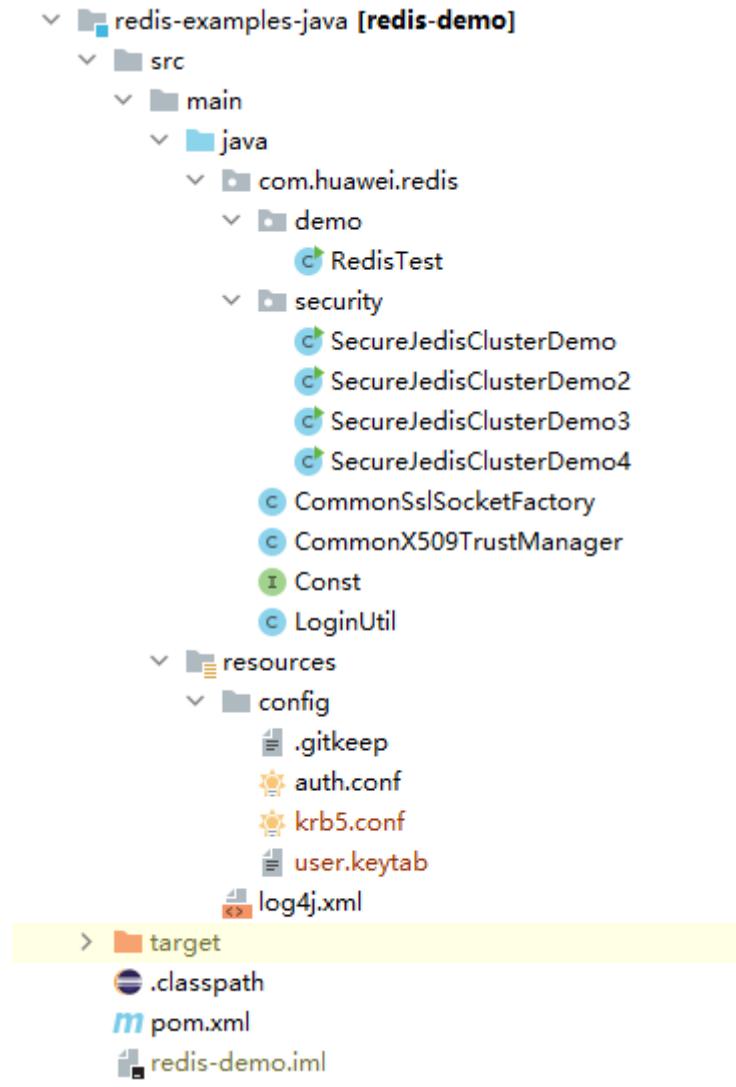
**Figure 2-225** reload projects



**Step 2** Compile and run an application.

Place the configuration file directory and modify the code to match the login user.  
See [Figure 2-226](#).

Figure 2-226 Directory list of **redis-demo** to be compiled



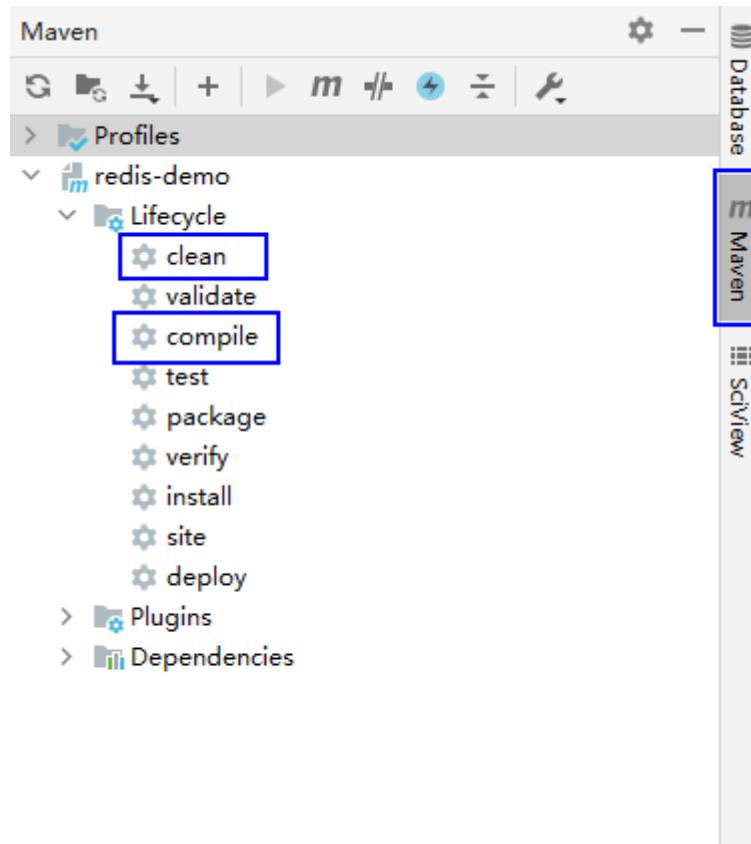
1. Compile an application.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.

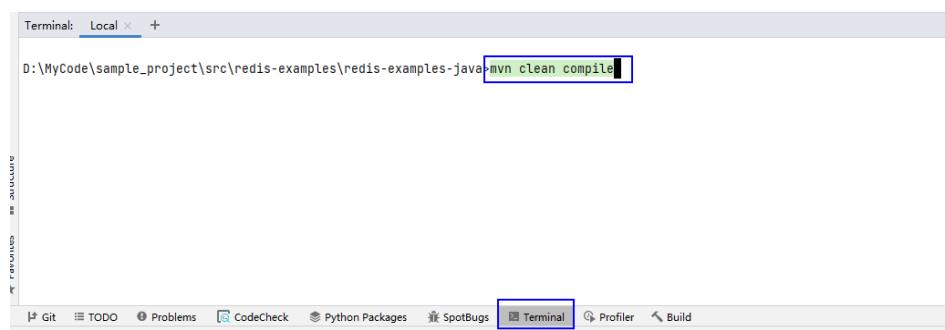
Figure 2-227 clean and compile tools of Maven



- Method 2:

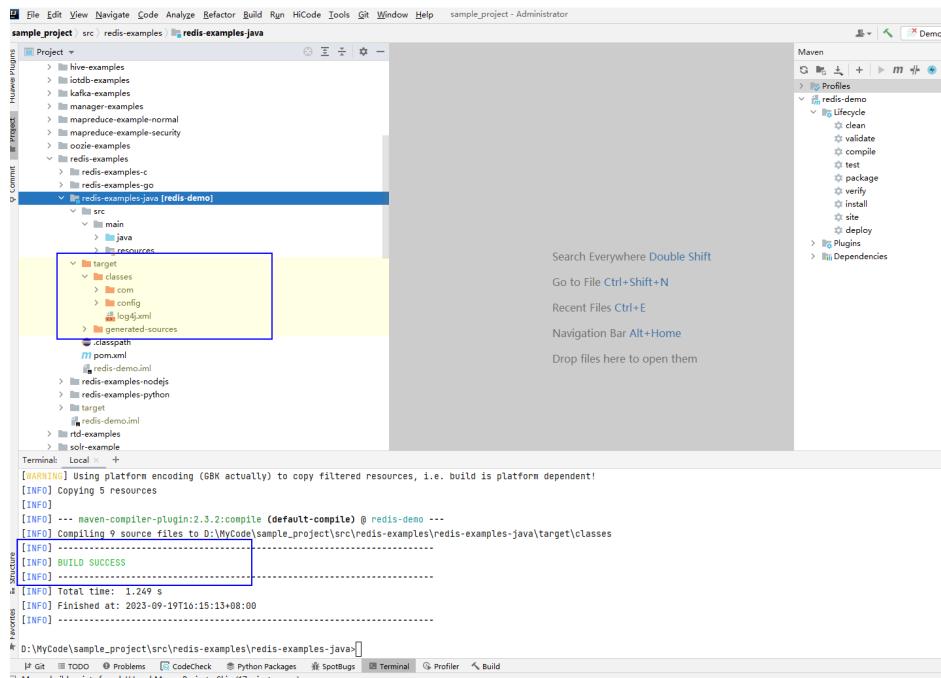
Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

Figure 2-228 Enter mvn clean compile in the IDEA Terminal text box



After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

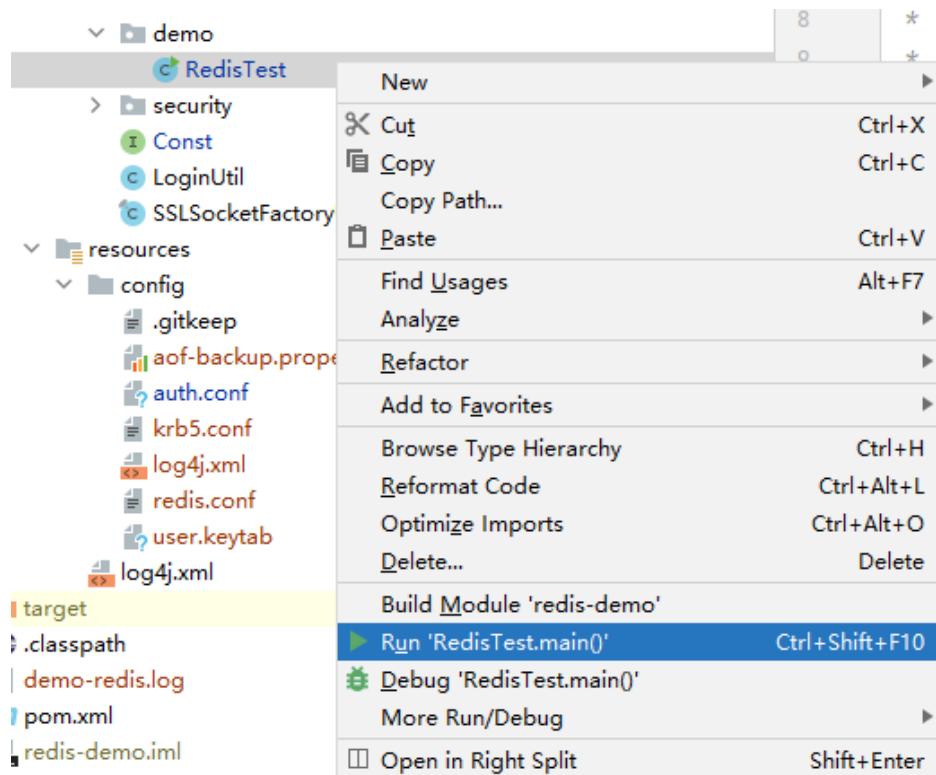
Figure 2-229 Compilation completed



## 2. Run the program.

Right-click **RedisTest.java**, and choose **Run 'RedisTest.main()'** from the shortcut menu to run the application project.

Figure 2-230 Run the application



----End

### 2.14.4.1.2 Viewing Commissioning Results

#### Scenario

After an Redis application is run, you can check the running result through one of the following methods:

- Viewing the IntelliJ IDEA running result.
- Logging in to the Redis Shell client to view the command output.

#### Procedure

- Viewing the IntelliJ IDEA running result

```
2019-06-22 16:23:27,519 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:93) - User sid-user01, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:37,921 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:101) - User sid-user01, session id: null
2019-06-22 16:23:38,084 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:107) - Value: hello
2019-06-22 16:23:38,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:111) - After append, value: hello world
2019-06-22 16:23:38,332 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:117) - User message, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:38,564 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:132) - All messages: [Hello how are you?, Fine thanks. I'm having fun with redis., I should look into this NOSQL thing ASAP]
2019-06-22 16:23:38,611 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:135) - Message count: 3
2019-06-22 16:23:38,665 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:139) - First message: Hello how are you?
2019-06-22 16:23:38,705 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:141) - After one pop, message count: 2
2019-06-22 16:23:39,016 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:157) - length of 1000000 is 7
2019-06-22 16:23:39,092 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:162) - User J001's name is John
2019-06-22 16:23:39,150 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:165) - {name=John, id=J001, gender=male, salary=1000000, age=35}
2019-06-22 16:23:39,430 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:179) - User L002's salary is 250000
2019-06-22 16:23:39,472 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:183) - all fields: [name, id, salary, gender, age]
2019-06-22 16:23:39,510 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:186) - all values: [L002, Lucy, 250000, 25, female]
2019-06-22 16:23:39,563 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:190) - partial field values: [L002, Lucy]
2019-06-22 16:23:39,605 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:194) - Exist field gender? true
2019-06-22 16:23:39,695 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:199) - after del field age, rest fields: [name, id, salary, gender]
2019-06-22 16:23:39,810 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:207) - key1's value is : value1
2019-06-22 16:23:39,888 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:211) - Can delete key by UNLINK.
2019-06-22 16:23:40,044 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:224) - Set size: 3
2019-06-22 16:23:40,122 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:228) - Set size: 3
2019-06-22 16:23:40,163 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:231) - Set: [TreeSet, HashSet, SortedSet]
2019-06-22 16:23:40,249 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:235) - Set: [TreeSet, HashSet]
2019-06-22 16:23:40,287 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:238) - TreeSet is set's member: true
2019-06-22 16:23:40,676 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:256) - All hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto, Linus
```

```
Torvalds]
2019-06-22 16:23:40,718 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:259) - Size: 6
2019-06-22 16:23:40,767 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:262) - Score: 1969.0
2019-06-22 16:23:40,808 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:265) - Count: 2
2019-06-22 16:23:40,860 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:269) - All
hackers 2: [Linus Torvalds, Yukihiro Matsumoto, Richard Stallman, Alan Kay, Claude Shannon, Alan
Turing]
2019-06-22 16:23:40,942 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:273) - All
hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto]
2019-06-22 16:23:41,106 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:284) - TTL: 5
2019-06-22 16:23:41,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:288) - KEY
type: string
2019-06-22 16:23:41,448 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:297) - List:
[1, 4, 6, 3, 8]
2019-06-22 16:23:41,490 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:300) - Sort
list: [1, 3, 4, 6, 8]
2019-06-22 16:23:42,129 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:343) - Result:
[www.google.cn, www.baidu.com, www.sina.com]
2019-06-22 16:23:42,301 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:60) - The
distance between test1 and test2 is 96.3242 KM.
2019-06-22 16:23:42,341 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:63) - Geo
location info is [(10.000002086162567,30.000000249977013),
(10.999999344348907,30.000000249977013)]
2019-06-22 16:23:42,380 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:66) -
geohash info of test1 and test2 is [sjr4et3f8v0, sjrfdm3fwto]
2019-06-22 16:23:42,424 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:72) - Get
location info by longitude and latitude :
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test1 :
146.8169
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test2 :
111.2263
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test3 :
1.0E-4
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test4 :
95.3394
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test5 :
482.4041
2019-06-22 16:23:42,462 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:80) - Get
location info by member :
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test3 :
0.0
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test4 :
95.3395
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test2 :
111.2264
2019-06-22 16:23:42,657 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:312) - Byte
conversion can be done by dump and restore
```

- Logging in to the Redis Shell client to view the command output:  
After the sample code is executed, keys are deleted. Therefore, you cannot directly use the shell command to view the application running result after the sample project is executed. The following uses sample code of **String Type Access** to explain how to use the shell command to view the application running result:

#### NOTE

Prerequisites:

- The client has been installed on the Linux.
  - Redis user security authentication has been performed on the client.
- Comment out **client.del(key);** in the last line of the **TestString()** method.  
**//client.del(key);**

- b. Perform operations as instructed in [Compiling and Running an Application](#).

- c. Run the following command on the Linux client. In this command, replace IP with the IP address of any node in the Redis cluster.

```
redis-cli -c -p 22400 -h /P get message
```

The value **A0BC9869FBC92933255A37A1D21167B2** set in the Java API is displayed:

```
[root@192-168-33-94 bin]#./redis-cli -c -p 22400 -h 192.168.33.94 get message  
"A0BC9869FBC92933255A37A1D21167B2"
```

#### NOTE

If channel encryption is enabled for Redis, run **redis-cli -c -p 22400 --tls -h /P get message**.

To enable channel encryption for Redis, log in to Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for **REDIS\_SSL\_ON**, and change the parameter value to **true** to enable SSL channel encryption.

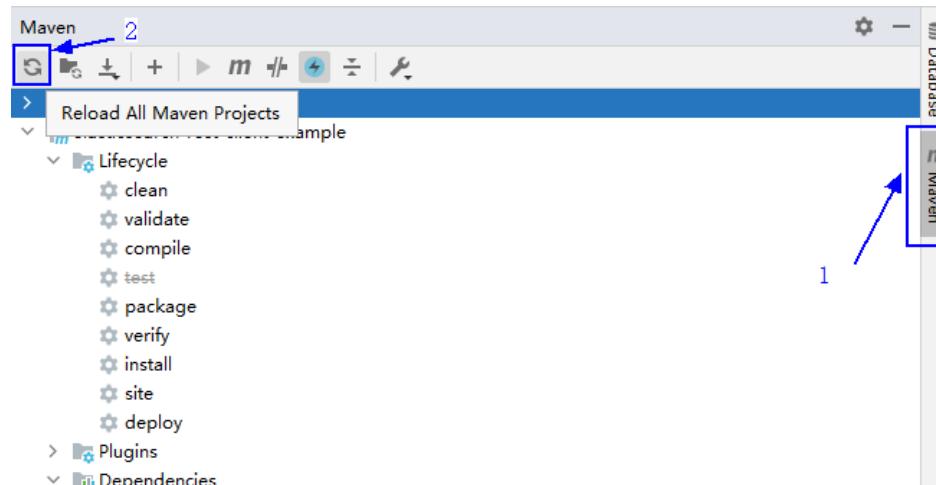
### 2.14.4.1.3 Commissioning a Spring Boot Program

You can run applications in the Windows environment after application code development is complete. If the local and cluster service planes can communicate with each other, you can perform the commissioning on the local host.

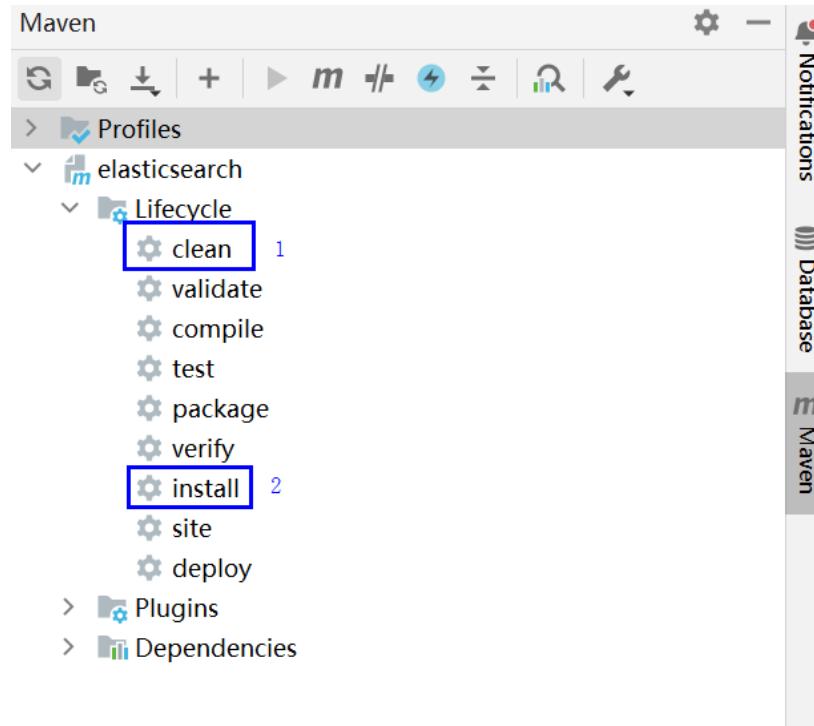
#### Step 1 Compile the Spring Boot sample to obtain **spring-boot-redis-1.0-SNAPSHOT.jar**.

1. Click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependency.

**Figure 2-231** Importing dependencies



2. Double-click **clean** and **install** in sequence to run the **maven clean** and **maven install** commands. If "Build Success" is displayed, the compilation is successful.



### Step 2 Read the configuration file.

1. Right-click **resource** in the sample code project and choose **Mark Directory as > Test Resource Root** from the shortcut menu.
2. Modify parameters in the **application.properties** file based on the site requirements.

### Step 3 Run the applications.

Right-click **SpringDataRedisApplication** and choose **Run'SpringDataRedisApplication...'** from the shortcut menu to start the Spring Boot service. If the Springboot service is started, the following information is displayed.

**Figure 2-232 Startup result**

```

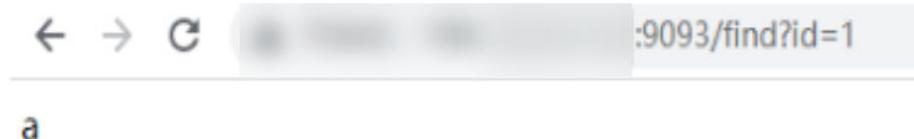
2023-02-14 14:52:40.499 INFO 21968 --- [main] c.j.redis.SpringdataRedisApplication : Starting SpringDataRedisApplication using Java 1.8.0_231 on A191122213 with PID 21968 (C:\MyCode\sample_project\tc\src\main\java\com\j_redis\SpringdataRedisApplication.java)
2023-02-14 14:52:40.501 INFO 21968 --- [main] c.j.redis.SpringdataRedisApplication : No active profile set, falling back to default profiles: default
2023-02-14 14:52:40.913 INFO 21968 --- [main] org.eclipse.jetty.util.log : Logging initialized @1449ms to org.eclipse.jetty.util.log.Slf4jLog
2023-02-14 14:52:40.973 INFO 21968 --- [main] o.s.o.a.e.JettyServerWebServerFactory : Server initialized with port: 9093
2023-02-14 14:52:40.981 INFO 21968 --- [main] org.eclipse.jetty.server.Server : jetty-9.4.35.v20201208; built: 2020-11-11T07:03:94Z; git: bdd54f03a5e0a7e280fab27f55c3c75ee8da89fb; jvm 1.8.0_231-b1
2023-02-14 14:52:41.001 INFO 21968 --- [main] o.s.o.a.e.JettyServerWebServerFactory : Registering Spring embedded web context
2023-02-14 14:52:41.001 INFO 21968 --- [main] o.s.o.a.e.JettyServerWebServerFactory : Initializing Spring embedded web context
2023-02-14 14:52:41.005 INFO 21968 --- [main] org.eclipse.jetty.server.ApplicationContext : Root WebApplicationContext: initialization completed in 466 ms
2023-02-14 14:52:41.050 INFO 21968 --- [main] org.eclipse.jetty.server.SessionHandler : DefaultSessionManager maxIdleTimeNoFlush
2023-02-14 14:52:41.050 INFO 21968 --- [main] org.eclipse.jetty.server.SessionHandler : No SessionScavenger set, using defaults
2023-02-14 14:52:41.050 INFO 21968 --- [main] org.eclipse.jetty.server.SessionHandler : node0 Scavenging every 600000ms
2023-02-14 14:52:41.041 INFO 21968 --- [main] o.e.j.server.handler.ContextHandler : Started o.s.w.e.JettyEmbeddedWebApplicationContext@52eacab4[application,/,[file:///C:/Users/s00518093/AppData/Local/Temp/jet
2023-02-14 14:52:41.055 INFO 21968 --- [main] o.s.r.util.CustomerRedisTemplateUtil : user=flankuser@HADOOP.COM, realm=HADOOP.COM

```

### Step 4 Open a browser, enter <http://localhost:9093/save> and save the data to Redis, as shown in the following figure.



**Step 5** Open a browser, enter **http://localhost:9093/find?id=1** and save the data to Redis, as shown in the following figure.



----End

## 2.14.4.2 Commissioning an Application in Linux

### 2.14.4.2.1 Compiling and Running an Application

#### Scenario

In a Linux environment, you can upload the JAR package to the Linux and then run an application after the code development is complete.

#### Prerequisites

JDK has been installed.

#### Procedure

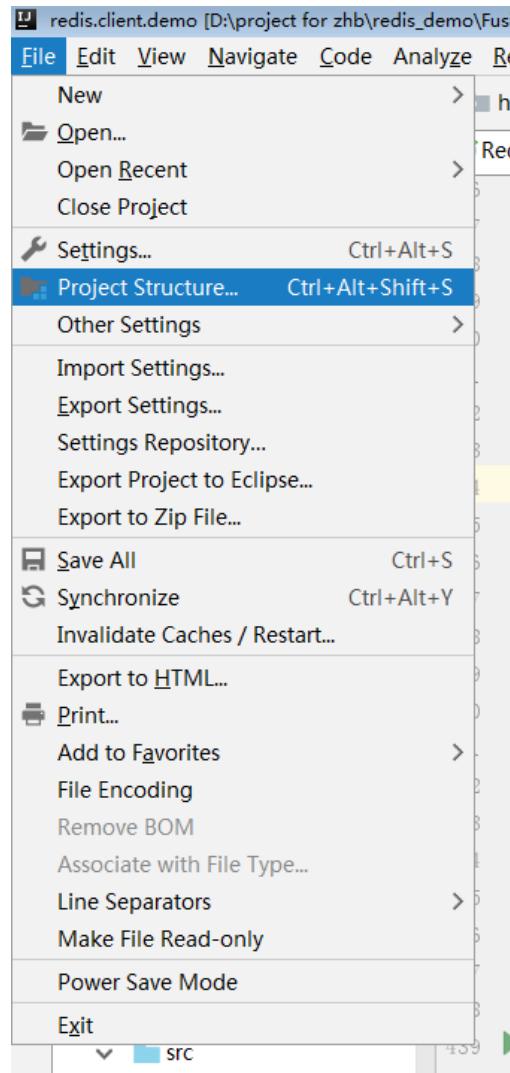
**Step 1** Export a JAR package.

1. On the IntelliJ IDEA **redis-examples** menu bar, choose **File > Project Structure...**, as shown in [Figure 2-233](#).

#### NOTE

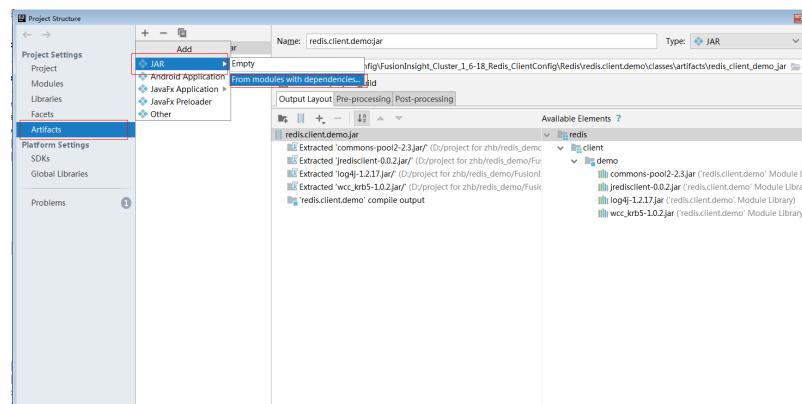
The preceding project name is for reference only. Use the actual project name in an actual scenario.

**Figure 2-233 "File > Project Structure..." shortcut menu**



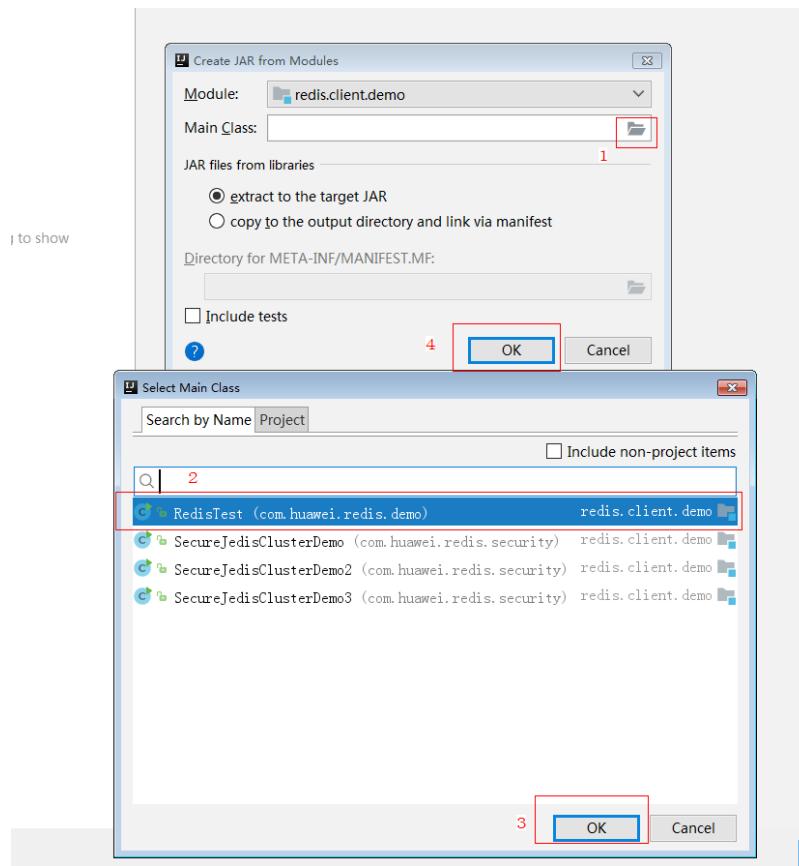
2. In the displayed **Project Structure** panel, select **Artifacts**. Then Click plus sign (+) and choose **JAR > From modules with dependencies**, as shown in **Figure 2-234**.

**Figure 2-234 Artifacts panel**



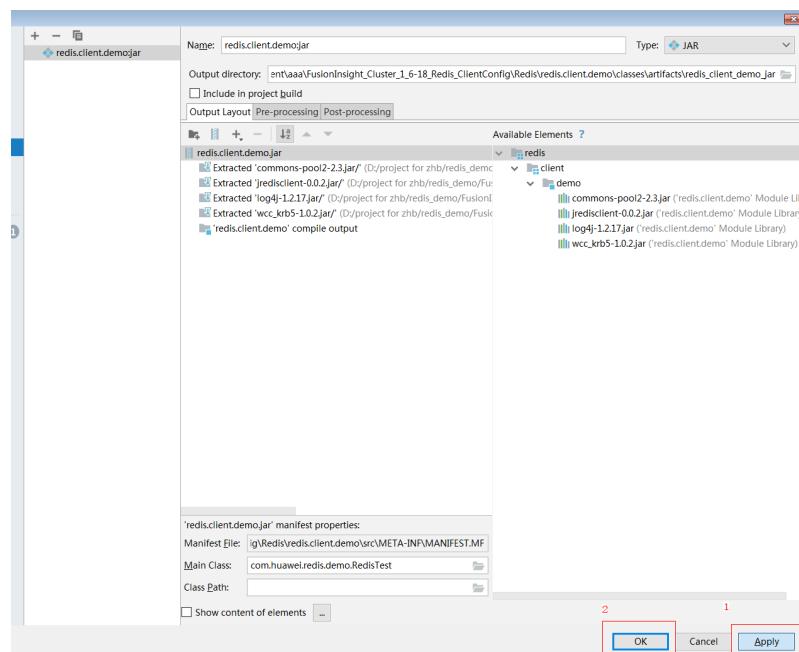
3. In the displayed **Create JAR from Modules** dialog box, click the folder icon, select **RedisTest** for **Main Class**, and click **OK**, as shown in **Figure 2-235**.

Figure 2-235 Specifying Main Class



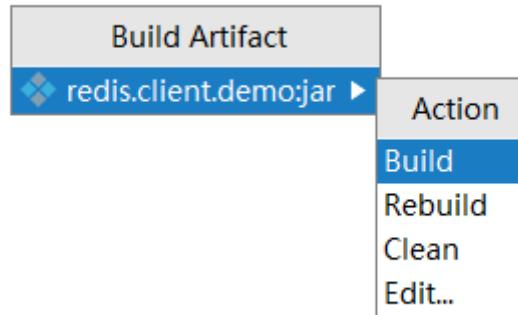
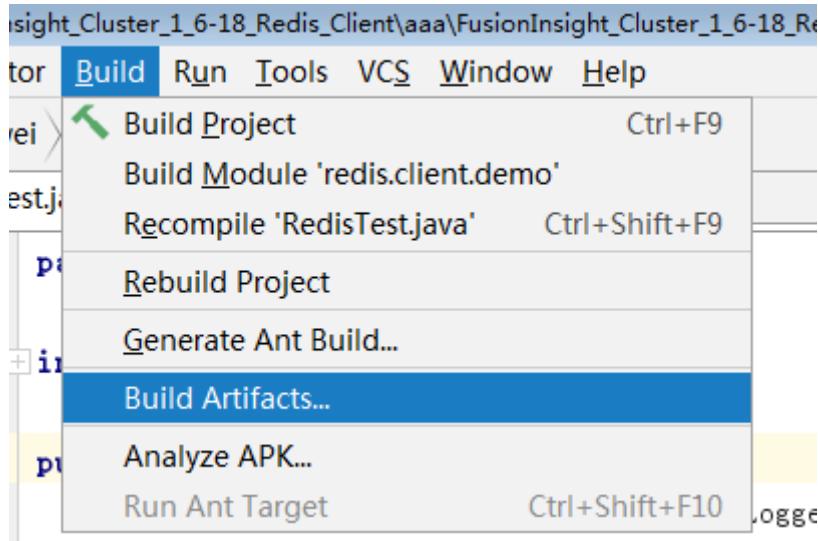
- Click the folder icon of **Output directory**, select the JAR package export path, and choose **Apply > OK**, as shown in [Figure 2-236](#).

Figure 2-236 Selecting a JAR package export path



5. Choose **Build > Build Artifacts**. In the displayed interface, choose **redis.client.demo:jar > Build** to export the JAR package, as shown in [Figure 2-237](#).

**Figure 2-237** export the JAR package



**Step 2** Prepare for the required JAR packages and configuration files.

In the Linux environment, create a directory, for example, **/opt/testone**, and create subdirectories **config** and **lib**. Upload the JAR packages exported in **Step 1** to the **lib** directory in Linux. Upload the conf configuration files in the **/src/main/resources/config** of the sample project to the **config** directory in Linux.

**Step 3** Go to **/opt/testone**, ensure that jdk has been installed, and java environment variables are set. Then, run the following commands to run the JAR packages:

```
java -cp .:lib/* com.huawei.redis.demo.RedisTest  
----End
```

## 2.14.4.2.2 Viewing Commissioning Results

### Scenario

After an Redis application is run, you can check the running result through one of the following methods:<0>

- Viewing the Linux running result
- Logging in to the Redis Shell client to view the command output

### Procedure

- Viewing the Linux running result

The following success information will be displayed:

```
2019-06-22 16:23:27,519 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:93) - User sid-user01, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:37,921 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:101) - User sid-user01, session id: null
2019-06-22 16:23:38,084 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:107) - Value: hello
2019-06-22 16:23:38,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:111) - After append, value: hello world
2019-06-22 16:23:38,332 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:117) - User message, session id: A0BC9869FBC92933255A37A1D21167B2
2019-06-22 16:23:38,564 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:132) - All messages: [Hello how are you?, Fine thanks. I'm having fun with redis., I should look into this NOSQL thing ASAP]
2019-06-22 16:23:38,611 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:135) - Message count: 3
2019-06-22 16:23:38,665 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:139) - First message: Hello how are you?
2019-06-22 16:23:38,705 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:141) - After one pop, message count: 2
2019-06-22 16:23:39,016 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:157) - length of 1000000 is 7
2019-06-22 16:23:39,092 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:162) - User J001's name is John
2019-06-22 16:23:39,150 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:165) - {name=John, id=J001, gender=male, salary=1000000, age=35}
2019-06-22 16:23:39,430 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:179) - User L002's salary is 250000
2019-06-22 16:23:39,472 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:183) - all fields: [name, id, salary, gender, age]
2019-06-22 16:23:39,510 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:186) - all values: [L002, Lucy, 250000, 25, female]
2019-06-22 16:23:39,563 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:190) - partial field values: [L002, Lucy]
2019-06-22 16:23:39,605 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:194) - Exist field gender? true
2019-06-22 16:23:39,695 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:199) - after del field age, rest fields: [name, id, salary, gender]
2019-06-22 16:23:39,810 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:207) - key1's value is : value1
2019-06-22 16:23:39,888 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:211) - Can delete key by UNLINK.
2019-06-22 16:23:40,044 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:224) - Set size: 3
2019-06-22 16:23:40,122 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:228) - Set size: 3
2019-06-22 16:23:40,163 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:231) - Set: [TreeSet, HashSet, SortedSet]
2019-06-22 16:23:40,249 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:235) - Set: [TreeSet, HashSet]
2019-06-22 16:23:40,287 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:238) - TreeSet is set's member: true
```

```
2019-06-22 16:23:40,676 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:256) - All
hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto, Linus
Torvalds]
2019-06-22 16:23:40,718 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:259) - Size: 6
2019-06-22 16:23:40,767 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:262) - Score:
1969.0
2019-06-22 16:23:40,808 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:265) - Count:
2
2019-06-22 16:23:40,860 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:269) - All
hackers 2: [Linus Torvalds, Yukihiro Matsumoto, Richard Stallman, Alan Kay, Claude Shannon, Alan
Turing]
2019-06-22 16:23:40,942 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:273) - All
hackers: [Alan Turing, Claude Shannon, Alan Kay, Richard Stallman, Yukihiro Matsumoto]
2019-06-22 16:23:41,106 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:284) - TTL: 5
2019-06-22 16:23:41,162 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:288) - KEY
type: string
2019-06-22 16:23:41,448 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:297) - List:
[1, 4, 6, 3, 8]
2019-06-22 16:23:41,490 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:300) - Sort
list: [1, 3, 4, 6, 8]
2019-06-22 16:23:42,129 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:343) - Result:
[www.google.cn, www.baidu.com, www.sina.com]
2019-06-22 16:23:42,301 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:60) - The
distance between test1 and test2 is 96.3242 KM.
2019-06-22 16:23:42,341 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:63) - Geo
location info is [(10.000002086162567,30.000000249977013),
(10.999999344348907,30.000000249977013)]
2019-06-22 16:23:42,380 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:66) -
geohash info of test1 and test2 is [sjr4et3f8v0, sjrfdm3fwto]
2019-06-22 16:23:42,424 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:72) - Get
location info by longitude and latitude :
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test1 :
146.8169
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test2 :
111.2263
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test3 :
1.0E-4
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test4 :
95.3394
2019-06-22 16:23:42,425 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:74) - test5 :
482.4041
2019-06-22 16:23:42,462 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:80) - Get
location info by member :
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test3 :
0.0
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test4 :
95.3395
2019-06-22 16:23:42,463 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:82) - test2 :
111.2264
2019-06-22 16:23:42,657 [main] INFO com.huawei.redis.demo.RedisTest (RedisTest.java:312) - Byte
conversion can be done by dump and restore
```

- Logging in to the Redis Shell client to view the command output:

After the sample code is executed, keys are deleted. Therefore, you cannot directly use the shell command to view the application running result after the sample project is executed. The following uses sample code of **String Type Access** to explain how to use the shell command to view the application running result:

#### NOTE

Prerequisites:

- The client has been installed on the Linux.
  - Redis user security authentication has been performed on the client.
- i. Comment out **client.del(key);** in the last line of the **TestString()** method.

- //client.del(key);
- ii. Perform operations as instructed in [Compiling and Running an Application](#).
  - iii. Run the following command on the Linux client. In this command, replace IP with the IP address of any node in the Redis cluster.  
`redis-cli -c -p 22400 -h /P get message`

The value **A0BC9869FBC92933255A37A1D21167B2** set in the Java API is displayed:

```
[root@192-168-33-94 bin]#./redis-cli -c -p 22400 -h 192.168.33.94 get message  
"A0BC9869FBC92933255A37A1D21167B2"
```

 NOTE

If channel encryption is enabled for Redis, run `redis-cli -c -p 22400 --tls -h /P get message`.

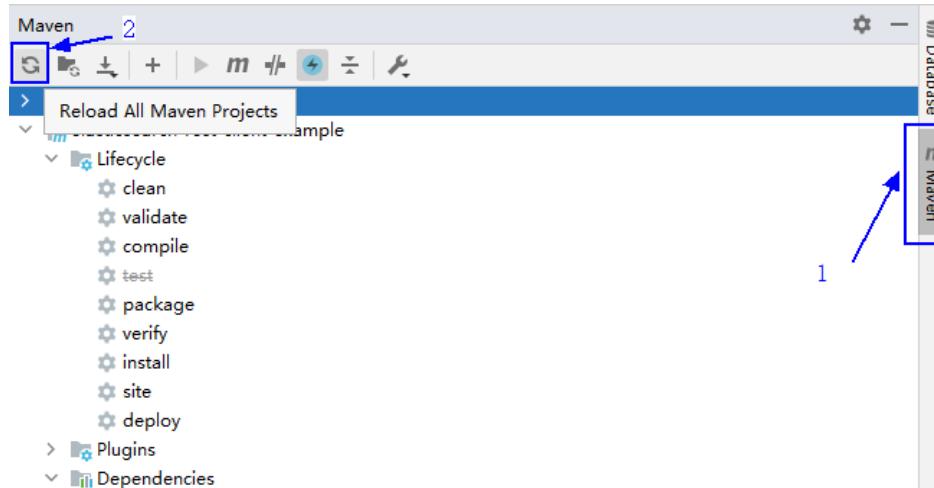
To enable channel encryption for Redis, log in to Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for **REDIS\_SSL\_ON**, and change the parameter value to **true** to enable SSL channel encryption.

#### 2.14.4.2.3 Commissioning the Spring Boot Program

**Step 1** Compile the Spring Boot sample to obtain **spring-boot-redis-1.0-SNAPSHOT.jar**.

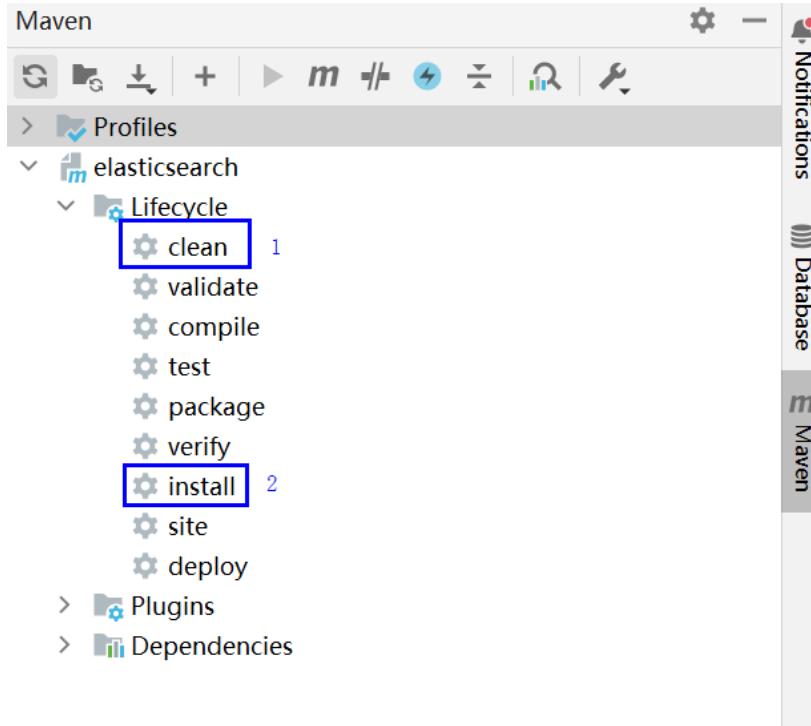
1. Click **Reload All Maven Projects** in the Maven window on the right of IDEA to import Maven project dependency.

**Figure 2-238** Importing dependencies



2. Double-click **clean** and **install** in sequence to run the **maven clean** and **maven install** commands. If "Build Success" is displayed, the compilation is successful.

A JAR file containing the **spring-boot-redis-** field is generated in the **target** directory of the sample project.



**Step 2** Create the `/opt/spring` directory on the Linux OS and upload the JAR packages whose names contain `spring-boot-redis-` in the target directory generated in [Step 1](#) to the directory.

**Step 3** Create the `/opt/spring/config` directory on Linux, modify the `application.properties` file based on the site requirements, and upload the file to the current directory.

**Step 4** Run the following command in `/opt/spring/config` to start Spring Boot:

```
java -jar spring-boot-redis-1.0-SNAPSHOT.jar
```

The command output is as follows.

**Step 5** Open a browser, enter `http://IP address of the Linux server:9093/save` in the address box, and save the data to Redis, as shown in the following figure.



- Step 6** Open a browser, enter `http://IP address of the Linux server:9093/find?id=1`, and save the data to Redis, as shown in the following figure.



## 2.14.5 More Information

### 2.14.5.1 External Interfaces

#### 2.14.5.1.1 Shell

You can directly perform operations on Redis using Shell on the server. Versions of Shell interfaces of Redis need to be consistent with those in the open-source community. For details, see <http://redis.io/commands>.

Methods of running the Shell command:

- Step 1** After the client is installed, go to the Redis client installation directory and run the `source bigdata_env` command. For a cluster in security mode, run the `kinit login user name`.

- Step 2** run the following command:

`redis-cli -h IP -p port`



If channel encryption is enabled for Redis, run `redis-cli -c -h IP -p port --tls`.

To enable channel encryption for Redis, log in to Manager, choose **Cluster > Services > Redis > Configurations > All Configurations**, search for `REDIS_SSL_ON`, and change the parameter value to `true` to enable SSL channel encryption.

- Step 3** Go to the running mode of the Redis command (also called CLI client connection).

192.168.0.11:22400>

- Step 4** Run the `help` command to obtain the help information of the Redis command parameters.

----End

#### 2.14.5.1.2 Java API

Redis APIs are consistent with community Jedis APIs. For details, see <https://github.com/xetorthio/jedis>.

## 2.15 RTD Development Guide

## 2.15.1 Overview

### 2.15.1.1 Application Development Overview

FusionInsight RTD is an enterprise-level distributed real-time decision-making platform that suits for mass data processing in high-concurrency and low latency scenarios. It supports user-defined rules and horizontal expansion. It bridges the "last mile" towards business decision-making, providing enterprises with precise risk control and marketing.

### 2.15.1.2 Common Concepts

FusionInsight RTD is a component of MRS. It consists of multiple components and provides the following functions:

- RTDService
  - As the core component of RTD, RTDService functions as the unified web definition entry of RTD and allows users to define tenants, event sources, dimensions, variables, models, and rules. RTDService is deployed in active/standby mode to ensure availability.
- Containers
  - The Containers service provides physical environments for the running of Business Logic Unit (BLU) instances and controls the start and stop of the BLUs.
  - The Containers service provides Access Load Balance (ALB) to connect to load balancers. ALB implements socket access. Specifically, it distributes requests of different projects to service instances on the platform based on different processing policies and implements conversion between protocol interfaces. ALB is not provided as an independent service but integrated in Containers.
- MOTService
  - MOTService provides fast and large-throughput access capabilities and uses stored procedures to quickly process service logic at the database layer. It is deployed in active/standby mode.

### 2.15.1.3 Development Process

This section describes how to use Java APIs to develop RTD applications.

[Figure 2-239](#) and [Table 2-151](#) describe the phases in the development process.

Figure 2-239 RTD development process

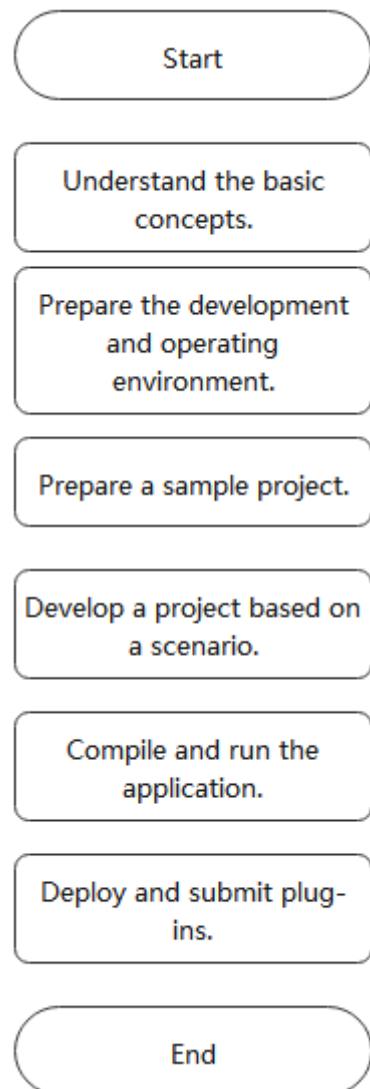


Table 2-151 Description of RTD development process

Step	Description	Reference Documents
Understand the basic concepts.	Before developing an application, learn basic concepts of RTD and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>
Prepare the development and operating environment.	The Java language is recommended for RTD application development. You can use the IntelliJ IDEA tool.	<a href="#">Preparing the Development and Operating Environment</a>

Step	Description	Reference Documents
Prepare a sample project.	RTD provides sample projects in various scenarios. Sample projects can be imported for studying.	<a href="#">Configuring and Importing Sample Projects</a>
Develop a project based on a scenario.	Decision engine and decision-making flow customization are supported, and dynamic variables can be extended.	<a href="#">Developing an Application</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Compiling and Packaging</a>

## 2.15.2 Environment Preparation

### 2.15.2.1 Preparing the Development and Operating Environment

[Table 2-152](#) describes the environment required for application development.

**Table 2-152** Items needed for environment preparation

Item	Description
OS	Windows 7 or later is supported. Network environment: The local development environment must interconnect with the cluster service-plane network.
JDK installation	Basic configuration of the development and running environment. The version requirements are as follows: The JDK version must be the same as the version of FusionInsight Manager to be accessed. For details about the JDK version, see the corresponding version document or contact the system administrator.
IntelliJ IDEA installation and configuration	Basic configuration of the development environment. The version must be 2019.1 or other compatible versions.

## 2.15.2.2 Configuring and Importing Sample Projects

### Procedure

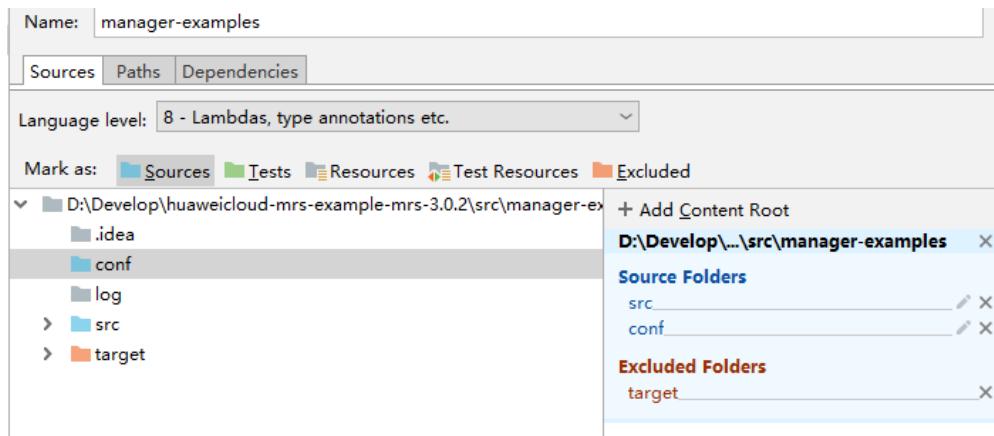
- Step 1** Obtain the sample project folder **rtd-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
1. Open IntelliJ IDEA and choose **File > New > Project from Existing Sources**. In the displayed **Select File or Directory to Import** window, select the sample code folder.
  2. Select the **pom.xml** file in the sample project folder, select **Import project from external model > Maven** as prompted, and click **Next** until **Finish** is displayed.

#### NOTE

The sample code is a Maven project. You can adjust the project configuration as required. The operations vary according to the IntelliJ IDEA version.

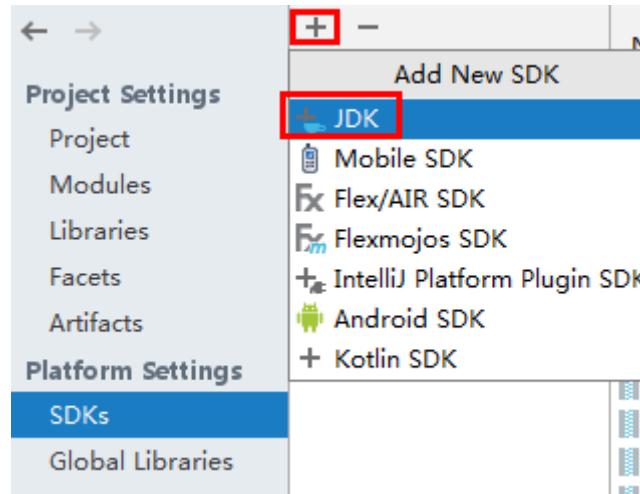
3. Add the **src** and **conf** directories in the project to the source file path. After the project is imported, choose **File > Project Structure** on the menu bar of IntelliJ IDEA. In the displayed window, choose **Project Settings > Modules**.

Select the current project, click the **src** folder, click **Sources** on the right of **Mark as**, click the **conf** folder, and click **Sources** on the right of **Mark as**. Click **Apply** and then **OK**.

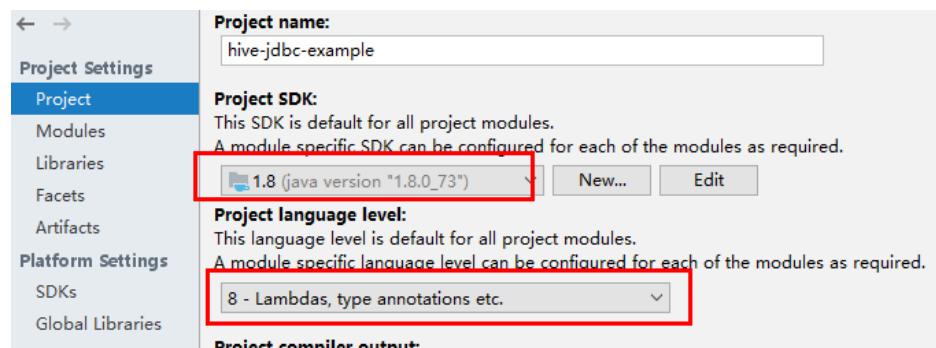


- Step 3** Set the JDK of the project.

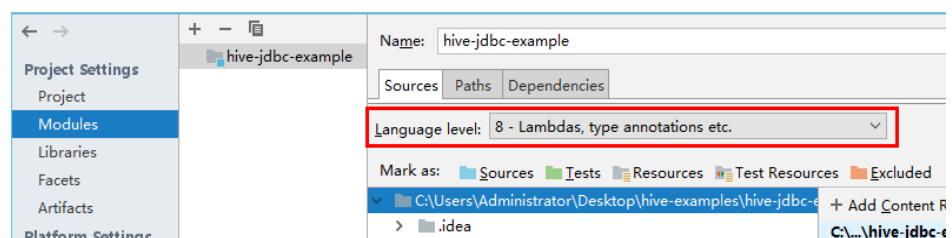
1. On the menu bar of IntelliJ IDEA, choose **File > Project Structure....** The **Project Structure** window is displayed.
2. Choose **SDKs**, click the plus sign (+), and select **JDK**.



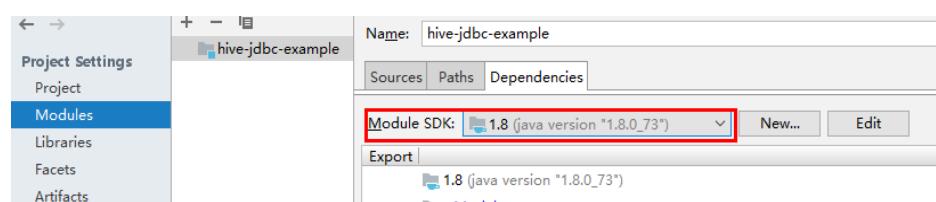
3. On the **Select Home Directory for JDK** page that is displayed, select the JDK directory and click **OK**.
4. Click **Apply**.
5. Select **Project**, select the JDK added in **SDKs** from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.



6. Click **Apply**.
7. Choose **Modules**. On the **Sources** tab page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**.



On the **Dependencies** tab page, change the value of **Module SDK** to the JDK added in **SDKs**.

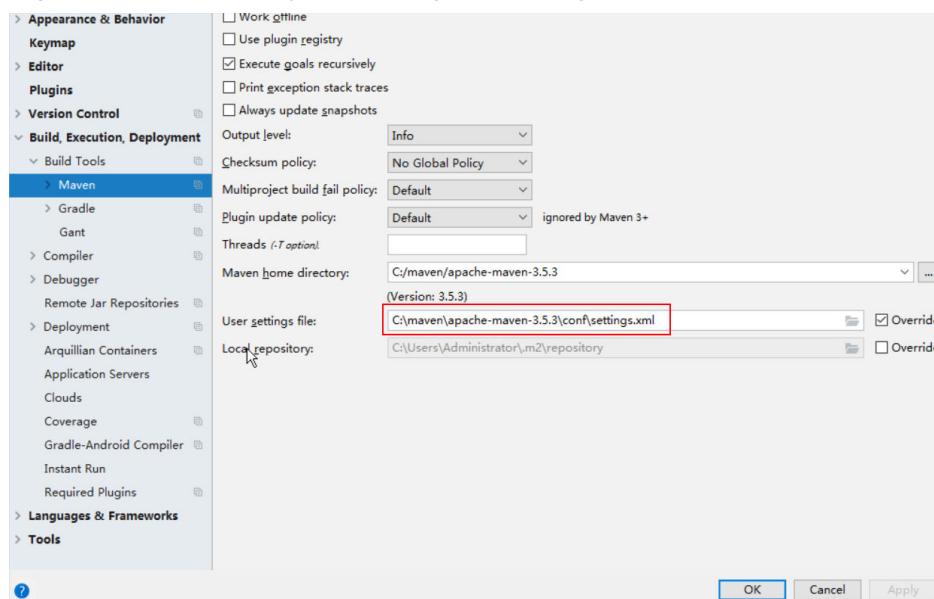


8. Click **Apply** and then **OK**.

**Step 4** Configure Maven.

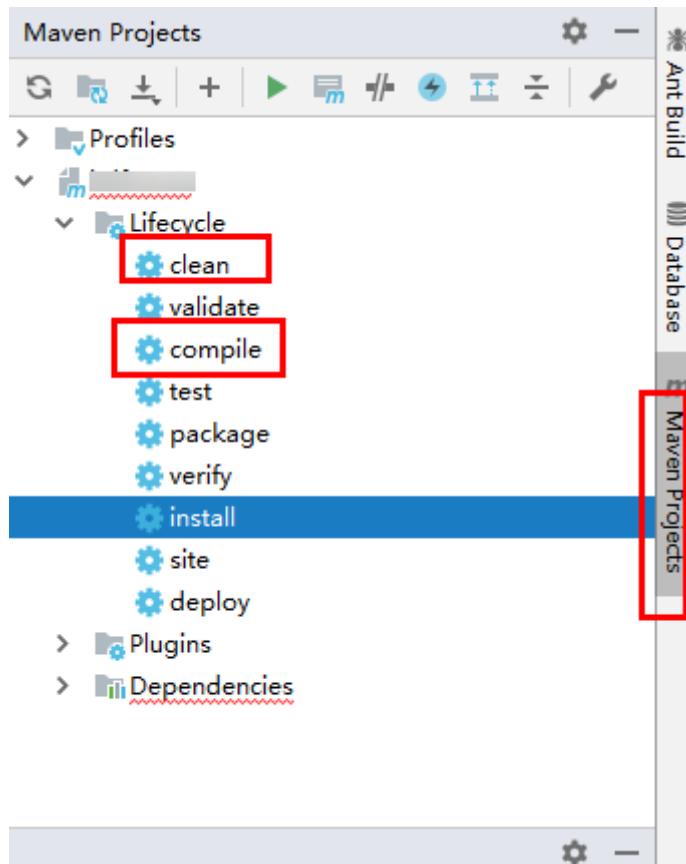
1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, choose **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is *<Local Maven installation directory>\conf\settings.xml*.

**Figure 2-240** Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.
4. Click **Apply** and then **OK**.
5. On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the displayed page, choose **Project name > Lifecycle** and run the **clean** and **compile** scripts.

Figure 2-241 Maven Projects page



----End

## 2.15.3 Developing an Application

### 2.15.3.1 Event Source Custom Plug-in Program

#### 2.15.3.1.1 Scenario

Users can customize and orchestrate decision-making flows and extended variables based on actual service requirements to meet more complex service requirements.

#### Data Preparation

Use Maven to package the entire project. By default, the following two files are generated:

- rtdexecutor-plugins-bundle-\*-.zip
  - This file contains only the content related to the DecisionFlowPlugin module.
  - If the content related to the DecisionFlowPlugin module is modified, the associated BLU will be restarted (rolling restart) after the update is performed on the RTD web UI.

- **rtdexecutor-plugins-bundle-\*-update.zip**
  - This file contains the contents of all modules of the current project.
  - If the content related to the VarsExtensionPlugin module is modified and updated on the RTD web UI, the update is loaded in hot loading mode and the associated BLU is not restarted.

 NOTE

You can modify the **assembly** configuration file as required.

## File Modification

Modify the **pom.xml** and **plugin.properties** files for each submodule in the project.

- **pom.xml**

```
<properties>
<plugin.id>DecisionFlowPlugin</plugin.id>
<plugin.class>com.huawei.fi.plugins.ExampleDecisionFlowPlugin</plugin.class>
<plugin.version>0.1.0</plugin.version>
<plugin.provider>cmb</plugin.provider>
</properties>
```
- **plugin.properties**

```
plugin.id=DecisionFlowPlugin
plugin.class=com.huawei.fi.plugins.decisionflow.ExampleDecisionFlowPlugin
plugin.version=0.1.0
plugin.provider=cmb
```

 NOTE

Modify all parameters except **plugin.id** based on the site requirements.

### 2.15.3.1.2 Development Guidelines

#### Project Description

- Project name: **rtdexecutor-plugins**
- The following sub-modules are included:
  - DecisionFlowPlugin: used for custom orchestration of decision-making flows and can contain the **RTDEventHandler** class implemented by users.
  - VarsExtensionPlugin: used for packet extension and variable extension. You can implement the subclasses of **EventFieldsExtension** and **DynamicVarsExtension** to customize packets and variables.
- The names of sub-modules in a project are fixed and cannot be customized.
- The project must contain the **DecisionFlowPlugin** submodule. The **VarsExtensionPlugin** submodule is optional.

### 2.15.3.1.3 Sample Code

#### DecisionFlowPlugin

The following code snippets are stored in **com/huawei/fi/plugins/decisionflow**.

- **ExampleDecisionFlowPlugin.java**

The custom plug-in needs to inherit the plug-in class of **pf4j** and overwrite the default construction method, as well as **start** and **stop** methods. When

the BLU associated with the RTD event source is started or stopped, the **start** or **stop** method is invoked.

```
public class ExampleDecisionFlowPlugin extends Plugin
{
    private static final Logger LOG = Logger.getLogger(ExampleDecisionFlowPlugin.class);

    public ExampleDecisionFlowPlugin(PluginWrapper wrapper)
    {
        super(wrapper);
    }

    @Override
    public void start()
    {
        LOG.info("ExampleDecisionFlowPlugin.start()");
    }

    @Override
    public void stop()
    {
        LOG.info("ExampleDecisionFlowPlugin.stop()");
    }
}
```

- **ExampleScoreEventHandler.java**

Implement the **RTDEventHandler** subclass based on the site requirements.  
This is optional.

```
public class ExampleScoreEventHandler extends RTDEventHandler
{
}
```

- **ExampleDecisionFlowExtension.java**

- Inherit **DecisionFlowExtension** and add **pf4j** annotations to the subclass so that the subclass can be loaded by **pf4j**.

```
@Extension
public class ExampleDecisionFlowExtension extends DecisionFlowExtension
{
}
```

- Overwrite the **getPipelineDecisionServiceMode** method of **DecisionFlowExtension** to set the decision-making type (real-time or quasi-real-time mode) based on the service scenario.

```
/**
 * Obtain the decision-making type and customize it based on the business scenario
 * requirements.
 * Real-time mode: Constants.DECISION_SERVICE_MODE_RT
 * Quasi-real-time mode: Constants.DECISION_SERVICE_MODE_NRT
 * @return int decision mode
 */
@Override
public int getPipelineDecisionServiceMode()
{
    return Constants.DECISION_SERVICE_MODE_RT;
}
```

- Override the **orchestrateDecisionFlow** method of **DecisionFlowExtension** to orchestrate the **handler** process of a custom decision-making flow based on service requirements.

```
@Override
public void orchestrateDecisionFlow(RTDPipeline pipeline)
{
    String runtimeModel = RTDEnvironment.getProperty("plugin.RuntimeMode");
    LOG.info("RuntimeMode: " + runtimeModel + ".");
    // for testing the development mode
    if (RuntimeMode.DEVELOPMENT.toString().equalsIgnoreCase(runtimeModel))
    {
```

```

        pipeline.setInboundEventHandler(new EventInboundEventHandler());
        pipeline.setOutboundEventHandler(new EventOutboundEventHandler());

        pipeline.addEventHandler(new DataPreEventHandler());
    }
else
{
    pipeline.setInboundEventHandler(new EventInboundEventHandler());
    pipeline.setOutboundEventHandler(new EventOutboundEventHandler());
    // Instructions:
    // 1. If the custom DataPreEventHandler is used to extend event variables,
    // pay attention to the package path of the DataPreEventHandler class.
    // Note: This mode does not support online dynamic update.
    // 2. If VarsExtensionPlugin is used to extend event variables,
    // DataPreEventHandler must be placed at the beginning of the entire pipeline,
    // DataPreEventHandler is a Huawei JAR built-in class.
    // The customer logic is in Extension of the VarsExtensionPlugin plug-in.
    // Note: This mode supports online dynamic update.
    pipeline.addEventHandler(new DataPreEventHandler());
    pipeline.addEventHandler(new InsertDataEventHandler());
    pipeline.addEventHandler(new CallFilterRulesEventHandler());
    pipeline.addEventHandler(new ComputeVarsEventHandler());
    // 1. If VarsExtensionPlugin is used to dynamically extend variables,
    // Add DynamicVarsEventHandler to the pipeline.
    // 2. If there is no dynamic extended variable, delete the handler from the pipeline.
    // 3. DynamicVarsEventHandler must be placed after ComputeVarsEventHandler.
    pipeline.addEventHandler(new DynamicVarsEventHandler());
    // Add the scoring model capability to the pipeline.
    pipeline.addEventHandler(new ScoreModelEventHandler());
    pipeline.addEventHandler(new CallProcRulesEventHandler());
    // The logic for adding custom decision-making is the same as the existing method.
    pipeline.addEventHandler(new ExampleScoreEventHandler());
}
}

```

- d. If you need to use the built-in packet extension and variable extension functions, import the class path of the handler when orchestrating the handler in the pipeline in 3.

```

import com.huawei.fi.rtdexecutor.RTDEnvironment;
import com.huawei.fi.rtdexecutor.common.Constants;
import com.huawei.fi.rtdexecutor.plugin.DecisionFlowExtension;
import com.huawei.fi.rtdexecutor.rtdeventhandlers. DataPreEventHandler;
import com.huawei.fi.rtdexecutor.rtdeventhandlers. DynamicVarsEventHandler;
import com.huawei.fi.rtdexecutor.rtdpipeline.RTDPipeline;

```

## VarsExtensionPlugin

The following code snippets are in the **com.huawei.fi.plugins.varsextension** package.

- **ExampleVarsExtensionPlugin.java**

The custom plug-in needs to inherit the plug-in class of **pf4j** and overwrite the default construction method, as well as **start** and **stop** methods. When the BLU associated with the RTD event source is started or stopped, the **start** or **stop** method is invoked.

In the current version, only one implementation of **EventFieldExtension** and **DynamicVarsExtension** is loaded for **ExampleVarsExtensionPlugin**. Even if the plug-in contains multiple implementations of **EventFieldExtension** and **DynamicVarsExtension**, only one implementation is loaded.

```

public class ExampleVarsExtensionPlugin extends Plugin
{
    private static final Logger LOG = Logger.getLogger(ExampleVarsExtensionPlugin.class);

    public ExampleVarsExtensionPlugin(PluginWrapper wrapper)
    {

```

```
        super(wrapper);
    }

    @Override
    public void start() { LOG.info("ExampleVarsExtensionPlugin.start()"); }

    @Override
    public void stop()
    {
        LOG.info("ExampleVarsExtensionPlugin.stop()");
    }
}
```

- **ExampleEventFieldsExtension.java**

- In this example, packet field extension is implemented, and the BLU does not need to be restarted when **VarsExtensionPlugin** is updated. Inherit the **EventFieldsExtension** class, add the **pf4j** extension point annotation **@Extension**, and rewrite the parent class method. In the **init/destroy** method, initialize or destroy basic resources related to packet extension. **init/destroy** is invoked when the BLU associated with the event source is restarted or the plug-in is updated.

```
@Extension
public class ExampleEventFieldsExtension extends EventFieldsExtension
{
    @Override
    public void init()
    {
        LOG.info("{} init start ...", this.getClass().getName());
        super.init();
        // add your init code at here
        this.timeoutMS = 20L;
        LOG.info("{} init end ...", this.getClass().getName());
    }

    @Override
    public void destroy()
    {
        LOG.info("{} destroy start ...", this.getClass().getName());
        super.destroy();
        // add your destroy code at here
        LOG.info("{} destroy end ...", this.getClass().getName());
    }
}
```

- Implement the **compute** method in **EventFieldsExtension**. The definition and calculation of extended packets are implemented in this method. In the following example code, the **BANK\_CUSTOMER\_ID**, **BANK\_CUSTOMER\_AGE** and **BANK\_CUSTOMER\_SALARY** fields are extended.

```
@Override
public void compute(PipelineContext ctx, EventData eventData) throws EventHandlerException
{
    ExtendEventField<String> bankCustomerID = new
    ExtendEventField.String("BANK_CUSTOMER_ID");
    ExtendEventField<Integer> bankCustomerAge = new
    ExtendEventField.Integer("BANK_CUSTOMER_AGE");
    ExtendEventField<Double> bankCustomerSalary = new
    ExtendEventField.Double("BANK_CUSTOMER_SALARY");

    bankCustomerAge.setValue(32);
    bankCustomerID.setValue("0000001");

    String city = (String) eventData.getData().get("LBS_CITY");
    if("Boston".equals(city)) {
        bankCustomerSalary.setValue(7830.25);
    } else if("New York".equals(city)) {
```

```
        bankCustomerSalary.setValue(9765.40);
    } else {
        bankCustomerSalary.setValue(6300.62);
    }

    putComputeResult(ctx, bankCustomerID.getName(), bankCustomerID);
    putComputeResult(ctx, bankCustomerAge.getName(), bankCustomerAge);
    putComputeResult(ctx, bankCustomerSalary.getName(), bankCustomerSalary);
}
```

- After the plug-in code is developed, use the packet field definition function of event source management on the RTDService web UI to map the plug-in code to event variables for subsequent calculation. In addition, the default value is defined for this function. If the packet extension calculation in the plug-in fails, the default value defined here will be used in subsequent calculation steps.

## DynamicVarsExtension

- In the ExampleDynamicVarsExtension.java example, variables are dynamically extended. When **VarsExtensionPlugin** is updated, the BLU does not need to be restarted. Inherit the **DynamicVarsExtension** class, add the **pf4j** extension point annotation **@Extension**, and rewrite the parent class method. In the **init/destroy** method, initialize or destroy basic resources related to dynamic extended variables. **init/destroy** is invoked when the BLU associated with the event source is restarted or the plug-in is updated.

```
@Extension
@SuppressWarnings({"rawtypes", "unchecked"})
public class ExampleDynamicVarsExtension extends DynamicVarsExtension
{
    /**
     * init your public resource at here,
     * plugins create or update, this method will be invoked
     *
     */
    @Override
    public void init()
    {
        LOG.info("{} init start ...", this.getClass().getName());
        super.init();
        // add your init code at here
        this.timeoutMS = 20L;
        LOG.info("{} init end ...", this.getClass().getName());
    }

    /**
     * recycle your resource at here,
     * plugins destroy or update, this method will be invoked
     *
     */
    @Override
    public void destroy()
    {
        LOG.info("{} destroy start ...", this.getClass().getName());
        super.destroy();
        // add your destroy code at here
        LOG.info("{} destroy end ...", this.getClass().getName());
    }
}
```

- When implementing the **DynamicVarsExtension** subclass, you can override the **verify** method to add some custom checks. This method is invoked when the plug-in is uploaded. If **false** is returned, the update fails.

```
public abstract class DynamicVarsExtension implements ExtensionPoint
{
```

```
public boolean verify()
{
    return true;
}
```

- To dynamically extend variables, implement the **addRequiredInputVars** method to define the event variables, real-time query variables, and batch variables that these variables depend on during calculation. When the update plug-in is uploaded, the system verifies the existence and online status of the variables defined in the method. If the verification fails, the system displays an update error. The definition of the dependent variable is very important. If the dependent variable is not defined or omitted, the calculation of the dynamic extended variable may be abnormal or incorrect, affecting the decision-making result of the request. In the following example, the plug-in depends on event variable **ev\_client\_ip**, real-time query variable **rv\_online\_pay**, and batch variable **bv\_history\_pay**.

```
@Override
public List<String> addRequiredInputVars()
{
    // define input vars which required in your code and built-in RTD platform,
    // include event vars, batch vars, runtime query vars
    List<String> inputVars = new ArrayList<>();
    inputVars.add("ev_client_ip");
    inputVars.add("rv_online_pay");
    inputVars.add("bv_history_pay");

    return inputVars;
}
```

- To dynamically extend variables, implement the **defineDynamicEventVars** method to define the dynamic extended variables contained in the **DynamicVarsExtension** implementation class. The dynamic extended variable must start with **ev\_d\_**, and the type and default value must be specified. In the following example, two dynamic extended variables are defined. The names are **ev\_d\_local\_city** and **ev\_d\_total\_pay**, the types are **String** and **Long**, and the default values are **Boston** and **11**. Currently, dynamic extended variables support five types: Integer, Long, String, Double, and Timestamp.

To dynamically extend variables, you need to implement the **compute** method and calculate the variables based on the variable values in the current pipeline request. The obtained variable values must be defined in **addRequiredInputVars** of the current implementation class.

```
@Override
public Map<String, DynamicEventVar<?>> defineDynamicEventVars()
{
    DynamicEventVar<String> localCity = new DynamicEventVar.String("ev_d_local_city", "Boston");
    DynamicEventVar<Long> totalPay = new DynamicEventVar.Long("ev_d_total_pay", 11L);

    Map<String, DynamicEventVar<?>> dynamicEventVarMap = new HashMap<>();
    dynamicEventVarMap.put(localCity.getName(), localCity);
    dynamicEventVarMap.put(totalPay.getName(), totalPay);

    return dynamicEventVarMap;
}
```

- During the calculation, when obtaining the dependent variable value from the pipeline, check whether the value is null and matches the type. Otherwise, null pointers and type conversion exceptions may occur.

```
@Override
public void compute(PipelineContext ctx, EventData eventData) throws EventHandlerException
{
    DynamicEventVar<?> definedTotalPay = getDefineDynamicEventVars().get("ev_d_total_pay");
    DynamicEventVar<Long> totalPay = computeTotalPay(ctx, eventData, definedTotalPay);
```

```
putComputeResult(ctx, totalPay.getName(), totalPay);

DynamicEventVar<?> definedLocalCity = getDefineDynamicEventVars().get("ev_d_local_city");
DynamicEventVar<String> localCity = computeLocalCity(ctx, eventData, definedLocalCity);
putComputeResult(ctx, localCity.getName(), localCity);
}
```

## End-to-End Timeout Control

Unified timeout control is supported for the pipeline. The timing starts when a request enters the BLU. If the threshold is exceeded, the request is returned quickly.

## Internal Dotting Timeout Control for Plug-ins

The timeout control is based on the timing dotting code. It determines whether the plug-in calculation exceeds the threshold from the time when the **compute** method is executed to the time when the dotting code is executed. The timeout threshold can be initialized in the **init** method or customized for a single timing point.

- In the following DynamicVarsExtensions sample code, `checkTimeoutMS(ctx, timeoutMS, true)` is the specific format of the timing dotting code. The first parameter indicates that the current dotting threshold is the value of the **timeoutMS** attribute. The second parameter indicates the exception handling mode after the timeout. If the value is **true**, the request is interrupted and the subsequent calculation is not performed. If the value is **false**, the subsequent variable calculation logic is skipped. The variables that are not calculated are calculated based on the default values, and the remaining steps are calculated normally.

```
private DynamicEventVar<Long> computeTotalPay(PipelineContext ctx, EventData eventData,
DynamicEventVar totalPayVar) throws EventHandlerException
{
    // get batch vars from eventData to compute dynamicEventVar
    Long historyPay = (Long) eventData.getBatchVars().get("bv_history_pay");
    historyPay = Objects.isNull(historyPay) ? 0L : historyPay;

    // get real time vars from eventData to compute dynamicEventVar
    Long onlinePay = (Long) eventData.getRtqVars().get("rv_online_pay");
    onlinePay = Objects.isNull(onlinePay) ? 0L : onlinePay;

    Long totalPay = historyPay + onlinePay;
    if(totalPay != 0L) {
        totalPayVar.setValue(totalPay);
    } else {
        totalPayVar.setException("required input vars maybe not exists or offline.");
    }

    checkTimeoutMS(ctx, timeoutMS, true);

    return totalPayVar;
}
```

- To ensure that original packet data can be properly imported to the database, **EventFieldExtension** provides only the `checkTimeoutMS(ctx, timeoutMS)` API. The subsequent variable calculation logic is skipped, and the variables that are not calculated are calculated based on the default values in the subsequent steps, and the remaining steps are calculated normally.

## Runtime Exception

If a runtime exception is thrown during variable calculation, the current request is quickly returned and all subsequent calculation steps are skipped.

## Checked Exception

If a checked exception is thrown during variable calculation, the remaining calculation of dynamic extended variables is skipped. However, the subsequent calculation steps defined in the pipeline continue to be processed.

- If an expected exception occurs during the calculation and you want to output the information to the result, you can invoke **DynamicEventVar**. The **setException()** method is used to set exception information. The exception information is displayed in the **EventVarsException** field in the calculation result.
- If the value of a defined dynamic extended variable fails to be set during request calculation, the default value will be used in subsequent calculation and will be displayed in the **EventVarsException** field in the calculation result.
- The following is an example of the **RTDExecutor\_result.log** file.

```
2017-05-08 16:43:47.047 INFO [pipelineTaskConsumer-2] - RTDResult :  
{"seqNo":"1580206d9e0f4e0a82b43911c2203f1d","dsType":"plugins_1","ddUpdate":"","statusCode":2,"statusMessage":"","data":null,"batchData":[],"eventVars":null,"batchVars":null,"rtqVars":null,"modelResults":null,"filterRules":null,"procRules":null,"userdata":{},"event_inbound_time":1494233027021,"total_time":20,"out_mq_total_time":0,"multi_proc_time":0,"EventFieldException":{},"plugin_dataprep_cost_time":0,"insert_time":11,"FilterRulesException":{},"EventVarsException":{"ev_d_hello":"compute fail for some reason, use default value."},"BatchVarsException":{},"RtqVarsException":{},"rtqVarExtraData":[],"plugin_dynamicvars_cost_time":1,"model_score_time":0,"ProcRulesException":{"exception_pr_rule2":"No message specified."}, "procRuleExtraData":[],"custom_user_data_key":"CustomUserDataValue","plugin_custom_cost_time":0,"HostIP":null,"ContainerId":null,"PipelineId":944044635,"ThreadId":1261,"TenantId":"1","EventSourceId":"1493972947028","TraceFlag":0,"rtdResults":{},"score":92,"scoreResult":9,"syncModeStatusCode":0,"syncModeStatusMessage":"OK!"}, "batchId":""}
```

### 2.15.3.2 Decision Engine Custom Program

#### 2.15.3.2.1 Scenario

Users can customize decision engine configurations to control the output of scoring results of multiple rule engines, meeting more complex service requirements.

#### 2.15.3.2.2 Development Guidelines

- Check whether the current event source plug-in is an extended decision-making flow and whether the plug-in code contains a rating decision-making flow. For details, see [Event Source Custom Plug-in Program](#).
- Determine whether to make decisions on the stored procedure rules of the current event source.

### 2.15.3.2.3 Sample Code

#### RtdDroolsFileGeneratorTest

The following code snippets are in the **RtdDroolsFileGeneratorTest** class in the **com/huawei/fi/rtd/drools/generator** package in the **test** directory.

```
package com.huawei.fi.rtd.drools.generator;
import com.huawei.fi.rtd.drools.generator.common.AlgorithmType;
import com.huawei.fi.rtd.drools.generator.common.ZipFileUtil;
import org.apache.commons.lang3.StringUtils;
import org.junit.*;
import org.junit.rules.TemporaryFolder;
import org.kie.api.runtime.KieContainer;
import java.io.*;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
public class RtdDroolsFileGeneratorTest {

    private static final String RTD_DROOLS_K_MODULE_XML = "kmodule.xml";
    private static final String RTD_META_DATA_PROPERTIES = "rtd_meta_data.properties";
    private static final String RTD_DROOLS_DECISION_JSON = "drools_contract.json";
    private static final String RTD_DROOLS_FILE_PREFIX = "rtd_drools_decision_";
    private static final int BUFFER = 10 * 1024;
    private KieContainer kieContainer;
    private RtdDroolsDecision droolsDecision;
    @Rule
    public TemporaryFolder temporaryFolder = new TemporaryFolder();
    @Before
    public void setUp() {
        droolsDecision = new RtdDroolsDecision(AlgorithmType.AVERAGE.name());
        RtdDroolsElementsGroup weightGroupOne = new RtdDroolsElementsGroup("drools_group_01",
AlgorithmType.WEIGHTING.name());
        weightGroupOne.setSalience(99);
        weightGroupOne.addElement(new RtdDroolsElement("pr_rule_01", 0.5));
        weightGroupOne.addElement(new RtdDroolsElement("pr_rule_02", 0.3));
        RtdDroolsElementsGroup weightGroupTwo = new RtdDroolsElementsGroup("drools_group_02",
AlgorithmType.WEIGHTING.name());
        weightGroupTwo.setSalience(99);
        weightGroupTwo.addElement(new RtdDroolsElement("pr_rule_02"));
        weightGroupTwo.addElement(new RtdDroolsElement("pr_rule_03"));
        RtdDroolsElementsGroup averageGroup = new RtdDroolsElementsGroup("drools_group_03",
AlgorithmType.AVERAGE.name());
        averageGroup.setSalience(99);
        averageGroup.addElement(new RtdDroolsElement("pr_rule_02"));
        averageGroup.addElement(new RtdDroolsElement("pr_rule_03"));
        averageGroup.addElement(new RtdDroolsElement("pr_rule_04"));
        RtdDroolsElementsGroup maximumGroup = new RtdDroolsElementsGroup("drools_group_04",
AlgorithmType.MAXIMUM.name());
        maximumGroup.setSalience(99);
        maximumGroup.addElement(new RtdDroolsElement("pr_rule_01"));
        maximumGroup.addElement(new RtdDroolsElement("pr_rule_02"));
        maximumGroup.addElement(new RtdDroolsElement("pr_rule_03"));
        RtdDroolsElementsGroup minimumGroup = new RtdDroolsElementsGroup("drools_group_05",
AlgorithmType.MINIMUM.name());
        minimumGroup.setSalience(99);
        minimumGroup.addElement(new RtdDroolsElement("pr_rule_01"));
        minimumGroup.addElement(new RtdDroolsElement("pr_rule_02"));
        minimumGroup.addElement(new RtdDroolsElement("pr_rule_03"));
        droolsDecision.addElementsGroup(weightGroupOne);
        droolsDecision.addElementsGroup(weightGroupTwo);
        droolsDecision.addElementsGroup(averageGroup);
        droolsDecision.addElementsGroup(maximumGroup);
        droolsDecision.addElementsGroup(minimumGroup);
    }
    @After
}
```

```
public void tearDown() {  
    if (kieContainer != null) {  
        kieContainer.dispose();  
    }  
  
    kieContainer = null;  
}  
  
/**  
 * Run the following commands to generate a decision engine file on the local host:  
 */  
@Test  
public void input_invalid_contract_json_and_out_put_zip_file_then_compare_content_success() {  
  
    RtdDroolsFileGenerator generator = RtdDroolsFileGenerator.INSTANCE;  
    String jsonContract = JSONObject.toJSONString(droolsDecision);  
    File result = null;  
    try {  
        // Specify the directory to be downloaded.  
        File tempDirectory = new File("d:/");  
        result = generator.outputZipFile(jsonContract, tempDirectory.getPath(), "rtd-drools-example.zip");  
    } catch (Exception e) {  
        e.printStackTrace();  
        Assert.fail("Create zip file fail ....");  
    }  
  
    // ZipInputStream zipln = null;  
    // try {  
    //     zipln = new ZipInputStream(new BufferedInputStream(new FileInputStream(result)));  
    //     compareZipFileContentOneByOne(zipln);  
    // } catch (Exception e) {  
    //     Assert.fail("Unzip file fail ....");  
    // } finally {  
    //     ZipFileUtil.closeIOResource(zipln);  
    // }  
    //  
    // boolean deleteSuccess = result.delete();  
    // Assert.assertTrue(deleteSuccess);  
}  
  
@Test  
public void input_invalid_contract_json_and_out_put_zip_stream_then_compare_content_success() {  
  
    RtdDroolsFileGenerator generator = RtdDroolsFileGenerator.INSTANCE;  
    String jsonContract = JSONObject.toJSONString(droolsDecision);  
    ByteArrayOutputStream outputBaos = generator.outputZipStream(jsonContract);  
    ByteArrayInputStream bais = new ByteArrayInputStream(outputBaos.toByteArray());  
    ZipInputStream zipln = null;  
    try {  
        zipln = new ZipInputStream(new BufferedInputStream(bais));  
        compareZipFileContentOneByOne(zipln);  
    } catch (Exception e) {  
        Assert.fail("Unzip file fail ....");  
    } finally {  
        ZipFileUtil.closeIOResource(zipln);  
    }  
}  
  
private void compareZipFileContentOneByOne(ZipInputStream zipln) throws IOException {  
    for (ZipEntry zipEntry; (zipEntry = zipln.getNextEntry()) != null; ) {  
        ByteArrayOutputStream baos = new ByteArrayOutputStream();  
        byte data[] = new byte[BUFFER];  
        for (int count; (count = zipln.read(data, 0, BUFFER)) != -1; ) {  
            baos.write(data, 0, count);  
        }  
    }  
}
```

```
String fileContent = new String(baos.toByteArray());
ZipFileUtil.closeIOResource(baos);
if (zipEntry.getName().equals(RTD_DROOLS_K_MODULE_XML)) {
    Assert.assertTrue(fileContent.contains("rtdDecisionStateless"));
} else if (zipEntry.getName().equals(RTD_META_DATA_PROPERTIES)) {
    Assert.assertTrue(fileContent.contains("drools_group_01"));
} else if (zipEntry.getName().startsWith(RTD_DROOLS_FILE_PREFIX)) {
    Assert.assertTrue(fileContent.contains("package com.huawei.fi.rtd.drools.common"));
} else if (zipEntry.getName().equals(RTD_DROOLS_DECISION_JSON)) {
    Assert.assertTrue(StringUtils.isNotBlank(fileContent));
} else {
    Assert.fail("Invalid file name ....");
}
}

}
```

## Sample Description

- Obtain all rule names of the current event source and specify the algorithm to be decided in the `setup()` method. For example, obtain the minimum score of rules `pr_rule_01`, `pr_rule_02` and `pr_rule_03`.

```
RtdDroolsElementsGroup minimumGroup = new RtdDroolsElementsGroup("drools_group_05",
AlgorithmType.MINIMUM.name());
minimumGroup.setSalience(99);
minimumGroup.addElement(new RtdDroolsElement("pr_rule_01"));
minimumGroup.addElement(new RtdDroolsElement("pr_rule_02"));
minimumGroup.addElement(new RtdDroolsElement("pr_rule_03"));
```

- Run the `input_invalid_contract_json_and_out_put_zip_file_then_compare_content_success()` command to download the file to the local host after compiling the custom decision-making code. Specify the download directory and comment out the following code for deleting the compressed package when you download the file.

```
/*
 * Run the following commands to generate a decision engine file on the local host:
 */
@Test
public void input_invalid_contract_json_and_out_put_zip_file_then_compare_content_success() {

RtdDroolsFileGenerator generator = RtdDroolsFileGenerator.INSTANCE;
String jsonContract = JSONObject.toJSONString(droolsDecision);
File result = null;
try {
    // Specify the directory to be downloaded.
    File tempDirectory = new File("d:/");
    result = generator.outputZipFile(jsonContract, tempDirectory.getPath(), "rtd-drools-
example.zip");
} catch (Exception e) {
    e.printStackTrace();
    Assert.fail("Create zip file fail ....");
}

// ZipInputStream zipIn = null;
// try {
//     zipIn = new ZipInputStream(new BufferedInputStream(new FileInputStream(result)));
//     compareZipFileContentOneByOne(zipIn);
// } catch (Exception e) {
//     Assert.fail("Unzip file fail ....");
// } finally {
//     ZipFileUtil.closeIOResource(zipIn);
// }
// boolean deleteSuccess = result.delete();
```

```
//     Assert.assertTrue(deleteSuccess);  
}
```

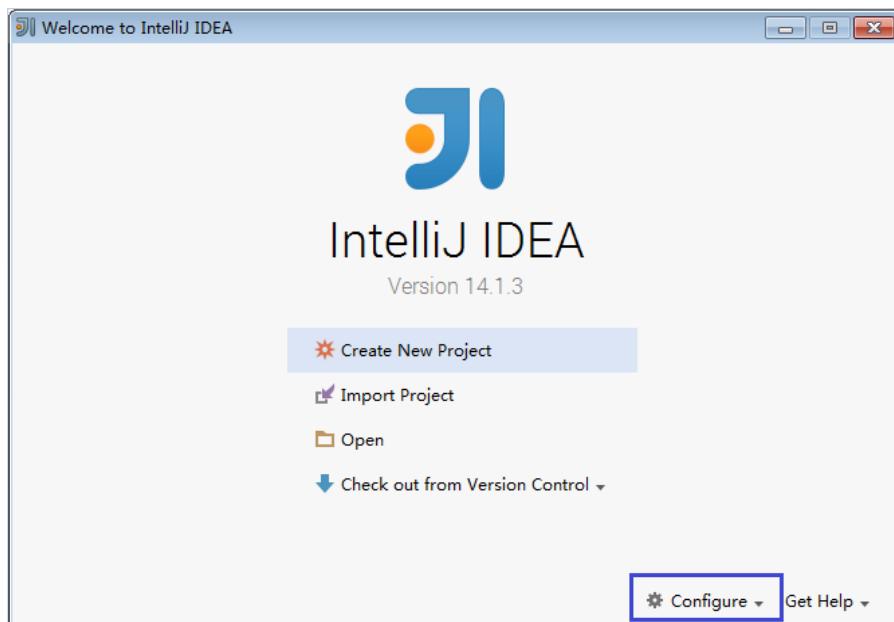
## 2.15.4 Commissioning the Application

### 2.15.4.1 Compiling and Packaging

**Step 1** Before importing the sample project, configure JDK for IntelliJ IDEA.

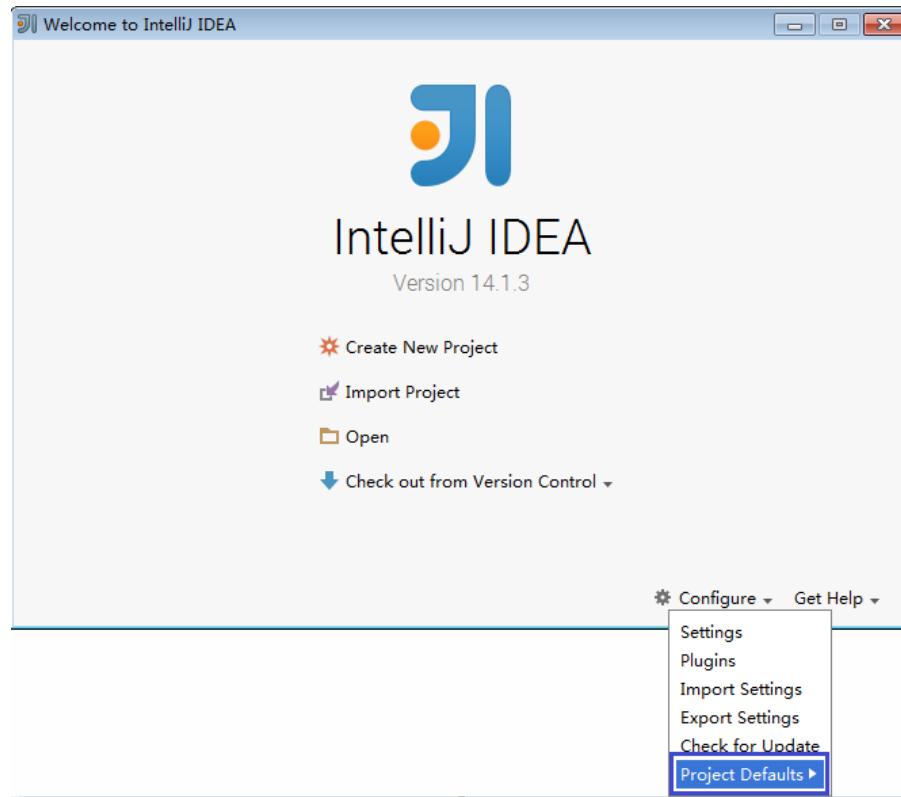
1. Start IntelliJ IDEA and click **Configure**.

**Figure 2-242** Choosing Configure



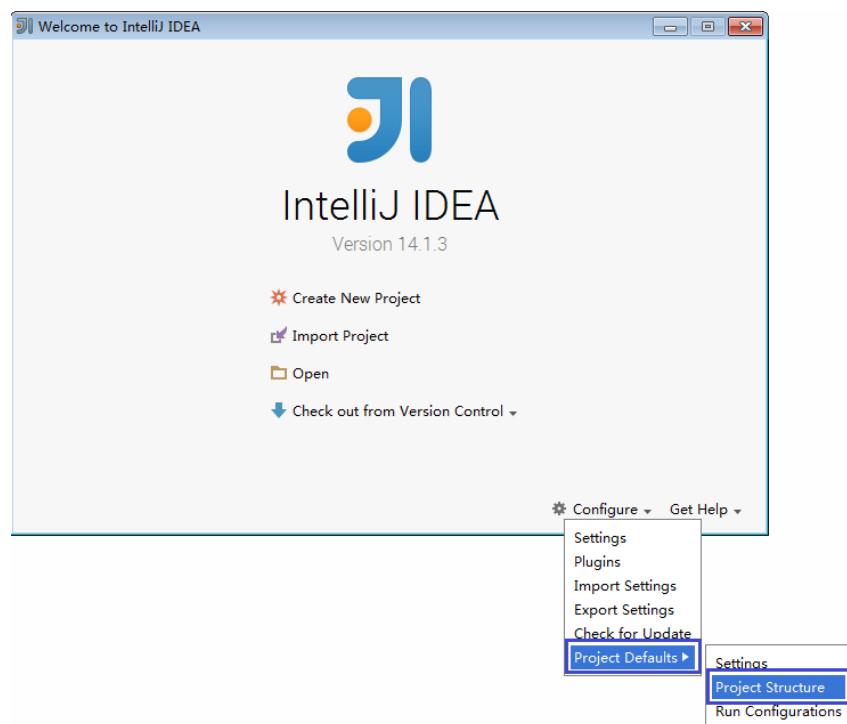
2. Choose **Project Defaults** from the **Configure** drop-down list.

Figure 2-243 Choosing Project Defaults



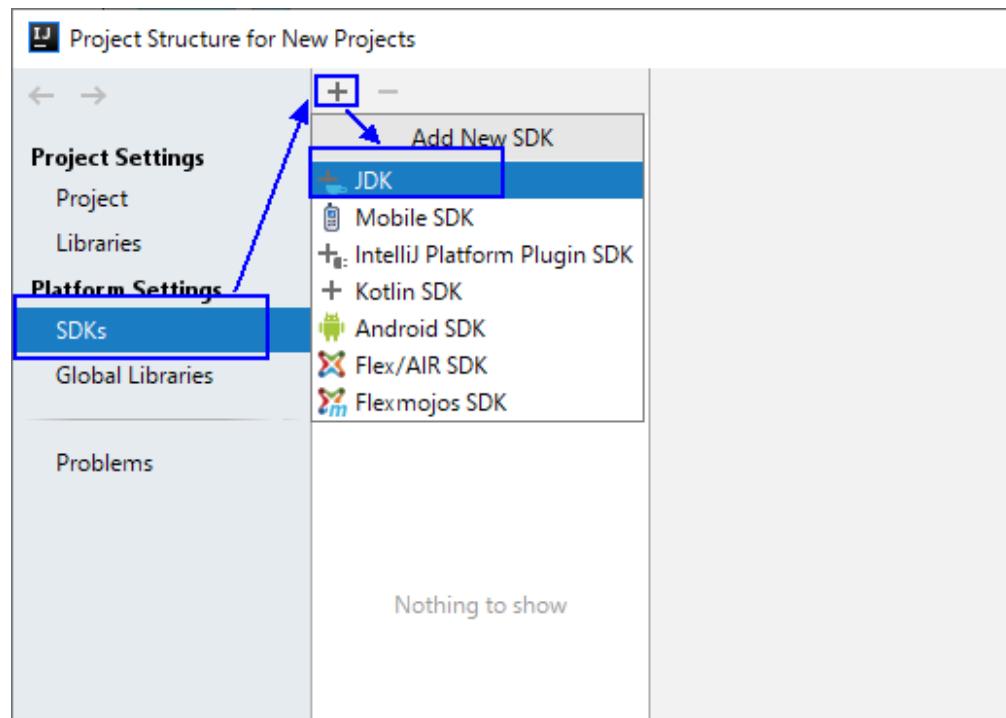
3. Select **Project Structure** from **Project Defaults**.

Figure 2-244 Project Defaults



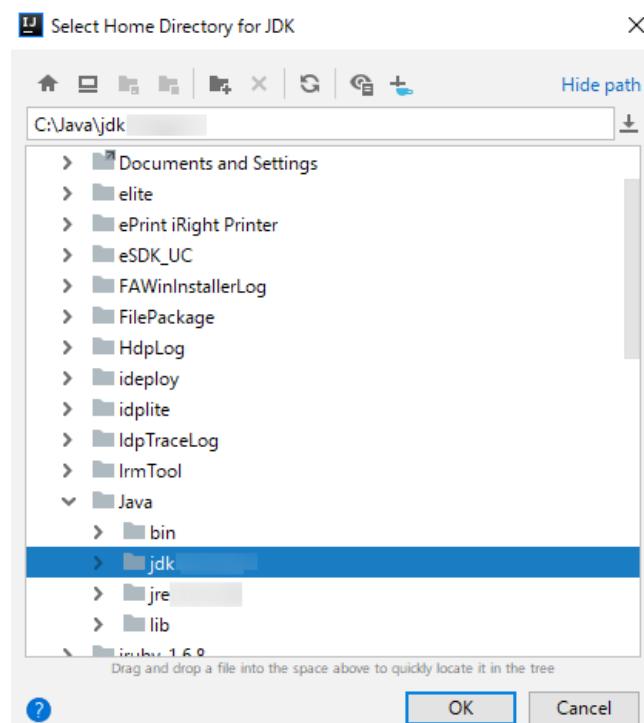
4. On the **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 2-245 Adding the JDK



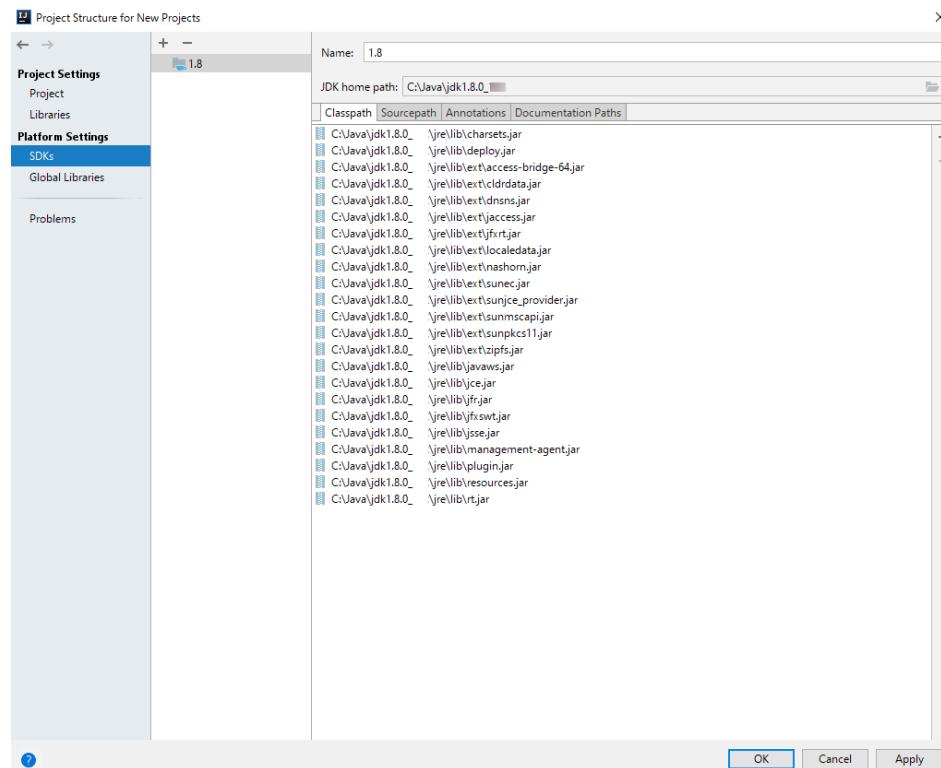
5. On the displayed **Select Home Directory for JDK** page, select the JDK directory and click **OK**.

Figure 2-246 Selecting the JDK directory



6. After selecting the JDK, click **OK** to complete the configuration.

Figure 2-247 Completing the configuration



**Step 2** Import the Java sample projects to the IDEA.

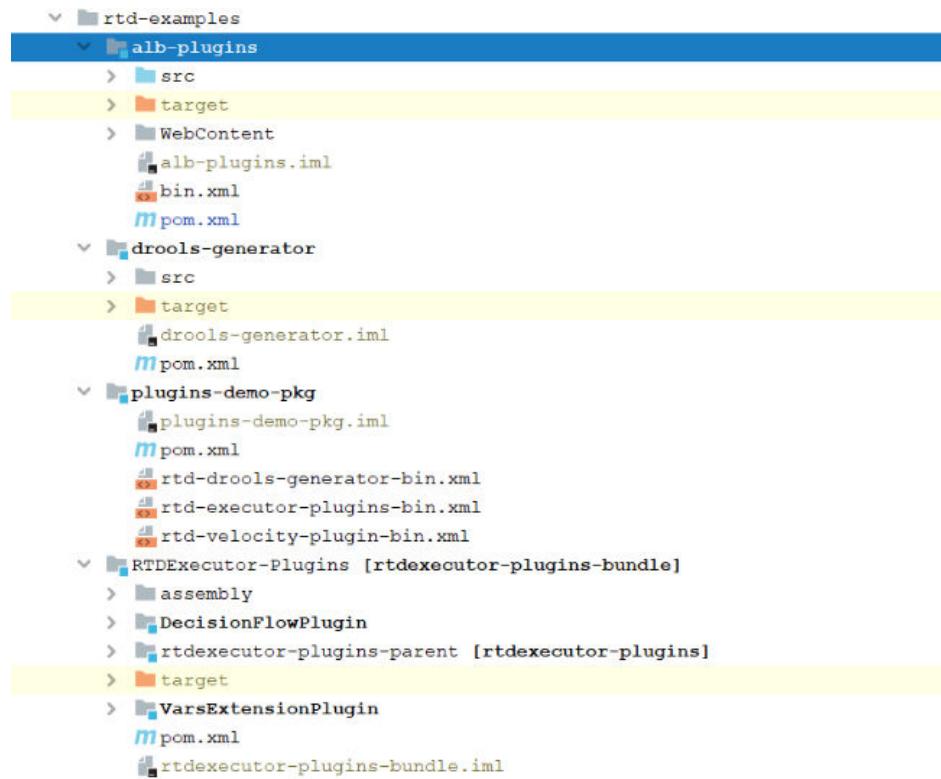
1. Start the IntelliJ IDEA. On the **Quick Start** page, select **Import Project**. Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Select **File > Import project...** to import projects.

Figure 2-248 Importing a project (on the Quick Start page)



2. Select the directory for storing the imported projects and click **OK**.

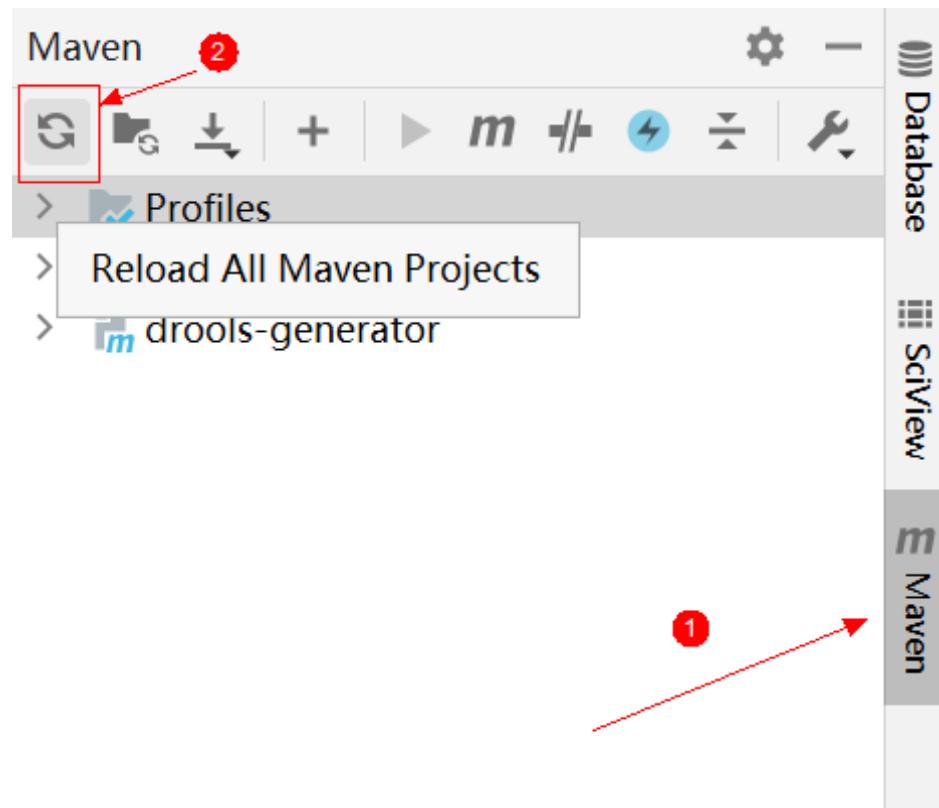
Figure 2-249 Select File or Directory to Import



3. Retain the default dependency library that is automatically recognized by IDEA and the suggested module structure, and click **Next**.

**Step 3** In IntelliJ IDEA, click **Reload All Maven Projects** in the Maven window on the right to import Maven project dependencies.

Figure 2-250 Reload projects

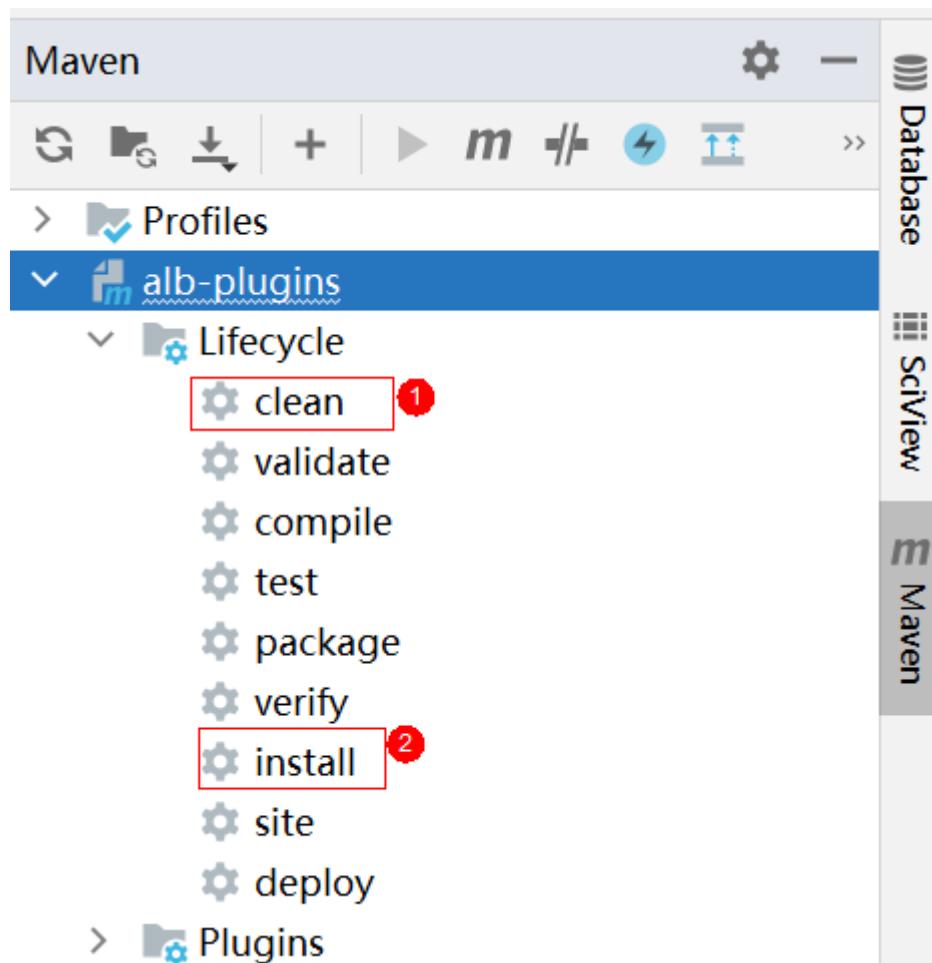


**Step 4** Compile the application.

There are two compilation methods:

- Method 1
  - a. Choose **Maven**, locate the target project name, and double-click **clean** under **Lifecycle** to run the **clean** command of Maven.
  - b. Choose **Maven**, locate the target project name, and double-click **install** under **Lifecycle** to run the **install** command of Maven.

Figure 2-251 Maven clean and install commands



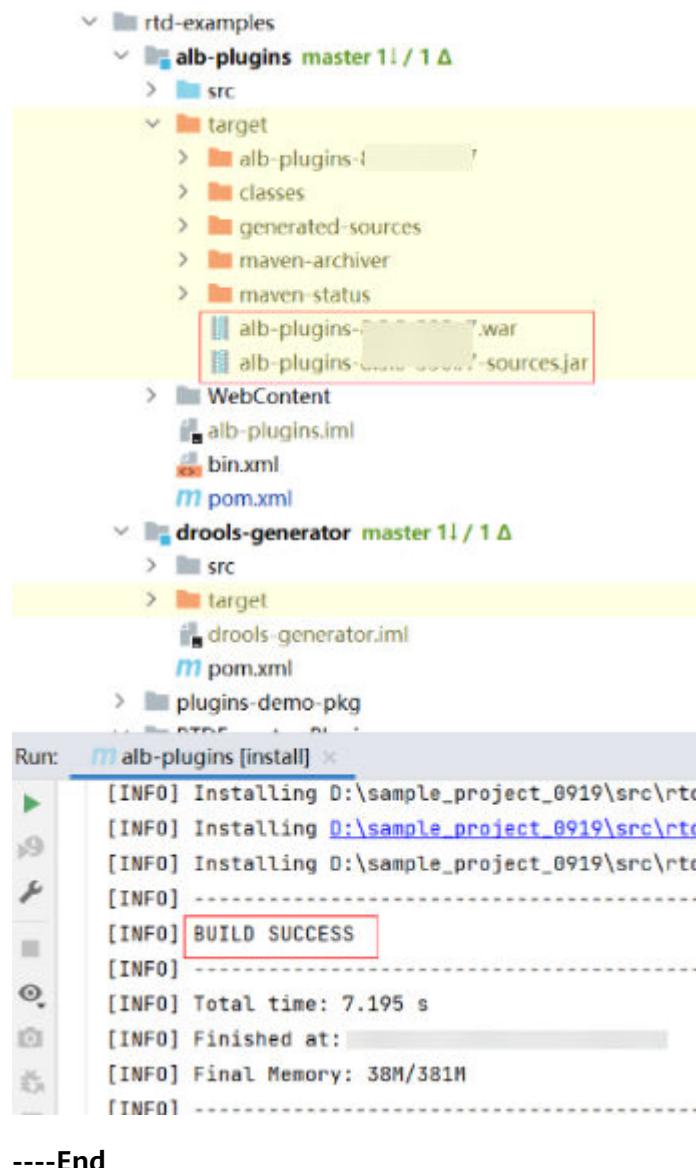
- Method 2: Go to the directory where the **pom.xml** file is located in the **Terminal** window of IDEA, and run the **mvn clean install** command to build the file.

Figure 2-252 Entering "mvn clean install" to Idea Terminal

The screenshot shows the IntelliJ IDEA terminal window. The command 'mvn clean install' is typed into the terminal. The terminal tab is highlighted with a red box. The bottom navigation bar shows tabs for 'Git', 'Run', 'Terminal' (highlighted), 'TODO', 'Problems', 'UI Studio', 'Profiler', and 'SpotBugs'.

**Step 5** Obtain the JAR or WAR package from the **target** directory which is generated after "BUILD SUCCESS" is displayed.

Figure 2-253 Compilation completed



## 2.16 Solr Development Guide

### 2.16.1 Overview

#### 2.16.1.1 Application Development Overview

##### Solr Introduction

Solr is an independent enterprise search server based on Apache Lucene. It provides Representational State Transfer (REST)-similar Hypertext Transfer Protocol (HTTP)/Extensible Markup Language (XML) and JavaScript object notation (JSON) application programming interfaces (APIs). Main functions of Solr include full-text search, hit highlighting, faceted search, near real-time indexing,

dynamic clustering, database integration, rich document (for example, Word and PDF) handling, and geographic information search.

As an outstanding enterprise search server, Solr supports the following features:

- Advanced full-text search function
- Optimized large-capacity network traffic
- Standard open APIs, including XML, JSON, and HTTP
- Comprehensive HTML management interface
- Java management extensions (JMX) for monitoring servers
- Linear scalability, automatic index replication, and automatic failover and restoration
- Near real-time indexing
- XML configuration enabling flexibility and adaptability
- Extensible plug-in architecture

### 2.16.1.2 Common Concepts

Solr can be deployed in multiple modes, such as single-node mode, master-slave mode, and cloud mode. This document describes the SolrCloud deployment mode. SolrCloud is a distributed search solution in Solr4.0 based on Solr and ZooKeeper.

- **Collection**

Collection is a complete logical index in a SolrCloud cluster. A Collection can be divided into multiple Shards that use the same Config Set.

- **Config Set**

Config Set is a group of configuration files required by Solr Cores to provide services. A Config Set includes solrconfig.xml (SolrConfigXml) and schema.xml (SchemaXml).

- **Core**

Core refers to Solr Core. A Solr includes one or multiple Solr Cores. Each Solr Core independently provides indexing and query functions. Each Solr Core corresponds to an index or a Collection Shard. In SolrCloud, Solr configurations are stored in ZooKeeper.

- **Replica**

Replica is a copy of a Shard. Each Replica is in a Solr Core.

- **Shard**

Shard is a logical section of a Collection. Each Shard has one or multiple replicas, among which a leader is elected.

- **Leader**

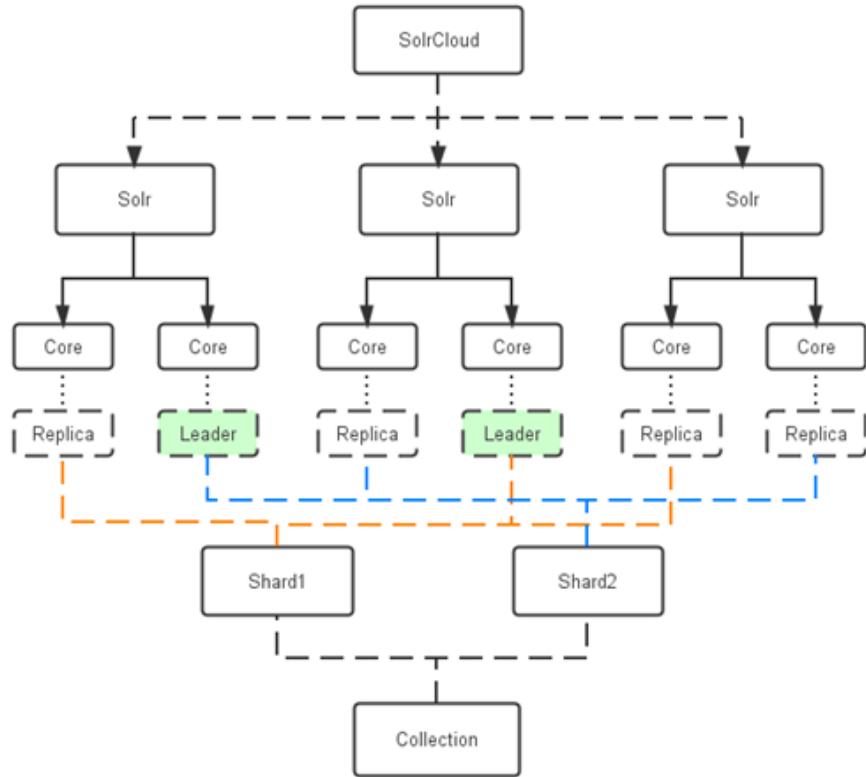
Leader is a Shard replica elected from multiple replicas. When documents are indexed, SolrCloud transfers them to the leader, and the leader distributes them to replicas of all Shards.

- **Zookeeper**

ZooKeeper is mandatory in SolrCloud. It provides distributed lock and leader election functions.

- **Collection and Solr entities**

**Figure 2-254 Collection and Solr entities**



### 2.16.1.3 Development Process

This document describes HBase application development based on the Java API.

For information about each phase in the development process, see [Figure 2-255](#) and [Table 2-153](#).

Figure 2-255 Solr application development process

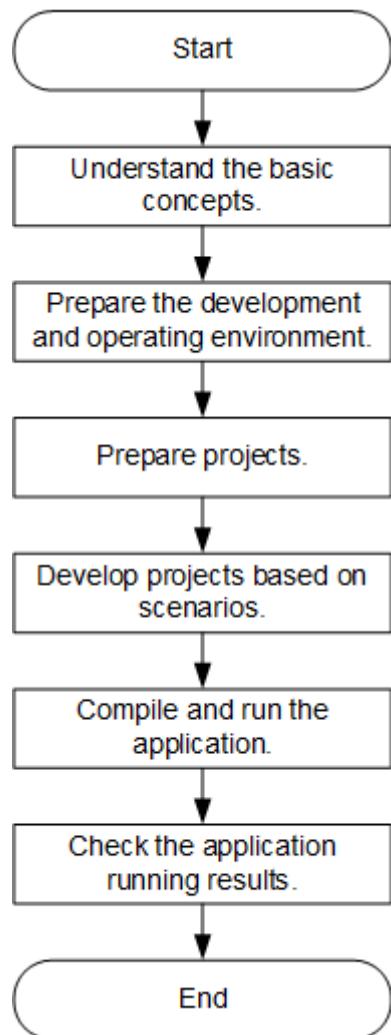


Table 2-153 Solr application development process description

Phase	Description	Reference Document
Understand the basic concepts.	Before developing an application, learn basic concepts of Solr and understand the scenario requirements and design tables.	<a href="#">Common Concepts</a>

Phase	Description	Reference Document
Prepare the development and operating environment.	The Java language is recommended for the development of Solr applications. The IntelliJ IDEA tool can be used. The Solr running environment is the Solr client. Install and configure the client according to the guide.	<a href="#">Development and Operating Environment</a>
Prepare projects.	Import an example project according to the guide.	<a href="#">Configuring and Importing Sample Projects</a>
Develop projects based on scenarios.	Provide an example project using Java language, covering an entire process for example project from query index to delete index.	<a href="#">Developing an Application</a>
Compile and run the application.	Compile the developed application and submit it for running.	<a href="#">Application Commissioning</a>
View application running results.	Application running results are written into a specified path.	<a href="#">Application Commissioning</a>

## 2.16.2 Environment Preparation

### 2.16.2.1 Development and Operating Environment

#### Preparing Development Environment

[Table 2-154](#) describes the environment required for secondary development.

**Table 2-154** Development environment

Item	Description
OS	<ul style="list-style-type: none"><li>• Development environment: Windows OS. Windows 7 or later is supported.</li><li>• Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the running environment must be able to communicate with the cluster service plane network.</li></ul>
Installing JDK	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>• For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- Oracle JDK 1.8</li><li>- IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>• For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>- OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>- BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• For security purposes, the server supports only TLS V1.2 or later.</li><li>• By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstance-tls-oracle#matchsslcontext_tls</a>.</li><li>• For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>
Installing and configuring IntelliJ IDEA	<p>Tool used for developing HBase applications. The version must be 2019.1 or other compatible version.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>

Item	Description
JUnit plug-in installation	Basic configuration for the development environment.
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
7-zip	Used to decompress .zip and .rar packages. 7-Zip 16.04 is supported.

## Preparing an Operating Environment

During application development, you need to prepare the code running and commissioning environment to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. **Accessing FusionInsight Manager of an MRS Cluster.** In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Complete Client**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file. Then, decompress **FusionInsight\_Cluster\_1\_Services\_ClientConfig.tar** file to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig** directory on the local PC. The directory name cannot contain spaces.

    - b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\Solr\config** and manually import the configuration file to the configuration file directory (usually the **conf** folder) of the Solr sample project.
    - c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

### NOTE

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client on the node. For example, the client installation directory is **/opt/hadoopclient**.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig \Solr\config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Solr/config/* root@IP address of the client node:/opt/hadoopclient/conf
```
  - c. Check the network connection of the client node.  
During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

## 2.16.2.2 Configuring and Importing Sample Projects

### Background

Obtain the Solr development example project. Import the project to IntelliJ IDEA for learning.

### Prerequisites

Ensure that the difference between the local PC time and the MRS cluster time is less than 5 minutes. If the time difference cannot be determined, contact the system administrator. You can view the MRS cluster time in the lower-right corner on FusionInsight Manager.

## Procedure

**Step 1** Obtain the sample project folder **solr-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).

**Step 2** Import the example project to the IntelliJ IDEA development environment.

1. Choose **File > Open... > Open File or Project**.
2. Enter the path of the solr-example sample project.
3. Select the solr-example example project folder, and click **OK**.

**Step 3** Change the **zkHost** value in the **solr-example.properties** to the **-DzkHost** value queried on the Dashboard on the **Solr Admin** page.

Log in to FusionInsight Manager as a user who has the Solr page access permission, choose **Cluster > Services > Solr**, and click the link next to **Solr WebUI**. The Solr Admin page is displayed.

**Step 4** Change the **zkSslEnable** value in the **solr-example.properties** to the value of **ssl.enabled** of ZooKeeper.

Log in to FusionInsight Manager, choose **Cluster > Services > ZooKeeper > Configurations > All Configurations**, search for and record the value of **ssl.enabled** of the ZooKeeper service.

**Step 5** Change the **createNodeSet** value in the **solr-example.properties** to the IP address and port number of the Solr instance. Use commas (,) to separate multiple values.

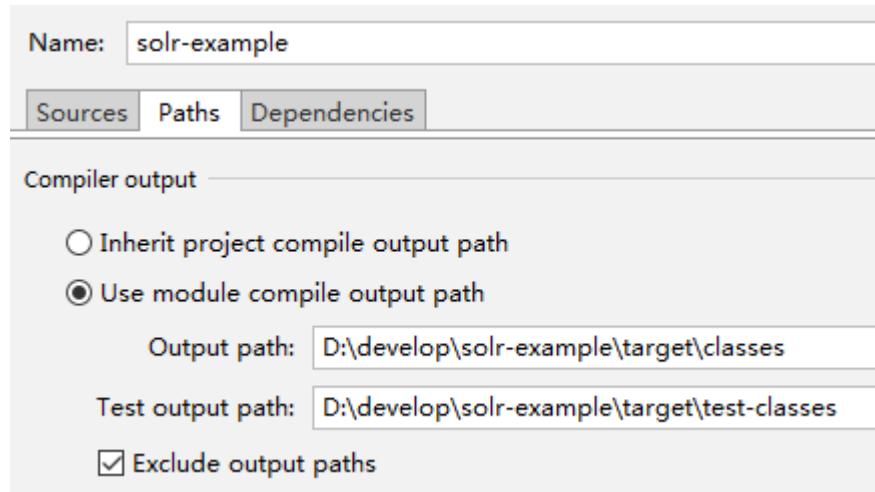
Log in to FusionInsight Manager, choose **Cluster > Services > Solr > Instance**, and view the service IP address of the SolrServer instance. The default port number is **21100**.

**Step 6** Change the **SOLR\_KBS\_ENABLED** value in the **solr-example.properties** to **false**.

The configuration example of the **solr-example.properties** file is as follows:

```
zkHost=192.168.1.189:24002,192.168.1.228:24002,192.168.1.150:24002/solr
zkSslEnable=false
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL=
principal=
zkClientTimeout=20000
zkConnectTimeout=30000
SOLR_KBS_ENABLED=false
COLLECTION_NAME=col_test
DEFAULT_CONFIG_NAME=confWithSchema
shardNum=3
replicaNum=1
maxShardsPerNode=1
autoAddReplicas=false
assignToSpecifiedNodeSet=false
createNodeSet=192.168.1.136:21100_solr,192.168.1.228:21100_solr
isNeedZkClientConfig=true
```

**Step 7** Before running the **solr-example** sample project, configure **Compiler output** for the project. Otherwise, the error message "the output path is not specified" may be displayed. Specifically, choose **File > Project Structure > Modules** and configure the compilation output path of the **solr-example** sample project, as shown in the following figure.



#### NOTE

You can customize **Compiler output**. Ensure that the path exists. To change the output path during the project running, create the required path, and then configure the path by following the instruction in the preceding figure. The modification takes effect after the compilation.

----End

## 2.16.3 Developing an Application

### 2.16.3.1 Typical Scenario Description

Based on typical scenarios, users can quickly learn and master the Solr development process and know important interface functions.

#### Scenario

Provided that a user needs an application to manage employees and search employee personal information. Solr can provide the searching service. The search process is as follows:

- Check whether the index to be created exists. If the index exists, delete it.
- Create an index for personal information data of employees.
- Insert the personal information data of employees into the index.
- Query personal information of employees based on the search criteria.
- Delete personal information of employees based on the search criteria.

#### NOTE

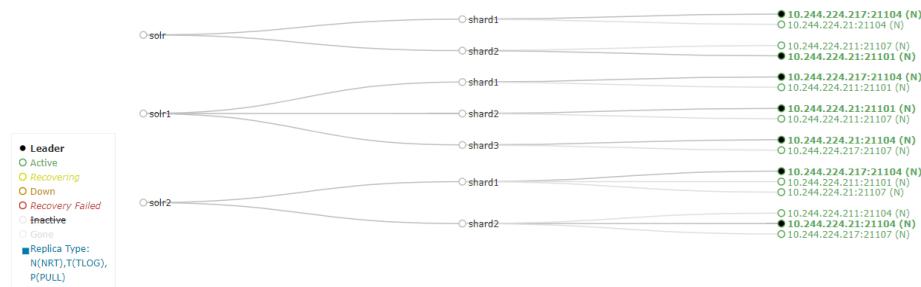
The schema information about the column corresponding to the data table of employees needs to be planned in Solr index configurations.

### 2.16.3.2 Development Idea

The core function of Solr is search. You need to create a Collection before searching. This topic uses the SolrCloud deployment mode as an example. In the

following figure, two collections are created, in which three instances of collection 1 are distributed over three nodes and each shard in collection 2 has two replicas, as shown in [Figure 2-256](#).

**Figure 2-256** Collection relationships



Operations involved in example codes include establishing a connection with the Solr server, querying a Collection, deleting a Collection, creating a Collection, adding one document, adding multiple documents, simple search, and deleting a document. You can learn advanced Solr operations based on basic example codes.

Example codes are described in the following sequence:

1. Establish a connection between the client and the CloudSolr server.
2. Querying a Collection;
3. Deleting a Collection;
4. Creating a Collection;
5. Add one or multiple documents.
6. Query documents.
7. Delete documents.
8. Release the CloudSolr connection.

#### NOTE

In this development guide, the used Solr configuration files are all built-in. If users want to customize the configuration set, create and upload the configuration set.

### 2.16.3.3 Example Code Description

#### 2.16.3.3.1 Initializing Solr

##### Function

Solr initialization is a prerequisite for using application programming interfaces (APIs) provided by Solr. Solr initialization aims to connect to SolrCloud.

#### NOTE

Call `cloudSolrClient.close ()` to close requested resources after the Solr operation is complete.

## Example Code

The following code snippet belongs to the **getCloudSolrClient** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
/*
 *Initialize CloudSolrClient instance, and collect SolrCloud
 */
private CloudSolrClient getCloudSolrClient(String zkHost) throws SolrException {
    Builder builder = new CloudSolrClient.Builder();
    builder.withZkHost(zkHost);
    CloudSolrClient cloudSolrClient = builder.build();
    cloudSolrClient.setZkClientTimeout(zkClientTimeout);
    cloudSolrClient.setZkConnectTimeout(zkConnectTimeout);
    cloudSolrClient.connect();
    LOG.info("The cloud Server has been connected !!!!");
    ZkStateReader zkStateReader = cloudSolrClient.getZkStateReader();
    ClusterState cloudState = zkStateReader.getClusterState();
    LOG.info("The zookeeper state is : {}", cloudState);
    return cloudSolrClient;
}
```

## Precautions

The three values **zkHost**, **zkClientTimeout**, and **zkConnectTimeout** can be configured in the **solr-example.properties** file under the **conf** directory. Change the **zkHost** value in the **solr-example.properties** to the **-DzkHost** value queried on the Dashboard of the Solr Admin page.

### 2.16.3.3.2 Querying a Collection

#### Function

By calling **process (cloudSolrClient)** in the **CollectionAdminRequest.List** and the returned responses, users can obtain the names of all collections.

## Example Code

The following code snippet belongs to the **queryAllCollections** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private List<String> queryAllCollections(CloudSolrClient
    cloudSolrClient) throws SolrException {
    CollectionAdminRequest.List list = new CollectionAdminRequest.List();

    CollectionAdminResponse listRes = null;
    try {
        listRes = list.process(cloudSolrClient);
    } catch (SolrServerException | IOException e) {
        LOG.error("Failed to list collection", e);
        throw new SolrException("Failed to list collection");
    } catch (Exception e) {
        LOG.error("Failed to list collection", e);
        throw new SolrException("unknown exception");
    }

    List<String> collectionNames = (List<String>)
    listRes.getResponse().get("collections");
    LOG.info("All existed collections : {}", collectionNames);
    return collectionNames;
}
```

### 2.16.3.3.3 Deleting a Collection

#### Function

By calling **process (cloudSolrClient)** in the **CollectionAdminRequest.Deleted** and the returned responses, users can check whether the deleting collection operation is successfully executed.

#### Example Code

The following code snippet belongs to the **createCollection** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void deleteCollection(CloudSolrClient cloudSolrClient) throws SolrException {
    CollectionAdminRequest.Delete delete = CollectionAdminRequest.deleteCollection(collectionName);
    CollectionAdminResponse response = null;
    try {
        response = delete.process(cloudSolrClient);
    } catch (SolrServerException | IOException e) {
        LOG.error("Failed to delete collection", e);
        throw new SolrException("Failed to create collection");
    } catch (Exception e) {
        LOG.error("Failed to delete collection", e);
        throw new SolrException("unknown exception");
    }
    if (response.isSuccess()) {
        LOG.info("Success to delete collection[{}]", collectionName, COLLECTION_NAME);
    } else {
        LOG.error("Failed to delete collection[{}], cause : {}", collectionName, response.getErrorMessages());
        throw new SolrException("Failed to delete collection");
    }
}
```

### 2.16.3.3.4 Creating a Collection

#### Function

By calling **process (cloudSolrClient)** in the **CollectionAdminRequest.Create** and the returned responses, users can check whether the creating collection operation is successfully executed.

#### Example Code

The following code snippet belongs to the **createCollection** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void createCollection(CloudSolrClient cloudSolrClient) throws SolrException {
    CollectionAdminRequest.Create create = CollectionAdminRequest.createCollection(collectionName,
        defaultConfigName, shardNum, replicaNum);
    CollectionAdminResponse response = null;
    try {
        response = create.process(cloudSolrClient);
    } catch (SolrServerException e) {
        LOG.error("Failed to create collection", e);
        throw new SolrException("Failed to create collection");
    } catch (IOException e) {
        LOG.error("Failed to create collection", e);
        throw new SolrException("Failed to create collection");
    } catch (Exception e) {
        LOG.error("Failed to create collection", e);
        throw new SolrException("unknown exception");
    }
}
```

```
        }
        if (response.isSuccess()) {
            LOG.info("Success to create collection[{}]", collectionName);
        } else {
            LOG.error("Failed to create collection[{}], cause : {}", collectionName, response.getErrorMessages());
            throw new SolrException("Failed to create collection");
        }
    }
```

## Precautions

1. **collectionName** is the name of the index to be created. **defaultConfigName** is the configuration set used by the index. The configuration set is the default Solr confWithSchema. Users can run the client script to download the configuration client information or view the information on the web page of the Solr Admin. **shardNum** indicates the number of fragments. **replicaNum** indicates the number of copies. All of these items can be configured in the **solr-example.properties** file under the **conf** directory and the configuration sets can be changed based on practical application.
2. It is suggested to limit the alias name and collection name **collectionName** at 255 characters while creating a collection, otherwise it might affect the performance of ZooKeeper and Solr. For more information please refer to Solr service configuration **SOLR\_COLLECTION\_CORE\_MAX\_LENGTH**.

### 2.16.3.3.5 Adding a Document

## Function

The index data is added by calling the adding method of cloudSolrClient or by constructing UpdateRequest to call request method of cloudSolrClient.

### Example Code 1

The following code snippet belongs to the **addDocs** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void addDocs(CloudSolrClient cloudSolrClient) throws SolrException {
    Collection<SolrInputDocument> documents = new ArrayList<SolrInputDocument>();
    for (Integer i = 0; i < 5; i++) {
        SolrInputDocument doc = new SolrInputDocument();
        doc.addField("id", i.toString());
        doc.addField("name", "Luna_" + i);
        doc.addField("features", "test" + i);
        doc.addField("price", (float) i * 1.01);
        documents.add(doc);
    }
    try {
        cloudSolrClient.add(documents);
        LOG.info("success to add index");
    } catch (SolrServerException e) {
        LOG.error("Failed to add document to collection", e);
        throw new SolrException("Failed to add document to collection");
    } catch (IOException e) {
        LOG.error("Failed to add document to collection", e);
        throw new SolrException("Failed to add document to collection");
    } catch (Exception e) {
        LOG.error("Failed to add document to collection", e);
        throw new SolrException("unknown exception");
    }
}
```

## Example Code 2

The following code snippet belongs to the **addDocs2** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void addDocs2(CloudSolrClient cloudSolrClient) throws  
SolrException{  
    UpdateRequest request = new UpdateRequest();  
    Collection<SolrInputDocument> documents = new ArrayList<>();  
    for (Integer i = 5; i < 10; i++) {  
        SolrInputDocument doc = new SolrInputDocument();  
        doc.addField("id", i.toString());  
        doc.addField("name", "Tom" + i);  
        doc.addField("features", "test" + i);  
        doc.addField("price", (float) i * 1.01);  
        documents.add(doc);  
    }  
    request.add(documents);  
    try {  
        cloudSolrClient.request(request);  
        cloudSolrClient.commit();  
    } catch (SolrServerException | IOException e) {  
        LOG.error("Failed to add document to collection", e);  
        throw new SolrException("Failed to add document to  
collection");  
    }  
}
```

### 2.16.3.3.6 Querying a Document

## Function

The index data can be queried by constructing a SolrQuery instance and calling the `cloudSolrClient.query` API.

## Example Code

The following code snippet belongs to the **queryIndex** method in the **TestSample** class of the **com.huawei.fusioninsight.solr.example** package.

```
private void queryIndex(CloudSolrClient cloudSolrClient) throws SolrException {  
    SolrQuery query = new SolrQuery();  
    query.setQuery("name:Luna*");  
  
    try {  
        QueryResponse response = cloudSolrClient.query(query);  
        SolrDocumentList docs = response.getResults();  
        LOG.info("Query wasted time : {}ms", response.getQTime());  
  
        LOG.info("Total doc num : {}", docs.getNumFound());  
        for (SolrDocument doc : docs) {  
            LOG.info("doc detail : " + doc.getFieldValueMap());  
        }  
    } catch (SolrServerException e) {  
        LOG.error("Failed to query document", e);  
        throw new SolrException("Failed to query document");  
    } catch (IOException e) {  
        LOG.error("Failed to query document", e);  
        throw new SolrException("Failed to query document");  
    } catch (Exception e) {  
        LOG.error("Failed to query document", e);  
        throw new SolrException("unknown exception");  
    }  
}
```

### 2.16.3.3.7 Deleting a Document

#### Function

The data of a specified index can be deleted by calling `cloudSolrClient.deleteByQuery`.

#### Example Code

The following code snippet belongs to the `removeIndex` method in the `TestSample` class of the `com.huawei.fusioninsight.solr.example` package.

```
private void removeIndex(CloudSolrClient cloudSolrClient) throws SolrException {
    try {
        cloudSolrClient.deleteByQuery("*:*");
        cloudSolrClient.commit();
        LOG.info("Success to delete index");
    } catch (SolrServerException | IOException e){
        LOG.error("Failed to remove document", e);
        throw new SolrException("Failed to remove document");
    }
}
```

## 2.16.4 Application Commissioning

### 2.16.4.1 Commissioning an Application in Windows

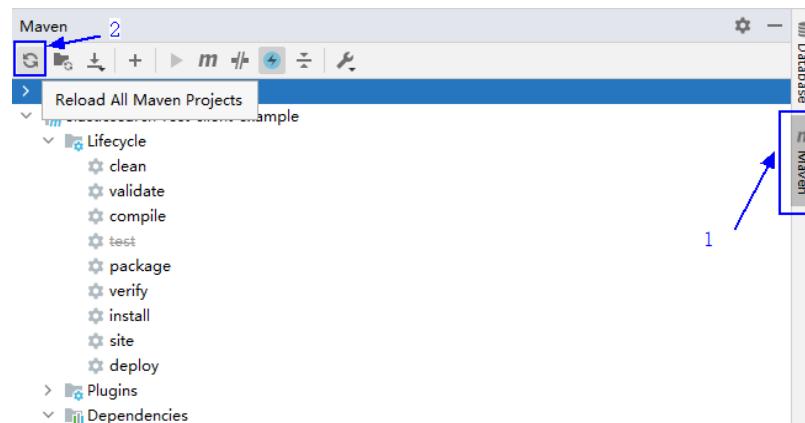
#### 2.16.4.1.1 Compiling and Running an Application

##### Scenario

After the code development is complete, you can run the application in the Windows development environment. If the network between the local and the cluster service plane is normal, you can perform the commissioning on the local host.

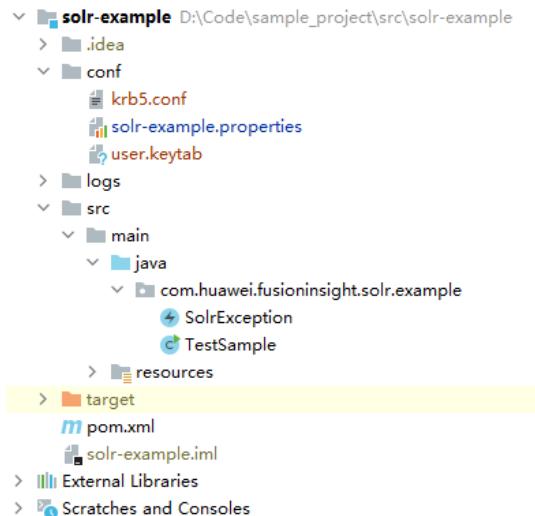
##### Procedure

- Step 1** Click **Reimport All Maven Projects** in the Maven window on the right of the IDEA to import the Maven project dependency.



## Step 2 Compile and run an application.

1. The following figure shows the file list after the configuration file has been placed and the code has been modified to match the login user. (The authentication files krb5 and keytab are not required in common mode.)



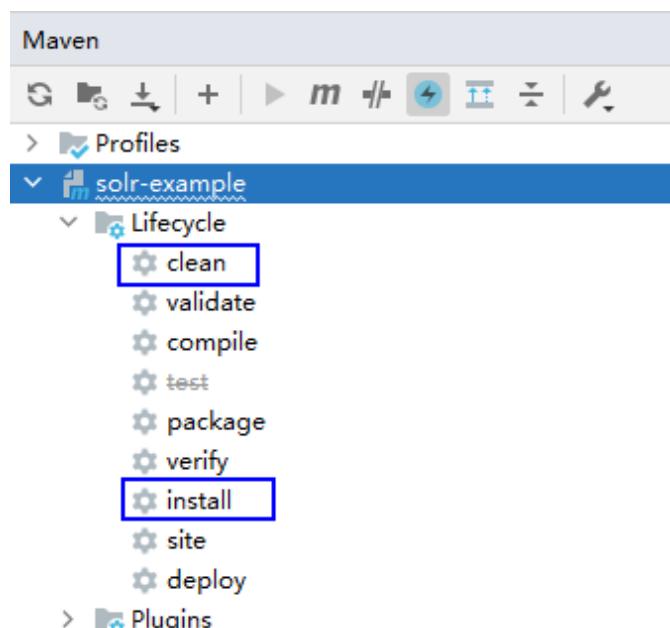
2. Compile an application.

There are two compilation modes.

- Method 1:

Choose **Maven** > *Sample project name* > **Lifecycle** > **clean** and double-click **clean** to run the **clean** command of Maven.

Choose **Maven** > *Sample project name* > **Lifecycle** > **compile**, double click **compile** to run the **compile** command of Maven.



- Method 2:

Go to the directory where the **pom.xml** file is located in the **Terminal** window in the lower part of the IDEA, and run the **mvn clean compile** command to compile the **pom.xml** file.

```
Terminal: Local x +
Microsoft Windows
(c) Microsoft Corporation.

D:\code\sample_project>cd src\solr-example

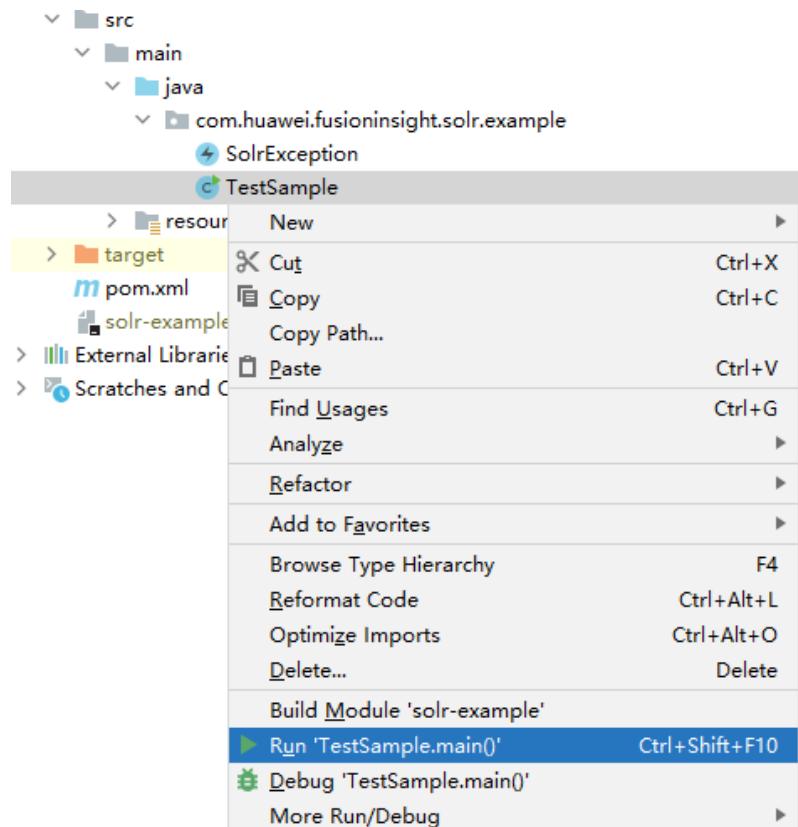
D:\code\sample_project\src\solr-example>mvn clean compile
```

3. After the compilation is complete, the message "Build Success" is displayed and the **target** directory is generated.

```
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ solr-example ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to D:\code\sample_project\src\solr-example\target\c
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.621 s
[INFO] Finished at: 2023-09-19T17:18:06+08:00
[INFO] -----
```

4. Run the program.

Right-click the **TestSample** file and choose **Run 'TestSample.main()'** from the shortcut menu to run samples.



----End

### 2.16.4.1.2 Viewing Commissioning Results

#### Scenario

After a Solr application is run, you can use one of the following methods to view the running result:

- Viewing the IntelliJ IDEA running result
- Viewing Solr logs
- Logging in to the Solr WebUI.

#### Procedure

The following information is displayed in the running results:

```
2018-09-11 10:05:04,809 | INFO | main | Using default ZkCredentialsProvider |
org.apache.solr.common.cloud.SolrZkClient.createZkCredentialsToAddAutomatically(SolrZkClient.java:213)
2018-09-11 10:05:04,866 | INFO | main | Client environment:zookeeper.version=x.x.x, built on 05/16/2018
16:22 GMT | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,871 | INFO | main | Client
environment:host.name=DGGY5M004449271.china.huawei.com |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,871 | INFO | main | Client environment:java.version=*** |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,871 | INFO | main | Client environment:java.vendor=Oracle Corporation |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,871 | INFO | main | Client environment:java.home=C:\Program Files\Java\jre*** |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,872 | INFO | main | Client environment:java.class.path=.....
2018-09-11 10:05:04,872 | INFO | main | Client environment:java.library.path=.....
2018-09-11 10:05:04,873 | INFO | main | Client environment:java.io.tmpdir=C:\Users\M00444~1\AppData
\Local\Temp\ | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,873 | INFO | main | Client environment:java.compiler=<NA> |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,873 | INFO | main | Client environment:os.name=Windows 7 |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,873 | INFO | main | Client environment:os.arch=amd64 |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,874 | INFO | main | Client environment:os.version=6.1 |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,874 | INFO | main | Client environment:user.name=m00444927 |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,875 | INFO | main | Client environment:user.home=C:\Users\m00444927 |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,875 | INFO | main | Client environment:user.dir=D:\Downloads
\FusionInsight_Cluster_<Cluster ID>_Solr_Client_Putong\FusionInsight_Cluster_<Cluster
ID>.Solr_ClientConfig\FusionInsight_Cluster_<Cluster ID>.Solr_ClientConfig\Solr\solr-example-putong |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,879 | INFO | main | Client environment:os.memory.free=231MB |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,879 | INFO | main | Client environment:os.memory.max=3639MB |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,879 | INFO | main | Client environment:os.memory.total=245MB |
org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2018-09-11 10:05:04,884 | INFO | main | Initiating client connection, connectString=10.121.16.150:24002/
solr sessionTimeout=20000 watcher=org.apache.solr.common.cloud.SolrZkClient$3@7c30a502 |
org.apache.zookeeper.ZooKeeper.<init>(ZooKeeper.java:861)
2018-09-11 10:05:05,852 | INFO | main | zookeeper.request.timeout is not configured. Using default value
120000. | org.apache.zookeeper.ClientCnxn.initRequestTimeout(ClientCnxn.java:150)
2018-09-11 10:05:05,853 | INFO | main | zookeeper.client.bind.port.range is not configured. |
org.apache.zookeeper.ClientCnxn.initClientBindingPort(ClientCnxn.java:177)
2018-09-11 10:05:05,853 | INFO | main | zookeeper.client.bind.address is not configured. |
org.apache.zookeeper.ClientCnxn.initClientBindAddress(ClientCnxn.java:194)
2018-09-11 10:05:05,854 | INFO | main | Waiting for client to connect to ZooKeeper |
org.apache.solr.common.cloud.ConnectionManager.waitForConnected(ConnectionManager.java:219)
2018-09-11 10:05:05,859 | INFO | main-SendThread(10.121.16.150:24002) | connecting to 10.121.16.150
```

```
24002 | org.apache.zookeeper.client.FourLetterWordMain.send4LetterWord(FourLetterWordMain.java:126)
2018-09-11 10:05:05,876 | INFO | main-SendThread(10.121.16.150:24002) | Got server principal from the
server and it is null | org.apache.zookeeper.ClientCnxn
$SendThread.getServerPrincipalName(ClientCnxn.java:1184)
2018-09-11 10:05:05,876 | INFO | main-SendThread(10.121.16.150:24002) | Using server principal
zookeeper/10.121.16.150 | org.apache.zookeeper.ClientCnxn
$SendThread.getServerPrincipalName(ClientCnxn.java:1224)
2018-09-11 10:05:05,878 | INFO | main-SendThread(10.121.16.150:24002) | Opening socket connection to
server 10.121.16.150/10.121.16.150:24002. Will not attempt to authenticate using SASL (unknown error) |
org.apache.zookeeper.ClientCnxn$SendThread.logStartConnect(ClientCnxn.java:1329)
2018-09-11 10:05:05,883 | INFO | main-SendThread(10.121.16.150:24002) | Socket connection established,
initiating session, client: /10.121.17.109:64205, server: 10.121.16.150/10.121.16.150:24002 |
org.apache.zookeeper.ClientCnxn$SendThread.primeConnection(ClientCnxn.java:1053)
2018-09-11 10:05:05,901 | INFO | main-SendThread(10.121.16.150:24002) | Session establishment
complete on server 10.121.16.150/10.121.16.150:24002, sessionid = 0x10019248eb8d0bea, negotiated
timeout = 20000 | org.apache.zookeeper.ClientCnxn$SendThread.onConnected(ClientCnxn.java:1607)
2018-09-11 10:05:05,913 | INFO | zkCallback-3-thread-1 | Watcher
org.apache.solr.common.cloud.ConnectionManager@40674302 name:ZooKeeperConnection
Watcher:10.121.16.150:24002/solr got event WatchedEvent state:SyncConnected type:None path:null
path:null type:None |
org.apache.solr.common.cloud.ConnectionManager.process(ConnectionManager.java:111)
2018-09-11 10:05:05,914 | INFO | main | Client is connected to ZooKeeper |
org.apache.solr.common.cloud.ConnectionManager.waitForConnected(ConnectionManager.java:237)
2018-09-11 10:05:05,914 | INFO | main | Using default ZkACLProvider |
org.apache.solr.common.cloud.SolrZkClient.createZkACLProvider(SolrZkClient.java:229)
2018-09-11 10:05:05,917 | INFO | main | Updating cluster state from ZooKeeper... |
org.apache.solr.common.cloud.ZkStateReader.createClusterStateWatchersAndUpdate(ZkStateReader.java:375
)
2018-09-11 10:05:05,969 | INFO | main | Loaded empty cluster properties |
org.apache.solr.common.cloud.ZkStateReader.loadClusterProperties(ZkStateReader.java:883)
2018-09-11 10:05:05,985 | INFO | main | Updated live nodes from ZooKeeper... (0) -> (2) |
org.apache.solr.common.cloud.ZkStateReader.refreshLiveNodes(ZkStateReader.java:690)
2018-09-11 10:05:06,036 | INFO | main | The cloud Server has been connected !!!! |
com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:159)
2018-09-11 10:05:06,036 | INFO | main | The zookeeper state is : live nodes:[10.121.16.150:21100_solr,
10.121.17.35:21100_solr]collections:{col_test=LazyCollectionRef(col_test)} |
com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:163)
.....
2018-09-11 10:05:07,905 | INFO | main | All existed collections : [col_test] |
com.huawei.fusioninsight.solr.example.TestSample.queryAllCollections(TestSample.java:326)
2018-09-11 10:05:08,295 | INFO | zkCallback-3-thread-1 | A collections change: [WatchedEvent
state:SyncConnected type:NodeChildrenChanged path:/collections], has occurred - updating... |
org.apache.solr.common.cloud.ZkStateReader$CollectionsChildWatcher.process(ZkStateReader.java:1039)
2018-09-11 10:05:08,851 | INFO | main | Success to delete collection[col_test] |
com.huawei.fusioninsight.solr.example.TestSample.deleteCollection(TestSample.java:302)
2018-09-11 10:05:08,899 | INFO | zkCallback-3-thread-1 | A collections change: [WatchedEvent
state:SyncConnected type:NodeChildrenChanged path:/collections], has occurred - updating... |
org.apache.solr.common.cloud.ZkStateReader$CollectionsChildWatcher.process(ZkStateReader.java:1039)
2018-09-11 10:05:09,025 | INFO | zkCallback-3-thread-1 | A cluster state change: [WatchedEvent
state:SyncConnected type:NodeDataChanged path:/clusterstate.json], has occurred - updating... (live nodes
size: [2]) | org.apache.solr.common.cloud.ZkStateReader
$LegacyClusterStateWatcher.process(ZkStateReader.java:1006)
2018-09-11 10:05:12,551 | INFO | main | Success to create collection[col_test] |
com.huawei.fusioninsight.solr.example.TestSample.createCollection(TestSample.java:279)
2018-09-11 10:05:12,849 | INFO | main | success to add index |
com.huawei.fusioninsight.solr.example.TestSample.addDocs(TestSample.java:218)
2018-09-11 10:05:15,051 | INFO | main | Query wasted time : 132ms |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:175)
2018-09-11 10:05:15,052 | INFO | main | Total doc num : 10 |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:177)
2018-09-11 10:05:15,053 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=2.02,
price_c=2.02,XXX,_version_=1611274922003267584} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,054 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=3.03,
price_c=3.0300000000000002,XXX,_version_=1611274922116513792} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,054 | INFO | main | doc detail : {id=5, name=Luna_5, features=[test5], price=5.05,
price_c=5.05,XXX,_version_=161127492244440064} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
```

```
2018-09-11 10:05:15,054 | INFO | main | doc detail : {id=6, name=Luna_6, features=[test6], price=6.06, price_c=6.060000000000005,XXX, _version_=1611274922248634368} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,054 | INFO | main | doc detail : {id=7, name=Luna_7, features=[test7], price=7.07, price_c=7.07,XXX, _version_=1611274922249682944} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,055 | INFO | main | doc detail : {id=9, name=Luna_9, features=[test9], price=9.09, price_c=9.09,XXX, _version_=1611274922250731520} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,055 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0, price_c=0.0,XXX, _version_=1611274918758973440} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,055 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=1.01, price_c=1.01,XXX, _version_=1611274918860685312} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,055 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=4.04, price_c=4.04,XXX, _version_=1611274918863831040} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,055 | INFO | main | doc detail : {id=8, name=Luna_8, features=[test8], price=8.08, price_c=8.08,XXX, _version_=1611274918993854464} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,165 | INFO | main | Success to delete index | com.huawei.fusioninsight.solr.example.TestSample.removeIndex(TestSample.java:196)
2018-09-11 10:05:15,224 | INFO | main | Query wasted time : 35ms | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:175)
2018-09-11 10:05:15,224 | INFO | main | Total doc num : 10 | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:177)
2018-09-11 10:05:15,224 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=2.02, price_c=2.02,XXX, _version_=1611274922003267584} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,224 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=3.03, price_c=3.030000000000002,XXX, _version_=1611274922116513792} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,225 | INFO | main | doc detail : {id=5, name=Luna_5, features=[test5], price=5.05, price_c=5.05,XXX, _version_=1611274922244440064} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,226 | INFO | main | doc detail : {id=6, name=Luna_6, features=[test6], price=6.06, price_c=6.060000000000005,XXX, _version_=1611274922248634368} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,226 | INFO | main | doc detail : {id=7, name=Luna_7, features=[test7], price=7.07, price_c=7.07,XXX, _version_=1611274922249682944} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,226 | INFO | main | doc detail : {id=9, name=Luna_9, features=[test9], price=9.09, price_c=9.09,XXX, _version_=1611274922250731520} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,226 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0, price_c=0.0,XXX, _version_=1611274918758973440} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,226 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=1.01, price_c=1.01,XXX, _version_=1611274918860685312} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,226 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=4.04, price_c=4.04,XXX, _version_=1611274918863831040} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,227 | INFO | main | doc detail : {id=8, name=Luna_8, features=[test8], price=8.08, price_c=8.08,XXX, _version_=1611274918993854464} | com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:179)
2018-09-11 10:05:15,236 | INFO | main | Session: 0x10019248eb8d0bea closed | org.apache.zookeeper.ZooKeeper.close(ZooKeeper.java:1325)
2018-09-11 10:05:15,236 | INFO | main-EventThread | EventThread shut down for session: 0x10019248eb8d0bea | org.apache.zookeeper.ClientCnxn$EventThread.run(ClientCnxn.java:614)
```

## NOTE

In the Windows environment, the following exception occurs but does not affect services.

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

## Log description

The log level is **INFO** by default and more detailed information can be viewed by changing the log level (**DEBUG**, **INFO**, **WARN**, **ERROR**, and **FATAL**). You can modify the **log4j.properties** file to change log levels, for example:

```
# Set root category priority to info and its only appender to console.  
log4j.rootCategory=info,console,R  
#log4j.debug=true  
  
# console is set to be a ConsoleAppender using a PatternLayout.  
log4j.appender.console=org.apache.log4j.ConsoleAppender  
log4j.appender.console.Threshold=DEBUG  
log4j.appender.console.layout=org.apache.log4j.PatternLayout  
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n  
  
# R is set to be a File appender using a PatternLayout.  
log4j.appender.R=org.apache.log4j.RollingFileAppender  
log4j.appender.R.Append=true  
log4j.appender.R.Threshold=debug  
log4j.appender.R.MaxFileSize=1024KB  
log4j.appender.R.MaxBackupIndex=10  
log4j.appender.R.File=logs/solrclient.log  
log4j.appender.R.layout=org.apache.log4j.PatternLayout  
log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} | %-5p | %t | %m | %l%n
```

### 2.16.4.2 Commissioning an Application in Linux

#### 2.16.4.2.1 Compiling and Running an Application When a Client Is Installed

##### Scenario

In a Linux environment where a Solr client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

##### Prerequisites

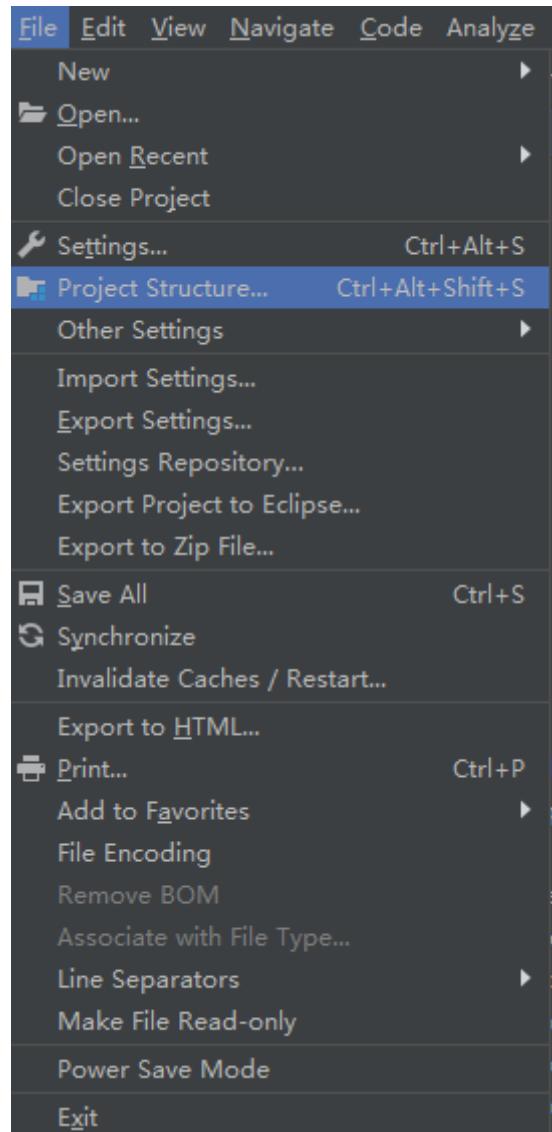
You have installed the Solr client.

##### Procedure

###### Step 1 Export the jar package.

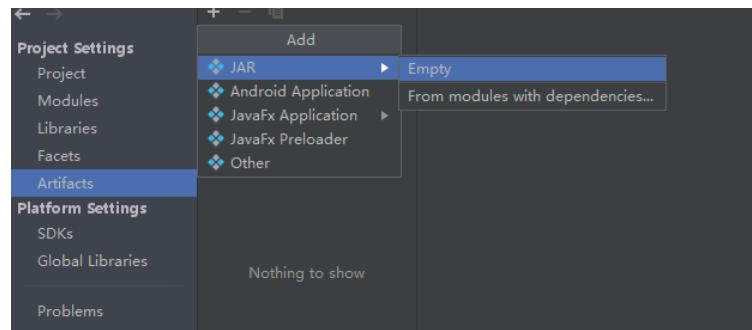
1. Log in to IntelliJ IDEA and choose **File > Project Structure**.

Figure 2-257 Project Structure



2. Select **Artifacts**, click +, and choose **JAR > Empty**.

Figure 2-258 Adding artifacts

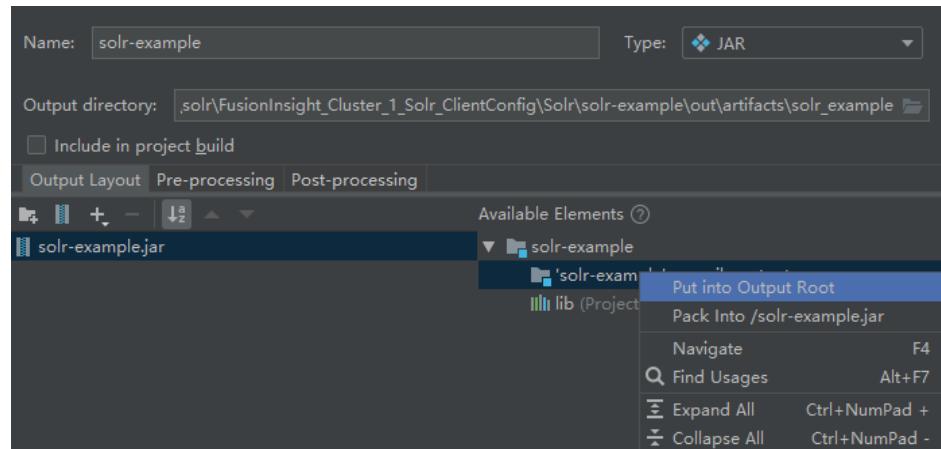


3. Change the value of **Name** to a user-defined name, for example, **solr-example**.



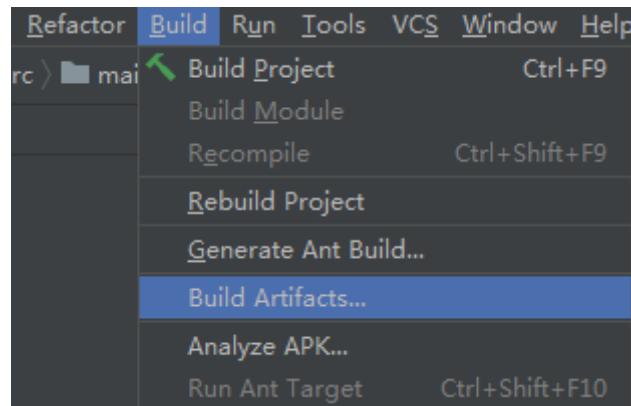
4. Select **Output Layout**, right-click 'solr-example' compile output in **Available Elements** on the right, and choose **Put Into Output Root**.

**Figure 2-259** Adding output



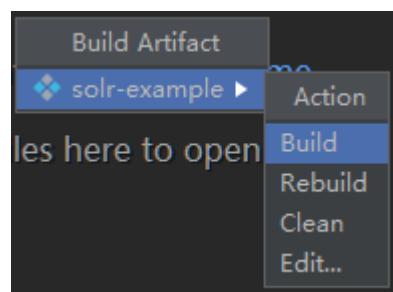
5. Click **Apply** to save the modification, and then click **OK** to exit.
6. Choose **Build > Build Artifacts....**

**Figure 2-260** Preparing for compilation

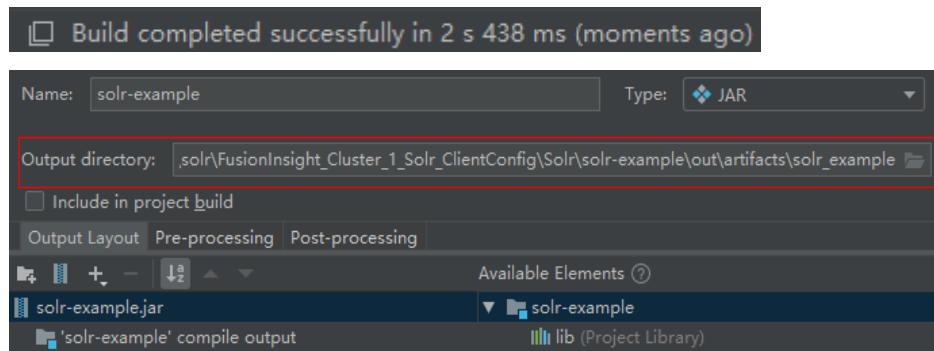


7. In the dialog box that is displayed, choose **solr-example > Build** to compile the artifact.

**Figure 2-261** Compiling an artifact



8. If information similar to the following is displayed in the event log, the JAR file has been successfully generated. Obtain the JAR file from the **Output directory** configured in **Artifacts**.



### Step 2 Run the jar package.

1. Assume that the client installation directory is **/opt/hadoopclient**. Copy the exported jar package **solr-example.jar** to **/opt/hadoopclient/Solr/tools/solr-example** in the client installation directory and copy the **conf** directory files of the project to **/opt/hadoopclient/Solr/tools/solr-example/conf**.
2. Run **source /opt/hadoopclient/bigdata\_env**, and then switch to the code function directory **cd /opt/hadoopclient/Solr/tools/solr-example**.
3. Run the java command:  

```
java -cp /opt/hadoopclient/Solr/tools/lib/*:solr-example.jar:/opt/hadoopclient/Solr/tools/solr-example/conf/com.huawei.fusioninsight.solr.example.TestSample
```

----End

#### 2.16.4.2.2 Compiling and Running an Application When No Client Is Installed

##### Scenario

In a Linux environment where no Solr client is installed, you can upload the JAR package to the Linux and then run an application after the code development is complete.

##### Prerequisites

JDK has been installed on Linux. The version must be the same as JDK version of the jar package exported from IntelliJ IDEA. Java environment variables have been set.

##### Procedure

**Step 1** Export the jar package. For details, see [Step 1](#).

**Step 2** Access the directory where the **pom.xml** file is stored in the **Terminal** window of the IDEA or other command line tools. Run the following command to generate the **lib** folder in the directory where **pom.xml** is located. The folder contains the JAR package on which the sample project depends.

```
mvn dependency:copy-dependencies -DoutputDirectory=lib
```

**Step 3** Copy the **lib** directory which the application development environment requires and conf directory to any directory in the client operating environment, for example, **/opt/solr-example**, and then copy the jar package generated in the application environment to **/opt/solr-example/lib/**.

**Step 4** On the client, run the **cd /opt/solr-example** command to switch to the copy directory.

**Step 5** Run the jar commands:

```
java -cp /opt/solr-example/lib/*:/opt/solr-example/conf/  
com.huawei.fusioninsight.solr.example.TestSample
```

----End

### 2.16.4.2.3 Viewing Commissioning Results

#### Scenario

After a Solr application is run, you can use one of the following methods to view the running result:

- Viewing the command output.
- Viewing Solr logs.
- Logging in to the Solr WebUI.

#### Procedure

The following information is displayed in the running results:

```
2020-09-16 11:23:06,740 | INFO | main | Client environment:zookeeper.version=x.x.x., built on 09/11/2020  
14:26 GMT | org.apache.zookeeper.Environment.logEnv(Environment.java:109)  
2020-09-16 11:23:06,740 | INFO | main | Client environment:host.name=kwephispra41854 |  
org.apache.zookeeper.Environment.logEnv(Environment.java:109)  
2020-09-16 11:23:06,741 | INFO | main | Client environment:java.version=*** |  
org.apache.zookeeper.Environment.logEnv(Environment.java:109)  
2020-09-16 11:23:06,741 | INFO | main | Client environment:java.vendor=Huawei Technologies Co., Ltd |  
org.apache.zookeeper.Environment.logEnv(Environment.java:109)  
2020-09-16 11:23:06,741 | INFO | main | Client environment:java.home=/opt/client_solr_800/JRE/jre*** |  
org.apache.zookeeper.Environment.logEnv(Environment.java:109)  
2020-09-16 11:23:06,742 | INFO | main | Client environment:java.class.path=/opt/client_solr_800/Solr/solr-  
example/lib/httpcore-4.4.10.jar:/opt/client_solr_800/Solr/solr-example/lib/hadoop-auth-XXX.jar:/opt/  
client_solr_800/Solr/solr-example/lib/zookeeper-XXX.jar:/opt/client_solr_800/Solr/solr-example/lib/  
log4j-1.2.17-atlassian-13.jar:/opt/client_solr_800/Solr/solr-example/lib/httpmime-4.5.6.jar:/opt/  
client_solr_800/Solr/solr-example/lib/stax2-api-3.1.4.jar:/opt/client_solr_800/Solr/solr-example/lib/  
guava-25.1-jre.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-lang-2.6.jar:/opt/client_solr_800/Solr/  
solr-example/lib/commons-configuration2-2.1.1.jar:/opt/client_solr_800/Solr/solr-example/lib/log4j-  
core-2.12.0.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-collections4-4.2.jar:/opt/client_solr_800/  
Solr/solr-example/lib/jackson-annotations-2.10.0.jar:/opt/client_solr_800/Solr/solr-example/lib/log4j-  
api-2.12.0.jar:/opt/client_solr_800/Solr/solr-example/lib/solr-core-XXX.jar:/opt/client_solr_800/Solr/solr-  
example/lib/http2-common-9.4.19.v20190610.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-  
codec-1.11.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-lang3-3.8.1.jar:/opt/client_solr_800/Solr/  
solr-example/lib/log4j-web-2.12.0.jar:/opt/client_solr_800/Solr/solr-example/lib/httpclient-4.5.6.jar:/opt/  
client_solr_800/Solr/solr-example/lib/jetty-client-9.4.19.v20190610.jar:/opt/client_solr_800/Solr/solr-  
example/lib/woodstox-core-asl-4.4.1.jar:/opt/client_solr_800/Solr/solr-example/lib/jackson-  
core-2.10.0.jar:/opt/client_solr_800/Solr/solr-example/lib/log4j-1.2-api-2.12.0.jar:/opt/client_solr_800/Solr/  
solr-example/lib/slf4j-api-1.7.24.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-  
logging-1.1.3.jar:/opt/client_solr_800/Solr/solr-example/lib/solr-example.jar:/opt/client_solr_800/Solr/solr-  
example/lib/http2-http-client-transport-9.4.19.v20190610.jar:/opt/client_solr_800/Solr/solr-example/lib/solr-  
solrj-XXX.jar:/opt/client_solr_800/Solr/solr-example/lib/zookeeper-jute-XXX.jar:/opt/client_solr_800/Solr/solr-  
example/lib/jackson-databind-2.10.0.jar:/opt/client_solr_800/Solr/solr-example/lib/hadoop-common-  
XXX.jar:/opt/client_solr_800/Solr/solr-example/lib/http2-server-9.4.19.v20190610.jar:/opt/client_solr_800/Solr/
```

```
solr-example/lib/log4j-slf4j-impl-2.12.0.jar:/opt/client_solr_800/Solr/solr-example/lib/http2-client-9.4.19.v20190610.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-cli-1.2.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-collections-3.2.2.jar:/opt/client_solr_800/Solr/solr-example/lib/commons-io-2.5.jar:/opt/client_solr_800/Solr/solr-example/lib/http2-hpack-9.4.19.v20190610.jar:/opt/client_solr_800/Solr/solr-example/lib/hadoop-common-XXX.jar:/opt/client_solr_800/Solr/solr-example/conf/ | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,743 | INFO | main | Client environment:java.library.path=/opt/client_solr_800/JRE/jre1.8.0_242/lib:/opt/client_solr_800/KrbClient/kerberos/lib::/usr/java/packages/lib/aarch64:/lib:/usr/lib | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,744 | INFO | main | Client environment:java.io.tmpdir=/tmp | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,744 | INFO | main | Client environment:java.compiler=<NA> | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,744 | INFO | main | Client environment:os.name=Linux | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,745 | INFO | main | Client environment:os.arch=aarch64 | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,745 | INFO | main | Client environment:os.version=4.19.36-vhulk1907.1.0.h619.eulerosv2r8.aarch64 | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,745 | INFO | main | Client environment:user.name=root | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,746 | INFO | main | Client environment:user.home=/root | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,746 | INFO | main | Client environment:user.dir=/opt/client_solr_800/Solr/solr-example/lib | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,746 | INFO | main | Client environment:os.memory.free=798MB | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,747 | INFO | main | Client environment:os.memory.max=12855MB | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,747 | INFO | main | Client environment:os.memory.total=866MB | org.apache.zookeeper.Environment.logEnv(Environment.java:109)
2020-09-16 11:23:06,752 | INFO | main | Initiating client connection, connectString=10.244.224.211:24002,10.244.224.21:24002,10.244.224.217:24002/solr sessionTimeout=20000 watcher=org.apache.solr.common.cloud.SolrZkClient$ProcessWatchWithExecutor@6a2f6f80 | org.apache.zookeeper.ZooKeeper.<init>(ZooKeeper.java:878)
2020-09-16 11:23:06,762 | INFO | main | jute.maxbuffer value is 4194304 Bytes | org.apache.zookeeper.ClientCnxnSocket.initProperties(ClientCnxnSocket.java:238)
2020-09-16 11:23:06,774 | INFO | main | zookeeper.request.timeout value is 120000. feature.enabled=true | org.apache.zookeeper.ClientCnxn.initRequestTimeout(ClientCnxn.java:1706)
2020-09-16 11:23:06,776 | INFO | main | zookeeper.client.bind.port.range is not configured. | org.apache.zookeeper.common.whitelist.ClientBindingHelper.initClientBindingPort(ClientBindingHelper.java:94)
2020-09-16 11:23:06,777 | INFO | main | zookeeper.client.bind.address is not configured. | org.apache.zookeeper.common.whitelist.ClientBindingHelper.initClientBindAddress(ClientBindingHelper.java:61)
2020-09-16 11:23:06,779 | INFO | main | Waiting for client to connect to ZooKeeper | org.apache.solr.common.cloud.ConnectionManager.waitForConnected(ConnectionManager.java:233)
2020-09-16 11:23:06,786 | INFO | main-SendThread(10.244.224.21:24002) | connecting to kwephispra41854 24002 | org.apache.zookeeper.client.FourLetterWordMain.send4LetterWord(FourLetterWordMain.java:140)
2020-09-16 11:23:06,797 | INFO | main-SendThread(10.244.224.21:24002) | Got server principal from the server and it is null | org.apache.zookeeper.ServerPrincipalProvider.getServerPrincipalFromServer(ServerPrincipalProvider.java:77)
2020-09-16 11:23:06,798 | INFO | main-SendThread(10.244.224.21:24002) | Using server principal zookeeper/10.244.224.21 | org.apache.zookeeper.ServerPrincipalProvider.getServerPrincipal(ServerPrincipalProvider.java:57)
2020-09-16 11:23:06,967 | INFO | main-SendThread(10.244.224.21:24002) | Client successfully logged in. | org.apache.zookeeper.Login.login(Login.java:312)
2020-09-16 11:23:06,969 | INFO | Thread-1 | TGT refresh thread started. | org.apache.zookeeper.Login$1.run(Login.java:136)
2020-09-16 11:23:06,975 | INFO | main-SendThread(10.244.224.21:24002) | Client will use GSSAPI as SASL mechanism. | org.apache.zookeeper.util.SecurityUtils$1.run(SecurityUtils.java:128)
2020-09-16 11:23:06,977 | INFO | Thread-1 | TGT valid starting at: Wed Sep 16 11:23:06 CST 2020 | org.apache.zookeeper.Login.getRefreshTime(Login.java:330)
2020-09-16 11:23:06,977 | INFO | Thread-1 | TGT expires: Thu Sep 17 11:23:06 CST 2020 | org.apache.zookeeper.Login.getRefreshTime(Login.java:331)
2020-09-16 11:23:06,978 | INFO | Thread-1 | TGT refresh sleeping until: Thu Sep 17 06:46:32 CST 2020 | org.apache.zookeeper.Login$1.run(Login.java:194)
2020-09-16 11:23:07,324 | INFO | main-SendThread(10.244.224.21:24002) | Opening socket connection to server kwephispra41854/10.244.224.21:24002. Will attempt to SASL-authenticate using Login Context section 'Client' | org.apache.zookeeper.ClientCnxn$SendThread.logStartConnect(ClientCnxn.java:1149)
2020-09-16 11:23:07,334 | INFO | main-SendThread(10.244.224.21:24002) | Socket connection established,
```

```
initiating session, client: /10.244.224.21:40720, server: kwephispra41854/10.244.224.21:24002 |  
org.apache.zookeeper.ClientCnxn$SendThread.primeConnection(ClientCnxn.java:996)  
2020-09-16 11:23:07,348 | INFO | main-SendThread(10.244.224.21:24002) | Session establishment complete  
on server kwephispra41854/10.244.224.21:24002, connectionid = 0x18002d1373b604b9, negotiated timeout  
= 20000 | org.apache.zookeeper.ClientCnxn$SendThread.onConnected(ClientCnxn.java:1432)  
2020-09-16 11:23:07,357 | INFO | zkConnectionManagerCallback-5-thread-1 | zkClient has connected |  
org.apache.solr.common.cloud.ConnectionManager.process(ConnectionManager.java:125)  
2020-09-16 11:23:07,359 | INFO | main | Client is connected to ZooKeeper |  
org.apache.solr.common.cloud.ConnectionManager.waitForConnected(ConnectionManager.java:251)  
2020-09-16 11:23:07,435 | INFO | main | Updated live nodes from ZooKeeper... (0) -> (2) |  
org.apache.solr.common.cloud.ZkStateReader.refreshLiveNodes(ZkStateReader.java:859)  
2020-09-16 11:23:07,470 | INFO | main | Cluster at  
10.244.224.211:24002,10.244.224.21:24002,10.244.224.217:24002/solr ready |  
org.apache.solr.client.solrj.impl.ZkClientClusterStateProvider.getZkStateReader(ZkClientClusterStateProvider.ja  
va:177)  
2020-09-16 11:23:07,471 | INFO | main | The cloud Server has been connected !!!! |  
com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:204)  
2020-09-16 11:23:07,471 | INFO | main | The zookeeper state is : znodeVersion: 0  
live nodes:[10.244.224.21:21101_solr, 10.244.224.211:21101_solr]  
collections:{} | com.huawei.fusioninsight.solr.example.TestSample.getCloudSolrClient(TestSample.java:208)  
2020-09-16 11:23:07,933 | WARN | main | Unable to load native-hadoop library for your platform... using  
builtin-java classes where applicable |  
org.apache.hadoop.util.NativeCodeLoader.<clinit>(NativeCodeLoader.java:60)  
log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).  
log4j:WARN Please initialize the log4j system properly.  
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.  
2020-09-16 11:23:08,459 | INFO | main | All existed collections : [] |  
com.huawei.fusioninsight.solr.example.TestSample.queryAllCollections(TestSample.java:354)  
2020-09-16 11:23:16,115 | INFO | main | Success to create collection[col_test_0914] |  
com.huawei.fusioninsight.solr.example.TestSample.createCollection(TestSample.java:311)  
2020-09-16 11:23:16,545 | INFO | main | success to add index |  
com.huawei.fusioninsight.solr.example.TestSample.addDocs(TestSample.java:251)  
2020-09-16 11:23:19,369 | INFO | main | Query wasted time : 779ms |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:221)  
2020-09-16 11:23:19,369 | INFO | main | Total doc num : 5 |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:222)  
2020-09-16 11:23:19,370 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0,  
price_c=0,XXX, _version_=1677959203708207104} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:19,371 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=2.0,  
price_c=2,XXX, _version_=1677959203714498560} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:19,371 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=8.0,  
price_c=8,XXX, _version_=1677959203715547136} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:19,371 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=4.0,  
price_c=4,XXX, _version_=1677959203709255680} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:19,372 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=6.0,  
price_c=6,XXX, _version_=1677959203716595712} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:20,224 | INFO | main | Success to delete index |  
com.huawei.fusioninsight.solr.example.TestSample.removeIndex(TestSample.java:239)  
2020-09-16 11:23:20,445 | INFO | main | Query wasted time : 211ms |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:221)  
2020-09-16 11:23:20,446 | INFO | main | Total doc num : 5 |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:222)  
2020-09-16 11:23:20,446 | INFO | main | doc detail : {id=0, name=Luna_0, features=[test0], price=0.0,  
price_c=0,XXX, _version_=1677959203708207104} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:20,447 | INFO | main | doc detail : {id=1, name=Luna_1, features=[test1], price=2.0,  
price_c=2,XXX, _version_=1677959203714498560} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:20,447 | INFO | main | doc detail : {id=4, name=Luna_4, features=[test4], price=8.0,  
price_c=8,XXX, _version_=1677959203715547136} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)  
2020-09-16 11:23:20,448 | INFO | main | doc detail : {id=2, name=Luna_2, features=[test2], price=4.0,  
price_c=4,XXX, _version_=1677959203709255680} |  
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)
```

```
2020-09-16 11:23:20,448 | INFO | main | doc detail : {id=3, name=Luna_3, features=[test3], price=6.0, price_c=6,XXX, _version_=1677959203716595712} |
com.huawei.fusioninsight.solr.example.TestSample.queryIndex(TestSample.java:224)
2020-09-16 11:23:20,564 | INFO | main | Connectionid: 0x18002d1373b604b9 closed |
org.apache.zookeeper.ZooKeeper.close(ZooKeeper.java:1457)
2020-09-16 11:23:20,564 | INFO | main-EventThread | EventThread shut down for connectionid: 0x18002d1373b604b9 | org.apache.zookeeper.ClientCnxn$EventThread.run(ClientCnxn.java:541)
```

## 2.16.5 More Information

### 2.16.5.1 External Interfaces

#### 2.16.5.1.1 Shell

You can directly perform operations on Solr by running Shell client scripts.

Methods of running the Shell command (assuming that the client installation directory is: **/opt/hadoopclient**):

```
source /opt/hadoopclient/bigdata_env
```

Run **/opt/hadoopclient/Solr/solr-client/bin/solrctl --help** to obtain the help information for specific Solr parameters.

```
usage: solrctl [options] command [command-arg] [command [command-arg]] ...
```

Options:

```
--help
```

```
--quiet
```

Commands:

```
confset [--generate path [-schemaless | -confWithHBase]]
```

```
[--create name path -force]
```

```
[--update name path -force]
```

```
[--get name path]
```

```
[--putfile path localpath]
```

```
[--getfile path filename]
```

```
[--clearfile path]
```

```
[--delete name]
```

```
[--list]
```

```
collection [--create name -s <numShards>
```

```
[ -a Create collection with autoAddReplicas=true]
```

```
[ -S <true|false> Create collection true or false with sharedFsReplication]
```

```
[ -b <true|false> Create collection true or false with autoShardBalance]
```

```
[ -c <collection.configName>]
```

```
[ -r <replicationFactor>]
```

```
[ -m <maxShardsPerNode>]
```

```
[ -n <createNodeSet>]
```

```
[ -f <router.field>]
```

```
[ -p name=value ...]
```

```
[ --modify name]
```

```
[ -a <true|false> Modify collection true or false with autoAddReplicas]
```

```
[ -b <true|false> Modify collection true or false with autoShardBalance]
```

```
[ -r <replicationFactor>]
```

```
[ -m <maxShardsPerNode>]
```

```
[ --delete name]
```

```
[ --reload name]
```

```
[ --splitshard collection shard]
```

```
[ --createshard collection shard]
```

```
[ --deleteshard collection shard]
```

```
[ --createalias aliasname collections]
```

```
[ --deletealias aliasname]
```

```
[ --deleterepllica [-p name=value]...]
```

```
[--addreplica [-p name=value]...]
[--deletedocs name]
[--list]
[--replacenode source_node target_node
[-p Replace node with parallel=true]
[-t <timeout in ms>]
[-c Replace node with skipCheck=false]
]

cluster [--get-solrxml file]
[--put-solrxml file]
[--set-property name value]
[--remove-property name]
[--list-free-node]
[--list-local-disk-live-node]
```

### 2.16.5.1.2 Java API

Common Solr Java classes are as follows:

- CloudSolrClient: core class of client applications. It encapsulates HttpClient and communicates with SolrCloud.
- CollectionAdminRequest: various request APIs. It is displayed in the internal form, such as CollectionAdminRequest.Create, CollectionAdminRequest.List, and CollectionAdminRequest.Delete.
- SolrInputDocument: index record.
- SolrQuery: index query class.

For details about the API of each class, see:

[https://lucene.apache.org/solr/6\\_2\\_0/solr-solrj/index.html?overview-summary.html](https://lucene.apache.org/solr/6_2_0/solr-solrj/index.html?overview-summary.html).

### 2.16.5.1.3 Web UI

#### Scenario

The Web UI displays the Solr cluster status, including summary of a cluster and information about SolrServer, SolrServerAdmin, snapshots, and running processes. You can better understand the Solr cluster status based on information displayed on the Web UI.



#### NOTE

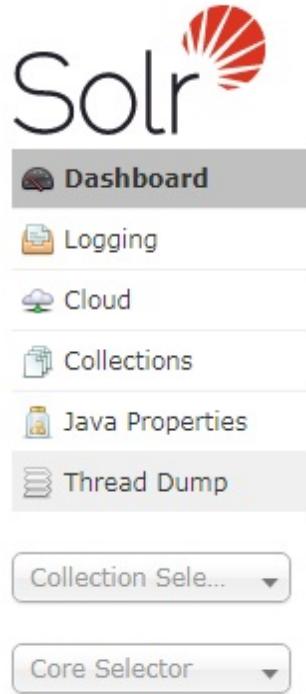
Contact the administrator to obtain a service account that has the right to access the WebUI and obtain its password.

#### Procedure

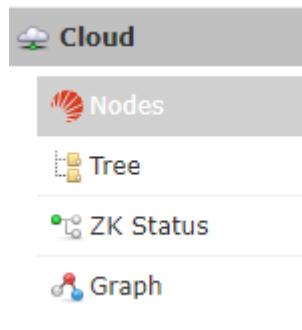
**Step 1** [Accessing FusionInsight Manager of an MRS Cluster](#). Choose **Cluster** > *Name of the desired cluster* > **Services** > **Solr** > **SolrServerAdmin** and open the Web UI in Solr.

**Step 2** The Solr web UI page includes several parts, as shown in [Figure 2-262](#).

Figure 2-262 Solr Web UI control menu



- DashBoard: displays some system parameters, such as JVM startup parameters and JVM memory usage.
- Logging: displays logs at the warn and upper levels on the page, and provides the function of dynamically adjusting the level of logs.
- Cloud: displays each structure of the Collection.



- Nodes: displays the IP addresses, ports, and resource usage of each node.
- Tree: displays the tree-like structure for the directory files of index in Zookeeper.
- ZK Status: displays the status tables of each ZooKeeper node.
- Graph: displays the relationship among each Collection, shard, and replica.
- Collections: Management UI of collections. It provides functions such as Add, Delete, Reload, Create Alias and Delete Alias.
- Java Properties: lists all environment variables used by Solr.
- Thread Dump: thread stack, lists information about all thread stacks running in the background.

- Suggestions: cluster suggestions, which list suggestions provided by Solr. If no suggestion is available, the table data is empty.
- Core Selector: is used to choose a core as well as create data indexes for and search data in the core.

----End

## 2.17 Spark Development Guide

### 2.17.1 Overview

#### 2.17.1.1 Application Development Overview

##### Spark Introduction

Spark is a distributed batch processing system as well as an analysis and mining engine. It provides an iterative memory computation framework and supports the development in multiple programming languages, including Scala, Java, and Python. The application scenarios of Spark include:

- Data processing: Spark can process data quickly and has fault tolerance and scalability.
- Iterative computation: Spark supports iterative computation to meet the requirements of multi-step data processing logic.
- Data mining: Based on massive data, Spark can handle complex data mining and analysis and support multiple data mining and machine learning algorithms.
- Streaming processing: Spark supports stream processing at a seconds-level delay and supports multiple external data sources.
- Query analysis: Spark supports standard SQL query analysis, provides the DSL (DataFrame), and supports multiple external inputs.

This section focuses on the application development guides of Spark, Spark SQL and Spark Streaming.

##### Spark Development API Introduction

Spark supports the development in multiple programming languages, including Scala, Java, and Python. Since Spark is developed in Scala and Scala is easy to read, users are advised to develop Spark application in Scala.

Divided by different languages, the APIs of Spark are listed in [Table 2-155](#).

**Table 2-155** Spark APIs

Function	Description
Scala API	Indicates the API in Scala. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Scala</a> . Since Scala is easy to read, users are advised to use Scala APIs in the program development.
Java API	Indicates the API in Java. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Java</a> .
Python API	Indicates the API in Python. For common APIs of Spark Core, Spark SQL and Spark Streaming, see <a href="#">Python</a> .

Divided by different modes, APIs listed in the preceding table are used in the development of Spark Core and Spark Streaming. Spark SQL supports CLI and JDBCServer for accessing. There are two ways to access the JDBCServer: Beeline and the JDBC client code. For details, see [JDBCServer Interface](#).

 **NOTE**

For spark-sql, spark-shell and spark-submit (which application contains SQL operations), do not use the **proxy user** parameter to submit a task.

### 2.17.1.2 Basic Concepts

#### Basic Concepts

- **RDD**

Resilient Distributed Dataset (RDD) is a core concept of Spark. It indicates a read-only and partitioned distributed dataset. Partial or all data of this dataset can be cached in the memory and reused between computations.

##### RDD Generation

- An RDD can be generated from the Hadoop file system or other storage systems that are compatible with Hadoop, such as Hadoop Distributed File System (HDFS).
- A new RDD can be transferred from a parent RDD.
- An RDD can be converted from a collection.

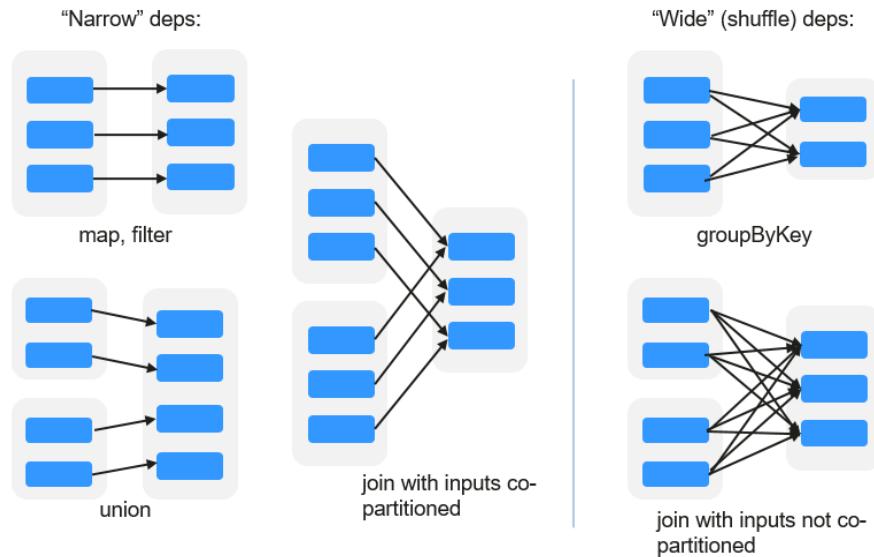
##### RDD Storage

- Users can select different storage levels to store an RDD for reuse. (There are 11 storage levels to store an RDD.)
- The current RDD is stored in the memory by default. When the memory is insufficient, the RDD overflows to the disk.

- **RDD Dependency**

The RDD dependency includes the narrow dependency and wide dependency.

**Figure 2-263** RDD dependency



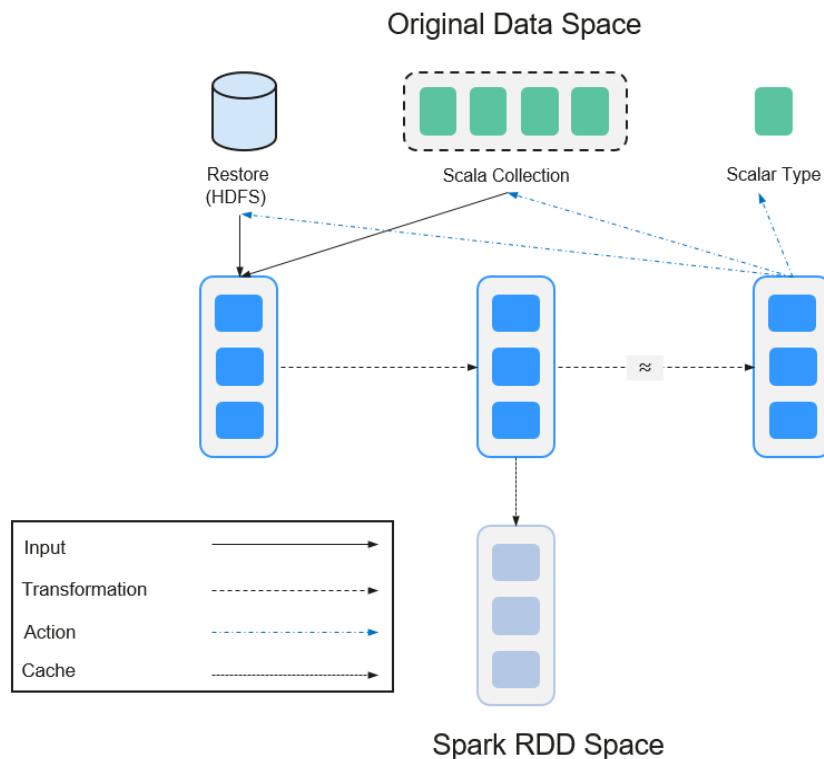
- **Narrow dependency:** Each partition of the parent RDD is used by at most one partition of the child RDD partition.
- **Wide dependency:** Partitions of the child RDD depend on all partitions of the parent RDD due to shuffle operations.

The narrow dependency facilitates the optimization. Logically, each RDD operator is a fork/join process. Fork the RDD to each partition, and then perform the computation. After the computation, join the results, and then perform the fork/join operation on next operator. It takes a long period of time to directly translate the RDD to physical implementation. There are two reasons: Each RDD (even the intermediate results) must be physicalized to the memory or storage, which takes time and space; the partitions can be joined only when the computation of all partitions is complete (if the computation of a partition is slow, the entire join process is slowed down). If the partitions of the child RDD narrowly depend on the partitions of the parent RDD, the two fork/join processes can be combined to optimize the entire process. If the relationship in the continuous operator sequence is narrow dependency, multiple fork/join processes can be combined to reduce the time for waiting and improve the performance. This is called pipeline optimization in Spark.

- **Transformation and Action (RDD Operations)**

Operations on RDD include transformation (the returned value is an RDD) and action (the returned value is not an RDD). [Figure 2-264](#) shows the process. The transformation is lazy, which indicates that the transformation from one RDD to another RDD is not immediately executed. Spark only records the transformation but does not execute it immediately. The real computation is started only when the action is started. The action returns results or writes the RDD data into the storage system. The action is the driving force for Spark to start the computation.

Figure 2-264 RDD operation



The data and operation model of RDD are quite different from those of Scala.

```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()
errors.count()
```

- The **textFile** operator reads log files from the HDFS and returns **file** (as an RDD).
- The filter operator filters rows with ERROR and assigns them to errors (a new RDD). The filter operator is a transformation.
- The cache operator caches errors for future use.
- The count operator returns the number of rows of errors. The count operator is an action.

#### Transformation includes the following types:

- The RDD elements are regarded as simple elements:

The input and output have the one-to-one relationship, and the partition structure of the result RDD remains unchanged, for example, map.

The input and output have the one-to-many relationship, and the partition structure of the result RDD remains unchanged, for example, flatMap (one element becomes a sequence containing multiple elements and then flattens to multiple elements).

The input and output have the one-to-one relationship, but the partition structure of the result RDD changes, for example, union (two RDDs integrates to one RDD, and the number of partitions becomes the sum of the number of partitions of two RDDs) and coalesce (partitions are reduced).

Operators of some elements are selected from the input, such as filter, distinct (duplicate elements are deleted), subtract (elements only exist in this RDD are retained), and sample (samples are taken).

- The RDD elements are regarded as Key-Value pairs.
  - Perform the one-to-one calculation on the single RDD, such as mapValues (the partition mode of the source RDD is retained, which is different from map).
  - Sort the single RDD, such as sort and partitionBy (partitioning with consistency, which is important to the local optimization).
  - Restructure and reduce the single RDD based on key, such as groupByKey and reduceByKey.
  - Join and restructure two RDDs based on key, such as join and cogroup.

 NOTE

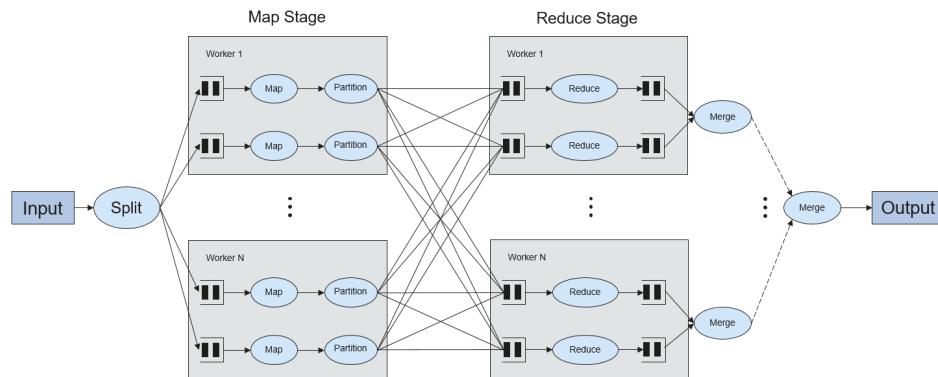
The later three operations involve sorting and are called shuffle operations.

**Action is classified into the following types:**

- Generate scalar configuration items, such as count (the number of elements in the returned RDD), reduce, fold/aggregate (the number of scalar configuration items that are returned), and take (the number of elements before the return).
  - Generate the Scala collection, such as collect (import all elements in the RDD to the Scala collection) and lookup (look up all values corresponds to the key).
  - Write data to the storage, such as saveAsTextFile (which corresponds to the preceding textFile).
  - Check points, such as checkpoint. When Lineage is quite long (which occurs frequently in graphics computation), it takes a long period of time to execute the whole sequence again when a fault occurs. In this case, checkpoint is used as the check point to write the current data to stable storage.
- **Shuffle**

Shuffle is a specific phase in the MapReduce framework, which is located between the Map phase and the Reduce phase. If the output results of Map are to be used by Reduce, each output result must be hashed based on the key and distributed to the corresponding Reducer. This process is called Shuffle. Shuffle involves the read and write of the disk and the transmission of the network, so that the performance of Shuffle directly affects the operation efficiency of the entire program.

The following figure describes the entire process of the MapReduce algorithm.

**Figure 2-265 Algorithm process**

Shuffle is a bridge connecting data. The following describes the implementation of shuffle in Spark.

Shuffle divides the Job of a Spark into multiple stages. The former stages contain one or multiple `ShuffleMapTasks`, and the last stage contains one or multiple `ResultTasks`.

- **Spark Application Structure**

The Spark application structure includes the initial `SparkContext` and the main program.

- Initial `SparkContext`: constructs the operation environment of the Spark application.

Constructs the `SparkContext` object. For example:

```
new SparkContext(master, appName, [SparkHome], [jars])
```

Parameter description

**master**: indicates the link string. The link modes include local, YARN-cluster, and YARN-client.

**appName**: indicates the application name.

**SparkHome**: indicates the directory where Spark is installed in the cluster.

**jars**: indicates the code and dependency package of the application.

- Main program: processes data.

For submitting applications details, see <https://spark.apache.org/docs/3.3.1/submitting-applications.html>.

- **Spark Shell Command**

The basic Spark shell command supports the submitting of the Spark application. The Spark shell command is as follows.

```
./bin/spark-submit \
--class <main-class> \
--master <master-url> \
... # other options
<application-jar> \
[application-arguments]
```

Parameter description:

--class: indicates the name of the class of the Spark application.

--master: indicates the master that the Spark application links, such as YARN-client and YARN-cluster.

application-jar: indicates the path of the jar package of the Spark application.

application-arguments: indicates the parameter required to submit the Spark application. (This parameter can be empty.)

- **Spark JobHistory Server**

The Spark web UI is used to monitor the details in each phase of the Spark framework of a running or historical Spark job and provide the log display, which helps users to develop, configure, and optimize the job in more fine-grained units.

## Spark SQL Basic Concepts

### DataSet

A DataSet is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations. Each Dataset also has an untyped view called a DataFrame, which is a Dataset of Row.

DataFrame is a structured distributed data set composed of several columns, which is similar to a table in the relationship database or the data frame in R/Python. DataFrame is a basic concept in Spark SQL, and can be created by using multiple methods, such as structured data set, Hive table, external database, or RDD.

## Spark Streaming Basic Concepts

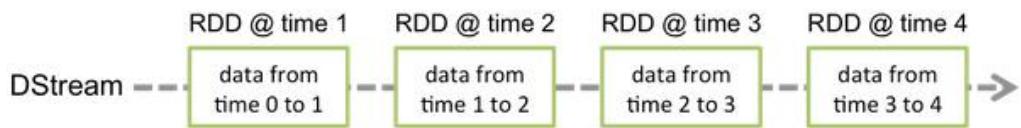
### DStream

The DStream (Discretized Stream) is an abstract concept provided by the Spark Streaming.

The DStream is a continuous data stream which is obtained from the data source or transformed and generated by the inflow. In essence, a DStream is a series of continuous RDDs. The RDD is a distributed dataset which can be read only and divided into partitions.

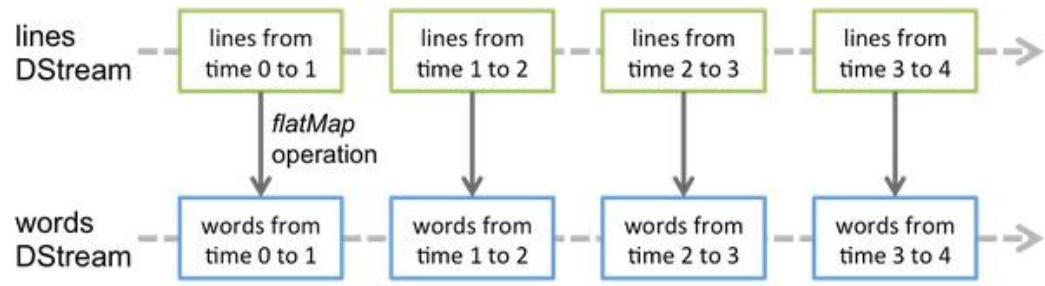
Each RDD in the DStream contains the data of a partition, as shown in [Figure 2-266](#).

**Figure 2-266** Relationship between DStream and RDD



All operators applied in the DStream are translated to the operations in the lower RDD, as shown in [Figure 2-267](#). The transformation of the lower RDDs is calculated by using the Spark engine. Most operation details are concealed in the DStream operators and High-level APIs are provided for developers.

Figure 2-267 DStream operator transfer



## Structured Streaming Basic Concepts

- **Input Source**

Input data sources. Data sources must support data replay based on the offset. Different data sources have different fault tolerance capabilities.

- **Sink**

Data output. Sink must support idempotence write operations. Different Sinks have different fault tolerance capabilities.

- **outputMode**

Result output mode, which can be:

- Complete Mode: The entire updated result table will be written to the external storage. It is up to the storage connector to decide how to handle writing of the entire table.
- Append Mode: If an interval is triggered, only the new rows appended in the Result Table will be written into an external system. This mode is applicable only to a result set that has already existed and will not be updated.
- Update Mode: If an interval is triggered, only updated data in the result table will be written into an external system, which is the difference between the Complete Mode and Update Mode.

- **Trigger**

Output trigger. Currently, the following trigger types are supported:

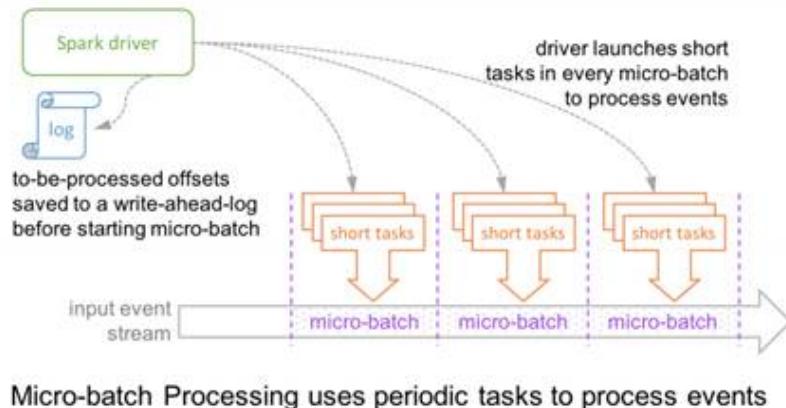
- Default: Micro-batch mode. After a batch is complete, the next batch is automatically executed.
- Specific interval: Processing is performed at a specific interval.
- One-time execution: Query is performed only once.
- Continuous mode: This is an experimental feature. In this mode, the stream processing delay can be decreased to 1 ms.

Structured Streaming supports the micro-batch mode and continuous mode. The micro-batch mode cannot ensure low-delay but has a larger throughput within the same time. The continuous mode is suitable for data processing requiring millisecond-level delay, which is an experimental feature.

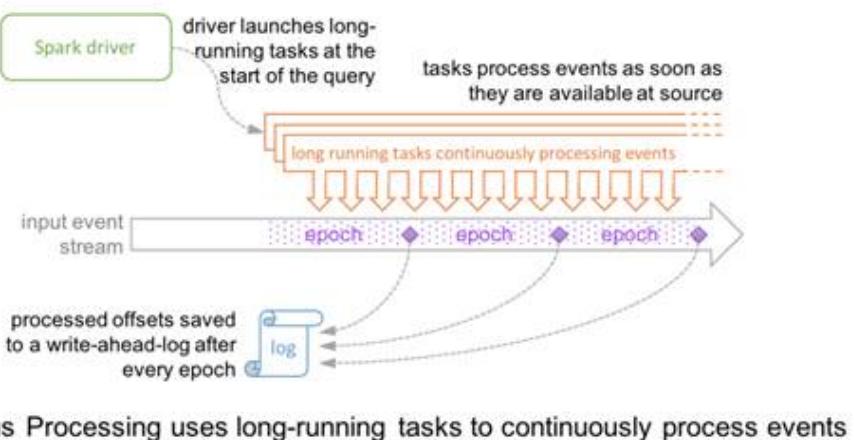
### NOTE

In the current version, if the stream-to-batch joins function is required, **outputMode** must be set to **Append Mode**.

**Figure 2-268** Running process in micro-batch mode



**Figure 2-269** Running process in continuous mode



### 2.17.1.3 Development Process

Spark includes Spark Core, Spark SQL and Spark Streaming, whose development processes are the same.

[Figure 2-270](#) and [Table 2-156](#) describe the stages in the development process.

Figure 2-270 Spark development process

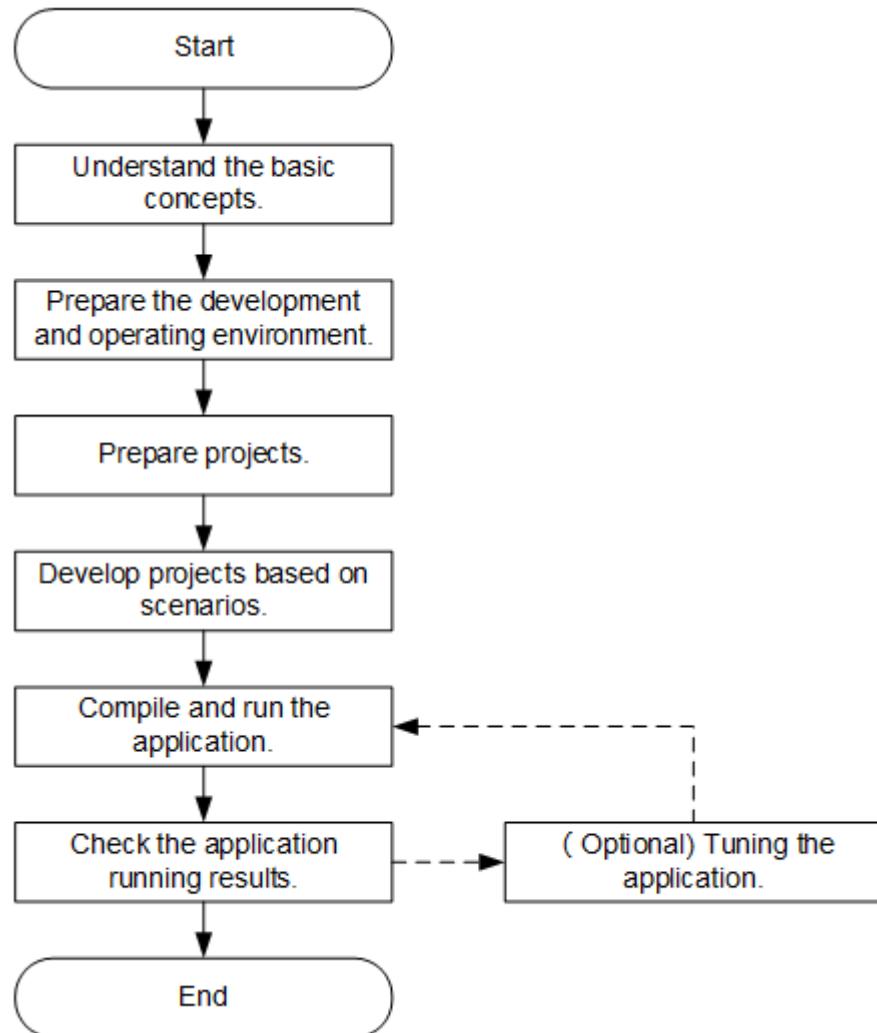


Table 2-156 Description of Spark development process

Stage	Description	Reference
Understand the basic concepts.	Before the application development, the basic concepts of Spark are required to be understood. Choose the concepts required to be understood based on the actual scenario. The basic concepts include the basic concept of Spark Core, basic concept of Spark SQL and basic concept of Spark Streaming.	<a href="#">Basic Concepts</a>
Prepare the development and operating environment.	The Spark application is developed in Scala, Java, and Python. The IDEA tool is recommended to prepare development environments in different languages based on the reference. The running environment of Spark is the Spark client. Install and configure the client based on the reference.	<a href="#">Development and Operating Environment</a>

Stage	Description	Reference
Prepare projects.	Spark provides sample projects in various scenarios. Sample projects can be imported for studying. Or you can create a new Spark project based on the reference.	<a href="#">Configuring and Importing Sample Projects</a> <a href="#">Creating a New Project (Optional)</a>
Develop projects based on scenarios.	Sample projects in different languages including Scala, Java, and Python are provided. Sample projects in different scenarios including Streaming, SQL, JDBC client program, and Spark on HBase are also provided. This helps users to learn about the programming interfaces of all Spark components quickly.	<a href="#">Developing the Project</a>
Compile and run the application.	Users compile the developed application and deliver it for running based on the reference.	<a href="#">Compiling and Running the Application</a>
Check the application running results.	Application running results are stored in the directory specified by users. Users can also check the running results through the UI.	<a href="#">Checking the Commissioning Result</a>
Tune the application.	Based on the application running results, tune the application to meet the requirements of the service scenario. After application tuning, compile and run the application again.	"Component Operation Guide" > "Using Spark" > "Spark Performance Tuning" in <i>MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)</i> .

## 2.17.2 Preparing for the Environment

### 2.17.2.1 Development and Operating Environment

Table 2-157 describes the environment required for secondary development.

**Table 2-157** Development environment

Preparation Item	Description
OS	<ul style="list-style-type: none"><li>Development environment: Windows OS. Windows 7 or later is supported.</li><li>Operating environment: Windows OS or Linux OS. If the program needs to be commissioned locally, the runtime environment must be able to communicate with the cluster service plane network.</li></ul>
JDK installation	<p>Basic configuration of the development and running environment. The version requirements are as follows:</p> <p>The server and client support only the built-in OpenJDK. Other JDKs cannot be used.</p> <p>If the JAR packages of the SDK classes that need to be referenced by the customer applications run in the application process, the JDK requirements are as follows:</p> <ul style="list-style-type: none"><li>For x86 nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>Oracle JDK 1.8</li><li>IBM JDK 1.8.0.7.20 and 1.8.0.6.15</li></ul></li><li>For Arm nodes that run clients, use the following JDKs:<ul style="list-style-type: none"><li>OpenJDK 1.8.0_402 (built-in JDK, which can be obtained from the <b>JDK</b> folder in the cluster client installation directory.)</li><li>BiSheng JDK 1.8.0_402</li></ul></li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>For security purposes, the server supports only TLS V1.2 or later.</li><li>By default, the IBM JDK supports only TLS V1.0. If the IBM JDK is used, set the startup parameter <b>com.ibm.jsse2.overrideDefaultTLS</b> to <b>true</b>. After the setting, the IBM JDK supports TLS V1.0, V1.1, and V1.2. For details, see <a href="https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls">https://www.ibm.com/docs/en/sdk-java-technology/8?topic=customization-matching-behavior-sslcontextgetinstancetls-oracle#matchsslcontext_tls</a>.</li><li>For details about the BiSheng JDK, see <a href="https://www.hikunpeng.com/en/developer/devkit/compiler/jdk">https://www.hikunpeng.com/en/developer/devkit/compiler/jdk</a>.</li></ul>

Preparation Item	Description
IntelliJ IDEA installation and configuration	<p>It is a tool used to develop Spark applications. Version 2019.1 or other compatible versions are recommended.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If the IBM JDK is used, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK.</li><li>• If the Oracle JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK.</li><li>• If the Open JDK is used, ensure that the JDK configured in IntelliJ IDEA is the Open JDK.</li><li>• Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA programs.</li></ul>
Installation of Maven	Basic configurations of the development environment. It can be used for project management throughout the lifecycle of software development.
Scala installation	It is the basic configuration for the Scala development environment. The required version is 2.12.14.
Scala plug-in installation	It is the basic configuration for the Scala development environment. The required version is 2018.2.11 or other compatible versions.
Editra installation	Editra is an editor in the Python development environment and is used to compile Python programs. You can also use other IDEs for Python programming.
7-zip	Used to decompress .zip and .rar packages, 7-Zip 16.04 is supported.
Python installation	Its version must be 3.7 or later.

## Preparing a Runtime Environment

During application development, you need to prepare the environment for code running and commissioning to verify that the application is running properly.

- If the local Windows development environment can communicate with the cluster service plane network, download the cluster client to the local host; obtain the cluster configuration file required by the commissioning program; configure the network connection, and commission the program in Windows.
  - a. **Accessing FusionInsight Manager of an MRS Cluster** In the upper right corner of the home page, click **Download Client**. Set **Select Client Type** to **Configuration Files Only**. Select the platform type based on the type of the node where the client is to be installed (select **x86\_64** for the x86 architecture and **aarch64** for the Arm architecture) and click **OK**. After the client files are packaged and generated, download the client to the local PC as prompted and decompress it.

For example, if the client file package is **FusionInsight\_Cluster\_1\_Services\_Client.tar**, decompress it to obtain the

**FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles.tar** file, and then decompress it to the **D:\FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles** directory on the local PC. The directory name cannot contain spaces.

- b. Go to the client decompression path **FusionInsight\_Cluster\_1\_Services\_ClientConfig\_ConfigFiles\Spark\config** and manually import the configuration file to the configuration file directory (usually the **resources** folder) of the Spark sample project.

**Table 2-158** describes the main configuration files.

**Table 2-158** Configuration files

File	Function
carbon.properties	CarbonData configuration file
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.
spark-defaults.conf	Configures Spark parameters.
spark-env.sh	Spark environment variable configuration file

- c. During application development, if you need to commission the application in the local Windows system, copy the content in the **hosts** file in the decompression directory to the **hosts** file of the node where the client is located. Ensure that the local host can communicate correctly with the hosts listed in the **hosts** file in the decompression directory.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- The local **hosts** file in a Windows environment is stored in, for example, **C:\WINDOWS\system32\drivers\etc\hosts**.

- If you use the Linux environment for commissioning, you need to prepare the Linux node where the cluster client is to be installed and obtain related configuration files.
  - a. Install the client in a directory, for example, **/opt/hadoopclient**, on the node.  
Ensure that the difference between the client time and the cluster time is less than 5 minutes.  
For details about how to install a cluster client, see [Installing a Client](#).
  - b. **Accessing FusionInsight Manager of an MRS Cluster**. Download the cluster client software package to the active management node and decompress it. Then, log in to the active management node as user **root**. Go to the decompression path of the cluster client and copy all configuration files in the **FusionInsight\_Cluster\_1\_Services\_ClientConfig/Spark/config** directory to the **conf** directory where the compiled JAR package is stored for subsequent commissioning, for example, **/opt/hadoopclient/conf**.  
For example, if the client software package is **FusionInsight\_Cluster\_1\_Services\_Client.tar** and the download path is **/tmp/FusionInsight-Client** on the active management node, run the following command:  

```
cd /tmp/FusionInsight-Client  
tar -xvf FusionInsight_Cluster_1_Services_Client.tar  
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar  
cd FusionInsight_Cluster_1_Services_ClientConfig  
scp Spark/config/* root@IP address of the client node:/opt/  
hadoopclient/conf
```

**Table 2-159** Configuration files

File	Function
carbon.properties	CarbonData configuration file
core-site.xml	Configures HDFS parameters.
hdfs-site.xml	Configures HDFS parameters.
hbase-site.xml	Configures HBase parameters.
hive-site.xml	Configures Hive parameters.
jaas-zk.conf	Java authentication configuration file
log4j-executor.properties	executor log configuration file
mapred-site.xml	Hadoop mapreduce configuration file
ranger-spark-audit.xml	Ranger audit log configuration file
ranger-spark-security.xml	Ranger permission management configuration file
yarn-site.xml	Configures Yarn parameters.

File	Function
spark-defaults.conf	Configures Spark parameters.
spark-env.sh	Spark environment variable configuration file

- c. Check the network connection of the client node.

During the client installation, the system automatically configures the **hosts** file on the client node. You are advised to check whether the **/etc/hosts** file contains the host names of the nodes in the cluster. If no, manually copy the content in the **hosts** file in the decompression directory to the **hosts** file on the node where the client resides, to ensure that the local host can communicate with each host in the cluster.

### 2.17.2.2 Configuring and Importing Sample Projects

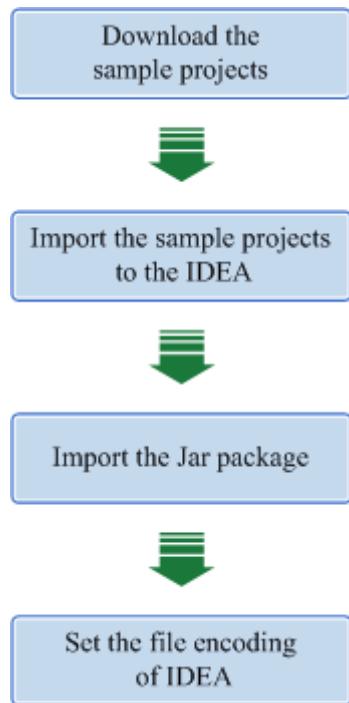
#### Scenario

Spark provides sample projects for multiple scenarios, including Java projects and Scala projects. This helps users to learn Spark projects quickly.

Import methods of Java and Scala projects are the same. Sample projects developed by using the Python do not need to be imported, and you only need to open the Python file (\*.py).

The import of Java sample codes is used as a sample in the following procedure. [Figure 2-271](#) shows the procedure of importing sample projects.

**Figure 2-271** Procedure of importing sample projects



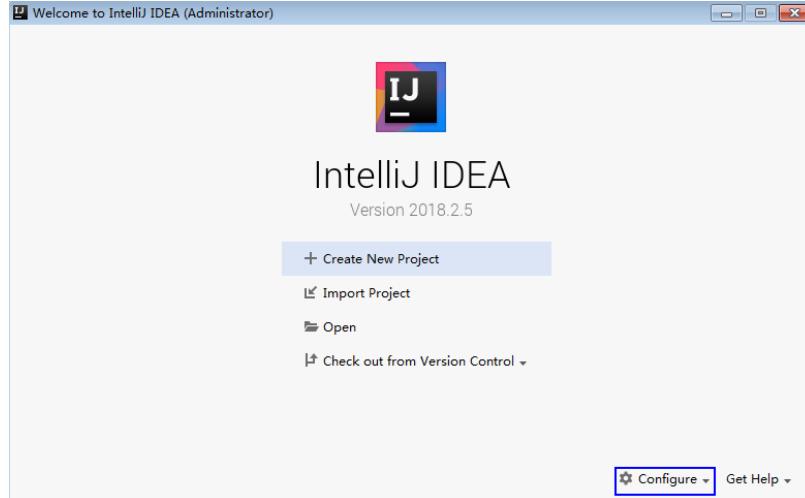
## Procedure

**Step 1** Obtain multiple sample projects such as Scala and Spark Streaming in the **sparknormal-examples** folder in the **spark-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#)

**Step 2** After the IntelliJ IDEA and JDK are installed, configure the JDK in IntelliJ IDEA.

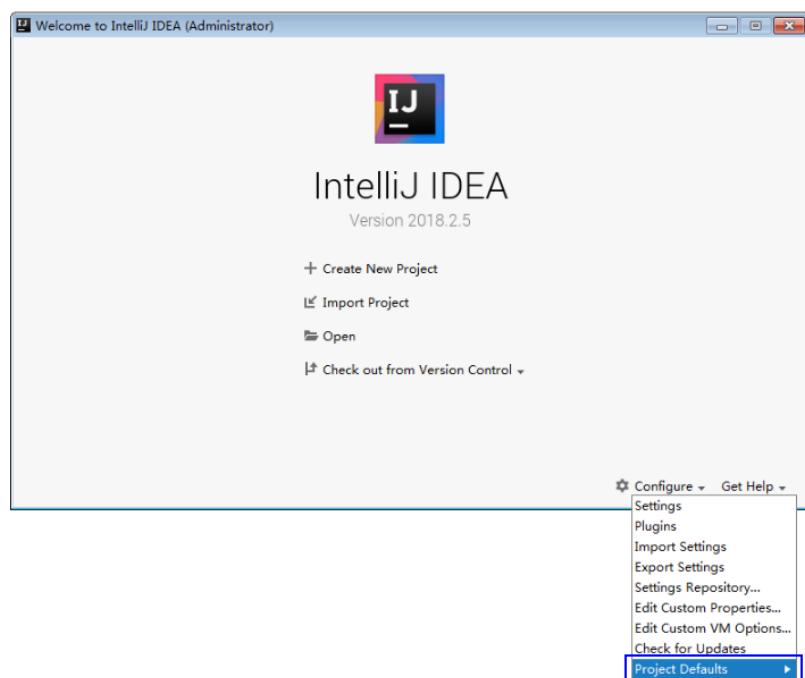
1. Start the IntelliJ IDEA and select **Configure**.

**Figure 2-272 Quick Start**



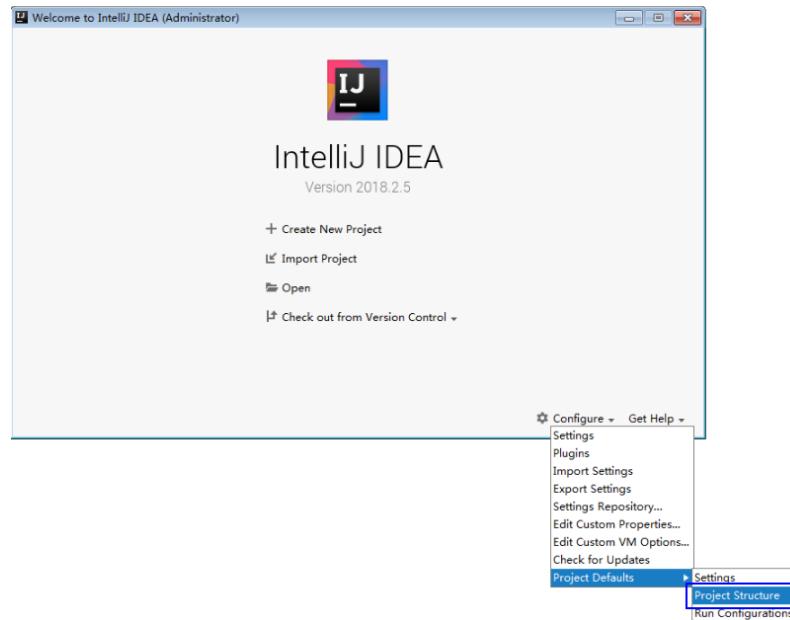
2. Select **Project Defaults** from the **Configure** drop-down list.

**Figure 2-273 Configure**



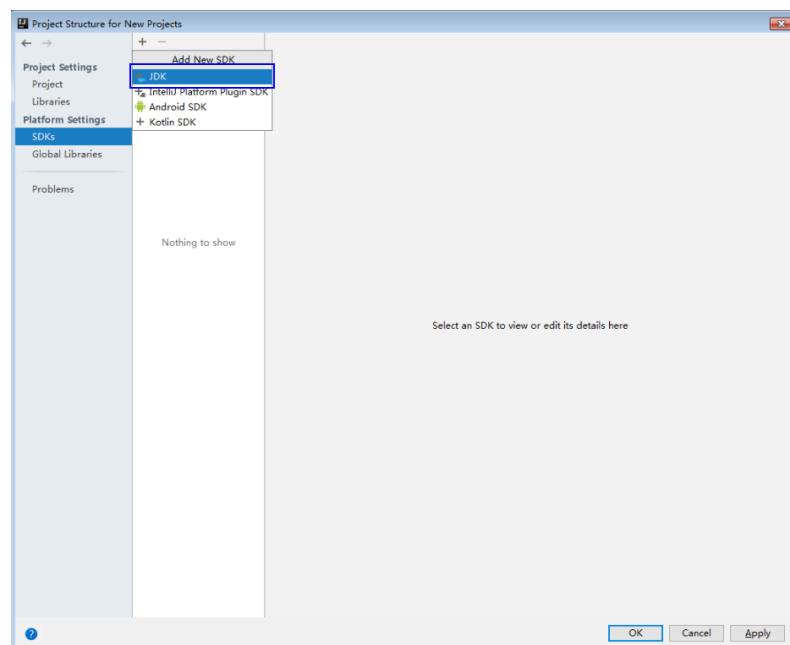
3. Select **Project Structure** from **Project Defaults**.

Figure 2-274 Project Defaults



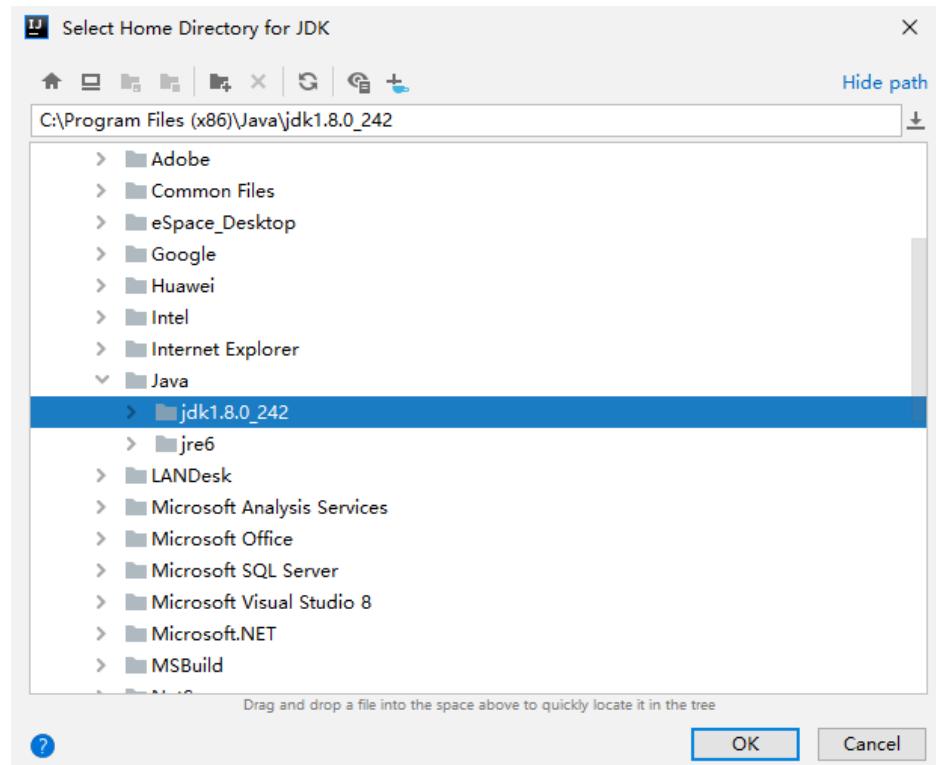
4. On the displayed **Project Structure** page, select **SDKs** and click the plus sign to add the JDK.

Figure 2-275 Adding the JDK



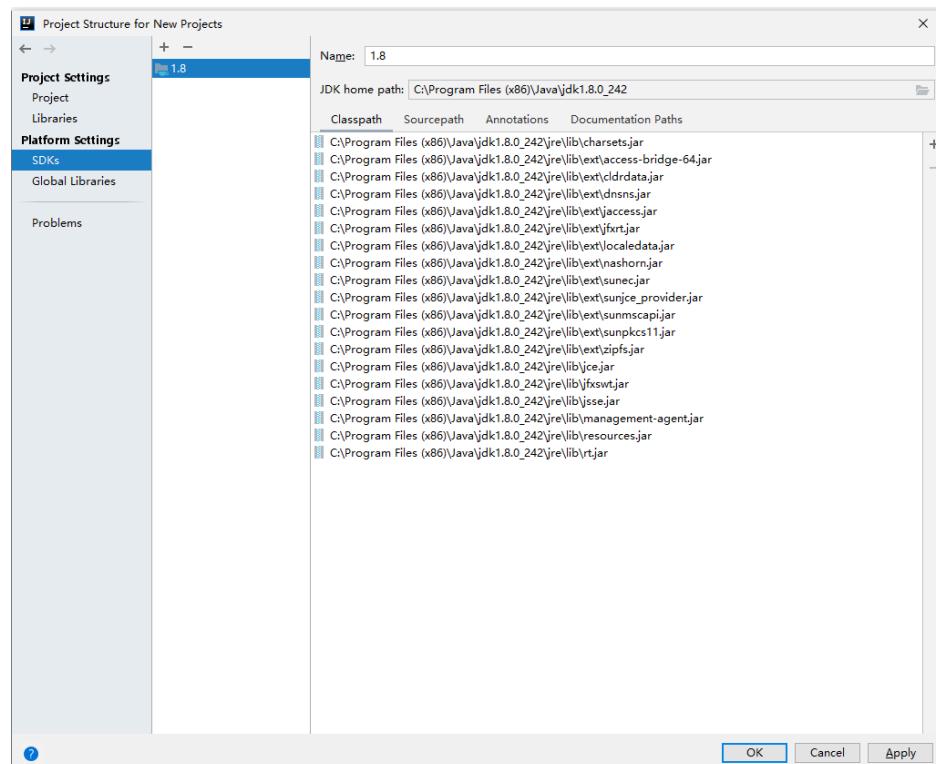
5. In the displayed **Select Home Directory for JDK** window, select a home directory for the JDK and click **OK**.

Figure 2-276 Selecting a home directory for the JDK



- After selecting the JDK, click **OK** to complete the configuration.

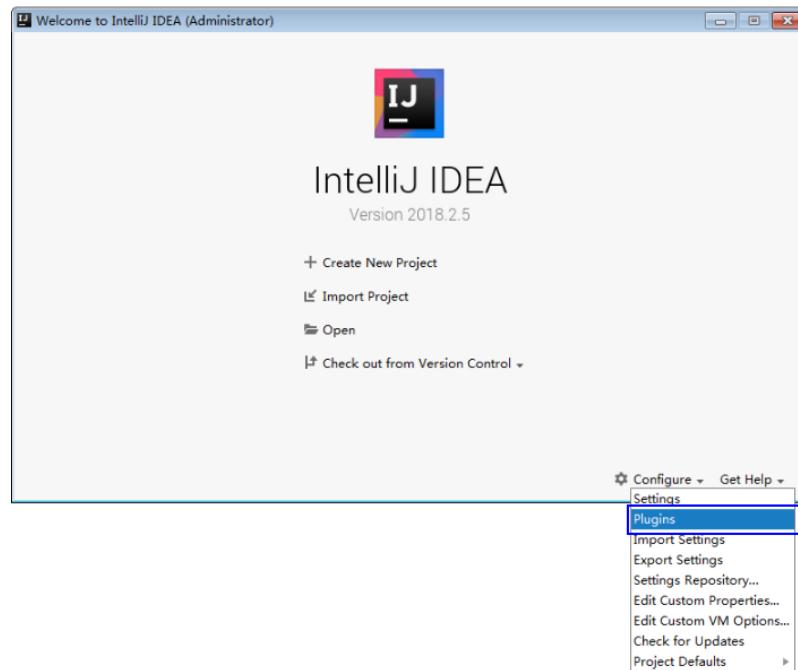
Figure 2-277 Completing the configuration



**Step 3** (Optional) If the Scala development environment is used, install the Scala plug-in in IntelliJ IDEA.

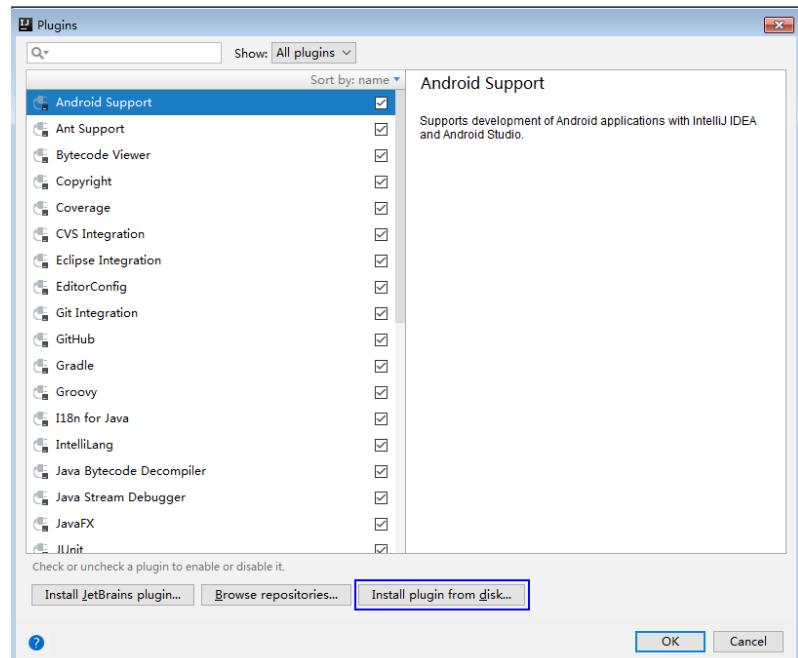
1. Select **Plugins** from the **Configure** drop-down list.

**Figure 2-278 Plugins**

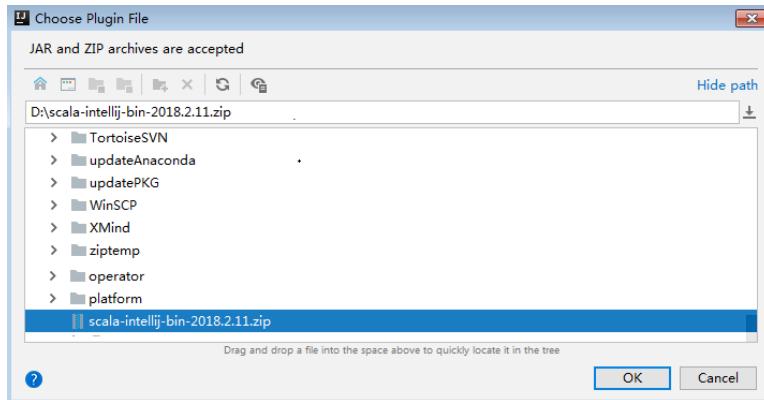


2. On the **Plugins** page, select **Install plugin from disk**.

**Figure 2-279 Install plugin from disk**

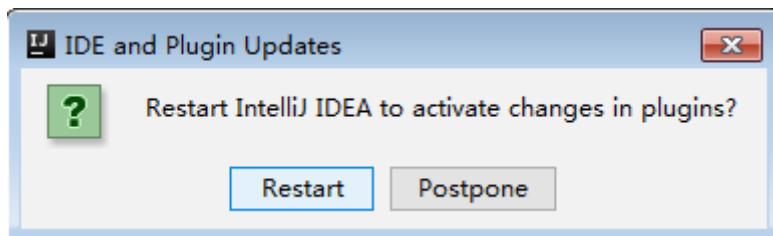


3. On the **Choose Plugin File** page, select the Scala plugin file of the corresponding version and click **OK**.



4. On the **Plugins** page, click **Apply** to install the Scala plugin.
5. On the displayed **Plugins Changed** page, click **Restart** for the configuration to take effect.

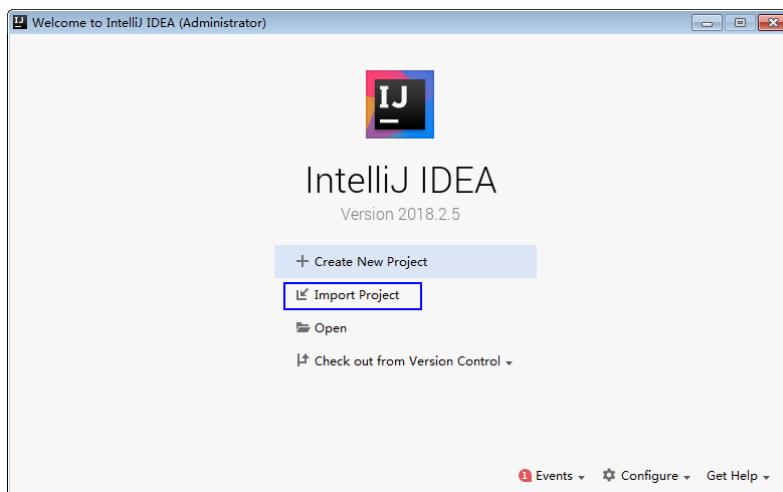
**Figure 2-280 Plugins Changed**



**Step 4** Import the Java sample projects to the IDEA.

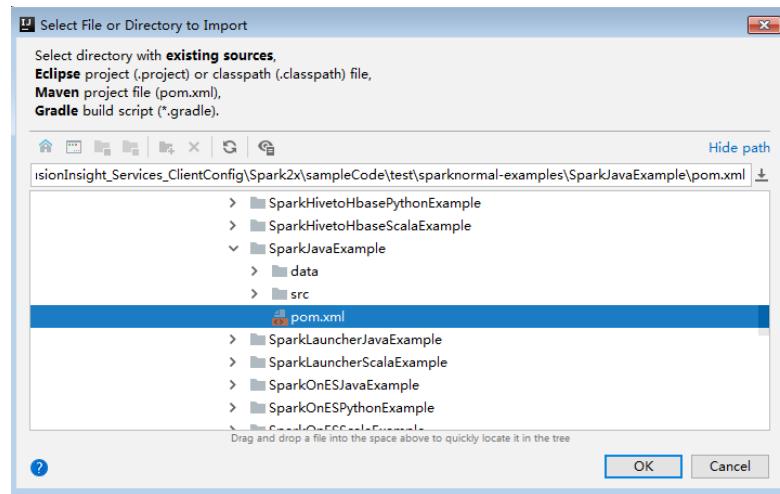
1. Start the IntelliJ IDEA. On the **Quick Start** page, select **Import Project**. Alternatively, for the used IDEA tool, add the project directly from the IDEA home page. Select **File > Import project...** to import projects.

**Figure 2-281 Import Project (on the Quick Start page)**



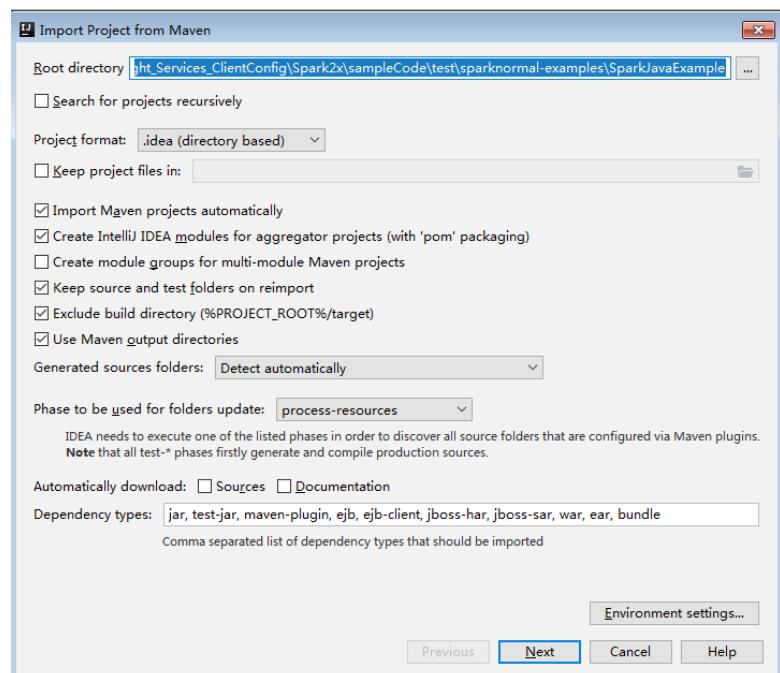
2. Select the directory to store the imported projects and the pom file, and click **OK**.

Figure 2-282 Select File or Directory to Import



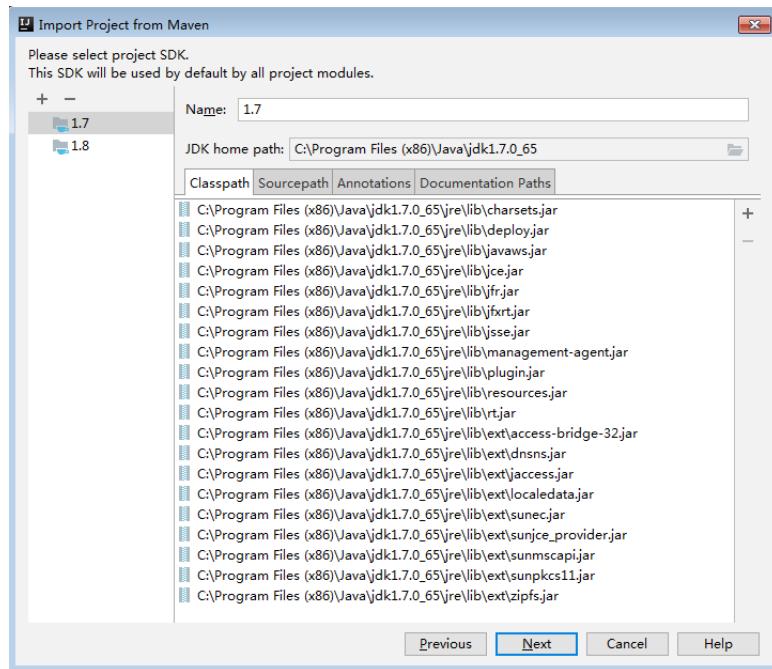
3. Confirm the import directory and project name, and click **Next**.

Figure 2-283 Import Project from Maven



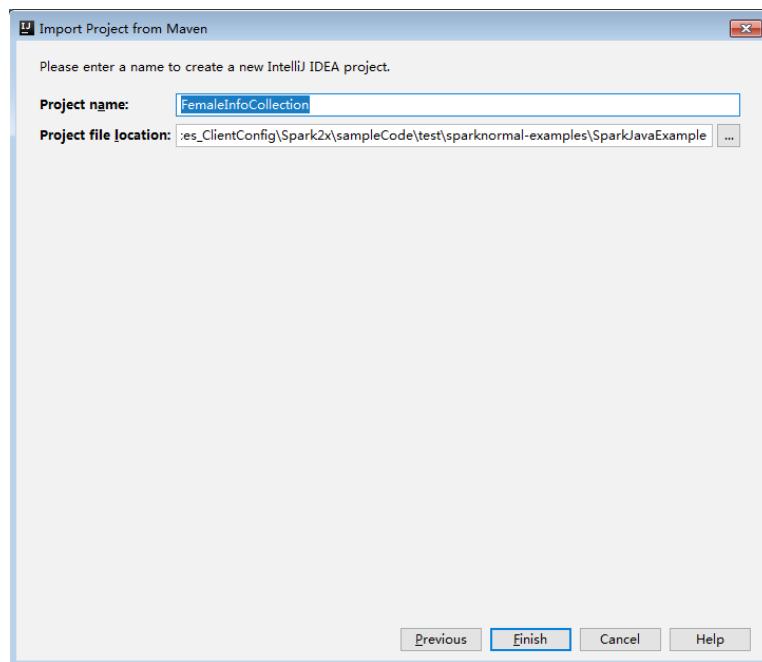
4. Select the projects to import and click **Next**.
5. Confirm the project JDK and click **Next**.

Figure 2-284 Select project SDK



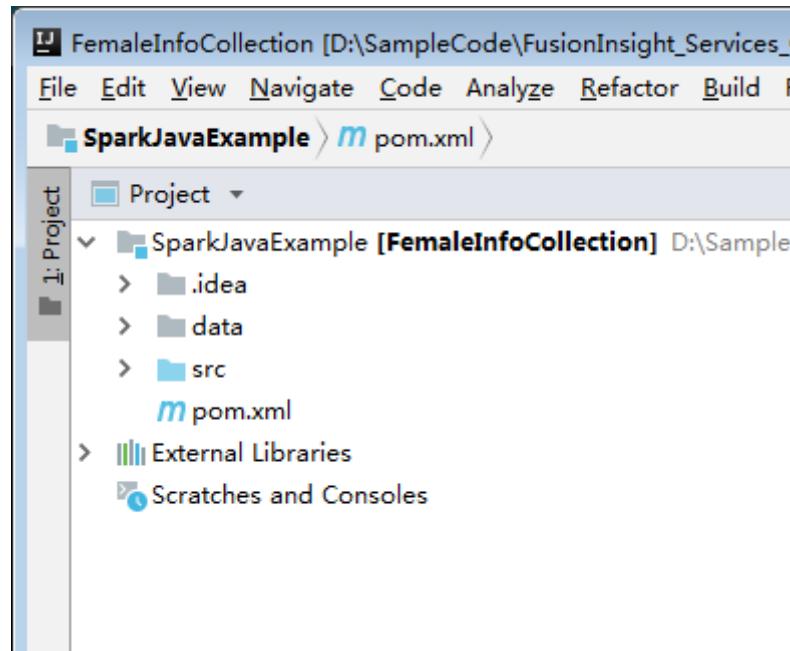
6. Confirm the project name and project file location, and click **Finish** to complete the import.

Figure 2-285 Confirm the project name and file location



7. After the import, the imported projects are displayed on the IDEA home page.

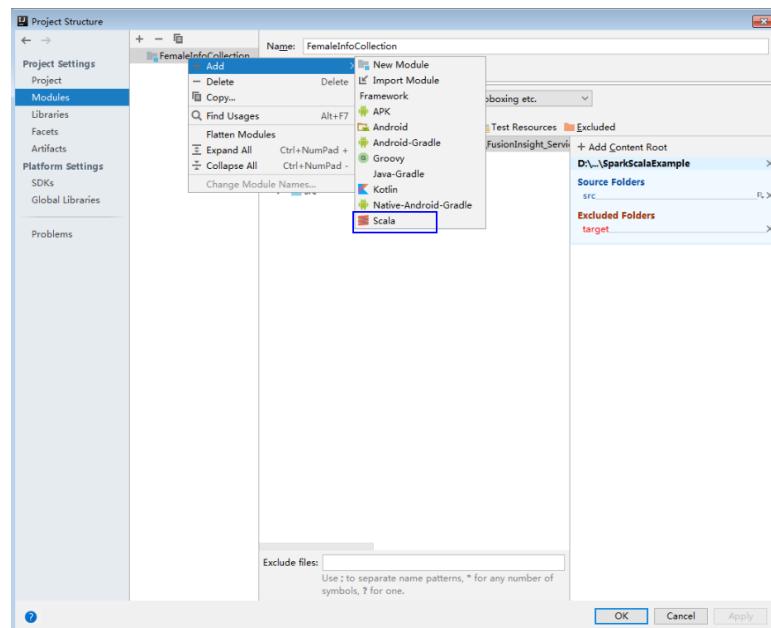
Figure 2-286 Imported projects



**Step 5** (Optional) If a sample application developed in Scala is imported, configure the language for the project.

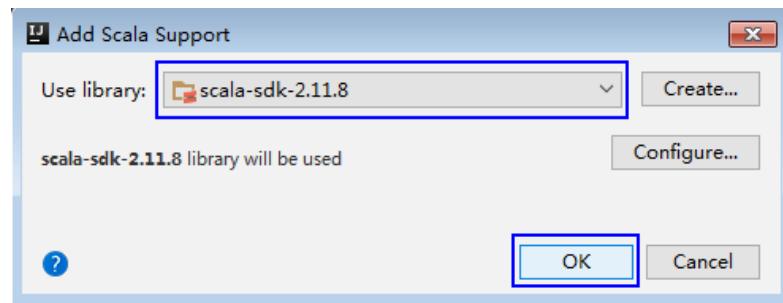
1. On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
2. Choose **Modules**, right-click the project name, and choose **Add > Scala**.

Figure 2-287 Selecting the Scala language



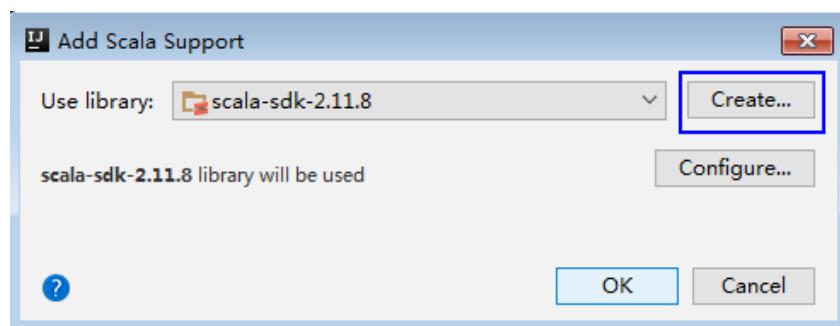
3. Wait until IDEA identifies Scala SDK, select the dependency JAR packages in the **Add Scala Support** dialog box, and then click **OK**.

Figure 2-288 Add Scala Support



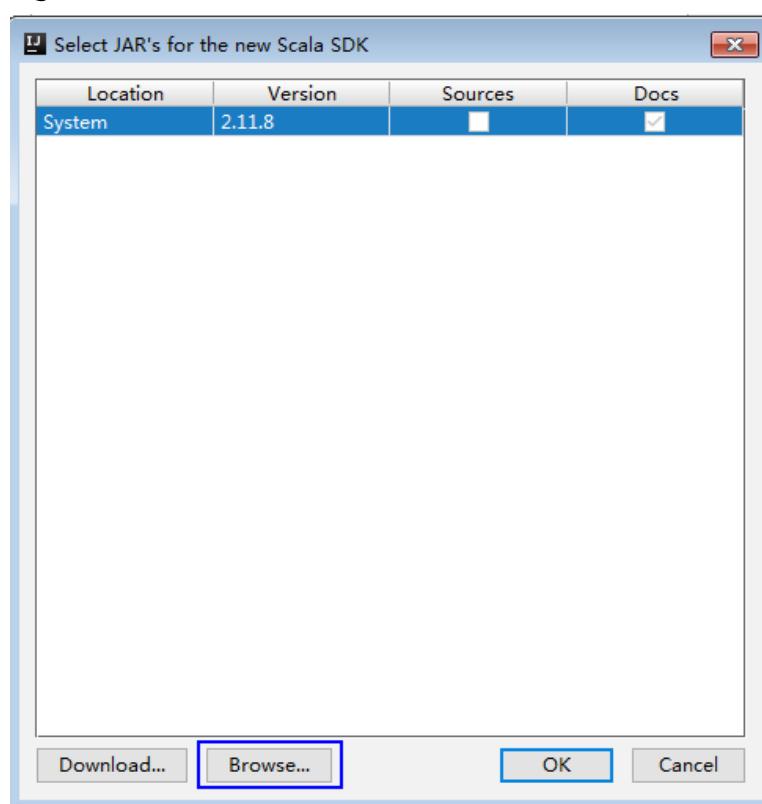
4. If IDEA fails to identify Scala SDK, you are required to create a Scala SDK.
  - a. Click **Create...**.

Figure 2-289 Create...



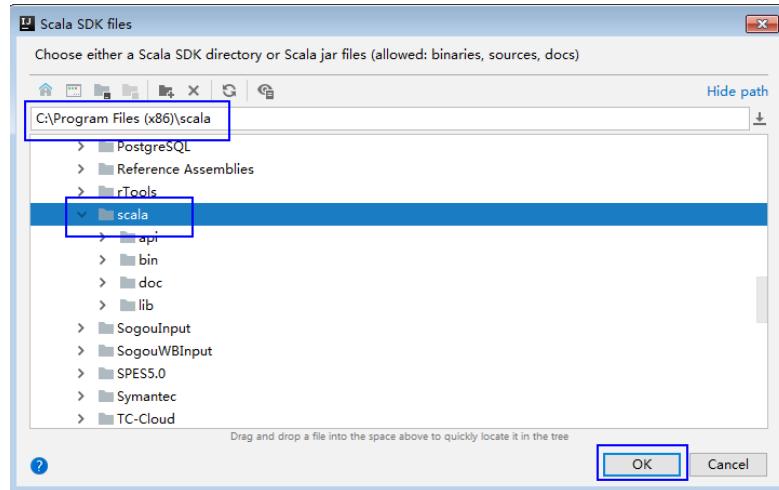
- b. On the **Select JAR's for the new Scala SDK** page, click **Browse...**

Figure 2-290 Select JAR's for the new Scala SDK



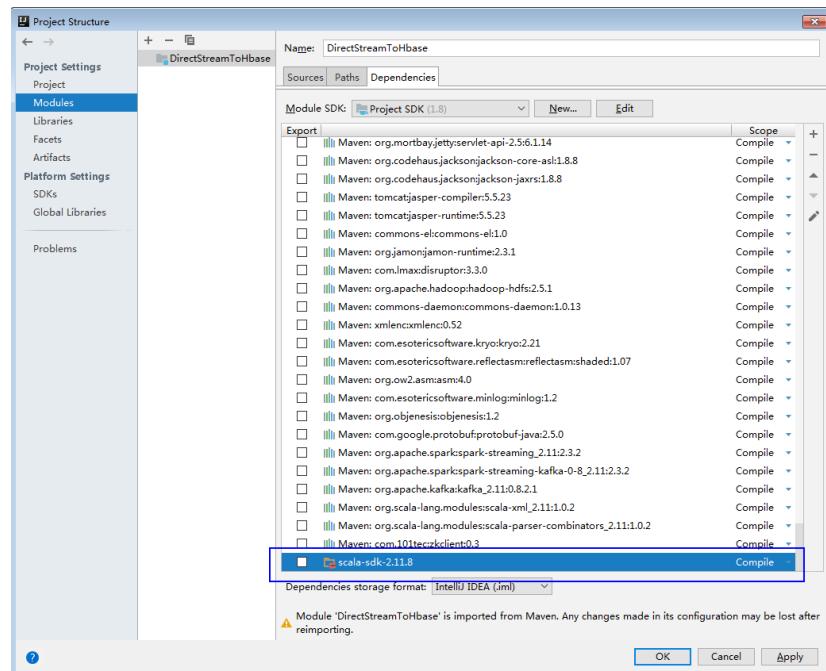
- c. On the **Scala SDK files** page, select the **scala sdk** directory, and then click **OK**.

**Figure 2-291 Scala SDK files**



5. Click **OK**.

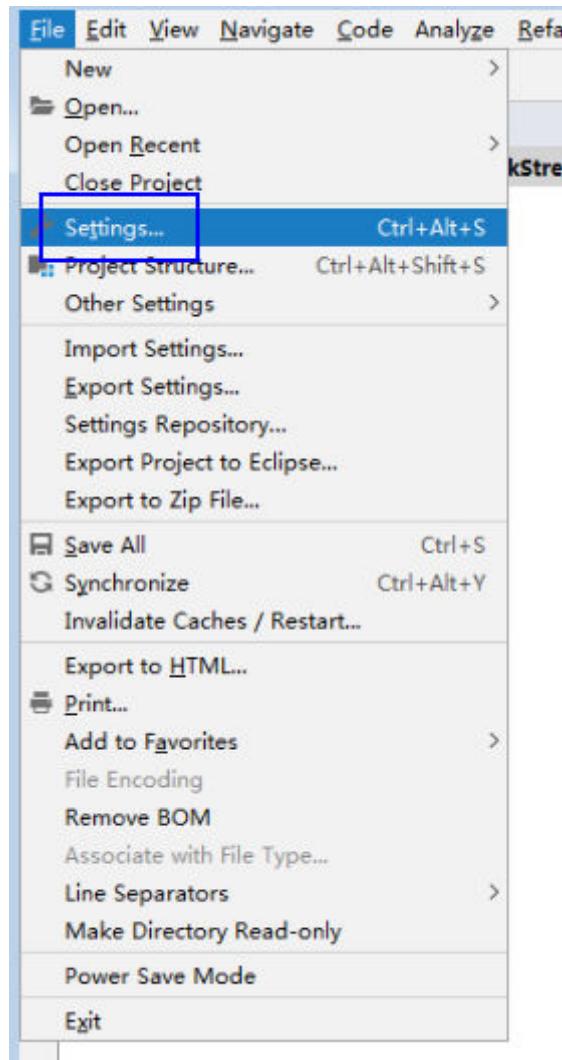
**Figure 2-292 Successful configuration**



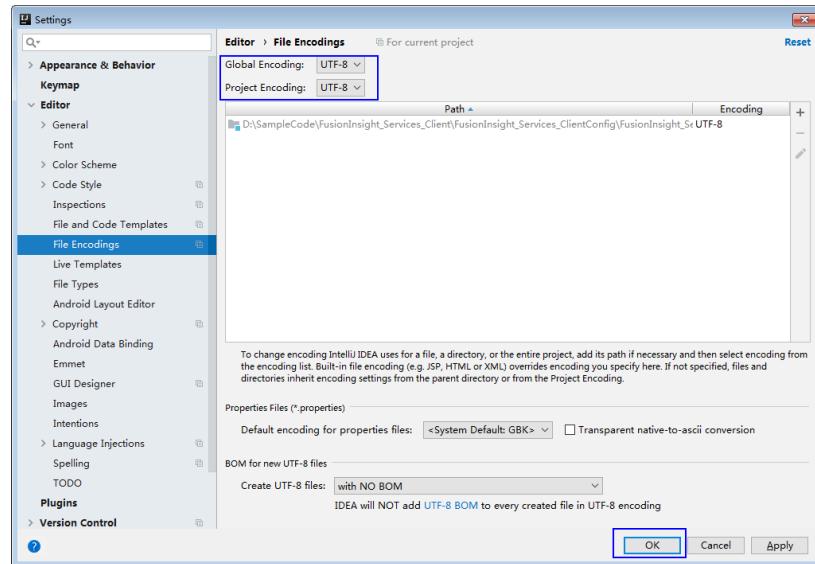
**Step 6** Set the file encoding of IDEA and solve the display of garble characters.

1. On the IDEA home page, choose **File > Settings....**

Figure 2-293 Choosing Settings



2. Configure the encoding.
  - a. On the **Settings** page, choose **Editor > File Encodings**.
  - b. In the **Global Encoding** and **Project Encoding** drop-down lists, select **UTF-8**, respectively.
  - c. Click **Apply**.
  - d. Click **OK** to complete the encoding configuration.



----End

## Sample Code Path Description

**Table 2-160** Sample code path description

Sample Code Project	Sample Name	Sample Development Language
SparkJavaExample	Spark Core Project	Java
SparkScalaExample	Spark Core Project	Scala
SparkPythonExample	Spark Core Project	Python
SparkSQLJavaExample	Spark SQL Project	Java
SparkSQLScalaExample	Spark SQL Project	Scala
SparkSQLPythonExample	Spark SQL Project	Python
SparkThriftServerJavaExample	Accessing the Spark SQL Through JDBC	Java
SparkThriftServerScalaExample	Accessing the Spark SQL Through JDBC	Scala
SparkOnHbaseJavaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Java
SparkOnHbaseScalaExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkOnHbasePythonExample-AvroSource	Spark on HBase-Performing Operations on Data in Avro Format	Python
SparkOnHbaseJavaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Java
SparkOnHbaseScalaExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Scala
SparkOnHbasePythonExample-HbaseSource	Spark on HBase-Performing Operations on the HBase Data Source	Python
SparkOnHbaseJavaExample-JavaHBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Java
SparkOnHbaseScalaExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Scala
SparkOnHbasePythonExample-HBaseBulkPutExample	Spark on HBase-Using the BulkPut Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Java
SparkOnHbaseScalaExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Scala
SparkOnHbasePythonExample-HBaseBulkGetExample	Spark on HBase-Using the BulkGet Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Java
SparkOnHbaseScalaExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Scala
SparkOnHbasePythonExample-HBaseBulkDeleteExample	Spark on HBase-Using the BulkDelete Interface	Python
SparkOnHbaseJavaExample-JavaHBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Java
SparkOnHbaseScalaExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Scala
SparkOnHbasePythonExample-HBaseBulkLoadExample	Spark on HBase-Using the BulkLoad Interface	Python

Sample Code Project	Sample Name	Sample Development Language
SparkOnHbaseJavaExample- JavaHBaseForEachPartitionExam- ple	Spark on HBase-Using the foreachPartition Interface	Java
SparkOnHbaseScalaExample- HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Scala
SparkOnHbasePythonExample- HBaseForEachPartitionExample	Spark on HBase-Using the foreachPartition Interface	Python
SparkOnHbaseJavaExample- JavaHBaseDistributedScanExam- ple	Spark on HBase- Distributedly Scanning HBase Tables	Java
SparkOnHbaseScalaExample- HBaseDistributedScanExample	Spark on HBase- Distributedly Scanning HBase Tables	Scala
SparkOnHbasePythonExample- HBaseDistributedScanExample	Spark on HBase- Distributedly Scanning HBase Tables	Python
SparkOnHbaseJavaExample- JavaHBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Java
SparkOnHbaseScalaExample- HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Scala
SparkOnHbasePythonExample- HBaseMapPartitionExample	Spark on HBase-Using the mapPartition Interface	Python
SparkOnHbaseJavaExample- JavaHBaseStreamingBulkPutEx- ample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Java
SparkOnHbaseScalaExample- HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Scala
SparkOnHbasePythonExample- HBaseStreamingBulkPutExample	Spark on HBase-Writing Data to HBase Tables In Batches Using SparkStreaming	Python
SparkHbasetoHbaseJavaExample	Reading Data from HBase and Write It Back to HBase	Java
SparkHbasetoHbaseScalaExam- ple	Reading Data from HBase and Write It Back to HBase	Scala

Sample Code Project	Sample Name	Sample Development Language
SparkHbasetoHbasePythonExample	Reading Data from HBase and Write It Back to HBase	Python
SparkHivetoHbaseJavaExample	Reading Data from Hive and Write It to HBase	Java
SparkHivetoHbaseScalaExample	Reading Data from Hive and Write It to HBase	Scala
SparkHivetoHbasePythonExample	Reading Data from Hive and Write It to HBase	Python
SparkStreamingKafka010JavaExample	Streaming Connecting to Kafka0-10	Java
SparkStreamingKafka010ScalaExample	Streaming Connecting to Kafka0-10	Scala
SparkStructuredStreamingJavaExample	Structured Streaming Project	Java
SparkStructuredStreamingScalaExample	Structured Streaming Project	Scala
SparkStructuredStreamingPythonExample	Structured Streaming Project	Python
StructuredStreamingADScalaExample	Structured Streaming Stream-Stream Join	Scala
SparkOnEsJavaExample	Spark on Elasticsearch	Java
SparkOnEsScalaExample	Spark on Elasticsearch	Scala
SparkOnEsPythonExample	Spark on Elasticsearch	Python
SparkOnSolrJavaExample	Spark on Solr	Java
SparkOnSolrScalaExample	Spark on Solr	Scala
SparkOnSolrPythonExample	Spark on Solr	Python
StructuredStreamingStateScalaExample	Structured Streaming Status Operation	Scala
SparkHbasetoCarbonJavaExample	Synchronizing HBase Data from Spark to CarbonData	Java
SparkOnHudiJavaExample	Using Spark to Perform Basic Hudi Operations	Java
SparkOnHudiPythonExample	Using Spark to Perform Basic Hudi Operations	Python

Sample Code Project	Sample Name	Sample Development Language
SparkOnHudiScalaExample	Using Spark to Perform Basic Hudi Operations	Scala
SparkOnClickHouseJavaExample	Using Spark to Perform Basic ClickHouse Operations	Java
SparkOnClickHouseScalaExample	Using Spark to Perform Basic ClickHouse Operations	Scala
SparkOnClickHousePythonExample	Using Spark to Perform Basic ClickHouse Operations	Python

### 2.17.2.3 Creating a New Project (Optional)

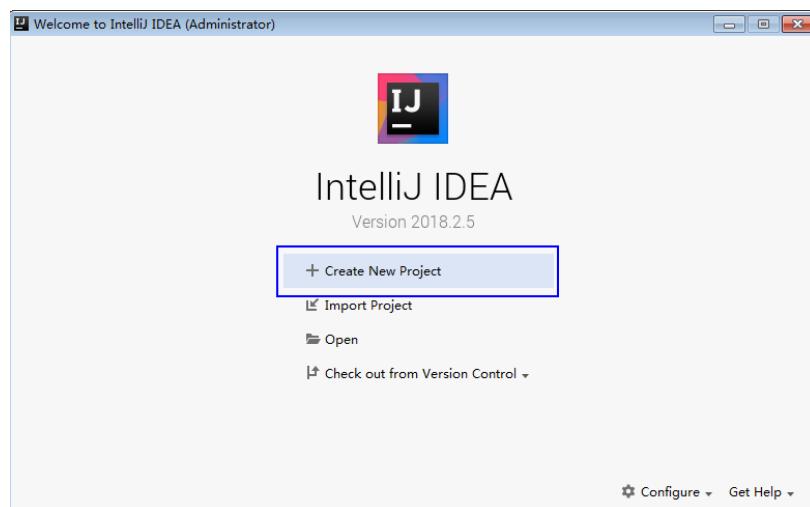
#### Scenario

Besides importing Spark sample projects, the IDEA can be used to create a Spark project. A Scala project is used as an example in the following steps.

#### Procedure

**Step 1** Start the IDEA and select **Create New Project**.

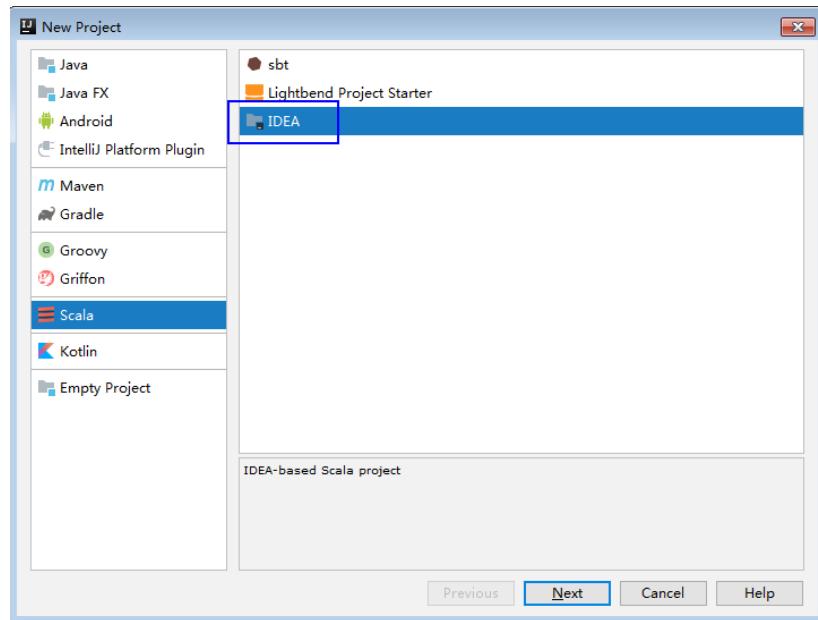
**Figure 2-294** Create a project



**Step 2** On the **New Project** page, select **Scala** as the development environment, select **IDEA**, and click **Next**.

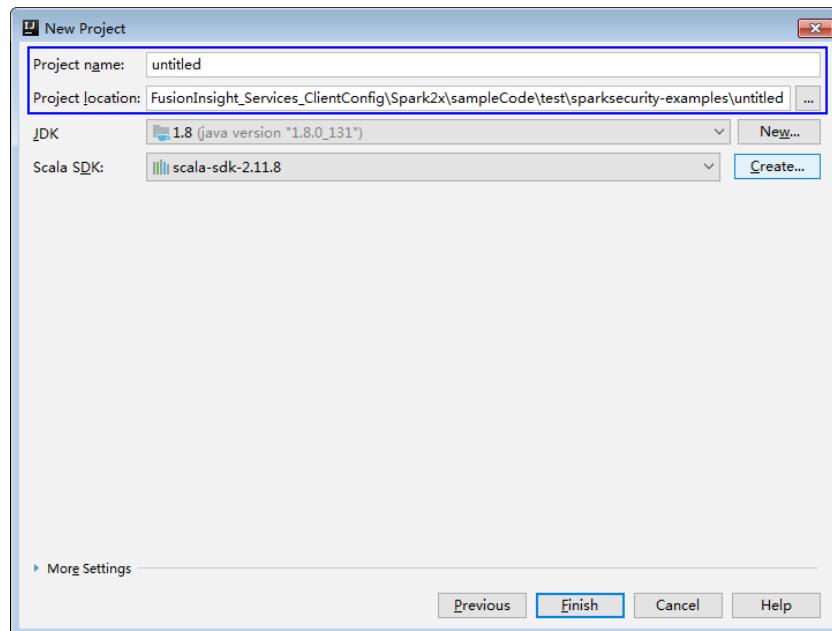
If a new project in Java is required, choose corresponding parameters.

Figure 2-295 Select the development language



**Step 3** On the project information page, specify **Project name**, **Project location**, **Project JDK**, and **Scala SDK**, and click **Finish** to complete the project creation.

Figure 2-296 Add the project information



----End

#### 2.17.2.4 Configuring the Python3 Sample Project

##### Scenario

To run the Python3 interface sample code of the Spark component of MRS, perform the following operations.

## Procedure

### Step 1 Install Python3 of 3.7 or a higher version on the client.

The Python version can be viewed by running the **python3** command on the CLI of the client. The following information indicates that the Python version is 3.8.2.

```
Python 3.8.2 (default, Jun 23 2020, 10:26:03)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

### Step 2 Setuptools 47.3.1 must be installed on the client.

To obtain the software, visit the official websites.

<https://pypi.org/project/setuptools/#files>

Copy the downloaded setuptools package to the client, decompress the package, go to the decompressed directory, and run the **python3 setup.py install** command in the CLI of the client.

The following information indicates that setuptools 47.3.1 is installed successfully.

```
Finished processing dependencies for setuptools==47.3.1
```

### Step 3 Install Python on the client.

1. Obtain the sample project folder **python3-examples** in the **src\hive-examples** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
2. Go to the **python3-examples** folder.
3. Go to the **dependency\_python3.6**, **dependency\_python3.7**, or **dependency\_python3.8** folder based on the Python3 version.
4. Run the **whereis easy\_install** command to find the path of the **easy\_install** program. If there are multiple paths, run the **easy\_install --version** command to select the **easy\_install** path corresponding to the **setuptools** version, for example, **/usr/local/bin/easy\_install**.
5. Run the **easy\_install** command to install the EGG files in the **dependency\_python3.x** folder in sequence. Example:

```
/usr/local/bin/easy_install future-0.18.2-py3.8.egg
```

If the following information is displayed, the EGG file is successfully installed.

```
Finished processing dependencies for future==0.18.2
```

----End

## 2.17.3 Developing the Project

### 2.17.3.1 Spark Core Project

#### 2.17.3.1.1 Overview

#### Scenarios

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20  
YuanJing,male,10  
GuoYijun,male,5  
CaiXuyu,female,50  
Liyuan,male,20  
FangBo,female,50  
LiuYang,female,20  
YuanJing,male,10  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20  
YuanJing,male,10  
CaiXuyu,female,50  
FangBo,female,50  
GuoYijun,male,5  
CaiXuyu,female,50  
Liyuan,male,20  
CaiXuyu,female,50  
FangBo,female,50  
LiuYang,female,20  
YuanJing,male,10  
FangBo,female,50  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy the log contents of **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt** respectively.
2. Create **/tmp/input** on HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:
  - a. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a hdfs dfs command provides the same function.) to create a directory.
  - b. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the **hadoop fs -put input\_data1.txt /tmp/input** and **hadoop fs -put input\_data2.txt /tmp/input** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Read the source file data.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

- Run the Scala and Java sample programs.
  - **bin/spark-submit --class**  
`com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --deploy-mode client /opt/female/FemaleInfoCollection-1.0.jar <inputPath>`
  - `<inputPath>` indicates the input path in HDFS
- Run the Python sample program.
  - **bin/spark-submit --master yarn --deploy-mode client** `/opt/female/SparkPythonExample/collectFemaleInfo.py <inputPath>`
  - `<inputPath>` indicates the input path in HDFS.

### 2.17.3.1.2 Java Example Code

## Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

## Example Code

The following code segment is only an example. For details, see the `com.huawei.bigdata.spark.examples.FemaleInfoCollection` class.

```
// Create a configuration class SparkConf, and then create a SparkContext.  
SparkSession spark = SparkSession  
    .builder()  
    .appName("CollectFemaleInfo")  
    .config("spark.some.config.option", "some-value")  
    .getOrCreate();  
  
// Read the source file data, and transfer each row of records to an element of the RDD.  
JavaRDD<String> data = spark.read()  
    .textFile(args[0])  
    .javaRDD();
```

```
// Split each column of each record, and generate a Tuple.  
JavaRDD<Tuple3<String, String, Integer>> person = data.map(new  
Function<String, Tuple3<String, String, Integer>>()  
{  
    private static final long serialVersionUID = -2381522520231963249L;  
  
    public Tuple3<String, String, Integer> call(String s) throws Exception  
{  
        // Split a row of data by commas (,).  
        String[] tokens = s.split(",");  
  
        // Integrate the three split elements to a ternary Tuple.  
        Tuple3<String, String, Integer> person = new Tuple3<String, String, Integer>(tokens[0], tokens[1],  
Integer.parseInt(tokens[2]));  
        return person;  
    }  
});  
  
// Use the filter function to filter the data information about the time that female netizens spend online.  
JavaRDD<Tuple3<String, String, Integer>> female = person.filter(new  
Function<Tuple3<String, String, Integer>, Boolean>()  
{  
    private static final long serialVersionUID = -4210609503909770492L;  
  
    public Boolean call(Tuple3<String, String, Integer> person) throws Exception  
{  
        // Filter the records of which the gender in the second column is female.  
        Boolean isFemale = person._2().equals("female");  
        return isFemale;  
    }  
});  
  
// Aggregate the total time that each female netizen spends online.  
JavaPairRDD<String, Integer> females = female.mapToPair(new PairFunction<Tuple3<String, String,  
Integer>, String, Integer>()  
{  
    private static final long serialVersionUID = 8313245377656164868L;  
  
    public Tuple2<String, Integer> call(Tuple3<String, String, Integer> female) throws Exception  
{  
        // Extract the two columns representing the name and online time for the sum of online time by  
name during further operations.  
        Tuple2<String, Integer> femaleAndTime = new Tuple2<String, Integer>(female._1(), female._3());  
        return femaleAndTime;  
    }  
});  
JavaPairRDD<String, Integer> femaleTime = females.reduceByKey(new Function2<Integer, Integer,  
Integer>()  
{  
    private static final long serialVersionUID = -3271456048413349559L;  
  
    public Integer call(Integer integer, Integer integer2) throws Exception  
{  
        // Sum two online time durations of the same female netizen.  
        return (integer + integer2);  
    }  
});  
  
// Filter the information about female netizens who spend more than 2 hours online.  
JavaPairRDD<String, Integer> rightFemales = females.filter(new Function<Tuple2<String, Integer>,  
Boolean>()  
{  
    private static final long serialVersionUID = -3178168214712105171L;  
  
    public Boolean call(Tuple2<String, Integer> s) throws Exception  
{  
        // Extract the total time that female netizens spend online, and determine whether the time is  
more than 2 hours.  
        if(s._2() > (2 * 60))  
            return true;  
        else  
            return false;  
    }  
});
```

```
{  
    return true;  
}  
return false;  
});  
  
// Print the information about female netizens who meet the requirements.  
for(Tuple2<String, Integer> d: rightFemales.collect())  
{  
    System.out.println(d._1() + "," + d._2());  
}
```

### 2.17.3.1.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see the com.huawei.bigdata.spark.examples.FemaleInfoCollection class.

Example: CollectMapper class

```
val spark = SparkSession  
.builder()  
.appName("CollectFemaleInfo")  
.config("spark.some.config.option", "some-value")  
.getOrCreate()  
  
// Read data. This code indicates the data path that the input parameter args(0) specifies.  
val text = spark.sparkContext.textFile(args(0))  
// Filter the data information about the time that female netizens spend online.  
val data = text.filter(_.contains("female"))  
// Aggregate the time that each female netizen spends online.  
val femaleData:RDD[(String,Int)] = data.map{line =>  
    val t= line.split(',')  
    (t(0),t(2).toInt)  
}.reduceByKey(_ + _)  
// Filter the information about female netizens who spend more than 2 hours online, and export the results.  
val result = femaleData.filter(line => line._2 > 120)  
result.collect().map(x => x._1 + ',' + x._2).foreach(println)  
spark.stop()
```

### 2.17.3.1.4 Python Example Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Example Code

The following code segment is only an example. For details, see **collectFemaleInfo.py**.

```
def contains(str, substr):  
    if substr in str:  
        return True
```

```
return False

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: CollectFemaleInfo <file>"
        exit(-1)

    spark = SparkSession \
        .builder \
        .appName("CollectFemaleInfo") \
        .getOrCreate()

    """
    The following programs are used to implement the following functions:
    1. Read data. This code indicates the data path that the input parameter argv[1] specifies. - text
    2. Filter data about the time that female netizens spend online. - filter
    3. Aggregate the total time that each female netizen spends online. - map/map/reduceByKey
    4. Filter information about female netizens who spend more than 2 hours online. - filter
    """

    inputPath = sys.argv[1]
    result = spark.read.text(inputPath).rdd.map(lambda r: r[0])\
        .filter(lambda line: contains(line, "female")) \
        .map(lambda line: line.split(',')) \
        .map(lambda dataArr: (dataArr[0], int(dataArr[2]))) \
        .reduceByKey(lambda v1, v2: v1 + v2) \
        .filter(lambda tupleVal: tupleVal[1] > 120) \
        .collect()
    for (k, v) in result:
        print k + "," + str(v)

    # Stop SparkContext.
    spark.stop()
```

## 2.17.3.2 Spark SQL Project

### 2.17.3.2.1 Overview

#### Scenarios

Develop a Spark application to perform the following operations on logs about dwell durations of netizens for shopping online:

- Collect statistics on female netizens who dwell on online shopping for more than 2 hours at a weekend.
- The first column in the log file records names, the second column records gender, and the third column records the dwell duration in the unit of minute. Three attributes are separated by commas (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20
YuanJing,male,10
```

```
CaiXuyu,female,50  
FangBo,female,50  
GuoYijun,male,5  
CaiXuyu,female,50  
Liyuan,male,20  
CaiXuyu,female,50  
FangBo,female,50  
LiuYang,female,20  
YuanJing,male,10  
FangBo,female,50  
GuoYijun,male,50  
CaiXuyu,female,50  
FangBo,female,60
```

## Data Preparation

Save log files in the Hadoop distributed file system (HDFS).

1. Create text files **input\_data1.txt** and **input\_data2.txt** on a local computer, and copy the log contents of **log1.txt** to **input\_data1.txt** and **log2.txt** to **input\_data2.txt** respectively.
2. Create **/tmp/input** on HDFS client path, and run the following commands to upload **input\_data1.txt** and **input\_data2.txt** to **/tmp/input**:
  - a. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a hdfs dfs command provides the same function.) to create a directory.
  - b. Go to the **/tmp/input** directory on the HDFS client, on the Linux HDFS client, run the **hadoop fs -put input\_data1.txt /tmp/input** and **hadoop fs -put input\_data2.txt /tmp/input** commands to upload data files.

## Development Idea

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

The process includes:

- Create a table and import the log files into the table.
- Filter the data information of the time that female netizens spend online.
- Aggregate the total time that each female netizen spends online.
- Filter the information of female netizens who spend more than 2 hours online.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, **/opt/female/**) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code:

- Run the Scala and Java sample programs.
  - **bin/spark-submit --class**  
*com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --deploy-mode client /opt/female/SparkSqlScalaExample-1.0.jar <inputPath>*
  - *<inputPath>* indicates the input path in HDFS.
- Run the Python sample program.
  - **bin/spark-submit --master yarn --deploy-mode client** */opt/female/SparkSQLPythonExample/SparkSQLPythonExample.py <inputPath>*
  - *<inputPath>* indicates the input path in HDFS.

### 2.17.3.2.2 Java Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files (The class name and file name must be the same as those in the actual code. The following is only an example).

#### Sample Code

The following code segment is only an example. For details, see `com.huawei.bigdata.spark.examples.FemaleInfoCollection`.

```
public static void main(String[] args) throws Exception {
    SparkSession spark = SparkSession
        .builder()
        .appName("CollectFemaleInfo")
        .config("spark.some.config.option", "some-value")
        .getOrCreate();

    // Convert RDD to DataFrame through the implicit conversion.
    JavaRDD<FemaleInfo> femaleInfoJavaRDD = spark.read().textFile(args[0]).javaRDD().map(
        new Function<String, FemaleInfo>() {
            @Override
            public FemaleInfo call(String line) throws Exception {
                String[] parts = line.split(",");
                FemaleInfo femaleInfo = new FemaleInfo();
                femaleInfo.setName(parts[0]);
                femaleInfo.setGender(parts[1]);
                femaleInfo.setStayTime(Integer.parseInt(parts[2].trim()));
                return femaleInfo;
            }
        });
}

// Register table.
Dataset<ROW> schemaFemaleInfo = spark.createDataFrame(femaleInfoJavaRDD,FemaleInfo.class);
schemaFemaleInfo.registerTempTable("FemaleInfoTable");

// Run SQL query
Dataset<ROW> femaleTimeInfo = spark.sql("select * from " +
    "(select name,sum(stayTime) as totalStayTime from FemaleInfoTable " +
    "where gender = 'female' group by name )" +
    " tmp where totalStayTime >120");

// Collect the columns of a row in the result.
List<String> result = femaleTimeInfo.javaRDD().map(new Function<Row, String>() {
    public String call(Row row) {
        return row.getString(0) + "," + row.getLong(1);
    }
}).collect();
```

```
    System.out.println(result);
    spark.stop();
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.3.1/sql-programming-guide.html#running-sql-queries-programmatically>.

### 2.17.3.2.3 Scala Sample Code

#### Function

Collects the information of female netizens who spend more than 2 hours in online shopping on the weekend from the log files.

#### Sample Code

The following code segment is only an example. For details, see com.huawei.bigdata.spark.examples.FemaleInfoCollection.

```
object FemaleInfoCollection
{
    //Table structure, used for mapping the text data to df
    case class FemaleInfo(name: String, gender: String, stayTime: Int)
    def main(args: Array[String]) {
        //Configure the Spark application name.
        val spark = SparkSession
            .builder()
            .appName("FemaleInfo")
            .config("spark.some.config.option", "some-value")
            .getOrCreate()
        import spark.implicits._

        //Convert RDD to DataFrame through the implicit conversion, then register a table.
        spark.sparkContext.textFile(args(0)).map(_.split(","))
            .map(p => FemaleInfo(p(0), p(1), p(2).trim.toInt))
            .toDF.registerTempTable("FemaleInfoTable")
        //Use SQL statements to filter the data information of the time that female netizens spend online, and aggregate data of the same name.
        val femaleTimelInfo = spark.sql("select name,sum(stayTime) as stayTime from FemaleInfoTable where gender = 'female' group by name")
        //Filter the information of female netizens who spend more than 2 hours online and output the result.
        val c = femaleTimelInfo.filter("stayTime >= 120").collect().foreach(println)
        spark.stop()
    }
}
```

In the preceding code example, data processing logic is implemented by SQL statements. It can also be implemented by invoking the SparkSQL interface in Scala/Java/Python code. For details, see <http://spark.apache.org/docs/3.3.1/sql-programming-guide.html#running-sql-queries-programmatically>.

### 2.17.3.2.4 Python Sample Code

#### Function

Collect information about female netizens who have spent more than 2 hours in online shopping on the weekend.

## Sample Code

The following code segment is only an example. For details, see [SparkSQLPythonExample](#).

```
# -*- coding:utf-8 -*-

import sys
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

def contains(str1, substr1):
    if substr1 in str1:
        return True
    return False

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: SparkSQLPythonExample.py <file>"
        exit(-1)

    # Initialize the SparkSession and SQLContext.
    sc = SparkSession.builder.appName("CollectFemaleInfo").getOrCreate()
    sqlCtx = SQLContext(sc)

    #Convert RDD to DataFrame.
    inputPath = sys.argv[1]
    inputRDD = sc.read.text(inputPath).rdd.map(lambda r: r[0])\
        .map(lambda line: line.split(','))\
        .map(lambda dataArr: (dataArr[0], dataArr[1], int(dataArr[2])))\
        .collect()
    df = sqlCtx.createDataFrame(inputRDD)

    # Register a table.
    df.registerTempTable("FemaleInfoTable")

    # Run SQL query statements and display the result.
    FemaleTimeInfo = sqlCtx.sql("SELECT * FROM " +
        "(SELECT _1 AS Name,SUM(_3) AS totalStayTime FROM FemaleInfoTable " +
        "WHERE _2 = 'female' GROUP BY _1 )" +
        " WHERE totalStayTime >120").show()

    sc.stop()
```

### 2.17.3.3 Accessing the Spark SQL Through JDBC

#### 2.17.3.3.1 Overview

##### Scenarios

Users customize JDBCServer clients and use JDBC connections to create, load data to, query, and delete data tables.

 **NOTE**

Spark allows you to add extension identifiers to JDBC connection strings. These extension identifiers are printed in JDBCServer audit logs to distinguish SQL sources. To add extension identifiers, add the following content to the end of the connection strings in the **`$SPARK_HOME/bin/spark-beeline`** script in the client directory:

**auditAddition=xxx**

```
cd $HOME/spark  
CURRENT_PATH=$(cd $HOME;pwd)  
echo "CURRENT PATH is $CURRENT_PATH"  
echo "It's running the $1 spark baseline, it calls $CURRENT_PATHbaseline  
and try to connect to the 50070 server successfully"  
  
CURRENT_PATH/baseline $1 >> $1.log  
tail -f $1.log
```

*xxx* is the custom identifier. The identifier can contain a maximum of 256 bytes and only letters, digits, underscores (\_), commas (,), and colons (:) are allowed.

After the SQL statement is executed, the configured extension identifier is printed in the HiveServer audit log `/var/log/Bigdata/audit/spark/jdbcserver/jdbcserver-audit.log`. The following is an example.

## Data Preparation

**Step 1** Ensure that the JDBCServer service has been started in multi-active instance HA mode and at least one instance provides connections for client. Create the **/home/data** file on every available instance nodes of the JDBCServer. The file content is as follows:

Miranda,32  
Karlie,23  
Candice,27

**Step 2** Ensure that the user whose starts the JDBCServer has the read and write permission on the file.

**Step 3** Ensure that the `hive-site.xml` file exists in `classpath`, and set parameters required for the client connection. For details about parameters required for the JDBCServer, see [JDBCServer Interface](#).

-----End

## Development Idea

1. Create a child table in the default database.
  2. Add data in **/home/data** to the child table.
  3. Query data in the child table.
  4. Delete the child table.

# Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
  - Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

## Running Tasks

Go to the Spark client directory and run the **java -cp** command to run the code  
(The class name and file name must be the same as those in the actual code. The following is only an example).

- Run the Java sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/  
conf:/opt/female/SparkThriftServerJavaExample-1.0.jar  
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/  
conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

- Run the Scala sample code:

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/  
conf:/opt/female/SparkThriftServerExample-1.0.jar  
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/  
conf/hive-site.xml $SPARK_HOME/conf/spark-defaults.conf
```

### NOTE

After the SSL feature of ZooKeeper is enabled for the cluster (check the **ssl.enabled** parameter of the ZooKeeper service), add the **-Dzookeeper.client.secure=true -Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty** parameter to the command:

```
java -Dzookeeper.client.secure=true -  
Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -cp  
$SPARK_HOME/jars/*:$SPARK_HOME/jars/hive/*:$SPARK_HOME/conf:/opt/female/  
SparkThriftServerJavaExample-1.0.jar  
com.huawei.bigdata.spark.examples.ThriftServerQueriesTest $SPARK_HOME/conf/hive-  
site.xml $SPARK_HOME/conf/spark-defaults.conf
```

### 2.17.3.3.2 Java Sample Code

#### Function Description

The JDBC API of the user-defined client is used to submit a data analysis task and return the results.

#### Example Codes

- Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). The following is an example:

```
ArrayList<String> sqlList = new ArrayList<String>();  
sqlList.add("CREATE TABLE CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS  
TERMINATED BY ','");  
sqlList.add("LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD");  
sqlList.add("SELECT * FROM child");  
sqlList.add("DROP TABLE child");  
executeSql(url, sqlList);
```

### NOTE

The **data** file in the sample project must be placed in the **home** directory of the host where the JDBCServer is located.

- Step 2** Concatenate the JDBC URL.

```
Configuration config = new Configuration();  
config.addResource(new Path(args[0]));
```

```
String zkUrl = config.get("spark.deploy.zookeeper.url");
String auditAddition = "auditAddition=sparktest1111test;";

String zkNamespace = null;
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace");
if (zkNamespace != null) {
    //Remove redundant characters from configuration items
    zkNamespace = zkNamespace.substring(1);
}

StringBuilder sb = new StringBuilder("jdbc:hive2://"
    + zkUrl
    + "/serviceDiscoveryMode=zooKeeper;"
    + "zooKeeperNamespace="
    + zkNamespace + ","
    + auditAddition
);
String url = sb.toString();
```

**Step 3** Load the Hive JDBC driver.

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

**Step 4** Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The **zk.quorum** in the connection string can be replaced by **spark.deploy.zookeeper.url** in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server. The configuration method is as follows:

Before using the **DriverManager.getConnection** method to obtain the JDBC connection, add the **DriverManager.setLoginTimeout(n)** method to configure a timeout interval. **n** indicates the timeout interval for waiting for the return result from the server. The unit is second, the type is **Int**, and the default value is **0** (indicating never timing out).

```
static void executeSql(String url, ArrayList<String> sqls) throws ClassNotFoundException, SQLException {
    try {
        Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
    } catch (Exception e) {
        e.printStackTrace();
    }
    Connection connection = null;
    PreparedStatement statement = null;

    try {
        connection = DriverManager.getConnection(url);
        for (int i = 0 ; i < sqls.size(); i++) {
            String sql = sqls.get(i);
            System.out.println("---- Begin executing sql: " + sql + " ----");
            statement = connection.prepareStatement(sql);
            ResultSet result = statement.executeQuery();
            ResultSetMetaData metaData = result.getMetaData();
            Integer colNum = metaData.getColumnCount();
            for (int j = 1; j <= colNum; j++) {
                System.out.println(result.getColumnName(j) + "\t");
            }
            System.out.println();

            while (result.next()) {
                for (int j = 1; j <= colNum; j++){
                    System.out.println(result.getString(j) + "\t");
                }
                System.out.println();
            }
        }
    } finally {
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
        System.out.println("---- Done executing sql: " + sql + " ----");
    }

} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (null != statement) {
        statement.close();
    }
    if (null != connection) {
        connection.close();
    }
}
```

----End

### 2.17.3.3.3 Scala Sample Code

#### Function Description

The JDBC interface of the user-defined client is used to submit the data analysis task and return the results.

#### Example Codes

- Step 1** Define an SQL statement. The SQL statement must be a single statement that cannot contain the semicolon (;). The following is an example:

```
val sqlList = new ArrayBuffer[String]
sqlList += "CREATE TABLE CHILD (NAME STRING, AGE INT) " +
"ROW FORMAT DELIMITED FIELDS TERMINATED BY ','"
sqlList += "LOAD DATA LOCAL INPATH '/home/data' INTO TABLE CHILD"
sqlList += "SELECT * FROM child"
sqlList += "DROP TABLE child"
```



The data file in the sample project must be placed into the Home directory of the host where the JDBCServer is located.

- Step 2** Concatenate the JDBC URL.

```
val config: Configuration = new Configuration()
config.addResource(new Path(args(0)))
val zkUrl = config.get("spark.deploy.zookeeper.url")
val auditAddtion = "auditAddtion=sparktest1111test;"

var zkNamespace: String = null
zkNamespace = fileInfo.getProperty("spark.thriftserver.zookeeper.namespace")
//Remove redundant characters from configuration items
if (zkNamespace != null) zkNamespace = zkNamespace.substring(1)

val sb = new StringBuilder("jdbc:hive2://"
+ zkUrl
+ "/serviceDiscoveryMode=zooKeeper;"
+ "zooKeeperNamespace="
+ zkNamespace + ";"
+ auditAddtion
);
val url = sb.toString()
```

- Step 3** Load the Hive JDBC driver. Obtain the JDBC connection, execute the HQL, export the obtained column name and results to the console, and close the JDBC connection.

The **zk.quorum** in the connection string can be replaced by **spark.deploy.zookeeper.url** in the configuration file.

In network congestion, configure the timeout of the connection between the client and JDBCServer to avoid the suspending of the client due to timeless wait of the return from the server. The configuration method is as follows:

Before using the **DriverManager.getConnection** method to obtain the JDBC connection, add the **DriverManager.setLoginTimeout(n)** method to configure a timeout interval. **n** indicates the timeout interval for waiting for the return result from the server. The unit is second, the type is **Int**, and the default value is **0** (indicating never timing out).

```
def executeSql(url: String, sqls: Array[String]): Unit = {
    //Load the Hive JDBC driver.
    Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance()

    var connection: Connection = null
    var statement: PreparedStatement = null
    try {
        connection = DriverManager.getConnection(url)
        for (sql <- sqls) {
            println(s"---- Begin executing sql: $sql ----")
            statement = connection.prepareStatement(sql)

            val result = statement.executeQuery()

            val resultMetaData = result.getMetaData
            val colNum = resultMetaData.getColumnCount
            for (i <- 1 to colNum) {
                print(resultMetaData.getColumnName(i) + "\t")
            }
            println()

            while (result.next()) {
                for (i <- 1 to colNum) {
                    print(result.getString(i) + "\t")
                }
                println()
            }
            println(s"---- Done executing sql: $sql ----")
        }
    } finally {
        if (null != statement) {
            statement.close()
        }

        if (null != connection) {
            connection.close()
        }
    }
}
```

----End

## 2.17.3.4 Spark on HBase

### 2.17.3.4.1 Performing Operation on Data in Avro Format

#### Scenario

Users can use HBase as data sources in Spark applications. In this example, data is stored in HBase in Avro format. Data is read from the HBase, and the read data is filtered.

## Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'ExampleAvrotable','rowkey','cf1'  
create 'ExampleAvrotableInsert','rowkey','cf1'
```

## Development Guideline

1. Create an RDD.
2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and performs simple operations on the data.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name of the case code is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. Run the following commands in the **\$SPARK\_HOME** directory.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource  
SparkOnHbaseJavaExample-1.0.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --conf  
spark.yarn.user.classpath.first=true --jars  
SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar  
AvroSource.py
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.AvroSource SparkOnHbaseJavaExample-1.0.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar AvroSource.py
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbaseJavaExample.

```
public static void main(JavaSparkContext jsc) throws IOException {
    SQLContext sqlContext = new SQLContext(jsc);
    Configuration hbaseconf = new HBaseConfiguration().create();
    JavaHBaseContext hBaseContext = new JavaHBaseContext(jsc, hbaseconf);
    List list = new ArrayList<AvroHBaseRecord>();
    for(int i=0; i<=255 ; ++i){
        list.add(AvroHBaseRecord.apply(i));
    }
    try{
        Map<String, String> map = new HashMap<String, String>();
        map.put(HBaseTableCatalog.tableCatalog(), catalog);
        map.put(HBaseTableCatalog.newTable(), "5");
        sqlContext.createDataFrame(list,
        AvroHBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
        Dataset<Row> ds = withCatalog(sqlContext,catalog);
        ds.show();
        ds.printSchema();
        ds.registerTempTable("ExampleAvrotable");
        Dataset<Row> c= sqlContext.sql("select count(1) from ExampleAvrotable");
        c.show();
        Dataset<Row> filtered = ds.select("col0", "col1.favorite_array").where("col0 = 'name1'");
        filtered.show();
        java.util.List<Row> collected = filtered.collectAsList();
        if (collected.get(0).get(1).toString().equals("number1")) {
            throw new UserCustomizedSampleException("value invalid", new Throwable());
        }
        if (collected.get(0).get(1).toString().equals("number2")) {
            throw new UserCustomizedSampleException("value invalid", new Throwable());
        }
        Map avroCatalogInsertMap = new HashMap<String, String>();
        avroCatalogInsertMap.put("avroSchema" , AvroHBaseRecord.schemaString);
        avroCatalogInsertMap.put(HBaseTableCatalog.tableCatalog(), avroCatalogInsert);
        ds.write().options(avroCatalogInsertMap).format("org.apache.hadoop.hbase.spark").save();
        Dataset<Row> newDS = withCatalog(sqlContext,avroCatalogInsert);
        newDS.show();
        newDS.printSchema();
        if (newDS.count() != 256) {
            throw new UserCustomizedSampleException("value invalid", new Throwable());
        }
        ds.filter("col1.name = 'name5' || col1.name <= 'name5'").select("col0","col1.favorite_color",
        "col1.favorite_number").show();
    } finally{
        jsc.stop();
    }
}
```

```
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("AvroSourceExample")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)
    val hbaseConf = HBaseConfiguration.create()
    val hBaseContext = new HBaseContext(sc, hbaseConf)
    import sqlContext.implicits_
    def withCatalog(cat: String): DataFrame = {
        sqlContext
            .read
            .options(Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
avroCatalog))
            .format("org.apache.hadoop.hbase.spark")
            .load()
    }
    val data = (0 to 255).map { i =>
        AvroHBaseRecord(i)
    }
    try {
        sc.parallelize(data).toDF.write.options(
            Map(HBaseTableCatalog.tableCatalog -> catalog, HBaseTableCatalog.newTable -> "5"))
            .format("org.apache.hadoop.hbase.spark")
            .save()

        val df = withCatalog(catalog)
        df.show()
        df.printSchema()
        df.registerTempTable("ExampleAvrotable")
        val c = sqlContext.sql("select count(1) from ExampleAvrotable")
        c.show()

        val filtered = df.select($"col0", $"col1.favorite_array").where($"col0" === "name001")
        filtered.show()
        val collected = filtered.collect()
        if (collected(0).getSeq[String](1)(0) != "number1") {
            throw new UserCustomizedSampleException("value invalid")
        }
        if (collected(0).getSeq[String](1)(1) != "number2") {
            throw new UserCustomizedSampleException("value invalid")
        }

        df.write.options(
            Map("avroSchema" -> AvroHBaseRecord.schemaString, HBaseTableCatalog.tableCatalog ->
avroCatalogInsert,
                HBaseTableCatalog.newTable -> "5"))
            .format("org.apache.hadoop.hbase.spark")
            .save()
        val newDF = withCatalog(avroCatalogInsert)
        newDF.show()
        newDF.printSchema()
        if (newDF.count() != 256) {
            throw new UserCustomizedSampleException("value invalid")
        }
        df.filter($"col1.name" === "name005" || $"col1.name" <= "name005")
            .select("col0", "col1.favorite_color", "col1.favorite_number")
            .show()
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the AvroSource file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""
[Note]
Pyspark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("AvroSourceExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.AvroSource')
# Create a class instance, invoke the method, and transfer the sc._jsc parameter.
spark._jvm.AvroSource().execute(spark._jsc)
# Stop SparkSession.
spark.stop()
```

### 2.17.3.4.2 Performing Operations on the HBase Data Source

#### Scenario

Users can use HBase as data sources in Spark applications, write DataFrame to HBase, read data from HBase, and filter the read data.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'HBaseSourceExampleTable','rowkey','cf1','cf2','cf3','cf4','cf5','cf6','cf7', 'cf8'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase to treat it as the data source and write the generated RDD into HBase tables.
3. Read data from HBase tables and performs simple operations on the data.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name of the case code is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. Run the following commands in the **\$SPARK\_HOME** directory.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource SparkOnHbaseJavaExample-1.0.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars /opt/female/protobuf-java-2.5.0.jar --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.datasources.HBaseSource SparkOnHbaseJavaExample-1.0.jar
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --conf spark.yarn.user.classpath.first=true --jars SparkOnHbaseJavaExample-1.0.jar,/opt/female/protobuf-java-2.5.0.jar HBaseSource.py
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseSource** file in **SparkOnHbaseJavaExample**.

```
public static void main(String args[]) throws IOException{
    SparkConf sparkConf = new SparkConf().setAppName("HBaseSourceExample");
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    SQLContext sqlContext = new SQLContext(jsc);
```

```
Configuration conf = HBaseConfiguration.create();
JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc,conf);
try{
    List<HBaseRecord> list = new ArrayList<HBaseRecord>();
    for(int i=0 ; i<256; i++){
        list.add(new HBaseRecord(i));
    }
    Map map = new HashMap<String, String>();
    map.put(HBaseTableCatalog.tableCatalog(), cat);
    map.put(HBaseTableCatalog.newTable(), "5");
    System.out.println("Before insert data into hbase table");
    sqlContext.createDataFrame(list,
        HBaseRecord.class).write().options(map).format("org.apache.hadoop.hbase.spark").save();
    Dataset<Row> ds = withCatalog(sqlContext, cat);
    System.out.println("After insert data into hbase table");
    ds.printSchema();
    ds.show();
    ds.filter("key <= 'row5'").select("key","col1").show();
    ds.registerTempTable("table1");
    Dataset<Row> tempDS = sqlContext.sql("select count(col1) from table1 where key < 'row5'");
    tempDS.show();
} finally {
    jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseSource` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseSourceExample")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)
    val conf = HBaseConfiguration.create()
    val hbaseContext = new HBaseContext(sc,conf)
    import sqlContext.implicits._

    def withCatalog(cat: String): DataFrame = {
        sqlContext
            .read
            .options(Map(HBaseTableCatalog.tableCatalog->cat))
            .format("org.apache.hadoop.hbase.spark")
            .load()
    }

    val data = (0 to 255).map { i =>
        HBaseRecord(i)
    }
    try{
        sc.parallelize(data).toDF.write.options(
            Map(HBaseTableCatalog.tableCatalog -> cat, HBaseTableCatalog.newTable -> "5"))
            .format("org.apache.hadoop.hbase.spark")
            .save()
        val df = withCatalog(cat)
        df.show()
        df.filter($"col0" <= "row005")
            .select($"col0", $"col1").show()
        df.registerTempTable("table1")
        val c = sqlContext.sql("select count(col1) from table1 where col0 < 'row050'")
        c.show()
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseSource file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement the required operations.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("HBaseSourceExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.datasources.HBaseSource')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.HBaseSource().execute(spark._jsc)
# Stop the SparkSession instance.
spark.stop()
```

### 2.17.3.4.3 Using the BulkPut Interface

#### Scenario

Users can use the HBaseContext method to use HBase in Spark applications and write the constructed RDD into HBase.

#### Data Planning

On the client, run the **hbase shell** command to go to the HBase command line and run the following commands to create HBase tables to be used in the sample code:

```
create 'bulktable','cf1'
```

#### Development Guideline

1. Create an RDD.
2. Perform operations on HBase in HBaseContext mode and write the generated RDD into the HBase table.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hBaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa  
mple SparkOnHbaseJavaExample-1.0.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample-1.0.jar HBaseBulkPutExample.py bulktable  
cf1
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExa  
mple SparkOnHbaseJavaExample-1.0.jar bulktable cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars  
SparkOnHbaseJavaExample-1.0.jar HBaseBulkPutExample.py bulktable  
cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **JavaHBaseBulkPutExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws Exception{  
    if (args.length < 2) {  
        System.out.println("JavaHBaseBulkPutExample " +  
                           "{tableName} {columnFamily}");  
        return;  
    }  
    String tableName = args[0];  
    String columnFamily = args[1];  
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkPutExample " + tableName);
```

```
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
try {
    List<String> list = new ArrayList<String>(5);
    list.add("1," + columnFamily + ",1,1");
    list.add("2," + columnFamily + ",1,2");
    list.add("3," + columnFamily + ",1,3");
    list.add("4," + columnFamily + ",1,4");
    list.add("5," + columnFamily + ",1,5");
    list.add("6," + columnFamily + ",1,6");
    list.add("7," + columnFamily + ",1,7");
    list.add("8," + columnFamily + ",1,8");
    list.add("9," + columnFamily + ",1,9");
    list.add("10," + columnFamily + ",1,10");
    JavaRDD<String> rdd = jsc.parallelize(list);
    Configuration conf = HBaseConfiguration.create();
    JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
    hbaseContext.bulkPut(rdd,
        TableName.valueOf(tableName),
        new PutFunction());
    System.out.println("Bulk put into Hbase successfully!");
} finally {
    jsc.stop();
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkPutExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
  if (args.length < 2) {
    System.out.println("HBaseBulkPutTimestampExample {tableName} {columnFamily} are missing an argument")
    return
  }
  val tableName = args(0)
  val columnFamily = args(1)
  val sparkConf = new SparkConf().setAppName("HBaseBulkPutTimestampExample " +
    tableName + " " + columnFamily)
  val sc = new SparkContext(sparkConf)
  try {
    val rdd = sc.parallelize(Array(
      (Bytes.toBytes("1"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1")))),
      (Bytes.toBytes("2"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2")))),
      (Bytes.toBytes("3"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3")))),
      (Bytes.toBytes("4"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4")))),
      (Bytes.toBytes("5"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5")))),
      (Bytes.toBytes("6"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("6")))),
      (Bytes.toBytes("7"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("7")))),
      (Bytes.toBytes("8"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("8")))),
      (Bytes.toBytes("9"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("9")))),
      (Bytes.toBytes("10"),
       Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("10")))))
    val conf = HBaseConfiguration.create()
    val timeStamp = System.currentTimeMillis()
    val hbaseContext = new HBaseContext(sc, conf)
    hbaseContext.bulkPut([(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])](rdd,
      TableName.valueOf(tableName),
      (putRecord) => {
```

```
    val put = new Put(putRecord._1)
    putRecord._2.foreach((putValue) => put.addColumn(putValue._1, putValue._2,
      timeStamp, putValue._3))
    put
  }
} finally {
  sc.stop()
}
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkPutExample file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseBulkPutExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkPutExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkPutExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 2.17.3.4.4 Using the BulkGet Interface

#### Scenario

Users can use the HBaseContext method to use HBase in Spark applications, construct the rowkey of the data to be obtained into RDDs, and obtain the data corresponding to the rowkey in the HBase tables through the BulkGet interface of HBaseContext.

#### Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#)

#### Development Guideline

1. Create RDDs containing the rowkey to be obtained.
2. Perform operations on HBase in HBaseContext mode and obtain data corresponding to rowkey in HBase tables through the BulkGet interface of HBaseContext.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).

- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

 NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

**bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample SparkOnHbaseJavaExample-1.0.jar bulktable**

Python version. (The file name must be the same as the actual one. The following is only an example.)

**bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample-1.0.jar HBaseBulkGetExample.py bulktable**

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

**bin/spark-submit --master yarn --deploy-mode cluster --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample SparkOnHbaseJavaExample-1.0.jar bulktable**

Python version. (The file name must be the same as the actual one. The following is only an example.)

**bin/spark-submit --master yarn --deploy-mode cluster --jars SparkOnHbaseJavaExample-1.0.jar HBaseBulkGetExample.py bulktable**

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseBulkGetExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws IOException{
    if (args.length < 1) {
        System.out.println("JavaHBaseBulkGetExample {tableName}");
        return;
    }
    String tableName = args[0];
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try {
```

```
List<byte[]> list = new ArrayList<byte[]>(5);
list.add(Bytes.toBytes("1"));
list.add(Bytes.toBytes("2"));
list.add(Bytes.toBytes("3"));
list.add(Bytes.toBytes("4"));
list.add(Bytes.toBytes("5"));
JavaRDD<byte[]> rdd = jsc.parallelize(list);
Configuration conf = HBaseConfiguration.create();
JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
List resultList = hbaseContext.bulkGet(TableName.valueOf(tableName), 2, rdd, new GetFunction(),
    new ResultFunction()).collect();
for(int i =0 ;i<resultList.size();i++){
    System.out.println(resultList.get(i));
}
} finally {
    jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkGetExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
  if (args.length < 1) {
    println("HBaseBulkGetExample {tableName} missing an argument")
    return
  }
  val tableName = args(0)
  val sparkConf = new SparkConf().setAppName("HBaseBulkGetExample " + tableName)
  val sc = new SparkContext(sparkConf)
  try {
    //[(Array[Byte])]
    val rdd = sc.parallelize(Array(
      Bytes.toBytes("1"),
      Bytes.toBytes("2"),
      Bytes.toBytes("3"),
      Bytes.toBytes("4"),
      Bytes.toBytes("5"),
      Bytes.toBytes("6"),
      Bytes.toBytes("7")))
    val conf = HBaseConfiguration.create()
    val hbaseContext = new HBaseContext(sc, conf)
    val getRdd = hbaseContext.bulkGet[Array[Byte], String](
      TableName.valueOf(tableName),
      2,
      rdd,
      record => {
        System.out.println("making Get")
        new Get(record)
      },
      (result: Result) => {
        val it = result.listCells().iterator()
        val b = new StringBuilder
        b.append(Bytes.toString(result.getRow) + ":")
        while (it.hasNext) {
          val cell = it.next()
          val q = Bytes.toString(CellUtil.cloneQualifier(cell))
          if (q.equals("counter")) {
            b.append("(" + q + "," + Bytes.toLong(CellUtil.cloneValue(cell)) + ")")
          } else {
            b.append("(" + q + "," + Bytes.toString(CellUtil.cloneValue(cell)) + ")")
          }
        }
        b.toString()
      })
    getRdd.collect().foreach(v => println(v))
  } finally {
```

```
        sc.stop()  
    }
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkGetExample file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-  
"""  
[Note]  
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to  
implement required operations.  
"""  
from py4j.java_gateway import java_import  
from pyspark.sql import SparkSession  
# Create a SparkSession instance.  
spark = SparkSession\  
    .builder\  
    .appName("JavaHBaseBulkGetExample")\  
    .getOrCreate()  
# Import required class to sc._jvm.  
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkGetExample')  
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.  
spark._jvm.JavaHBaseBulkGetExample().execute(spark._jsc, sys.argv)  
# Stop the SparkSession instance.  
spark.stop()
```

### 2.17.3.4.5 Using the BulkDelete Interface

#### Scenario

Users can use the HBaseContext method to use HBase in Spark applications, construct rowkey of the data to be deleted into RDDs, and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of HBaseContext.

#### Data Planning

Perform operations based on the HBase tables and data in the tables that are created in [Using the BulkPut Interface](#).

#### Development Guideline

1. Create RDDs containing the rowkey to be deleted.
2. Perform operations on the HBase in HBaseContext mode and delete the data corresponding to the rowkey in HBase tables through the BulkDelete interface of HBaseContext.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hBaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE  
xample SparkOnHbaseJavaExample-1.0.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample-1.0.jar HBaseBulkDeleteExample.py bulktable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteE  
xample SparkOnHbaseJavaExample-1.0.jar bulktable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars  
SparkOnHbaseJavaExample-1.0.jar HBaseBulkDeleteExample.py bulktable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the **HBaseBulkDeleteExample** file in **SparkOnHbaseJavaExample**.

```
public static void main(String[] args) throws IOException {  
    if (args.length < 1) {  
        System.out.println("JavaHBaseBulkDeleteExample {tableName}");  
        return;  
    }  
    String tableName = args[0];  
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkDeleteExample " + tableName);  
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
    try {  
        List<byte[]> list = new ArrayList<byte[]>(5);  
        list.add(Bytes.toBytes("1"));  
        list.add(Bytes.toBytes("2"));  
    }  
}
```

```
list.add(Bytes.toBytes("3"));
list.add(Bytes.toBytes("4"));
list.add(Bytes.toBytes("5"));
JavaRDD<byte[]> rdd = jsc.parallelize(list);
Configuration conf = HBaseConfiguration.create();
JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
hbaseContext.bulkDelete(rdd,
    TableName.valueOf(tableName), new DeleteFunction(), 4);
System.out.println("Bulk Delete successfully!");
} finally {
    jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkDeleteExample` file in `SparkOnHbaseScalaExample`.

```
def main(args: Array[String]) {
    if (args.length < 1) {
        println("HBaseBulkDeleteExample {tableName} missing an argument")
        return
    }
    val tableName = args(0)
    val sparkConf = new SparkConf().setAppName("HBaseBulkDeleteExample " + tableName)
    val sc = new SparkContext(sparkConf)
    try {
        // [Array[Byte]]
        val rdd = sc.parallelize(Array(
            Bytes.toBytes("1"),
            Bytes.toBytes("2"),
            Bytes.toBytes("3"),
            Bytes.toBytes("4"),
            Bytes.toBytes("5")
        ))
        val conf = HBaseConfiguration.create()
        val hbaseContext = new HBaseContext(sc, conf)
        hbaseContext.bulkDelete[Array[Byte]](rdd,
            TableName.valueOf(tableName),
            putRecord => new Delete(putRecord),
            4)
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseBulkDeleteExample` file in `SparkOnHbasePythonExample`.

```
def main(args: Array[String]) {
# -*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseBulkDeleteExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
```

```
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkDeleteExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseBulkDeleteExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 2.17.3.4.6 Using the BulkLoad Interface

#### Scenario

Users can use HBaseContext to perform operations on HBase in Spark applications, construct rowkey of the data to be inserted into RDDs, and write RDDs to HFiles through the BulkLoad interface of HBaseContext. The following command is used to import the generated HFiles to the HBase table and will not be described in this section.

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles {hfilePath}
{tableName}
```

#### Data Planning

1. Run the **hbase shell** command on the client to go to the HBase command line.
2. Run the following command to create HBase tables:  
`create 'bulkload-table-test','f1','f2'`

#### Development Guideline

1. Construct the data to be imported into RDDs.
2. Perform operations on HBase in HBaseContext mode and write RDDs into HFiles through the BulkLoad interface of HBaseContext.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

##### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

#### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExa  
mple SparkOnHbaseJavaExample-1.0.jar /tmp/hfile bulkload-table-test
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars  
SparkOnHbaseJavaExample-1.0.jar HBaseBulkLoadExample.py /tmp/hfile  
bulkload-table-test
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class  
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseBulkLoadExa  
mple SparkOnHbaseJavaExample-1.0.jar /tmp/hfile bulkload-table-test
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars  
SparkOnHbaseJavaExample-1.0.jar HBaseBulkLoadExample.py /tmp/hfile  
bulkload-table-test
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseBulkLoadExample file in SparkOnHbaseJavaExample.

```
public static void main(String[] args) throws IOException{  
    if (args.length < 2) {  
        System.out.println("JavaHBaseBulkLoadExample {outputPath} {tableName}");  
        return;  
    }  
    String outputPath = args[0];  
    String tableName = args[1];  
    String columnFamily1 = "f1";  
    String columnFamily2 = "f2";  
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName);  
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
    try {  
        List<String> list= new ArrayList<String>();  
        // row1  
        list.add("1," + columnFamily1 + ",b,1");  
        // row3  
        list.add("3," + columnFamily1 + ",a,2");  
        list.add("3," + columnFamily1 + ",b,1");  
        list.add("3," + columnFamily2 + ",a,1");  
        /* row2 */  
        list.add("2," + columnFamily2 + ",a,3");  
        list.add("2," + columnFamily2 + ",b,3");  
        JavaRDD<String> rdd = jsc.parallelize(list);  
        Configuration conf = HBaseConfiguration.create();  
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);  
        hbaseContext.bulkLoad(rdd, TableName.valueOf(tableName),new BulkLoadFunction(), outputPath,  
        new HashMap<byte[], FamilyHFileWriteOptions>(), false, HConstants.DEFAULT_MAX_FILE_SIZE);  
    } finally {  
        jsc.stop();  
    }  
}
```

```
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkLoadExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    if(args.length < 2) {
        println("HBaseBulkLoadExample {outputPath} {tableName}")
        return
    }
    LoginUtil.loginWithUserKeytab()
    val Array(outputPath, tableName) = args
    val columnFamily1 = "f1"
    val columnFamily2 = "f2"
    val sparkConf = new SparkConf().setAppName("JavaHBaseBulkLoadExample " + tableName)
    val sc = new SparkContext(sparkConf)
    try {
        val arr = Array("1," + columnFamily1 + ",b,1",
                      "2," + columnFamily1 + ",a,2",
                      "3," + columnFamily1 + ",b,1",
                      "3," + columnFamily2 + ",a,1",
                      "4," + columnFamily2 + ",a,3",
                      "5," + columnFamily2 + ",b,3")

        val rdd = sc.parallelize(arr)
        val config = HBaseConfiguration.create
        val hbaseContext = new HBaseContext(sc, config)
        hbaseContext.bulkLoad[String](rdd,
                                      TableName.valueOf(tableName),
                                      (putRecord) => {
            if(putRecord.length > 0) {
                val strArray = putRecord.split(",")
                val kfq = new KeyFamilyQualifier(Bytes.toBytes(strArray(0)), Bytes.toBytes(strArray(1)),
                                                Bytes.toBytes(strArray(2)))
                val ite = (kfq, Bytes.toBytes(strArray(3)))
                val itea = List(ite).iterator
                itea
            } else {
                null
            },
            outputPath)
        } finally {
            sc.stop()
        }
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseBulkLoadPythonExample file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
```

```
.builder\  
.appName("JavaHBaseBulkLoadExample")\  
.getOrCreate()  
# Import required class to sc._jvm.  
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.HBaseBulkLoadPythonExample')  
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.  
spark._jvm.HBaseBulkLoadPythonExample().hbaseBulkLoad(spark._jsc, sys.argv[1], sys.argv[2])  
# Stop the SparkSession instance.  
spark.stop()
```

### 2.17.3.4.7 Using the foreachPartition Interface

#### Scenario

Users can use HBaseContext to perform operations on HBase in the Spark application, construct rowkey of the data to be inserted into RDDs, and write RDDs to HBase tables through the mapPartition interface of HBaseContext.

#### Data Planning

1. Run the **hbase shell** command on the client to go to the HBase command line.
2. Run the following command to create an HBase table:  
**create 'table2','cf1'**

#### Development Guideline

1. Construct the data to be imported into RDDs.
2. Perform operations on HBase in HBaseContext mode and concurrently write data to HBase through the foreachPatition interface of HBaseContext.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

##### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

#### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- **yarn-client mode:**

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachParti
tionExample SparkOnHbaseJavaExample-1.0.jar table2 cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample-1.0.jar HBaseForEachPartitionExample.py
table2 cf1
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachParti
tionExample SparkOnHbaseJavaExample-1.0.jar table2 cf1
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars
SparkOnHbaseJavaExample-1.0.jar HBaseForEachPartitionExample.py
table2 cf1
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseForEachPartitionExample file in SparkOnHbaseJavaExample.

```
public static void main(String[] args) throws IOException {
    if (args.length < 1) {
        System.out.println("JavaHBaseForEachPartitionExample {tableName} {columnFamily}");
        return;
    }
    final String tableName = args[0];
    final String columnFamily = args[1];
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseBulkGetExample " + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try {
        List<byte[]> list = new ArrayList<byte[]>(5);
        list.add(Bytes.toBytes("1"));
        list.add(Bytes.toBytes("2"));
        list.add(Bytes.toBytes("3"));
        list.add(Bytes.toBytes("4"));
        list.add(Bytes.toBytes("5"));
        JavaRDD<byte[]> rdd = jsc.parallelize(list);
        Configuration conf = HBaseConfiguration.create();
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
        hbaseContext.foreachPartition(rdd,
            new VoidFunction<Tuple2<Iterator<byte[]>, Connection>>() {
                public void call(Tuple2<Iterator<byte[]>, Connection> t)
                    throws Exception {
                    Connection con = t._2();
                    Iterator<byte[]> it = t._1();
                    BufferedMutator buf = con.getBufferedMutator(TableName.valueOf(tableName));
                    while (it.hasNext()) {
                        byte[] b = it.next();
                        Put put = new Put(b);
                        put.add(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"), b);
                        buf.mutate(put);
                    }
                }
            });
    }
}
```

```
        mutator.flush();
        mutator.close();

    }
}
} finally {
    jsc.stop();
}
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseForEachPartitionExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    if (args.length < 2) {
        println("HBaseForEachPartitionExample {tableName} {columnFamily} are missing arguments")
        return
    }
    val tableName = args(0)
    val columnFamily = args(1)
    val sparkConf = new SparkConf().setAppName("HBaseForEachPartitionExample " +
        tableName + " " + columnFamily)
    val sc = new SparkContext(sparkConf)
    try {
        //[(Array[Byte], Array[(Array[Byte], Array[Byte], Array[Byte])])]
        val rdd = sc.parallelize(Array(
            (Bytes.toBytes("1"),
                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("1"))),
                (Bytes.toBytes("2"),
                    Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("2"))),
                    (Bytes.toBytes("3"),
                        Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("3"))),
                        (Bytes.toBytes("4"),
                            Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("4"))),
                            (Bytes.toBytes("5"),
                                Array((Bytes.toBytes(columnFamily), Bytes.toBytes("1"), Bytes.toBytes("5")))))
                    )))
            )))
        val conf = HBaseConfiguration.create()
        val hbaseContext = new HBaseContext(sc, conf)
        rdd.hbaseForEachPartition(hbaseContext,
            (it, connection) => {
                val m = connection.getBufferedMutator(Table.Name.valueOf(tableName))
                it.foreach(r => {
                    val put = new Put(r._1)
                    r._2.foreach((putValue) =>
                        put.addColumn(putValue._1, putValue._2, putValue._3))
                    m.mutate(put)
                })
                m.flush()
                m.close()
            })
    } finally {
        sc.stop()
    }
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseForEachPartitionExample file in SparkOnHbasePythonExample.

```
# -*- coding:utf-8 -*-
"""
"""

[Note]
Pyspark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.
```

```
"""
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseForEachPartitionExample")\
    .getOrCreate()
# Import the required class to sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseForEachPartitionExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseForEachPartitionExample().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

#### 2.17.3.4.8 Distributedly Scanning HBase Tables

##### Scenario

Users can use HBaseContext to perform operations on HBase in Spark applications and use HBase RDDs to scan HBase tables based on specific rules.

##### Data Planning

Use HBase tables created in [Performing Operation on Data in Avro Format](#)

##### Development Guideline

1. Set the scanning rule, for example: setCaching.
2. Use specific rules to scan the HBase table.

##### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

##### NOTE

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

##### Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS
canExample SparkOnHbaseJavaExample-1.0.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode client --jars
SparkOnHbaseJavaExample-1.0.jar HBaseDistributedScanExample.py
ExampleAvrotable
```

- yarn-cluster mode:

Java/Scala version (The class name must be the same as the actual code. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --class
com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedS
canExample SparkOnHbaseJavaExample-1.0.jar ExampleAvrotable
```

Python version. (The file name must be the same as the actual one. The following is only an example.)

```
bin/spark-submit --master yarn --deploy-mode cluster --jars
SparkOnHbaseJavaExample-1.0.jar HBaseDistributedScanExample.py
ExampleAvrotable
```

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseDistributedScanExample file in SparkOnHbaseJavaExample.

```
public static void main(String[] args) throws IOException{
    if (args.length < 1) {
        System.out.println("JavaHBaseDistributedScan {tableName}");
        return;
    }
    String tableName = args[0];
    SparkConf sparkConf = new SparkConf().setAppName("JavaHBaseDistributedScan " + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try {
        Configuration conf = HBaseConfiguration.create();
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
        Scan scan = new Scan();
        scan.setCaching(100);
        JavaRDD<Tuple2<ImmutableBytesWritable, Result>> javaRdd =
            hbaseContext.hbaseRDD(tableName.valueOf(tableName), scan);
        List<String> results = javaRdd.map(new ScanConvertFunction()).collect();
        System.out.println("Result Size: " + results.size());
    } finally {
        jsc.stop();
    }
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseDistributedScanExample file in SparkOnHbaseScalaExample.

```
def main(args: Array[String]) {
    if (args.length < 1) {
        println("HBaseDistributedScanExample {tableName} missing an argument")
        return
    }
    val tableName = args(0)
```

```
val sparkConf = new SparkConf().setAppName("HBaseDistributedScanExample " + tableName )
val sc = new SparkContext(sparkConf)
try {
    val conf = HBaseConfiguration.create()
    val hbaseContext = new HBaseContext(sc, conf)
    val scan = new Scan()
    scan.setCaching(100)
    val getRdd = hbaseContext.hbaseRDD(TableName.valueOf(tableName), scan)
    getRdd.foreach(v => println(Bytes.toString(v._1.get())))
    println("Length: " + getRdd.map(r => r._1.copyBytes()).collect().length);
} finally {
    sc.stop()
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseDistributedScanExample` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-
# -*- coding:utf-8 -*-
"""
[Note]
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to
implement required operations.
"""

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession
# Create a SparkSession instance.
spark = SparkSession\
    .builder\
    .appName("JavaHBaseDistributedScan")\
    .getOrCreate()
# Import the required class int sc._jvm.
java_import(spark._jvm,
'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseDistributedScanExample')
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.
spark._jvm.JavaHBaseDistributedScan().execute(spark._jsc, sys.argv)
# Stop the SparkSession instance.
spark.stop()
```

### 2.17.3.4.9 Using the mapPartition Interface

#### Scenario

Users can use the `HBaseContext` method to perform operations on HBase in Spark applications and use the `mapPartition` interface to traverse HBase tables in parallel.

#### Data Planning

Use HBase tables created in [Using the foreachPartition Interface](#).

#### Development Guideline

1. Construct RDDs corresponding to rowkey in HBase tables to be traversed.
2. Use the `mapPartition` interface to traverse the data corresponding to the rowkey and perform simple operations.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `$SPARK_HOME`) on the server where the Spark client is located.

### NOTE

To run the Spark on HBase example program, set `spark.yarn.security.credentials.hbase.enabled` (`false` by default) in the `spark-defaults.conf` file on the Spark client to `true`. Changing the `spark.yarn.security.credentials.hbase.enabled` value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to `false`.) Set the value of the configuration item `spark.inputFormat.cache.enabled` to `false`.

## Submitting Commands

Assume that the JAR package name is `spark-hBaseContext-test-1.0.jar` that is stored in the `$SPARK_HOME` directory on the client. The following commands are executed in the `$SPARK_HOME` directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
`bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample SparkOnHbaseJavaExample-1.0.jar table2`  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
`bin/spark-submit --master yarn --deploy-mode client --jars SparkOnHbaseJavaExample-1.0.jar HBaseMapPartitionExample.py table2`
- yarn-cluster mode:  
Java/Scala version (The class name must be the same as the actual code. The following is only an example.)  
`bin/spark-submit --master yarn --deploy-mode cluster --class com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample SparkOnHbaseJavaExample-1.0.jar table2`  
Python version. (The file name must be the same as the actual one. The following is only an example.)  
`bin/spark-submit --master yarn --deploy-mode cluster --jars SparkOnHbaseJavaExample-1.0.jar HBaseMapPartitionExample.py table2`

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the `JavaHBaseMapPartitionExample` file in `SparkOnHbaseJavaExample`.

```
public static void main(String args[]) throws IOException {
    if(args.length <1){
        System.out.println("JavaHBaseMapPartitionExample {tableName} is missing an argument");
        return;
    }
}
```

```

    }
    final String tableName = args[0];
    SparkConf sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try{
        List<byte []> list = new ArrayList();
        list.add(Bytes.toBytes("1"));
        list.add(Bytes.toBytes("2"));
        list.add(Bytes.toBytes("3"));
        list.add(Bytes.toBytes("4"));
        list.add(Bytes.toBytes("5"));
        JavaRDD<byte []> rdd = jsc.parallelize(list);
        Configuration hbaseconf = HBaseConfiguration.create();
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, hbaseconf);
        JavaRDD getrdd = hbaseContext.mapPartitions(rdd, new
FlatMapFunction<Tuple2<Iterator<byte[]>,Connection>, Object>() {
            public Iterator call(Tuple2<Iterator<byte[]>, Connection> t)
                throws Exception {
                Table table = t._2.getTable(tableName);
                //go through rdd
                List<String> list = new ArrayList<String>();
                while(t._1.hasNext()){
                    byte[] bytes = t._1.next();
                    Result result = table.get(new Get(bytes));
                    Iterator<Cell> it = result.listCells().iterator();
                    StringBuilder sb = new StringBuilder();
                    sb.append(Bytes.toString(result.getRow()) + ":");
                    while(it.hasNext()){
                        Cell cell = it.next();
                        String column = Bytes.toString(cell.getQualifierArray());
                        if(column.equals("counter")){
                            sb.append("(" + column + "," + Bytes.toLong(cell.getValueArray()) + ")");
                        } else {
                            sb.append("(" + column + "," + Bytes.toString(cell.getValueArray()) + ")");
                        }
                    }
                    list.add(sb.toString());
                }
                return list.iterator();
            }
        });
        List<byte[]> resultList = getrdd.collect();
        if(null == resultList || 0 == resultList.size()){
            System.out.println("Nothing matches!");
        }else{
            for(int i = 0; i < resultList.size(); i++){
                System.out.println(resultList.get(i));
            }
        }
    } finally {
        jsc.stop();
    }
}

```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the HBaseMapPartitionExample file in SparkOnHbaseScalaExample.

```

def main(args: Array[String]) {
    if (args.length < 1) {
        println("HBaseMapPartitionExample {tableName} is missing an argument")
        return
    }
    val tableName = args(0)
    val sparkConf = new SparkConf().setAppName("HBaseMapPartitionExample " + tableName)
    val sc = new SparkContext(sparkConf)
    try {
        //[[Array[Byte]]]
    }
}

```

```
val rdd = sc.parallelize(Array(  
    Bytes.toBytes("1"),  
    Bytes.toBytes("2"),  
    Bytes.toBytes("3"),  
    Bytes.toBytes("4"),  
    Bytes.toBytes("5")))  
val conf = HBaseConfiguration.create()  
val hbaseContext = new HBaseContext(sc, conf)  
val b = new StringBuilder  
val getRdd = rdd.hbaseMapPartitions[String](hbaseContext, (it, connection) => {  
    val table = connection.getTable(TableName.valueOf(tableName))  
    it.map{r =>  
        //batching would be faster. This is just an example  
        val result = table.get(new Get(r))  
        val it = result.listCells().iterator()  
        b.append(Bytes.toString(result.getRow) + ":")  
        while (it.hasNext) {  
            val cell = it.next()  
            val q = Bytes.toString(cell.getQualifierArray)  
            if (q.equals("counter")) {  
                b.append("(" + q + "," + Bytes.toLong(cell.getValueArray) + ")")  
            } else {  
                b.append("(" + q + "," + Bytes.toString(cell.getValueArray) + ")")  
            }  
        }  
        b.toString()  
    }  
})  
getRdd.collect().foreach(v => println(v))  
} finally {  
    sc.stop()  
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseMapPartitionExample` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-  
'''  
[Note]  
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code.  
'''  
from py4j.java_gateway import java_import  
from pyspark.sql import SparkSession  
# Create a SparkSession instance.  
spark = SparkSession\  
    .builder\  
    .appName("JavaHBaseMapPartitionExample")\  
    .getOrCreate()  
# Import the required class to sc._jvm.  
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.hbasecontext.JavaHBaseMapPartitionExample')  
# Create a class instance and invoke the method. Transfer the sc._jsc parameter.  
spark._jvm.JavaHBaseMapPartitionExample().execute(spark._jsc, sys.argv)  
# Stop the SparkSession instance.  
spark.stop()
```

### 2.17.3.4.10 Writing Data to HBase Tables In Batches Using SparkStreaming

#### Scenario

Users can use `HBaseContext` to perform operations on HBase in Spark applications and write streaming data to HBase tables using the `streamBulkPut` interface.

## Data Planning

1. Create a session connected to the client and run the **hbase shell** command in the session to go to the HBase command line.
2. Run the following command in the HBase command line to create an HBase table:  
**create 'streamingTable','cf1'**
3. In another session of the client, run the Linux command to construct a port for receiving data. The command may be different for servers running different operating systems. For the SUSE operating system, the following command is used: **netcat -lk 9999**.

 **NOTE**

To construct a port for receiving data, you need to install netcat on the server where the client is located.

## Development Guideline

1. Use SparkStreaming to continuously read data from a specific port.
2. Write the read Dstream to HBase tables through the streamBulkPut interface.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **\$SPARK\_HOME**) on the server where the Spark client is located.

 **NOTE**

To run the Spark on HBase example program, set **spark.yarn.security.credentials.hbase.enabled** (**false** by default) in the **spark-defaults.conf** file on the Spark client to **true**. Changing the **spark.yarn.security.credentials.hbase.enabled** value does not affect existing services. (To uninstall the HBase service, you need to change the value of this parameter back to **false**.) Set the value of the configuration item **spark.inputFormat.cache.enabled** to **false**.

## Submitting Commands

Assume that the JAR package name is **spark-hbaseContext-test-1.0.jar** that is stored in the **\$SPARK\_HOME** directory on the client. The following commands are executed in the **\$SPARK\_HOME** directory, and Java is displayed before the class name of the Java interface. For details, see the sample code.

- yarn-client mode:  
Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) **\${ip}** must be the IP address of the host where the **nc -lk 9999** command is executed.  
**bin/spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample SparkOnHbaseJavaExample-1.0.jar \${ip} 9999 streamingTable cf1**

Python version. (The file name must be the same as the actual one. The following is only an example.)

**bin/spark-submit --master yarn --jars SparkOnHbaseJavaExample-1.0.jar HBaseStreamingBulkPutExample.py \${ip} 9999 streamingTable cf1**

- yarn-cluster mode:

Java/Scala version. (The class name must be the same as the actual code. The following is only an example.) \${ip} must be the IP address of the host where the nc -lk 9999 command is executed.

**bin/spark-submit --master yarn --deploy-mode client --deploy-mode cluster --class com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample SparkOnHbaseJavaExample-1.0.jar \${ip} 9999 streamingTable cf1**

Python version. (The file name must be the same as the actual one. The following is only an example.)

**bin/spark-submit --master yarn --deploy-mode cluster --jars SparkOnHbaseJavaExample-1.0.jar HBaseStreamingBulkPutExample.py \${ip} 9999 streamingTable cf1**

## Java Sample Code

The following code snippet is only for demonstration. For details about the code, see the JavaHBaseStreamingBulkPutExample file in SparkOnHbaseJavaExample.



### NOTE

The **awaitTerminationOrTimeout()** method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
public static void main(String[] args) throws IOException {
    if (args.length < 4) {
        System.out.println("JavaHBaseBulkPutExample " +
                           "{host} {port} {tableName}");
        return;
    }
    String host = args[0];
    String port = args[1];
    String tableName = args[2];
    String columnFamily = args[3];
    SparkConf sparkConf =
        new SparkConf().setAppName("JavaHBaseStreamingBulkPutExample " +
                                   tableName + ":" + port + ":" + tableName);
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);
    try {
        JavaStreamingContext jssc =
            new JavaStreamingContext(jsc, new Duration(1000));
        JavaReceiverInputDStream<String> javaDstream =
            jssc.socketTextStream(host, Integer.parseInt(port));
        Configuration conf = HBaseConfiguration.create();
        JavaHBaseContext hbaseContext = new JavaHBaseContext(jsc, conf);
        hbaseContext.streamBulkPut(javaDstream,
                                  TableName.valueOf(tableName),
                                  new PutFunction(columnFamily));
        jssc.start();
        jssc.awaitTerminationOrTimeout(60000);
        jssc.stop(false,true);
    }catch(InterruptedException e){
        e.printStackTrace();
    } finally {
```

```
        jsc.stop();  
    }  
}
```

## Scala Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseStreamingBulkPutExample` file in `SparkOnHbaseScalaExample`.

### NOTE

The `awaitTerminationOrTimeout()` method is used to set the task timeout interval (in milliseconds). You are advised to set this parameter based on the expected task execution time.

```
def main(args: Array[String]): Unit = {  
    val host = args(0)  
    val port = args(1)  
    val tableName = args(2)  
    val columnFamily = args(3)  
    val conf = new SparkConf()  
    conf.setAppName("HBase Streaming Bulk Put Example")  
    val sc = new SparkContext(conf)  
    try {  
        val config = HBaseConfiguration.create()  
        val hbaseContext = new HBaseContext(sc, config)  
        val ssc = new StreamingContext(sc, Seconds(1))  
        val lines = ssc.socketTextStream(host, port.toInt)  
        hbaseContext.streamBulkPut[String](lines,  
            TableName.valueOf(tableName),  
            (putRecord) => {  
                if (putRecord.length() > 0) {  
                    val put = new Put(Bytes.toBytes(putRecord))  
                    put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("foo"), Bytes.toBytes("bar"))  
                    put  
                } else {  
                    null  
                }  
            })  
        ssc.start()  
        ssc.awaitTerminationOrTimeout(60000)  
        ssc.stop(stopSparkContext = false)  
    } finally {  
        sc.stop()  
    }  
}
```

## Python Sample Code

The following code snippet is only for demonstration. For details about the code, see the `HBaseStreamingBulkPutExample` file in `SparkOnHbasePythonExample`.

```
# -*- coding:utf-8 -*-  
"""  
[Note]  
PySpark does not provide HBase-related APIs. In this example, Python is used to invoke Java code to  
implement required operations.  
"""  
from py4j.java_gateway import java_import  
from pyspark.sql import SparkSession  
# Create a SparkSession instance.  
spark = SparkSession\  
    .builder\  
    .appName("JavaHBaseStreamingBulkPutExample")\  
    .getOrCreate()  
# Import required class to sc._jvm.  
java_import(spark._jvm,
```

```
'com.huawei.bigdata.spark.examples.streaming.JavaHBaseStreamingBulkPutExample'  
# Create class instance and invoke the method. Transfer the sc._jsc parameter.  
spark._jvm.JavaHBaseStreamingBulkPutExample().execute(spark._jsc, sys.argv)  
# Stop SparkSession.  
spark.stop()
```

## 2.17.3.5 Reading Data from HBase and Write It Back to HBase

### 2.17.3.5.1 Overview

#### Scenarios

Assume table1 of HBase stores the user consumption amount of the current day and table2 stores the history consumption data.

In table1, key=1 and cf:cid=100 indicate that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

#### Data Preparation

Use the Spark-Beeline tool to create table1 and table2 (Spark table and HBase table, respectively), and insert data by HBase.

**Step 1** Ensure that JDBCServer is started. On the Spark client, perform the following operations using the Spark-Beeline command tool:

**Step 2** Use the Spark-Beeline tool to create Spark table1:

```
create table table1  
(  
    key string,  
    cid string  
)  
using org.apache.spark.sql.hbase.HBaseSource  
options(  
    hbaseTableName "table1",  
    keyCols "key",  
    colsMapping "cid=cf.cid");
```

**Step 3** Run the following command on HBase to insert data to table1:

```
put 'table1', '1', 'cf:cid', '100'
```

**Step 4** Use the Spark-Beeline tool to create Spark table2:

```
create table table2
(
    key string,
    cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
    hbaseTableName "table2",
    keyCols "key",
    colsMapping "cid=cf.cid");
```

**Step 5** Run the following command on HBase to insert data to table2 :

```
put 'table2', '1', 'cf:cid', '1000'
```

**Step 6** Run the following command to update **table1**:

```
flush 'table1'
----End
```

## Development Idea

1. Query the data in table1.
2. Query the data in table2 using the key value of table1.
3. Add up the queried data.
4. Write the results of the preceding step to table2.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file (The class name and file name must be the same as those in the actual code. The following is only an example). For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.



### NOTE

Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services. If you want to uninstall the HBase service, change the value back to false first.)

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the **bin/spark-submit** script to run the code:

- Run Java or Scala sample code.  
**bin/spark-submit --jars --conf spark.yarn.user.classpath.first=true --class com.huawei.bigdata.spark.examples.SparkHbasetoHbase --master yarn --deploy-mode client /opt/female/SparkHbasetoHbase-1.0.jar**
- Run the Python sample program.

### NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR package and place it in the same driver class directory. When running the Python program, configure **--jars** to load the JAR package to the directory where the Python file resides.

```
bin/spark-submit --master yarn --deploy-mode client --conf
spark.yarn.user.classpath.first=true --jars /opt/female/
SparkHbasetoHbasePythonExample/SparkHbasetoHbase-1.0.jar,/opt/female/
protobuf-java-2.5.0.jar /opt/female/SparkHbasetoHbasePythonExample/
SparkHbasetoHbasePythonExample.py
```

### 2.17.3.5.2 Java Example Code

## Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

## Example Code

For details about the code, see the class  
`com.huawei.bigdata.spark.examples.SparkHbasetoHbase`.

Example code:

```
/*
 * calculate data from hbase1/hbase2,then update to hbase2
 */
public class SparkHbasetoHbase {

    public static void main(String[] args) throws Exception {

        SparkConf conf = new SparkConf().setAppName("SparkHbasetoHbase");
        conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
        conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator");
        JavaSparkContext jsc = new JavaSparkContext(conf);
        // Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the
        //classpath.
        Configuration hbConf = HBaseConfiguration.create(jsc.hadoopConfiguration());

        // Declare the information of the table to be queried.
        Scan scan = new org.apache.hadoop.hbase.client.Scan();
        scan.addFamily(Bytes.toBytes("cf"));//column family
        org.apache.hadoop.hbase.protobuf.generated.ClientProtos.Scan proto = ProtobufUtil.toScan(scan);
        String scanToString = Base64.encodeBytes(proto.toByteArray());
        hbConf.set(TableInputFormat.INPUT_TABLE, "table1");//table name
        hbConf.set(TableInputFormat.SCAN, scanToString);
    }
}
```

```
// Obtain the data in the table through the Spark interface.  
JavaPairRDD rdd = jsc.newAPIHadoopRDD(hbConf, TableInputFormat.class,  
ImmutableBytesWritable.class, Result.class);  
  
// Traverse every Partition in the HBase table1 and update the HBase table2  
//If less data, you can use rdd.foreach()  
  
rdd.foreachPartition(  
    new VoidFunction<Iterator<Tuple2<ImmutableBytesWritable, Result>>>() {  
        public void call(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws Exception {  
            hBaseWriter(iterator);  
        }  
    }  
);  
  
jsc.stop();  
}  
  
/**  
 * write to table2 in exetutor  
 *  
 * @param iterator partition data from table1  
 */  
private static void hBaseWriter(Iterator<Tuple2<ImmutableBytesWritable, Result>> iterator) throws  
IOException {  
    //read hbase  
    String tableName = "table2";  
    String columnFamily = "cf";  
    String qualifier = "cid";  
    Configuration conf = HBaseConfiguration.create();  
  
    Connection connection = null;  
    Table table = null;  
  
    try {  
        connection = ConnectionFactory.createConnection(conf);  
        table = connection.getTable(Table.Name.valueOf(tableName));  
  
        List<Get> rowList = new ArrayList<Get>();  
        List<Tuple2<ImmutableBytesWritable, Result>> table1List = new  
ArrayList<Tuple2<ImmutableBytesWritable, Result>>();  
        while (iterator.hasNext()) {  
            Tuple2<ImmutableBytesWritable, Result> item = iterator.next();  
            Get get = new Get(item._2().getRow());  
            table1List.add(item);  
            rowList.add(get);  
        }  
  
        //get data from hbase table2  
        Result[] resultDataBuffer = table.get(rowList);  
  
        //set data for hbase  
        List<Put> putList = new ArrayList<Put>();  
        for (int i = 0; i < resultDataBuffer.length; i++) {  
            Result resultData = resultDataBuffer[i];//hbase2 row  
            if (!resultData.isEmpty()) {  
                //query hbase1Value  
                String hbase1Value = "";  
                Iterator<Cell> it = table1List.get(i)._2().listCells().iterator();  
                while (it.hasNext()) {  
                    Cell c = it.next();  
                    // query table1 value by column family and column qualifier  
                    if (columnFamily.equals(Bytes.toString(CellUtil.cloneFamily(c)))  
                        && qualifier.equals(Bytes.toString(CellUtil.cloneQualifier(c)))) {  
                        hbase1Value = Bytes.toString(CellUtil.cloneValue(c));  
                    }  
                }  
            }  
        }  
    }  
}
```

```
String hbase2Value = Bytes.toString(resultData.getValue(columnFamily.getBytes(),
qualifier.getBytes()));
Put put = new Put(table1List.get(i)._2().getRow());

//calculate result value
int resultValue = Integer.parseInt(hbase1Value) + Integer.parseInt(hbase2Value);
//set data to put
put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifier),
Bytes.toBytes(String.valueOf(resultValue)));
putList.add(put);
}

if (putList.size() > 0) {
    table.put(putList);
}
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (table != null) {
        try {
            table.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    if (connection != null) {
        try {
            // Close the HBase connection
            connection.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

}
```

### 2.17.3.5.3 Scala Example Code

## Function

Call HBase API using Spark to operate HBase table1, analyze the data, and then write the analyzed data to HBase table2.

## Example Code

For details about code, see  
[com.huawei.bigdata.spark.examples.SparkHbasetoHbase](#).

Example code:

```
/*
 * calculate data from hbase1/hbase2,then update to hbase2
 */
object SparkHbasetoHbase {

    case class FemaleInfo(name: String, gender: String, stayTime: Int)

    def main(args: Array[String]) {

        val conf = new SparkConf().setAppName("SparkHbasetoHbase")
        conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
        conf.set("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator")
        val sc = new SparkContext(conf)
    }
}
```

```
// Create the configuration parameter to connect the HBase. The hbase-site.xml must be included in the  
classpath.  
val hbConf = HBaseConfiguration.create(sc.hadoopConfiguration)  
  
// Declare the information of the table to be queried.  
val scan = new Scan()  
scan.addFamily(Bytes.toBytes("cf")) //column family  
val proto = ProtobufUtil.toScan(scan)  
val scanToString = Base64.encodeBytes(proto.toByteArray)  
hbConf.set(TableInputFormat.INPUT_TABLE, "table1") //table name  
hbConf.set(TableInputFormat.SCAN, scanToString)  
  
// Obtain the data in the table through the Spark interface.  
val rdd = sc.newAPIHadoopRDD(hbConf, classOf[TableInputFormat], classOf[ImmutableBytesWritable],  
classOf[Result])  
  
// Traverse every Partition in the HBase table1 and update the HBase table2  
//If less data, you can use rdd.foreach()  
rdd.foreachPartition(x => hBaseWriter(x))  
  
sc.stop()  
}  
  
/**  
 * write to table2 in exetutor  
 *  
 * @param iterator partition data from table1  
 */  
def hBaseWriter(iterator: Iterator[(ImmutableBytesWritable, Result)]): Unit = {  
    //read hbase  
    val tableName = "table2"  
    val columnFamily = "cf"  
    val qualifier = "cid"  
    val conf = HBaseConfiguration.create()  
  
    var table: Table = null  
    var connection: Connection = null  
  
    try {  
        connection = ConnectionFactory.createConnection(conf)  
        table = connection.getTable(TableName.valueOf(tableName))  
  
        val iteratorArray = iterator.toArray  
        val rowList = new util.ArrayList[Get]()  
        for (row <- iteratorArray) {  
            val get = new Get(row._2.getRow)  
            rowList.add(get)  
        }  
  
        //get data from hbase table2  
        val resultDataBuffer = table.get(rowList)  
  
        //set data for hbase  
        val putList = new util.ArrayList[Put]()  
        for (i <- 0 until iteratorArray.size) {  
            val resultData = resultDataBuffer(i) //hbase2 row  
            if (!resultData.isEmpty) {  
                //query hbase1Value  
                var hbase1Value = ""  
                val it = iteratorArray(i)._2.listCells().iterator()  
                while (it.hasNext) {  
                    val c = it.next()  
                    // query table1 value by column family and column qualifier  
                    if (columnFamily.equals(Bytes.toString(CellUtil.cloneFamily(c)))  
                        && qualifier.equals(Bytes.toString(CellUtil.cloneQualifier(c)))) {  
                        hbase1Value = Bytes.toString(CellUtil.cloneValue(c))  
                    }  
                }  
            }  
        }  
    }  
}
```

```
val hbase2Value = Bytes.toString(resultData.getValue(columnFamily.getBytes, qualifier.getBytes))
val put = new Put(iteratorArray(i)._2.getRow)

    //calculate result value
    val resultValue = hbase1Value.toInt + hbase2Value.toInt
    //set data to put
    put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifier),
    Bytes.toBytes(resultValue.toString))
    putList.add(put)
}
}

if (putList.size() > 0) {
    table.put(putList)
}
} catch {
    case e: IOException =>
        e.printStackTrace();
} finally {
    if (table != null) {
        try {
            table.close()
        } catch {
            case e: IOException =>
                e.printStackTrace();
        }
    }
    if (connection != null) {
        try {
            //Close the HBase connection.
            connection.close()
        } catch {
            case e: IOException =>
                e.printStackTrace()
        }
    }
}
}

/**
 * Define serializer class.
 */
class MyRegistrar extends KryoRegistrar {
    override def registerClasses(kryo: Kryo) {
        kryo.register(classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable])
        kryo.register(classOf[org.apache.hadoop.hbase.client.Result])
        kryo.register(classOf[Array[(Any, Any)]])
        kryo.register(classOf[Array[org.apache.hadoop.hbase.Cell]])
        kryo.register(classOf[org.apache.hadoop.hbase.NoTagsKeyValue])
        kryo.register(classOf[org.apache.hadoop.hbase.protobuf.generated.ClientProtos.RegionLoadStats])
    }
}
```

#### 2.17.3.5.4 Python Example Code

### Function

Use Spark to call an HBase API to operate HBase table1 and write the data analysis result of table1 to HBase table2.

### Example Code

PySpark does not provide HBase related APIs. In this example, Python with the Java programming language invoked is used.

The following code snippets are used as an example. For complete codes, see [SparkHbasetoHbasePythonExample](#).

```
# -*- coding:utf-8 -*-

from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

# Create the SparkContext and set kryo serialization.
spark = SparkSession\
    .builder\
    .appName("SparkHbasetoHbase") \
    .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
    .config("spark.kryo.registrator", "com.huawei.bigdata.spark.examples.MyRegistrator") \
    .getOrCreate()

# Import the class that will run into sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHbasetoHbase')

# Create a class instance and invoke the method.
spark._jvm.SparkHbasetoHbase().hbasetohbase(spark._jsc)

# Stop the SparkSession
spark.stop()
```

## 2.17.3.6 Reading Data from Hive and Write It to HBase

### 2.17.3.6.1 Overview

#### Scenarios

Assume that table **person** of Hive stores the user consumption amount of the current day and table2 of HBase stores the history consumption data.

In table person, name=1 and account=100 indicates that the consumption amount of user 1 in the current day is 100 CNY.

In table2, key=1 and cf:cid=1000 indicate that the history consumption amount of user 1 is 1000 CNY.

The Spark application shall achieve the following function:

Add the current consumption amount (100) to the history consumption amount (1000).

The running result is that the total consumption amount of user 1 (key=1) in table2 is 1100 CNY (cf:cid=1100).

#### Data Preparation

Before developing the application, create the Hive table **person** and insert data to it. Create HBase table2.

##### Step 1 Place the source log file to HDFS.

1. Create a blank file **log1.txt** in the local and write the following content to the file:  
1,100
2. Create a directory **/tmp/input** in HDFS and copy the **log1.txt** file to the directory.

- a. On the Linux HDFS client, run the **hadoop fs -mkdir /tmp/input** command (a **hdfs dfs** command provides the same function.) to create a directory.
- b. On the Linux HDFS client, run the **hadoop fs -put log1.txt /tmp/input** command to upload the data file.

**Step 2** Store the imported data to the Hive table.

Ensure that JDBCServer is started. Use the Beeline tool to create a Hive table and insert data to it.

1. Run the following commands to create the Hive table person:

```
create table person
(
    name STRING,
    account INT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ESCAPED BY '\\'
STORED AS TEXTFILE;
```

2. Run the following command to insert data to the table:

```
load data inpath '/tmp/input/log1.txt' into table person;
```

**Step 3** Create a HBase table:

Ensure that JDBCServer is started. Use the Spark-beeline command tool to create a HBase table and insert data to it.

1. Run the following commands to create the HBase table table2:

```
create table table2
(
    key string,
    cid string
)
using org.apache.spark.sql.hbase.HBaseSource
options(
    hbaseTableName "table2",
    keyCols "key",
    colsMapping "cid=cf.cid");
```

2. Run the following command to insert data to the table:

```
put 'table2', '1', 'cf:cid', '1000'
```

----End

## Development Idea

1. Query the data in Hive table person.
2. Query the data in table2 using the key value of table person.
3. Add the queried data.

4. Write the results of the preceding step to table2.

## Packaging the Project

1. Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
2. Upload the JAR file to any directory (for example, `/opt/female/`) on the server where the Spark client is located.

### NOTE

Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services. If you want to uninstall the HBase service, change the value back to false first.)

## Running Tasks

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

- Run Java or Scala sample code.
  - `bin/spark-submit --class com.huawei.bigdata.spark.examples.SparkHivetoHbase --master yarn --deploy-mode client /opt/female/SparkHivetoHbase-1.0.jar`
- Run the Python sample program

### NOTE

- PySpark does not provide HBase-related APIs. Therefore, Python is used to invoke Java code in this sample. Use Maven to pack the provided Java code into a JAR file and place it in the same directory. When running the Python program, use `--jars` to load the JAR file to `classpath`
  - `bin/spark-submit --master yarn --deploy-mode client --jars /opt/female/SparkHivetoHbasePythonExample/SparkHivetoHbase-1.0.jar /opt/female/SparkHivetoHbasePythonExample/SparkHivetoHbasePythonExample.py`

### 2.17.3.6.2 Java Sample Code

## Function

In a Spark application, users can use Spark to call a Hive API to operate a Hive table, and write the data analysis result of the Hive table to an HBase table.

## Code Sample

The following code snippets are used as an example. For complete codes, see [com.huawei.bigdata.spark.examples.SparkHivetoHbase](#).

```
/**  
 * Read data from the Hive table, and obtain the corresponding record from the HBase table based on the  
 * key value. Sum the obtained two data records and update the sum result to the HBase table.  
 */  
public class SparkHivetoHbase {
```

```
public static void main(String[] args) throws Exception {
    // Use the Spark API to obtain table data.
    SparkConf conf = new SparkConf().setAppName("SparkHivetoHbase");
    JavaSparkContext jsc = new JavaSparkContext(conf);
    HiveContext sqlContext = new org.apache.spark.sql.hive.HiveContext(jsc);

    Dataset<Row> dataFrame = sqlContext.sql("select name, account from person");

    // Traverse every partition in the Hive table and update data to the HBase table.
    // If the number of data records is small, you can use the foreach() method.
    dataFrame.toJavaRDD().foreachPartition(
        new VoidFunction<Iterator<Row>>() {
            public void call(Iterator<Row> iterator) throws Exception {
                hBaseWriter(iterator);
            }
        });
    spark.stop();
}

/**
 * Update records in the HBase table on the executor.
 *
 * @param iterator Partition data in the Hive table.
 */
private static void hBaseWriter(Iterator<Row> iterator) throws IOException {
    // Read the HBase table.
    String tableName = "table2";
    String columnFamily = "cf";
    Configuration conf = HBaseConfiguration.create();
    Connection connection = null;
    Table table = null;
    try {
        connection =ConnectionFactory.createConnection(conf);
        table = connection.getTable(Table.Name.valueOf(tableName));
        List<Row> table1List = new ArrayList<Row>();
        List<Get> rowList = new ArrayList<Get>();
        while (iterator.hasNext()) {
            Row item = iterator.next();
            Get get = new Get(item.getString(0).getBytes());
            table1List.add(item);
            rowList.add(get);
        }
        // Obtain the records in the HBase table.
        Result[] resultDataBuffer = table.get(rowList);
        // Modify records in the HBase table.
        List<Put> putList = new ArrayList<Put>();
        for (int i = 0; i < resultDataBuffer.length; i++) {
            // Hive table value
            Result resultData = resultDataBuffer[i];
            if (!resultData.isEmpty()) {
                // get hiveValue
                int hiveValue = table1List.get(i).getInt(1);
                // Obtain the HBase table value based on the column family and column.
                String hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes(), "cid".getBytes()));
                Put put = new Put(table1List.get(i).getString(0).getBytes());
                // Calculate the result.
                int resultValue = hiveValue + Integer.valueOf(hbaseValue);
                // Set the result to the Put object.
                put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"),
                    Bytes.toBytes(String.valueOf(resultValue)));
                putList.add(put);
            }
        }
        if (putList.size() > 0)
            table.put(putList);
    }
```

```
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (table != null) {
                try {
                    table.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (connection != null) {
                try {
                    // Close the HBase connection.
                    connection.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

### 2.17.3.6.3 Scala Example Code

#### Function

In Spark application, call Hive API using Spark to operate Hive table, analyze the data, and then write the analyzed data to the HBase table.

#### Example Code

For details about code, see  
[com.huawei.bigdata.spark.examples.SparkHbasetoHbase](#).

##### Example code

```
/*
 * calculate data from hive/hbase,then update to hbase
 */
object SparkHivetoHbase {

    case class FemaleInfo(name: String, gender: String, stayTime: Int)

    def main(args: Array[String]) {
        // Obtain the data in the table through the Spark interface.

        val spark = SparkSession
            .builder()
            .appName("SparkHiveHbase")
            .config("spark.sql.warehouse.dir", "spaek-warehouse")
            .enableHiveSupport()
            .getOrCreate()

        import spark.implicits._

        val dataFrame = spark.sql("select name, account from person")

        // Traverse every Partition in the hive table and update the hbase table
        // If less data, you can use rdd.foreach()
        dataFrame.rdd.foreachPartition(x => hBaseWriter(x))

        spark.stop()
    }

    /**
     * write to hbase table in exetutor

```

```
*  
* @param iterator partition data from hive table  
*/  
def hBaseWriter(iterator: Iterator[Row]): Unit = {  
    // read hbase  
    val tableName = "table2"  
    val columnFamily = "cf"  
    val conf = HBaseConfiguration.create()  
  
    var table: Table = null  
    var connection: Connection = null  
  
    try {  
        connection = ConnectionFactory.createConnection(conf)  
        table = connection.getTable(Table.Name.valueOf(tableName))  
  
        val iteratorArray = iterator.toArray  
        val rowList = new util.ArrayList[Get]()  
        for (row <- iteratorArray) {  
            val get = new Get(row.getString(0).getBytes)  
            rowList.add(get)  
        }  
  
        // get data from hbase table  
        val resultDataBuffer = table.get(rowList)  
  
        // set data for hbase  
        val putList = new util.ArrayList[Put]()
        for (i < 0 until iteratorArray.size) {  
            // hbase row  
            val resultData = resultDataBuffer(i)  
            if (!resultData.isEmpty) {  
                // get hiveValue  
                var hiveValue = iteratorArray(i).getInt(1)  
  
                // get hbaseValue by column Family and colomn qualifier  
                val hbaseValue = Bytes.toString(resultData.getValue(columnFamily.getBytes, "cid".getBytes))  
                val put = new Put(iteratorArray(i).getString(0).getBytes)  
  
                // calculate result value  
                val resultValue = hiveValue + hbaseValue.toInt  
  
                // set data to put  
                put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes("cid"),  
                Bytes.toBytes(resultValue.toString))
                putList.add(put)  
            }  
        }  
  
        if (putList.size() > 0) {  
            table.put(putList)  
        }  
    } catch {  
        case e: IOException =>  
            e.printStackTrace();  
    } finally {  
        if (table != null) {  
            try {  
                table.close()  
            } catch {  
                case e: IOException =>  
                    e.printStackTrace();  
            }  
        }  
        if (connection != null) {  
            try {  
                //Close the HBase connection.  
                connection.close()  
            } catch {  
                case e: IOException =>  
                    e.printStackTrace();  
            }  
        }  
    }  
}
```

```
        case e: IOException =>
          e.printStackTrace()
      }
    }
}
}
```

### 2.17.3.6.4 Python Example Code

#### Function

In a Spark application, use Spark to call a Hive API to operate a Hive table, and write the data analysis result of the Hive table to an HBase table.

#### Example Code

PySpark does not provide HBase related APIs. In this example, Python with the Java programming language invoked is used.

The following code snippets are used as an example. For complete codes, see [SparkHivetoHbasePythonExample](#).

```
# -*- coding:utf-8 -*-
from py4j.java_gateway import java_import
from pyspark.sql import SparkSession

# Create the SparkSession
spark = SparkSession\
  .builder\
  .appName("SparkHivetoHbase") \
  .getOrCreate()

# Import the class that will run into sc._jvm.
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkHivetoHbase')

# Create a class instance and invoke the method.
spark._jvm.SparkHivetoHbase().hivetohbase(spark._jsc)

# Stop the SparkSession
spark.stop()
```

### 2.17.3.7 Streaming Connecting to Kafka0-10

#### 2.17.3.7.1 Overview

#### Scenarios

Assume that the Kafka component receives one word record every 1 second.

The developed Spark application needs to achieve the following function:

Calculate the sum of records for each word in real time.

**log1.txt** example file:

```
LiuYang
YuanJing
GuoYijun
CaiXuyu
```

Liyuan  
FangBo  
LiuYang  
YuanJing  
GuoYijun  
CaiXuyu  
FangBo

## Data Planning

Spark Streaming sample project data is stored in the Kafka component. A user with Kafka permission sends data to Kafka component.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.
2. Create the **input\_data1.txt** file in the local and copy the content of the **log1.txt** file to the **input\_data1.txt** file.

On the client installation node, create the **/home/data** directory and upload the **input\_data1.txt** file to the **/home/data** directory.

3. Create a topic.

{*IP address of the Kafka cluster*} indicates the service IP address of the Broker service.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --bootstrap-server <IP address of the Kafka cluster:21005> --command-config $KAFKA_HOME/config/client.properties --replication-factor 1 --partitions 3 --topic {Topic}
```

4. Start the Producer of Kafka and send data to Kafka.

```
java -cp {ClassPath}  
com.huawei.bigdata.spark.examples.StreamingExampleProducer  
{BrokerList} {Topic}
```

In this command, **ClassPath** must contain the absolute path of the Kafka JAR package on the Spark client in addition to the path of the sample JAR package, for example: **/opt/hadoopclient/Spark/spark/jars/\*:/opt/hadoopclient/Spark/spark/jars/streamingClient010/\*:{ClassPath}**.

## Development Approach

1. Receive data from Kafka and generate DStream.
2. Collect the statistics of word records by category.
3. Calculate and print the result.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, **/opt**) on the server where the Spark client is located.
- Prepare dependency packages and upload the following JAR packages to the **\$SPARK\_HOME/jars/streamingClient010** directory on the server where the Spark client is located.
  - **spark-streaming-kafkaWriter-xxx.cbu.mrs.jar**
  - **kafka-clients-xxx.jar**

- spark-sql-kafka-xxx.cbu.mrs.xxx.jar
- spark-streaming-kafka-xxx.cbu.mrs.xxx.jar
- spark-token-provider-kafka-xxx.cbu.mrs.xxx.jar

 NOTE

- For the dependency package whose version number contains "cbu.mrs", download from the Huawei open-source image site.
- For the dependency package whose version number does not contain "cbu.mrs", obtain them from the Maven central repository.

## Running Tasks

When running the sample program, you need to specify <checkpointDir>, <brokers>, <topic>, and <batchTime>. <checkPointDir> indicates the path for storing the program result backup in HDFS. <brokers> indicates the Kafka address for obtaining metadata. <topic> indicates the topic name read from Kafka. <batchTime> indicates the interval for Streaming processing in batches.

 NOTE

The path of Spark Streaming's Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**, and the path of the Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, **--jars \${files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar);IFS=,; echo "\${files[\*]}"}"**.

Go to the Spark client directory and run the following commands to invoke the bin/spark-submit script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example.):

- Sample code (**Spark Streaming read Kafka 0-10 Write To Print**)  
**bin/spark-submit --master yarn --deploy-mode client --jars \${files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar);IFS=,; echo "\${files[\*]}"}" --class com.huawei.bigdata.spark.examples.KafkaWordCount /opt/SparkStreamingKafka010JavaExample-1.0.jar <checkpointDir> <brokers> <topic> <batchTime>**
- Sample code (**Spark Streaming Write To Kafka 0-10**)  
**bin/spark-submit --master yarn --deploy-mode client --jars \${files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar);IFS=,; echo "\${files[\*]}"}" --class com.huawei.bigdata.spark.examples.JavaDstreamKafkaWriter /opt/SparkStreamingKafka010JavaExample-1.0.jar <groupId> <brokers> <topics>**

### 2.17.3.7.2 Java Example Code

## Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

## Code Sample (Streaming Read Data from Kafka 0-10)

The following code is an example. For details, see  
`com.huawei.bigdata.spark.examples.KafkaWordCount`.

```
/*
 * Consumes messages from one or more topics in Kafka.
 * <checkPointDir> is the Spark Streaming checkpoint directory.
 * <brokers> is for bootstrapping and the producer will only use it for getting metadata
 * <topics> is a list of one or more kafka topics to consume from
 * <batchTime> is the Spark Streaming batch duration in seconds.
 */
public class KafkaWordCount
{
    public static void main(String[] args) {
        JavaStreamingContext ssc = createContext(args);
        //The Streaming system starts.
        ssc.start();
        try {
            ssc.awaitTermination();
        } catch (InterruptedException e) {
        }
    }

    private static JavaStreamingContext createContext(String[] args) {
        String checkPointDir = args[0];
        String brokers = args[1];
        String topics = args[2];
        String batchTime = args[3];

        // Create a Streaming startup environment.
        SparkConf sparkConf = new SparkConf().setAppName("KafkaWordCount");
        JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, new
Duration(Long.parseLong(batchTime) * 1000));

        //Configure the CheckPoint directory for the Streaming.
        //This parameter is mandatory because of existence of the window concept.
        ssc.checkpoint(checkPointDir);

        // Get the list of topic used by kafka
        String[] topicArr = topics.split(",");
        Set<String> topicSet = new HashSet<String>(Arrays.asList(topicArr));
        Map<String, Object> kafkaParams = new HashMap();
        kafkaParams.put("bootstrap.servers", brokers);
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("group.id", "DemoConsumer");

        LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();
        ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicSet, kafkaParams);

        // Create direct kafka stream with brokers and topics
        // Receive data from the Kafka and generate the corresponding DStream
        JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(ssc,
locationStrategy, consumerStrategy);

        // Obtain field properties in each row.
        JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {
            @Override
            public String call(ConsumerRecord<String, String> tuple2) throws Exception {
                return tuple2.value();
            }
        });

        // Aggregate the total time that calculate word count
        JavaPairDStream<String, Integer> wordCounts = lines.mapToPair(
            new PairFunction<String, String, Integer>() {
                @Override
                public Tuple2<String, Integer> call(String s) {
                    return new Tuple2<String, Integer>(s, 1);
                }
            });
    }
}
```

```
        }
    }).reduceByKey(new Function2<Integer, Integer, Integer>() {
        @Override
        public Integer call(Integer i1, Integer i2) {
            return i1 + i2;
        }
    }).updateStateByKey(
        new Function2<List<Integer>, Optional<Integer>, Optional<Integer>>() {
            @Override
            public Optional<Integer> call(List<Integer> values, Optional<Integer> state) {
                int out = 0;
                if (state.isPresent()) {
                    out += state.get();
                }
                for (Integer v : values) {
                    out += v;
                }
                return Optional.of(out);
            }
        });
    });

    // print the results
    wordCounts.print();
    return ssc;
}
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see [com.huawei.bigdata.spark.examples.DstreamKafkaWriter](#).

### NOTE

It is advisable to use the new API `createDirectStream` instead of the old API `createStream` for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
/**
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
public class JavaDstreamKafkaWriter {

    public static void main(String[] args) throws InterruptedException {
        if (args.length != 3) {
            System.err.println("Usage: JavaDstreamKafkaWriter <groupId> <brokers> <topic>");
            System.exit(1);
        }

        final String groupId = args[0];
        final String brokers = args[1];
        final String topic = args[2];

        SparkConf sparkConf = new SparkConf().setAppName("KafkaWriter");

        // Populate Kafka properties
        Map<String, Object> kafkaParams = new HashMap<String, Object>();
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("value.serializer", "org.apache.kafka.common.serialization.ByteArraySerializer");
        kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        kafkaParams.put("bootstrap.servers", brokers);
        kafkaParams.put("group.id", groupId);
        kafkaParams.put("auto.offset.reset", "smallest");
```

```
// Create Spark Java streaming context
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));

// Populate data to write to kafka
List<String> sentData = new ArrayList();
sentData.add("kafka_writer_test_msg_01");
sentData.add("kafka_writer_test_msg_02");
sentData.add("kafka_writer_test_msg_03");

// Create Java RDD queue
Queue<JavaRDD<String>> sent = new LinkedList();
sent.add(ssc.sparkContext().parallelize(sentData));

// Create java Dstream with the data to be written
JavaDStream wStream = ssc.queueStream(sent);

// Write to kafka
JavaStreamKafkaWriterFactory.fromJavaDStream(wStream).writeToKafka(
    JavaConverters.mapAsScalaMapConverter(kafkaParams).asScala(),
    new Function<String, ProducerRecord<String, byte[]>>() {
        public ProducerRecord<String, byte[]> call(String s) throws Exception {
            return new ProducerRecord(topic, s.toString().getBytes());
        }
    });
}

ssc.start();
ssc.awaitTermination();
}
```

### 2.17.3.7.3 Scala Example Code

## Function

In Spark applications, use Streaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word, and write the data to Kafka 0-10.

### Code Sample (Streaming Read Data from Kafka 0-10)

The following code is an example. For details, see  
[com.huawei.bigdata.spark.examples.KafkaWordCount](#).

```
/*
 * Consumes messages from one or more topics in Kafka.
 * <checkPointDir> is the Spark Streaming checkpoint directory.
 * <brokers> is for bootstrapping and the producer will only use it for getting metadata
 * <topics> is a list of one or more kafka topics to consume from
 * <batchTime> is the Spark Streaming batch duration in seconds.
 */
public class KafkaWordCount
{
    public static void main(String[] args) {
        JavaStreamingContext ssc = createContext(args);
        //The Streaming system starts.
        ssc.start();
        try {
            ssc.awaitTermination();
        } catch (InterruptedException e) {
        }
    }

    private static JavaStreamingContext createContext(String[] args) {
        String checkPointDir = args[0];
        String brokers = args[1];
        String topics = args[2];
        String batchSize = args[3];
    }
}
```

```
// Create a Streaming startup environment.
SparkConf sparkConf = new SparkConf().setAppName("KafkaWordCount");
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, new
Duration(Long.parseLong(batchSize) * 1000));
//Configure the CheckPoint directory for the Streaming.
//This parameter is mandatory because of existence of the window concept.
? ssc.checkpoint(checkPointDir);
// Get the list of topic used by kafka
String[] topicArr = topics.split(",");
Set<String> topicSet = new HashSet<String>(Arrays.asList(topicArr));
Map<String, Object> kafkaParams = new HashMap();
kafkaParams.put("bootstrap.servers", brokers);
kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
kafkaParams.put("group.id", "DemoConsumer");
LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();
ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicSet, kafkaParams);
// Create direct kafka stream with brokers and topics
// Receive data from the Kafka and generate the corresponding DStream
JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(ssc,
locationStrategy, consumerStrategy);
// Obtain field properties in each row.
JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {
    @Override
    public String call(ConsumerRecord<String, String> tuple2) throws Exception {
        return tuple2.value();
    }
});
// Aggregate the total time that calculate word count
JavaPairDStream<String, Integer> wordCounts = lines.mapToPair(
    new PairFunction<String, String, Integer>() {
?
    @Override
    public Tuple2<String, Integer> call(String s) {
        return new Tuple2<String, Integer>(s, 1);
    }
}).reduceByKey(new Function2<Integer, Integer, Integer>() {
    @Override
    public Integer call(Integer i1, Integer i2) {
        return i1 + i2;
    }
}).updateStateByKey(
    new Function2<List<Integer>, Optional<Integer>, Optional<Integer>>() {
        @Override
        public Optional<Integer> call(List<Integer> values, Optional<Integer> state) {
            int out = 0;
            if (state.isPresent()) {
                out += state.get();
            }
            for (Integer v : values) {
                out += v;
            }
            return Optional.of(out);
        }
    });
// print the results
wordCounts.print();
return ssc;
}
}
```

## Example Code (Streaming Write To Kafka 0-10)

The following code segment is only an example. For details, see com.huawei.bigdata.spark.examples.DstreamKafkaWriter.

 NOTE

After updates to Spark, it is advisable to use the new API `createDirectStream` instead of the old API `createStream` for application development. The old API continues to exist, but its performance and stability are worse than the new API.

```
package com.huawei.bigdata.spark.examples

import scala.collection.mutable
import scala.language.postfixOps

import com.huawei.spark.streaming.kafka010.KafkaWriter_
import org.apache.kafka.clients.producer.ProducerRecord
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.{Milliseconds, StreamingContext}

/**
 * Example code to demonstrate the usage of dstream.writeToKafka API
 *
 * Parameter description:
 * <groupId> is the group ID for the consumer.
 * <brokers> is for bootstrapping and the producer will only use
 * <topic> is a kafka topic to consume from.
 */
object DstreamKafkaWriter {
    def main(args: Array[String]) {

        if (args.length != 3) {
            System.err.println("Usage: DstreamKafkaWriter <groupId> <brokers> <topic>")
            System.exit(1)
        }

        val Array(groupId, brokers, topic) = args
        val sparkConf = new SparkConf().setAppName("KafkaWriter")

        // Populate Kafka properties
        val kafkaParams = Map[String, String](
            "bootstrap.servers" -> brokers,
            "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
            "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
            "value.serializer" -> "org.apache.kafka.common.serialization.ByteArraySerializer",
            "key.serializer" -> "org.apache.kafka.common.serialization.StringSerializer",
            "group.id" -> groupId,
            "auto.offset.reset" -> "latest"
        )

        // Create Spark streaming context
        val ssc = new StreamingContext(sparkConf, Milliseconds(500));

        // Populate data to write to kafka
        val sendData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02",
                           "kafka_writer_test_msg_03")

        // Create RDD queue
        val sent = new mutable.Queue[RDD[String]]()
        sent.enqueue(ssc.sparkContext.makeRDD(sendData))

        // Create Dstream with the data to be written
        val wStream = ssc.queueStream(sent)

        // Write to kafka
        wStream.writeToKafka(kafkaParams,
            (x: String) => new ProducerRecord[String, Array[Byte]](topic, x.getBytes))

        ssc.start()
        ssc.awaitTermination()
    }
}
```

## 2.17.3.8 Structured Streaming Project

### 2.17.3.8.1 Overview

#### Scenarios

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Data Planning

Sample project data of StructuredStreaming is stored in Kafka. A user with Kafka permission sends data to Kafka.

1. Ensure that the cluster, including HDFS, Yarn, Spark, and Kafka is successfully installed.
2. Create a topic.

{IP address of the Kafka cluster} indicates the service IP address of the Broker service.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --bootstrap-server <IP address of the Kafka cluster>:21005 --command-config $KAFKA_HOME/config/client.properties --replication-factor 1 --partitions 1 --topic {Topic}
```

3. Start the Producer of Kafka and send data to Kafka.

{ClassPath} indicates the storage path of engineer JAR packages and is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*: {ClassPath} com.huawei.bigdata.spark.examples.KafkaWordCountProducer {BrokerList} {Topic} {messagesPerSec} {wordsPerMessage}
```

#### Development Approach

1. Receive data from Kafka and generate DataStreamReader.
2. Collect the statistics of word records.
3. Calculate and print the result.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, /opt) on the server where the Spark client is located.

#### Running Tasks

When running the sample application, you need to specify <brokers>, <subscribe-type>, <topic>, and <checkpointDir>.

- <brokers> indicates the Kafka address for obtaining metadata.

- <subscribe-type> indicates the Kafka subscription type (for example, `subscribe`),
- <topic> indicates the name of the topic read from Kafka.
- <checkpointDir> indicates the path for storing the `checkpoint` file, which can be a local path or an HDFS path.

 NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is `$SPARK_HOME/jars`. Whereas the path of the Spark Structured Streaming Kafka dependency package is `$SPARK_HOME/jars/streamingClient010`. Therefore, when running an application, you need to add a configuration item to the `spark-submit` command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars ${files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,; echo "${files[*]}"}`

 CAUTION

When submitting a structured stream task, you need to run the `--jars` command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clientsjar` file from the `$SPARK_HOME/jars/streamingClient010` directory to the `$SPARK_HOME/jars` directory. Otherwise, the "class not found" error is reported.

Go to the Spark client directory and run the following commands to invoke the `bin/spark-submit` script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example).

- Run Java or Scala sample code:

```
bin/spark-submit --master yarn --deploy-mode client --deploy-mode
client --conf spark.driver.userClassPathFirst=true --conf
spark.executor.userClassPathFirst=true --jars ${files=($SPARK_HOME/jars/
streamingClient010/*.jar); IFS=,; echo "${files[*]}"} --class
com.huawei.bigdata.spark.examples.KafkaWordCount /opt/
SparkStructuredStreamingScalaExample-1.0.jar <brokers> <subscribe-type>
<topic> <checkpointDir>
```

The configuration example is as follows:

*If an error indicating that the user does not have the permission to read and write the local directory is reported, the `spark.sql.streaming.checkpointLocation` parameter must be specified, and the user must have the read and write permissions on the directory specified by this parameter.*

- Run the Python sample code:

 NOTE

When running the Python sample code, you need to add the JAR package of the Java project to the `streamingClient/` directory.

```
bin/spark-submit --master yarn --deploy-mode client --deploy-mode
client --conf spark.driver.userClassPathFirst=true --conf
spark.executor.userClassPathFirst=true --jars ${files=($SPARK_HOME/jars/
streamingClient010/*.jar); IFS=,; echo "${files[*]}"} /opt/female/
SparkStructuredStreamingPythonExample/KafkaWordCount.py <brokers>
<subscribe-type> <topic> <checkpointDir>
```

### 2.17.3.8.2 Java Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see [com.huawei.bigdata.spark.examples.KafkaWordCount](#).

##### NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receptor. The default value is **append**. To change the value, see the **outputMode** description of **Spark > Scala** in *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

```
public class KafkaWordCount
{
    public static void main(String[] args) throws Exception {
        if (args.length < 3) {
            System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +
                "<subscribe-type> <topics>");
            System.exit(1);
        }

        String bootstrapServers = args[0];
        String subscribeType = args[1];
        String topics = args[2];

        SparkSession spark = SparkSession
            .builder()
            .appName("KafkaWordCount")
            .getOrCreate();

        // Create DataSet representing the stream of input lines from kafka
        Dataset<String> lines = spark
            .readStream()
            .format("kafka")
            .option("kafka.bootstrap.servers", bootstrapServers)
            .option(subscribeType, topics)
            .load()
            .selectExpr("CAST(value AS STRING)")
            .as(Encoders.STRING());

        // Generate running word count
        Dataset<Row> wordCounts = lines.flatMap(new FlatMapFunction<String, String>() {
            @Override
            public Iterator<String> call(String x) {
                return Arrays.asList(x.split(" ")).iterator();
            }
        }, Encoders.STRING()).groupByKey("value").count();

        // Start running the query that prints the running counts to the console
        StreamingQuery query = wordCounts.writeStream()
            .outputMode("complete")
            .format("console")
            .start();

        query.awaitTermination();
    }
}
```

### 2.17.3.8.3 Scala Sample Code

#### Function

In Spark applications, use StructuredStreaming to invoke Kafka interface to obtain word records. Collect the statistics of records for each word.

#### Code Sample

The following code is an example. For details, see [com.huawei.bigdata.spark.examples.KafkaWordCount](#).

##### NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receptor. The default value is **append**. To change the value, see the **outputMode** description of **Spark > Scala** in *MapReduce Service (MRS) 3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

```
object KafkaWordCount {  
    def main(args: Array[String]): Unit = {  
        if (args.length < 3) {  
            System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +  
                "<subscribe-type> <topics>")  
            System.exit(1)  
        }  
  
        val Array(bootstrapServers, subscribeType, topics) = args  
  
        val spark = SparkSession  
            .builder  
            .appName("KafkaWordCount")  
            .getOrCreate()  
  
        import spark.implicits._  
  
        // Create DataSet representing the stream of input lines from kafka  
        val lines = spark  
            .readStream  
            .format("kafka")  
            .option("kafka.bootstrap.servers", bootstrapServers)  
            .option(subscribeType, topics)  
            .load()  
            .selectExpr("CAST(value AS STRING)")  
            .as[String]  
  
        // Generate running word count  
        val wordCounts = lines.flatMap(_.split(" ")).groupBy("value").count()  
  
        // Start running the query that prints the running counts to the console  
        val query = wordCounts.writeStream  
            .outputMode("complete")  
            .format("console")  
            .start()  
  
        query.awaitTermination()  
    }  
}
```

#### 2.17.3.8.4 Python Sample Code

### Function

In Spark applications, use StructuredStreaming to invoke Kafka APIs to obtain word records. Classify word records to obtain the number of records of each word.

### Sample Code

The following code segment is only an example. For details, see [SecurityKafkaWordCount](#).

#### NOTE

When there is new available data in Streaming DataFrame/Dataset, **outputMode** indicates data written to the Streaming receiver. The default value is **append**. To change the value, see the **outputMode** description of **Spark > Scala** in *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split

if __name__ == "__main__":
    if len(sys.argv) < 3:
        print("Usage: <bootstrapServers> <subscribeType> <topics>")
        exit(-1)

    bootstrapServers = sys.argv[1]
    subscribeType = sys.argv[2]
    topics = sys.argv[3]

    # Initialize the SparkSession.
    spark = SparkSession.builder.appName("KafkaWordCount").getOrCreate()

    # Create the DataFrame of input lines stream from Kafka.
    # In security mode, set spark/conf/jaas.conf and jaas-zk.conf to KafkaClient.
    lines = spark.readStream.format("kafka")\
        .option("kafka.bootstrap.servers", bootstrapServers)\.
        .option(subscribeType, topics)\.
        .load()\.
        .selectExpr("CAST(value AS STRING)")

    # Split lines into words.
    words = lines.select(explode(split(lines.value, " ")).alias("word"))
    # Generate the running word count.
    wordCounts = words.groupBy("word").count()

    # Start to query whether the running counts are printed to the console.
    query = wordCounts.writeStream\
        .outputMode("complete")\
        .format("console")\
        .start()

    query.awaitTermination()
```

#### 2.17.3.9 Structured Streaming Stream-Stream Join

### 2.17.3.9.1 Overview

#### Scenarios

An advertisement service involves advertisement request events, advertisement display events, and advertisement click events. The advertiser needs to collect statistics on valid advertisement displays and advertisement click data.

Known conditions are as follows:

1. Each time a user requests an advertisement, an advertisement request event is generated and saved to the adRequest topic of Kafka.
2. After an advertisement is requested, the advertisement may be displayed for multiple times. Each time the advertisement is displayed, an advertisement display event is generated and saved to the adShow topic of Kafka.
3. Each advertisement may be clicked for multiple times. Each time it is clicked, an advertisement click event is generated and saved to the adClick topic of Kafka.
4. For advertisement display:
  - a. If the duration from the time when a request is generated to the time when the advertisement is displayed exceeds A minutes, the display is invalid.
  - b. Advertisement display events generated during A minutes are valid events.
5. For advertisement click:
  - a. If the duration from the display event to the click event exceeds B minutes, the click is invalid.
  - b. If there are multiple click events within B minutes, only the first click event is valid.

The simple data structure in this scenario is as follows:

- Advertisement request event  
Data structure: adID^reqTime
- Advertisement display event  
Data structure: adID^showID^showTime
- Advertisement click event  
Data structure: adID^showID^clickTime

Data association relationships are as follows:

- The advertisement request event is associated with the advertisement display event through the adID.
- The advertisement display event is associated with the advertisement click event through the adID and showID.

Data requirements:

- The delay from the time when data is generated to the time when the data reaches the stream processing engine does not exceed two hours.

- The time for advertisement request events, advertisement display events, and advertisement click events reach the stream processing engine may not be in sequence or be aligned with each other.

## Data Planning

1. Enable users with the Kafka permission to generate simulation data in Kafka.

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/
streamingClient010/*:{ClassPath}
com.huawei.bigdata.spark.examples.KafkaADEventProducer {BrokerList}
{timeOfProduceReqEvent} {eventTimeBeforeCurrentTime} {reqTopic}
{reqEventCount} {showTopic} {showEventMaxDelay} {clickTopic}
{clickEventMaxDelay}
```

### NOTE

- Ensure that the clusters are installed, including HDFS, Yarn, Spark, and Kafka.
- Start Kafka Producer and enable it to send data to Kafka.
- {ClassPath} indicates the storage path of engineer JAR packages and is specified by the user. For details, see [Compiling and Running the Application](#).

Command example:

```
java -cp /opt/hadoopclient/Spark/spark/conf:/opt/
StructuredStreamingADScalaExample-1.0.jar:/opt/hadoopclient/Spark/
spark/jars/*:/opt/hadoopclient/Spark/spark/jars/streamingClient010/*
com.huawei.bigdata.spark.examples.KafkaADEventProducer
10.132.190.170:21005,10.132.190.165:21005 2h 1h req 10000000 show 5m
click 5m
```

This command creates three topics on Kafka: req, show, and click. Ten million data records of request events are generated in two hours. The time range of the request events is from one hour ahead of the current time to the current time, and up to five display events are randomly generated for each request event. The time range of the display events is from the request event time to five minutes after the request event time. Up to five click events are randomly generated for each display event. The time range of the click events is from the display event time to five minutes after the display event time.

## Development Guideline

1. Use structured streaming to receive data from Kafka and generate request flows, display flows, and click flows.
2. Perform join query of data in request flows, display flows, and click flows.
3. Write the statistics result to Kafka.
4. Monitor the flow processing task status in the application.

## Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file. For details, see [Compiling and Running the Application](#).
- Upload the JAR package to any directory (for example, `/opt`) on the server where the Spark client is located.
- Upload the `user.keytab` and `krb5.conf` files to the server where the client is installed.

## Running Tasks

When running the sample program, you need to specify `<kafkaBootstrapServers>`, `<maxEventDelay>`, `<reqTopic>`, `<showTopic>`, `<maxShowDelay>`, `<clickTopic>`, `<maxClickDelay>`, `<triggerInterver>`, and `<checkpointDir>`.

- `<kafkaBootstrapServers>` indicates the Kafka address for obtaining metadata.
- `<maxEventDelay>` indicates the maximum delay from data generation to stream processing.
- `<reqTopic>` indicates the topic name of the request event.
- `<showTopic>` indicates the topic name of the display event.
- `<maxShowDelay>` indicates the maximum delay for effectively displaying the event.
- `<clickTopic>` indicates the topic name of the click event.
- `<maxClickDelay>` indicates the maximum delay of a valid click event.
- `<triggerInterver>` indicates the interval for triggering a stream processing task.
- `<checkpointDir>` indicates the path for storing the **checkpoint** file, which can be a local path or an HDFS path.

### NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is `$SPARK_HOME/jars`. Whereas the path of the Spark Structured Streaming Kafka dependency package is `$SPARK_HOME/jars/streamingClient010`. Therefore, when running an application, you need to add a configuration item to the `spark-submit` command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars ${files}($SPARK_HOME/jars/streamingClient010/*.jar); IFS=;; echo "${files[*]}")`

### CAUTION

When submitting a structured stream task, you need to run the `--jars` command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clientsjar` file from the `$SPARK_HOME/jars/streamingClient010` directory to the `$SPARK_HOME/jars` directory. Otherwise, the "class not found" error is reported.

Go to the Spark client directory and run the following command to invoke the `bin/spark-submit` script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

```
bin/spark-submit --master yarn --deploy-mode client --jars ${files}($SPARK_HOME/jars/streamingClient010/*.jar); IFS=;; echo "${files[*]}") --conf "spark.sql.streaming.statefulOperator.checkCorrectness.enabled=false" --class com.huawei.bigdata.spark.examples.KafkaADCount /opt/StructuredStreamingADScalaExample-1.0.jar <kafkaBootstrapServers> <maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> <clickTopic> <maxClickDelay> <triggerInterver> <checkpointDir>
```

### 2.17.3.9.2 Scala Example Code

#### Function

Structured Streaming is used to read advertisement request data, display data, and click data from Kafka, obtain effective display statistics and click statistics in real time, and write the statistics to Kafka.

#### Example code

The following code snippets are used as an example. For complete codes, see com.huawei.bigdata.spark.examples.KafkaADCount.

```
/*
 * Run the Structured Streaming task to collect statistics on valid advertisement display and click data. The
result is written into Kafka.
 */

object KafkaADCount {
    def main(args: Array[String]): Unit = {
        if (args.length < 12) {
            System.err.println("Usage: KafkaWordCount <bootstrap-servers> " +
                "<maxEventDelay> <reqTopic> <showTopic> <maxShowDelay> " +
                "<clickTopic> <maxClickDelay> <triggerInterver> " +
                "<checkpointLocation> <protocol> <service> <domain>")
            System.exit(1)
        }

        val Array(bootstrapServers, maxEventDelay, reqTopic, showTopic,
            maxShowDelay, clickTopic, maxClickDelay, triggerInterver, checkpointLocation,
            protocol, service, domain) = args

        val maxEventDelayMills = JavaUtils.timeStringAs(maxEventDelay, TimeUnit.MILLISECONDS)
        val maxShowDelayMills = JavaUtils.timeStringAs(maxShowDelay, TimeUnit.MILLISECONDS)
        val maxClickDelayMills = JavaUtils.timeStringAs(maxClickDelay, TimeUnit.MILLISECONDS)
        val triggerMills = JavaUtils.timeStringAs(triggerInterver, TimeUnit.MILLISECONDS)

        val spark = SparkSession
            .builder
            .appName("KafkaADCount")
            .getOrCreate()

        spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

        import spark.implicits._

        // Create DataSet representing the stream of input lines from kafka
        val reqDf = spark
            .readStream
            .format("kafka")
            .option("kafka.bootstrap.servers", bootstrapServers)
            .option("kafka.security.protocol", protocol)
            .option("kafka.sasl.kerberos.service.name", service)
            .option("kafka.kerberos.domain.name", domain)
            .option("subscribe", reqTopic)
            .load()
            .selectExpr("CAST(value AS STRING)")
            .as[String]
            .map{
                _split('^') match {
                    case Array(reqAdID, reqTime) => ReqEvent(reqAdID,
                        Timestamp.valueOf(reqTime))
                }
            }
            .as[ReqEvent]
            .withWatermark("reqTime", maxEventDelayMills +
```

```
maxShowDelayMills + " millisecond")

val showDf = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option("subscribe", showTopic)
    .load()
    .selectExpr("CAST(value AS STRING)")
    .as[String]
    .map{
        _split('^') match {
            case Array(showAdID, showID, showTime) => ShowEvent(showAdID,
                showID, Timestamp.valueOf(showTime))
        }
    }
    .as[ShowEvent]
    .withWatermark("showTime", maxEventDelayMills +
        maxShowDelayMills + maxClickDelayMills + " millisecond")

val clickDf = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option("subscribe", clickTopic)
    .load()
    .selectExpr("CAST(value AS STRING)")
    .as[String]
    .map{
        _split('^') match {
            case Array(clickAdID, clickShowID, clickTime) => ClickEvent(clickAdID,
                clickShowID, Timestamp.valueOf(clickTime))
        }
    }
    .as[ClickEvent]
    .withWatermark("clickTime", maxEventDelayMills + " millisecond")

val showStaticsQuery = reqDf.join(showDf,
    expr(s"""
        reqAdID = showAdID
        AND showTime >= reqTime + interval ${maxShowDelayMills} millisecond
    """))
    .selectExpr("concat_ws('^', showAdID, showID, showTime) as value")
    .writeStream
    .queryName("showEventStatics")
    .outputMode("append")
    .trigger(Trigger.ProcessingTime(triggerMills.millis))
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option("topic", "showEventStatics")
    .start()

val clickStaticsQuery = showDf.join(clickDf,
    expr(s"""
        showAdID = clickAdID AND
        showID = clickShowID AND
        clickTime >= showTime + interval ${maxClickDelayMills} millisecond
    """), joinType = "rightouter")
    .dropDuplicates("showAdID")
    .selectExpr("concat_ws('^', clickAdID, clickShowID, clickTime) as value")
```

```
.writeStream
.queryName("clickEventStatics")
.outputMode("append")
.trigger(Trigger.ProcessingTime(triggerMills.millis))
.format("kafka")
.option("kafka.bootstrap.servers", bootstrapServers)
.option("kafka.security.protocol", protocol)
.option("kafka.sasl.kerberos.service.name", service)
.option("kafka.kerberos.domain.name", domain)
.option("topic", "clickEventStatics")
.start()

new Thread(new Runnable {
    override def run(): Unit = {
        while(true) {
            println("-----get showStatic progress-----")
            //println(showStaticsQuery.lastProgress)
            println(showStaticsQuery.status)
            println("-----get clickStatic progress-----")
            //println(clickStaticsQuery.lastProgress)
            println(clickStaticsQuery.status)
            Thread.sleep(10000)
        }
    }
}).start

spark.streams.awaitAnyTermination()
}
```

## 2.17.3.10 Spark on Elasticsearch

### 2.17.3.10.1 Overview

#### Scenarios

In Spark, you can create, delete, and query indexes using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.

#### Data Planning

Perform the following operations to save the data files in HDFS:

1. Obtain a test data file and obtain the **people.json** file in the **data** directory of the sample code.
2. Create the **spark-on-es** folder in the HDFS and upload the **people.json** file to the directory **/spark-on-es** of the HDFS. The command is as follows:
  - a. On the HDFS client, run the following commands to obtain security authentication:  
**cd /opt/hadoopclient**  
**kinit <Service user for authentication>**
  - b. On a Linux-based HDFS client, run the **hadoop fs -mkdir /user/spark-on-es** command (the **hdfs dfs** command provides the same function).
  - c. On the Linux HDFS client, run the **hadoop fs -put people.json /user/spark-on-es** command to upload the data file.

## Development Idea

1. Initialize basic configuration parameters of Elasticsearch.
2. Create an Elasticsearch client.
3. Create indexes of Elasticsearch.
4. Initialize the SparkContext.
5. Use the **BulkRequest** command to add data in batches in Elasticsearch.
6. Spark reads the test data (the **people.json** file obtained in **Data Planning**) from HDFS and adds the data to the Elasticsearch indexes created in 3 using the elasticsearch-spark API.
7. Query data in the Elasticsearch indexes. Two query methods displayed in the sample are as follows:
  - Query data using the elasticsearch-spark API.
  - Query data using the Elasticsearch Scroll API.
8. Operate the queried data by using the Spark Application. The sample demonstrates how to group and sort the queried data.

## Configuration Files

The content of the **esParams.properties** configuration file is as follows:

```
# IP address and port number of the node in the Elasticsearch cluster. Enter a value in the format of "IP
address:port number". Separate multiple values with commas (,).
esServerHost=10.10.10.11:24100,10.10.10.12:24100,10.10.10.13:24100
# Whether to enable the sniffer function.
snifferEnable=true
connectTimeout=5000
socketTimeout=60000
connectionRequestTimeout=3000
# Maximum number of connections on a single route.
maxConnPerRoute=100
maxConnTotal=1000
maxRetryTimeoutMillis=60000

# Whether the cluster is in security mode. If yes, set the parameter to true; if no, set it to false.
isSecureMode=false
# Whether to enable the SSL mode of Elasticsearch. If the cluster is in security mode, set the parameter to
true; otherwise, set it to false.
sslEnabled=false

# Field to be returned. Separate multiple values with commas (,).
# For example: https://www.elastic.co/guide/en/elasticsearch/hadoop/current/configuration.html
esFilterField=name,age,createdTime
# The maximum of Elasticsearch batch query is 10,000 by default.
# It is recommended that the value be less than 50,000. Otherwise, out of memory (OOM) may occur.
esScrollSize=10000
# This parameter is used to set the number of Spark data partitions. It is supported by Elasticsearch 5.0 and
later.
# This parameter, together with scroll.size, can improve the throughput and task concurrency.
# If it is not set, the number of Spark partitions is equal to that of index shards in Elasticsearch.
esInputMaxDocsPerPartition=500000

# The name of the index to be operated in Elasticsearch, the number of shards, and the number of replicas.
esIndex=people
esShardNum=5
esReplicaNum=1

# Information required for group statistics and range query in Elasticsearch.
esGroupField=age
esQueryField=createdTime
```

```
# Start value of the range query, which is included in the range.  
esQueryRangeBegin=2010-01-01T00:00:00Z  
# End value of the range query, which is not included in the range.  
esQueryRangeEnd=2015-12-31T23:59:59Z  
# JSON character string of Elasticsearch DSL, which cannot contain line breaks.  
# Only the query parameter is supported. Other parameters such as _source and size are not supported.  
esQueryJsonString={"query": {"range": { "%s": {"gte": "%s", "lt": "%s"} } } }
```

## Preparation Operations Before Running

In non-security mode, the **esParams.properties** file needs to be read in the SparkOnEs sample code. For details about the **esParams.properties** file, see the example in the conf directory of the sample code.

For the Python sample, you need to modify the Python code and set **es.nodes** in **SparkOnEsPythonExample.py** to the IP address of the node where EsNode resides in the cluster.

## Packaging the Project

1. Modify the project configuration to make it the same as that of the cluster to be tested (You can find the Python sample configuration in the **SparkOnEsPythonExample.py** file.).
2. Use the Maven tool provided by IDEA to package the project and generate **target\SparkOnEs-1.0.jar**. For details, see [Compiling and Running the Application](#).
3. Upload the generated JAR package to any directory (for example, **/opt/spark-on-es/**) on the server where the Spark client is located.
4. Upload the **esParams.properties**, **user.keytab**, and **krb5.conf** files to the directory in [3](#).
5. Prepare the dependency package. Upload the following JAR package to the **/opt/spark-on-es/libs/** directory on the server where the Spark client is located.

The following JAR packages are required for running the sample code:

- elasticsearch-xxx.cbu.mrs.xxx.jar
- elasticsearch-core-xxx.cbu.mrs.xxx.jar
- elasticsearch-rest-client-xxx.cbu.mrs.xxx.jar
- elasticsearch-rest-high-level-client-xxx.cbu.mrs.xxx.jar
- elasticsearch-spark-20\_2.12-7.12.0.jar
- elasticsearch-x-content-xxx.cbu.mrs.xxx.jar
- lang-mustache-client-xxx.cbu.mrs.xxx.jar
- log4j-api-2.12.0.jar
- log4j-core-2.12.0.jar
- lucene-core-8.7.0.jar
- rank-eval-client-xxx.cbu.mrs.xxx.jar
- commons-httpclient-\*jar

## Running a Task

1. In client mode (The class name and file name must be the same as those in the actual code. The following is only an example):

```
cd /opt/spark-on-es
spark-submit --class com.huawei.bigdata.spark.examples.SparkOnEs --
master yarn --deploy-mode client --conf
spark.driver.userClassPathFirst=true --jars ${files=(/opt/spark-on-es/libs/
*.jar);IFS=,; echo "${files[*]}"} ./SparkOnEs-1.0.jar
```

In the preceding command, `/opt/spark-on-es/libs/` indicates the path of the external dependency JAR package.

2. In cluster mode (The class name and file name must be the same as those in the actual code. The following is only an example):

```
cd /opt/spark-on-es
spark-submit --class com.huawei.bigdata.spark.examples.SparkOnEs --
master yarn --deploy-mode cluster --conf
spark.driver.userClassPathFirst=true --jars ${files=(/opt/spark-on-es/libs/
*.jar);IFS=,; echo "${files[*]}"} --files ./esParams.properties ./
SparkOnEs-1.0.jar
```

In the preceding command, the `--files` parameter specifies the configuration files required for program running. Use commas (,) to separate multiple files.

## Querying Results

1. Query data in Elasticsearch

```
# In a cluster in common mode, use HTTP (not HTTPS) for query.
curl -XGET 'http://10.10.10.11:24100/_cat/indices?v'
```

```
# Run the following commands to query the range of data in the people index.
curl -XPOST 'http://10.10.10.11:24100/people/_search?pretty' -H 'Content-Type:application/json' -d '
{
    "query": {
        "range": {
            "createdTime": {"gte": "2010-01-01T00:00:00Z", "lt": "2015-12-31T23:59:59Z"}
        }
    }
}'
```

2. Query files in HDFS.

```
# View the grouping result file saved in HDFS.
hdfs dfs -ls /user/spark-on-es/group-result
```

```
# View the content of a file. For example, view the content of file part-00001.
hdfs dfs -cat /user/spark-on-es/group-result/part-00001
```

### 2.17.3.10.2 Java Sample Code

#### Function

In Spark, you can read, write, and collect statistics on the data in Elasticsearch using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.

#### Sample Code

The following code snippets are used as an example. For complete codes, see the following `SparkOnEsJavaExample` project:

```
public static void main(String[] args) {
    LOG.info("***** Start to run the Spark on ES test.");
    try {
        // init the global properties
```

```
initProperties();

String path = System.getProperty("user.dir") + File.separator;
HwRestClient hwRestClient = new HwRestClient(path);
restClient = hwRestClient.getRestClient();
highLevelClient = new RestHighLevelClient((hwRestClient.getRestClientBuilder()));

// Check whether the index to be added has already exist, remove the exist one
if (exist(esIndex)) {
    deleteIndex(esIndex);
}
createIndex(esIndex);

SparkOnEs instance = new SparkOnEs();
// Put data by ES bulk request
instance.putDataByBulk(highLevelClient);

// Create a configuration class SparkConf,
// meanwhile set the Secure configuration that the Elasticsearch Cluster needed,
// finally create a SparkContext.
SparkConf conf =
    new SparkConf()
        .setAppName("SparkOnEs")
        .set("es.nodes", esServerHost)
        // when you specified in es.nodes, then es.port is not necessary
        // .set("es.port", "24100")
        .set("es.nodes.discovery", "true")
        .set("es.index.auto.create", "true")
        .set("es.internal.spark.sql.pushdown", "true")
        .set("es.read.source.filter", esFilterField)
        .set("es.scroll.size", esScrollSize)
        .set("es.input.max.docs.per.partition", esInputMaxDocsPerPartition);

JavaSparkContext jsc = new JavaSparkContext(conf);

instance.putHdfsData(jsc); // Put data from HDFS into ES
instance.queryDataByDataset(conf); // Query data from ES
instance.queryDataByRestClient(highLevelClient, jsc);
instance.groupBySpark(jsc); // Group data from ES

jsc.stop();
} catch (IOException e) {
    LOG.error("***** There are exceptions in main.", e);
} finally {
    closeResources();
}
}
```

### 2.17.3.10.3 Scala Sample Code

#### Function

In Spark, you can read, write, and collect statistics on the data in Elasticsearch using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.

#### Sample Code

The following code snippets are used as an example. For complete codes, see the following `SparkOnEsScalaExample` project:

```
object SparkOnEs {
  def main(args: Array[String]): Unit = {
    val sparkOnEs = new SparkOnEs
    sparkOnEs.LOG.info("***** Start to run the Spark on ES test.")

    try {
```

```
// init the global properties
sparkOnEs.initProperties()

val path = System.getProperty("user.dir") + File.separator
val hwRestClient: HwRestClient = new HwRestClient(path)
sparkOnEs.restClient = hwRestClient.getRestClient
sparkOnEs.highLevelClient = new RestHighLevelClient(hwRestClient.getRestClientBuilder)

// Check whether the index to be added has already exist, remove the exist one
if (sparkOnEs.exist(sparkOnEs.esIndex)) sparkOnEs.deleteIndex(sparkOnEs.esIndex)
sparkOnEs.createIndex(sparkOnEs.esIndex)

// Put data by ES bulk request
sparkOnEs.putDataByBulk(sparkOnEs.highLevelClient)

// Create a configuration class SparkConf,
// meanwhile set the Secure configuration that the Elasticsearch Cluster needed,
// finally create a SparkContext.
val conf: SparkConf = new SparkConf().setAppName("SparkOnEs")
  .set("es.nodes", sparkOnEs.esServerHost)
  // when you specified in es.nodes, then es.port is not necessary
  // .set("es.port", "24100")
  .set("es.nodes.discovery", "true")
  .set("es.index.auto.create", "true")
  .set("es.internal.spark.sql.pushdown", "true")
  .set("es.read.source.filter", sparkOnEs.esFilterField)
  .set("es.scroll.size", sparkOnEs.esScrollSize)
  .set("es.input.max.docs.per.partition", sparkOnEs.esInputMaxDocsPerPartition)

val jsc: JavaSparkContext = new JavaSparkContext(conf)

sparkOnEs.putHdfsData(jsc) // Put data from HDFS into ES
sparkOnEs.queryDataByDataset(conf) // Query data from ES
sparkOnEs.queryDataByRestClient(sparkOnEs.highLevelClient, jsc)
sparkOnEs.groupBySpark(jsc) // Group data from ES

jsc.close()
} catch {
  case e: IOException =>
    sparkOnEs.LOG.error("***** There are exceptions in main.", e)
} finally {
  sparkOnEs.closeResources()
}
```

### 2.17.3.10.4 Python Sample Code

#### Function

In Spark, you can read, write, and collect statistics on the data in Elasticsearch using Elasticsearch native APIs and the APIs provided by elasticsearch-spark.

#### Sample Code

The following code snippets are used as an example. For complete codes, see the following SparkOnEsPythonExample project.

```
# -*- coding:utf-8 -*-
"""
[Note] Since PySpark doesn't support the Elasticsearch, this example calls Java code through Python.
"""

from py4j.java_gateway import java_import
from pyspark import SparkConf, SparkContext

# create the SparkConf
```

```
conf = SparkConf().setAppName("SparkOnEs")

# the configurations for Elasticsearch cluster
conf.set("es.nodes", "10.10.10.11:24100,10.10.10.12:24100,10.10.10.13:24100")
# when you specified in es.nodes, then es.port is not necessary
# conf.set("es.port", "24100")
conf.set("es.nodes.discovery", "true")
conf.set("es.index.auto.create", "true")
conf.set("es.internal.spark.sql.pushdown", "true")
conf.set("es.read.source.filter", "name,age,createdTime")
conf.set("es.scroll.size", "10000")
conf.set("es.input.max.docs.per.partition", "500000")

# create the SparkContext
sc = SparkContext(conf = conf)

# import the program's class to sc._jvm
java_import(sc._jvm, "com.huawei.bigdata.spark.examples.SparkOnEs")

# create an instance of the above class, then invoke its method to run the program
sc._jvm.SparkOnEs().main(sc._jsc)

# stop the SparkContext
sc.stop()
```

## Running a Task

PySpark does not provide APIs for operating Elasticsearch. In this sample code, Python is used to invoke Java for task running. When running the task, you need to add the JAR package of the Java project to the **libs/** directory and upload the JAR package and other dependency packages to the class path.

1. In client mode:

```
cd /opt/spark-on-es
spark-submit --master yarn --deploy-mode client --conf
spark.driver.userClassPathFirst=true --jars ${files=(/opt/spark-on-es/libs/
*.jar);IFS=,; echo "${files[*]}"} SparkOnEsPythonExample.py
```

In the preceding command, `/opt/spark-on-es/libs/` indicates the path of the external dependency JAR package, **including the sample JAR package**.

2. In cluster mode:

```
cd /opt/spark-on-es
spark-submit --master yarn --deploy-mode cluster --conf
spark.driver.userClassPathFirst=true --jars ${files=(/opt/spark-on-es/libs/
*.jar);IFS=,; echo "${files[*]}"} --files ./esParams
SparkOnEsPythonExample.py
```

In the preceding command, the `--files` parameter specifies the configuration files required for program running. Use commas (,) to separate multiple files.

### 2.17.3.11 Spark on Solr

#### 2.17.3.11.1 Overview

##### Scenarios

Users can use Spark to invoke Solr APIs to operate data in Solr. In the Spark application, users can use the Solr APIs to create, query, and delete indexes.

## Development Idea

1. Initialize the Solr configuration and obtain a Solr client.
2. Create Solr indexes.
3. Add the data to the indexes.
4. Read the data in the indexes and use Spark to perform simple operations.
5. Delete the created indexes.
6. Read the data in the indexes again.

## Dependency

- solr-solrj-xxx.jar
- junit-xxx.jar
- spark-core\_2.12-xxx.jar
- hadoop-common-xxx.jar
- hadoop-mapreduce-client-core-xxx.jar
- slf4j-log4j1xxx.jar
- commons-logging-xxx.jar
- commons-langxxx.jar
- jackson-databind-xxx.jar
- jetty-client-xxx.jar
- httpmime-xxx.jar

## Configuration Files

```
solr-example.properties

#ZooKeeper address
zkHost=10.111.111.111:24002/solr
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL=zookeeper/hadoop.<System domain name>
#User to be authenticated, which can be explicitly specified in the code. (In the sample code, the user is
explicitly specified as super.)
principal=super
zkClientTimeout=20000
zkConnectTimeout=30000
# Whether it is in security mode. If it is in security mode, set this parameter to true, and set to false if it is
in non-security mode.
SOLR_KBS_ENABLED=false
COLLECTION_NAME=col_test
DEFAULT_CONFIG_NAME=confWithSchema
#Set this parameter based on the number of instance nodes. You are advised to set this parameter to the
number of instance nodes.
shardNum=3
replicaNum=1
maxShardsPerNode=1
autoAddReplicas=false
assignToSpecifiedNodeSet=false
#Solr service SolrServer instance IP
createNodeSet=10.111.111.111:21101_solr,10.222.222.222:21101_solr
```

## Configuration Operations Before Running

The **solr-example.properties** configuration file of Solr is needed in the SparkOnSolr sample code. For details about the **solr-example.properties** file, see the example in the **conf** directory of the sample code.

### Yarn-client mode

1. Copy the **solr-example.properties** file to the **\$SPARK\_HOME/bin** running directory on the client.
2. Extra JAR packages are required for running Solr. Therefore, you need to add the JAR package that is required by the Spark application and not included by the cluster to the class path of Driver. Modify **spark.driver.extraClassPath** in the **spark-defaults.conf** configuration file on the client. The configuration method is as follows:

```
spark.driver.extraClassPath = /opt/hadoopclient/Spark/customJars/*
```



The JAR packages to be added to the sample code include **httpmime-xxx.jar** and **jetty-client-xxx.jar** (xxx indicates the version number of the JAR package, which can be obtained from the  **\${SOLR\_HOME}/lib/** directory.). Add or delete as required.

3. Run the following command. (The class name and file name must be the same as those in the actual code. The following is only an example):

```
./spark-submit --class com.huawei.bigdata.spark.examples.SparkOnSolr --jars /opt/hadoopclient/Spark/customJars/jetty-client-xxx.jar,/opt/hadoopclient/Spark/customJars/httpmime-xxx.jar --master yarn --deploy-mode client ./SparkOnSolr.jar
```

### Yarn-cluster mode

1. Copy the **es-example.properties** file to any directory on the node on which the client is located.
2. Extra JAR packages are required for running Solr. Therefore, you need to add the JAR package required by the Spark application to the cluster and modify **spark.yarn.dist.innerfiles** and **spark.yarn.cluster.driver.extraClassPath** in the **spark-defaults.conf** file on the client. The configuration method is as follows:

- Distribute the extra JAR packages to each node of the cluster by using the configuration item.

```
spark.yarn.dist.files = /opt/hadoopclient/Spark/customJars/httpmime-xxx.jar,/opt/hadoopclient/Spark/customJars/jetty-client-xxx.jar
```

- Add the specified JAR package to the class path of Driver in Yarn-cluster mode by using the configuration item.

```
spark.yarn.cluster.driver.extraClassPath = $PWD/httpmime-xxx.jar:$PWD/jetty-client-xxx.jar
```

3. When running the Spark program, add the **--file** parameter and path of the **solr-example.properties** file to the command line. For example:

```
./spark-submit --class com.huawei.bigdata.spark.examples.SparkOnSolr --files ./solr-example.properties --master yarn --deploy-mode cluster ./SparkOnSolr.jar
```

## 2.17.3.11.2 Java Sample Code

### Function

In the Spark application, Solr client APIs are used to create indexes, insert data, and search for data.

## Sample Code

The following code snippets are used as an example. For complete codes, see the following SparkOnSolrJavaExample project:

```
/*
 * 1-Initialize Solr configuration and obtain the Solr client;
 * 2>Create Solr indexes;
 * 3-Add data to the indexes;
 * 4-Read the data in the indexes and use Spark to perform simple operations;
 * 5>Delete the created indexes;
 * 6-Read the data in the indexes again.
 */

public class SparkOnSolr{
    public static void main(String[] args) throws Exception {
        SparkOnSolr testSample = new SparkOnSolr();
        testSample.initProperties();
        SparkConf conf = new SparkConf().setAppName("SparkOnSolr");
        JavaSparkContext jsc = new JavaSparkContext(conf);
        CloudSolrClient cloudSolrClient = null;
        try {
            cloudSolrClient = testSample.getCloudSolrClient(zkHost);
            List<String> collectionNames = testSample.queryAllCollections(cloudSolrClient);
            //delete the collection if exists
            if (collectionNames.contains(collectionName)) {
                testSample.deleteCollection(cloudSolrClient);
            }
            testSample.createCollection(cloudSolrClient);
            cloudSolrClient.setDefaultCollection(collectionName);
            testSample.addDocs(cloudSolrClient);
            testSample.addDocs2(cloudSolrClient);
            //Some time is needed to build the index
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
            }
            List<Map<String, Object>> queryResult1 = testSample.queryIndex(cloudSolrClient);
            List<List<String>> resultList = new ArrayList<>();
            for(int i=0; i < queryResult1.size(); i++){
                resultList.add(testSample.transferMapToList(queryResult1.get(i)));
            }
            JavaRDD<List<String>> resultRDD = jsc.parallelize(resultList);
            System.out.println("Before delete, the total count is:" + resultRDD.count() + "\n");
            testSample.removeIndex(cloudSolrClient);
            //Some time is needed to delete the index
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
            }
            List<Map<String, Object>> queryResult2 = testSample.queryIndex(cloudSolrClient);
            JavaRDD<Map<String, Object>> resultRDD2 = jsc.parallelize(queryResult2);
            System.out.println("After delete, the total count is:" + resultRDD2.count() + "\n");
        } catch (SolrException e) {
            throw new SolrException(e.getMessage());
        } finally {
            if (cloudSolrClient != null) {
                try {
                    cloudSolrClient.close();
                } catch (IOException e) {
                    LOG.warn("Failed to close cloudSolrClient", e);
                }
            }
            jsc.stop();
        }
    }
}
```

### 2.17.3.11.3 Scala Sample Code

#### Function

In the Spark application, Solr client APIs are used to create indexes, insert data, and search for data. Finally, Spark performs simple operations on the queried data.

#### Sample Code

The following code snippets are used as an example. For complete codes, see the following `SparkOnSolrScalaExample` project:

```
/*
* 1-Initialize Solr configuration and obtain the Solr client;
* 2-Create Solr indexes;
* 3-Add data to the indexes;
* 4-Read the data in the indexes and use Spark to perform simple operations;
* 5>Delete the created indexes;
* 6-Read the data in the indexes again.
*/
object SparkOnSolr{
    def main(args: Array[String]): Unit ={
        val testSample = new SparkOnSolr
        testSample.initProperties()
        val conf = new SparkConf().setAppName("SparkOnSolr")
        val sc = new SparkContext(conf)
        var cloudSolrClient: CloudSolrClient = null
        try {
            cloudSolrClient = testSample.getCloudSolrClient(testSample.zkHost)
            val collectionNames = testSample.queryAllCollections(cloudSolrClient)
            //delete the collection if exists
            if (collectionNames.contains(testSample.collectionName)) testSample.deleteCollection(cloudSolrClient)
            testSample.createCollection(cloudSolrClient)
            cloudSolrClient.setDefaultCollection(testSample.collectionName)
            testSample.addDocs(cloudSolrClient)
            testSample.addDocs2(cloudSolrClient)
            //Some time is needed to build the index
            try
                Thread.sleep(2000)
            catch {
                case e: InterruptedException =>
            }
            val queryResult1 = testSample.queryIndex(cloudSolrClient)
            import collection.JavaConversions._
            val resultRDD = sc.parallelize(queryResult1)
            System.out.println("Before delete, the total count is:" + resultRDD.count + "\n")
            testSample.removeIndex(cloudSolrClient)
            //Some time is needed to delete the index
            try
                Thread.sleep(2000)
            catch {
                case e: InterruptedException =>
            }
            val queryResult2 = testSample.queryIndex(cloudSolrClient)
            val resultRDD2 = sc.parallelize(queryResult2)
            System.out.println("After delete, the total count is:" + resultRDD2.count + "\n")
        } catch {
            case e: SolrException =>
                throw new SolrException(e.getMessage, null)
        } finally if (cloudSolrClient != null) try
            cloudSolrClient.close()
        catch {
            case e: IOException =>
                testSample.LOG.warn("Failed to close cloudSolrClient", e)
        }
        sc.stop()
    }
}
```

```
}
```

#### 2.17.3.11.4 Python Sample Code

### Function

In the Spark application, Solr client APIs are used to create indexes, insert data, and search for data. Finally, Spark performs simple operations on the queried data.

### Sample Code

PySpark does not provide Solr APIs. This example is implemented by using Python to invoke Java. When the application is running, you need to add some JAR packages of Java to the class path.

1. In Yarn-client mode, use the **--jars** parameter to load the Java **SparkOnSolr.jar** package to be invoked.

```
./spark-submit --jars SparkOnSolr.jar --master yarn-client  
SparkOnSolrPythonExample.py
```

2. In Yarn-cluster mode, modify the configuration item in the **spark-defaults.conf** file to add the **SparkOnSolr.jar** file to the class path. The configuration method is as follows:

- Distribute the extra JAR packages to each node of the cluster by using the configuration item.

```
spark.yarn.dist.innerfiles = /opt/hadoopclient/Spark/customJars/  
SparkOnSolr.jar,/opt/hadoopclient/Spark/customJars/  
httpmime-4.4.1.jar
```

- Add the specified JAR package to the class path of Driver in Yarn-cluster mode by using the configuration item.

```
spark.yarn.cluster.driver.extraClassPath = $PWD/  
SparkOnSolr.jar:$PWD/httpmime-4.4.1.jar
```

- Run the following command in Yarn-cluster mode:

```
./spark-submit --files ./solr-example.properties --master yarn-cluster  
--jars SparkOnSolr.jar SparkOnSolrPythonExample.py
```

The following code snippets are used as an example. For complete codes, see the following **SparkOnSolrPythonExample** project:

```
# -*- coding:utf-8 -*-
from py4j.java_gateway import java_import
from pyspark import SparkContext

# Create SparkContext
spark = SparkContext(appName='SparkOnSolr')
# Import the class to be run to sc._jvm
java_import(spark._jvm, 'com.huawei.bigdata.spark.examples.SparkOnSolr')

# Create class instances and invoke the method
spark._jvm.SparkOnSolr().sparkonsolr(spark._jsc)

# Stop SparkContext
spark.stop()
```

#### 2.17.3.12 Structured Streaming Status Operation

### 2.17.3.12.1 Overview

#### Scenarios

Assume that you need to collect statistics on the number of events in each session and the start and end timestamp of the sessions.

You need to export the sessions that are in the updated state in this batch.

#### Data Planning

1. Generate simulated data in Kafka (the Kafka permission is required).
2. Ensure that the clusters are installed, including HDFS, Yarn, Spark, and Kafka.
3. Create a topic.

{IP address of the Kafka cluster} indicates the service IP address of the Broker service.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --bootstrap-server <IP address of the Kafka cluster>:21005 --command-config $KAFKA_HOME/config/client.properties --replication-factor 1 --partitions 1 --topic {Topic}
```

4. Start the Producer of Kafka and send data to Kafka.  
{ClassPath} indicates the storage path of the project JAR package that is specified by the user. For details, see [Compiling and Running the Application](#).

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/*:$SPARK_HOME/jars/streamingClient010/*:{ClassPath} com.huawei.bigdata.spark.examples.KafkaProducer {brokerlist} {topic} {number of events produce every 0.02s}
```

Example:

```
java -cp /opt/hadoopclient/Spark/spark/conf:/opt/StructuredStreamingState-1.0.jar:/opt/hadoopclient/Spark/spark/jars/*:/opt/hadoopclient/Spark/spark/jars/streamingClient010/* com.huawei.bigdata.spark.examples.KafkaProducer xxx.xxx.xxx:21005,xxx.xxx.xxx:21005,xxx.xxx.xxx:21005 mytopic 10
```

#### Development Guideline

1. Receive data from Kafka and generate the corresponding DataStreamReader.
2. Collect statistics by category.
3. Calculate the result and print it.

#### Packaging the Project

- Use the Maven tool provided by IDEA to pack the project and generate a JAR file.
- Upload the JAR package to any directory (for example, `/opt`) on the server where the Spark client is located.

#### Running Tasks

When running the sample application, you need to specify `<brokers>`, `<subscribe-type>`, `<topic>`, and `<checkpointLocation>`.

- <brokers> indicates the Kafka address for obtaining metadata.
- <subscribe-type> indicates the Kafka consumption mode.
- <topic> indicates the Kafka topic to be consumed.
- <checkpointLocation> indicates the path for storing the checkpoint of the Spark task.

 NOTE

The path of the Spark Structured Streaming Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**. Whereas the path of the Spark Streaming Structured Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the dependency package of Spark Streaming Kafka, for example, `--jars ${files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,: echo "${files[*]}"}"`

Go to the Spark client directory and run the following command to invoke the **bin/spark-submit** script to run the code (The class name and file name must be the same as those in the actual code. The following is only an example):

- `bin/spark-submit --master yarn --deploy-mode client --jars ${files=($SPARK_HOME/jars/streamingClient010/*.jar); IFS=,: echo "${files[*]}"}" --class com.huawei.bigdata.spark.examples.kafkaSessionization /opt/StructuredStreamingState-1.0.jar <brokers> <subscribe-type> <topic> <checkpointLocation>`

 CAUTION

When submitting a structured stream task, you need to run the **--jars** command to specify the path of the Kafka-related JAR file. For the current version, you need to copy the `kafka-clientsjar` file from the **\$SPARK\_HOME/jars/streamingClient010** directory to the **\$SPARK\_HOME/jars** directory. Otherwise, the "class not found" error is reported.

### 2.17.3.12.2 Scala Sample Code

#### Function

In the Spark structure flow application, the number of events in each session and the start and end timestamp of the sessions are collected in different batches. At the same time, the system exports the sessions that are in the updated state in this batch.

#### Code Example

The following code snippets are used as an example. For complete codes, see `com.huawei.bigdata.spark.examples.kafkaSessionization`.

 NOTE

When new data is available in Streaming DataFrame/Dataset, **outputMode** is used for configuring data written to the Streaming receiver.

```
object kafkaSessionization {
  def main(args: Array[String]): Unit = {
    if (args.length < 7) {
      System.err.println("Usage: kafkaSessionization <bootstrap-servers> " +
        "<subscribe-type> <protocol> <service> <domain> <topics> <checkpointLocation>")
      System.exit(1)
    }

    val Array(bootstrapServers, subscribeType, protocol, service, domain, topics, checkpointLocation) = args

    val spark = SparkSession
      .builder
      .appName("kafkaSessionization")
      .getOrCreate()

    spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

    spark.streams.addListener(new StreamingQueryListener {

      @volatile private var startTime: Long = 0L
      @volatile private var endTime: Long = 0L
      @volatile private var numRecs: Long = 0L

      override def onQueryStarted(event: StreamingQueryListener.QueryStartedEvent): Unit = {
        println("Query started: " + event.id)
        startTime = System.currentTimeMillis
      }

      override def onQueryProgress(event: StreamingQueryListener.QueryProgressEvent): Unit = {
        println("Query made progress: " + event.progress)
        numRecs += event.progress.numInputRows
      }

      override def onQueryTerminated(event: StreamingQueryListener.QueryTerminatedEvent): Unit = {
        println("Query terminated: " + event.id)
        endTime = System.currentTimeMillis
      }
    })
  }

  import spark.implicits._

  val df = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option(subscribeType, topics)
    .load()
    .selectExpr("CAST(value AS STRING)")
    .as[String]
    .map { x =>
      val splitStr = x.split(",")
      (splitStr(0), Timestamp.valueOf(splitStr(1)))
    }.as[(String, Timestamp)].flatMap { case(line, timestamp) =>
      line.split(" ").map(word => Event(sessionId = word, timestamp))}
    }

    // Sessionize the events. Track number of events, start and end timestamps of session, and
    // and report session updates.
    val sessionUpdates = df
      .groupByKey(event => event.sessionId)
      .mapGroupsWithState[SessionInfo, SessionUpdate](GroupStateTimeout.ProcessingTimeTimeout) {

      case (sessionId: String, events: Iterator[Event], state: GroupState[SessionInfo]) =>

        // If timed out, then remove session and send final update
        if (state.hasTimedOut) {

```

```

    val finalUpdate =
      SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = true)
      state.remove()
      finalUpdate
  } else {
    // Update start and end timestamps in session
    val timestamps = events.map(_.timestamp.getTime).toSeq
    val updatedSession = if (state.exists) {
      val oldSession = state.get
      SessionInfo(
        oldSession.numEvents + timestamps.size,
        oldSession.startTimestampMs,
        math.max(oldSession.endTimestampMs, timestamps.max))
    } else {
      SessionInfo(timestamps.size, timestamps.min, timestamps.max)
    }
    state.update(updatedSession)

    // Set timeout such that the session will be expired if no data received for 10 seconds
    state.setTimeoutDuration("10 seconds")
    SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = false)
  }
}
// Start running the query that prints the session updates to the console
val query = sessionUpdates
  .writeStream
  .outputMode("update")
  .format("console")
  .start()

query.awaitTermination()
}

```

```

object kafkaSessionization {
  def main(args: Array[String]): Unit = {
    if (args.length < 7) {
      System.err.println("Usage: kafkaSessionization <bootstrap-servers> " +
        "<subscribe-type> <protocol> <service> <domain> <topics> <checkpointLocation>")
      System.exit(1)
    }

    val Array(bootstrapServers, subscribeType, protocol, service, domain, topics, checkpointLocation) = args

    val spark = SparkSession
      .builder
      .appName("kafkaSessionization")
      .getOrCreate()

    spark.conf.set("spark.sql.streaming.checkpointLocation", checkpointLocation)

    spark.streams.addListener(new StreamingQueryListener {

      @volatile private var startTime: Long = 0L
      @volatile private var endTime: Long = 0L
      @volatile private var numRecs: Long = 0L

      override def onQueryStarted(event: StreamingQueryListener.QueryStartedEvent): Unit = {
        println("Query started: " + event.id)
        startTime = System.currentTimeMillis
      }

      override def onQueryProgress(event: StreamingQueryListener.QueryProgressEvent): Unit = {
        println("Query made progress: " + event.progress)
        numRecs += event.progress.numInputRows
      }

      override def onQueryTerminated(event: StreamingQueryListener.QueryTerminatedEvent): Unit = {
        println("Query terminated: " + event.id)
        endTime = System.currentTimeMillis
      }
    })
  }
}

```

```
}

import spark.implicits._

val df = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("kafka.security.protocol", protocol)
    .option("kafka.sasl.kerberos.service.name", service)
    .option("kafka.kerberos.domain.name", domain)
    .option(subscribeType, topics)
    .load()
    .selectExpr("CAST(value AS STRING)")
    .as[String]
    .map { x =>
        val splitStr = x.split(",")
        (splitStr(0), Timestamp.valueOf(splitStr(1)))
    }.as[(String, Timestamp)].flatMap { case(line, timestamp) =>
        line.split(" ").map(word => Event(sessionId = word, timestamp))}

// Sessionize the events. Track number of events, start and end timestamps of session, and
// and report session updates.
val sessionUpdates = df
    .groupByKey(event => event.sessionId)
    .mapGroupsWithState[SessionInfo, SessionUpdate](GroupStateTimeout.ProcessingTimeTimeout) {

    case (sessionId: String, events: Iterator[Event], state: GroupState[SessionInfo]) =>

        // If timed out, then remove session and send final update
        if (state.hasTimedOut) {
            val finalUpdate =
                SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = true)
            state.remove()
            finalUpdate
        } else {
            // Update start and end timestamps in session
            val timestamps = events.map(_.timestamp.getTime).toSeq
            val updatedSession = if (state.exists) {
                val oldSession = state.get
                SessionInfo(
                    oldSession.numEvents + timestamps.size,
                    oldSession.startTimestampMs,
                    math.max(oldSession.endTimestampMs, timestamps.max))
            } else {
                SessionInfo(timestamps.size, timestamps.min, timestamps.max)
            }
            state.update(updatedSession)
        }

        // Set timeout such that the session will be expired if no data received for 10 seconds
        state.setTimeoutDuration("10 seconds")
        SessionUpdate(sessionId, state.get.durationMs, state.get.numEvents, expired = false)
    }
}
// Start running the query that prints the session updates to the console
val query = sessionUpdates
    .writeStream
    .outputMode("update")
    .format("console")
    .start()

query.awaitTermination()
}
```

### 2.17.3.13 Synchronizing HBase Data from Spark to CarbonData

### 2.17.3.13.1 Overview

#### Scenarios

Data is written to HBase in real time for point query services and is synchronized to CarbonData tables in batches at a specified interval for analytical query services.

#### Data Preparation

##### NOTE

Before running the sample program, set the `spark.yarn.security.credentials.hbase.enabled` configuration item to `true` in the `spark-defaults.conf` configuration file of Spark client. (The default value is `false`. Changing the value to `true` does not affect existing services.) If you want to uninstall the HBase service, change the value back to `false` first.

1. Create an HBase table and construct data with `key`, `modify_time`, and `valid` columns. `key` of each data record is unique in the table. `modify_time` indicates the modification time, and `valid` indicates whether the data is valid. In this example, `1` indicates that the data is valid, and `0` indicates that the data is invalid.

For example, go to HBase Shell and run the following commands:

```
create 'hbase_table','key','info'  
put 'hbase_table','1','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','1','info:valid','1'  
put 'hbase_table','2','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','2','info:valid','1'  
put 'hbase_table','3','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','3','info:valid','0'  
put 'hbase_table','4','info:modify_time','2019-11-22 23:28:39  
put 'hbase_table','4','info:valid','1'
```

##### NOTE

The values of `modify_time` in the preceding information can be set to the time earlier than the current time.

```
put 'hbase_table','5','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','5','info:valid','1'  
put 'hbase_table','6','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','6','info:valid','1'  
put 'hbase_table','7','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','7','info:valid','0'  
put 'hbase_table','8','info:modify_time','2021-03-03 15:20:39  
put 'hbase_table','8','info:valid','1'  
put 'hbase_table','4','info:valid','0'  
put 'hbase_table','4','info:modify_time','2021-03-03 15:20:39'
```

 NOTE

The values of **modify\_time** in the preceding information can be set to the time within 30 minutes after the sample program is started. (30 minutes is the default synchronization interval of the sample program and can be modified.)

```
put 'hbase_table','9','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','9','info:valid','1'  
put 'hbase_table','10','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','10','info:valid','1'  
put 'hbase_table','11','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','11','info:valid','0'  
put 'hbase_table','12','info:modify_time','2021-03-03 15:32:39  
put 'hbase_table','12','info:valid','1'
```

 NOTE

The values of **modify\_time** in the preceding information can be set to the time from 30 minutes to 60 minutes after the sample program is started, that is, the second synchronization period.

2. Run the following commands to create a Hive foreign table for HBase in SparkSQL:

```
create table external_hbase_table(key string ,modify_time STRING, valid STRING)
```

```
using org.apache.spark.sql.hbase.HBaseSource
```

```
options("hbaseTableName "hbase_table", "keyCols "key", "colsMapping "modify_time=info.modify_time,valid=info.valid");
```

3. Run the following command to create a CarbonData table in SparkSQL:

```
create table carbon01(key string,modify_time STRING, valid STRING) stored as carbodata;
```

4. Initialize and load all data in the current HBase table to the CarbonData table.

```
insert into table carbon01 select * from external_hbase_table where valid='1';
```

5. Run the following **spark-submit** command:

```
spark-submit --master yarn --deploy-mode client --class com.huawei.bigdata.spark.examples.HBaseExternalHivetoCarbon /opt/example/HBaseExternalHivetoCarbon-1.0.jar
```

### 2.17.3.13.2 Java Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.spark.examples.HBaseExternalHivetoCarbon](#).

```
public static void main(String[] args) throws Exception {  
    spark = SparkSession.builder().appName("HBaseExternalHiveToCarbon").getOrCreate();  
  
    Timer timer = new Timer();  
    timer.schedule(new TimerTask() {  
        public void run() {  
            timeEnd = timeStart + TIMEWINDOW;  
  
            queryTimeStart = transferDateToStr(timeStart);  
            queryTimeEnd = transferDateToStr(timeEnd);  
        }  
    }, 0, 1000);  
}
```

```
//run delete logic
cmdsb = new StringBuilder();
cmdsb.append("delete from ")
.append(carbonTableName)
.append(" where key in (select key from ")
.append(externalHiveTableName)
.append(" where modify_time>'")
.append(queryTimeStart)
.append("") and modify_time<'")
.append(queryTimeEnd)
.append("") and valid='0')");
spark.sql(cmdsb.toString());

//run insert logic
cmdsb = new StringBuilder();
cmdsb.append("insert into ")
.append(carbonTableName)
.append(" select * from ")
.append(externalHiveTableName)
.append(" where modify_time>'")
.append(queryTimeStart)
.append("") and modify_time<'")
.append(queryTimeEnd)
.append("") and valid='1");
spark.sql(cmdsb.toString());

timeStart = timeEnd;
}
}, TIMEWINDOW, TIMEWINDOW);
}
```

## 2.17.3.14 Using Spark to Perform Basic Hudi Operations

### 2.17.3.14.1 Overview

#### Scenarios

This section describes how to use Spark to perform operations such as data insertion, query, update, incremental query, query at a specific time point, and data deletion on Hudi.

For details, see the sample code.

#### Packaging the Project

1. Use the Maven tool provided by IDEA to package the project and generate the JAR file. For details, see [Compiling and Running the Application](#).



The Python sample code does not need to be packaged using Maven.

2. Upload the generated JAR file to any directory (for example, `/opt/example/`) on the server where the Spark client is located.

#### Running Tasks

1. Log in to the Spark client node and run the following commands:

`source Client installation directory/bigdata_env`

`source Client installation directory/Hudi/component_env`

2. After compiling and building the sample code, you can use the **spark-submit** command to perform the write, update, query, and delete operations in sequence.

- Run the Java sample program.

```
spark-submit --class  
com.huawei.bigdata.hudi.examples.HoodieWriteClientExample /opt/  
example/hudi-java-examples-1.0.jar hdfs://hacluster/tmp/example/  
hoodie_java hoodie_java
```

**/opt/example/hudi-java-examples-1.0.jar** indicates the JAR file path, **hdfs://hacluster/tmp/example/hoodie\_java** indicates the storage path of the Hudi table, and **hoodie\_java** indicates the name of the Hudi table.

- Run the Scala sample program.

```
spark-submit --class  
com.huawei.bigdata.hudi.examples.HoodieDataSourceExample /opt/  
example/hudi-scala-examples-1.0.jar hdfs://hacluster/tmp/example/  
hoodie_scala hoodie_scala
```

**/opt/example/hudi-scala-examples-1.0.jar** indicates the JAR file path, **hdfs://hacluster/tmp/example/hoodie\_scala** indicates the storage path of the Hudi table, and **hoodie\_Scala** indicates the name of the Hudi table.

- Run the Python sample program.

```
spark-submit /opt/example/HudiPythonExample.py hdfs://  
hacluster/tmp/huditest/example/python hudi_trips_cow
```

**hdfs://hacluster/tmp/huditest/example/python** indicates the storage path of the Hudi table, and **hudi\_trips\_cow** indicates the name of the Hudi table.

### 2.17.3.14.2 Java Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.bigdata.hudi.examples.HoodieWriteClientExample](#).

Create a client object to operate Hudi:

```
String tablePath = args[0];  
String tableName = args[1];  
SparkConf sparkConf = HoodieExampleSparkUtils.defaultSparkConf("hoodie-client-example");  
JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
// Generator of some records to be loaded in.  
HoodieExampleDataGenerator<HoodieAvroPayload> dataGen = new HoodieExampleDataGenerator<>();  
// initialize the table, if not done already  
Path path = new Path(tablePath);  
FileSystem fs = FSUtils.getFs(tablePath, jsc.hadoopConfiguration());  
if (!fs.exists(path)) {  
    HoodieTableMetaClient.initTableType(jsc.hadoopConfiguration(), tablePath,  
    HoodieTableType.valueOf(tableName),  
    tableName, HoodieAvroPayload.class.getName());  
}  
  
// Create the write client to write some records in  
HoodieWriteConfig cfg = HoodieWriteConfig.newBuilder().withPath(tablePath)  
    .withSchema(HoodieExampleDataGenerator.TRIP_EXAMPLE_SCHEMA).withParallelism(2, 2)  
    .withDeleteParallelism(2).forTable(tableName)  
    .withIndexConfig(HoodieIndexConfig.newBuilder().withIndexType(HoodieIndex.IndexType.BLOOM).build())  
)  
    .withCompactionConfig(HoodieCompactionConfig.newBuilder().archiveCommitsWith(20,  
30).build()).build();
```

```
SparkRDDWriteClient<HoodieAvroPayload> client = new SparkRDDWriteClient<>(new  
HoodieSparkEngineContext(jsc), cfg);
```

Insert data:

```
String newCommitTime = client.startCommit();  
LOG.info("Starting commit " + newCommitTime);  
List<HoodieRecord<HoodieAvroPayload>> records = dataGen.generateInserts(newCommitTime, 10);  
List<HoodieRecord<HoodieAvroPayload>> recordsSoFar = new ArrayList<>(records);  
JavaRDD<HoodieRecord<HoodieAvroPayload>> writeRecords = jsc.parallelize(records, 1);  
client.upsert(writeRecords, newCommitTime);
```

Update data:

```
newCommitTime = client.startCommit();  
LOG.info("Starting commit " + newCommitTime);  
List<HoodieRecord<HoodieAvroPayload>> toBeUpdated = dataGen.generateUpdates(newCommitTime, 2);  
records.addAll(toBeUpdated);  
recordsSoFar.addAll(toBeUpdated);  
writeRecords = jsc.parallelize(records, 1);  
client.upsert(writeRecords, newCommitTime);
```

Delete data:

```
newCommitTime = client.startCommit();  
LOG.info("Starting commit " + newCommitTime);  
// just delete half of the records  
int numToDelete = recordsSoFar.size() / 2;  
List<HoodieKey> toBeDeleted =  
recordsSoFar.stream().map(HoodieRecord::getKey).limit(numToDelete).collect(Collectors.toList());  
JavaRDD<HoodieKey> deleteRecords = jsc.parallelize(toBeDeleted, 1);  
client.delete(deleteRecords, newCommitTime);
```

Compress data.

```
if (HoodieTableType.valueOf(tableType) == HoodieTableType.MERGE_ON_READ) {  
    Option<String> instant = client.scheduleCompaction(Option.empty());  
    JavaRDD<WriteStatus> writeStatuses = client.compact(instant.get());  
    client.commitCompaction(instant.get(), writeStatuses, Option.empty());  
}
```

### 2.17.3.14.3 Scala Example Code

The following code snippets are used as an example. For complete code, see [com.huawei.bigdata.hudi.examples.HoodieDataSourceExample](#).

Insert data:

```
def insertData(spark: SparkSession, tablePath: String, tableName: String, dataGen:  
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {  
val commitTime: String = System.currentTimeMillis().toString  
val inserts = dataGen.convertToStringList(dataGen.generateInserts(commitTime, 20))  
spark.sparkContext.parallelize(inserts, 2)  
val df = spark.read.json(spark.sparkContext.parallelize(inserts, 1)).df.write.format("org.apache.hudi").  
options(getQuickstartWriteConfigs).  
option(PRECOMBINE_FIELD_OPT_KEY, "ts").  
option(RECORDKEY_FIELD_OPT_KEY, "uuid").  
option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").  
option(TABLE_NAME, tableName).  
mode(Overwrite).  
save(tablePath)}
```

Query data.

```
def queryData(spark: SparkSession, tablePath: String, tableName: String, dataGen:  
HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {  
val roViewDF = spark.  
read.  
format("org.apache.hudi").  
load(tablePath + "/*/*/*")
```

```

roViewDF.createOrReplaceTempView("hudi_ro_table")
spark.sql("select fare, begin_lon, begin_lat, ts from hudi_ro_table where fare > 20.0").show()
// +-----+-----+-----+
// |      fare|    begin_lon|    begin_lat| ts|
// +-----+-----+-----+
// |98.88075495133515|0.39556048623031603|0.17851135255091155|0.0|
// ...
spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from hudi_ro_table").show()
// +-----+-----+-----+-----+
// |_hoodie_commit_time| _hoodie_record_key|_hoodie_partition_path|      rider|
driver|          fare|               |
// +-----+-----+-----+-----+
// | 20191231181501|31cafb9f-0196-4b1...| 2020/01/02|rider-1577787297889|
driver-1577787297889| 98.88075495133515|
// ...
}

```

### Update data:

```

def updateData(spark: SparkSession, filePath: String, tableName: String, dataGen: HoodieExampleDataGenerator[HoodieAvroPayload]): Unit = {
  val commitTime: String = System.currentTimeMillis().toString
  val updates = dataGen.convertToStringList(dataGen.generateUpdates(commitTime, 10))
  val df = spark.read.json(spark.sparkContext.parallelize(updates, 1))
  df.write.format("org.apache.hudi").
    options(getQuickstartWriteConfigs).
    option(PRECOMBINE_FIELD_OPT_KEY, "ts").
    option(RECORDKEY_FIELD_OPT_KEY, "uuid").
    option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
    option(TABLE_NAME, tableName).
    mode(Append).
    save(filePath)
}

```

### Incremental query:

```

def incrementalQuery(spark: SparkSession, filePath: String, tableName: String) {
  import spark.implicits._
  val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by commitTime").map(k => k.getString(0)).take(50)
  val beginTime = commits(commits.length - 2)

  val incViewDF = spark.
    read.
    format("org.apache.hudi").
    option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
    option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
    load(filePath)
  incViewDF.createOrReplaceTempView("hudi_incr_table")
  spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare > 20.0").show()
}

```

### Query at a specific time point:

```

def pointInTimeQuery(spark: SparkSession, filePath: String, tableName: String) {
  import spark.implicits._
  val commits = spark.sql("select distinct(_hoodie_commit_time) as commitTime from hudi_ro_table order by commitTime").map(k => k.getString(0)).take(50)
  val beginTime = "000"
  // Represents all commits > this time.
  val endTime = commits(commits.length - 2)
  // commit time we are interested in
  //incrementally query data
  val incViewDF = spark.read.format("org.apache.hudi").
    option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL).
    option(BEGIN_INSTANTTIME_OPT_KEY, beginTime).
    option(END_INSTANTTIME_OPT_KEY, endTime).
    load(filePath)
}

```

```
incViewDF.createOrReplaceTempView("hudi_incr_table")
spark.sql("select `hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_incr_table where fare > 20.0").show()
```

### 2.17.3.14.4 Python Sample Code

#### Using Python to Write Data to a Hudi Table

The following code snippets are used as an example. For complete code, see [sparknormal-examples.SparkOnHudiPythonExample.hudi\\_python\\_write\\_example](#).

Insert data:

```
#insert
inserts = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateInserts(10))
df = spark.read.json(spark.sparkContext.parallelize(inserts, 2))
hoodie_options = {
    'hoodie.table.name': tableName,
    'hoodie.datasource.write.recordkey.field': 'uuid',
    'hoodie.datasource.write.partitionpath.field': 'partitionpath',
    'hoodie.datasource.write.table.name': tableName,
    'hoodie.datasource.write.operation': 'insert',
    'hoodie.datasource.write.precombine.field': 'ts',
    'hoodie.upsert.shuffle.parallelism': 2,
    hoodie.insert.shuffle.parallelism': 2
}
df.write.format("hudi"). \
    options(**hoodie_options). \
    mode("overwrite"). \
    save(basePath)
```

Query data.

```
tripsSnapshotDF = spark. \
    read. \
    format("hudi"). \
    load(basePath + "/*/*/*/*")
tripsSnapshotDF.createOrReplaceTempView("hudi_trips_snapshot")
spark.sql("select fare, begin_lon, begin_lat, ts from hudi_trips_snapshot where fare > 20.0").show()
spark.sql("select _hoodie_commit_time, _hoodie_record_key, _hoodie_partition_path, rider, driver, fare from hudi_trips_snapshot").show()
```

Update data:

```
updates = sc._jvm.org.apache.hudi.QuickstartUtils.convertToStringList(dataGen.generateUpdates(10))
df = spark.read.json(spark.sparkContext.parallelize(updates, 2))
df.write.format("hudi"). \
    options(**hoodie_options). \
    mode("append"). \
    save(basePath)
```

Query incremental data:

```
spark. \
    read. \
    format("hudi"). \
    load(basePath + "/*/*/*/*"). \
    createOrReplaceTempView("hudi_trips_snapshot")
incremental_read_options = {
    'hoodie.datasource.query.type': 'incremental',
    'hoodie.datasource.read.begin.instanttime': beginTime,
}
tripsIncrementalDF = spark.read.format("hudi"). \
    options(**incremental_read_options). \
    load(basePath)
tripsIncrementalDF.createOrReplaceTempView("hudi_trips_incremental")
```

```
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_incremental where fare > 20.0").show()
```

Query data at a specific time point:

```
# Represents all commits > this time.  
beginTime = "000"  
endTime = commits[len(commits) - 2]  
point_in_time_read_options = {  
    'hoodie.datasource.query.type': 'incremental',  
    'hoodie.datasource.read.end.instanttime': endTime,  
    'hoodie.datasource.read.begin.instanttime': beginTime  
}  
  
tripsPointInTimeDF = spark.read.format("hudi"). \  
    options(**point_in_time_read_options). \  
    load(basePath)  
  
tripsPointInTimeDF.createOrReplaceTempView("hudi_trips_point_in_time")  
spark.sql("select `_hoodie_commit_time`, fare, begin_lon, begin_lat, ts from hudi_trips_point_in_time where fare > 20.0").show()
```

Delete data:

```
# Obtain the total number of records.  
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()  
# Obtain two records to be deleted.  
ds = spark.sql("select uuid, partitionpath from hudi_trips_snapshot").limit(2)  
# Delete the records.  
hudi_delete_options = {  
    'hoodie.table.name': tableName,  
    'hoodie.datasource.write.recordkey.field': 'uuid',  
    'hoodie.datasource.write.partitionpath.field': 'partitionpath',  
    'hoodie.datasource.write.table.name': tableName,  
    'hoodie.datasource.write.operation': 'delete',  
    'hoodie.datasource.write.precombine.field': 'ts',  
    'hoodie.upsert.shuffle.parallelism': 2,  
    'hoodie.insert.shuffle.parallelism': 2  
}  
from pyspark.sql.functions import lit  
deletes = list(map(lambda row: (row[0], row[1]), ds.collect()))  
df = spark.sparkContext.parallelize(deletes).toDF(['uuid', 'partitionpath']).withColumn('ts', lit(0.0))  
df.write.format("hudi"). \  
    options(**hudi_delete_options). \  
    mode("append"). \  
    save(basePath)  
# Perform the query in the same way.  
roAfterDeleteViewDF = spark. \  
    read. \  
    format("hudi"). \  
    load(basePath + "/*/*/*")  
roAfterDeleteViewDF.registerTempTable("hudi_trips_snapshot")  
# Return (total - 2) records.  
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").count()  
spark.sql("select uuid, partitionpath from hudi_trips_snapshot").show()
```

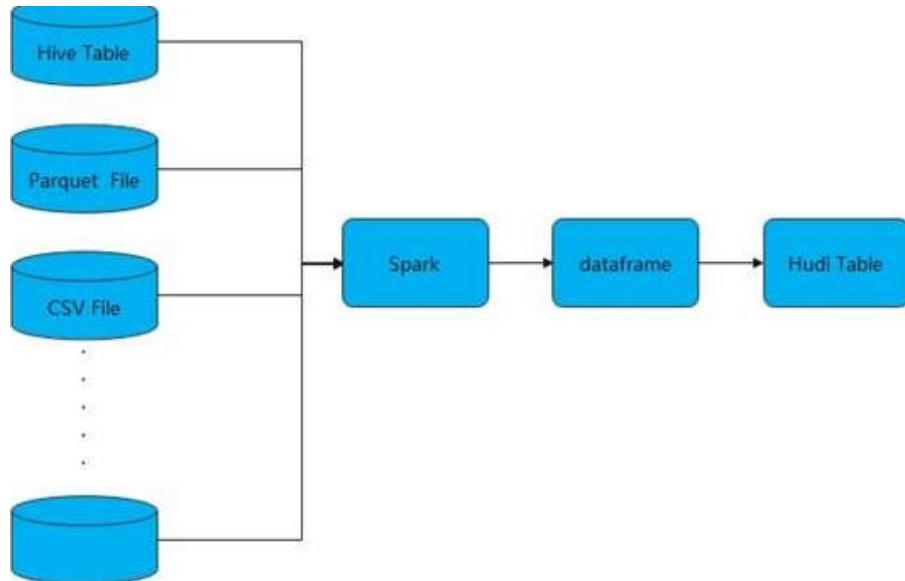
## Importing Inventory Data for Generating Hudi Tables

### Application Scenarios

In production scenarios, customers' inventory data is usually stored in HDFS, Hive, or OBS. After Hudi is introduced, the inventory data or tables need to be converted into Hudi tables for subsequent stream writing. This section describes how to import inventory data to generate Hudi tables.

### Solution Design

Spark can read data in various formats, such as Hive tables and HDFS files, and convert it to **DataFrame**. Therefore, Spark DataSource is used to initialize a Hudi table. The following describes how to read raw data, convert it to **DataFrame**, and write the data into a Hudi table.



Sample scripts are provided in the Hudi sample code to help users quickly initialize data to the data lake.

For details about the development code, see the scripts in sample code `sparknormal-examples.SparkOnHudiPythonExample.hudi_init_example`.

- File description

The **hudiwriteconfig.conf** file contains various write parameters of the Hudi table. For details, visit Hudi official website at <https://hudi.apache.org/docs/quick-start-guide/>.

**inithudi.py** is a Python script used to initialize Hudi tables. The source data to be written has various type, and therefore is not parameterized in this example. You can modify the read logic of 20 rows as required.

For example:

For parquet files: `base_data = spark.read.parquet("/tmp/tb_base")`.

For CSV files: `base_data = spark.read.format("csv").load("tmp/huditest/csvfile.csv")`.

Directly read the inventory table: `base_data = spark.sql("select * from hive_table")`.

- Job submission

- Use WinSCP to upload **inithudi.py** and **hudiwriteconfig.conf** to the same directory on the client node as user **root**.

- Run the following commands to set the environment variables:

```
source Client installation directory/bigdata_env
```

```
source Client installation directory/Hudi/component_env
```

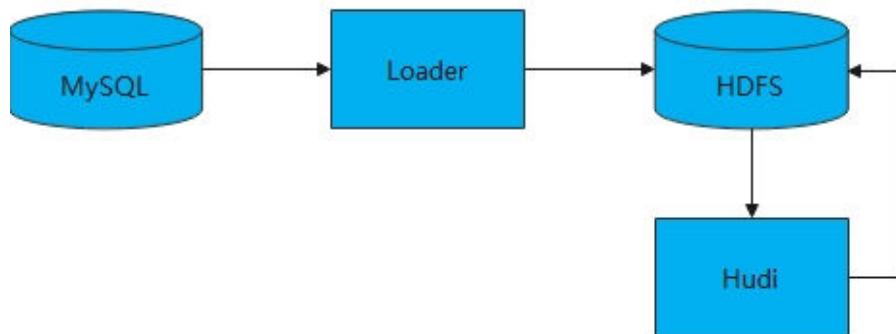
- Use **spark-submit** to submit a job:

```
spark-submit --master yarn --driver-memory 1g --executor-memory  
1g --executor-cores 1 --num-executors 2 inithudi.py  
hudiwriteconfig.conf
```

## Using Loader to Synchronize Data from a Relational Database to a Hudi Table

### Overview

In production scenarios, customers have many existing relational database tables. However, with the increase of data analysis requirements, historical inventory data needs to be imported from relational databases to the data lake if the data lake architecture is used. Hudi, which is built on the Hadoop file system, is a storage framework of the data lake. It provides the capabilities of updating and deleting data and consuming changed data. Loader is a data migration tool provided by MRS. This document describes how to use Loader and Hudi to quickly synchronize MySQL tables to Hudi tables.



### Solution Design

The preceding figure shows the overall process. Use Loader to establish a connection to access the MySQL databases and create a job to synchronize data to HDFS. After the job is complete, submit a Spark job to write data to Hudi and then the synchronized data in the HDFS is converted to Hudi tables.

The highlight of this solution is that users can synchronize data from original relational databases to Hudi tables in the data lake with few manual operations.

For details about the development code, see the scripts in sample code  
**sparknormal-examples.SparkOnHudiPythonExample.hudi\_init\_loader2hudi\_example**.

### Prerequisites

- Necessary components, such as Spark, Loader, HDFS, and Hive, have been installed in the MRS cluster, and corresponding component permissions have been obtained.
- The required database driver has been loaded by Loader, and authentication rules have been configured.
- A MySQL table has been prepared. The following figure shows the field types and data.

col0	varchar	255				
col1	bigint					
col2	timestamp					
col3	char	1				
col4	date					
col5	decimal	10	4			
col6	double					
col7	float					
col8	int					

### Editing the Configuration File

1. Edit the **table.conf** file to set the connection information, as shown in the following figure.

```
[connection]
connectionName=myconnection
connectionString=jdbc:mysql://10.xxx.xxx.193:3306/mysql?useUnicode=true&characterEncoding=utf8&serverTimezone=UTC
jdbcDriver=com.mysql.jdbc.Driver
username=xxx
password=xxx

[job]
jobName=myjob
schemaName=mysql
tableName=testloader
outputDirectory=/tmp/
trans=/opt/loader2hudi/testloader.json

[hudi]
precombineFiled=col2
recordkeyFiled=col0
table_type=COPY_ON_WRITE
keygenerator=org.apache.hudi.keygen.NonpartitionedKeyGenerator
writeOperation=bulk_insert
extractor=org.apache.hudi.hive.NonPartitionedExtractor
parallelism=10
writeMode=Overwrite
partitionFields=""
```

The **trans** file in the figure is a configuration file used by Loader, and it needs to be set in advance to specify fields to be synchronized. This file is in JSON format and can be named **tableloader.json**.

Note that Loader is incompatible with **date** fields during data synchronization to Hive. Therefore, you are advised to set **date** fields to the **timestamp** type.

2. After the synchronization task is created, place the **properties.conf**, **test.py**, and **pyhudi.py** files in the same directory, and place the **mysql-to-hdfs.json** file in the directory specified in the configuration file.

```
source Client installation directory/bigdata_env
source Client installation directory/Hudi/component_env
python3 loader2hudi.py testloader.conf
```

3. After the synchronization task is complete, log in to Hive Beeline to view the synchronized data.

## 2.17.3.15 Compiling User-defined Configuration Items for Hudi

### 2.17.3.15.1 HoodieDeltaStreamer

Compile a user-defined conversion class for **Transformer**.

Compile a user-defined schema for **SchemaProvider**.

Add the following parameters when running **HoodieDeltaStreamer**:

```
--schemaprovider-class Defined schema class --transformer-class Defined transform class
```

Examples of **Transformer** and **SchemaProvider**:

### 2.17.3.15.2 User-defined Partitioner

Compile a user-defined partitioner class that inherits **BulkInsertPartitioner** and add the following configuration when writing data to Hudi:

`.option(BULKINSERT_USER_DEFINED_PARTITIONER_CLASS, <User-defined partitioner class package name + class name>)`

Example of the user-defined partitioner:

```
public class HoodieSortExample<T extends HoodieRecordPayload>
    implements BulkInsertPartitioner<JavaRDD<HoodieRecord<T>>> {
    @Override
    public JavaRDD<HoodieRecord<T>> repartitionRecords(JavaRDD<HoodieRecord<T>> records, int
outputSparkPartitions) {
        JavaPairRDD<String,
                    HoodieRecord<T>> stringHoodieRecordJavaPairRDD = records.coalesce(outputSparkPartitions)
            .mapToPair(record -> new Tuple2<>(new StringBuilder().append(record.getPartitionPath())
                .append("+")
                .append(record.getRecordKey())
                .toString(), record));
        JavaRDD<HoodieRecord<T>> hoodieRecordJavaRDD =
stringHoodieRecordJavaPairRDD.mapPartitions(partition -> {
            List<Tuple2<String, HoodieRecord<T>>> recordList = new ArrayList<>();
            for (; partition.hasNext(); ) {
                recordList.add(partition.next());
            }
            Collections.sort(recordList, (o1, o2) -> {
                if (o1._1().split("[+])[0] == o2._1().split("[+])[0]) {
                    return Integer.parseInt(o1._1().split("[+])[1]) - Integer.parseInt(o2._1().split("[+])[1]);
                } else {
                    return o1._1().split("[+])[0].compareTo(o2._1().split("[+])[0]);
                }
            });
            return recordList.stream().map(e -> e._2).iterator();
        });
        return hoodieRecordJavaRDD;
    }

    @Override
    public boolean arePartitionRecordsSorted() {
        return true;
    }
}
```

## 2.17.3.16 Using Spark to Write Data to ClickHouse

### 2.17.3.16.1 Overview

#### Scenarios

In Spark applications, users can use the native ClickHouse JDBC APIs and the Spark JDBC driver to create, query, and insert ClickHouse databases and tables.

#### Development Guidelines

Use the ClickHouse JDBC driver to create databases and tables, and insert data. Then, call the Spark JDBC APIs to read data from the ClickHouse table, convert the data, and write the converted data to the ClickHouse table.

The process is as follows:

- Create a ClickHouse database and table, and insert data into the table.
- Use the Spark JDBC API to read data from the ClickHouse table.
- Register a temporary table, process the field ID in the table, and return a new data set.
- Append the new data set to the ClickHouse table.

## Preparations

- Prepare user information, including the username and password. The user must have the ClickHouse administrator rights.
- Prepare the names of ClickHouse database and table to be created. The database and table names must be different from the existing ones. In addition, you need to create a ClickHouse logical cluster on FusionInsight Manager in advance. For details, see "Creating a User-Defined ClickHouse Logical Cluster" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*. For example, creating logical cluster **default\_cluster**.

## Packaging the Project

1. Use the Maven tool provided by IDEA to package the project and generate the JAR file.
2. Upload the generated JAR file to the Spark client directory, for example, **/opt/hadoopclient/Spark/spark**.
3. ClickHouse JDBC driver package **clickhouse-jdbc\*.jar** is automatically downloaded to the local Maven library during JAR file packaging and compilation. Upload the package to the Spark client directory on the server, for example, **/opt/hadoopclient/Spark/spark**. The JAR package version in the following steps is only an example.

## Running Tasks

Go to the Spark client directory, source environment variables, and perform user authentication. Assume that the following commands are executed in the **spark** directory of the Spark client. Call the **bin/spark-submit** script to run the code. The running commands are as follows (the class name and file name must be consistent with those in the actual code. The following is only an example):

- Run the Java sample program.  
**bin/spark-submit --master yarn --deploy-mode client/cluster --jars clickhouse-jdbc-0.3.1-h0.cbu.mrs.313.r1-SNAPSHOT.jar --class com.huawei.bigdata.spark.examples.SparkOnClickHouseExample SparkOnClickHouseJavaExample-1.0.jar <jdbcurl> <clickHouseDBName> <clickHouseTableName> <userName>**
- Run the Scala sample program.  
**bin/spark-submit --master yarn --deploy-mode client/cluster --jars clickhouse-jdbc-0.3.1-h0.cbu.mrs.313.r1-SNAPSHOT.jar --class com.huawei.bigdata.spark.examples.SparkOnClickHouseExample SparkOnClickHouseScalaExample-1.0.jar <jdbcurl> <clickHouseDBName> <clickHouseTableName> <userName>**
- Run the Python sample program.  
**bin/spark-submit --master yarn --deploy-mode client/cluster --jars SparkOnClickHousePythonExample-1.0.jar,clickhouse-jdbc-0.3.1-h0.cbu.mrs.313.r1-SNAPSHOT.jar SparkOnClickHousePythonExample.py <jdbcurl> <clickHouseDBName> <clickHouseTableName> <userName>**

#### NOTE

- **jdbcurl** is the ClickHouse JDBC connection string, for example, **jdbc:clickhouse://{ip}:{port}**. To query the port number, perform the following steps:  
Log in to FusionInsight Manager, choose **Cluster > Services > ClickHouse**, click **Logic Cluster**, and obtain the port number from the **HTTP Balancer Port** column in the cluster list.
- **clickHouseDBName** is the name of the ClickHouse database to be created, for example, **ckdb001**.
- **clickHouseTableName** is the name of the ClickHouse table to be created, for example, **ckdb01**.
- **userName** indicates the prepared username.

### 2.17.3.16.2 Java Sample Code

The following code snippets are used as an example. For complete code, see: <com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.java>.

## Procedure

**Step 1** Prepare the spark-clickhouse sample code. For details, see [Overview](#).

**Step 2** Import the sample project to IDEA.

**Step 3** Create a table and insert data using the ClickHouse JDBC API.

```
private static void clickHouseExecute(ClickHouseStatement ckStatement, String clickHouseDB, String clickHouseTable) throws SQLException {
    String createDB = "CREATE DATABASE " + clickHouseDB + " ON CLUSTER default_cluster";
    String createTable = "CREATE TABLE " + clickHouseDB + "." + clickHouseTable + " ON CLUSTER
default_cluster " +
        "('EventDate` DateTime,`id` UInt64,`name` String,`age` UInt8,`address` String)" +
        "ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}'" + clickHouseDB + "/" + clickHouseTable
+ "", '{replica}')"
        + "PARTITION BY toYYYYMM(EventDate) ORDER BY id";
    String insertData = "insert into " + clickHouseDB + "." + clickHouseTable + " (id, name, age, address)
values (1, 'zhangsan', 17, 'xian'), (2, 'lisi', 36, 'beijing')";
    ckStatement.execute(createDB);
    ckStatement.execute(createTable);
    ckStatement.execute(insertData);
}
```

**Step 4** Package the code and upload it to the Spark client directory, for example, **/opt/hadoopclient/Spark/spark**.

**Step 5** Go to the client directory and run the following commands:

```
cd Cluster client installation directory
```

```
source bigdata_env
```

```
source Spark/component_env
```

**Step 6** Run the task.

```
dbName: ckdb001
dbName: ckdb003
dbName: ckdb005
dbName: ckdb01
dbName: ckdb02
dbName: ckdb03
dbName: default
dbName: system
dbName: test
[Elapsed]770
2022-03-19 09:08:14,970 | WARN  | main | The enable mv value "null" is : | org.apache.carbondata.core.util.CarbonProperties.validateEnableMV(Carb
2022-03-19 09:08:15,002 | WARN  | main | The value "LOCALLOCK" configures r current file system. Use the default value HDFSLOCK instead. | org.apa
s.validateAndConfigureLockType(CarbonProperties.java:444)
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 1|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 2| lisi| 36| [REDACTED]
+-----+-----+-----+
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 21|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 22| lisi| 36| [REDACTED]
```

- Step 7** View the running result on the ClickHouse client. Assume that the created ClickHouse database is **ckdb0001** and the table is **cktbo1**.

`clickhouse client --host IP address of the ClickHouse instance --port 21423`

```
:) select * from ckdb0001.cktbo1;
SELECT *
FROM ckdb0001.cktbo1
Query id: 0fd090e8-c81c-4752-9a4d-6 [REDACTED] 3e
+-----+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 21|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 22| lisi| 36| [REDACTED]
+-----+-----+-----+
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 1|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 2| lisi| 36| [REDACTED]
+-----+-----+-----+
4 rows in set. Elapsed: 0.004 sec.
```

----End

### 2.17.3.16.3 Scala Sample Code

The following code snippets are used as an example. For complete code, see [com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.Scala](https://github.com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.Scala).

### Procedure

- Step 1** Prepare the spark-clickhouse sample code. For details, see [Overview](#).

**Step 2** Import the sample project to IDEA.

**Step 3** Create a table and insert data using the ClickHouse JDBC API.

```
def clickHouseExecute(ckStatement: ClickHouseStatement, clickHouseDB: String, clickHouseTable: String) {  
    val createDB = s"CREATE DATABASE ${clickHouseDB} ON CLUSTER default_cluster"  
    val createTable = s"CREATE TABLE ${clickHouseDB}.${clickHouseTable} ON CLUSTER default_cluster "  
    +"(`EventDate` DateTime, `id` UInt64, `name` String, `age` UInt8, `address` String)" +s"ENGINE =  
    ReplicatedMergeTree('/clickhouse/tables/{shard}/${clickHouseDB}/${clickHouseTable}', '{replica}')"  
    +"PARTITION BY toYYYYMM(EventDate) ORDER BY id"  
    val insertData = s"insert into ${clickHouseDB}.${clickHouseTable} (id, name, age, address) values (1,  
    'zhangsan', 17, 'xian'), (2, 'lisi', 36, 'beijing')"  
    executeSqlText(ckStatement, createDB)  
    executeSqlText(ckStatement, createTable)  
    executeSqlText(ckStatement, insertData)  
}
```

**Step 4** Package the code and upload it to the Spark client directory, for example, **/opt/hadoopclient/Spark/spark**.

**Step 5** Go to the client directory and run the following commands:

```
cd Cluster client installation directory
```

```
source bigdata_env
```

```
source Spark/component_env
```

**Step 6** Run the task. The code running result is as follows:

```
dbName: ckdb001  
dbName: ckdb003  
dbName: ckdb005  
dbName: ckdb01  
dbName: ckdb02  
dbName: ckdb03  
dbName: default  
dbName: system  
dbName: test  
[Elapsed]770  
2022-03-19 09:08:14,970 | WARN  | main | The enable_mv value "null" is set to false. | org.apache.carbondata.core.util.CarbonProperties.validateEnableMV(CarbonProperties.java:444)  
2022-03-19 09:08:15,002 | WARN  | main | The value "LOCALLOCK" configured in the current file system. Use the default value HDFSLOCK instead. | org.apache.carbondata.core.util.CarbonProperties.validateAndConfigureLockType(CarbonProperties.java:444)  
+-----+-----+-----+-----+  
| EventDate| id| name|age|address|  
+-----+-----+-----+-----+  
|1970-01-01 08:00:00| 1|zhangsan| 17|  
|1970-01-01 08:00:00| 2| lisi| 36|  
+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| EventDate| id| name|age|address|  
+-----+-----+-----+-----+  
|1970-01-01 08:00:00| 21|zhangsan| 17|  
|1970-01-01 08:00:00| 22| lisi| 36|  
+-----+-----+-----+-----+
```

**Step 7** View the running result on the ClickHouse client. Assume that the created ClickHouse database is **ckdb0001** and the table is **cktb01**.

```
clickhouse client --host IP address of the ClickHouse instance --port 21423
```

```
:) select * from ckdb0001.cktb01;

SELECT *
FROM ckdb0001.cktb01

Query id: 0fd090e8-c81c-4752-9a4d-63e

+-----+-----+-----+-----+-----+
| EventDate | id | name | age | address |
+-----+-----+-----+-----+-----+
| 1970-01-01 08:00:00 | 21 | zhangsan | 17 |          |
| 1970-01-01 08:00:00 | 22 | lisi | 36 |          |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| EventDate | id | name | age | address |
+-----+-----+-----+-----+-----+
| 1970-01-01 08:00:00 | 1 | zhangsan | 17 |          |
| 1970-01-01 08:00:00 | 2 | lisi | 36 |          |
+-----+-----+-----+-----+-----+

4 rows in set. Elapsed: 0.004 sec.
```

----End

#### 2.17.3.16.4 Python Sample Code

The following code snippets are used as an example. For complete code, see [com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.java](https://github.com/huawei/bigdata/spark/examples/SparkOnClickHouseExample.java).

### Procedure

**Step 1** Prepare the spark-clickhouse sample code. For details, see [Overview](#).

**Step 2** Import the sample project to IDEA.

**Step 3** Create a table and insert data using the ClickHouse JDBC API.

```
private static void clickHouseExecute(ClickHouseStatement ckStatement, String clickHouseDB, String
clickHouseTable) throws SQLException {
    String createDB = "CREATE DATABASE " + clickHouseDB + " ON CLUSTER default_cluster";
    String createTable = "CREATE TABLE " + clickHouseDB + "." + clickHouseTable + " ON CLUSTER
default_cluster " +
        "('EventDate` DateTime,'id` UInt64,'name` String,'age` UInt8,'address` String)" +
        "ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}'/" + clickHouseDB + "/" + clickHouseTable
+ "", '{replica}')"
        + "PARTITION BY toYYYYMM(EventDate) ORDER BY id";
    String insertData = "insert into " + clickHouseDB + "." + clickHouseTable + " (id, name,age,address)
values (1, 'zhangsan',17,'xian'), (2, 'lisi',36,'beijing')";
    ckStatement.execute(createDB);
    ckStatement.execute(createTable);
    ckStatement.execute(insertData);
}
```

**Step 4** Package the code and upload it to the Spark client directory, for example, `/opt/hadoopclient/Spark/spark`.

**Step 5** Go to the client directory and run the following commands:

```
cd Cluster client installation directory
```

```
source bigdata_env
```

```
source Spark/component_env
```

**Step 6** Run the task. The code running result is as follows:

```
dbName: ckdb001
dbName: ckdb003
dbName: ckdb005
dbName: ckdb01
dbName: ckdb02
dbName: ckdb03
dbName: default
dbName: system
dbName: test
[Elapsed]770
2022-03-19 09:08:14,970 | WARN  | main | The enable mv value "null" is : |
| org.apache.carbondata.core.util.CarbonProperties.validateEnableMV(Carb
2022-03-19 09:08:15,002 | WARN  | main | The value "LOCALLOCK" configure
r current file system. Use the default value HDFSLOCK instead. | org.apa
s.validateAndConfigureLockType(CarbonProperties.java:444)
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 1|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 2| lisi| 36| [REDACTED]
+-----+-----+-----+
+-----+-----+-----+
|   EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 21|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 22| lisi| 36| [REDACTED]
```

- Step 7** View the running result on the ClickHouse client. Assume that the created ClickHouse database is **ckdb0001** and the table is **cktbo1**.

`clickhouse client --host IP address of the ClickHouse instance --port 21423`

```
:) select * from ckdb0001.cktbo1;

SELECT *
FROM ckdb0001.cktbo1

Query id: 0fd090e8-c81c-4752-9a4d-6 [REDACTED] 3e

+-----+-----+-----+-----+-----+
| EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 21|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 22| lisi| 36| [REDACTED]
+-----+-----+-----+
+-----+-----+-----+
| EventDate| id| name|age|address|
+-----+-----+-----+
|1970-01-01 08:00:00| 1|zhangsan| 17| [REDACTED]
|1970-01-01 08:00:00| 2| lisi| 36| [REDACTED]

4 rows in set. Elapsed: 0.004 sec.
```

----End

### 2.17.3.16.5 SQL Example

Spark allows you to use SQL statements to create a Spark table and associate it with the ClickHouse table. When running the **Spark SQL CLI** command, you need to add `--jars clickhouse-jdbc-xxxjar`. For details about the actual JAR package name, see [Using Spark to Write Data to ClickHouse](#). The syntax for creating a table is as follows:

```
create table spark_cktbo1 using org.apache.spark.sql.jdbc options(url
"jdbcurl", user "userName", ssl "true", isCheckConnection "true", sslMode
```

```
"none", driver "ru.yandex.clickhouse.ClickHouseDriver", dbtable  
"clickHouseDBName.clickHouseTableName");
```

 NOTE

- **jdbcurl** is the ClickHouse JDBC connection string, for example, **jdbc:clickhouse://{IP address}:{Port number}**. The port number is **21425**.
- **clickHouseDBName** is the name of the ClickHouse database to be created, for example, **ckdb001**.
- **clickHouseTableName** is the name of the ClickHouse table to be created, for example, **ckdb01**.
- **userName** indicates the prepared username.

After the ClickHouse table is created, you can read, write, and query the **spark\_cktb01** table to perform the same operations on the ClickHouse table.

## 2.17.4 Commissioning the Application

### 2.17.4.1 Commissioning Applications on Windows

#### 2.17.4.1.1 Compiling and Running Applications

##### Scenario

You can run applications in the Windows environment after application code development is complete. The procedures for running applications developed using Scala or Java are the same on IDEA.

 NOTE

- In the Windows environment, only the sample code for accessing Spark SQL using JDBC is provided.
- Ensure that the Maven image repository of the SDK in the Huawei image site has been configured for Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

##### Procedure

**Step 1** Obtain the sample code.

Download the Maven project source code and configuration file of the sample project. For details, see [Obtaining Sample Projects](#).

Import the sample code to IDEA.

**Step 2** Obtain configuration files.

Obtain the files from the cluster client. Download the **hive-site.xml** and **spark-defaults.conf** files from **\$SPARK\_HOME/conf** to a local directory.

**Step 3** Upload data to HDFS.

1. Create a data text file on Linux and save the following data to the data file:

```
Miranda,32  
Karlie,23  
Candice,27
```

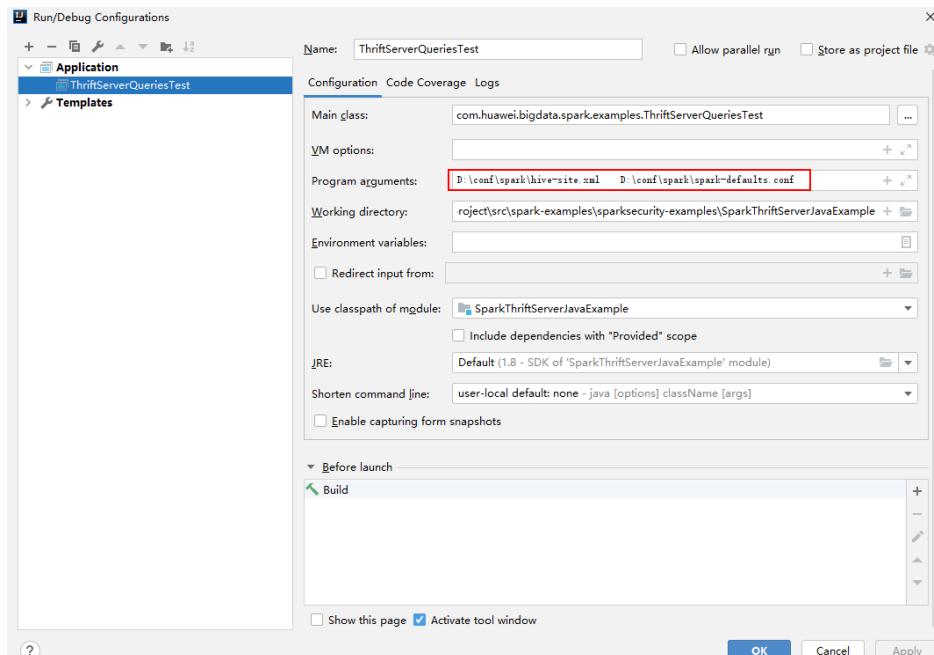
2. On the HDFS client running the Linux OS, run the **hadoop fs -mkdir /data** command (or the **hdfs dfs** command) to create a directory.
3. On the HDFS client running the Linux OS, run the **hadoop fs -put data /data** command to upload the data file.

**Step 4** Configure related parameters in the sample code.

Change the SQL statement for loading data to **LOAD DATA INPATH 'hdfs:/data/data'** INTO TABLE CHILD.

```
ArrayList<String> sqlList = new ArrayList<>();
sqlList.add("CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY"
    + " ','");
sqlList.add("LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD");
sqlList.add("SELECT * FROM child");
sqlList.add("DROP TABLE child");
executeSql(url, sqlList);
```

**Step 5** Add running parameters to the **hive-site.xml** and **spark-defaults.conf** files when the application is running.



**Step 6** Run the application.

----End

#### 2.17.4.1.2 View Debugging Results

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/apache/logging/log4j/log4j-slf4j-impl/2.6.2/log4j-slf4j-
impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/D:/mavenlocal/org/slf4j/slf4j-log4j12/1.7.30/slf4j-log4j12-1.7.30.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to
the console.
---- Begin executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',' ----
Result
---- Done executing sql: CREATE TABLE IF NOT EXISTS CHILD (NAME STRING, AGE INT) ROW FORMAT
```

```
DELIMITED FIELDS TERMINATED BY ',' ----
---- Begin executing sql: LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD ----
Result
---- Done executing sql: LOAD DATA INPATH 'hdfs:/data/data' INTO TABLE CHILD ----
---- Begin executing sql: SELECT * FROM child ----
NAME AGE
Miranda 32
Karlie 23
Candice 27
---- Done executing sql: SELECT * FROM child ----
---- Begin executing sql: DROP TABLE child ----
Result
---- Done executing sql: DROP TABLE child ----

Process finished with exit code 0
```

## 2.17.4.2 Commissioning an Application in Linux

### 2.17.4.2.1 Compiling and Running the Application

#### Scenario

After the program codes are developed, you can upload the codes to the Linux client for running. The running procedures of applications developed in Scala or Java are the same.

#### NOTE

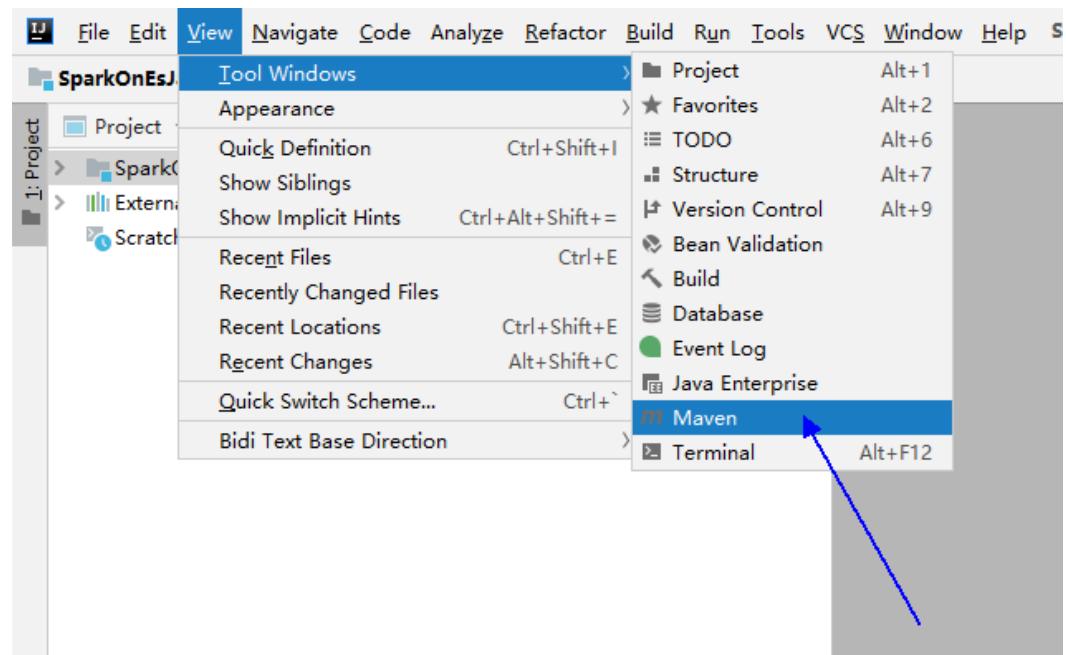
- The Spark application developed in Python does not need to build Artifacts as a jar. You just need to copy the sample projects to the compiler.
- It is needed to ensure that the version of Python installed on the worker and driver is consistent, otherwise the following error will be reported: "Python in worker has different version %s than that in driver %s."
- Ensure that Maven image repository of the SDK in the Huawei image site has been configured in Maven. For details, see [Configuring Huawei Open-Source Mirrors](#).

#### Procedure

##### Step 1 In the IntelliJ IDEA, open the Maven tool window.

On the main page of the IDEA, choose **View > Tool Windows > Maven** to open the Maven tool window.

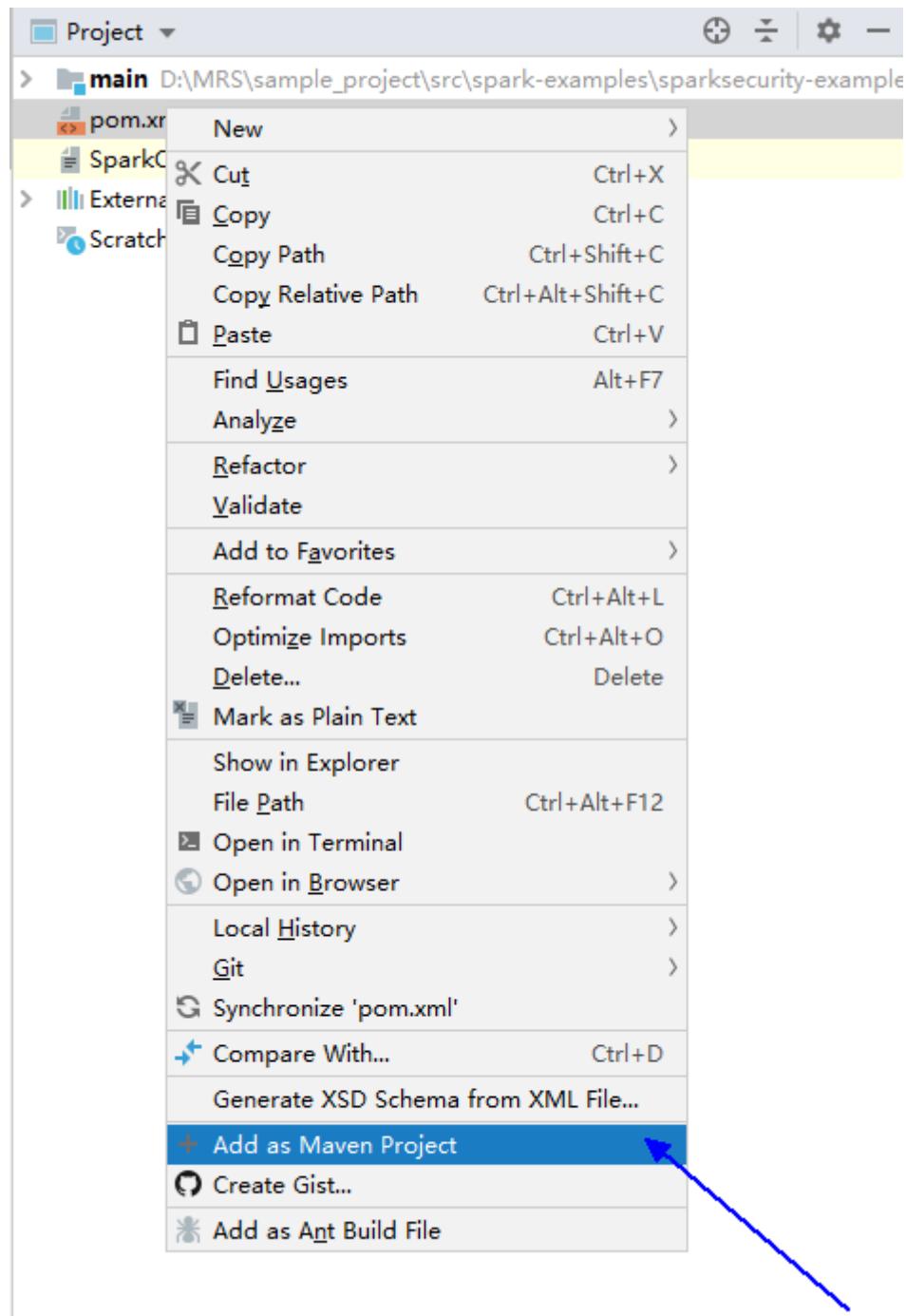
Figure 2-297 Opening the Maven tool window



If the project is not imported using Maven, perform the following operations:

Right-click the **pom** file in the sample code project and choose **Add as Maven Project** from the shortcut menu to add a Maven project.

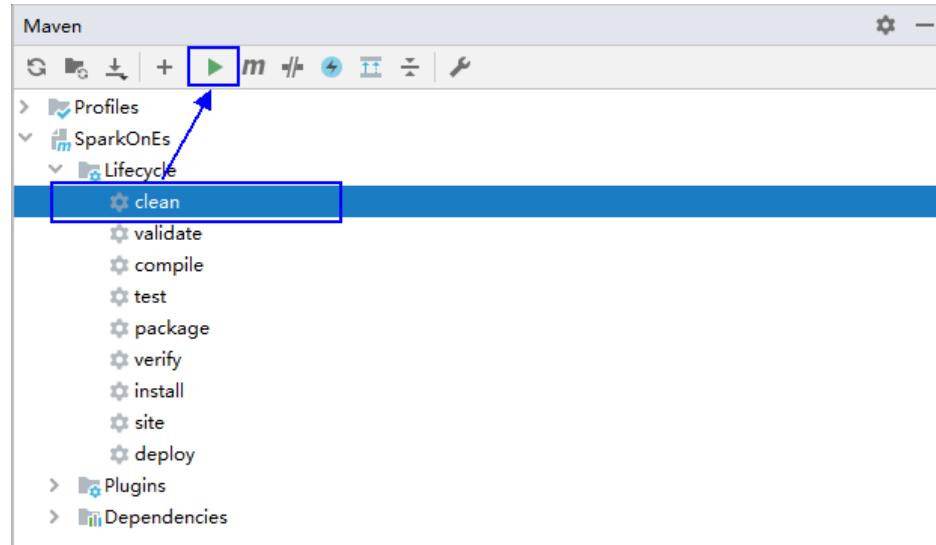
Figure 2-298 Adding a Maven project



**Step 2** Use Maven to generate a JAR file.

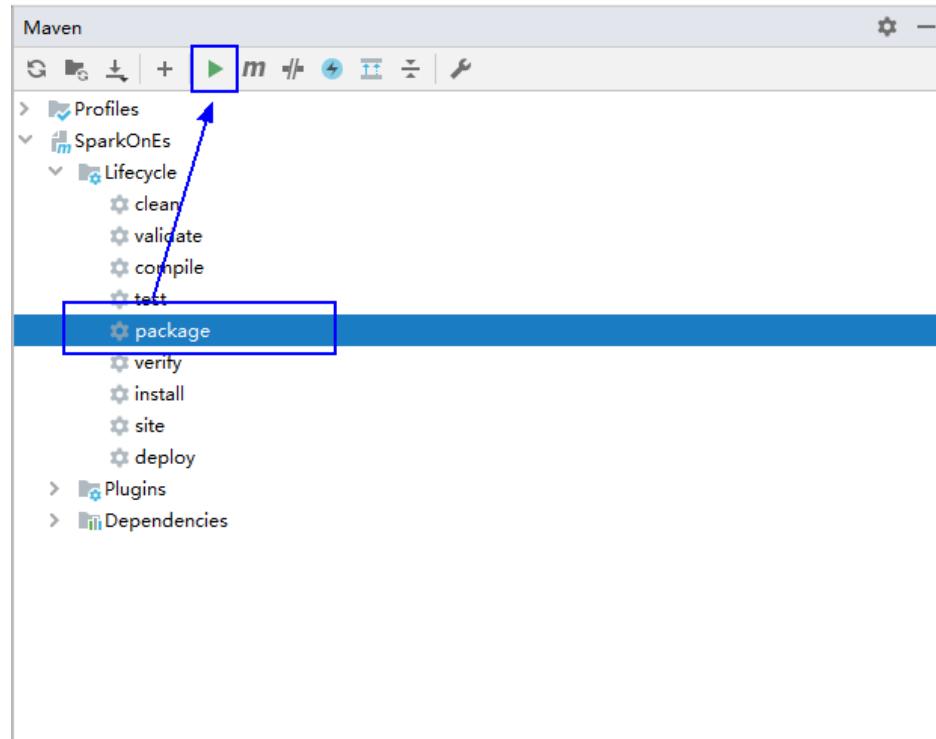
1. In the Maven tool window, select **clean** from **Lifecycle** to execute the Maven building process.

**Figure 2-299** Selecting **clean** from **Lifecycle** and execute the Maven building process



2. In the Maven tool window, select **package** from **Lifecycle** and execute the Maven building process.

**Figure 2-300** Selecting **package** from **Lifecycle** and execute the Maven build process.



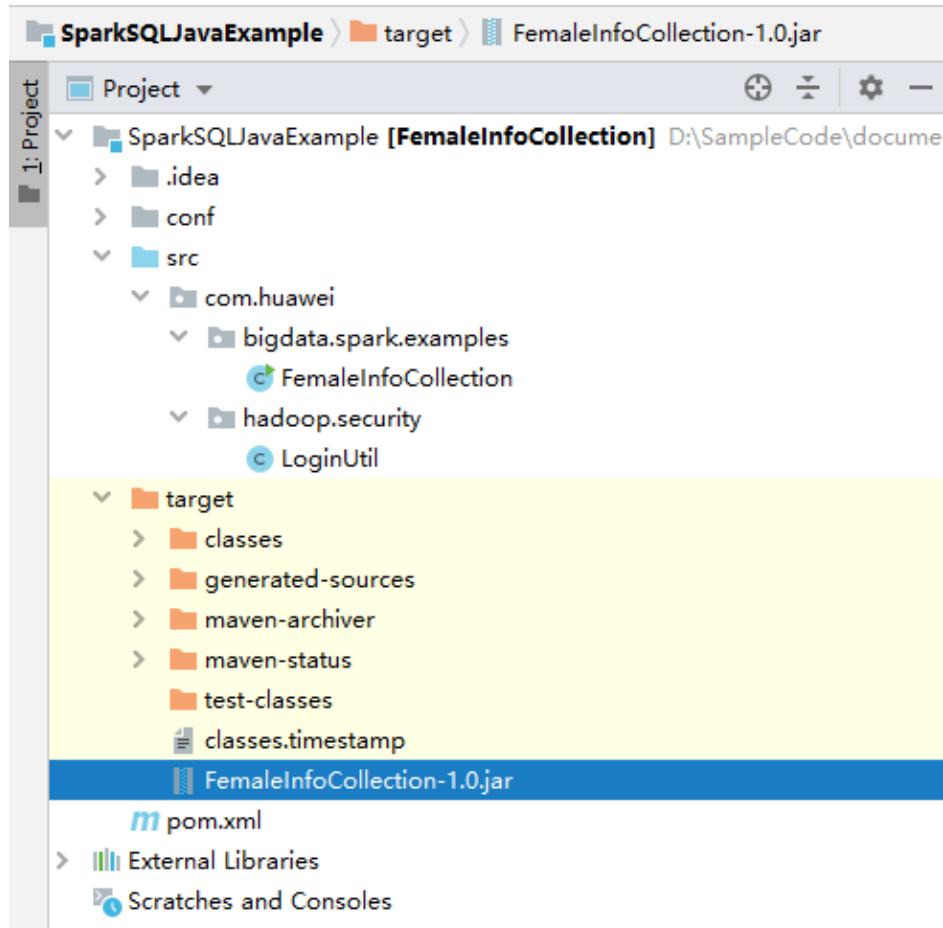
If the following information is displayed in **Run:**, the packaging is successful.

Figure 2-301 Packaging success message

```
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FemaleInfoCollection ---
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ FemaleInfoCollection ---
[INFO] Building jar: D:\SampleCode\document\VT\code\sparksecurity-examples\SparkSQLJavaExample\target\FemaleInfoCollection-1.0.jar
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 19.427 s
[INFO] Finished at: 2020-09-21T11:17:31+08:00
[INFO] -----
```

3. You can obtain the JAR package from the target folder in the project directory.

Figure 2-302 Obtaining the JAR Package



- Step 3** Copy the JAR file generated in **Step 2** (for example, **CollectFemaleInfo.jar**) to the Spark operating environment (that is, the Spark client), for example, **/opt/female**. Run the Spark application. For details about the example application, see [Developing the Project](#).

**⚠ CAUTION**

Do not restart the HDFS service or all DataNode instances during Spark job running. Otherwise, the job may fail and some JobHistory data may be lost.

----End

### 2.17.4.2.2 Checking the Commissioning Result

#### Scenario

After a Spark application is run, you can check the running result through one of the following methods:

- Viewing the command output.
- Logging in to the Spark web UI.
- Viewing Spark logs.

#### Procedure

- **Check the operating result data of the Spark application.**

The data storage directory and format are specified by users in the Spark application. You can obtain the data in the specified file.

- **Check the status of the Spark application.**

The Spark contains the following two web UIs:

- The Spark UI displays the status of applications being executed.

The Spark UI contains the **Spark Jobs**, **Spark Stages**, **Storage**, **Environment**, and **Executors** parts. Besides these parts, **Streaming** is displayed for the Streaming application.

On the YARN web UI, find the Spark application. Click **ApplicationMaster** in the last column of the application information. The Spark UI page is displayed.

- The History Server UI displays the status of all Spark applications.

The History Server UI displays information such as the application ID, application name, start time, end time, execution time, and user to whom the application belongs. After the application ID is clicked, the Spark UI of the application is displayed.

- **View Spark logs to learn application running conditions.**

The logs of Spark offer immediate visibility into application running conditions. You can adjust application programs based on the logs. Log related information can be referenced to MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)"Component Operation Guide" > "Using Spark" > "Spark Logs"

### 2.17.5 More Information

#### 2.17.5.1 Common APIs

##### 2.17.5.1.1 Java

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- JavaSparkContext: external interface of Spark, which is used to provide the functions of Spark for Java applications that invoke this class, for example, connecting Spark clusters and generating RDDs, accumulations, and broadcasts. It functions as a container.
- SparkConf: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- JavaRDD: class used to define the JavaRDD in the Java application, which functions like the RDD (Resilient Distributed Dataset) class of Scala.
- JavaPairRDD: indicates the JavaRDD in the key-value format. This class provides methods such as groupByKey and reduceByKey.
- Broadcast: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- StorageLevel: data storage levels, including memory (MEMORY\_ONLY), disk (DISK\_ONLY), and memory+disk (MEMORY\_AND\_DISK).

The JavaRDD supports two types of operations, transformation and action. [Table 2-161](#) and [Table 2-162](#) show the common methods.

**Table 2-161** Transformation

Method	Description
<R> JavaRDD<R> map(Function<T,R> f)	Returns a new RDD by applying a function to all elements of this RDD.
JavaRDD<T> filter(Function<T,Boolean> f)	Invokes Function on all elements of the RDD and returns the element that is <b>true</b> .
<U> JavaRDD<U> flatMap(FlatMapFunction<T,U> f)	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
JavaRDD<T> sample(boolean withReplacement, double fraction, long seed)	Returns a sampled subset of this RDD.
JavaRDD<T> distinct(int numPartitions)	Returns a new RDD containing the distinct elements in this RDD.
JavaPairRDD<K,Iterable<V>> groupByKey(int numPartitions)	Returns (K,Seq[V]) and combines the values of the same key to a set.

Method	Description
JavaPairRDD<K,V> reduceByKey(Function 2<V,V,V> func, int numPartitions)	Invokes Function on the values of the same key.
JavaPairRDD<K,V> sortByKey(boolean ascending, int numPartitions)	Sorts by key in ascending or descending order. Ascending is of the boolean type.
JavaPairRDD<K,scala.T uple2<V,W>> join(JavaPairRDD<K,W > other)	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numTasks</b> indicates the number of concurrent tasks.
JavaPairRDD<K,scala.T uple2<Iterable<V>,Iter able<W>>> cogroup(JavaPairRDD <K,W> other, int numPartitions)	Returns the dataset of <K,scala.Tuple2<Iterable<V>,Iterable<W>>> when the (K,V) and (K,W) datasets exist. <b>numTasks</b> indicates the number of concurrent tasks.
JavaPairRDD<T,U> cartesian(JavaRDDLik e<U,> other)	Returns the Cartesian product of the RDD and other RDDs.

**Table 2-162 Action**

Method	Description
T reduce(Function2<T,T, T> f)	Invokes Function2 on elements of the RDD.
java.util.List<T> collect()	Returns an array that contains all of the elements in this RDD.
long count()	Returns the number of elements in the RDD.
T first()	Returns the first element in this RDD.
java.util.List<T> take(int num)	Returns the first <i>n</i> elements in this RDD.
java.util.List<T> takeSample(boolean withReplacement, int num, long seed)	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.

Method	Description
void saveAsTextFile(String path, Class<? extends org.apache.hadoop.io. compress.Compressio nCodec> codec)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
java.util.Map<K, Object > countByKey()	Counts the appearance times of each key.
void foreach(VoidFunction <T> f)	Applies a function f to all elements of this RDD.
java.util.Map<T, Long> countByValue()	Counts the times that each element of the RDD occurs.

**Table 2-163** New APIs of Spark core

API	Description
public java.util.concurrent.atomic.AtomicBool ean isSparkContextDown()	Determines whether sparkContext is shut down completely. The initial value is <b>false</b> .  The value <b>true</b> indicates that sparkContext is shut down completely. The value <b>false</b> indicates that sparkContext is not shut down.  For example, <b>jsc.sc().isSparkContextDown().get() == true</b> indicates that sparkContext is shut down completely.

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- JavaStreamingContext: main entrance of the Spark Streaming, which is used to provide methods for creating the DStream. Intervals by batch need to be set in the input parameter.
- JavaDStream: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- JavaPairDStream: interface of KV DStream, which is used to provide the reduceByKey and join operations.
- JavaReceiverInputDStream<T>: specifies any inflow accepting data from the network.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 2-164** Spark Streaming methods

Method	Description
JavaReceiverInputDStream<java.lang.String> socketStream(java.lang.String hostname,int port)	Creates an inflow to receive data from the corresponding hostname and port through the TCP socket. Received data is resolved to the UTF8 format. The default storage level is the Memory+Disk.
JavaDStream<java.lang.String> textFileStream(java.lang.String directory)	Creates an inflow to detect new files compatible with the Hadoop file system, and read it as a text file. The directory of the input parameter is an HDFS directory.
void start()	Starts the Spark Streaming computing.
void awaitTermination()	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
void stop()	Stops the Spark Streaming computing.
<T> JavaDStream<T> transform(java.util.List<JavaDStream<?>> dstreams,Function2<java.util.List<JavaRDD<?>>,Time,JavaRDD<T>> transformFunc)	Performs the Function operation on each RDD to obtain a new DStream. In this function, the sequence of the JavaRDDs must be the same as the corresponding DStreams.
<T> JavaDStream<T> union(JavaDStream<T> first,java.util.List<JavaDStream<T>> rest)	Creates a unified DStream from multiple DStreams with the same type and sliding window.

**Table 2-165** Spark Streaming enhancement interface

Method	Description
JAVADStreamKafkaWriter.writeToKafka()	Writes data from the DStream into Kafka in batch.
JAVADStreamKafkaWriter.writeToKafkaBySingle()	Writes data from DStream into Kafka one by one.

## Spark SQL Common Interfaces

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of the Spark SQL function and DataFrame.
- DataFrame: a distributed dataset organized by naming columns.
- DataFrameReader: interface for loading the DataFrame from external storage systems.
- DataFrameStatFunctions: implementation the statistic function of the DataFrame.
- UserDefinedFunction: function defined by users.

Common Actions methods are described in the following table.

**Table 2-166** Spark SQL methods

Method	Description
Row[] collect()	Returns an array containing all DataFrame columns.
long count()	Returns the number of DataFrame rows.
DataFrame describe(java.lang.String... cols)	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
Row first()	Returns the first row.
Row[] head(int n)	Returns the first <i>n</i> rows.
void show()	Displays the first 20 rows in table.
Row[] take(int n)	Returns the first <i>n</i> rows in the DataFrame.

**Table 2-167** Basic DataFrame Functions

Method	Description
void explain(boolean extended)	Prints the logical plan and physical plan of the SQL.
void printSchema()	Prints the schema information to the console.
registerTempTable	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
DataFrame toDF(java.lang.String... colNames)	Returns a DataFrame whose columns are renamed.
DataFrame sort(java.lang.String sortCol,java.lang.String... sortCols)	Sorts columns in ascending or descending orders based on different columns.
GroupedData rollup(Column... cols)	Performs multiple-dimension crankback on the specified columns in the DataFrame.

### 2.17.5.1.2 Scala

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- `SparkContext`: external interface of Spark, which is used to provide the functions of Spark for Scala applications that invoke this class, for example, connecting Spark clusters and generating RDDs.
- `SparkConf`: Spark application configuration class, which is used to configure the application name, execution model, and executor memory.
- `Resilient Distributed Dataset (RDD)`: defines the RDD class in the Spark application. The class provides the data collection operation methods, such as `map` and `filter`.
- `PairRDDFunctions`: provides computation operations for the RDD data of the key-value pair, such as `groupByKey`.
- `Broadcast`: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- `StorageLevel`: data storage levels, including memory (`MEMORY_ONLY`), disk (`DISK_ONLY`), and memory+disk (`MEMORY_AND_DISK`).

RDD supports two types of operations, transformation and action. [Table 2-168](#) and [Table 2-169](#) describe the common methods.

**Table 2-168** Transformation

Method	Description
<code>map[U](f: (T) =&gt; U): RDD[U]</code>	Returns a new RDD by applying a function to all elements of this RDD.
<code>filter(f: (T) =&gt; Boolean): RDD[T]</code>	Invokes the <code>f</code> method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.
<code>flatMap[U](f: (T) =&gt; TraversableOnce[U]) (implicit arg0: ClassTag[U]): RDD[U]</code>	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
<code>sample(withReplacement: Boolean, fraction: Double, seed: Long = Utils.random.nextLong()): RDD[T]</code>	Returns a sampled subset of this RDD.
<code>union(other: RDD[T]): RDD[T]</code>	Returns a new RDD, contains source RDD and the group of RDD's elements.

Method	Description
distinct([numPartition s: Int]): RDD[T]	Returns a new RDD containing the distinct elements in this RDD.
groupByKey(): RDD[(K, Iterable[V])]	Returns (K,Iterable[V]) and combines the values of the same key to a set.
reduceByKey(func: (V, V) => V[, numPartitions: Int]): RDD[(K, V)]	Invokes func on the values of the same key.
sortByKey(ascending: Boolean = true, numPartitions: Int = self.partitions.length): RDD[(K, V)]	Sorts by key in ascending or descending order. Ascending is of the boolean type.
join[W](other: RDD[(K, W)][, numPartitions: Int]): RDD[(K, (V, W))]	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cogroup[W](other: RDD[(K, W)], numPartitions: Int): RDD[(K, (Iterable[V], Iterable[W]))]	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(T, U)]	Returns the Cartesian product of the RDD and other RDDs.

**Table 2-169 Action**

API	Description
reduce(f: (T, T) => T):	Invokes f on elements of the RDD.
collect(): Array[T]	Returns an array that contains all of the elements in this RDD.
count(): Long	Returns the number of elements in the dataset.
first(): T	Returns the first element in this RDD.
take(num: Int): Array[T]	Returns the first <i>n</i> elements in this RDD.

API	Description
takeSample(withReplacement: Boolean, num: Int, seed: Long = Utils.random.nextLong()): Array[T]	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.
saveAsTextFile(path: String): Unit	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
saveAsSequenceFile(path: String, codec: Option[Class[_ <: CompressionCodec]] = None): Unit	This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system.
countByKey(): Map[K, Long]	Counts the appearance times of each key.
foreach(func: (T) => Unit): Unit	Applies a function f to all elements of this RDD.
countByValue() (implicit ord: Ordering[T] = null): Map[T, Long]	Counts the times that each element of the RDD occurs.

**Table 2-170** New APIs of Spark core

API	Description
isSparkContextDown:AtomicBoolean	Determines whether sparkContext is shut down completely. The initial value is <b>false</b> .  The value <b>true</b> indicates that sparkContext is shut down completely. The value <b>false</b> indicates that sparkContext is not shut down.  For example, <b>sc.isSparkContextDown.get() == true</b> indicates that sparkContext is shut down completely.

## Spark Streaming Common Interfaces

Spark Streaming mainly uses the following classes:

- StreamingContext: main entrance of the Spark Streaming, which is used to provide methods for creating the DStream. Intervals by batch need to be set in the input parameter.
- dstream.DStream: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.
- dstream.PairDStreamFunctions: the DStream of key-value, common operations are groupByKey and reduceByKey.

The cooperated Java APIs of Spark Streaming are JavaStreamingContext, JavaDStream, JavaPairDStream.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 2-171** Spark Streaming methods

Method	Description
socketTextStream(hostName: String, port: Int, storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2): ReceiverInputDStream[String]	Creates an input stream from the TCP source host:port.
start():Unit	Starts Spark Streaming computing.
awaitTermination(timeout: long):Unit	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit	Stops the Spark Streaming computing.
transform[T](dstreams: Seq[DStream[_]], transformFunc: (Seq[RDD[_]], Time) ? RDD[T])(implicit arg0: ClassTag[T]): DStream[T]	Performs the function operation on each RDD to obtain a new DStream.
UpdateStateByKey(func)	Updates the status of DStream. To use this method, you need to define the state and state update functions.
window(windowLength, slideInterval)	Generates a new DStream by batch calculating according to the window of the source DStream.

Method	Description
countByWindow(windowLength, slideInterval)	Returns the number of sliding window elements in the stream.
reduceByWindow(func, windowLength, slideInterval)	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
join(otherStream, [numTasks])	Performs a join operation between different Spark Streamings.
DStreamKafkaWriter.writeToKafka()	Writes data from the DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from the DStream into Kafka one by one.

**Table 2-172** Spark Streaming enhancement interface

Method	Description
DStreamKafkaWriter.writeToKafka()	Writes data from the DStream into Kafka in batch.
DStreamKafkaWriter.writeToKafkaBySingle()	Writes data from the DStream into Kafka one by one.

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- SQLContext: main entrance of the Spark SQL function and DataFrame.
- DataFrame: a distributed dataset organized by naming columns.
- HiveContext: main entrance for obtaining data stored in Hive.

**Table 2-173** Common Actions methods

Method	Description
collect(): Array[Row]	Returns an array containing all DataFrame columns.
count(): Long	Returns the number of DataFrame rows.
describe(cols: String*): DataFrame	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.

Method	Description
first(): Row	Returns the first row.
Head(n:Int): Row	Returns the first <i>n</i> rows.
show numRows: Int, truncate: Boolean): Unit	Displays DataFrame in a table.
take(n:Int): Array[Row]	Returns the first <i>n</i> rows in the DataFrame.

**Table 2-174** Basic DataFrame Functions

Method	Description
explain(): Unit	Prints the logical plan and physical plan of the SQL.
printSchema(): Unit	Prints the schema information to the console.
registerTempTable(tableName: String): Unit	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
toDF(colNames: String*): DataFrame	Returns a DataFrame whose columns are renamed.

### 2.17.5.1.3 Python

To avoid API compatibility or reliability issues after updates to the open-source Spark, it is advisable to use APIs of the version you are currently using. For details about APIs, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

## Spark Core Common Interfaces

Spark mainly uses the following classes:

- pyspark.SparkContext: external API of Spark. It provides the functions of Spark for Python applications that invoke this class, for example, connecting Spark clusters, creating RDDs, and broadcasting variables.
- pyspark.SparkConf: Spark application configuration class. It is used to set an application name, execution mode, and executor memory.
- pyspark.RDD: class used to define the RDD in the Spark application. The class provides the data collection operation methods, such as map and filter.
- pyspark.Broadcast: broadcast variable class. This class retains one read-only variable, and caches it on each machine, instead of saving a copy for each task.
- pyspark.StorageLevel: data storage levels, including memory (MEMORY\_ONLY), disk (DISK\_ONLY), and memory+disk (MEMORY\_AND\_DISK).

- `pyspark.sql.SQLContext`: Main entry point for SparkSQL functionality. It can be used to create DataFrame, register DataFrame as a table, and execute SQL on a table.
- `pyspark.sql.DataFrame`: A distributed collection of data grouped into named columns. A DataFrame is equivalent to a relational table in Spark SQL, and can be created using various functions in SQLContext.
- `pyspark.sql.DataFrameNaFunctions`: Functionality for working with missing data in DataFrame.
- `pyspark.sql.DataFrameStatFunctions`: function in DataFrame for statistics. It calculates the variance between columns and sample covariance.

The RDD supports two types of operations, transformation and action. [Table 2-175](#) and [Table 2-176](#) show the common methods.

**Table 2-175** Transformation

Method	Description
<code>map(f, preservesPartitioning=False)</code>	Returns a new RDD by applying a function to all elements of this RDD.
<code>filter(f)</code>	Invokes the Func method for all RDD elements to generate a satisfied data set that is returned in the form of RDD.
<code>flatMap(f, preservesPartitioning=False)</code>	Returns a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
<code>sample(withReplacement, fraction, seed=None)</code>	Returns a sampled subset of this RDD.
<code>union(rdds)</code>	Returns a new RDD, contains source RDD and the group of RDD's elements.
<code>distinct([numPartitions: Int]): RDD[T]</code>	Returns a new RDD containing the distinct elements in this RDD.
<code>groupByKey(): RDD[(K, Iterable[V])]</code>	Returns (K,Iterable[V]) and combines the values of the same key to a set.
<code>reduceByKey(func, numPartitions=None)</code>	Invokes Func on the values of the same key.
<code>sortByKey(ascending=True, numPartitions=None, keyfunc=function &lt;lambda&gt;)</code>	Sorts by key in ascending or descending order. Ascending is of the boolean type.
<code>join(other, numPartitions)</code>	Returns the dataset of (K,(V,W)) when the (K,V) and (K,W) datasets exist. <b>numPartitions</b> indicates the number of concurrent tasks.

Method	Description
cogroup(other, numPartitions)	Returns the dataset of (K, (Iterable[V], Iterable[W])) when the (K,V) and (K,W) datasets of two key-value pairs exist. <b>numPartitions</b> indicates the number of concurrent tasks.
cartesian(other)	Returns the Cartesian product of the RDD and other RDDs.

**Table 2-176 Action**

API	Description
reduce(f)	Invokes Func on elements of the RDD.
collect()	Returns an array that contains all of the elements in this RDD.
count()	Returns the number of elements in the dataset.
first()	Returns the first element in this RDD.
take(num)	Returns the first num elements of the RDD.
takeSample(withReplacement, num, seed)	Samples the dataset randomly and returns a dataset of num elements. <b>withReplacement</b> indicates whether replacement is used.
saveAsTextFile(path, compressionCodecClass)	Writes the dataset to a text file, HDFS, or file system supported by HDFS. Spark converts each record to a row of records and then writes it to the file.
saveAsSequenceFile(path, compressionCodecClass=None)	This API can be used only on the key-value pair, and then it generates SequenceFile and writes the file to the local or Hadoop file system.
countByKey()	Counts the appearance times of each key.
foreach(func)	Runs the function on each element of the dataset.
countByValue()	Counts the times that each value of the RDD occurs.

## Spark Streaming Common APIs

Spark Streaming mainly uses the following classes:

- `pyspark.streaming.StreamingContext`: main entrance of Spark Streaming. It provides methods for creating the DStream. A batch interval needs to be set in the input parameter.
- `pyspark.streaming.DStream`: a type of data which indicates the RDDs continuous sequence. It indicates the continuous data flow.

- `dstream.PairDStreamFunctions`: the DStream of key-value, common operations are `groupByKey` and `reduceByKey`.  
The cooperated Java APIs of Spark Streaming are `JavaStreamingContext`, `JavaDStream`, `JavaPairDStream`.

Common methods of Spark Streaming are the same as those of Spark Core. The following table describes some special Spark Streaming methods.

**Table 2-177** Spark Streaming common interfaces

Method	Description
<code>socketTextStream(hostname, port, storageLevel)</code>	Creates an input stream from the TCP source host:port.
<code>start()</code>	Starts Spark Streaming computing.
<code>awaitTermination(timeout)</code>	Terminates the await of the process, which is similar to pressing <b>Ctrl+C</b> .
<code>stop(stopSparkContext, stopGraceFully)</code>	Stops Spark Streaming computing. <b>stopSparkContext</b> is used to determine whether SparkContext needs to be terminated. <b>StopGracefully</b> is used to determine whether to wait for all the received data to be processed.
<code>UpdateStateByKey(func)</code>	Updates the status of DStream. To use this method, you need to define the state and state update functions.
<code>window(windowLength, slideInterval)</code>	Generates a new DStream by batch calculating according to the window of the source DStream.
<code>countByWindow(windowLength, slideInterval)</code>	Returns the number of sliding window elements in the stream.
<code>reduceByWindow(func, windowLength, slideInterval)</code>	When the key-value pair of DStream is invoked, a new key-value pair of DStream is returned. The value of each key is obtained by aggregating the reduce function in batches in the sliding window.
<code>join(other,numPartitions)</code>	Performs a join operation between different Spark Streamings.

## SparkSQL Common Interfaces

Spark SQL mainly uses the following classes:

- `pyspark.sql.SQLContext`: main entrance of the Spark SQL function and `DataFrame`.
- `pyspark.sql.DataFrame`: a distributed dataset organized by naming columns.

- `pyspark.sql.HiveContext`: main entrance for obtaining data stored in Hive.
- `pyspark.sql.DataFrameStatFunctions`: Functionality for statistic functions with DataFrame.
- `pyspark.sql.functions`: A collection of builtin functions.
- `pyspark.sql.Window`: Utility functions for defining window in DataFrames.

**Table 2-178** Spark SQL common Actions

Method	Description
<code>collect()</code>	Returns an array containing all DataFrame columns.
<code>count()</code>	Returns the number of DataFrame rows.
<code>describe()</code>	Counts the statistic information, including the counting, average value, standard deviation, minimum value and maximum value.
<code>first()</code>	Returns the first row.
<code>head(n)</code>	Returns the first <i>n</i> rows.
<code>show()</code>	Displays DataFrame in a table.
<code>take(num)</code>	Returns the first num rows in the DataFrame.

**Table 2-179** Basic DataFrame Functions

Method	Description
<code>explain()</code>	Prints the logical plan and physical plan of the SQL.
<code>printSchema()</code>	Prints the schema information to the console.
<code>registerTempTable(name)</code>	Registers the DataFrame as a temporary table, whose period is bound to the SQLContext.
<code>toDF()</code>	Returns a DataFrame whose columns are renamed.

#### 2.17.5.1.4 REST API

##### Function Description

The Spark REST API presents some web UI metrics in the JSON format, providing users with a simpler method to create new visualization and monitoring tools. The REST API can be used to query information about running and historical applications. The open-source Spark REST API allows users to query information about Jobs, Stages, Storage, Environment, and Executors. In the FusionInsight version, the REST API used to query SQL, JDBC/ODBC server, and Streaming information is added. For more information about the open-source REST API, see <https://spark.apache.org/docs/3.3.1/monitoring.html#rest-api>.

## Preparing Running Environment

Install the FusionInsight client. Install a FusionInsight client on the node. For example, install the client in the **/opt/hadoopclient** directory.

## REST API

You can use the following commands to dodge the REST API filter and directly obtain the application information:

### NOTICE

In normal mode, the JobHistory supports only the HTTP protocol. Therefore, use the HTTP protocol in the URL of the following command.

- Obtaining information about all applications on the JobHistory node:

- Command:  

```
curl http://192.168.227.16:22500/api/v1/applications?mode=monitoring --insecure
```

**192.168.227.16** indicates the service IP address of the JobHistory node and **22500** indicates the port number of the JobHistory node.

- Command output:

```
[ {  
    "id" : "application_1517290848707_0008",  
    "name" : "Spark Pi",  
    "attempts" : [ {  
        "startTime" : "2018-01-30T15:05:37.433CST",  
        "endTime" : "2018-01-30T15:06:04.625CST",  
        "lastUpdated" : "2018-01-30T15:06:04.848CST",  
        "duration" : 27192,  
        "sparkUser" : "sparkuser",  
        "completed" : true,  
        "startTimeEpoch" : 1517295937433,  
        "endTimeEpoch" : 1517295964625,  
        "lastUpdatedEpoch" : 1517295964848  
    } ]  
}, {  
    "  
    id" : "application_1517290848707_0145",  
    "name" : "Spark shell",  
    "attempts" : [ {  

```

- Analysis:

With this command, you can query information about all Spark applications in the current cluster, including running applications and the completed applications. **Table 2-180** provides information about each application.

**Table 2-180** Parameter description

Parameter	Description
id	Application ID.
name	Application name.
attempts	Attempts executed by the application, including the attempt start time, attempt end time, user who initiates the attempts, and status indicating whether the attempts are completed.

- Obtaining information about a specific application on the JobHistory node:

- Command:

```
curl http://192.168.227.16:22500/api/v1/applications/application_1517290848707_0008?  
mode=monitoring --insecure
```

**192.168.227.16** indicates the service IP address of the JobHistory node, **22500** indicates the port number of the JobHistory node, and **application\_1517290848707\_0008** indicates the application ID.

- Command output:

```
{  
    "id" : "application_1517290848707_0008",  
    "name" : "Spark Pi",  
    "attempts" : [ {  
        "startTime" : "2018-01-30T15:05:37.433CST",  
        "endTime" : "2018-01-30T15:06:04.625CST",  
        "lastUpdated" : "2018-01-30T15:06:04.848CST",  
        "duration" : 27192,  
        "sparkUser" : "sparkuser",  
        "completed" : true,  
        "startTimeEpoch" : 1517295937433,  
        "endTimeEpoch" : 1517295964625,  
        "lastUpdatedEpoch" : 1517295964848  
    } ]  
}
```

- Analysis:

With this command, you can query information about a Spark application. **Table 2-180** provides information about the application.

- Obtain the information about the Executor of a running application:

- Command of alive executors list:

```
curl http://192.168.169.84:26000/proxy/application_1478570725074_0046/api/v1/applications/  
application_1478570725074_0046/executors?mode=monitoring --insecure
```

- Command of all executors(alive&dead) list:

```
curl http://192.168.169.84:26000/proxy/application_1478570725074_0046/api/v1/applications/  
application_1478570725074_0046/allexecutors?mode=monitoring --insecure
```

**192.168.169.84** indicates the service IP address of the master node of the ResourceManager, **26000** indicates the port number of the ResourceManager, and **application\_1478570725074\_0046** indicates the application ID in YARN.

- Command output:

```
[{  
    "id" : "driver",  
    "hostPort" : "192.168.169.84:23886",  
    "status" : "RUNNING",  
    "appMasterHost" : "192.168.169.84",  
    "appMasterPort" : 23888  
}]
```

```
"isActive" : true,
"rddBlocks" : 0,
"memoryUsed" : 0,
"diskUsed" : 0,
"activeTasks" : 0,
"failedTasks" : 0,
"completedTasks" : 0,
"totalTasks" : 0,
"totalDuration" : 0,
"totalInputBytes" : 0,
"totalShuffleRead" : 0,
"totalShuffleWrite" : 0,
"maxMemory" : 278019440,
"executorLogs" : { }
}, {
"id" : "1",
"hostPort" : "192.168.169.84:23902",
"isActive" : true,
"rddBlocks" : 0,
"memoryUsed" : 0,
"diskUsed" : 0,
"totalCores" : 1,
"maxTasks" : 1,
"activeTasks" : 0,
"failedTasks" : 0,
"completedTasks" : 0,
"totalTasks" : 0,
"totalDuration" : 0,
"totalGCTime" : 139,
"totalInputBytes" : 0,
"totalShuffleRead" : 0,
"totalShuffleWrite" : 0,
"maxMemory" : 555755765,
"executorLogs" : {
  "stdout" : "https://XTJ-224:26010/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stdout?start=-4096",
  "stderr" : "https://XTJ-224:26010/node/containerlogs/
container_1478570725074_0049_01_000002/admin/stderr?start=-4096"
}
}
]
```

- Analysis:

With this command, you can query information about all executors including drivers of the current application. [Table 2-181](#) provides basic information about each executor.

**Table 2-181** Parameter description

Parameter	Description
id	Executor ID.
hostPort	IP address and port number of the node where the Executor resides. Format: <i>IP address:port number</i> .
executorLogs	Path to Executor logs.

## Enhanced REST API

- SQL related: obtaining all the SQL statements and those with the longest execution time.

- SparkUI command:  
`curl http://192.168.195.232:26000/proxy/application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/SQL?mode=monitoring --insecure`  
**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **26000** indicates the port number of the ResourceManager, and **application\_1476947670799\_0053** indicates the application ID in YARN.
- JobHistory command:  
`curl http://192.168.227.16:22500/api/v1/applications/application_1478570725074_0004/SQL?mode=monitoring --insecure`  
**192.168.227.16** indicates the service IP address of the JobHistory node, **22500** indicates the port number of the JobHistory node, and **application\_1478570725074\_0004** indicates the application ID.
- Command output:

The command output of the SparkUI and JobHistory commands is as follows:

```
{  
  "longestDurationOfCompletedSQL" : [ {  
    "id" : 0,  
    "status" : "COMPLETED",  
    "description" : "getCallSite at SQLExecution.scala:48",  
    "submissionTime" : "2016/11/08 15:39:00",  
    "duration" : "2 s",  
    "runningJobs" : [ ],  
    "successedJobs" : [ 0 ],  
    "failedJobs" : [ ]  
  },  
  "sqls" : [ {  
    "id" : 0,  
    "status" : "COMPLETED",  
    "description" : "getCallSite at SQLExecution.scala:48",  
    "submissionTime" : "2016/11/08 15:39:00",  
    "duration" : "2 s",  
    "runningJobs" : [ ],  
    "successedJobs" : [ 0 ],  
    "failedJobs" : [ ]  
  }]  
}
```

- Analysis:  
After running this command, you can obtain all the SQL statements executed by the current application (the **sqls** part of the command output) and the SQL statements with the longest execution time (the **longestDurationOfCompletedSQL** part of the command output). The information about each SQL statement is listed in [Table 2-182](#).

**Table 2-182** Parameter description

Parameter	Description
id	SQL statement ID.
status	Execution status of the SQL statement, which can be: running, completed, and failed.
runningJobs	Jobs that are being executed generated by the SQL statement.

Parameter	Description
successedJobs	Jobs that are successfully executed generated by the SQL statement.
failedJobs	Job that fails to be executed generated by the SQL statement.

- JDBC server related: obtaining the number of sessions, number of being-executed SQL statements, information about all sessions, and information about SQL statements.

- Command:

```
curl http://192.168.195.232:26000/proxy/application_1476947670799_0053/api/v1/applications/application_1476947670799_0053/sqlserver?mode=monitoring --insecure
```

**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **26000** indicates the port number of the ResourceManager, and **application\_1476947670799\_0053** indicates the application ID in YARN.

- Command output:

```
{
  "sessionNum" : 1,
  "runningSqlNum" : 0,
  "sessions" : [ {
    "user" : "spark",
    "ip" : "192.168.169.84",
    "sessionId" : "9dfec575-48b4-4187-876a-71711d3d7a97",
    "startTime" : "2016/10/29 15:21:10",
    "finishTime" : "",
    "duration" : "1 minute 50 seconds",
    "totalExecute" : 1
  }],
  "sqls" : [ {
    "user" : "spark",
    "jobId" : [ ],
    "groupId" : "e49ff81a-230f-4892-a209-a48abea2d969",
    "startTime" : "2016/10/29 15:21:13",
    "finishTime" : "2016/10/29 15:21:14",
    "duration" : "555 ms",
    "statement" : "show tables",
    "state" : "FINISHED",
    "detail" : "== Parsed Logical Plan ==\nShowTablesCommand None\n== Analyzed Logical Plan ==\n\nShowTablesCommand None\n== Cached Logical Plan ==\nShowTablesCommand None\n== Optimized Logical Plan ==\nShowTablesCommand None\n== Physical Plan ==\nExecutedCommand\nShowTablesCommand None\nCode Generation: true"
  }]
}
```

- Analysis:

After running this command, you can query the number of sessions in the current JDBC application, number of being-executed SQL statements, and information about all sessions and SQL statements. The information about each session is listed in [Table 2-183](#), and the information about each SQL statement is listed in [Table 2-184](#).

**Table 2-183** Session parameter description

Parameter	Description
user	User to whom the session connects.
ip	IP address of the node where the session resides.
sessionId	Session ID.
startTime	Time when the session starts the connection.
finishTime	Time when the session ends the connection.
duration	Connection duration of the session.
totalExecute	Number of SQL statements executed by the session.

**Table 2-184** SQL parameter description

Parameter	Description
user	User who executes the SQL statement.
jobId	IDs of jobs contained in the SQL statement.
groupId	ID of the group where the SQL statement resides.
startTime	Start time.
finishTime	End time.
duration	SQL statement execution duration.
statement	SQL statement.
detail	Logical/Physical plan.

- Streaming related: obtaining the average input frequency, average scheduling delay, average execution duration, and average value of the overall delay.
  - Command:  
`curl http://192.168.195.232:26000/proxy/application_1477722033672_0008/api/v1/applications/application_1477722033672_0008/streaming/statistics?mode=monitoring --insecure`  
**192.168.195.232** indicates the service IP address of the master node of the ResourceManager, **26000** indicates the port number of the ResourceManager, and **application\_1477722033672\_0008** indicates the application ID in YARN.

- Command:

```
{  
  "startTime" : "2018-12-25T08:58:10.836GMT",  
  "batchDuration" : 1000,  
  "numReceivers" : 1,  
  "numActiveReceivers" : 1,  
  "numInactiveReceivers" : 0,  
  "numTotalCompletedBatches" : 373,  
  "numRetainedCompletedBatches" : 373,  
  "numActiveBatches" : 0,  
  "numProcessedRecords" : 1,  
  "numReceivedRecords" : 1,  
  "avgInputRate" : 0.002680965147453083,  
  "avgSchedulingDelay" : 14,  
  "avgProcessingTime" : 47,  
  "avgTotalDelay" : 62  
}
```

- Analysis:

After running this command, you can query the average input frequency (unit: events/sec), average scheduling delay (unit: ms), average execution time (unit: ms), and average value of the total delay (unit: ms) of the current Streaming application.

## 2.17.5.2 Common CLIs

For details about how to use the Spark CLIs, visit the official website <http://spark.apache.org/docs/3.3.1/quick-start.html>.

### Common CLI

Common Spark CLIs are described as follows:

- **spark-shell**

It provides an easy way to learn APIs, which is similar to the tool for interactive data analysis. It supports two languages including Scala and Python. In the Spark directory, run the `./bin/spark-shell` command to log in the interactive interface of Scala, obtain data from the HDFS, and perform the RDD.

For example: a row of codes can count all words in a file.

```
scala> sc.textFile("hdfs://10.96.1.57:9000//wordcount_data.txt").flatMap(l => l.split(" ")).map(w => (w,1)).reduceByKey(_+_).collect()
```

- **spark-submit**

It is used to submit the Spark application to the Spark cluster for running and return the running results. The class, master, jar and input parameter need to be specified.

For example: Run the GroupByTest example in the jar. There are four input parameters and the specified running mode of the cluster is local single platform.

```
./bin/spark-submit --class org.apache.spark.examples.GroupByTest --  
master local[1] examples/jars/spark-examples_xxx.cbu.mrs.xxx.jar 6 10 10  
3
```

- **spark-sql**

It is used to run the Hive metadata service and query command lines in the local or cluster mode. If its logical plan needs to be queried, add "explain extended" before the SQL statement.

For example:

**Select key from src group by key**

- **run-example**

It is used to run or debug the default example in the Spark open-source community.

For example: Run the SparkPi.

**./run-example SparkPi 100**

### 2.17.5.3 JDBCServer Interface

#### Overview

JDBCServer is another HiveServer2 implementation in Hive. It uses the Spark structured query language (SQL) to process the SQL statements, providing higher performance than Hive.

JDBCServer is a JDBC interface. Users can use JDBC to connect to JDBCServer to access SparkSQL data. When JDBCServer is started, a SparkSQL application is started, and the clients connected through JDBC can share the resources of the SparkSQL application, that is, different users can share data. When JDBCServer is started, a listener is also enabled to wait for the JDBC client to submit the connection and query requests. Therefore, when configuring JDBCServer, you need to configure at least the host name and port number of JDBCServer. If you want to use Hive data, you also need to provide the URIs of Hive MetaStore.

JDBCServer starts a JDBC service on port 22550 of the installation node by default. (If you want to change the port, configure the **hive.server2.thrift.port** parameter.) You can connect to JDBCServer using Beeline or running the JDBC client code to run SQL statements.

For other information about the JDBCServer, visit the Spark official website: <http://spark.apache.org/docs/3.3.1/sql-programming-guide.html#distributed-sql-engine>.

#### Beeline

For details about the Beeline connection modes provided by the open source community, visit <https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients>.

```
sh CLIENT_HOME/spark/bin/beeline -u "jdbc:hive2://<zkNode_IP>:<zkNode_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver;"
```

#### NOTE

- In the preceding information, <zkNode1\_IP>:<zkNode1\_Port>,<zkNode2\_IP>:<zkNode2\_Port>,<zkNode3\_IP>:<zkNode3\_Port> indicates the ZooKeeper URL. Use commas (,) to separate multiple URLs, for example, 192.168.81.37:24002,192.168.195.232:24002,192.168.169.84:24002.
- **sparkthriftserver** indicates the ZooKeeper directory, where a random JDBCServer instance is connected to the client.

## JDBC Client Codes

Log in to JDBCServer by using the JDBC client codes and access the Spark SQL data. For details, see [Accessing the Spark SQL Through JDBC](#).

## Enhanced Features

Compared with the open source community, Huawei provides two enhanced features: JDBCServer HA solution and setting of the JDBCServer connection timeout interval.

- In the JDBCServer HA solution, multiple active JDBCServer nodes provide services at the same time. When one node is faulty, new client connections are allocated to other active nodes to ensure uninterrupted services for the cluster. The operations of using Beeline or JDBC client code for connection are the same.
- Set the timeout interval for the connection between the client and JDBCServer.
  - Beeline  
In the case of network congestion, this feature prevents the Beeline from being suspended due to the infinite waiting for the response of the server. The configuration method is as follows:  
When the Beeline is started, append **--socketTimeOut=n**, where **n** indicates the timeout interval (unit: second) for waiting for the service response. The default value is **0**, indicating that the beeline never times out. You are advised to set this parameter to the maximum value allowed based on the service scenario.
  - JDBC Client Codes  
In the case of network congestion, this feature prevents the client from being suspended due to the infinite waiting for the response of the server. The method is as follows:  
Before using the **DriverManager.getConnection** method to obtain the JDBC connection, add the **DriverManager.setLoginTimeout(n)** method to configure a timeout interval. **n** indicates the timeout interval for waiting for the return result from the server. The unit is second, the type is **Int**, and the default value is **0** (indicating never timing out). You are advised to set this parameter to the maximum value allowed based on the service scenario.

## 2.17.5.4 Structured Streaming Functions and Reliability

### Functions Supported by Structured Streaming

1. ETL operations on streaming data are supported.
2. Schema inference and partitioning of streaming DataFrames or Datasets are supported.
3. Operations on the streaming DataFrames or Datasets are supported, including SQL-like operations without types (such as select, where, and groupBy) and RDD operations with types (such as map, filter, and flatMap).
4. Aggregation calculation based on Event Time and processing of delay data are supported.
5. Deduplication of streaming data is supported.
6. Status computing is supported.
7. Stream processing tasks can be monitored.
8. Batch-stream join and stream join are supported.

The following table lists the supported join operations.

Left Table	Right Table	Supported Join Type	Description
Static	Static	All types	In stream processing, join operations with no streaming data involved are supported.
Stream	Static	Inner	Supported, but stateless.
		Left Outer	Supported, but stateless.
		Right Outer	Not supported.
		Full Outer	Not supported.
Stream	Stream	Inner	Supported. The watermark or time range can be used to clear status of the left and right tables.
		Left Outer	Conditionally supported. The watermark can be used to clear status of the left table, and watermark and time range must be used for the right table.
		Right Outer	Conditionally supported. The watermark can be used to clear status of the right table, and watermark and time range must be used for the left table.
		Full Outer	Not supported.

## Functions Not Supported by Structured Streaming

1. Multi-stream aggregation is not supported.
2. The following operations of obtaining multiple rows are not supported: limit, first, and take.
3. Distinct is not supported.
4. Sorting is supported only when output mode is complete.
5. The external connection between streams and static data sets is conditionally supported.
6. Immediate query and result return on some data sets are not supported.
  - count(): Instead of returning a single count from streaming data sets, it uses ds.groupBy().count() to return a streaming data set containing the running count.
  - foreach(): The ds.writeStream.foreach(...) is used to replace it.
  - show(): The output console sink is used to replace it.

## Structured Streaming Reliability

Based on the checkpoint and WAL mechanisms, Structured Streaming provides end-to-end exactly-once error tolerance semantics for the sources that can be replayed and the idempotent sinks that support repeated processing.

1. You can enable the checkpoint function by setting option ("checkpointLocation", "checkpoint path") in the program.

When data is restored from checkpoint, the application or configuration may change. Some changes may result in the failure in restoring data from the checkpoint. The restrictions are as follows:

- a. The number or type of sources cannot be changed.
- b. The source type and query statement determine whether the source parameter can change. For example:
  - The parameters related to rate control can be added, deleted, or modified. For example:  
`spark.readStream.format("kafka").option("subscribe", "topic")` is changed to `spark.readStream.format("kafka").option("subscribe", "topic").option("maxOffsetsPerTrigger", ...)`.
  - Unexpected problems may occur when the topic/file of the consumption is modified. For example:  
`spark.readStream.format("kafka").option("subscribe", "topic")` is changed to `spark.readStream.format("kafka").option("subscribe", "newTopic")`.
- c. The type of sink changes. Specific sinks can be combined. The specific scenarios need to be verified. For example:
  - File sink can be changed to kafka sink. Kafka processes only new data.
  - Kafka sink cannot be changed to file sink.
  - Kafka sink can be changed to foreach sink, and vice versa.

- d. The sink type and query statement determine whether the sink parameter can change. For example:
  - The output path of file sink cannot be changed.
  - The output topic of Kafka sink can be changed.
  - The custom operator code in foreach sink can be changed, but the change result depends on the user code.
- e. The projection, filter, and map-like operations can be changed in some scenarios. For example:
  - Filters can be added and deleted. For example: `sdf.selectExpr("a")` is changed to `sdf.where(...).selectExpr("a").filter(...)`.
  - When Output schema is the same, projections can be changed. For example: `sdf.selectExpr("stringColumn AS json").writeStream` is changed to `sdf.select(to_json(...).as("json")).writeStream`.
  - If Output schema is different, projections can be changed in some conditions. For example: when `sdf.selectExpr("a").writeStream` is changed to `sdf.selectExpr("b").writeStream`, no error occurs only when the sink supports schema conversion from a to b.
- f. In some scenarios, the status restoration will fail after status is changed.
  - Streaming aggregation: For example, in the `sdf.groupBy("a").agg(...)` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
  - Streaming deduplication: For example, in the `sdf.dropDuplicates("a")` operation, the type or quantity of the grouping key or the aggregation key cannot be changed.
  - Stream-stream join: For example, in the `sdf1.join(sdf2, ...)` operation, the schema of the association key cannot be changed, and the join type cannot be changed. Change of other join conditions may lead to uncertainty results.
  - Any status computing: For example, in the `sdf.groupByKey(...).mapGroupsWithState(...)` or `sdf.groupByKey(...).flatMapGroupsWithState(...)` operation, the schema or timeout type of the user-defined status cannot be changed. Users can customize the state-mapping function change, but the change result depends on the user code. If schema changes are required, users can encode or decode the status data into binary data to support schema migration.

## 2. Fault tolerance list of Source

Sources	Supported Options	Fault Tolerance Supported	Description
File source	<b>path</b> : file path, which is mandatory. <b>maxFilesPerTrigger</b> : maximum number of files in each trigger. The default value is infinity. <b>latestFirst</b> : whether to process limited number of new files. The default value is <b>false</b> . <b>fileNameOnly</b> : whether the file name is used as the new file for verification instead of using the complete path. (Default value: <b>false</b> )	Supported	Paths with wildcard are supported, but multiple paths separated by commas (,) are not supported. Files must be placed in a given directory in atomic mode, which can be implemented through file movement in most file systems.
Socket Source	<b>host</b> : IP address of the connected node, which is mandatory. <b>port</b> : connected port, which is mandatory.	Not supported.	-
Rate Source	<b>rowsPerSecond</b> : number of rows generated per second. The default value is 1. <b>rampUpTime</b> : rising time before the speed specified by <b>rowsPerSecond</b> is reached. <b>numPartitions</b> : concurrency degree for generating data rows.	Supported	-
Kafka Source	For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html</a> .	Supported	-

### 3. Fault tolerance list of Sink

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
File Sink	Append	<b>Path:</b> The specified file format must be specified. For details, see APIs in DataFrameWriter .	exactly-once	Data can be written to partition tables. Time-based partition is better.
Kafka Sink	Append, Update, Complete	For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html</a> .	at-least-once	For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html">https://spark.apache.org/docs/3.3.1/structured-streaming-kafka-integration.html</a> .
Foreach Sink	Append, Update, Complete	None	Depends on Foreach Writer.	For details, see <a href="https://spark.apache.org/docs/3.3.1/structured-streaming-programming-guide.html#using-foreach">https://spark.apache.org/docs/3.3.1/structured-streaming-programming-guide.html#using-foreach</a> .
ForeachBatch Sink	Append, Update, Complete	None	Depends on operator.	For details, see <a href="https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch">https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach-and-foreachbatch</a> .
Console Sink	Append, Update, Complete	<b>numRows:</b> number of rows printed in each round. The default value is <b>20</b> . <b>truncate:</b> whether to clear the output when the output is too long. The default value is <b>true</b> .	Not supported	-

Sinks	Supported Output Mode	Supported Options	Fault Tolerance	Description
Memory Sink	Append, Complete	None	Not supported. In complete mode, the entire table is rebuilt after the query is restarted.	-

## 2.17.5.5 FAQ

### 2.17.5.5.1 How to Add a User-Defined Library

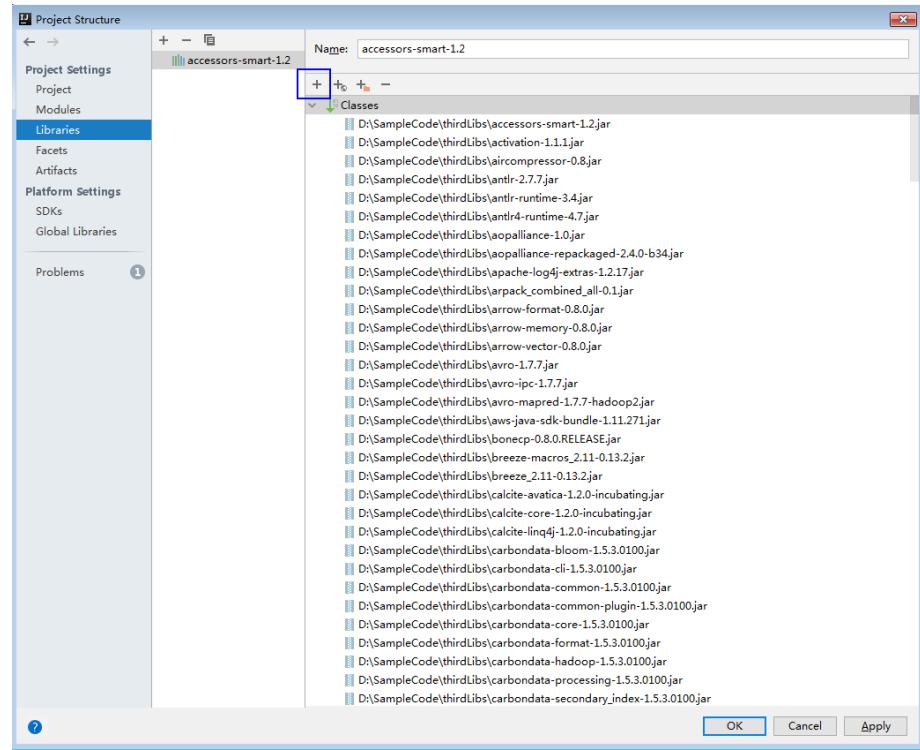
#### Question

In the development of the Spark application, users may add a user-defined dependent library which is different from the sample project. This section describes how to add a dependent library with user-defined codes into the project.

#### Answer

- Step 1** On the main page of the IDEA, choose **File > Project Structures...** to access the **Project Structure** page.
- Step 2** Select the **Libraries** tab, click **+** on the following page, and add the local dependent library.

Figure 2-303 Add the Dependent Library

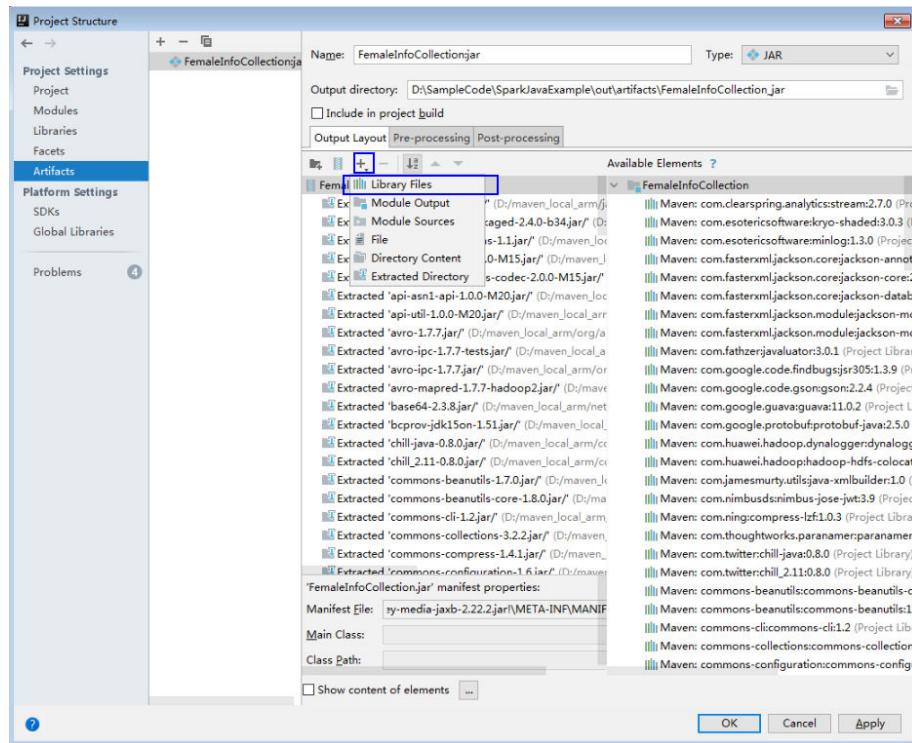


**Step 3** Click **Apply** to load the dependent library and click **OK** to complete the configuration.

**Step 4** Add the dependent library when building Artifacts. This is because the user-defined dependent library does not exist in the operating environment. By adding the library, the created jar contains the user-defined dependent library, which ensures that the Spark application runs properly.

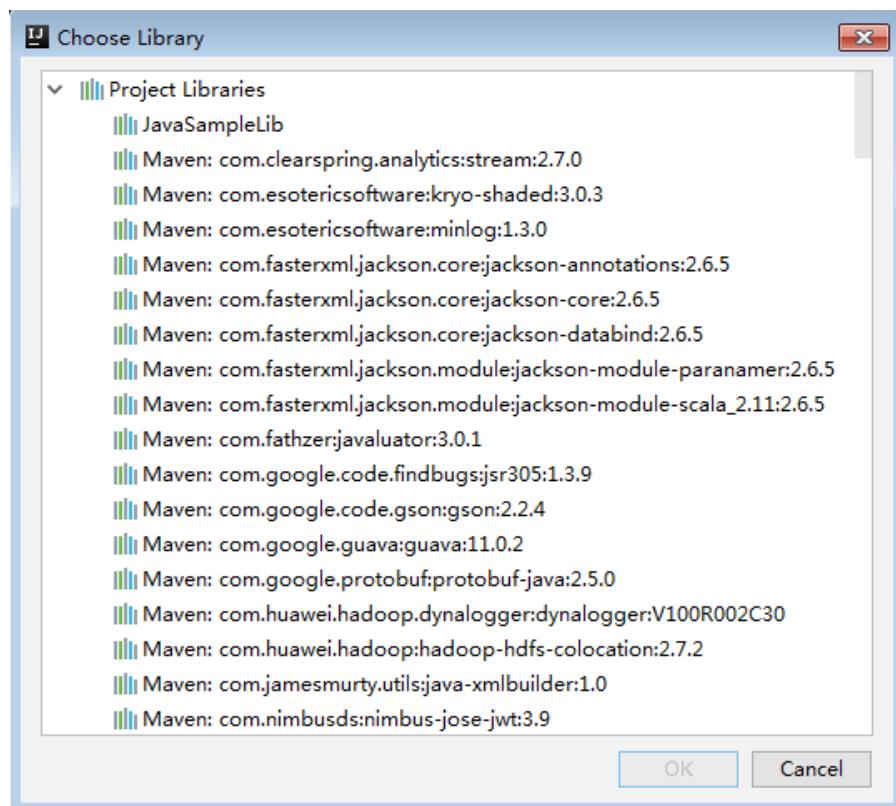
1. On the **Project Structure** page, select the **Artifacts** tab.
2. Click **+** and select **Library Files** in the right window to add the dependent library.

Figure 2-304 Add Library Files



3. Choose the dependent library to be added and click **OK**.

Figure 2-305 Choose Libraries



4. Click **Apply** to load the dependent library and click **OK** to complete the configuration.

---End

### 2.17.5.5.2 How to Automatically Load Jars Packages?

#### Question

Before importing a project by using the IDEA tool, if Maven has been configured in the IDEA tool, the IDEA tool will automatically load the Jars packages in the Maven configuration. When the automatically loaded Jars packages do not match with the application, the project fails to be built. How to Automatically Load Jars Packages?

#### Answer

After a project is imported, to manually delete the automatically loaded Jars packages, perform the following steps:

1. On the IDEA tool, choose **File > Project Structures....**
2. Select **Libraries** and select the Jars packages that are automatically imported. Right-click the mouse and select **Delete**.

### 2.17.5.5.3 Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?

#### Question

While the KafkaWordCount task (org.apache.spark.examples.streaming.KafkaWordCount) is being submitted by running the spark-submit script, the log file shows that the Kafka-related class does not exist. The KafkaWordCount sample is provided by the Spark open-source community.

#### Answer

When Spark is deployed, the following jar packages are saved in the **\$ {SPARK\_HOME}/jars/streamingClient010** directory on the client and the **\$ {BIGDATA\_HOME}/FusionInsight\_Spark\_8.3.1/install/FusionInsight-Spark-3.3.1/spark/jars/streamingClient010** directory on the server:

- kafka-clients-xxx.jar
- kafka\_2.12-xxx.jar
- spark-streaming-kafka-xxx.cbu.mrs.xxx.jar
- spark-token-provider-kafka-xxx.cbu.mrs.xxx.jar

Because **\$SPARK\_HOME/jars/streamingClient010/\*** is not added in to classpath by default, add "spark.executor.extraClassPath" and "spark.driver.extraClassPath" in the command.

When the application is submitted and run, add following parameters in the command. See [Compiling and Running the Application](#).

```
--jars $SPARK_CLIENT_HOME/jars/streamingClient010/kafka-client-2.4.0.jar,$SPARK_CLIENT_HOME/jars/streamingClient010/kafka_2.12-*jar,$SPARK_CLIENT_HOME/jars/streamingClient010/spark-streaming-kafka-xxx.cbu.mrs.xxx.jar
```

You can run the preceding command to submit the self-developed applications and sample projects.

To submit the sample projects such as KafkaWordCount provided by Spark open source community, you need to add other parameters in addition to **--jars**. Otherwise, the `ClassNotFoundException` error will occur. The configurations in yarn-client and yarn-cluster modes are as follows:

- **yarn-client mode**

In the configuration file **spark-defaults.conf** on the client, add the path of the client dependency package, for example `$SPARK_HOME/jars/streamingClient010/*`, (in addition to **--jars**) to the **spark.driver.extraClassPath** parameter.

- **yarn-cluster mode**

Perform any one of the following configurations in addition to **--jars**:

- In the configuration file **spark-defaults.conf** on the client, add the path of the server dependency package, for example,  `${BIGDATA_HOME}/FusionInsight_Spark_8.3.1/install/FusionInsight-Spark-3.3.1/spark/jars/streamingClient010/*`, to the **spark.yarn.cluster.driver.extraClassPath** parameter.
- Delete the `original-spark-examples_2.12-3.3.1-xxx.jar` packages from all the server nodes.
- In the **spark-defaults.conf** configuration file on the client, modify (or add and modify) the parameter **spark.driver.userClassPathFirst** to **true**.

#### 2.17.5.5.4 Why Does Kafka Fail to Receive the Data Written Back by Spark Streaming?

##### Question

While a running Spark Streaming task is writing data back to Kafka, Kafka cannot receive the written data and Kafka logs contain the following error information:

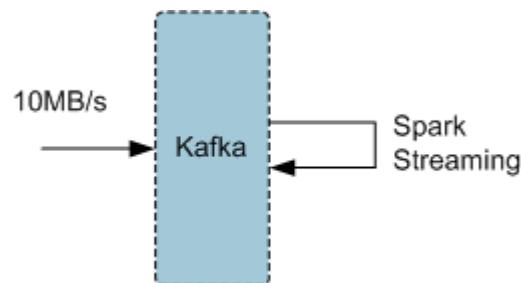
```
2016-03-02 17:46:19,017 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to / 10.91.8.208 due to invalid request: Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,155 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to / 10.91.8.208. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,270 | INFO | [kafka-network-thread-21005-0] | Closing socket connection to / 10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,513 | INFO | [kafka-network-thread-21005-1] | Closing socket connection to / 10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor (Logging.scala:68)
2016-03-02 17:46:19,763 | INFO | [kafka-network-thread-21005-2] | Closing socket connection to / 10.91.8.208 due to invalid request:
Request of length 122371301 is not valid, it is larger than the maximum size of 104857600 bytes. | kafka.network.Processor (Logging.scala:68)
53393 [main] INFO org.apache.hadoop.mapreduce.Job - Counters: 50
```

## Answer

As shown in the figure below, the logic defined in Spark Streaming applications is as follows: reading data from Kafka -> executing processing -> writing result data back to Kafka.

Imagine that data is written into Kafka at a data rate of 10 MB/s, the interval (defined in Spark Streaming) between write-back operations is 60s, and a total of 600-MB data needs to be written back into Kafka. If Kafka defines that a maximum of 500-MB data can be received at a time, then the size of written-back data exceeds the threshold, triggering the error information.

**Figure 2-306 Application scenario**



Solution:

- Method 1: On Spark Streaming, reduce the interval between write-back operations to avoid the size of written-back data exceeding the threshold defined by Kafka. The recommended interval is 5-10 seconds.
- Method 2: Increase the threshold defined in Kafka. It is advisable to increase the threshold by adjusting the **socket.request.max.bytes** parameter of Kafka service on FusionInsight Manager.

### 2.17.5.5 Why a Spark Core Application Is Suspended Instead of Being Exited When Driver Memory Is Insufficient to Store Collected Intensive Data?

## Question

A Spark Core application is attempting to collect intensive data and store it into the Driver. When the Driver runs out of memory, the Spark Core application is suspended. Why does the Spark Core application not exit?

The following is the log information displayed at the time of out-of-memory (OOM) error.

```
16/04/19 15:56:22 ERROR Utils: Uncaught exception in thread task-result-getter-2
java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newArray(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
```

```
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$anonfun$run$1.apply$mcV$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anonfun$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anonfun$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
Exception in thread "task-result-getter-2" java.lang.OutOfMemoryError: Java heap space
at java.lang.reflect.Array.newArray(Native Method)
at java.lang.reflect.Array.newInstance(Array.java:75)
at java.io.ObjectInputStream.readArray(ObjectInputStream.java:1671)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1707)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1345)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:71)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:91)
at org.apache.spark.scheduler.DirectTaskResult.value(TaskResult.scala:94)
at org.apache.spark.scheduler.TaskResultGetter$anonfun$run$1.apply$mcV$sp(TaskResultGetter.scala:66)
at org.apache.spark.scheduler.TaskResultGetter$$anonfun$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.scheduler.TaskResultGetter$$anonfun$3$anonfun$run$1.apply(TaskResultGetter.scala:57)
at org.apache.spark.util.Utils$.logUncaughtExceptions(Utils.scala:1716)
at org.apache.spark.scheduler.TaskResultGetter$anon$3.run(TaskResultGetter.scala:56)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
```

## Answer

If memory of the Driver is insufficient to store the intensive data that has been collected, the OOM error is reported and the Driver performs garbage collection repeatedly to reclaim the memory occupied by garbage. The Spark Core application remains suspended while garbage collection is under way.

If you expect the Spark Core application to exit forcibly in the event of OOM error, add the following information to the configuration option **spark.driver.extraJavaOptions** in the Spark client configuration file **\$SPARK\_HOME/conf/spark-defaults.conf** when you start the Spark Core application for the first time:  
`-XX:OnOutOfMemoryError='kill -9 %p'`

### 2.17.5.5.6 Why the Name of the Spark Application Submitted in Yarn-Cluster Mode Does not Take Effect?

#### Question

The name of the Spark application submitted in yarn-cluster mode does not take effect, whereas the Spark application name submitted in yarn-client mode takes effect. As shown in [Figure 2-307](#), the first application is submitted in yarn-client mode and the application name **Spark Pi** takes effect. However, the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode.

**Figure 2-307** Submitting the application

application_146359073605_0001	yarn	Spark Pi	SPARK	tenant_zm	Sat May 28 11:58:27 +0800 2016 2016	Sat May 28 11:59:51 +0800 2016 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	History	N/A
application_146359073605_0002	yarn	org.apache.spark.examples.SparkPi	SPARK	tenant_zm	Sat May 28 11:58:28 +0800 2016 2016	Sat May 28 11:59:51 +0800 2016 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A	History	N/A

#### Answer

The reason is that the setAppName execution sequence of a task submitted in yarn-client mode is different from that submitted in yarn-cluster mode. In yarn-client mode, the setAppName is read before the application is registered in yarn. However, in yarn-cluster mode, the setAppName is read after the application registers with yarn, so the name of the second application does not take effect.

#### Troubleshooting solution

When submitting tasks using the spark-submit script, set **--name** the same as the application name in sparkconf.setAppName (appname).

For example, if the application name is **Spark Pi**, run the following command to add the application name after **--name** when submitting the application in yarn-cluster mode:

```
./spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name SparkPi jars/original-spark-examples*.jar 10
```

### 2.17.5.5.7 How to Perform Remote Debugging Using IDEA?

#### Question

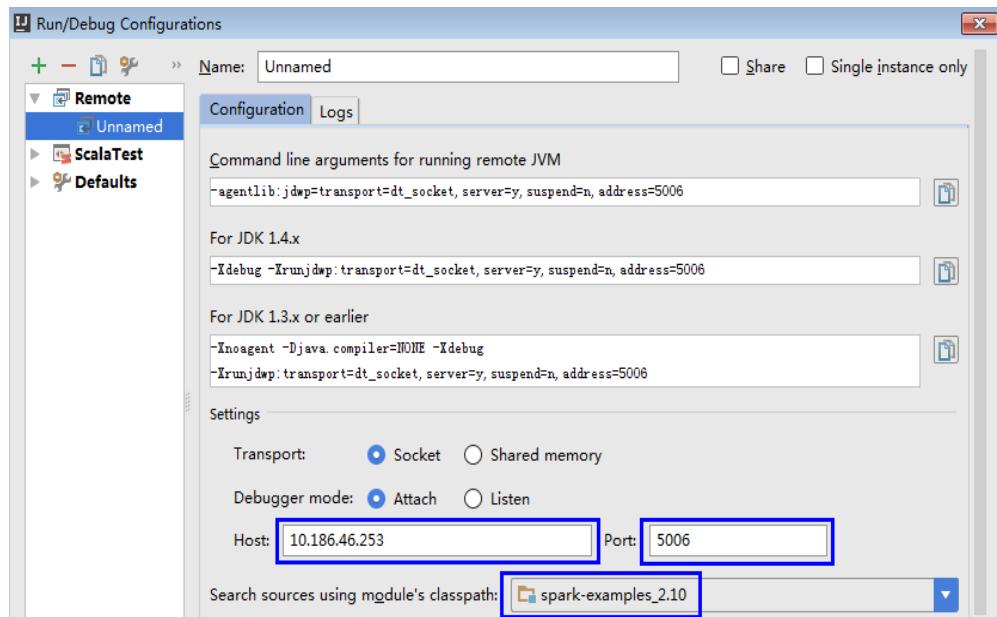
How to perform remote debugging using IDEA during Spark secondary development?

#### Answer

The SparkPi application is used as an example here to illustrate how to perform remote debugging using IDEA.

1. Open the project and choose **Run > Edit Configurations**.
2. In the displayed window, click at the upper left corner, and then choose **Remote** on the drop-down menu, as shown in [Figure 2-308](#).

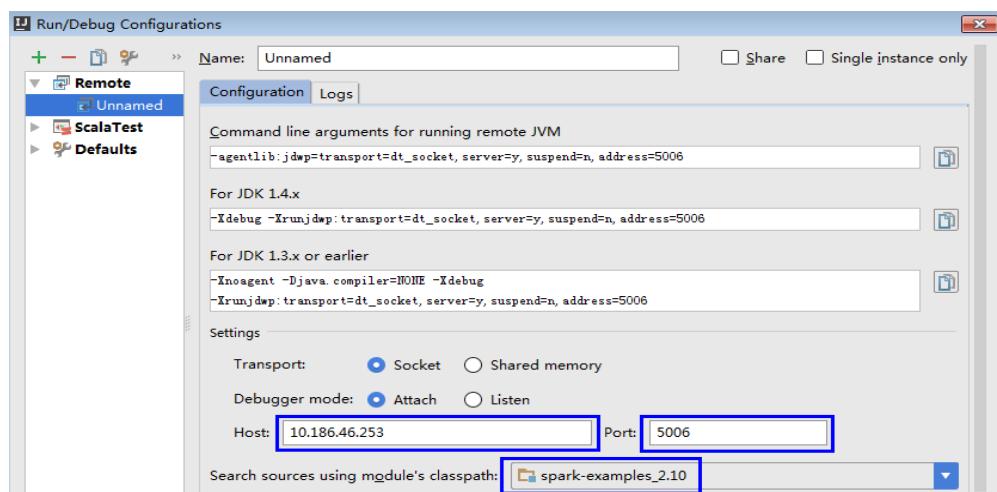
**Figure 2-308 Choosing Remote**



- Configure the **Host**, **Port**, and **Search source using module's classpath**, as shown in [Figure 2-309](#).

**Host** indicates the IP address of the Spark client and **Port** indicates the debugging port. Ensure that the port is available on the VM.

**Figure 2-309 Configuring parameters**



#### NOTE

If the value of **Port** is changed, the debugging command of **For JDK1.4.x** must be changed accordingly. For example, if the value of **Port** is changed to **5006**, the debugging command must be changed to **-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=5006**, which will be used during the startup of Spark.

- Run the following command to remotely start SparkPi on the Spark client:  
**./spark-submit --master yarn-client --driver-java-options "-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=5006" --class**

```
org.apache.spark.examples.SparkPi /opt/Fl-Client/Spark/spark/examples/jars/  
spark-examples_2.12-3.3.1-xxx.jar
```

Change the **--class** and JAR package in the preceding command to the **--class** and JAR package of the actual application. Change the **-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=5006** to the **For JDK1.4.x** debugging command obtained in [3](#).

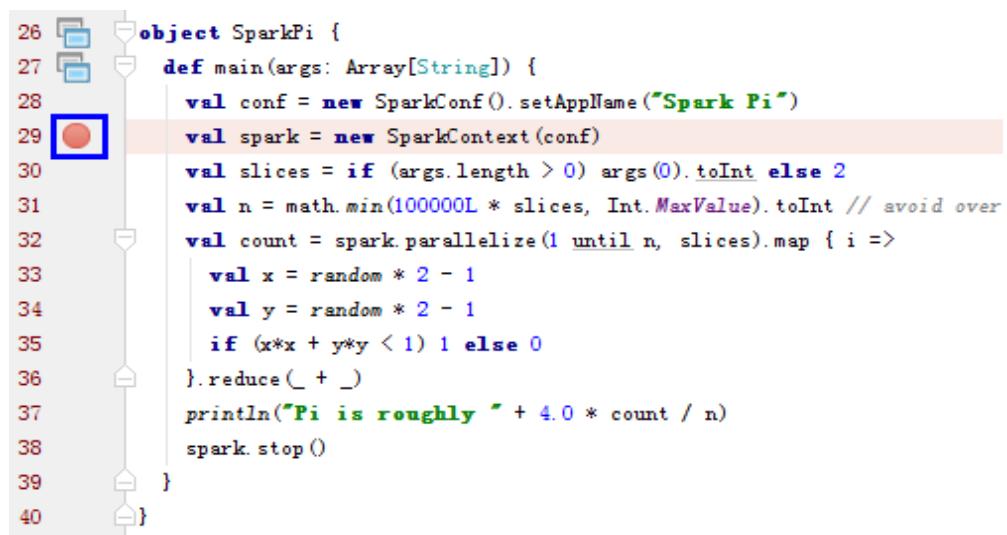
**Figure 2-310** Command for running Spark

```
[root@host2 bin]# ./spark-submit --master yarn-client --driver-java-options "-Xde  
bug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5006" --class org.a  
pache.spark.examples.SparkPi /opt/client/Spark2x/spark/examples/jars/spark-examp  
les_2.11-2.1.0.jar  
Listening for transport dt_socket at address: 5006
```

5. Set the debugging breakpoint.

Click the blank area on the left of the code editing window to select the breakpoint of code. [Figure 2-311](#) illustrates how to select the breakpoint of the code in row 29 of **SparkPi.scala**.

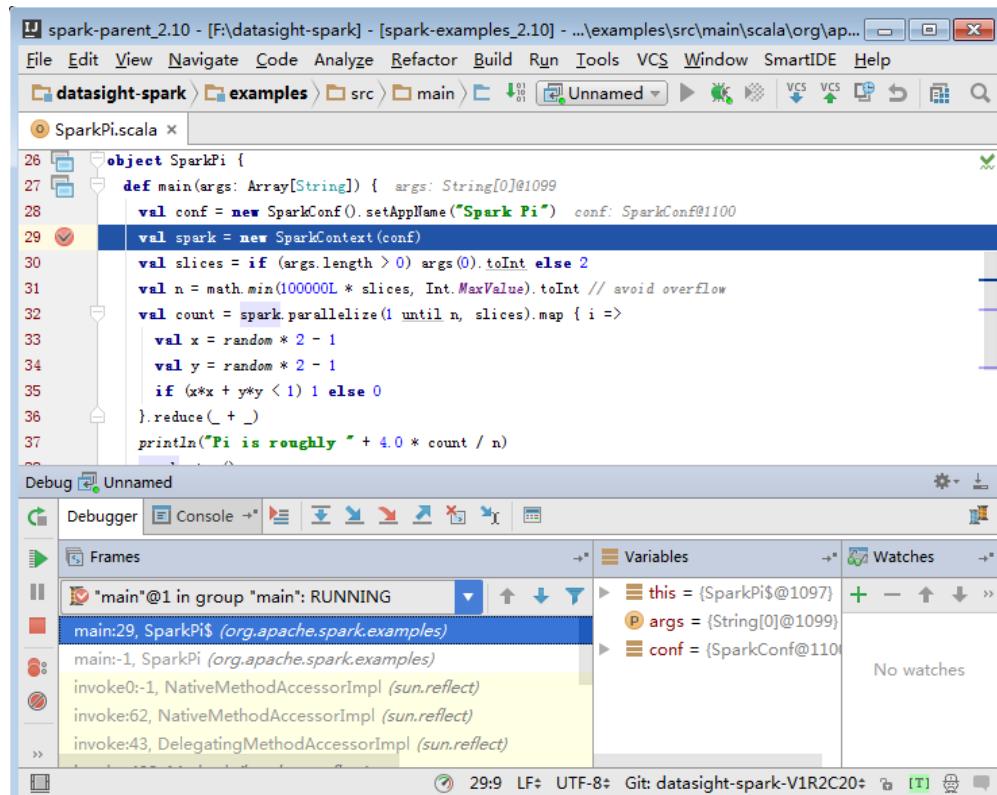
**Figure 2-311** Setting the breakpoint



6. Start the debugging.

On the menu bar of IDEA, choose **Run > Debug 'Unnamed'** to open a debugging window. Start the debugging of **SparkPi**, for example, performing step-by-step debugging, checking call stack information, and tracking variable values, as shown in [Figure 2-312](#).

Figure 2-312 Debugging



### 2.17.5.5.8 How to Submit the Spark Application Using Java Commands?

#### Question

How to submit the Spark application using Java commands in addition to spark-submit commands?

#### Answer

Use the org.apache.spark.launcher.SparkLauncher class and run Java command to submit the Spark application. The procedure is as follows:

**Step 1** Define the org.apache.spark.launcher.SparkLauncher class. The SparkLauncherJavaExample and SparkLauncherScalaExample are provided by default as sample codes. You can modify the input parameters of sample codes as required.

- If you use Java as the development language, you can compile the SparkLauncher class by referring to the following code:

```
public static void main(String[] args) throws Exception {
    System.out.println("com.huawei.bigdata.spark.examples.SparkLauncherExample <mode>
<jarPath> <app_main_class> <appArgs>");
    SparkLauncher launcher = new SparkLauncher();
    launcher.setMaster(args[0])
        .setAppResource(args[1]) // Specify user app jar path
        .setMainClass(args[2]);
    if (args.length > 3) {
        String[] list = new String[args.length - 3];
        for (int i = 3; i < args.length; i++) {
            list[i-3] = args[i];
        }
    }
}
```

```
// Set app args
launcher.addAppArgs(list);
}

// Launch the app
Process process = launcher.launch();
// Get Spark driver log
new Thread(new ISRRunnable(process.getErrorStream())).start();
int exitCode = process.waitFor();
System.out.println("Finished! Exit code is " + exitCode);
}
```

- If you use Scala as the development language, you can compile the `SparkLauncher` class by referring to the following code:

```
def main(args: Array[String]) {
    println(s"com.huawei.bigdata.spark.examples.SparkLauncherExample <mode> <jarPath>
<app_main_class> <appArgs>")
    val launcher = new SparkLauncher()
    launcher.setMaster(args(0))
    .setAppResource(args(1)) // Specify user app jar path
    .setMainClass(args(2))
    if (args.drop(3).length > 0) {
        // Set app args
        launcher.addAppArgs(args.drop(3): _*)
    }

    // Launch the app
    val process = launcher.launch()
    // Get Spark driver log
    new Thread(new ISRRunnable(process.getErrorStream())).start()
    val exitCode = process.waitFor()
    println(s"Finished! Exit code is $exitCode")
}
```

**Step 2** Develop the Spark application based on the service logic and configure constant values such as the main class of the user-compiled Spark application. For details about different scenarios, see [Developing the Project](#).

- If you use the security mode, you are advised to prepare the security authentication code, service application code, and related configurations according to the security requirements.

#### NOTE

In yarn-cluster mode, security authentication cannot be added to the Spark project, because security authentication must be completed before the application is started. Therefore, users need to add security authentication code or run commands to perform security authentication. There is security authentication code in the sample code. In yarn-cluster mode, modify the corresponding security code before running the operation.

- In normal mode, prepare the service application code and related configurations.

**Step 3** Call the `org.apache.spark.launcher.SparkLauncher.launch()` function to submit user applications.

1. Generate jar packages from the `SparkLauncher` application and user applications, and upload the jar packages to the Spark node of the application. For details about how to generate jar packages, see [Compiling and Running the Application](#).
  - The compilation dependency package of `SparkLauncher` is **spark-launcher\_xxx.cbu.mrs.xxx.jar**. Please obtain it from the **jars** directory of **FusionInsight\_Spark\_8.3.1.tar.gz** in the **Software** folder.

- The compilation dependency packages of user applications vary with the code. You need to load the dependency package based on the compiled code.
- 2. Upload the dependency jar package of the application to a directory, for example, **\$SPARK\_HOME/jars** (the node where the application will run).  
Upload the dependency packages of the SparkLauncher class and the application to the **jars** directory on the client. The dependency package of the sample code has existed in the **jars** directory on the client.

 **NOTE**

If you want to use the Spark Launcher class, the node where the application runs must have the Spark client installed. The running of the Spark Launcher class is dependent on the configured environment variables, running dependency package, and configuration files.

- 3. In the node where the Spark application is running, run the following command to submit the application. Then you can check the running situation through Spark web UI and check the result by obtaining specified files. See [Checking the Commissioning Result](#) for details.

```
java -cp $SPARK_HOME/conf:$SPARK_HOME/jars/  
*:SparkLauncherExample.jar  
com.huawei.bigdata.spark.examples.SparkLauncherExample yarn-  
client /opt/female/FemaleInfoCollection.jar  
com.huawei.bigdata.spark.examples.FemaleInfoCollection <inputPath>
```

----End

### 2.17.5.5.9 A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used

#### Question

A Message Stating "Problem performing GSS wrap" Is Displayed When IBM JDK Is Used.

#### Answer

##### Possible Causes

The authentication fails because the duration for creating a JDBC connection on IBM JDK exceeds the timeout duration for user authentication (one day by default).

 **NOTE**

The authentication mechanism of IBM JDK differs from that of Oracle JDK. IBM JDK checks time but does not detect external time update. Therefore, time is not updated even though **relogin** is called.

##### Solution

When one JDBC connection fails, disable this connection, and create a new connection to continue performing previous operations.

## 2.17.5.5.10 Application Fails When ApplicationManager Is Terminated During Data Processing in the Cluster Mode of Structured Streaming

### Question

If ApplicationManager is terminated during data processing in the cluster mode of Structured Streaming, the following information is displayed when the application is executed, indicating an error:

```
2017-05-09 20:46:02,393 | INFO | main |
  client token: Token { kind: YARN_CLIENT_TOKEN, service: } |
  diagnostics: User class threw exception: org.apache.spark.sql.AnalysisException: This query does not
support recovering from checkpoint location. Delete hdfs://hacluster/structuredtest/checkpoint/offsets to
start over;
  ApplicationMaster host: 10.96.101.170
  ApplicationMaster RPC port: 0
  queue: default
  start time: 1494333891969
  final status: FAILED
  tracking URL: https://9-96-101-191:26001/proxy/application_1493689105146_0052/
  user: spark2x | org.apache.spark.internal.Logging$class.logInfo(Logging.scala:54)
Exception in thread "main" org.apache.spark.SparkException: Application application_1493689105146_0052
finished with failed status
```

### Answer

**Possible causes:** The error occurs because **recoverFromCheckpointLocation** is determined as **false** but the checkpoint directory is configured.

The value of the **recoverFromCheckpointLocation** parameter is the result of the **outputMode == OutputMode.Complete()** statement in the code. (The default **outputMode** is **append**.)

**Solution:** When compiling an application, you can change the data output mode based on the actual conditions. For operations to change the output mode by calling **outputMode**, see the **outputMode** description of **Spark > Scala** in *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)*.

When the output mode is changed to **complete**, the value of **recoverFromCheckpointLocation** is determined as **true**. No error would be indicated if the checkpoint directory is configured at the time.

## 2.17.5.5.11 Restrictions on Restoring the Spark Application from the checkpoint

### Question

The Spark application can be restored from the checkpoint and continues to execute the task from the breakpoint of the last task, ensuring that data is not lost. However, in some cases, the Spark application fails to be restored from the checkpoint.

### Answer

The checkpoint contains the object serialization information, task execution status information, and configuration information of the Spark application. Therefore, the Spark application cannot be restored from the checkpoint if the following problems exist:

1. The service code is changed and the serialVersionUID is not specified in the changed class.
2. The internal Spark class is changed and the serialVersionUID is not specified in the changed class.

Besides, some configuration items are stored in the checkpoint. Therefore, if some configuration items of the service are modified, the configuration items may remain unchanged when the service is restored from the checkpoint. Currently, only the following configurations are reloaded when the service is restored from the checkpoint.

```
"spark.yarn.app.id",
"spark.yarn.app.attemptId",
"spark.driver.host",
"spark.driver.bindAddress",
"spark.driver.port",
"spark.master",
"spark.yarn.jars",
"spark.yarn.keytab",
"spark.yarn.principal",
"spark.yarn.credentials.file",
"spark.yarn.credentials.renewalTime",
"spark.yarn.credentials.updateTime",
"spark.ui.filters",
"spark.mesos.driver.frameworkId",
"spark.yarn.jars"
```

## Solution

Manually delete the checkpoint directory and restart the service program.



Deleting a file or folder is a high-risk operation. Ensure that the file or folder is no longer required before performing this operation.

### 2.17.5.5.12 Support for Third-party JAR Packages on x86 and TaiShan Platforms

#### Question

How to enable Spark to support the third-party JAR packages (for example, custom UDF packages) if these packages have two versions (x86 and TaiShan)?

#### Answer

If the third-party JAR packages (for example, custom UDF packages) have two versions (x86 and TaiShan), the following hybrid solution is used:

- Step 1** Go to the installation directory of the Spark SparkResource on the server. The cluster may be installed on multiple nodes. You can go to any SparkResource node to go to the installation directory of the SparkResource.
- Step 2** Prepare the .jar package, for example, xx.jar packages for the x86 and TaiShan platforms. Copy the xx.jar packages of x86 and TaiShan to the x86 and TaiShan folders respectively.
- Step 3** Run the following commands in the current directory to compress the JAR packages:

```
zip -qDj spark-archive-x86.zip x86/*
zip -qDj spark-archive-arm.zip arm/*
```

```
total 20210
lwx----- 2 omm wheel      63 Dec 14 17:57 arm
lwx----- 2 omm wheel      4096 Dec 14 17:57 bin
lwxr-x-- 2 omm ficommon   4096 Dec 14 17:57 carbonlib
rw----- 1 omm wheel      1403 Dec 15 12:51 child.crt
rw----- 1 omm wheel      1097 Dec 15 12:51 child.csr
rw----- 1 omm wheel      6296 Dec 15 12:51 child.keystore
lwx----- 2 omm wheel      326 Dec 14 17:57 conf
lwx----- 3 omm wheel      18 Dec 14 17:54 examples
lwxr-x-- 4 omm ficommon  12288 Dec 14 17:57 jars
rw----- 1 omm wheel      18045 Dec 14 17:54 LICENSE
rw----- 1 omm wheel      26366 Dec 14 17:54 NOTICE
lwx----- 5 omm wheel      129 Dec 14 17:57 python
lwx----- 3 omm wheel      17 Dec 14 17:54 R
rw----- 1 omm wheel      501 Dec 14 17:54 README
rw----- 1 omm wheel      20 Dec 14 17:54 RELEASE
lwx----- 2 omm wheel      4096 Dec 15 17:14 sbin
lwr-r--r- 1 root root    14904892 Dec 15 17:14 spark-archive-2x-arm.zip
lwr-r--r- 1 root root    13881100 Dec 15 17:15 spark-archive-2x-x86.zip
rw----- 1 omm wheel      244 Dec 14 17:57 version.properties
lwx----- 2 omm wheel      63 Dec 14 17:57 x86
lwx----- 2 omm wheel      42 Dec 14 17:54 yarn
```

- Step 4** Run the following command to check the .jar package on which the Spark of HDFS depends:

```
hdfs dfs -ls /user/spark/jars/8.3.1
```



Change the version number 8.3.1 as required.

Run the following commands to move the .jar package files from HDFS, for example, /tmp.

```
hdfs dfs -mv /user/spark/jars/8.3.1/spark-archive-arm.zip /tmp
```

```
hdfs dfs -mv /user/spark/jars/8.3.1/spark-archive-x86.zip /tmp
```

- Step 5** Run the following commands to upload the **spark-archive-arm.zip** and **spark-archive-x86.zip** packages in [Step 3](#) to the **/user/spark/jars/8.3.1** directory of HDFS:

```
hdfs dfs -put spark-archive-arm.zip /user/spark/jars/8.3.1/
```

```
hdfs dfs -put spark-archive-x86.zip /user/spark/jars/8.3.1/
```

After the upload is complete, delete local files **spark-archive-arm.zip** and **spark-archive-x86.zip**.

- Step 6** Perform [Step 1](#) to [Step 2](#) for other SparkResource nodes.

- Step 7** Log in to the web UI and restart the JDBCServer instance of Spark.

- Step 8** After the restart, update the client configuration. Copy xx.jar for the corresponding platform to the Spark installation directory **\$install\_home/Spark/spark/jars** on the client according to client server type (x86 or TaiShan). **\$install\_home** indicates the installation path of the client. Replace it with the actual one. If the local installation directory is **/opt/hadoopclient**, copy the corresponding xx.jar to the **/opt/hadoopclient/Spark/spark/jars** folder.

----End

## 2.17.5.5.13 What Should I Do If a Large Number of Directories Whose Names Start with **blockmgr-** or **spark-** Exist in the /tmp Directory on the Client Installation Node?

### Question

After the system runs for a long time, there are many directories whose names start with **blockmgr-** or **spark-** in the **/tmp** directory on the node where the client is installed.

**Figure 2-313** Residual directory example

```
blockmgr-934dc20a-d5f0-4adf-a28f-c376ef0fe01d  
blockmgr-f514f38b-209c-4a65-985a-2a6c61d0ee00  
spark-33f95b4b-be82-4290-bde3-07b76c797085  
spark-988e28a7-0416-4115-8d6e-3a62a75f1f46
```

### Answer

During the running of Spark tasks, the driver creates a local temporary directory whose name starts with **spark-** for storing service JAR packages and configuration files. In addition, the driver creates a local temporary directory with the name starting with **blockmgr-** for storing block data. The two directories are automatically deleted when the Spark application running is finished.

The path for storing the two directories is preferentially specified by the environment variable **SPARK\_LOCAL\_DIRS**. If the environment variable is not configured, use the value of **spark.local.dir** as the path for storing the directories. If the environment variable and the preceding parameter both are not configured, use the value of **java.io.tmpdir**. By default, **spark.local.dir** is set to **/tmp** on the client. Therefore, the **/tmp** directory is used by default.

In some special cases, for example, the driver process does not exit normally, for example, the **kill -9** command ends the process, or the Java virtual machine crashes. As a result, the directory cannot be deleted and remains in the system.

Currently, only the driver processes in yarn-client mode and local mode may confront the preceding problem. In yarn-cluster mode, the temporary directory of the process in the container is configured as the temporary directory of the container. When the container exits, the container automatically clears the directory. Therefore, this problem does not occur in yarn-cluster mode.

### Solution

In Linux, you can configure automatic directory clearing for the **/tmp** temporary directory. Alternatively, you can change the value of **spark.local.dir** in the **spark-defaults.conf** configuration file on the client, specify the temporary directory to a specified directory, and configure a clear mechanism for the directory.

## 2.17.5.5.14 Error Code 139 Is Reported When Python Pipeline Runs in the Arm Environment

### Question

Error code 139 is displayed when the pipeline of the Python plug-in is used on the TaiShan server. The error information is as follows:

```
subprocess exited with status 139
```

### Answer

The python program uses both **libcrypto.so** and **libssl.so**. If the native library directory of Hadoop is added to **LD\_LIBRARY\_PATH**, the **libcrypto.so** in the **hadoop native** library is used and the **libssl.so** provided by the system is used (because the **hadoop native** directory does not contain this package). The versions of the two libraries do not match. As a result, a segment error occurs when the Python file is running.

### Solution

Modify the **spark-default.conf** file in the **conf** directory of the Spark client. Clear the values of **spark.driver.extraLibraryPath**, **spark.yarn.cluster.driver.extraLibraryPath**, and **spark.executor.extraLibraryPath**.

## 2.17.5.5.15 What Should I Do If the Structured Streaming Task Submission Way Is Changed?

### Question

When submitting a structured streaming task, users need to run the **--jars** command to specify the Kafka JAR package path, for example, **--jars /kafkadir/kafka-clients-x.x.x.jar,/kafkadir/kafka\_2.11-x.x.x.jar**. However, in the current version, users need to configure additional items. Otherwise, an error is reported, indicating that the class is not found.

### Answer

The Spark kernel of the current version depends on the Kafka JAR package, which is used by the structured streaming. Therefore, when submitting a structured streaming task, you need to add the Kafka JAR package path to the library directory of the driver of this task to ensure that the driver can properly load the Kafka package.

### Solution

1. The following operations need to be performed additionally when a structured streaming task in Yarn-client mode is submitted:  
Copy the path of **spark.driver.extraClassPath** in the **spark-default.conf** file in the Spark client directory, and add the Kafka JAR package path to its end. When submitting a structured stream task, add the **--conf** statement to combine these two configuration items. For example, if the Kafka JAR package path is **/kafkadir**, you need to add **--conf**

- `spark.driver.extraClassPath=/opt/hadoopclient/Spark/spark/conf/:/opt/hadoopclient/Spark/spark/jars/*:/opt/hadoopclient/Spark/spark/x86/*:/kafkadir/*` when submitting the task.
2. The following operations need to be performed additionally when a structured streaming task in **Yarn-cluster** mode is submitted:  
Copy the path of `spark.yarn.cluster.driver.extraClassPath` in the **spark-default.conf** file in the Spark client directory, and add relative paths of Kafka JAR packages to its end. When submitting a structured stream task, add the `--conf` statement to combine these two configuration items. For example, if the Kafka JAR package paths are `kafka-clients-x.x.x.jar` and `kafka_2.11-x.x.x.jar`, you need to add `--conf spark.yarn.cluster.driver.extraClassPath=/home/huawei/Bigdata/common/runtime/security:/kafka-clients-x.x.x.jar:/kafka_2.11-x.x.x.jar` when submitting the task.
  3. In the current version, the structured streaming of Spark does not support versions earlier than Kafka2.x. In the upgrade scenario, use the client of earlier versions.

## 2.17.5.5.16 Migrating Spark Streaming's Interconnection from Kafka 0.8 to Kafka 0.10

### Overview

Kafka 0.8 is not recommended for interconnection with Spark 2.3, and is not supported in Spark 3.0.0. If **spark-streaming-kafka-0-8** is used in Spark Streaming after Spark 2.3 is upgraded to Spark 3.1.1, you need to change the dependency to **spark-streaming-kafka-0-10** after the upgrade.

### Difference Analysis

- Maven dependency versions

After Spark Streaming is interconnected to **spark-streaming-kafka-0-10**, the Maven dependency versions in the code change. The related dependencies are as follows:

#### **org.apache.spark:**

- **spark-core\_2.11:** Change `artifactId` to **spark-core\_2.12** and upgrade its version number to the one of the cluster as Scala is upgraded to 2.12.
- **spark-streaming\_2.11:** Change `artifactId` to **spark-streaming\_2.12** and upgrade its version number to the one of the cluster as Scala is upgraded to 2.12.
- **spark-streaming-kafka-0-8\_2.11:** Removed. Use **spark-streaming-kafka-0-10\_2.12** instead and upgrade the version to the one of the cluster.
- **spark-streaming-kafkaWriter-0-8\_2.11:** Removed. Use **spark-streaming-kafkaWriter-0-10\_2.12** instead and upgrade the version to the one of the cluster.

#### **org.apache.kafka:**

- **kafka\_2.11:** Upgrade from version 0.8.2.1 to the corresponding version of the cluster.
- **kafka-clients:** Add this dependency, and its version number should be the same as that of the cluster.

- Related interfaces

**spark-streaming-kafka-0-8** can obtain data from Kafka APIs **Receiver-based Approach** and **Direct Approach (No Receivers)**. In **spark-streaming-kafka-0-10**, only the method using **Direct Approach** is retained.

Differences exist in related APIs. For details, see the sample code **SparkStreamingKafka010ScalaExample** and **SparkStreamingKafka010JavaExample**

## Solution

- Maven dependency versions

Change the dependency versions referenced in the code. Add dependencies if no one exists.

### **org.apache.spark**

- **spark-core\_2.11**: Change **artifactId** to **spark-core\_2.12** as Scala is upgraded to 2.12.
- **spark-streaming\_2.11**: Change to **spark-streaming\_2.12** and change the version number to the one of Spark in the cluster.
- **spark-streaming-kafka-0-8\_2.11**: Replace it with **spark-streaming-kafka-0-10\_2.12** and change the version number to the one of Spark in the cluster.
- **spark-streaming-kafkaWriter-0-8\_2.11**: Replace it with **spark-streaming-kafkaWriter-0-10\_2.12** and change the version number to the one of Spark in the cluster.

### **org.apache.kafka**

- **kafka\_2.11**: Change the version number from 0.8.2.1 to the one of Kafka in the cluster.
- **kafka-clients**: Add this dependency, and its version number should be the same as that of the cluster.

- For details about related APIs, see the sample code **SparkStreamingKafka010ScalaExample** and **SparkStreamingKafka010JavaExample**.

- Job submission mode

- Brokers need to use port **21005**, for example, **IP address of the Kafka service Broker instance:21005**.
- The path of Spark Streaming's Kafka dependency package on the client is different from that of other dependency packages. For example, the path of other dependency packages is **\$SPARK\_HOME/jars**, and the path of the Kafka dependency package is **\$SPARK\_HOME/jars/streamingClient010**. Therefore, when running an application, you need to add a configuration item to the **spark-submit** command to specify the path of the Spark Streaming's Kafka dependency package, for example, **--jars \${files=(\$SPARK\_HOME/jars/streamingClient010/\*.jar);IFS=,:;echo "\${files[\*]}"}\*\***.
- The running sample depends on the **kafka-clients** component. Therefore, if there is no **kafka-clients-xxx.jar** package in the **\$CLIENT/Spark/spark/jars** directory, copy the **kafka-clients-xxx.jar** file from the **\$CLIENT/Spark/spark/jars/streamingClient010** directory to the

**\$CLIENT/Spark/spark/jars** directory. Download the JAR file of the **spark-streaming-kafkaWriter-0-10** cluster version from the Maven repository and save the file to the **\$CLIENT/Spark/spark/jars/streamingClient010** directory.

- When you run the sample, an error is reported indicating that the class does not exist. For details, see [Why the "Class Does not Exist" Error Is Reported While the SparkStreamingKafka Project Is Running?](#).

## References

Spark Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher): <https://spark.apache.org/docs/3.1.1/streaming-kafka-0-10-integration.html>

Spark Streaming + Kafka Integration Guide (Kafka broker version 0.8.2.1 or higher): <https://spark.apache.org/docs/2.4.5/streaming-kafka-0-8-integration.html>

Modify the Java sample code. The following code snippet is for reference only.

```
FemaleInfoCollectionPrint.java
// Parameter description:
// <checkPointDir> indicates the checkPoint directory.
// <batchTime> is the interval for Streaming processing in batches.
// <topics> are topics subscribed in Kafka. Multiple topics are separated by commas (,).
// <brokers> is the Kafka address for obtaining metadata.

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.spark.streaming.api.java.JavaInputDStream;
import org.apache.spark.streaming.kafka010.*;
import scala.Tuple2;
import scala.Tuple3;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.streaming.Duration;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import java.util.*;

public class FemaleInfoCollectionPrint {
    public static void main(String[] args) throws Exception {

        String checkPointDir = args[0];
        String batchTime = args[1];
        String topics = args[2];
        String brokers = args[3];

        Duration batchDuration = Durations.seconds(Integer.parseInt(batchTime));

        SparkConf conf = new SparkConf().setAppName("DataSightStreamingExample");
        JavaStreamingContext jssc = new JavaStreamingContext(conf, batchDuration);

        // Set the CheckPoint directory of Streaming.
        jssc.checkpoint(checkPointDir);

        // Assemble a Kafka topic list.
        String[] topicArr = topics.split(",");
        Set<String> topicsSet = new HashSet<String>(Arrays.asList(topicArr));
        Map<String, Object> kafkaParams = new HashMap();
        kafkaParams.put("bootstrap.servers", brokers);
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        kafkaParams.put("group.id", "DemoConsumer");
```

```
LocationStrategy locationStrategy = LocationStrategies.PreferConsistent();
ConsumerStrategy consumerStrategy = ConsumerStrategies.Subscribe(topicsSet, kafkaParams);

// Create a Kafka stream using brokers and topics.
// Receive data from Kafka and generate the corresponding DStream.
JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.createDirectStream(jssc,
locationStrategy, consumerStrategy);

// Obtain the field attribute of each row.
JavaDStream<String> lines = messages.map(new Function<ConsumerRecord<String, String>, String>() {
@Override
public String call(ConsumerRecord<String, String> tuple2) throws Exception {
return tuple2.value();
}
});

JavaDStream<Tuple3<String, String, Integer>> records = lines
.map(new Function<String, Tuple3<String, String, Integer>>() {
public Tuple3<String, String, Integer> call(String line) throws Exception {
String[] elems = line.split(",");
return new Tuple3<String, String, Integer>(elems[0], elems[1], Integer.parseInt(elems[2]));
}
});

// Filter the data information of the time that female netizens spend online.
JavaDStream<Tuple2<String, Integer>> femaleRecords = records
.filter(new Function<Tuple3<String, String, Integer>, Boolean>() {
public Boolean call(Tuple3<String, String, Integer> line) throws Exception {
if (line._2().equals("female")) {
return true;
} else {
return false;
}
}
}).map(new Function<Tuple3<String, String, Integer>, Tuple2<String, Integer>>() {
public Tuple2<String, Integer> call(Tuple3<String, String, Integer> stringStringIntegerTuple3)
throws Exception {
return new Tuple2<String, Integer>(stringStringIntegerTuple3._1(),
stringStringIntegerTuple3._3());
}
});

// Filter the data of netizens whose consecutive online duration exceeds the threshold.
JavaDStream<Tuple2<String, Integer>> upTimeUser = femaleRecords
.filter(new Function<Tuple2<String, Integer>, Boolean>() {
public Boolean call(Tuple2<String, Integer> stringIntegerTuple2) throws Exception {
if (stringIntegerTuple2._2() > 30) {
return true;
} else {
return false;
}
}
});

// Print the result.
upTimeUser.print();

// Enable Streaming.
jssc.start();
jssc.awaitTermination();
}

}

JavaDstreamKafkaWriter.java
/**
 * Parameter description:
 * <checkPointDir> is the checkPoint directory.
 * <topics> are topics subscribed in Kafka. Multiple topics are separated by commas (,).
 * <brokers> is the Kafka address for obtaining metadata.
```

```
*/  
  
import com.huawei.spark.streaming.kafka010.JavaDStreamKafkaWriterFactory;  
import org.apache.kafka.clients.producer.ProducerRecord;  
import org.apache.spark.SparkConf;  
import org.apache.spark.api.java.JavaRDD;  
import org.apache.spark.api.java.function.Function;  
import org.apache.spark.streaming.Durations;  
import org.apache.spark.streaming.api.java.JavaDStream;  
import org.apache.spark.streaming.api.java.JavaStreamingContext;  
import scala.collection.JavaConverters;  
import java.util.*;  
  
public class JavaDstreamKafkaWriter {  
    public static void main(String[] args) throws InterruptedException {  
        if (args.length != 3) {  
            System.err.println("Usage: JavaDstreamKafkaWriter <groupId> <brokers> <topic>");  
            System.exit(1);  
        }  
  
        final String groupId = args[0];  
        final String brokers = args[1];  
        final String topic = args[2];  
  
        SparkConf sparkConf = new SparkConf().setAppName("KafkaWriter");  
  
        // Assemble a Kafka topic list.  
        Map<String, Object> kafkaParams = new HashMap<String, Object>();  
        kafkaParams.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");  
        kafkaParams.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");  
        kafkaParams.put("value.serializer", "org.apache.kafka.common.serialization.ByteArraySerializer");  
        kafkaParams.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");  
        kafkaParams.put("bootstrap.servers", brokers);  
        kafkaParams.put("group.id", groupId);  
        kafkaParams.put("auto.offset.reset", "smallest");  
  
        // Create context of the Java Spark Streaming.  
        JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.milliseconds(500));  
  
        // Add data to be written to Kafka.  
        List<String> sentData = new ArrayList();  
        sentData.add("kafka_writer_test_msg_01");  
        sentData.add("kafka_writer_test_msg_02");  
        sentData.add("kafka_writer_test_msg_03");  
  
        // Create a Java RDD queue.  
        Queue<JavaRDD<String>> sent = new LinkedList();  
        sent.add(ssc.sparkContext().parallelize(sentData));  
  
        // Create a Java DStream for writing data.  
        JavaDStream wStream = ssc.queueStream(sent);  
  
        // Write data to Kafka.  
        JavaDStreamKafkaWriterFactory.fromJavaDStream(wStream)  
            .writeToKafka(  
                JavaConverters.mapAsScalaMapConverter(kafkaParams).asScala(),  
                new Function<String, ProducerRecord<String, byte[]>>() {  
                    @Override  
                    public ProducerRecord<String, byte[]> call(String s) throws Exception {  
                        return new ProducerRecord(topic, s.toString().getBytes());  
                    }  
                });  
  
        ssc.start();  
        ssc.awaitTermination();  
    }  
}
```

Modify the Scala sample code. The following code snippet is for reference only.

```
FemaleInfoCollectionPrint.scala
// Parameter description:
// <checkPointDir> is the checkPoint directory.
// <batchTime> is the interval for Streaming processing in batches.
// <topics> are topics subscribed in Kafka. Multiple topics are separated by commas (,).
// <brokers> is the Kafka address for obtaining metadata.

import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka010.{ConsumerStrategies, KafkaUtils, LocationStrategies}
import org.apache.spark.streaming.{Seconds, StreamingContext}

object FemaleInfoCollectionPrint {
def main(args: Array[String]) {

    val Array(checkPointDir, batchTime, topics, brokers) = args

    // Set up a Streaming startup environment.
    val sparkConf = new SparkConf().setAppName("KafkaWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(batchTime.toLong))

    // Set the CheckPoint directory of Streaming.
    //This parameter is mandatory because of existence of the window concept.
    ssc.checkpoint(checkPointDir)

    // Obtain the list of topics used by Kafka.
    val topicArr = topics.split(",")
    val topicSet = topicArr.toSet
    val kafkaParams = Map[String, String](
        "bootstrap.servers" -> brokers,
        "value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
        "key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
        "group.id" -> "DemoConsumer"
    );

    val locationStrategy = LocationStrategies.PreferConsistent
    val consumerStrategy = ConsumerStrategies.Subscribe[String, String](topicSet, kafkaParams)

    // Create a Kafka stream using brokers and topics.
    // Receive data from Kafka and generate the corresponding DStream.
    val stream = KafkaUtils.createDirectStream[String, String](ssc, locationStrategy, consumerStrategy)
    val lines = stream.transform( rdd =>
        rdd.map(r => (r.value))
    )

    // Obtain the field attribute of each row.
    val records = lines.map(getRecord)
    // Filter the data information of the time that female netizens spend online.
    val femaleRecords = records.filter(_.value == "female")
    .map(x => (x._1, x._3))
    // Filter netizens whose consecutive online duration exceeds the threshold, and obtain the result.
    femaleRecords.filter(_.value > 30).print()

    // Start Streaming.
    ssc.start()
    ssc.awaitTermination()
}

// Obtain field functions.
def getRecord(line: String): (String, String, Int) = {
    val elems = line.split(",")
    val name = elems(0)
    val sexy = elems(1)
    val time = elems(2).toInt
    (name, sexy, time)
}

DstreamKafkaWriter.scala
/**
 * Parameter description:
 */
```

```
* <checkPointDir> is the checkPoint directory.
* <topics> is topics subscribed in Kafka. Multiple topics are separated by commas (,).
* <brokers> is the Kafka address for obtaining metadata.
*/
import org.apache.kafka.clients.producer.ProducerRecord
import com.huawei.spark.streaming.kafka010.KafkaWriter._
import scala.collection.mutable
import scala.language.postfixOps
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.{Milliseconds, StreamingContext}

object DstreamKafkaWriter {
def main(args: Array[String]) {

if (args.length != 3) {
System.err.println("Usage: DstreamKafkaWriter <groupId> <brokers> <topic>")
System.exit(1)
}

val Array(groupId, brokers, topic) = args
val sparkConf = new SparkConf().setAppName("KafkaWriter")

// Fill the properties of Kafka.
val kafkaParams = Map[String, String](
"bootstrap.servers" -> brokers,
"value.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
"key.deserializer" -> "org.apache.kafka.common.serialization.StringDeserializer",
"value.serializer" -> "org.apache.kafka.common.serialization.ByteArraySerializer",
"key.serializer" -> "org.apache.kafka.common.serialization.StringSerializer",
"group.id" -> groupId,
"auto.offset.reset" -> "latest"
)

// Create the context of Streaming.
val ssc = new StreamingContext(sparkConf, Milliseconds(500));

// Add data to be written to Kafka.
val sendData = Seq("kafka_writer_test_msg_01", "kafka_writer_test_msg_02",
"kafka_writer_test_msg_03")

// Create an RDD queue.
val sent = new mutable.Queue[RDD[String]]()
sent.enqueue(ssc.sparkContext.makeRDD(sendData))

// Create a DStream for writing data.
val wStream = ssc.queueStream(sent)

// Write data to Kafka.
wStream.writeToKafka(kafkaParams,
(x: String) => new ProducerRecord[String, Array[Byte]](topic, x.getBytes))

// Start Streaming.
ssc.start()
ssc.awaitTermination()
}
}
```

Modify the Producer sample code of Java and Scala. The following code snippet is for reference only.

```
StreamingExampleProducer.java
/**
* Parameter description:
* <topics> is topics subscribed in Kafka. Multiple topics are separated by commas (,).
* <brokers> is the Kafka address for obtaining metadata.
*/
import org.apache.kafka.clients.producer.KafkaProducer;
```

```
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
import java.io.*;import java.util.Properties;

public class StreamingExampleProducer {
public static void main(String[] args) throws IOException {
if (args.length < 2) {
printUsage();
}
String brokerList = args[0];
String topic = args[1];
String filePath = "/home/data/";
Properties props = new Properties();
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokerList);
props.put(ProducerConfig.CLIENT_ID_CONFIG, "DemoProducer");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
Producer<String, String> producer = new KafkaProducer<String, String>(props);

for (int m = 0; m < Integer.MAX_VALUE / 2; m++) {
File dir = new File(filePath);
File[] files = dir.listFiles();
if (files != null) {
for (File file : files) {
if (file.isDirectory()) {
System.out.println(file.getName() + "This is a directory!");
} else {
BufferedReader reader = null;
reader = new BufferedReader(new FileReader(filePath + file.getName()));
String tempString = null;
while ((tempString = reader.readLine()) != null) {
// Judge empty lines.
if (!tempString.isEmpty()) {
producer.send(new ProducerRecord<String, String>(topic, tempString));
}
}
// Check that the stream is closed.
reader.close();
}
}
}
try {
Thread.sleep(3);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}

private static void printUsage() {
System.out.println("Usage: {brokerList} {topic}");
}
```

## 2.18 YARN Development Guide

### 2.18.1 Overview

#### Intended Audience

This document is intended for development personnel who are experienced in Java development and want to develop YARN applications.

## YARN Introduction

YARN is a distributed resource management system that is used to improve resource usage in the distributed cluster environment. Resources include memory, I/O, network, and disk resources. YARN is developed to address the shortage of the original MapReduce framework. At the beginning, MapReduce committers periodically modified existing codes. As codes increase and because the original MapReduce framework was designed improperly, modification on the original MapReduce framework becomes more difficult. Therefore, MapReduce committers decided to re-design the MapReduce framework to provide a next-generation MapReduce (MRv2/Yarn) framework that supports high scalability, availability, reliability, backward compatibility, and resource usage. The next-generation MapReduce (MRv2/Yarn) framework supports more computing frameworks in addition to the MapReduce framework.

## Basic Concepts

- **ResourceManager (RM)**

ResourceManager is a global resource manager that manages and allocates resources in the system. ResourceManager consists of two components: Scheduler and Applications Manager.

- **ApplicationMaster (AM)**

Each application submitted by users includes an ApplicationMaster. The ApplicationMaster provides the following functions:

- Negotiates with the ResourceManager Scheduler to obtain resources (represented by Containers).
- Allocates resource to internal tasks.
- Communicates with NodeManager to start or stop tasks.
- Monitors all tasks with the running status, and applies for resources again for tasks when tasks fail to run to restart the tasks.

- **NodeManager (NM)**

NodeManager is the resource and task manager of each node. On one hand, NodeManager periodically reports resource usage of the local node and the running status of each Container to ResourceManager. On the other hand, NodeManager receives and processes requests from ApplicationMaster for starting or stopping Containers.

- **Container**

Container is a resource abstract in YARN. Container encapsulates multidimensional resources of a node, such as memory, CPU, disk, and network resources. When ApplicationMaster applies for resources from ResourceManager, ResourceManager returns resources in a Container to ApplicationMaster.

## 2.18.2 Interfaces

### 2.18.2.1 Command

You can use YARN commands to perform operations on YARN clusters, such as starting ResourceManager, submitting applications, killing applications, querying node status, and downloading container logs.

For details about the commands, see:

[http://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/  
YarnCommands.html](http://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/YarnCommands.html)

## Common Commands

YARN commands can be used by common users and administrators. Common users can run a few of YARN commands, such as **jar** and **logs**. Administrators have permissions to run most YARN commands.

Users can run the following command to query usage of YARN:

**yarn --help**

Usage: Go to any directory on the YARN client, execute source command to import the environment variables, and run the command. The command format is as follows:

**yarn [--config *confdir*] *COMMAND***

**Table 2-185** describes the commands.



The version 8.3.1 is used as an example. Replace it with the actual version number.

**Table 2-185** Common commands

Command	Description
resourcemanager	<p>Runs a ResourceManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user <b>omm</b> (the client must have the execute permission of user <b>omm</b>), export environment variables first.</p> <ul style="list-style-type: none"><li>• <b>export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_ResourceManager/etc</b></li><li>• <b>export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_ResourceManager/etc</b></li></ul>
nodemanager	<p>Runs a NodeManager.</p> <p>Notice: Before running commands, for example, the following two commands, on the server as user <b>omm</b> (the client must have the execute permission of user <b>omm</b>), export environment variables first.</p> <ul style="list-style-type: none"><li>• <b>export YARN_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_NodeManager/etc</b></li><li>• <b>export HADOOP_CONF_DIR=\${BIGDATA_HOME}/FusionInsight_HD_8.3.1/1_10_NodeManager/etc</b></li></ul>

Command	Description
rmadmin	Runs administrator's tool for dynamically updating information.
version	Displays the version.
jar <jar>	Runs a JAR file.
logs	Obtains container logs.
classpath	Displays the class path of the Hadoop JAR package and other library files.
daemonlog	Obtains or sets the service log level.
CLASSNAME	Runs a class named by <i>CLASSNAME</i> .
top	Runs the cluster usage monitor tool.
-Dmapreduce.job.hdfs-servers	If OBS is connected, but the server still uses HDFS, you need to explicitly use this parameter in the command line to specify the HDFS address. The format is <b>hdfs:// {NAMESERVICE}</b> . Replace <i>NAMESERVICE</i> with the HDFS NameService name.  If the current HDFS has multiple NameService instances, you need to specify all NameService instances and separate them with commas (,), for example, <b>hdfs://nameservice1,hdfs://nameservice2</b> .

## Superior Scheduler Command

Superior Scheduler Engine provides a CLI that displays detail information of Superior Scheduler Engine. For executing superior command we need to use the "**<HADOOP\_HOME>/bin/superior**" script.

Here is the format of "superior" command.

```
<HADOOP_HOME>/bin/superior
```

```
Usage: superior [COMMAND | -help]
Where COMMAND is one of:
resourcepool           prints resource pool status
queue                  prints queue status
application            prints application status
policy                 prints policy status
```

Most commands print help when invoked without parameters.

- Superior **resourcepool** command :

This command displays resourcepool and associated policy related status and configuration information.

### NOTE

Superior resourcepool command can be used only by admin user and user with yarn admin rights.

Usage output:

```
>superior resourcepool

Usage: resourcepool [-help]
    [-list]
    [-status <resourcepoolname>]
-helpr          prints resource pool usage
-list           prints all resource pool summary report
-status <resourcepoolname> prints status and configuration of specified
               resource pool
```

- **resourcepool -list** prints resource pool summary in a table format. Here is an example:

```
> superior resourcepool -list
NAME      NUMBER_MEMBER      TOTAL_RESOURCE      AVAILABLE_RESOURCE
Pool1      4                  vcores 30,memory 1000  vcores 21,memory 80
Pool2      100                 vcores 100,memory 12800 vcores 30,memory 1000
default     2                  vcores 64,memory 128   vcores 40,memory 28
```

- **resourcepool -status <resourcepoolname>** prints resource pool detail information in a list format. Here is an example:

```
> superior resourcepool -status default
NAME: default
DESCRIPTION: System generated resource pool
TOTAL_RESOURCE: vcores 64,memory 128
AVAILABLE_RESOURCE: vcores 40,memory 28
NUMBER_MEMBER: 2
MEMBERS: node1,node2
CONFIGURATION:
|-- RESOURCE_SELECT:
|  _RESOURCES:
```

- Superior **queue** command

This command displays hierarchy queue information.

Usage output:

```
>superior queue
```

```
Usage: queue [-help]
    [-list] [-e] [[-name <queue_name>] [-r|-c]]
    [-status <queue_name>]
-c            only work with -name <queue_name> option. If this
             option is used, command will print information of
             specified queue and its direct children.
-e            only work with -list or -list -name option. If
             this option is used, command will print effective
             state of specified queue and all of its
             descendants.
-help         prints queue sub command usage
-list         prints queue summary report. This option can work
             with -name <queue_name> and -r options.
-name <queue_name>  print specified queue, this can work with -r
             option. By default, it will print queue's own
             information. When -r is defined, command will
             print all of its descendant queues. When -c is
             defined, it will print its direct children queues.
-r            only work with -name <queue_name> option. If this
             option is used, command will print information of
             specified queue and all of its descendants.
-status <queue_name> prints status of specified queue
```

- **queue -list** prints a table format of queue summary information. When command displays, command will display them based on queue hierarchy fashion. With SUBMIT ACL or ADMIN ACL rights for queue, user will be able to see queues. Here is an example:

```
> superior queue -list
NAME      STATE      NRUN_APP      NPEND_APP      NRUN_CONTAINER
NPEND_REQUEST  RES_INUSE      RES_REQUEST
```

root	OPEN ACTIVE	10	20	100	200	vcores 100,memory
1000	vcores 200,memory 2000					
root.Q1	OPEN ACTIVE	5	10	50	100	vcores 50,memory
500	vcores 100,memory 1000					
root.Q1.Q11	OPEN ACTIVE	5	10	50	100	vcores 50,memory
500	vcores 100,memory 1000					
root.Q1.Q12	CLOSE INACTIVE	0	0	0	0	vcores 0,memory
0	vcores 0,memory 0					
root.Q2	OPEN INACTIVE	5	10	50	100	vcores 50,memory
500	vcores 100,memory 1000					
root.Q2.Q21	OPEN ACTIVE	5	10	50	100	vcores 50,memory
500	vcores 100,memory 1000					

- **queue -list -name root.Q1** will display root.Q1 only.

> superior queue -list -name root.Q1	NAME	STATE	NRUN_APP	NPEND_APP	NRUN_CONTAINER	
	NPEND_REQUEST	RES_INUSE		RES_REQUEST		
	root.Q1	OPEN ACTIVE	5	10	50	100
	500	vcores 100,memory 1000				vcores 50,memory

- **queue -list -name root.Q1 -r** will print out root.Q1 and all of its descendants.

> superior queue -list -name root.Q1 -r	NAME	STATE	NRUN_APP	NPEND_APP	NRUN_CONTAINER	
	NPEND_REQUEST	RES_INUSE		RES_REQUEST		
	root.Q1	OPEN ACTIVE	5	10	50	100
	500	vcores 100,memory 1000				vcores 50,memory
	root.Q1.Q11	OPEN ACTIVE	5	10	50	100
	500	vcores 100,memory 1000				vcores 50,memory
	root.Q1.Q12	CLOSE INACTIVE	0	0	0	0
	0	vcores 0,memory 0				

- **queue -list -name root -c** will print out root and all of its direct children

> superior queue -list -name root -c	NAME	STATE	NRUN_APP	NPEND_APP	NRUN_CONTAINER	
	NPEND_REQUEST	RES_INUSE		RES_REQUEST		
	root	OPEN ACTIVE	10	20	100	200
	100,memory 1000	vcores 200,memory 2000				vcores
	root.Q1	OPEN ACTIVE	5	10	50	100
	50,memory 500	vcores 100,memory 1000				vcores
	root.Q2	OPEN INACTIVE	5	10	50	100
	50,memory 500	vcores 100,memory 1000				vcores

- **queue -status <queue\_name>** will display detail queue status and configuration.

With SUBMIT ACL, user will be able to see details except ACLS of queue.

User with ADMIN ACL rights for queue will be able to see queue details including ACL.

> superior queue -status root.Q1	NAME: root.Q1					
	OPEN_STATE:CLOSED					
	ACTIVE_STATE: INACTIVE					
	EOPEN_STATE: CLOSED					
	EACTIVE_STATE: INACTIVE					
	LEAF_QUEUE: Yes					
	NUMBER_PENDING_APPLICATION: 100					
	NUMBER_RUNNING_APPLICATION: 10					
	NUMBER_PENDING_REQUEST: 10					
	NUMBER_RUNNING_CONTAINER: 10					
	NUMBER_RESERVED_CONTAINER: 0					
	RESOURCE_REQUEST: vcores 3,memory 3072					
	RESOURCE_INUSE: vcores 2,memory 2048					
	RESOURCE_RESERVED: vcores 0,memory 0					
	CONFIGURATION:					
	-- DESCRIPTION: Spark session queue					
	-- MAX_PENDING_APPLICATION: 10000					
	--MAX_RUNNING_APPLICATION: 1000					
	--ALLOCATION_ORDER_POLICY: FIFO					

```
--DEFAULT_RESOURCE_SELECT: label1
|--MAX_MASTER_SHARE: 10%
|--MAX_RUNNING_APPLICATION_PER_USER : -1
|--MAX_ALLOCATION_UNIT: vcores 32,memory 12800
|--ACL_USERS: user1,user2
|--ACL_USERGROUPS: usergroup1,usergroup2
|-- ACL ADMINS: user1
|--ACL_ADMINGROUPS: usergroup1
```

- Superior **application** command

This command displays application related information.

Usage output:

```
>superior application
```

```
Usage: application [-help]
    [-list]
    [-status <application_id>]
-helpprints application sub command usage
-listprints all application summary report
-status <application_id> prints status of specified application
```

With the view access rights to application user can see application related information.

- **application -list** provides summary information of all application in a table format. Here is an example:

```
> superior application -list
ID          QUEUE      USER   NRUN_CONTAINER
NPEND_REQUEST    NRSV_CONTAINER   RES_INUSE
RES_REQUEST     RES_RESERVED
application_1482743319705_0005    root.SEQ.queueB  hbase  1
100           0           vcores 1,memory 1536      vcores 2000,memory
409600         vcores 0,memory 0
application_1482743319705_0006    root.SEQ.queueB  hbase  0
1             0           vcores 0,memory 0      vcores 1,memory
1536           vcores 0,memory 0
```

- **application -status <app\_id>** command displays detail information of specified application. Here is an example:

```
> superior application -status application_1443067302606_0609
ID: application_1443067302606_0609
QUEUE: root.Q1.Q11
USER: cchen
RESOURCE_REQUEST: vcores 3,memory 3072
RESOURCE_INUSE: vcores 2,memory 2048
RESOURCE_RESERVED:vcores 1, memory 1024
NUMBER_RUNNING_CONTAINER: 2
NUMBER_PENDING_REQUEST: 3
NUMBER_RESERVED_CONTAINER: 1
MASTER_CONTAINER_ID: application_1443067302606_0609_01
MASTER_CONTAINER_RESOURCE: node1.domain.com
BLACKLIST: node5,node8
DEMANDS:
|-- PRIORITY: 20
|-- MASTER: true
|-- CAPABILITY: vcores 2, memory 2048
|-- COUNT: 1
|-- RESERVED_RES : vcores 1, memory 1024
|-- RELAXLOCALITY: true
|-- LOCALITY: node1/1
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "application limit reached"
|-- ID: application_1443067302606_0609_03
|-- RESOURCE: node1.domain.com
|-- RESERVED_RES: vcores 1, memory 1024
|
|--PRIORITY: 1
|-- MASTER: false
```

```
-- CAPABILITY: vcores 1,memory 1024
|-- COUNT: 2
|-- RESERVED_RES: vcores 0, memory 0
|-- RELAXLOCLITY: true
|-- LOCALITY: node1/1,node2/1,rackA/2
|-- RESOURCESELECT: label1
|-- PENDINGREASON: "no available resource"
CONTAINERS:
|-- ID: application_1443067302606_0609_01
|-- RESOURCE: node1.domain.com
|-- CAPABILITY: vcores 1,memory 1024
|
|-- ID: application_1443067302606_0609_02
|-- RESOURCE: node2.domain.com
|-- CAPABILITY: vcores 1,memory 1024
```

- Superior **policy** command

This command displays policy related information.

 **NOTE**

Superior policy command can be used only by admin user and user with yarn admin rights.

Usage output:

```
>superior policy

Usage: policy [-help]
    [-list <resourcepoolname>] [-u] [-detail]
    [-status <resourcepoolname>]
-detailed      only work with -list option to show a
               summary information of resource pool
               distribution on queues, including reserve,
               minimum and maximum
-help          prints policy sub command usage
-list <resourcepoolname>   prints a summary information of resource
                           pool distribution on queue
-status <resourcepoolname> prints pool distribution policy
                           configuration and status of specified
                           resource pool
-u             only work with -list option to show a
               summary information of resource pool
               distribution on queues and also user
               accounts
```

- **policy -list <resourcepoolname>** print out a summary of queue distribution information. Here is an example:

```
>superior policy -list default
NAME: default
TOTAL_RESOURCE: vcores 16,memory 16384
AVAILABLE_RESOURCE: vcores 16,memory 16384

NAMERES_INUSERES_REQUEST
root.defaultvcores 0,memory 0vcores 0,memory 0
root.productionvcores 0,memory 0vcores 0,memory 0
root.production.BU1vcores 0,memory 0vcores 0,memory 0
root.production.BU2 vcores 0,memory 0vcores 0,memory 0
```

- **policy -list <resourcepoolname> -u** prints out also user level summary information

```
> superior policy -list default -u
NAME: default
TOTAL_RESOURCE: vcores 16,memory 16384
AVAILABLE_RESOURCE: vcores 16,memory 16384

NAMERES_INUSERES_REQUEST
root.defaultvcores 0,memory 0vcores 0,memory 0
root.default.[_others_]vcores 0,memory 0vcores 0,memory 0
root.productionvcores 0,memory 0vcores 0,memory 0
```

```
root.production.BU1vcores 0,memory 0vcores 0,memory 0  
root.production.BU1.[_others_]vcores 0,memory 0vcores 0,memory 0  
root.production.BU2vcores 0,memory 0vcores 0,memory 0  
root.production.BU2.[_others_]vcores 0,memory 0vcores 0,memory 0
```

- **policy -status <resourcepoolname>** print out policy detail of specified resource pool. Here is an example:  
> superior policy -status pool1  
NAME: pool1  
TOTAL\_RESOURCE: vcores 64,memory 128  
AVAILABLE\_RESOURCE: vcores 40,memory 28  
QUEUES:  
|-- NAME: root.Q1  
|-- RESOURCE\_USE: vcores 20, memory 1000  
|-- RESOURCE\_REQUEST: vcores 2,memory 100  
|--RESERVE: vcores 10, memory 4096  
|--MINIMUM: vcore 11, memory 4096  
|--MAXIMUM: vcores 500, memory 100000  
|--CONFIGURATION:  
|-- SHARE: 50%  
|-- RESERVE: vcores 10, memory 4096  
|-- MINIMUM: vcores 11, memory 4096  
|-- MAXIMUM: vcores 500, memory 100000  
|-- QUEUES:  
|-- NAME: root.Q1.Q11  
|-- RESOURCE\_USE: vcores 15, memory, 500  
|-- RESOURCE\_REQUEST: vcores 1, memory 50  
|-- RESERVE: vcores 0, memory 0  
|-- MINIMUM: vcores 0, memory 0  
|-- MAXIMUM: vcores -1, memory -1  
|-- USER\_ACCOUNTS:  
|-- NAME: user1  
|-- RESOURCE\_USE: vcores 1, memory 10  
|-- RESOURCE\_REQUEST: vcores 1, memory 50  
|  
|-- NAME: OTHERS  
|--RESOURCE\_USE: vcores 0, memory 0  
|-- RESOURCE\_REQUEST: vcores 0, memory 0  
|-- CONFIGURATION:  
|-- SHARE: 100%  
|-- USER\_POLICY:  
|-- NAME: user1  
|-- WEIGHT: 10  
|  
|-- NAME: OTHERS  
|-- WEIGHT: 1  
|-- MAXIMUM: vcores 10, memory 1000

## 2.18.2.2 Java API

For details about YARN application programming interfaces (APIs), see  
<http://hadoop.apache.org/docs/r3.3.1/api/index.html>.

## Common Interfaces

Common YARN Java classes are as follows:

- **ApplicationClientProtocol**

This class is used between the client and ResourceManager. The client submits applications to ResourceManager, queries the running status of applications, and kills applications using this protocol.

**Table 2-186** Common interfaces of ApplicationClientProtocol

Interface	Description
forceKillApplication(KillApplicationRequest request)	The client requests ResourceManager to stop a submitted task through this interface.
getApplicationAttemptReport(GetApplicationAttemptReportRequest request)	The client obtains the report of specified ApplicationAttempt from ResourceManager through this interface.
getApplicationAttempts(GetApplicationAttemptsRequest request)	The client obtains the report of all ApplicationAttempts from ResourceManager through this interface.
getApplicationReport(GetApplicationReportRequest request)	The client obtains an application report from ResourceManager through this interface.
getApplications(GetApplicationsRequest request)	The client obtains information about applications that meet filtering conditions from ResourceManager through this interface.
getClusterMetrics(GetClusterMetricsRequest request)	The client obtains metrics of clusters from ResourceManager through this interface.
getClusterNodes(GetClusterNodesRequest request)	The client obtains information about all nodes in a cluster from ResourceManager through this interface.
getContainerReport(GetContainerReportRequest request)	The client obtains the report of a Container from ResourceManager through this interface.
getContainers(GetContainersRequest request)	The client obtains the report of all Containers of an ApplicationAttempt from ResourceManager through this interface.
getDelegationToken(GetDelegationTokenRequest request)	The client obtains the delegation token through this interface. The delegation token is used by the container to access related services.
getNewApplication(GetNewApplicationRequest request)	The client obtains a new application ID through this interface for submitting a new application.
getQueueInfo(GetQueueInfoRequest request)	The client obtains queue information from ResourceManager through this interface.
getQueueUserAcls(GetQueueUserAclsInfoRequest request)	The client obtains queue access permission information about the current user from ResourceManager through this interface.

Interface	Description
moveApplicationAcrossQueues(MoveApplicationAcrossQueuesRequest request)	This interface is used to move an application to a new queue.
submitApplication(SubmitApplicationRequest request)	The client submits a new application to ResourceManager through this interface.

- ApplicationMasterProtocol

This class is used between ApplicationMaster and ResourceManager. ApplicationMaster registers with ResourceManager, and applies for resources and obtains the running status of each task from ResourceManager using this protocol.

**Table 2-187** Common interfaces of ApplicationMasterProtocol

Interface	Description
allocate(AllocateRequest request)	ApplicationMaster submits a resource allocation request through this interface.
finishApplicationMaster(FinishApplicationMasterRequest request)	ApplicationMaster notifies ResourceManager of running success or failure through this interface.
registerApplicationMaster(RegisterApplicationMasterRequest request)	ApplicationMaster registers with ResourceManager through this interface.

- ContainerManagementProtocol

This class is used between ApplicationMaster and NodeManager. ApplicationMaster requests NodeManager to start or terminate a Container or queries the Container running status using this protocol.

**Table 2-188** Common interfaces of ContainerManagementProtocol

Interface	Description
getContainerStatuses(GetContainerStatusesRequest request)	ApplicationMaster obtains the current status of the Containers from NodeManager through this interface.
startContainers(StartContainersRequest request)	ApplicationMaster requests NodeManager to start Containers through this interface.
stopContainers(StopContainersRequest request)	ApplicationMaster requests NodeManager to stop a series of allocated Containers through this interface.

### 2.18.2.3 REST API

#### Function Description

You can use HTTP REST APIs to query more information about Yarn jobs. Currently, you can only query some resources or jobs using REST APIs provided by Yarn. For details of HTTP REST APIs, see the following official guidelines:

[http://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/  
WebServicesIntro.html](http://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/WebServicesIntro.html)

#### Preparing Running Environment

1. Install a client on the node. For example, install a client in the **/opt/hadoopclient** directory. See details in "Installing a Client."
2. Go to the **/opt/hadoopclient** directory where the client is installed and run the following commands to initiate environment variables:  
**source bigdata\_env**

#### Procedure

1. Query the information about jobs that run on the Yarn.

- Command:

```
curl -k -i --negotiate -u : "http://10-120-85-2:26000/ws/v1/cluster/apps/"
```

##### NOTE

- **10-120-85-2**: host name of the active ResourceManager node.

You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the **Host Name of ResourceManager(Active)**.

- **26001**: port number of the ResourceManager.

You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and obtain the value of **yarn.resourcemanager.webapp.port**.

- The running result is displayed as follows:

```
{
  "apps": {
    "app": [
      {
        "id": "application_1461743120947_0001",
        "user": "spark",
        "name": "Spark-JDBCServer",
        "queue": "default",
        "state": "RUNNING",
        "finalStatus": "UNDEFINED",
        "progress": 10,
        "trackingUI": "ApplicationMaster",
        "trackingUrl": "http://10-120-85-2:26000/proxy/application_1461743120947_0001/",
        "diagnostics": "AM is launched. ",
        "clusterId": 1461743120947,
        "applicationType": "SPARK",
        "applicationTags": "",
        "startedTime": 1461804906260,
        "finishedTime": 0,
        "elapsedTime": 6888848,
        "amContainerLogs": "https://10-120-85-2:26000/node/containerlogs/container_e12_1461743120947_0001_01_000001/spark",
      }
    ]
  }
}
```

```
"amHostHttpAddress": "10-120-85-2:26000",
"allocatedMB": 1024,
"allocatedVcores": 1,
"runningContainers": 1,
"memorySeconds": 7053309,
"vcoreSeconds": 6887,
"preemptedResourceMB": 0,
"preemptedResourceVcores": 0,
"numNonAMContainerPreempted": 0,
"numAMContainerPreempted": 0,
"resourceRequests": [
{
  "capability": {
    "memory": 1024,
    "virtualCores": 1
  },
  "nodeLabelExpression": "",
  "numContainers": 0,
  "priority": {
    "priority": 0
  },
  "relaxLocality": true,
  "resourceName": "*"
},
],
"logAggregationStatus": "NOT_START",
"amNodeLabelExpression": ""
},
{
  "id": "application_1461722876897_0002",
  "user": "admin",
  "name": "QuasiMonteCarlo",
  "queue": "default",
  "state": "FINISHED",
  "finalStatus": "SUCCEEDED",
  "progress": 100,
  "trackingUI": "History",
  "trackingUrl": "http://10-120-85-2:26000/proxy/application_1461722876897_0002/",
  "diagnostics": "Attempt recovered after RM restart",
  "clusterId": 1461743120947,
  "applicationType": "MAPREDUCE",
  "applicationTags": "",
  "startedTime": 1461741052993,
  "finishedTime": 1461741079483,
  "elapsedTime": 26490,
  "amContainerLogs": "http://10-120-85-2:26000/node/containerlogs/
container_e11_1461722876897_0002_01_000001/admin",
  "amHostHttpAddress": "10-120-85-2:26000",
  "allocatedMB": -1,
  "allocatedVcores": -1,
  "runningContainers": -1,
  "memorySeconds": 158664,
  "vcoreSeconds": 52,
  "preemptedResourceMB": 0,
  "preemptedResourceVcores": 0,
  "numNonAMContainerPreempted": 0,
  "numAMContainerPreempted": 0,
  "amNodeLabelExpression": ""
}
]
}
```

- Result analysis:

On the interface, you can query the information of jobs that are running on the Yarn and obtain the common information that is displayed in **Table 1**.

**Table 2-189** Common information of jobs running on Yarn

Information	Description
user	Indicates the user who runs the job.
applicationType	Indicates the application types, such as MAPREDUCE or SPARK.
finalStatus	Indicates whether a job is executed successfully.
elapsedTime	Indicates the time to run a job.

## 2. Obtain the overall information of Yarn resources.

- Command:

```
curl -k -i --negotiate -u : "http://10-120-85-102:26000/ws/v1/cluster/metrics"
```

- The running result is displayed as follows:

```
{
  "clusterMetrics": {
    "appsSubmitted": 2,
    "appsCompleted": 1,
    "appsPending": 0,
    "appsRunning": 1,
    "appsFailed": 0,
    "appsKilled": 0,
    "reservedMB": 0,
    "availableMB": 23552,
    "allocatedMB": 1024,
    "reservedVirtualCores": 0,
    "availableVirtualCores": 23,
    "allocatedVirtualCores": 1,
    "containersAllocated": 1,
    "containersReserved": 0,
    "containersPending": 0,
    "totalMB": 24576,
    "totalVirtualCores": 24,
    "totalNodes": 3,
    "lostNodes": 0,
    "unhealthyNodes": 0,
    "decommissionedNodes": 0,
    "rebootedNodes": 0,
    "activeNodes": 3,
    "rmMainQueueSize": 0,
    "schedulerQueueSize": 0,
    "stateStoreQueueSize": 0
  }
}
```

- Result analysis:

On the interface, users can query the common information of jobs that are running in the cluster, as displayed in [Table 2](#).

**Table 2-190** Common information of jobs running in the cluster

Information	Description
appsSubmitted	Indicates the number of jobs that have been submitted.

Information	Description
appsCompleted	Indicates the number of jobs that have been completed.
appsPending	Indicates the number of jobs that have been suspended.
appsRunning	Indicates the number of jobs that are running.
appsFailed	Indicates the number of jobs that have failed.
appsKilled	Indicates the number of jobs that have been killed.
totalMB	Indicates the total memory of Yarn resources.
totalVirtualCores	Indicates the total VCores of Yarn resources.

## 2.18.2.4 REST APIs of Superior Scheduler

### Function Description

The REST/HTTP server is part of Superior Scheduler on YARN Resource Manager host and leverage existing YARN resource manager web service port. In the section below, we will denote this YARN *address:port* as *SS\_REST\_SERVER*.

Below uses HTTP as part of URL and only HTTP will be supported.

### Superior Scheduler Interfaces

- **Query Application**
  - Query \*all\* application within scheduler engine.
    - URL  
GET `http://<SS_REST_SERVER>/ws/v1/sscheduler/applications/list`
    -  **NOTE**

**SS\_REST\_SERVER** indicates *ResourceManager IP address:Port number*.

      - ResourceManager IP address: You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Instance**, and view the service IP address of any ResourceManager.
      - Port: HTTPS port number of the ResourceManager. You can log in to FusionInsight Manager, choose **Cluster > Services > Yarn > Configurations > All Configurations**, search for and view the value of `yarn.resourcemanager.webapp.port`.
  - Input  
None.

- **Output**

JSON Response:

```
{  
  "applicationlist": [  
    {  
      "id": "1020201_0123_12",  
      "queue": "root.Q1.Q11",  
      "user": "cchen",  
      "resource_request": {  
        "vcores" : 10,  
        "memory" : 100  
      },  
      "resource_inuse": {  
        "vcores" : 100,  
        "memory" : 2000  
      },  
      "number_running_container": 100,  
      "number_pending_request": 10  
    },  
    {  
      "id": "1020201_0123_15",  
      "queue": "root.Q2.Q21",  
      "user": "Jason",  
      "resource_request": {  
        "vcores" : 4,  
        "memory" : 100  
      },  
      "resource_inuse": {  
        "vcores" : 20,  
        "memory" : 200  
      },  
      "resource_reserved": {  
        "vcores" : 10,  
        "memory" : 100  
      },  
      "number_running_container": 20,  
      "number_pending_container": 4,  
      "number_reserved_container": 2  
    }  
  ]  
}
```

**Table 2-191** Parameters of all application

Attribute	Type	Description
applicationlist	array	Array of application IDs.
queue	String	Name of queue application.
user	String	Name of user who submits application.
resource_request	object	Currently requested resource, including vcores, memory, and so on.
resource_inuse	object	Currently resource in use, including vcores, memory, and so on.
resource_reserved	object	Currently reserved resource, including vcores, memory, and so on.
number_running_container	int	Total number of running containers. This maps to superior engine decisions.

Attribute	Type	Description
number_pending_request	int	Total number of pending requests, this is sum of all counts in all demands of allocation.
number_reserved_container	int	Total number of reserved containers, this maps to superior engine decisions.
id	String	Application ID.

- Query \*single\* application within scheduler engine.

- URL

```
GET http://<SS_REST_SERVER>/ws/v1/sscheduler/applications/{application_id}
```

- Input

None.

- Output

JSON Response:

```
{
  "applicationlist": [
    {
      "id": "1020201_0123_12",
      "queue": "root.Q1.Q11",
      "user": "cchen",
      "resource_request": {
        "vcores" : 3,
        "memory" : 3072
      },
      "resource_inuse": {
        "vcores" : 100,
        "memory" : 2048
      },
      "number_running_container": 2,
      "number_pending_request": 3,
      "number_reserved_container": 1,
      "master_container_id": 23402_3420842,
      "master_container_resource": node1.domain.com
      "blacklist": [
        {
          "resource": "node5"
        },
        {
          "resource": "node8"
        }
      ],
      "demand": [
        {
          "priority": 1,
          "ismaster": true,
          "capability": {
            "vcores": 2,
            "memory": 2048
          },
          "count": 1,
          "relaxlocality": true,
          "locality": [
            {
              "target": "node1",
              "count": 1,
            }
          ]
        }
      ]
    }
  ]
}
```

```
        "strict": false
    }
],
"resourceselect": "label1",
"pending_reason": "application limit reached",
"reserved_resource": {
    "vcores":1,
    "memory":1024
},
"reservations":[
    "id": "23402_3420878",
    "resource": "node1.domain.com",
    "reservedAmount": {
        "vcores":1,
        "memory":1024
    }
]
},
{
    "priority": 1,
    "ismaster": false,
    "capability": {
        "vcores": 1,
        "memory": 1024
    },
    "count": 2,
    "relaxlocality": true,
    "locality": [
        {
            "target": "node1",
            "count": 1,
            "strict": false
        },
        {
            "target": "node2",
            "count": 1,
            "strict": false
        },
        {
            "target": "rackA",
            "count": 2,
            "strict": false
        }
    ],
    "resourceselect": "label1",
    "pending_reason": "no available resource"
}
],
"containers": [
    {
        "id": "23402_3420842",
        "resource": "node1.domain.com",
        "capability": {
            "vcores": 1,
            "memory": 1024
        }
    },
    {
        "id": "23402_3420853",
        "resource": "node2.domain.com",
        "capability": {
            "vcores": 1,
            "memory": 1024
        }
    }
]
```

- Exceptions  
Application not found.

**Table 2-192** Parameters of single application

Attribute	Type	Description
application	object	Application object.
id	String	Application ID.
queue	String	Name of queue application.
user	String	Name of user who submits application.
resource_request	object	Currently requested resource, including vcores, memory, and so on.
resource_inuse	object	Currently resource in use, including vcores, memory, and so on.
resource_reserved	object	Currently reserved resource, including vcores, memory, and so on.
number_running_container	int	Total number of running containers. This maps to superior engine decisions.
number_pending_request	int	Total number of pending requests, this is sum of all counts in all demands of allocation.
number_reserved_container	int	Total number of reserved containers, this maps to superior engine decisions.
master_container_id	String	The master container ID.
master_container_resource	String	The host name that master container running on.
demand	array	Array of demand objects.
priority	int	Priority of demand.
ismaster	boolean	Is the demand corresponding to "application master".
capability	object	Capability object.
vcores, memory, ..	int	Numeric consumable resource attributes, defining the allocation "unit" for this demand.
count	int	Number of unit required.
relaxlocality	boolean	Locality requirement is preference, and not mandatory if cannot be satisfied.

Attribute	Type	Description
locality	object	Locality object.
target	string	Locality target's name (that is, node1, rack1..).
count	int	Number of resource "unit" required with this locality requirement.
strict	boolean	If this particular locality is mandatory or not.
resourceselect	String	Resource selection expression for the demand.
pending_reason	String	Reason why this demand is outstanding.
resource_reserved	object	Currently reserved resource for this demand, including vcores, memory, and so on.
reservations	array	Array of reserved container objects.
reservations:id	String	Reserved container ID.
reservations:resource	String	Where the container is allocated.
reservations:reserveAmount	object	The reserved amount of this reservation.
containers	array	Array of allocated container objects.
containers:id	String	Container ID.
containers:resource	String	Where the container is allocated.
containers:capability	object	Capability object.
containers:vcores, memory...	int	Numeric consumable resource attributes allocated to this container.

- Query Queue
  - Query \*all\* queues within scheduler engine, including leaf and all middle queues.
    - URL  
GET http://<SS\_REST\_SERVER>/ws/v1/sscheduler/queues/list
    - Input  
None.
    - Output  
JSON Response:

```
{  
    "queuelist": [  
        {  
            "name": "root.default",  
            "eopen_state": "OPEN",  
            "eactive_state": "ACTIVE",  
            "open_state": "OPEN",  
            "active_state": "ACTIVE",  
            "number_pending_application": 2,  
            "number_running_application": 10,  
            "number_pending_request": 2,  
            "number_running_container": 10,  
            "number_reserved_container": 1,  
            "resource_inuse" {  
                "vcores": 10,  
                "memory": 10240  
            },  
            "resource_request" {  
                "vcores": 2,  
                "memory": 2048  
            },  
            "resource_reserved" {  
                "vcores": 1  
                "memory": 1024  
            }  
        },  
        {  
            "name": "root.dev",  
            "eopen_state": "OPEN",  
            "eactive_state": "INACTIVE",  
            "open_state": "OPEN",  
            "active_state": "INACTIVE",  
            "number_pending_application": 2,  
            "number_running_application": 10,  
            "number_pending_request": 2,  
            "number_running_container": 10,  
            "number_reserved_container": 0,  
            "resource_inuse" {  
                "vcores": 10,  
                "memory": 10240  
            },  
            "resource_request" {  
                "vcores": 2,  
                "memory": 2048  
            },  
            "resource_reserved" {  
                "vcores": 0  
                "memory": 0  
            }  
        },  
        {  
            "name": "root.qa",  
            "eopen_state": "CLOSED",  
            "eactive_state": "ACTIVE",  
            "open_state": "CLOSED",  
            "active_state": "ACTIVE",  
            "number_pending_application": 2,  
            "number_running_application": 10,  
            "number_pending_request": 2,  
            "number_running_container": 10,  
            "number_reserved_container": 0,  
            "resource_inuse" {  
                "vcores": 10,  
                "memory": 10240  
            },  
            "resource_request" {  
                "vcores": 2,  
                "memory": 2048  
            }  
        }  
    ]  
}
```

```
        },
        "resource_reserved" {
            "vcores": 1
            "memory": 1024
        }
    },
]
}
```

**Table 2-193** Parameters of all queues

Attribute	Type	Description
queuelist	array	Array of queue names.
name	String	Queue name.
open_state	String	This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request.
eopen_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request.
active_state	String	This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. A "INACTIVE" queue does not schedule any application.
eactive_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "INACTIVE" queue does not schedule any application.
number_pending_application	int	Total number of pending applications.
number_running_application	int	Total number of running applications.
number_pending_request	int	Total number of pending request.
number_running_container	int	Total number of running containers.
numbert_reserved_container	int	Total number of reserved containers.
resource_request	object	Pending resource requests within queue in a form of vcores, memory, and so on.

Attribute	Type	Description
resource_inuse	object	In use resource within queue in a form of vcores, memory, and so on.
resource_reserved	object	Reserved resource within queue in form of vcores, memory, and so on.
active_state	String	Indicate either "ACTIVE" or "INACTIVE" state of the queue. A "INACTIVE" queue does not schedule any allocation.

- Query \*single\* queues within scheduler engine, including leaf and all middle queues.

- URL

```
GET http://<SS_REST_SERVER>/ws/v1/sscheduler/queues/
{queueName}
```

- Input

None.

- Output

JSON Response:

```
{
  "queue": {
    "name": "root.default",
    "eopen_state": "CLOSED",
    "eactive_state": "INACTIVE",
    "open_state": "CLOSED",
    "active_state": "INACTIVE",
    "leaf_queue": yes,
    "number_pending_application": 100,
    "number_running_application": 10,
    "number_pending_request": 10,
    "number_running_container": 10,
    "number_reserved_container": 1,
    "resource_inuse": {
      "vcores": 10,
      "memory": 10240
    },
    "resource_request": {
      "vcores": 2,
      "memory": 2048
    },
    "resource_reserved": {
      "vcores": 1,
      "memory": 1024
    }
  }

  "configuration": {
    "description": "Production spark queue",
    "max_pending_application": 10000,
    "max_running_application": 1000,
    "allocation_order_policy": "FIFO",
    "default_resource_select": "label1",
    "max_master_share": 10%,
    "max_running_application_per_user": -1,
    "max_allocation_unit": {
      "vcores": 32,
      "memory": 128000
    },
  }
}
```

```
"user_acl": [
  {
    "user": "user1"
  },
  {
    "group": "group1"
  }
],
"admin_acl": [
  {
    "user": "user2"
  },
  {
    "group": "group2"
  }
]
```

- Exceptions  
Queue not found.

**Table 2-194** Parameters of single queues

Attribute	Type	Description
queue	object	A queue object.
name	String	Queue name.
description	String	Purpose of queue.
open_state	String	This is inner state (self state) of queue. Indicate either "OPEN" or "CLOSED" effective state of the queue. A "CLOSED" queue does not accept any new allocation request.
eopen_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "CLOSED" queue does not accept any new allocation request.
active_state	String	This is inner state (self state) of queue. Indicate either "ACTIVE" or "INACTIVE" state of the queue. A "INACTIVE" queue does not schedule any application.
eactive_state	String	This is outer state of queue. An effective state is combination of queue own state and its ancestors. A "INACTIVE" queue does not schedule any application.
leaf_queue	boolean	Indicate whether queue is leaf or middle. Yes means leaf queue.
number_pending_application	int	Number of pending application currently. In case of middle or parent queue, this is the aggregated number of all children queue.

Attribute	Type	Description
number_running_application	int	Number of running application currently. In case of middle or parent queue, this is the aggregated number of all children queue.
number_pending_request	int	Number of pending demand; sum of count from each outstanding demand. In case of middle/parent queue, this is the aggregated number of all children queues.
number_running_container	int	Number of running containers. In case of middle or parent queue, this is the aggregated number of all children queues.
number_reserved_container	int	Number of reserved containers. In case of middle or parent queue, this is the aggregated number of children queues.
resource_request	object	Pending resource requests within queue in a form of vcores, memory etc.
resource_inuse	object	In use resource within queue in a form of vcores and memory etc.
resource_reserved	object	Reserved resources within queue in a form of vcores and memory etc.
configuration	object	A queue configuration object.
max_pending_application	int	Max number of pending application. In case of middle or parent queue, this is the aggregated number of all children queue.
max_running_application	int	Max number of running application. In case of middle/parent queue, this is the aggregated number of all children queue.
allocation_order_policy	String	Allocation policy, can be FIFO, PRIORITY, or FAIR.
max_running_application_per_user	int	Maximum number of running application per user.
max_master_share	string	Percentage of steady share of this queue.
max_allocation_unit	object	Maximum allowed resource per container in vcores and memory format.
default_resource_select	String	Default resource selection expression. It is used when an application does not specify one during its submission.
user_acl	array	Array of user, who have been given "user" rights on this queue.

Attribute	Type	Description
admin_acl	array	Array of user, who have been given "admin" rights on this queue.
group	String	User group name.
user	String	User name.

- Query Resource Pool
  - Query \*all\* resource pools within scheduler engine.
    - URL  
GET `http://<SS_REST_SERVER>/ws/v1/sscheduler/resourcepools/list`
    - Input  
None.
    - Output

```
JSON Response:
{
  "resourcepool_list": [
    {
      "name": "pool1",
      "description": "resource pool for crc",
      "number_member": 5,
      "members": [
        {
          "resource": "node1"
        },
        {
          "resource": "node2"
        },
        {
          "resource": "node3"
        },
        {
          "resource": "node4"
        },
        {
          "resource": "node5"
        }
      ],
      "available_resource": {
        "vcores": 60,
        "memory": 60000
      },
      "total_resource": {
        "vcores": 100,
        "memory": 128000
      },
      "configuration": {
        "resources": [
          {
            "resource": "node1"
          },
          {
            "resource": "node[2-5]"
          }
        ],
        "resource_select": "label1"
      }
    }
  ]
}
```

```
"name": "pool2",
"description": "resource pool for erc",
"number_member": 4
"members": [
{
"resource": "node6"
},
{
"resource": "node7"
},
{
"resource": "node8"
},
{
"resource": "node9"
}
],
"available_resource": {
"vcores": 56,
"memory": 48000
},
"total_resource": {
"vcores": 100,
"memory": 128000
},
"configuration": {
"resources": [
{
"resource": "node6"
},
{
"resource": "node[7-9]"
}
],
"resource_select": "label1"
},
{
"name": "default",
"description": "system-generated",
"number_member": 1,
"members": [
{
"resource": "node0"
}
],
"available_resource": {
"vcores": 8,
"memory": 8192
},
"total_resource": {
"vcores": 16,
"memory": 12800
}
}
]
```

**Table 2-195** Parameters of all resource pool

Attribute	Type	Description
resourcepool_list	array	Array of resource pool objects.
name	String	Resource pool name.

Attribute	Type	Description
number_member	int	Number of members in resource pool.
description	String	Description of the resource pool.
members	array	Arrays of resource name, which are current members of the resource pool.
resource	String	Resource name.
available_resource	object	Current available resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, available via this resource pool now.
total_resource	object	total resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, total via this resource pool now.
configuration	object	Configuration object.
resources	array	Array of resource name pattern configured.
resource	String	Resource name pattern.
resource_select	String	Resource select expression.

- Query \*single\* resource pools within scheduler engine.
  - URL  
GET http://<SS\_REST\_SERVER>/ws/v1/sscheduler/resourcepools/{resourcepoolname}
  - Input  
None.
  - Output

JSON Response:

```
{
  "resourcepool": {
    "name": "pool1",
    "description": "resource pool for crc",
    "number_member": 5
    "members": [
      {
        "resource": "node1"
      },
      {
        "resource": "node2"
      },
      {
        "resource": "node3"
      },
      {
        "resource": "node4"
      }
    ]
  }
}
```

```
        "resource": "node5"
    },
],
"available_resource": {
    "vcores": 60,
    "memory": 60000
},
"total_resource": {
    "vcores": 100,
    "memory": 128000
},
"configuration": {
    "resources": [
        {
            "resource": "node6"
        },
        {
            "resource": "node[7-9]"
        }
    ],
    "resource_select": "label1"
}
}
```

- Exceptions

Resource pool not found.

**Table 2-196** Parameters of single resource pool

Attribute	Type	Description
resourcepool	object	Resource pool object.
name	String	Resource pool name.
description	String	Description of the resource pool.
number_member	int	Number of members in resource pool.
members	array	Arrays of resource name, which are current members of the resource pool.
resource	String	Resource name.
available_resource	object	Current available resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, available via this resource pool now.
total_resource	object	total resource in this pool.
vcores, memory, ..	int	Numeric consumable resource attributes, total via this resource pool now.
configuration	object	Configuration object.
resources	array	Array of resource name pattern configured.
resource	String	Resource name pattern.

Attribute	Type	Description
resource_select	String	Resource select expression.

- Query **policiesxmlconf**

- URL  
GET http://<SS\_REST\_SERVER>/ws/v1/sscheduler/policiesxmlconf/list
- Input  
None.
- Output

```
<policies>
  <policlist>
    <resourcepool>default</resourcepool>
    <queues>
      <name>default</name>
      <fullname>root.default</fullname>
      <share>20.0</share>
      <reserve>memory 0,vcores 0 : 0.0%</reserve>
      <minimum>memory 0,vcores 0 : 20.0%</minimum>
      <maximum>memory 0,vcores 0 : 100.0%</maximum>
      <defaultuser>
        <maximum>100.0%</maximum>
        <weight>1.0</weight>
      </defaultuser>
    </queues>
  </policlist>
</policies>
```

# 3 Development Specifications

## 3.1 Development Environment Construction

### 3.1.1 Rules

**The development environment should be constructed strictly following the development guide. The required components should be downloaded from the client instead of open-source websites.**

Based on Hadoop open-source components, MRS enhances its functionality and security. Huawei does not update the versions of these components because component dependencies are complicated. Components sharing the same name with those downloaded from the open-source website cannot run on MRS.

**The time difference between the client and the cluster must be confirmed.**

Ensure that the time difference between the client and the MRS cluster is less than 5 minutes.

**The IBM JDK environment must be prepared.**

If the application is developed using IBM JDK and the keytab file of the cluster needs to be used for authentication, check whether the IBM JDK version meets the requirements. Otherwise, exceptions may occur.

## 3.2 Security Authentication

### 3.2.1 Rules

**Only one account is used in each process and a process needs explicit authentication only once.**

This rule must be met in the following scenarios (pay attention to this in the design phase):

1. A process can access multiple clusters at the same time (Each cluster has independent KrbServer and LdapServer.)
2. All applications running in a container (for example, Tomcat) belong to the same process.

**Cross-Manager trust must be configured for cross-Manager access scenarios so that applications need to be authenticated only once when accessing Managers.**

For how to configure cross-Manager trust, refer to the "Managing Mutual Trust Relationships Between Managers" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

## 3.2.2 Suggestions

### Account Management Principles

1. Service application should apply for new accounts instead of using original system accounts.
2. The new accounts should meet the principle of least privilege.

## 3.3 ClickHouse

### 3.3.1 Rules

#### Ensure That the Time on the Client Is the Same as That on the Server If the Cluster Is Installed in the Security Mode

If the cluster is of the security edition and Kerberos authentication is required, the time on the server must be the same as that on the client. Pay attention to the time difference conversion between time zones. If the time is inconsistent, the client authentication fails and subsequent service processes cannot be executed.

#### ClickHouse Uses Its Own ZooKeeper Service

ClickHouse relies heavily on ZooKeeper and does many read and write operations on it. To avoid affecting other services, each ClickHouse service should use its own ZooKeeper service.

#### Use partition fields and index fields of data tables properly

The MergeTree engine organizes and stores data in partition directories. During data query, partitions can be used to effectively skip useless data files and reduce data reading.

The MergeTree engine sorts data based on the index field and generates sparse indexes based on the **index\_granularity** configuration. Data can be quickly filtered based on index fields, reducing data reading and improving query performance.

## Insert a large volume of data at a low frequency

Each time data is inserted in ClickHouse, one or more part files are generated. If there are too many data parts, the pressure on merging increases and an exception may occur, affecting data insertion. You are advised to insert 100,000 rows at a time and ensure the frequency is no more than once per second.

## Do not use the character type to store data of the time, date, or numeric type

Especially when the time, date, or numeric field needs to be calculated or compared.

## The number of records in a single table (distributed table) cannot exceed trillions, and the number of records in a single table (local table) cannot exceed ten billions

The performance of querying trillions of tables is poor, and the cluster maintenance is difficult.

## Data lifecycle management must be considered during table design

The disk space is limited, and data lifecycle management needs to be considered. The MergeTree engine supports column fields and table-level TTL when creating tables. When the values in a column field expire, ClickHouse replaces them with the default values of the data type. If all values of a column in a partition have expired, ClickHouse deletes the column files in the partition directory from the file system. When the data in a table expires, the ClickHouse deletes all the corresponding rows.

## The external component ensures the idempotence of imported data

ClickHouse does not support transactions for data write. Use the external import module to control data idempotence. For example, if data of a batch fails to be imported, drop the corresponding partition data. After the fault is rectified, import the partition data again.

## When a local ClickHouse table is created, the partition by keyword must be carried. Otherwise, the table cannot be migrated on the ClickHouse data migration page of Manager

The ClickHouse data migration page depends on the partition field of the table during table data migration. If partition by is not used to create partitions when the table is created, the table cannot be migrated on the ClickHouse data migration page of Manager.

## Place a small table on the right for join query

When two tables are joined, the data in the right table is loaded to the memory, and then the data in the left table is traversed based on the data in the right table for matching. Placing the small table on the right reduces the number of match queries. According to the usage, the performance of joining a large table to a

small table is improved by several orders of magnitude compared with that of joining a small table to a large table.

### 3.3.2 Suggestions

#### Properly configure the maximum number of concurrent operations

ClickHouse has a high processing speed because it uses the parallel processing mechanism. Even if a query is performed, half of the CPU of the server is used by default. Therefore, the ClickHouse does not support high-concurrency query scenarios. The default maximum number of concurrent requests is 100. You can adjust this number as needed, but it should be no more than 200.

#### Deploy the load balancing component. The query is performed based on the load balancing component to prevent the performance from being affected due to heavy single-point query pressure

ClickHouse can connect to any node in the cluster for query. If the query is performed on one node, the node may be overloaded and the reliability is low. You are advised to use ClickHouseBalancer or other load balancing services to balance the query load and improve reliability.

#### Properly set the partition key, ensure that the number of partitions is less than 1000, and use the integer type for the partition field

1. You are advised to use `toYYYYMMDD` (table field `pt_d`) as the partition key. The table field `pt_d` is of the date type.
2. If hourly partitioning is required in the service scenario, use `toYYYYMMDD` (table field `pt_d`) and `toYYYYMMDD` (table field `pt_h`) as the joint partitioning key. `toYYYYMMDD` (table field `pt_h`) is an integer number of hours.
3. If data needs to be stored for many years, you are advised to create partitions by month, for example, `toYYYYMM` (table field `pt_d`).
4. Properly control the number of parts based on factors such as the data partition granularity, volume of data submitted in each batch, and data storage period.

#### During query, the most frequently used and most filtered fields are used as the primary keys. The fields are sorted in descending order of access frequency and dimension cardinality

Data is sorted and stored based on primary keys. When querying data, you can quickly filter data based on primary keys. Setting primary keys properly during table creation can greatly reduce the amount of data to be read and improve query performance. For example, if the service ID needs to be specified for all analysis, the service ID field can be used as the first field of the primary key.

## Properly set the sparse index granularity based on service scenarios

The primary key index of ClickHouse is stored by using a sparse index. The default sampling granularity of the sparse index is 8192 rows, that is, one record is selected from every 8192 rows in the index file.

Suggestions:

1. The smaller the index granularity is, the more effective the query in a small range is. This avoids the waste of query resources.
2. The larger the index granularity is, the smaller the index file is, and the faster the index file is processed.
3. If the table index granularity exceeds 1 billion, set this parameter to **16384**. Otherwise, set this parameter to **8192** or a smaller value.

## Local Table Creation Reference

Reference:

```
CREATE TABLE mybase_local.mytable
(
    `did` Int32,
    `app_id` Int32,
    `region` Int32,
    `pt_d` Date
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/mybase_local/mytable', '{replica}')
PARTITION BY toYYYYMMDD(pt_d)
ORDER BY (app_id, region)
SETTINGS index_granularity = 8192, use_minimalistic_part_header_in_zookeeper = 1;
```

Instructions:

1. Select a table engine:

**ReplicatedMergeTree**: MergeTree engine that supports the replica feature. It is the most commonly used engine.

2. Table information registration path on ZooKeeper, which is used to distinguish different configurations in the cluster:

**/clickhouse/tables/{shard}/{databaseName}/{tableName}: {shard}** indicates the shard name, **{databaseName}** indicates the database name, and **{tableName}** indicates the replicated table name.

3. **order by** primary key field:

The most frequently used and most filterable field is used as the primary key. The dimensions are sorted in ascending order of access frequency and dimension cardinality. It is recommended that the number of sorting fields be less than or equal to 4. Otherwise, the merge pressure is high. The sorting field cannot be null. If the sorting field is null, data conversion is required.

4. **partition by** field

The partition key cannot be null. If the field contains a null value, data conversion is required.

5. Table-level parameter configuration:

**index\_granularity**: sparse index granularity. The default value is **8192**.

**use\_minimalistic\_part\_header\_in\_zookeeper**: whether to enable the optimized storage mode of the new version for data storage in the ZooKeeper.

6. For details about how to create a table, visit <https://clickhouse.tech/docs/en/engines/table-engines/mergetree-family/mergetree/>.

## Distributed Table Creation Reference

Reference:

```
CREATE TABLE mybase.mytable AS mybase_local.mytable  
ENGINE = Distributed(cluster_3shards_2replicas, mybase_local, mytable, rand());
```

Instructions:

1. Name of the distributed table: **mybase.mytable**.
2. Name of the local table: **mybase\_local.mytable**.
3. Use **AS** to associate the distributed table with the local table to ensure that the field definitions of the distributed table are the same as those of the local table.
4. Parameter description of the distributed table engine:  
**cluster\_3shards\_2replicas**: name of a logical cluster.  
**mybase\_local**: name of the database where the local table is located.  
**mytable**: local table name.  
**rand()**: (optional) sharding key, which can be the raw data (such as did) of a column in the table or the result of a function call, such as rand(). Note that data must be evenly distributed in this key. Another common operation is to use the hash value of a column with a large difference, for example, **intHash64(user\_id)**.

## Select the minimum type that meets the requirements based on the fields in the service scenario table

Numerical type, such as UInt8/UInt16/UInt32/UInt64, Int8/Int16/Int32/Int64, Float32/Float64. The performance varies according to the length.

## Perform data analysis based on large and wide tables. Do not join large tables. Convert distributed join queries into join queries of local tables to improve performance

The performance of ClickHouse distributed join is poor. You are advised to aggregate data into a wide table on the model side and then import the table to ClickHouse. Queries in distributed join mode are converted to join queries on local tables. This eliminates the transmission of a large volume of data between nodes and reduces the volume of data involved in the calculation of local tables. The service layer summarizes data based on the local join results of all shards. The performance is improved remarkably.

## Properly set the part size

The **min\_bytes\_to\_rebalance\_partition\_over\_jbod** parameter indicates the minimum size of the part involved in automatic balancing and distribution among disks in a JBOD array. The value must be appropriately set.

If the value is smaller than **max\_bytes\_to\_merge\_at\_max\_space\_in\_pool/1024**, the ClickHouse server process fails to be started and unnecessary parts move between disks.

If the value of **min\_bytes\_to\_rebalance\_partition\_over\_jbod** is greater than that of **max\_data\_part\_size\_bytes** (maximum size of parts that can be stored on disks in one array), no part can meet the condition for automatic balancing.

## 3.4 Doris

### 3.4.1 Table Creation Rules

This topic describes the rules and suggestions for creating Doris tables.

#### Doris Table Creation Rules

- When creating a Doris table and specifying buckets, make sure that each bucket contains data ranging from 100 MB to 3 GB. Additionally, ensure that the maximum number of buckets in a single partition does not exceed 5,000.
- You must set a bucketing policy for tables that have over 500 million data records.
- Do not set too many bucketing columns in a table. Generally, one or two columns are enough. In addition, you need to ensure even data distribution and balanced query throughput.
  - Data should be evenly distributed to prevent data skew in some buckets, which affects data balancing and query efficiency.
  - The query throughput is reduced with bucketing and tailoring of query SQL statements to avoid full bucket scanning and improve query performance.
  - Preferentially use columns with evenly distributed data and those that are commonly used as query conditions as bucketing columns.

You can use the following methods to analyze whether data skew occurs:

**SELECT a, b, COUNT(\*) FROM tab GROUP BY a,b;**

Once the command is executed, verify if the variation in the number of data records among the groups is minimal. If the difference surpasses 2/3 or 1/2, select another bucket field.

- Do not use dynamic partitioning for less than 20 million data records. Dynamic partitioning generates partitions automatically, but users may overlook small tables. Consequently, numerous useless buckets are created in partitions.
- When creating a table, make sure to have three to five sort keys. Having too many sort keys can impede data writing and importing.
- If Auto Bucket is not used, bucketing should be determined by the data volume to enhance the performance of data import and query. Auto Bucket causes superfluous tablets and a large number of small files.
- Set at least 2 replicas when you create a table. The default replication factor is 3. Do not use a single backup.

## Doris Table Creation Suggestions

- Use no more than six materialized views in a single table. Do not nest materialized views or them in ETL tasks such as heavy aggregations and joins during data writing.
- If there are many historical partitions for a little historical data and the data is unbalanced or the data query probability is low, you can create historical partitions on a yearly/monthly basis and store all historical data in the corresponding partitions.  
To create history partition, use **FROM ("2000-01-01") TO ("i") INTERVAL 1 YEAR**.
- If the data volume is less than 10 million to 200 million, you do not need to set partitions (the Doris has a default partition). Instead, you can directly use the bucket policy.
- If more than 30% data skew occurs in the bucketing fields, do not use the hash bucketing policy. Instead, use the random bucketing policy. The related commands are as follows:  
**Create table ... DISTRIBUTED BY RANDOM BUCKETS 10 ...**
- The first field must be the most frequently queried one in the table you created. By default, you can quickly query data with prefix indexes. Select the column that is most frequently queried and has a high cardinality as the prefix index. The first 36 bytes of a row are used as the prefix index of the row by default (for varchar columns, the first 20 bytes are matched as the prefix index, and excessive bytes are truncated).
- To fuzzy match or use equivalent/in conditions in a query of more than 100 million data records, use inverted indexes (supported since Doris 2.x) or Bloomfilter. For orthogonal queries with low cardinality columns, use bitmap indexes. (The recommended cardinality of bitmap indexes ranges from 10000 to 100000.)
- Plan the number of fields to be used when creating a table. You can reserve dozens of integer or character fields. If fields are insufficient, you need to add fields temporarily at a high cost.

## 3.4.2 Data Change

This topic describes the rules and suggestions for changing Doris data.

### Doris Data Change Rules

- Do not directly use the **delete** or **update** statement to change data. Instead, use the **upsert** of the CDC.
- Avoid frequently adding or deleting fields in tables during peak hours. Reserve fields for future use when you create tables. If fields must be added or deleted, or field types and comments must be modified, stop writing and modifying tasks on the target table during off-peak hours and then re-create a table.
  - a. Create a table. The structure of the table is the same as that of the table you want to modify. Add new fields to the new table, delete unnecessary fields, or change field types.
  - b. Specify fields and insert them to the newly created table.

**INSERT INTO** Newly created table **SELECT** Specified fields **FROM** Existing table whose columns need to be modified;

 **NOTE**

To prevent high CPU or memory usage and minimize the impact on query service, you can import data to a new table in batches based on time if the table has a significant amount of data. The command is as follows:

`insert into tab1 select col from tab where date <= xx;`

- c. Exchange the names of the two tables. For more information, see [Exchange Tables](#).

**ALTER TABLE** [db.]tbl1 **REPLACE WITH TABLE** tbl2 [**PROPERTIES('swap' = 'true')**];

- Some queries may take a long time and consume a lot of memory and CPU resources. You can set the query timeout parameter **query\_timeout** that works on SQL statements or for a user.

## Doris Data Change Suggestions

When you run large SQL statements, set session variables with **hint** by using a method similar to **SELECT /\*+ SET\_VAR(query\_timeout = xxx\*)/ from table**. Do not set global system variables.

### 3.4.3 Naming Conventions

This topic describes the rules and suggestions for naming databases and tables.

#### Doris Naming Rules

The database character set must be UTF-8 and only UTF-8 is supported.

#### Doris Naming Suggestions

- Set database names in lowercase and use underscores (\_) to link words. A name must be less than 62 bytes.
- Table names of Doris are case sensitive. Set a name in lowercase and use underscores (\_) to link words. A name must be less than 64 bytes.

### 3.4.4 Data Query

This topic describes the rules and suggestions for querying Doris data.

#### Doris Data Query Rules

- When you are using the data query code, retry the query and issue the query again if the query fails.
- If the enumerated value of the constant **in** exceeds 1000, you must use a subquery.
- Do not use the Statement Execution Action REST APIs to execute a large number of SQL queries. These interfaces are used only for cluster maintenance.
- When dealing with query results exceeding 50,000 records, consider using either JDBC Catalog or the OUTFILE method to export the data. Otherwise,

allowing a large amount of data to accumulate on the front end (FE) can impact cluster stability.

- When performing interactive queries, export data using pagination with an offset limit. You can achieve this by using the ORDER BY command.
- If data is exported for a third party, use **outfile** or **export**.
- Utilize Colocation Join for joining more than two tables with over 300 million records.
- Avoid using **select \*** for querying large tables with hundreds of millions of records and specify the required fields instead.
  - Do not use **select \*** when you use SQL Block.
  - For high-concurrency point queries, enable row-based storage (supported by Doris 2.x) and use PreparedStatement.
- Bucketing conditions must be set for queries of tables of hundreds of millions records.
- Do not perform full-partition scan on partitioned tables.

## Doris Data Query Suggestions

- When an **INSERT INTO SELECT** statement inserts over 100 million data records, divide them into smaller batches for execution.
- Do not use OR as a JOIN condition.
- Do not frequently delete and modify data. Instead, delete data in batches occasionally with conditions to improve system stability and deletion efficiency.
- To return som data after sorting a large amount of data (more than 500 million records), reduce the data range for sorting. Sorting a large amount of data affects query performance. The following is an example:

Instead of using **from table order by datatime desc limit 10**, use **from table where datatime='2023-10-20' order by datatime desc limit 10**.
- Pay attention to the following points when using **parallel\_fragment\_exec\_instance\_num** to optimize query task performance:

This parameter determines the maximum number of fragments that can run simultaneously at the session level. Too many concurrent fragments will use up a significant amount of CPU resources. You can leave this parameter blank. If you need to set this parameter to accelerate query speed, comply with the following rules:

  - Do not set this parameter to take effect globally, that is, do not use the **set global** command to set this parameter.
  - Set this parameter to an even number (2 or 4). The maximum value cannot exceed half of the number of CPU cores on a single node.
  - Check the CPU usage before you set the parameter. You can set this parameter only when the CPU usage is less than 50%.
  - If you use **insert into select** to insert a large amount of data, do not set this parameter.

## 3.4.5 Data Import

This topic describes the technical suggestions for importing Doris data.

## Doris Data Import Suggestions

- Do not frequently perform the **update**, **delete**, or **truncate** operation. Perform an operation every several minutes. To use the **delete** operation, you must set the partitioning condition or primary key column.
- Avoid using **INSERT INTO tb1 VALUES("1"),("a")**; to frequently import small amounts of data. Instead, opt for StreamLoad, BrokerLoad, SparkLoad, or Flink Connector.
- When Flink writes data to Doris in real time, set the checkpoint based on the data volume of each batch. If the data volume of each batch is too small, a large number of small files will be generated. The recommended value is 60s.
- Do not use **insert values** as the main data write mode. StreamLoad, BrokerLoad, or SparkLoad is recommended for batch data import.
- If there are downstream dependencies or queries when you use **INSERT INTO WITH LABEL XXX SELECT** to import data, check whether the imported data is visible.

Run the **show load where label='xxx'** SQL command to check whether the current INSERT task is **VISIBLE**. The imported data is visible only when the status is **VISIBLE**.

- Streamload is suitable for importing data of less than 10 GB, and Brokerload is suitable for data of less than 100 GB. For large-scale data, use SparkLoad.
- Do not use Routine Load of Doris to import data. Instead, use Flink to query Kafka data and then write the data to Doris. This limits the amount of data to be imported in a single batch and avoids a large number of small files. If Routine Load has already been used to import data, set **max\_tolerable\_backend\_down\_num** to 1 on the FE before you change the import method to improve reliability.
- Import data in batches at a low frequency. The average interval for importing a single table must be greater than 30s. Import 1000 to 100000 rows of data each time at a recommended interval of 60s.

## 3.4.6 UDF Development

This topic describes the rules and suggestions for developing Doris UDF programs.

### Doris UDF Development Rules

- The UDF invocation must be thread-safe.
- Do not load external large files to the memory when implementing a UDF. Otherwise, the memory could be used up.
- Avoid a large number of recursive calls. Otherwise, stack overflow or OOM may occur.
- Do not create objects or arrays continuously. Otherwise, the memory could be used up.
- Use a Java UDF to capture and process possible exceptions. Do not send exceptions to services. Use the try-catch block to handle exceptions and record exception information if necessary.
- In the UDF, do not define static collection classes for storing temporary data or query large objects in external data. Otherwise, the memory usage is high.

- Ensure that the imported packages in the class do not conflict with the packages on the server. You can run the `grep -lr "Fully-qualified class name"` command to check JAR package conflicts. Use fully-qualified class names to avoid such conflicts.

## Doris UDF Development Suggestions

- To prevent stack memory overflow, do not copy a large amount of data.
- Do not concatenate a large number of strings. Otherwise, the memory usage is high.
- Java UDFs should use meaningful names so that other developers can easily understand their purpose. Use camel-case names and end a name with a UDF, for example, `MyFunctionUDF`.
- A Java UDF should specify the data type of the return value and must have a return value. Do not set the return value to `NULL` when it should be the default value or when an exception occurs. Use basic data types or Java classes as return value types.

## 3.4.7 Connection and Running

This topic describes the specifications you need comply with when you connect to Doris and run Doris tasks.

- Connect to the DBalancer instance and forward the connection request to the Doris FE service. This facilitates connection load balancing between FE nodes and prevents services from being affected when the connected FE is faulty.
- If a hardware fault or a single Doris instance failure occurs, Doris can continue executing running tasks, but newly submitted tasks cannot run. You should enable retry upon failures so that submitted jobs can still run in case of unknown exceptions.

## 3.5 Elasticsearch

### 3.5.1 Application Scenarios

- Types of the data to be searched are as follows: structured data (RDS), semi-structured data (web pages and XML files), and unstructured data. (logs, pictures, and images). Elasticsearch can perform a series of operations such as cleaning, word segmentation, and establishment of inverted indexes for the preceding data types, and then provide the full-text search capability.
- The search criteria are diversified (for example, too many fields are involved). The common query cannot meet the following requirements: Query simple words and phrases, or multiple forms of words or phrases in the full text.
- Read data is much more than written data.

## 3.5.2 Rules

### If the cluster is installed in the security mode, ensure that the time on the client is the same as that on the server

If the cluster is of the security edition and Kerberos authentication is required, the time on the server must be the same as that on the client. Pay attention to the time difference conversion between time zones. If the time is inconsistent, the client authentication fails and subsequent service processes cannot be executed.

### When a self-built user performs index data operations, authentication information needs to be configured and the corresponding read and write permissions must be assigned to the user

The created user must have the read and write permissions on the index to be operated. Select the administrator role for the user or grant the read and write permissions of the corresponding index to the user.

### Invoke the client closing function before the application ends

When the application ends, invoke the close() function of the client. Do not frequently create and close clients during task running.

### Set a proper number of data records when using batch requests

When the bulk command is used to index data in batches, the size of each batch of data to be submitted should be 5 to 15 MB.

For example, if the size of each data record is 1 KB, you are advised to submit 5000 data records in a batch.

Start the test from 5 MB and increase the size until the write performance cannot be improved.

### Each batch request processes only data of one index

For one bulk request, only data of one index is written. You are not advised to write data of multiple indexes in one bulk request. Data of different indexes is submitted in different bulk requests.

### Concurrently index data in multiple threads

The application program uses multiple threads to concurrently index data. It is recommended that the number of concurrent threads be one to two times of the number of CPU cores.

For example, for a 32-core host, set 30 to 60 threads.

### Run multiple clients to execute tasks concurrently

Run multiple clients to concurrently execute index and query tasks to improve performance.

## Configure the IP address and port list of multiple EsNode instances on the client

Configure IP address and port lists of multiple EsNode instances to enable load sharing and balancing, avoiding the performance bottleneck of a single EsNode instance.

It is recommended that the IP addresses and port numbers of all EsNode instances be configured in the client IP address list.

## Do not configure the IP address and port number of the EsMaster node on the client

The EsMaster instance is an important management process. To ensure the Elasticsearch cluster stability, you are not allowed to configure the IP address and port number of the EsMaster node in the IP address list of the client.

## Allocate sufficient heap memory to EsMaster and EsNode instances

On FusionInsight Manager, choose **Cluster > Name of the desired cluster > Services > Elasticsearch > Configurations > All Configurations**, search for **GC\_OPTS**, and change the values of **-Xms** and **-Xmx** in **GC\_OPTS**.

- Set **-Xmx** to a value less than or equal to 50% of the physical memory to ensure that sufficient physical memory is reserved for the operating system cache. If 50% of the machine memory is greater than the number of instances multiplied by 30 GB, set it to 30 GB. In other cases, set it to 50% of the machine memory divided by the number of instances.
- Do not set **-Xmx** to be greater than the value of **cutoff** (compressed oops) that is used by the JVM to compress the object pointer. The **cutoff** value may be different but is close to 31 GB. Therefore, you can set **-Xmx** to up to 30 GB.

### 3.5.3 Suggestions

#### Creating Elasticsearch Indexes

- To reduce the number of indexes and avoid huge mappings, store data with the same index structure in the same index.
- Do not place irrelevant data in the same index to avoid sparsity.



These suggestions are not recommended when you use the parent/child relationship between documents because this function is supported only by documents in the same index.

#### Properly plan the number of shards for indexes

Each Shard can process index and query requests. When setting the number of Shards, consider the following two aspects:

- It is recommended that the amount of data stored in a single shard be about 10 GB to 20 GB, and the maximum size be 30 GB.

- Determine the number of primary Shards according to the maximum data capacity of the index and the capacity of a single Shard.
- To improve data reliability, set the number of replica Shards properly. Set it to at least 1. If the storage space of the cluster is sufficient, you are advised to set it to 2.

 NOTE

Once the number of primary Shards is determined, it cannot be changed. The number of replica Shards can be modified as required.

## Do not return a large result set

Elasticsearch is designed as a search engine that makes it very good at acquiring the best document that matches the query. It is not suitable for retrieving all documents that match a particular query. To avoid deep paging, you're not allowed to query data records after 100,000 by page (from & size). In this case, use the Scroll API.

## Install the EsClient instance

For an Elasticsearch cluster with large read and write workloads, it is recommended that 3 to 10 EsClient instances be installed.

The EsClient instance is used to receive write requests, query requests and aggregate query results. It improves cluster performance and stability.

## 3.6 Flink

### 3.6.1 Applicable Scenarios

Flink is a unified computing framework that supports both batch processing and stream processing. It provides a stream data processing engine that supports data distribution and parallel computing. Flink features stream processing and is a top open-source stream processing engine in the industry.

Flink provides high-concurrency pipeline data processing, millisecond-level latency, and high reliability, making it suitable for low-latency data processing.

### 3.6.2 Rules

#### Delete Residual Directories If a Flink Task Stops Unexpectedly

After a FlinkServer instance is installed, the residual Flink directories are automatically deleted.

By default, only residual directories in the `/flink_base` directory of ZooKeeper and those in the `/flink/recovery` directory of HDFS are deleted.

## 3.6.3 Suggestions

### FlinkServer Usage

You are advised to submit Flink jobs using FlinkServer. FlinkServer supports the submission of Flink SQL jobs and Flink Jar jobs.

## 3.7 GraphBase

### 3.7.1 Rules

#### The token validity period in REST API authentication is 20 minutes by default, and can be customized

The token validity period in REST API authentication is 20 minutes. If the token is not used to interact with the server within 20 minutes, the token becomes invalid. To ensure the validity of the token, the invoker must interact with the server within 20 minutes using the token or obtain a new token after the token expires.

The token period can be customized in the GraphBase configuration interface to search for "graphserver.session.timeout" to set custom values.

### Proper Index Design

A proper index mechanism has a great impact on data query, especially in full graph query scenarios. The index design must be designed based on service query requirements. If no valid index is designed, all data in the entire graph may be forcibly scanned during data query, resulting in low query performance and even request timeout.

Typical application scenarios of indexes are as follows:

- Query an entity with a same property value (string or numeric type) as the specified one.  
For a large graph, if the service requires a query for entities that have the same property value with a specified property value, create indexes for the property (composite or mixed index) first. Otherwise, you need to perform a full graph query for the entities with the same property value as specified by the service in GraphBase, and perform full graph scan, which may cause query timeout.
- Query an entity by comparing property values (numeric type) with a specified one.  
In the case of a large graph, if the service requires the query of entities by comparing property values with a specified one, you need to create mixed indexes for the property first. You can use this property index to efficiently lock the database entity objects that meet the query conditions.
- Full-text Search by Condition
- Currently, only mixed indexes support full-text search in GraphBase. For queries similar to the following one, create mixed indexes:

Perform full graph query to query the entity with a particular area or qualifier (such as Xi'an or BEIDAJIE) contained in the value of the address property.

## Supplementary Indexes

If no index is created for a specified property at first but new indexes need to be created for the property after the property data is saved, the newly added indexes are supplementary indexes. This process involves recreating indexes.

After an index is created, index data is not automatically updated for the data imported previously. In this case, the user needs to invoke the interface for recreating an index to initiate an index data update.

## Using the API for Deleting a Graph Cautiously

The interface for deleting a graph not only deletes the definition of the graph, but deletes all the data in the graph. Before invoking this API, ensure that the data is no longer used.

## User Rights Description

Users who log in to GraphBase must have the permissions of the graphbaseadmin, graphbaseoperator, or graphbasedeveloper user group.

## Client Currency

The load sharing between multiple GraphServer in GraphBase is based on the HTTP session mode. Multiple requests of the same HTTP session are sent to the same GraphServer for processing. Therefore, in the high concurrency scenario, the client code should use multiple HTTP clients to share loading using multiple GraphServers and to increase the number of concurrent requests.

## has Usage Specifications in Gremlin Graph Query

1. Query the information about a vertex with a specified vertex label.  

```
gremlin> g.V().hasLabel('person')
```

This query requires full graph query for vertices. If the number of vertices meeting this search condition is large or the graph is large, the query times out.  
It is recommended that the output be limited.  

```
gremlin> g.V().hasLabel('person').limit(10)
```
2. Perform full graph query for vertices using specified property conditions.  

```
gremlin> g.V().has('key', 'value')
```

This query requires full graph query for vertices. If a large graph is accessed and no index is created for the property **key**, the query times out.  
To sum up, exercise caution when performing full graph query.

## 3.7.2 Recommendations

### Do Not Use REST APIs to Import a Large Amount of Data

If a large amount of data needs to be imported to GraphBase, use the batch import tool or real-time import tool based on site requirements. The REST API is

designed based on HTTP requests and is not suitable for importing a large amount of data.

## REST APIs for Periodical Data Import In Batches

If a large amount of data needs to be periodically imported to GraphBase in batches, you can periodically invoke the specified REST API to initiate a batch import task. The timer is provided by invoker.

# 3.8 HBase

## 3.8.1 Application Scenarios

Hadoop database (HBase) is a reliable, high-performance, column-oriented and scalable distributed storage system. Different from traditional relational databases, HBase is suitable for massive data process.

The HBase is applicable to the following scenarios:

- Massive data processing (higher than the TB or PB level).
- High-throughput demanding scenarios
- Scenarios that require efficient random read of massive data.
- Good-scalability demanding scenarios
- Concurrent processing of structured and unstructured data.
- Scenarios that do not require the Atomicity, Consistency, Isolation, Durability (ACID) feature provided by traditional relational databases.

HBase tables have the following features:

- Large: Each table contains a hundred million rows and one million columns.
- Column-oriented: Storage and rights control is implemented based on columns (families), and columns (families) are independently retrieved.
- Sparse: Null columns do not occupy storage space.

## 3.8.2 Rules

### Create a Configuration instance

Call the create() method of HBaseConfiguration to instantiate this class. Otherwise, the HBase configurations cannot be successfully loaded.

**Correct:**

```
//This part is declared in the class member variable declaration.  
private Configuration hbaseConfig = null;  
//Instantiate this class using its constructor function or initialization method.  
hbaseConfig = HBaseConfiguration.create();
```

**Incorrect:**

```
hbaseConfig = new Configuration();
```

## Share the Configuration instance

The HBase client codes obtain rights to interact with an HBase cluster by creating an HConnection with Zookeeper. Each HConnection has a Configuration instance. The created HConnection instances are cached. That is, if the HBase client needs to communicate with an HBase cluster, a Configuration instance is transferred to the cache. Then, the HBase client checks for an HConnection instance for the Configuration instance in the cache. If a match is found, the HConnection instance is returned. If no match is found, an HConnection instance will be created.

If the Configuration instance is frequently created, a lot of unnecessary HConnection instances will be created, causing the number of connections to Zookeeper to reach the upper limit.

Therefore, it is recommended that the client codes share the same **Configuration** instance.

## Create an Table instance

```
public abstract class TableOperationImpl {  
    private static Configuration conf = null;  
    private static Connection connection = null;  
    private static Table table = null;  
    private static TableName tableName = TableName.valueOf("sample_table");  
  
    public TableOperationImpl() {  
        init();  
    }  
    public void init() {  
        conf = ConfigurationSample.getConfiguration();  
        try {  
            connection = ConnectionFactory.createConnection(conf);  
            table = conn.getTable(tableName);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
    public void close() {  
        if (table != null) {  
            try {  
                table.close();  
            } catch (IOException e) {  
                System.out.println("Can not close table.");  
            } finally {  
                table = null;  
            }  
        }  
        if (connection != null) {  
            try {  
                connection.close();  
            } catch (IOException e) {  
                System.out.println("Can not close connection.");  
            } finally {  
                connection = null;  
            }  
        }  
    }  
    public void operate() {  
        init();  
        process();  
        close();  
    }  
}
```

## An Table instance cannot be used by multiple threads at the same time

Table is not thread safe for reads or write. If an Table instance is used by multiple threads at the same time, exceptions will occur.

## Cache a frequently used Table instance

Cache the Table instance that will be frequently used by a thread for a long period of time. A cached instance, however, will not be necessarily used by a thread permanently. In special circumstances, you need to rebuild an Table instance. See the next rule for details.

### Correct:



In this example, the Table instance is cached by Map. This method applies when multiple threads and Table instances are required. If an Table instance is used by only one thread and the thread has only one Table instance, Map need not be used.

```
//In this Map, TableName is the Key value. Cache all Table instances.  
private Map<String, Table> demoTables = new HashMap<String, Table>();  
//All Table instances share this Configuration instance.  
private Configuration demoConf = null;  
/**  
 * <Initialize an HTable class>  
 * <Detailed function description>  
 * @param tableName  
 * @return  
 * @throws IOException  
 * @see [class, class#method, class#member]  
 */  
private Table initNewTable(String tableName) throws IOException  
{  
    try (Connection conn = ConnectionFactory.createConnection(demoConf)){  
        return conn.getTable(tableName);  
    }  
}  
/**  
 * <Obtain Table instances>  
 * <Detailed function description>  
 * @see [class, class#method, class#member]  
 */  
private Table getTable(String tableName)  
{  
    if (demoTables.containsKey(tableName))  
    {  
        return demoTables.get(tableName);  
    } else {  
        Table table = null;  
        try {  
            table = initNewTable(tableName);  
            demoTables.put(tableName, table);  
        }  
        catch (IOException e)  
        {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
        return table;  
    }  
}
```

```
used
* because the Table is not thread safe. It is recommended that an Table instance be used by only one data
write thread at the same
*time.>
* @param dataList
* @param tableName
* @see [class, class#method, class#member]
*/
public void putData(List<Put> dataList, String tableName)
{Table table = getTable(tableName);
//Synchronization is not required if the Table instance is not shared by multiple threads.
//Note that Table is not thread safe.
synchronized (table)
{
try
{
table.put(dataList);
table.notifyAll();
}
catch (IOException e)
{
// When IOE is detected, the cached instance needs to be re-created.
try {
// Close the Connection.
table.close();
// Re-create the instance.
table = initNewTable(tableName);
} catch (IOException e1) {
// TODO
}
}
}
}
}
```

### Incorrect:

```
public void putDataIncorrect(List<Put> dataList, String tableName)
{Table table = null;
try
{
//Create an HTable instance each time when data is written.
table = initNewTable(tableName);
table.put(dataList);
}
catch (IOException e1)
{
// TODO Auto-generated catch block
e1.printStackTrace();
}
finally
{
table.close();
}
}
```

## Rebuild an Table instance

Rebuilt a cached Table when IOException is detected. See the example of the previous rule.

Do not call the following methods unless necessary:

- Configuration#clear

Do not call this method if a Configuration is used by an object or a thread.

The Configuration#clear method clears all attributes loaded. If this method is called for a Configuration used by Table, all the parameters of this

Configuration will be deleted from Table. As a result, an exception occurs when Table uses the Configuration the next time.

Therefore, avoid calling this method each time you rebuild an Table instance. Call this method when all the threads need to quit.

- **HConnectionManager#deleteAllConnections**

This method deletes all connections from the **Connection set**. As the Table stores the links to the connections, the connections being used cannot be stopped after the HConnectionManager#deleteAllConnections method is called, which eventually causes information leakage.

## Handle the data failed to write

Some data write operations may fail due to instant exceptions or process failures. Therefore, the data must be recorded so that it can be written to the HBase when the cluster is restored.

The failed data returned by the HBase client will not be automatically rewritten. The interface caller is only informed of the data failed to be written. To prevent data loss, measures must be taken to temporarily save the data in a file or in memory.

### Correct:

```
private List<Row> errorList = new ArrayList<Row>();  
/**  
 * <Insert data in PutList mode. >  
 * <Synchronization is not required if the method is not called by multiple threads.>  
 * @param put a data record  
 * @throws IOException  
 * @see [class, class#method, class#member]  
 */  
public synchronized void putData(Put put)  
{  
    // Temporarily cache data in this List.  
    dataList.add(put);  
    // Perform a Put operation when the dataList size reaches PUT_LIST_SIZE.  
    if (dataList.size() >= PUT_LIST_SIZE)  
    {  
        try  
        {  
            demoTable.put(dataList);  
        }  
        catch (IOException e)  
        {  
            // If RetriesExhaustedWithDetailsException occurs,  
            // certain data failed to be written, which  
            // is caused by process errors in the HBase cluster or migration of a large number of  
            // Regions.  
            if (e instanceof RetriesExhaustedWithDetailsException)  
            {  
                RetriesExhaustedWithDetailsException ree =  
                    (RetriesExhaustedWithDetailsException)e;  
                int failures = ree.getNumExceptions();  
                for (int i = 0; i < failures; i++)  
                {  
                    errorList.add(ree.getRow(i));  
                }  
            }  
            dataList.clear();  
        }  
    }  
}
```

## Release resources

Call the Close method to release resources when the ResultScanner and Table instances are not required. To enable the Close method to be called, add the Close method to the **finally** block.

### Correct:

```
ResultScanner scanner = null;
try
{
    scanner = demoTable.getScanner(s);
    //Do Something here.
}
finally
{
    scanner.close();
}
```

### Incorrect:

1. The code does not call the scanner.close() method to release resources.
2. The scanner.close() method is not placed in the **finally** block.

```
ResultScanner scanner = null;
scanner = demoTable.getScanner(s);
//Do Something here.
scanner.close();
```

## Add fault-tolerance mechanism for Scan

Exceptions, such as lease expiration, may occur when Scan is performed. Retry operations need to be performed when exceptions occur.

Retry operations can be applied in HBase-related interface methods to improve fault tolerance capabilities.

## Stop Admin as soon as it is not required

Stop Admin as soon as possible. Do not cache the same Admin instance for an extended period of time.

## 3.8.3 Suggestions

### Do not call the closeRegion method of Admin to close a Region

Admin interface provides an API to close a Region:

```
public void closeRegion(final String regionname, final String serverName)
```

When this method is used to close a Region, the HBase Client sends an RPC request to the RegionServer of the Region to be closed. The Master is unaware of the whole process. That is, the Master does not know even if the Region is closed. If the closeRegion method is called when the Master determines to migrate the Region based on the execution result of Balance, the Region cannot be closed or migrated. (In the current HBase version, this issue has not been resolved).

Therefore, do not call the closeRegion method of Admin to close a Region.

## Write data in PutList mode

Table provides two data write interfaces:

- public void put(final Put put) throws IOException
- public void put(final List<Put> puts) throws IOException

The second one is recommended because it provides better performance than the first one.

## Specify StartKey and EndKey for a Scan

A Scan with a specific range offers higher performance than a Scan without specific range.

### Example:

```
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("familyname"),Bytes.toBytes("columnname"));
scan.setStartRow( Bytes.toBytes("rowA")); // StartKey is rowA.
scan.setStopRow( Bytes.toBytes("rowB")); // EndKey is rowB.
for(Result result : demoTable.getScanner(scan)) {
// process Result instance
}
```

## Do not disable WAL

Write-Ahead-Log (WAL) allows data to be written in a log file before being stored in the database.

WAL is enabled by default. The Put class provides an interface to disable WAL:

```
public void setWriteToWAL(boolean write)
```

If WAL is disabled (writeToWAL is set to False), data of the last 1s (The time can be specified by the **hbase.regionserver.optionallogflushinterval** parameter on the RegionServer. It is 1s by default) will be lost. WAL can be disabled only when high data write speed is required and data loss of the last 1s is allowed.

## Set blockcache to true when creating a table or when Scan is performed

Set blockcache to true when a table is created or when Scan is performed on the HBase client. If there are a large number of repeated records, setting this parameter to true can improve efficiency.

By default, blockcache is true. Avoid setting this parameter to false forcibly, for example:

```
HColumnDescriptor fieldADesc = new HColumnDescriptor("value".getBytes());
fieldADesc.setBlockCacheEnabled(false);
```

## The HBase does not support query by Orderby or with the search criteria specified. It is based on the lexicographic order and can only be read by Rowkey.

HBase should not be used in scenarios of random query and sequencing.

## Suggestions on Services List Design

1. Pre-allocate regions in a balanced manner in order to improve concurrency capabilities.
2. Avoid excessive hotspot regions. Import the time factor to Rowkey if necessary.
3. It is preferred that concurrently accessed data be stored continuously. Concurrently read data should be stored nearby, on the same row and in the same cell.
4. Put frequently queried attributes property before Rowkey. Rowkey should be designed to match the main query criteria in terms of criterion sequencing.
5. Attributes with high dispersions should be contained in RowKey. Design the services list based on data dispersion and query scenarios.
6. Store redundant information to enhance indexing performance. Use secondary index to adapt to more query scenarios.
7. Enable automatic deletion of expired data by setting the expiration time and version quantity.

 NOTE

In the HBase, Regions busy writing data are called hotspot Region.

## 3.8.4 Examples

### Set Configuration parameters

To set up a connection between an HBase Client and the HBase Server, set the following parameters:

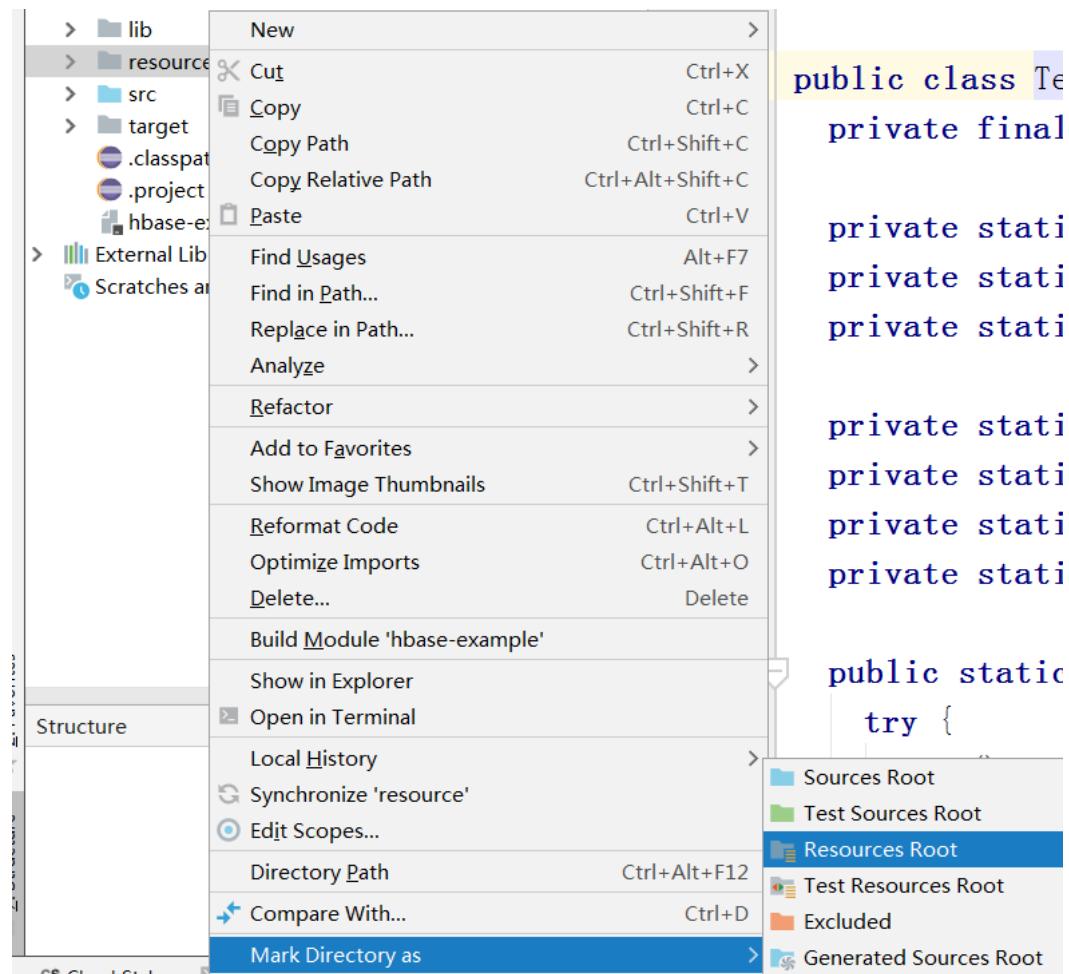
- hbase.zookeeper.quorum: IP address of Zookeeper. If there are multiple Zookeeper nodes, separate multiple IP addresses by a comma (,).
- hbase.zookeeper.property.clientPort: Port of Zookeeper.

 NOTE

The Configuration instance created by using HBaseConfiguration.create() will be automatically loaded with the configuration items in the following files:

- core-default.xml
- core-site.xml
- hbase-default.xml
- hbase-site.xml

Save these configuration files in **Source Folder**. To create a **Source Folder**, create a **resource** folder in the project, right-click the folder, and choose **Mark Directory as > Resources Root**.



The following table describes the parameters to be configured on the client.

#### NOTE

Do not change the values of these parameters.

Parameter	Description
hbase.client.pause	Specifies the time to wait before a retry is performed when an exception occurs. The actual time is calculated based on this value and the number of retries.
hbase.client.retries.number	Specifies the number of retries to be performed when an exception occurs.
hbase.client.retries.longer.multiplier	This parameter is related to the number of retries.
hbase.client.rpc.maxattempts	Specifies the number of retries when the RPC request is not successfully sent.
hbase.regionserver.lease.period	This parameter (in ms) is related to the Scanner timeout period.

Parameter	Description
hbase.client.write.buffer	This parameter is invalid if AutoFlush is enabled. If AutoFlush is disabled, the HBase Client caches the data to be written. When the size of the data cached reaches the specified limit, the HBase Client initiates a write operation to the HBase cluster.
hbase.client.scanner.caching	Specifies the number of rows allowed for a next request during a Scan.
hbase.client.keyvalue.maxsize	Specifies the maximum value of a keyvalue.
hbasehtable.threads.max	Specifies the maximum number of threads related to data operations in an HTable instance.
hbase.client.prefetch.limit	Before reading or writing data, the client must obtain the address of the Region. Therefore, a client can have some Region addresses pre-cached. This parameter is related to the configuration of the number of Region addresses pre-cached.

**Example:**

```
hbaseConfig = HBaseConfiguration.create();
//You do not need to set the following parameters if they are specified in the configuration files.
hbaseConfig.set("hbase.zookeeper.quorum", "172.16.100.1,172.16.100.2,172.16.100.3");
hbaseConfig.set("hbase.zookeeper.property.clientPort", "24002");
```

## Use HTablePool in multi-thread write operations

Use HTablePool for multiple data write threads. Observe the following when using HTablePool to perform multi-thread write operations:

1. Enable multiple date write threads to share the same HTablePool instance.
2. Specify maxSize of the HTableInterface instance when instantiating HTablePool. That is, instantiate the class using the following constructor function:

```
public HTablePool(final Configuration config, final int maxSize)
```

The value of maxSize can be determined based on Threads (the number of data write threads) and Tables (the number of user tables). Generally, maxSize cannot be greater than the product of Threads and Tables. (maxSize <= Threads x Tables)

3. The client thread obtains an HTableInterface instance with the table name of tableName using HTablePool#getTable(tableName).
4. An HTableInterface instance can be used by only one thread at a time.
5. If HTableInterface is not used, call HTablePool#putTable(HTableInterface table) to release it.

**Example:**

```
/**
```

```
* A certain number of retries is required after a data write failure. The time to wait before each retry is
```

```
determined based on the number of retries performed.  
*/  
private static final int[] RETRIES_WAITTIME = {1, 1, 1, 2, 2, 4, 4, 8, 16, 32};  
/**  
 * Specify the number of retries.  
 */  
private static final int RETRIES = 10;  
/**  
 * The unit of the time to wait after a failure.  
 */  
private static final int PAUSE_UNIT = 1000;  
private static Configuration hadoopConfig;  
private static HTablePool tablePool;  
private static String[] tables;  
/**  
 * <Initialize HTablePool>  
 * <Function description>  
 * @param config  
 * @see [class, class#method, class#member]  
 */  
public static void initTablePool()  
{  
    DemoConfig config = DemoConfig.getInstance();  
    if (hadoopConfig == null)  
    {  
        hadoopConfig = HBaseConfiguration.create();  
        hadoopConfig.set("hbase.zookeeper.quorum", config.getZookeepers());  
        hadoopConfig.set("hbase.zookeeper.property.clientPort", config.getZookeeperPort());  
    }  
    if (tablePool == null)  
    {  
        tablePool = new HTablePool(hadoopConfig, config.getTablePoolMaxSize());  
        tables = config.getTables().split(",");  
    }  
}  
public void run()  
{  
    // Initialize HTablePool. Initialize this instance only once because it is shared by multiple threads.  
    initTablePool();  
    for (;;) {  
        Map<String, Object> data = DataStorage.takeList();  
        String tableName = tables[(Integer) data.get("table")];  
        List<Put> list = (List) data.get("list");  
        // Use Row as the Key and save all puts in the List. This set is used only for querying the data failed to be  
        // written when a write operation fails, because the Server only returns the Row of the data failed.  
        Map<byte[], Put> rowPutMap = null;  
        // Perform the operation again if it fails (even if some of the data failed to be written). Only the data failed  
        // to be written is submitted each time.  
        INNER_LOOP :  
        for (int retry = 0; retry < RETRIES; retry++) {  
            // Obtain an HTableInterface instance from HTablePool. Release the instance if it is not required.  
            HTableInterface table = tablePool.getTable(tableName);  
            try {  
                table.put(list);  
                // The operation is successful.  
                break INNER_LOOP;  
            } catch (IOException e) {  
                // If the exception type is RetriesExhaustedWithDetailsException, some of the data failed to be written. The  
                // exception occurs because the processes in the HBase cluster are abnormal or a large number of Regions are  
                // being migrated.  
                // If the exception type is not RetriesExhaustedWithDetailsException, insert all the data in the list again.  
                if (e instanceof RetriesExhaustedWithDetailsException) {  
                    RetriesExhaustedWithDetailsException ree =
```

```
(RetriesExhaustedWithDetailsException)e;
int failures = ree.getNumExceptions();
System.out.println("In this operation, [" + failures + "] data records failed to be inserted.");
// Instantiate the Map when a retry is performed upon the first failure.
if (rowPutMap == null)
{
    rowPutMap = new HashMap<byte[], Put>(failures);
    for (int m = 0; m < list.size(); m++)
    {
        Put put = list.get(m);
        rowPutMap.put(put.getRow(), put);
    }
}
//Clear the original data and then add the data failed to be written.
list.clear();
for (int m = 0; m < failures; m++)
{
    list.add(rowPutMap.get(ree.getRow(m)));
}
}
}
finally
{
    // Release the instance after using it.
    tablePool.putTable(table);
}
// If an exception occurs, wait some time after releasing the HTableInterface instance.
try
{
    sleep(getWaitTime(retry));
}
catch (InterruptedException e1)
{
    System.out.println("Interruped");
}
}
}
}
```

## Create a Put instance

HBase is a column-oriented database. One column of data may correspond to multiple column families, and one column family may correspond to multiple columns. Before data is written, the column (column family name and column name) must be specified.

	ColumnFamily01					ColumnFamily02			
	column01	column02	column03	column04	column05	column01	column02	column03	column04
Row--01									
Row--02									
Row--03									
Row--04									
Row--05									
Row--06									
Row--07									
Row--08									

A Put instance must be created before a row of data is written in an HBase table. The Put instance data consists of (Key, Value). The Value can contain multiple columns of values.

When a (Key, Value) record is added to a Put instance, the family, qualifier, and value added are byte sets. Use the Bytes.toBytes method to convert character strings to byte sets. Do not use the String.toBytes method, because this method

cannot ensure correct data coding. Errors occur when the Key or Value contains Chinese characters.

**Example:**

```
//The column family name is privateInfo.  
private final static byte[] FAMILY_PRIVATE = Bytes.toBytes("privateInfo");  
//The privateInfo column family has two columns: "name" and "address".  
private final static byte[] COLUMN_NAME = Bytes.toBytes("name");  
private final static byte[] COLUMN_ADDR = Bytes.toBytes("address");  
/**  
 * <Create a Put instance. >  
 * <A put instance with one column family and two columns of data is created. >  
 * @param rowKey Key key value  
 * @param name name  
 * @param address address  
 * @return  
 * @see [class, class#method, class#member]  
 */  
public Put createPut(String rowKey, String name, String address)  
{  
    Put put = new Put(Bytes.toBytes(rowKey));  
    put.add(FAMILY_PRIVATE, COLUMN_NAME, Bytes.toBytes(name));  
    put.add(FAMILY_PRIVATE, COLUMN_ADDR, Bytes.toBytes(address));  
    return put;  
}
```

## Create an HBaseAdmin instance

**Example:**

```
private Configuration demoConf = null;  
private HBaseAdmin hbaseAdmin = null;  
/**  
 * <Constructor function>  
 * Import the Configuration instances.  
 */  
public HBaseAdminDemo(Configuration conf)  
{  
    this.demoConf = conf;  
    try  
    {  
        // Instantiate HBaseAdmin  
        hbaseAdmin = new HBaseAdmin(this.demoConf);  
    }  
    catch (MasterNotRunningException e)  
    {  
        e.printStackTrace();  
    }  
    catch (ZooKeeperConnectionException e)  
    {  
        e.printStackTrace();  
    }  
    }  
/**  
 * <Examples of method using>  
 * <For details about more methods, see the HBase interface documents. >  
 * @throws IOException  
 * @throws ZooKeeperConnectionException  
 * @throws MasterNotRunningException  
 * @see [Class, class#method, class#member]  
 */  
public void demo() throws MasterNotRunningException, ZooKeeperConnectionException, IOException  
{  
    byte[] regionName = Bytes.toBytes("mrtest,jjj,1315449869513.fc41d70b84e9f6e91f9f01affdb06703.");  
    byte[] encodeName = Bytes.toBytes("fc41d70b84e9f6e91f9f01affdb06703");  
    // Reallocate a Region.  
    hbaseAdmin.unassign(regionName, false);
```

```
// Actively initiate Balance.  
hbaseAdmin.balancer();  
// Move a Region. The second parameter is HostName+StartCode of RegionServer, for example,  
// host187.example.com,60020,1289493121758. If this parameter is set to null, the Region will be moved at  
random.  
hbaseAdmin.move(encodeName, null);  
// Check whether a table exists.  
hbaseAdmin.tableExists("tableName");  
// Check whether a table is activated.  
hbaseAdmin.isTableEnabled("tableName");  
}  
/**  
 * <Method used to rapidly create a table >  
 * <Create an HTableDescriptor instance, which contains description of the HTable to be created. Create the  
column families, which are associated with the HColumnDescriptor instance. In this example, the column  
family name is "columnName".>  
* @param tableName table name  
* @return  
* @see [Class, class#method, class#member]  
*/  
public boolean createTable(String tableName)  
{  
try {  
if (hbaseAdmin.tableExists(tableName)) {  
return false;  
}  
HTableDescriptor tableDesc = new HTableDescriptor(tableName);  
HColumnDescriptor fieldADesc = new HColumnDescriptor("columnName".getBytes());  
fieldADesc.setBlocksize(640 * 1024);  
tableDesc.addFamily(fieldADesc);  
hbaseAdmin.createTable(tableDesc);  
} catch (Exception e) {  
e.printStackTrace();  
return false;  
}  
return true;  
}
```

## 3.8.5 Appendix

### Parameters Batch and Caching for Scan

Batch: specifies the maximum number of data records returned each time when scan calls the next interface. It is related to the number of **columns** read each time.

Caching: specifies the maximum number of next values returned for an RPC request. It is related to the number of **rows** obtained by each RPC.

The following examples explain the functions of these two parameters in Scan:

A Region contains two rows (rowkey) of data in table A. Each row has 1000 columns, and each column has only one version, that is, each row has 1000 key values.

-	Colu A1	Colu A2	Colu A3	Colu A4	...	Colu N1	Colu N2	Colu N3	Colu N4
Row1	-	-	-	-	...	-	-	-	-
Row2	-	-	-	-	...	-	-	-	-

- **Example 1:** If Batch is not specified and Caching is 2, 2000 (Key, Value) records will be returned for each RPC request.
- **Example 2:** If Batch is set to 500 and Caching is 2, 1000 (Key, Value) records will be returned for each RPC request.
- **Example 3:** If Batch is set to 300 and Caching is 4, 1000 (Key, Value) records will be returned for each RPC request.

### Further explanation of Batch and Caching

- Each Caching indicates a chance of data request.
- The value of Batch determines whether a row of data can be read in a Caching. If the value of Batch is smaller than the total columns in a row, this row of data can be read in at least two Caching operations (the next Caching starts from the data where the previous caching stops).
- Each Caching cannot cross rows. That is, if the value of Batch is not reached after a row of data is read, data of the next row will not be read.

This can further explain the results of the previous examples.

- **Example 1:**  
Since Batch is not set, all columns of that row will be read by default. As Caching is 2, 2000 (Key, Value) records will be returned for each RPC request.
- **Example 2:**  
Because Batch is 500 and Caching is 2, a maximum of 500 columns of data will be read in each Caching. Therefore, 1000 (Key, Value) records will be returned after two times of caching.
- **Example 3:**  
Because Batch is 300 and Caching is 4, four times of caching are required to read 1000 data records. Therefore, only 1000 (Key, Value) records will be returned.

### Code example:

```
Scan s = new Scan();
//Set the start and end keys for data query.
s.setStartRow(Bytes.toBytes("01001686138100001"));
s.setStopRow(Bytes.toBytes("01001686138100002"));
s.setBatch(1000);
s.setCaching(100);
ResultScanner scanner = null;
try {
scanner = tb.getScanner(s);
for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {
for (KeyValue kv : rr.raw()) {
//Display the query results.
System.out.println("key:" + Bytes.toString(kv.getRow())
+ "getQualifier:" + Bytes.toString(kv.getQualifier())
+ "value" + Bytes.toString(kv.getValue()));
}
}
} catch (IOException e) {
System.out.println("error!" + e.toString());
} finally {
scanner.close();
}
```

## 3.9 HDFS

### 3.9.1 Application Scenarios

Hadoop distribute file system (HDFS) runs on commodity hardware. It provides high error tolerance. In addition, it supports high data access throughput and is suitable for applications that involve large-scale data sets.

The HDFS is applicable to the following scenarios:

- Massive data processing (higher than the TB or PB level).
- High-throughput demanding scenarios.
- High-reliability demanding scenarios.
- Good-scalability demanding scenarios.

The HDFS is not applicable to the scenarios that involve a large number of small files, random write, and low-latency read.

### 3.9.2 Rules

#### Set the HDFS NameNode metadata storage path

NameNode metadata is stored in  `${BIGDATA_DATA_HOME}/namenode/data` by default. This parameter sets the storage path of HDFS metadata.

#### Enable NameNode image backup for the HDFS

`fs.namenode.image.backup.enable` specifies whether to enable the NameNode image backup function. You need to set this parameter to `true`. Then the system can periodically back up the NameNode data.

#### Set the HDFS DataNode data storage path

DataNode data is stored in  `${BIGDATA_DATA_HOME}/hadoop/dataN/dn/datadir` by default. *N* indicates the number of directories is greater than or equal to 1.

For example,  `${BIGDATA_DATA_HOME}/hadoop/data1/dn/datadir`,  `${BIGDATA_DATA_HOME}/hadoop/data2/dn/datadir`.

After the storage path is set, data is stored in the corresponding directory of each mounted disk on a node.

#### Improve HDFS read/write performance

The data write process is as follows:

After receiving service data and obtaining the data block number and location from the NameNode, the HDFS client contacts DataNodes and establishes a pipeline with the DataNodes to be written. Then, the HDFS client writes data to DataNode1 using a proprietary protocol, and DataNode1 writes data to

DataNode2 and DataNode3 (three duplicates). After data is written, a message is returned to the HDFS client.

1. Set a proper block size. For example, set **dfs.blocksize** to **268435456** (256 MB).
2. It is not necessary to cache the big data that is not reused. In this case, set the following parameters to **false**:  
**dfs.datanode.drop.cache.behind.reads** and  
**dfs.datanode.drop.cache.behind.writes**

## Set the MapReduce intermediate file storage path

Only one default path is provided for storing MapReduce intermediate files, that is,  **\${hadoop.tmp.dir}/mapred/local**. It is recommended that intermediate files be stored on each disk.

For example, **/hadoop/hdfs/data1/mapred/local**, **/hadoop/hdfs/data2/mapred/local**, **/hadoop/hdfs/data3/mapred/local**. Directories that do not exist are automatically ignored.

## Release applied resources in finally during Java development.

Applied HDFS resources are released in try/finally and cannot be released outside the try statement only. Otherwise, resource leakage occurs.

## HDFS file operation APIs

Almost all Hadoop file operation classes are in the **org.apache.hadoop.fs** package. These APIs support operations such as opening, reading, writing, and deleting a file. FileSystem is the interface class provided for users in the Hadoop class library. FileSystem is an abstract class. Concrete classes can be obtained only using the get method. The get method has multiple overload versions, and the following get method is often used.

```
static FileSystem get(Configuration conf);
```

This class encapsulates almost all file operations, such as mkdir and delete. The program library framework for file operations is as follows:

```
operator()
{
    Obtain the Configuration object.
    Obtain the FileSystem object.
    Perform file operations.
}
```

## HDFS initialization method

HDFS initialization is a prerequisite for using APIs provided by HDFS.

To initialize HDFS, load the HDFS service configuration file, implement Kerberos security authentication, and instantiate FileSystem. Obtain keytab files for Kerberos security authentication in advance.

Example:

```
private void init() throws IOException {
    Configuration conf = new Configuration();
```

```
// Read a configuration file.  
conf.addResource("user-hdfs.xml");  
// Implement security authentication in security mode.  
if ("kerberos".equalsIgnoreCase(conf.get("hadoop.security.authentication"))){  
String PRINCIPAL = "username.client.kerberos.principal";  
String KEYTAB = "username.client.keytab.file";  
// Set the keytab key file.  
conf.set(KEYTAB, System.getProperty("user.dir") + File.separator + "conf" + File.separator +  
conf.get(KEYTAB));  
// Set the Kerberos configuration file path. */  
String krbfilepath = System.getProperty("user.dir") + File.separator + "conf" + File.separator + "krb5.conf";  
System.setProperty("java.security.krb5.conf", krbfilepath);  
// Implement login authentication. */  
SecurityUtil.login(conf, KEYTAB, PRINCIPAL);  
}  
// Instantiate FileSystem.  
fSystem = FileSystem.get(conf);  
}
```

## Upload local files to the HDFS

**FileSystem.copyFromLocalFile (Path src, Path dst)** is used to upload local files to a specified directory in the HDFS. *src* and *dst* indicate complete file paths.

Example:

```
public class CopyFile {  
    public static void main(String[] args) throws Exception {  
        Configuration conf=new Configuration();  
        FileSystem hdfs=FileSystem.get(conf);  
        //Local file  
        Path src =new Path("D:\\HebutWinOS");  
        //To the HDFS  
        Path dst =new Path("/");  
        hdfs.copyFromLocalFile(src, dst);  
        System.out.println("Upload to"+conf.get("fs.default.name"));  
        FileStatus files[]=hdfs.listStatus(dst);  
        for(FileStatus file:files){  
            System.out.println(file.getPath());  
        }  
    }  
}
```

## Create files on the HDFS

**FileSystem.mkdirs (Path f)** is used to create folders on HDFS. *f* indicates a complete folder path.

Example:

```
public class CreateDir {  
    public static void main(String[] args) throws Exception{  
        Configuration conf=new Configuration();  
        FileSystem hdfs=FileSystem.get(conf);  
        Path dfs=new Path("/TestDir");  
        hdfs.mkdirs(dfs);  
    }  
}
```

## Query the modification time of an HDFS file

**FileSystem.getModificationTime()** is used to query the modification time of a specified HDFS file.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem hdfs=FileSystem.get(conf);
    Path fpath =new Path("/user/hadoop/test/file1.txt");
    FileStatus fileStatus=hdfs.getFileStatus(fpath);
    long modiTime=fileStatus.getModificationTime();
    System.out.println("file1.txt modification time is"+modiTime);
}
```

## Read all files in an HDFS directory

**FileStatus.getPath()** is used to query all files in an HDFS directory.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem hdfs=FileSystem.get(conf);
    Path listf =new Path("/user/hadoop/test");

    FileStatus stats[]=hdfs.listStatus(listf);
    for(int i = 0; i < stats.length; ++i) {
        System.out.println(stats[i].getPath().toString());
    }
    hdfs.close();
}
```

## Query the location of a specified file in an HDFS cluster

**FileSystem.getFileBlockLocation (FileStatus file, long start, long len)** is used to query the location of a specified file in an HDFS cluster. *file* indicates a complete file path, and *start* and *len* specify the file path.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem hdfs=FileSystem.get(conf);
    Path fpath=new Path("/user/hadoop/cygwin");

    FileStatus filestatus = hdfs.getFileStatus(fpath);
    BlockLocation[] blkLocations = hdfs.getFileBlockLocations(filestatus, 0, filestatus.getLen());

    int blockLen = blkLocations.length;
    for(int i=0;i < blockLen;i++){
        String[] hosts = blkLocations[i].getHosts();
        System.out.println("block_"+i+"_location:"+hosts[0]);
    }
}
```

## Obtain all node names in an HDFS cluster

**DatanodeInfo.getHostName()** is used to obtain all node names in an HDFS cluster.

Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    FileSystem fs=FileSystem.get(conf);

    DistributedFileSystem hdfs = (DistributedFileSystem)fs;

    DatanodeInfo[] dataNodeStats = hdfs.getDataNodeStats();
```

```
for(int i=0;i < dataNodeStats.length;i++){
    System.out.println("DataNode_"+i+"_Name:"+dataNodeStats[i].getHostName());
}
```

## Multithread security login mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

login example code:

```
private Boolean login(Configuration conf){
    boolean flag = false;
    UserGroupInformation.setConfiguration(conf);
    try {
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

relogin example code:

```
public Boolean relogin(){
    boolean flag = false;
    try {
        UserGroupInformation.getLoginUser().reloginFromKeytab();
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "
+UserGroupInformation.isLoginKeytabBased());
        flag = true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return flag;
}
```

### NOTICE

Repetitive logins will cause a newly created session to overwrite the previous session. As a result, the previous session cannot be maintained or monitored, and some functions are unavailable after the previous session expires.

## 3.9.3 Suggestions

### Notes for reading and writing HDFS files

The HDFS does not support random read/write.

Data can be appended only to the end of an HDFS file.

Only data stored in the HDFS supports append. **edit.log** and metadata files do not support append. When using the append function, set **dfs.support.append** in *hdfs-site.xml* to **true**.

 NOTE

- **dfs.support.append** is disabled by default in open-source versions but enabled by default in FusionInsight versions.
- This parameter is a server parameter. You are advised to enable this parameter to use the append function.
- Store data in other modes, such as HBase, if the HDFS is not applicable.

## The HDFS is not suitable for storing a large number of small files

The HDFS is not suitable for storing a large number of small files because the metadata of small files will consume excessive memory resources of the NameNode.

## Back up HDFS data in three duplicates

Three duplicates are enough for DataNode data backup. System data security is improved when more duplicates are generated but system efficiency is reduced. When a node is faulty, data on the node is balanced to other nodes.

## Periodical HDFS Image Back-up

The system can back up the data on NameNode periodically after the image back-up parameter **fs.namenode.image.backup.enable** is set to **true**.

## Provide operations to ensure data reliability

When you invoke the write function to write data, HDFS client does not write the data to HDFS but caches it in the client memory. If the client is abnormal, power-off, the data will be lost. For high-reliability demanding data, invoke hflush to refresh the data to HDFS after writing finishes.

# 3.10 Hive

## 3.10.1 Application Scenarios

Hive is an open-source data warehouse framework built on Hadoop. It provides storage of structured data and basic data analysis services using the Hive query language (HQL), a language like the structured query language (SQL). Hive converts HQL statements to MapReduce or Spark tasks to query and analyze massive data stored in Hadoop clusters.

Hive provides the following features:

- Extracts, transforms, and loads (ETL) data using HQL.
- Analyzes massive structured data using HQL.
- Supports multiple data storage formats, including JavaScript object notation (JSON), comma separated values (CSV), TextFile, RCFile, ORCFILE, and SequenceFile.
- Multiple client connection modes. JDBC interfaces are supported.

Hive is applicable to offline massive data analysis (such as log and cluster status analysis), large scale data mining (such as user behavior analysis, interest region analysis, and region display), and other scenarios.

To ensure Hive high availability (HA), user data security, and service access security, Huawei MRS incorporates the following features based on Hive 3.1.0:

- Kerberos security authentication.
- Data file encryption.
- Comprehensive rights management.

## 3.10.2 Rules

### Load the Hive JDBC driver

The client software connects to HiveServer using Java database connectivity (JDBC). Therefore, you must load the JDBC driver class org.apache.hive.jdbc.HiveDriver for Hive.

Use the current class loader to load the driver class.

If there is no jar package in classpath, the client software throws "Class Not Found" and exits.

Example:

```
Class.forName("org.apache.hive.jdbc.HiveDriver").newInstance();
```

### Set up a database connection

The driver management class java.sql.DriverManager of JDK is used to obtain a connection to the Hive database.

The Hive database URL is url="jdbc:hive2://  
xxx10.64xxx.22xxx.231xxx:24002,10xxx.64xxx.22xxx.232xxx:24002,10xxx.64xxx.22xx  
x.233xxx:24002/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserv  
er2;sasl.qop=auth-conf;auth=KERBEROS;principal=hive/  
hadoop.hadoop.com@HADOOP.COM;user.principal=hive/  
hadoop.hadoop.com;user.keytab=conf/hive.keytab";

In this example, ZooKeeper is deployed on three nodes and the default port is 24002. **xxx.xxx.xxx.xxx** indicates each of the IP addresses of the three nodes. The user name and password are null or empty because authentication has been performed successfully.

Example:

```
// Set up a connection.  
connection = DriverManager.getConnection(url, "", "");
```

### Execute HQL

Note that the HQL statement cannot end with a semicolon (;).

**Correct:**

```
String sql = "SELECT COUNT(*) FROM employees_info";  
Connection connection = DriverManager.getConnection(url, "", "");
```

```
PreparedStatement statement = connection.prepareStatement(sql);
resultSet = statement.executeQuery();
```

**Incorrect:**

```
String sql = "SELECT COUNT(*) FROM employees_info;";
Connection connection = DriverManager.getConnection(url, "", "");
PreparedStatement statement = connection.prepareStatement(sql);
resultSet = statement.executeQuery();
```

## Close a database connection

After the client executes the HQL, close the database connection to prevent memory leakage.

Close the statement and connection objects of the JDK.

Example:

```
finally {
    if (null != statement) {
        statement.close();
    }

    // Close the JDBC connection.
    if (null != connection) {
        connection.close();
    }
}
```

## HQL syntax used to check for null values

Use **is null** to check whether a field is empty, that is, the field has no value. Use **is not null** to check whether a field is not null, that is, the field has a value.

If you use **is null** for a character whose type is String and length is 0, False is returned. Use **col = "** to check for null values, and use **col != "** to check for non-null values.

**Correct:**

```
select * from default.tbl_src where id is null;
select * from default.tbl_src where id is not null;
select * from default.tbl_src where name = "";
select * from default.tbl_src where name != "";
```

**Incorrect:**

```
select * from default.tbl_src where id = null;
select * from default.tbl_src where id != null;
select * from default.tbl_src where name is null;
select * from default.tbl_src where name is not null;
```

Note that the type of the id field in the tbl\_src table is Int, and the type of the name field is String.

## The client configuration parameters must be consistent with the server configuration parameters

If the configuration parameters of the Hive, YARN, and HDFS servers of the cluster are modified, the related parameter in a client program will be modified. You need to check whether the configuration parameters submitted to the HiveServer before the configuration parameters are modified are consistent with those on the

servers. If the configuration parameters are inconsistent, modify them on the client and submit them to the HiverServer. In the following example, if the parameter of YARN in the cluster is modified, the parameter submitted to the HiverServer from the Hive client and sample program before the modification must be reviewed and modified.

Initial state:

The parameter configuration of YARN in the cluster is as follows:

```
mapreduce.reduce.java.opts=-Xmx2048M
```

The parameter configuration on the client is as follows:

```
mapreduce.reduce.java.opts=-Xmx2048M
```

The parameter configuration of YARN in the cluster after the modification is as follows:

```
mapreduce.reduce.java.opts=-Xmx1024M
```

If the parameter in the client program is not changed, the parameter is still valid. This will result in insufficient memory for reducer and lead to MR running failure.

## Multithread security login mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

login example code:

```
private Boolean login(Configuration conf){  
    boolean flag = false;  
    UserGroupInformation.setConfiguration(conf);  
  
    try {  
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));  
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "+  
+UserGroupInformation.isLoginKeytabBased());  
        flag = true;  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    return flag;  
}
```

relogin example code:

```
public Boolean relogin(){  
    boolean flag = false;  
    try {  
  
        UserGroupInformation.getLoginUser().reloginFromKeytab();  
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "+  
+UserGroupInformation.isLoginKeytabBased());  
        flag = true;  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    return flag;  
}
```

## Prerequisites for using the REST interface of WebHCat to submit an MR task in Streaming mode

The REST interface depends on the streaming packages of Hadoop. Before submitting an MR task to WebHCat in Streaming mode, upload **hadoop-streaming-2.7.0.jar** to the specified path of the HDFS: **hdfs:///apps/templeton/hadoop-streaming-2.7.0.jar**. Log in to the node where the client and Hive service are installed. Assume that the client installation path is **/opt/hadoopclient**.

```
source /opt/hadoopclient/bigdata_env
```

Run the **kinit** command to log in to the node as the human-machine or machine-machine user.

```
hdfs dfs -put ${BIGDATA_HOME}/FusionInsight_HD_8.3.1/FusionInsight-Hadoop-*/hadoop/share/hadoop/tools/lib/hadoop-streaming-*jar /apps/templeton/
```

**/apps/templeton/** need to be modified based on different instances. The default instance uses **/apps/templeton/** and the Hive1 instance uses **/apps1/templeton/**. The others follow the same rule

## Read and write operations cannot be performed on the same table at the same time

Currently, Hive does not support concurrent operations. Read and write operations cannot be performed on the same table at the same time. Otherwise, query results may be inaccurate and tasks may fail.

## A bucket table does not support insert into

A bucket table does not support insert into, and only supports insert overwrite; otherwise, the number of files and the number of buckets will be inconsistent.

## Prerequisites for using some REST interfaces of WebHCat

Some REST interfaces of WebHCat depend on the JobHistoryServer instance of MapReduce. The interfaces are as follows:

- mapreduce/jar(POST)
- mapreduce/streaming(POST)
- hive(POST)
- jobs(GET)
- jobs/:jobid(GET)
- jobs/:jobid(DELETE)

## Hive Authorization Description

It is recommended that Hive authorization (databases, tables, or views) be performed on the Manager authorization page. Authorization in command-line interface is not recommended except in the **alter databases databases\_name set owner='user\_name'** scenario.

## Hive on HBase partition tables cannot be created

Data of Hive on HBase tables is stored on HBase. Because HBase tables are divided into multiple partitions that are scattered on RegionServer, Hive on HBase partition tables cannot be created on Hive.

## A Hive on HBase table does not support insert overwrite

HBase uses a RowKey to uniquely identify a record. If data to be inserted has the same RowKey as the existing data, HBase will use the new data to overwrite the existing data. If insert overwrite is performed for a Hive on HBase table on Hive, only data with the same RowKey will be overwritten.

## 3.10.3 Suggestions

### HQL - implicit type conversion

If the query statements use the field value for filtering, do not use the implicit type conversion of Hive to compile HQL. The reason is that the implicit type conversion is not conducive to code reading and migration.

#### Correct:

```
select * from default.tbl_src where id = 10001;  
select * from default.tbl_src where name = 'TestName';
```

#### Incorrect:

```
select * from default.tbl_src where id = '10001';  
select * from default.tbl_src where name = TestName;
```



#### NOTE

Note that the type of the id field in the tbl\_src table is Int, and the type of the name field is String.

### HQL - object name length

The HQL object names include table names, field names, view names, and index names. It is recommended that the object name not exceed 30 bytes.

An error is reported if an object name of Oracle exceeds 30 bytes. PT also limits object names to 30 bytes.

Excessive long object names are not conducive to code reading, migration, and maintenance.

### HQL - statistics of data records

To count the total number of records in a table, use select count(1) from table\_name.

To count the number of valid records for a field in a table, use select count(column\_name) from table\_name.

## JDBC - timeout limit

The JDBC provided by Hive supports timeout limit. The default value is 5 minutes. Users can use `java.sql.DriverManager.setLoginTimeout(int seconds)` to change the value. The unit of `seconds` is second.

## UDF Management

It is recommended that the administrator creates permanent UDF. This is done to avoid repeated execution of the add jar statement and UDF redefining.

UDF of Hive has some default properties. For example, the default value of **deterministic** is **true** (indicating that the same result will be returned for the same input), and the default value of **stateful** is **true**. Corresponding annotations should be added when user-defined UDF conducts an internal data summary. The following is an example:

```
@UDFType(deterministic = false)
Public class MyGenericUDAFEvaluator implements Closeable {
```

## Suggestions on Optimizing Table Partitions

1. It is advised to use partition tables and store data by day when data volume is large and statistics need to be collected on a daily basis.
2. In order to avoid excessive small files, add **distribute by** to the partition field during dynamic partition data insertion.

## Suggestions on Optimizing Storage File Formats

Hive supports multiple storage formats, including TextFile, RCFile, ORC, Sequence, and Parquet. If you want to save storage space or query certain fields for the most of time, use columnar storage, for example, ORC files, to create tables.

## 3.10.4 Examples

### JDBC Secondary Development Code Example 1

The following code example provides the following functions:

1. Provides the username and key file path in the JDBC URL address so that programs can automatically perform security logins and create Hive connections.
2. Runs HQL statements for creating, querying, and deleting tables.

```
package com.huawei.bigdata.hive.example;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Properties;
```

```
import org.apache.hadoop.conf.Configuration;
import com.huawei.bigdata.security.LoginUtil;

public class JDBCExample {
    private static final String HIVE_DRIVER = "org.apache.hive.jdbc.HiveDriver";

    private static final String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";
    private static final String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";
    private static final String ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL = "zookeeper/hadoop.hadoop.com";

    private static Configuration CONF = null;
    private static String KRB5_FILE = null;
    private static String USER_NAME = null;
    private static String USER_KEYTAB_FILE = null;

    private static String zkQuorum = null;//IP address and port list of a ZooKeeper node
    private static String auth = null;
    private static String sasl_qop = null;
    private static String zooKeeperNamespace = null;
    private static String serviceDiscoveryMode = null;
    private static String principal = null;
    private static String auditAddition = null;
    private static void init() throws IOException{
        CONF = new Configuration();

        Properties clientInfo = null;
        String userdir = System.getProperty("user.dir") + File.separator
            + "conf" + File.separator;
        InputStream fileInputStream = null;
        try{
            clientInfo = new Properties();
            // "hiveclient.properties" is the client configuration file. If the multi-instance feature is used, you need to
            replace the file with "hiveclient.properties" of the corresponding instance client.
            // "hiveclient.properties" file is stored in the config directory of the decompressed installation package of
            the corresponding instance client.
            String hiveclientProp = userdir + "hiveclient.properties" ;
            File propertiesFile = new File(hiveclientProp);
            fileInputStream = new FileInputStream(propertiesFile);
            clientInfo.load(fileInputStream);
        }catch (Exception e) {
            throw new IOException(e);
        }finally{
            if(fileInputStream != null){
                fileInputStream.close();
                fileInputStream = null;
            }
        }
        //The format of zkQuorum is "xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002";
        // "xxx.xxx.xxx.xxx" of zkQuorum indicates the IP address of the node where ZooKeeper locates. The
        default port is 24002.
        zkQuorum = clientInfo.getProperty("zk.quorum");
        auth = clientInfo.getProperty("auth");
        sasl_qop = clientInfo.getProperty("sasl.qop");
        zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
        serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
        principal = clientInfo.getProperty("principal");
        auditAddition = clientInfo.getProperty("auditAddition");
        // Set a user name for the newly created user, where xxx indicates the username created previously. For
        example, if the created user is user, USER_NAME is user.
        USER_NAME = "xxx";

        if ("KERBEROS".equalsIgnoreCase(auth)) {
            // Set the keytab and krb5 file path on the client.
            USER_KEYTAB_FILE = userdir + "user.keytab";
            KRB5_FILE = userdir + "krb5.conf";
            System.setProperty("java.security.krb5.conf", KRB5_FILE);
            System.setProperty(ZOOKEEPER_SERVER_PRINCIPAL_KEY, ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
        }
    }
}
```

```
}

/**
 * This example shows how to use the Hive JDBC interface to run the HQL command <br>
 * <br>
 *
 * @throws ClassNotFoundException
 * @throws IllegalAccessException
 * @throws InstantiationException
 * @throws SQLException
 * @throws IOException
 */
public static void main(String[] args) throws InstantiationException,
    IllegalAccessException, ClassNotFoundException, SQLException, IOException{
    // Parameter Initialization
    init();

    // Define HQL. HQL must be a single statement and cannot contain ";".
    String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT, name STRING)",
        "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};

    // Build JDBC URL
    StringBuilder sBuilder = new StringBuilder(
        "jdbc:hive2://").append(zkQuorum).append("/");

    if ("KERBEROS".equalsIgnoreCase(auth)) {
        sBuilder.append(";serviceDiscoveryMode=")
            .append(serviceDiscoveryMode)
            .append(";zooKeeperNamespace=")
            .append(zooKeeperNamespace)
            .append(";sasl.qop=")
            .append(sasl_qop)
            .append(";auth=")
            .append(auth)
            .append(";principal=")
            .append(principal)
            .append(";user.principal=")
            .append(USER_NAME)
            .append(";user.keytab=")
            .append(USER_KEYTAB_FILE);
    } else {
        //Normal mode
        sBuilder.append(";serviceDiscoveryMode=")
            .append(serviceDiscoveryMode)
            .append(";zooKeeperNamespace=")
            .append(zooKeeperNamespace)
            .append(";auth=none");
    }
    if (auditAddition != null && !auditAddition.isEmpty()) {
        strBuilder.append(";auditAddition=").append(auditAddition);
    }
    String url = sBuilder.toString();

    // Load the Hive JDBC driver.
    Class.forName(HIVE_DRIVER);

    Connection connection = null;
    try {
        // Obtain the JDBC connection.
        // If the normal mode is used, the second parameter needs to be set to a correct username. Otherwise,
        the anonymous user will be used for login.
        connection = DriverManager.getConnection(url, "", "");

        // Create a table.
        // If data needs to be imported to the table, use the load statement to import data to the table, for
        example, import data from the HDFS to the table.
        // load data inpath '/tmp/employees.txt' overwrite into table employees_info;
        execDDL(connection,sqls[0]);
        System.out.println("Create table success!");
    }
}
```

```
// Query the table.  
execDML(connection,sqls[1]);  
  
// Delete the table  
execDDL(connection,sqls[2]);  
System.out.println("Delete table success!");  
}catch (Exception e) {  
    System.out.println("Create connection failed : " + e.getMessage());  
}  
finally {  
    // Close the JDBC connection.  
    if (null != connection) {  
        connection.close();  
    }  
}
```

```
public static void execDDL(Connection connection, String sql)  
throws SQLException {  
    PreparedStatement statement = null;  
    try {  
        statement = connection.prepareStatement(sql);  
        statement.execute();  
    }  
    finally {  
        if (null != statement) {  
            statement.close();  
        }  
    }  
}
```

```
public static void execDML(Connection connection, String sql) throws SQLException {  
    PreparedStatement statement = null;  
    ResultSet resultSet = null;  
    ResultSetMetaData resultMetaData = null;  
  
    try {  
        // Execute HQL.  
        statement = connection.prepareStatement(sql);  
        resultSet = statement.executeQuery();  
  
        // Export the queried column names to the console.  
        resultMetaData = resultSet.getMetaData();  
        int columnCount = resultMetaData.getColumnCount();  
        for (int i = 1; i <= columnCount; i++) {  
            System.out.print(resultMetaData.getColumnName(i) + '\t');  
        }  
        System.out.println();  
  
        // Export the query results to the console.  
        while (resultSet.next()) {  
            for (int i = 1; i <= columnCount; i++) {  
                System.out.print(resultSet.getString(i) + '\t');  
            }  
            System.out.println();  
        }  
    }  
    finally {  
        if (null != resultSet) {  
            resultSet.close();  
        }  
  
        if (null != statement) {  
            statement.close();  
        }  
    }  
}
```

}

## JDBC Secondary Development Code Example 2

The following code example provides the following functions:

1. Does not provide the username and key file path in the JDBC URL address to create Hive connections. Users perform security logins by themselves.
2. Runs HQL statements for creating, querying, and deleting tables.

 NOTE

When accessing ZooKeeper, programs need to use the jaas configuration file, for example, **user.hive.jaas.conf**. The details are as follows:

```
Client {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useKeyTab=true  
    keyTab="D:\\workspace\\jdbc-examples\\conf\\user.keytab"  
    principal="xxx@HADOOP.COM"  
    useTicketCache=false  
    storeKey=true  
    debug=true;  
};
```

You need to modify the keyTab path (absolute path) and principal in the configuration file based on the actual environment, and set environment variable **java.security.auth.login.config** to the file path.

```
package com.huawei.bigdata.hive.example;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.SQLException;  
import java.util.Properties;  
  
import org.apache.hadoop.conf.Configuration;  
import com.huawei.bigdata.security.LoginUtil;  
  
public class JDBCExamplePreLogin {  
    private static final String HIVE_DRIVER = "org.apache.hive.jdbc.HiveDriver";  
  
    private static final String ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME = "Client";  
    private static final String ZOOKEEPER_SERVER_PRINCIPAL_KEY = "zookeeper.server.principal";  
    private static final String ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL = "zookeeper/hadoop";  
  
    private static Configuration CONF = null;  
    private static String KRB5_FILE = null;  
    private static String USER_NAME = null;  
    private static String USER_KEYTAB_FILE = null;  
  
    private static String zkQuorum = null;//IP address and port list of a ZooKeeper node  
    private static String auth = null;  
    private static String sasl_qop = null;  
    private static String zooKeeperNamespace = null;  
    private static String serviceDiscoveryMode = null;  
    private static String principal = null;  
    private static String auditAddition = null;  
    private static void init() throws IOException{  
        CONF = new Configuration();  
  
        Properties clientInfo = null;  
        String userdir = System.getProperty("user.dir") + File.separator  
            + "conf" + File.separator;  
        InputStream fileInputStream = null;  
        try{  
            clientInfo = new Properties();  
            // "hiveclient.properties" is the client configuration file. If the multi-instance feature is used, you  
            // need to replace the file with "hiveclient.properties" of the corresponding instance client.  
            // "hiveclient.properties" file is stored in the config directory of the decompressed installation  
            // package of the corresponding instance client.  
            String hiveclientProp = userdir + "hiveclient.properties" ;  
        } catch (IOException e){  
            e.printStackTrace();  
        }  
    }  
}
```

```

File propertiesFile = new File(hiveclientProp);
fileInputStream = new FileInputStream(propertiesFile);
clientInfo.load(fileInputStream);
}catch (Exception e) {
    throw new IOException(e);
}finally{
    if(fileInputStream != null){
        fileInputStream.close();
        fileInputStream = null;
    }
}
//The format of zkQuorum is "xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002";
// "xxx.xxx.xxx.xxx" of zkQuorum indicates the IP address of the node where ZooKeeper locates. The
default port is 24002.
zkQuorum = clientInfo.getProperty("zk.quorum");
auth = clientInfo.getProperty("auth");
sasl_qop = clientInfo.getProperty("sasl.qop");
zooKeeperNamespace = clientInfo.getProperty("zooKeeperNamespace");
serviceDiscoveryMode = clientInfo.getProperty("serviceDiscoveryMode");
principal = clientInfo.getProperty("principal");
auditAddition = clientInfo.getProperty("auditAddition");
// Set a user name for the newly created user, where xxx indicates the username created previously.
For example, if the created user is user, USER_NAME is user.
USER_NAME = "xxx";

if ("KERBEROS".equalsIgnoreCase(auth)) {
    // Set the keytab and krb5 file path on the client.
    USER_KEYTAB_FILE = userdir + "user.keytab";
    KRB5_FILE = userdir + "krb5.conf";

    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME, USER_NAME,
    USER_KEYTAB_FILE);
    LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
    ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);

    // Security mode
    // Zookeeper Login Authentication
    LoginUtil.login(USER_NAME, USER_KEYTAB_FILE, KRB5_FILE, CONF);
}
}

/**
 * This example shows how to use the Hive JDBC interface to run the HQL command <br>.
 * <br>
 *
 * @throws ClassNotFoundException
 * @throws IllegalAccessException
 * @throws InstantiationException
 * @throws SQLException
 * @throws IOException
 */
public static void main(String[] args) throws InstantiationException,
    IllegalAccessException, ClassNotFoundException, SQLException, IOException{
// Parameter Initialization
init();

// Define HQL. HQL must be a single statement and cannot contain ";".
String[] sqls = {"CREATE TABLE IF NOT EXISTS employees_info(id INT,name STRING)",
    "SELECT COUNT(*) FROM employees_info", "DROP TABLE employees_info"};

// Build JDBC URL
StringBuilder sBuilder = new StringBuilder(
    "jdbc:hive2://").append(zkQuorum).append("/");

if ("KERBEROS".equalsIgnoreCase(auth)) {
    sBuilder.append(";serviceDiscoveryMode=")
        .append(serviceDiscoveryMode)
        .append(";zooKeeperNamespace=")
        .append(zooKeeperNamespace)
}

```

```
.append(";sasl.qop=")
.append(sasl_qop)
.append(";auth=")
.append(auth)
.append(";principal=")
.append(principal);
} else {
// Normal mode
sBuilder.append(";serviceDiscoveryMode=")
.append(serviceDiscoveryMode)
.append(";zooKeeperNamespace=")
.append(zooKeeperNamespace)
.append(";auth=none");
}
if (auditAddition != null && !auditAddition.isEmpty()) {
strBuilder.append(";auditAddition=").append(auditAddition);
}
String url = sBuilder.toString();

// Load the Hive JDBC driver.
Class.forName(HIVE_DRIVER);

Connection connection = null;
try {
// Obtain the JDBC connection.
// If the normal mode is used, the second parameter needs to be set to a correct username.
Otherwise, the anonymous user will be used for login.
connection = DriverManager.getConnection(url, "", "");

// Create a table.
// If data needs to be imported to the table, use the load statement to import data to the table,
for example, import data from the HDFS to the table.
// load data inpath '/tmp/employees.txt' overwrite into table employees_info;
execDDL(connection,sqls[0]);
System.out.println("Create table success!");

// Query the table.
execDML(connection,sqls[1]);

// Delete the table
execDDL(connection,sqls[2]);
System.out.println("Delete table success!");
}
finally {
// Close the JDBC connection.
if (null != connection) {
connection.close();
}
}
}

public static void execDDL(Connection connection, String sql)
throws SQLException {
PreparedStatement statement = null;
try {
statement = connection.prepareStatement(sql);
statement.execute();
}
finally {
if (null != statement) {
statement.close();
}
}
}

public static void execDML(Connection connection, String sql) throws SQLException {
PreparedStatement statement = null;
ResultSet resultSet = null;
```

```
ResultSetMetaData resultMetaData = null;

try {
    // Execute HQL.
    statement = connection.prepareStatement(sql);
    resultSet = statement.executeQuery();

    // Export the queried column names to the console.
    resultMetaData = resultSet.getMetaData();
    int columnCount = resultMetaData.getColumnCount();
    for (int i = 1; i <= columnCount; i++) {
        System.out.print(resultMetaData.getColumnName(i) + '\t');
    }
    System.out.println();

    // Export the query results to the console.
    while (resultSet.next()) {
        for (int i = 1; i <= columnCount; i++) {
            System.out.print(resultSet.getString(i) + '\t');
        }
        System.out.println();
    }
} finally {
    if (null != resultSet) {
        resultSet.close();
    }

    if (null != statement) {
        statement.close();
    }
}
}
```

## HCatalog Secondary Development Code Example

The following code example demonstrates how to use the HCatInputFormat and HCatOutputFormat interfaces provided by HCatalog to submit MapReduce jobs.

```
public class HCatalogExample extends Configured implements Tool {

    public static class Map extends
        Mapper<LongWritable, HCatRecord, IntWritable, IntWritable> {
        int age;
        @Override
        protected void map(
            LongWritable key,
            HCatRecord value,
            org.apache.hadoop.mapreduce.Mapper<LongWritable, HCatRecord,
                IntWritable, IntWritable>.Context context)
            throws IOException, InterruptedException {
            age = (Integer) value.get(0);
            context.write(new IntWritable(age), new IntWritable(1));
        }
    }

    public static class Reduce extends Reducer<IntWritable, IntWritable,
        IntWritable, HCatRecord> {
        @Override
        protected void reduce(
            IntWritable key,
            java.lang.Iterable<IntWritable> values,
            org.apache.hadoop.mapreduce.Reducer<IntWritable, IntWritable,
                HCatRecord>.Context context)
            throws IOException, InterruptedException {
            int sum = 0;
            Iterator<IntWritable> iter = values.iterator();
            while (iter.hasNext()) {
                sum += iter.next().get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
}
```

```
        while (iter.hasNext()) {
            sum++;
            iter.next();
        }
        HCatRecord record = new DefaultHCatRecord(2);
        record.set(0, key.get());
        record.set(1, sum);

        context.write(null, record);
    }
}

public int run(String[] args) throws Exception {
    Configuration conf = getConf();
    String[] otherArgs = args;

    String inputTableName = otherArgs[0];
    String outputTableName = otherArgs[1];
    String dbName = "default";

    @SuppressWarnings("deprecation")
    Job job = new Job(conf, "GroupByDemo");

    HCatInputFormat.setInput(job, dbName, inputTableName);
    job.setInputFormatClass(HCatInputFormat.class);
    job.setJarByClass(HCatalogExample.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(WritableComparable.class);
    job.setOutputValueClass(DefaultHCatRecord.class);

    OutputJobInfo outputjobInfo = OutputJobInfo.create(dbName, outputTableName, null);
    HCatOutputFormat.setOutput(job, outputjobInfo);
    HCatSchema schema = outputjobInfo.getOutputSchema();
    HCatOutputFormat.setSchema(job, schema);
    job.setOutputFormatClass(HCatOutputFormat.class);

    return (job.waitForCompletion(true) ? 0 : 1);
}
public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new HCatalogExample(), args);
    System.exit(exitCode);
}
```

## 3.11 Hudi

### 3.11.1 Applicable Scenarios

**Full data analysis:** When an analysis needs to read full data in a table, you can use the real-time view of Hudi to provide the latest full data for the analysis engine.

**Quick data analysis:** When an analysis poses higher requirements on analysis performance than eventually consistent or full data, you can use the read-optimized view of Hudi to improve read efficiency.

**Incremental data analysis:** In incremental data extract, transform and load (ETL) and online analytical processing (OLAP) scenarios, you can use the incremental view of Hudi to read the latest incremental data or the incremental data

submitted at a specified time, eliminating the need to search the entire full data and significantly improving the read performance.

**Historical image data analysis:** To analyze data at a historical time point, you can use Multiversion Concurrency Control (MVCC) of Hudi to read image data of a specific version.

## 3.11.2 Suggestions

Currently, Hudi is mainly applicable to real-time data import to the lake and incremental data ETL. Stored historical data can be imported to Hudi tables in batches.

Copy on write (COW) tables apply to scenarios where the incremental data is basically new data and that have high requirements on data read performance.

Merge on read (MOR) tables apply to scenarios that have high requirements on data import performance and where the incremental data contains a large amount of added and updated data.

You are advised to use the date field to set partition paths in hoodie keys.

Configure Hudi resources for real-time data importing to the data lake in based on the number of Kafka partitions. One Kafka partition can be consumed by only one executor-core. Therefore, setting excessive executor-cores wastes resources.

Set the consumption batch parameters for Spark Streaming to write data to the data lake based on site requirements. Ensure that the interval between two batches is slightly smaller than the time required for consuming a batch of messages to write data into the Hudi table.

The degree of parallelism (DOP) of Hudi write operations cannot be too large. A proper DOP helps shorten the processing time.

## 3.12 IoTDB

### 3.12.1 Applicable Scenarios

IoTDB is a data management engine that integrates collection, storage, and analysis of time series data. Due to its lightweight structure, high performance, and ease of use together with its intense integration with Hadoop, IoTDB meets the requirements of high-speed write and complex analysis and query on massive time series data in industrial IoT applications.

### 3.12.2 Rules

#### Set a Proper Number of Storage Groups

A proper number of storage groups can improve performance. The system processes I/O requests first in the memory cache and then switch to the stored files or folders if no result is found. In this regard, a large number of storage groups will store too many files or folders, take up large amounts of memory, and slow down the system I/O speed as a result. A small number will reduce the concurrency and block write commands.

Set a balanced number of storage groups based on your data scale and usage scenarios to achieve better system performance.

## All Time Series Must Start with root and End with the Sensor

The time series can be considered as the complete path of the sensor that generates the time series data. In IoTDB, all time series must start with **root** and end with the sensor.

### 3.12.3 Suggestions

#### Use the Native Session API to Avoid SQL Concatenation

For the sample of calling the IoTDB Session API in a cluster in security mode, see [IoTDB Session](#). For that in a cluster in common mode, see [IoTDB Session](#).

#### Use Write APIs with High Performance Based on Service Requirements

The write APIs are ranked as follows in descending order of their performance:

insertTablets (inserts multiple tablets, from multiple rows of the same column on multiple devices) >  
insertTablet (inserts a tablet, from multiple rows of the same column on a single device) >  
insertRecordsOfOneDevice (inserts multiple records, from multiple rows of different columns on a single device) >  
insertRecords(Object value) (inserts multiple records, from multiple rows of different columns on multiple devices) >  
insertRecords(String value) (inserts multiple records, from multiple rows of different columns on multiple devices) >  
insertRecord (inserts a record, from only one row on a single device)

#### Do Not Use the Same Client to Initiate Concurrent Connections

An IoTDB client can connect to only one IoTDBServer. A large number of concurrent connections from the same client to IoTDBServer will deteriorate the connection performance. You can use multiple clients to connect to IoTDBServer based on service requirements to achieve load balancing.

#### Use SessionPool to Reuse Connections

Use the distributed technology to cache sessions so that the client does not need to create sessions for each read or write request. Alternatively, use SessionPool to reuse connections.

#### Close ResultSet and SessionDataSet in a Timely Manner

Close ResultSet and SessionDataSet in a timely manner to avoid resource wastes.

## 3.13 Kafka

### 3.13.1 Application Scenarios

Kafka is a distributed message releasing and subscription system. It provides features such as message persistence, high-throughput, multi-client support, and real-time message processing. Kafka is applicable to online and offline message consumption as well as Internet service massive data collection scenarios, such as conventional data collection, active website tracking, aggregation of operation data in statistics systems (monitoring data), and log collection.

Reasons for using the message system

- Decoupling: The message system inserts a hidden, data-based interface layer during data processing.
- Redundancy: Message queues are persistent, preventing data loss.
- Scalability: Message queues are decoupled from data processing, facilitating the expansion of data processing.
- Recoverability: Data processing can be recovered in case of failures.
- Sequence guarantee: Message queues help ensure the message sequence and keep the messages in a partition in order.
- Asynchronous communication: Messages can join the queue for further processing when necessary.

### 3.13.2 Rules

#### Create Topics by calling Kafka APIs (`AdminZkClient.createTopic`)

- For Java programming languages, correct examples are as follows:

```
import kafka.zk.AdminZkClient;
import kafka.zk.KafkaZkClient;
import kafka.admin.RackAwareMode;
...
KafkaZkClient kafkaZkClient = KafkaZkClient.apply(zkUrl, JaasUtils.isZkSecurityEnabled(),
zkSessionTimeoutMs, zkConnectionTimeoutMs, Int.MaxValue(), Time.SYSTEM, "", "", null);
AdminZkClient adminZkClient = new AdminZkClient(kafkaZkClient);
adminZkClient.createTopic(topic, partitions, replicas, new Properties(), RackAwareMode.Enforced
$.MODULE$);
...
```
- For Scala programming languages, correct examples are as follows:

```
import kafka.zk.AdminZkClient;
import kafka.zk.KafkaZkClient;
...
val kafkaZkClient: KafkaZkClient = KafkaZkClient.apply(zkUrl, JaasUtils.isZkSecurityEnabled(),
zkSessionTimeoutMs, zkConnectionTimeoutMs, Int.MaxValue, Time.SYSTEM, "", "")
val adminZkClient: AdminZkClient = new AdminZkClient(kafkaZkClient)
adminZkClient.createTopic(topic, partitions, replicas)
```

## The number of Partition copies must be less than or equal to the number of nodes

Copies of Topic Partitions in Kafka are used for improving data reliability. Copies of the same Partition are distributed on different nodes. Therefore, the number of Partition copies must be less than or equal to the number of nodes.

### Set the `fetch.message.max.bytes` parameter of the Consumer client

The value of `fetch.message.max.bytes` must be equal to or greater than the maximum number of bytes of messages that the Producer client generates each time. If the value is too small, the messages generated by the Producer client cannot be consumed successfully by the Consumer client.

## 3.13.3 Suggestions

### In the same group, the number of consumers and that of Topic Partitions to be consumed should be the same

If the number of consumers is greater than that of Topic Partitions, some consumers cannot consume Topics. If the number of consumers is smaller than that of Topic Partitions, concurrent consumption cannot be fully represented. Therefore, the number of consumers and that of Topic Partitions to be consumed should be the same.

### Avoid writing data with single ultra-large log

Data with single ultra-large log can affect efficiency and writing. Under such circumstance, modify the values of the `max.request.size` and `max.partition.fetch.bytes` configuration items when initializing Kafka producer instances and consumer instances, respectively.

For example, set `max.request.size` and `max.partition.fetch.bytes` to 5252880.

```
// Protocol type: configuration SASL_PLAINTEXT or PLAINTEXT
props.put(securityProtocol, kafkaProc.getValues(securityProtocol, "SASL_PLAINTEXT"));
// service name
props.put(saslKerberosServiceName, "kafka");
props.put("max.request.size", "5252880");
// Security protocol type
props.put(securityProtocol, kafkaProc.getValues(securityProtocol, "SASL_PLAINTEXT"));
// service name
props.put(saslKerberosServiceName, "kafka");
props.put("max.partition.fetch.bytes", "5252880");
```

## 3.14 Mapreduce

### 3.14.1 Application Scenarios

Files of Mapreduce are stored in the HDFS. MapReduce is a programming model used for parallel computation of large data sets (larger than 1 TB). It is advised to use MapReduce when the file being processed cannot be loaded to memory.

It is advised to use Spark if MapReduce is not required.

## 3.14.2 Rules

### Inherit the Mapper abstract class.

The map() and setup() methods are called during the Map procedure of a MapReduce task.

#### Example:

```
public static class MapperClass extends  
Mapper<Object, Text, Text, IntWritable> {  
    /**  
     * map input. The key indicates the offset of the original file, and the value is a row of characters in the  
     * original file.  
     * The map input key and value are provided by InputFormat. You do not need to set them. By default,  
     * TextInputFormat is used.  
     */  
    public void map(Object key, Text value, Context context)  
        throws IOException, InterruptedException {  
        //Custom implementation  
    }  
    /**  
     * The setup() method is called only once before the map() method of a map task or the reduce() method  
     * of a reduce task is called.*/  
    public void setup(Context context) throws IOException,  
        InterruptedException {  
        // Custom implementation  
    }  
}
```

### Inherit the Reducer abstract class.

The reduce() and setup() methods are called during the Reduce procedure of a MapReduce task.

#### Example:

```
public static class ReducerClass extends  
Reducer<Text, IntWritable, Text, IntWritable> {  
    /**  
     * @param The input is a collection iterator consisting of (key, value) pairs.  
     * Each map puts together all the pairs with the same key. The reduce method sums the number of the  
     * same keys.  
     * Call context.write(key, value) to write the output to the specified directory.  
     * Outputformat writes the (key, value) pairs output by reduce to the file system.  
     * By default, TextOutputFormat is used to write the reduce output to the HDFS.  
     */  
  
    public void reduce(Text key, Iterable<IntWritable> values,  
        Context context) throws IOException, InterruptedException {  
        // Custom implementation  
    }  
  
    /**  
     * The setup() method is called only once before the map() method of a map task or the reduce() method  
     * of a reduce task is called.  
     */  
  
    public void setup(Context context) throws IOException,  
        InterruptedException {  
  
        // Custom implementation. Context obtains the configuration information.  
    }
```

```
}
```

## Submit a MapReduce task.

Use the main() method to create a job, set parameters, and submit the job to the Hadoop cluster.

### Example:

```
public static void main(String[] args) throws Exception {
    Configuration conf = getConfiguration();
    // Input parameters for the main method: args[0] indicates the input path of the MR job. args[1] indicates
    // the output path of the MR job.
    String[] otherArgs = new GenericOptionsParser(conf, args)
        .getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: <in> <out>");
        System.exit(2);
    }
    Job job = new Job(conf, "job name");
    // Locate the jar package of the major task.
    job.setJar("D:\\job-examples.jar");
    // job.setJarByClass(TestWordCount.class);
    // Set the map and reduce classes to be executed. You can also specify them in the configuration file.
    job.setMapperClass(TOKENIZERMAPPERV1.class);
    job.setReducerClass(INTSUMREDUCERV1.class);
    // Set the combiner class. By default, it is not used. If it is used, it runs the same classes as reduce. Exercise
    // care when using the Combiner class. You can also specify the combiner class in the configuration file.
    job.setCombinerClass(INTSUMREDUCERV1.class);
    // Set the output type of the job. You can also specify it in the configuration file.
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    // Set the input and output paths for the job. You can also specify them in the configuration file.
    Path outputPath = new Path(otherArgs[1]);
    FileSystem fs = outputPath.getFileSystem(conf);
    // If the output path already exists, delete it.
    if (fs.exists(outputPath)) {
        fs.delete(outputPath, true);
    }
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

## 3.14.3 Suggestions

### Specify the global configuration items in mapred-site.xml

The mapred-site.xml file contains the following configuration items:

```
setMapperClass(Class <extends Mapper> cls) ->"mapreduce.job.map.class"
setReducerClass(Class<extends Reducer> cls) ->"mapreduce.job.reduce.class"
setCombinerClass(Class<extends Reducer> cls) ->"mapreduce.job.combine.class"
setInputFormatClass(Class<extends InputFormat> cls) ->"mapreduce.job.inputformat.class"
setJar(String jar) ->"mapreduce.job.jar"
setOutputFormat(Class< extends OutputFormat> theClass) ->"mapred.output.format.class"
setOutputKeyClass(Class<> theClass) ->"mapreduce.job.output.key.class"
setOutputValueClass(Class<> theClass) ->"mapreduce.job.output.value.class"
setPartitionerClass(Class<extends Partitioner> theClass) ->"mapred.partitionner.class"
setMapOutputCompressorClass(Class<extends CompressionCodec> codecClass)
->"mapreduce.map.output.compress"&"mapreduce.map.output.compress.codec"
setJobPriority(JobPriority prio) ->"mapreduce.job.priority"
setQueueName(String queueName) ->"mapreduce.job.queuename"
setNumMapTasks(int n) ->"mapreduce.job.maps"
setNumReduceTasks(int n) ->"mapreduce.job.reduces"
```

## 3.14.4 Examples

### Count the number of female netizens who dwell on online shopping for more than 2 hours at a weekend.

The operation involves three steps:

1. Filter the online time of female netizens in log files using the MapperClass inherited from the Mapper abstract class.
2. Calculate the online time of each female netizen and output information about the female netizens who dwell online for more than 2 hours using the ReducerClass inherited from the Reducer abstract class.
3. Use the main method to create a MapReduce job and then submit the MapReduce job to the Hadoop cluster.

#### Step 1: Use MapperClass to define the map() and setup() methods of the Mapper abstract class.

```
public static class MapperClass extends  
  
Mapper<Object, Text, Text, IntWritable> {  
    // Separator  
    String delim;  
    // Filter the sex.  
    String sexFilter;  
    private final static IntWritable timeInfo = new IntWritable(1);  
    private Text nameInfo = new Text();  
    /**  
     * map input. The key indicates the offset of the original file, and the value is a row of characters in the  
     * original file.  
     * The map input key and value are provided by InputFormat. You do not need to set them. By default,  
     * TextInputFormat is used.  
     */  
    public void map(Object key, Text value, Context context)  
        throws IOException, InterruptedException {  
        // A row of characters read.  
        String line = value.toString();  
        if (line.contains(sexFilter)) {  
            // Obtain the names.  
            String name = line.substring(0, line.indexOf(delim));  
            nameInfo.set(name);  
            // Obtain information about the online time.  
            String time = line.substring(line.lastIndexOf(delim),  
                line.length());  
            timeInfo.set(Integer.parseInt(time));  
            // map outputs (key, value) pairs.  
            context.write(nameInfo, timeInfo);  
        }  
    }  
    /**  
     * The setup() method is called only once before the map() method of a map task or the reduce() method  
     * of a reduce task is called.  
     */  
    public void setup(Context context) throws IOException,  
        InterruptedException {  
        // Obtain configuration information using Context.  
        sexFilter = delim + context.getConfiguration().get("log.sex.filter", "female") + delim;  
    }  
}
```

#### Step 2: Use CReducerClass to define the reduce() method of the Reducer abstract class.

```

public static class ReducerClass extends
Reducer<Text, IntWritable, Text, IntWritable> {
private IntWritable result = new IntWritable();
// Total time limit.
private int timeThreshold;
/**
 * @param The input is a collection iterator consisting of (key, value) pairs.
 * Each map puts together all the pairs with the same key. The reduce method sums the number of the
same keys.
 * Call context.write(key, value) to write the output to the specified directory.
 * Outputformat writes the (key, value) pairs output by reduce to the file system.
 * By default, TextOutputFormat is used to write the reduce output to the HDFS.
 */
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
// If the time is smaller than the time limit, no information will be returned.
if (sum < timeThreshold) {
return;
}
result.set(sum);
// reduce output: key: female netizen information, value: online time
context.write(key, result);
}
/**
 * The setup() method is called only once before the map() method of a map task or the reduce() method
of a reduce task is called.
*/
public void setup(Context context) throws IOException,
InterruptedException {
// Obtain configuration information using Context.
timeThreshold = context.getConfiguration().getInt(
"log.time.threshold", 120);
}
}

```

### Step 3: Use the main() method to create a job, set parameters, and submit the job to the Hadoop cluster.

```

public static void main(String[] args) throws Exception {
Configuration conf = getConfiguration();
// Input parameters for the main method: args[0] indicates the input path of the MR job. args[1] indicates
the output path of the MR job.
String[] otherArgs = new GenericOptionsParser(conf, args)
.getRemainingArgs();
if (otherArgs.length != 2) {
System.err.println("Usage: <in> <out>");
System.exit(2);
}
Job job = new Job(conf, "Collect Female Info");
// Locate the jar package of the major task.
job.setJar("D:\\mapreduce-examples\\hadoop-mapreduce-examples\\mapreduce-examples.jar");
// job.setJarByClass(TestWordCount.class);
// Set the map and reduce classes to be executed. You can also specify them in the configuration file.
job.setMapperClass(TokenizerMapperV1.class);
job.setReducerClass(IntSumReducerV1.class);
// Set the combiner class. By default, it is not used. If it is used, it runs the same classes as reduce. Exercise
care when using the Combiner class. You can also specify the combiner class in the configuration file.
job.setCombinerClass(IntSumReducerV1.class);
// Set the output type of the job. You can also specify it in the configuration file.
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
// Set the input and output paths for the job. You can also specify them in the configuration file.
Path outputPath = new Path(otherArgs[1]);
FileSystem fs = outputPath.getFileSystem(conf);
// If the output path already exists, delete it.

```

```
if (fs.exists(outputPath)) {  
    fs.delete(outputPath, true);  
}  
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
System.exit(job.waitForCompletion(true) - 0 : 1);  
}
```

## 3.15 Oozie

### 3.15.1 Application Scenarios

Oozie is a workflow engine used to manage Hadoop jobs. Oozie workflows are defined and described based on the directed acyclic graph (DAG). Oozie supports multiple workflow modes and workflow scheduled triggering mechanisms. Oozie provides features such as easy extensibility, convenient maintenance, and high reliability and works closely with each component in the Hadoop ecosystem.

Oozie workflows are classified into three types:

- Workflow: describes a complete basic service flow.
- Coordinator: is built on workflows, triggers workflows on a scheduled basis or based on the specified conditions.
- Bundle: is built on coordinators. It centrally schedules, controls, and manages coordinators.

Oozie provides the following features:

- Supports distribution, aggregation, and selection of workflow modes.
- Works closely with each component in the Hadoop ecosystem.
- Supports parameterized workflow variables.
- Supports scheduled workflow triggering.
- Supports high availability (HA).
- Provides a built-in web console that allows users to view and monitor workflows and view logs.

### 3.15.2 Rules

#### Modified configuration files except job.properties must be uploaded to the HDFS again

During workflow running, configuration files except **job.properties** are read from specified HDFS directories. If the file content in the HDFS is not synchronized, the modification does not take effect.

#### Each workflow has only one start node and one end node

Based on the syntax requirements, each workflow has only one start node and one end node. To handle the exceptions that may occur, multiple kill nodes can be configured.

## In the same workflow, each node can be traversed only once

A workflow does not support the ring structure, so each node can be traversed only once. The execution on the next node starts only when the execution on the current node ends.

## A default processing branch must be reserved during process branch selecting

A default processing branch must be reserved during node decision selecting. This prevents a workflow from entering uncontrollable error status and avoids uncontrollable process execution.

Correct:

```
<decision name="[NODE-NAME]">
  <switch>
    <case to="[NODE_NAME]">[PREDICATE]</case>
    ...
    <case to="[NODE_NAME]">[PREDICATE]</case>
    <default to="[NODE_NAME]" />
  </switch>
</decision>
```

## Fork and Join nodes must appear in pairs

Correct:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.2">
  ...
  <fork name="[FORK-NODE-NAME]">
    <path start="[NODE-NAME]" />
    ...
    <path start="[NODE-NAME]" />
  </fork>
  ...
  <join name="[JOIN-NODE-NAME]" to="[NODE-NAME]" />
  ...
</workflow-app>
```

## Tag sequence cannot be changed

The locations of tags in schemas are strictly restricted. Unnecessary tags can be omitted but the location sequence cannot be changed. The location sequence of tags in different schemas may be different.

Correct:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.2">
  ...
  <action name="[NODE-NAME]">
    <map-reduce>
      <resource-manager>[RESOURCE-MANAGER]</resource-manager>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]" />
        ...
        <mkdir path="[PATH]" />
        ...
      </prepare>
      ...
      <job-xml>[JOB-XML-FILE]</job-xml>
      <configuration>
```

```
<property>
    <name>[PROPERTY-NAME]</name>
    <value>[PROPERTY-VALUE]</value>
</property>
...
</configuration>
<file>[FILE-PATH]</file>
...
<archive>[FILE-PATH]</archive>
...
</map-reduce>
<ok to="[NODE-NAME]" />
<error to="[NODE-NAME]" />
</action>
...
</workflow-app>
```

## A Hive SQL statement ends with a semicolon (;) and supports single-row comment starting with double hyphens (--)

Correct:

```
create table A(id int, name string, dt string);
insert into A values(1, "a1", "20150625");
select * from A;
--drop table A;
```

## 3.15.3 Suggestions

### The name of a node must reflect the functions of a node

Standard name:

```
<action name="copyData">
    <fs>
        <delete path='${nameNode}/user/result' />
        <mkdir path='${nameNode}/user/result' />
        <move source='${nameNode}/output-data/map-reduce'
              target='${nameNode}/user/result' />
        <chmod path='${nameNode}/user/result'
              permissions='-rwxr-w-r-' dir-files='true' /></chmod>
    </fs>
    <ok to="end" />
    <error to="fail" />
</action>
```

Non-standard name:

```
<action name="fsNode">
    <fs>
        <delete path='${nameNode}/user/result' />
        <mkdir path='${nameNode}/user/result' />
        <move source='${nameNode}/output-data/map-reduce'
              target='${nameNode}/user/result' />
        <chmod path='${nameNode}/user/result'
              permissions='-rwxr-w-r-' dir-files='true' /></chmod>
    </fs>
    <ok to="end" />
    <error to="fail" />
</action>
```

### Configure pre-cleanup operations under Prepare for a task involving data overwriting

Correct:

```
<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="myfirstHadoopJob">
    <map-reduce>
      <resource-manager>foo:8021</resource-manager>
      <name-node>bar:25000</name-node>
      <prepare>
        <delete path="hdfs://foo:25000/usr/tucu/output-data"/>
      </prepare>
      <job-xml>/myfirstjob.xml</job-xml>
      <configuration>
        <property>
          <name>mapred.input.dir</name>
          <value>/usr/tucu/input-data</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/usr/tucu/input-data</value>
        </property>
        <property>
          <name>mapred.reduce.tasks</name>
          <value>${firstJobReducers}</value>
        </property>
        <property>
          <name>oozie.action.external.stats.write</name>
          <value>true</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="myNextAction"/>
    <error to="errorCleanup"/>
  </action>
  ...
</workflow-app>
```

## 3.16 Redis

### 3.16.1 Application Scenarios

Redis is a network-based and high-performance key-value memory database. It provides flexible data structures and data operations. It is mainly used in scenarios that feature high performance, low latency, diverse data structure access, and persistence. For example:

- Obtaining the latest N pieces of data, for example, the latest articles from a certain website.
- Ranking list application and operations to obtain top N piece of data. This operation is based on a certain condition, for example, sorting by times that people click Like, while the preceding operation gives priority to time.
- Applications that require precise expiration time, for example, user session information.
- Counter applications, for example, counters that record website access times.
- Constructing a queue system, for example, a message queue.
- Cache, for example, table data that is frequently accessed in a cached relational database.

## 3.16.2 Rules

### Replacing ClusterAdapter with JedisCluster and Replacing SingleAdapter with Jedis

After a version update, use the redis.clients.jedis.JedisCluster or redis.clients.jedis.Jedis interface.

The com.huawei.jredis.client.adapter.IJRedisClientAdapter interface is discarded.

### Disabling Resources After Use

After using jedis or JedisCluster, manually invoke the close() method to disable resources.

Code sample:

```
JedisCluster redis = new JedisCluster(set);
try {
    // SET key value associates key values of character strings to keys.
    redis.set("name", "wangjun1");
    redis.set("id", "123456");
} catch (Exception e) {
    e.printStackTrace();
} finally {
    redis.close();
}
```

### Using JedisPool in a Multithread Environment

Code sample:

```
JedisPool pool = new JedisPool(new JedisPoolConfig(), "localhost");
/// Jedis implements Closable. Hence, the jedis instance will be auto-closed after the last statement.
try (Jedis jedis = pool.getResource()) {
    /// ... do stuff here ... for example
    jedis.set("foo", "bar");
    String foobar = jedis.get("foo");
    jedis.zadd("sose", 0, "car"); jedis.zadd("sose", 0, "bike");
    Set<String> sose = jedis.zrange("sose", 0, -1);
}
/// ... when closing your application:
pool.destroy();
```

or

```
Jedis jedis = null;
try {
    jedis = pool.getResource();
    /// ... do stuff here ... for example
    jedis.set("foo", "bar");
    String foobar = jedis.get("foo");
    jedis.zadd("sose", 0, "car"); jedis.zadd("sose", 0, "bike");
    Set<String> sose = jedis.zrange("sose", 0, -1);
} finally {
    if (jedis != null) {
        jedis.close();
    }
}
/// ... when closing your application:
pool.destroy();
```

## Do Not Use \* Expressions, Such as KEYS \*

The time complexity of the KEYS command is  $O(n)$ , in which  $n$  indicates the number of keys to be returned. Therefore, the time complexity depends on a database size. When this operation is performed, other commands cannot be run in the instance.

You can run the SCAN command in a more friendly mode. The SCAN command is used to scan a database in incremental and iterative mode. This operation is performed by an iterator based on a cursor. You can stop or continue the operation as required. When a map set is used, use 16384 keys or less.

## Key Values Must Be Less than 512 MB

The system specifies that the maximum length of a value of the character string type in Redis is 512 MB.

## 3.16.3 Suggestions

### Parameter Interfaces of the Hash Type Being Recommended for hmset and hmget

It is recommended that less key pairs for hash-type data structure operations be used. The number of key pairs can be controlled based on JVM situation to prevent Java memory stack overflow.

### Method for Re-Creating jedisCluster

To re-create jedisCluster, set the Jmx attribute of GenericObjectPoolConfig to false; otherwise, re-creating jedisCluster objects will slow down.

Code sample:

```
GenericObjectPoolConfig config = new GenericObjectPoolConfig();
config.setJmxEnabled(false);
```

### Avoiding Operations Such as Scan in a Production Environment

The scan operation consumes more resources, and timeliness and accuracy of scan operation results cannot be ensured. Therefore, you are advised not to perform the scan operation in a production environment.

## 3.17 Solr

### 3.17.1 Application Scenarios

1. The data types to be retrieved are complex. Solr can clean, segment words, and create inverted index (create indexes) and then offer full-text retrieval (query) if the data to be queried include structured data (such as relational database), semi-structured data (such as web page and XML), and non-structured data (such as log, picture and image).
2. Search criteria are diversified, and ordinary search methods do not meet the requirements. For example, too many involved fields. The full-text retrieval

- (query) can include simple words and phrases or multiple forms of words and phrases.
3. Data read operations are far more than data write operations.

## 3.17.2 Rules

### Import Solr classes into Solr applications

Example:

```
//A class that needs to be imported when CloudSolrServer is created  
import org.apache.solr.client.solrj.impl.CloudSolrServer;  
//A class that needs to be imported when documents are created  
import org.apache.solr.common.SolrDocument;  
//A class that needs to be imported when index data is queried  
import org.apache.solr.client.solrj.SolrQuery;
```

### Configure the time consistency between the client and the server if a security version cluster is installed

Note: If a security version cluster is installed, the Kerberos authentication needs to be performed. The Kerberos authentication requires the time consistency between the server and the client. Pay attention to the time difference between different time zones when configuring the time consistency. If the time of the client and server is inconsistent, the client authentication fails and subsequent service processes cannot be executed.

### Configure authentication information and assign read/write rights when a customized user performs the index data operations in a collection

If a security version cluster is installed, you need to perform the Kerberos authentication to connect to the server. Perform the following operations to log in to the KDC.

```
private static void getConfiguration() {  
  
    // jaas.conf file, it is included in the client package file  
    System.setProperty("java.security.auth.login.config", "conf/jaas.conf");  
    // configuration file.  
    System.setProperty("java.security.krb5.conf", "conf/krb5.conf");  
}
```

Obtain the **krb5.conf** file and **user.keytable** file from the **User Management** page of FusionInsight Manager, and set principal to the username in **jaas.conf**.

The customized user needs to have read/write rights on the collection where operations will be performed. You can select the administrator role for the user or assign the collection read/write rights to the user.

### Configure HttpClient to the InsecureHttpClient mode if a security version cluster is installed

If a security version cluster is installed, you need to import the com.huawei.solr.client.solrj.impl.InsecureHttpClient package to the created HttpClient. The code is as follows.

```
ModifiableSolrParams params = new ModifiableSolrParams();  
params.set(HttpClientUtil.PROP_MAX_CONNECTIONS, 1000);
```

```
params.set(HttpClientUtil.PROP_MAX_CONNECTIONS_PER_HOST, 500);
HttpClient httpClient = HttpClientUtil.createClient(params);
if (SOLR_HDFS_KBS_ENABLED.equals("true")) {
    try {
        httpClient = new InsecureHttpClient(httpClient, params);
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
```

## Invoke the shutdown functions of cloudSolrServer and lbServer before an application program is stopped

Invoke `cloudSolrServer.shutdown()` and `lbServer.shutdown()` before an application program is stopped. If you invoke only `cloudSolrServer.shutdown()` in some abnormal scenarios (for example, the user does not have operation rights on the collection), `lbServer` starts a thread and then the program cannot be quit.

### 3.17.3 Suggestions

#### Collection Index Storage Location

**Table 3-1** Index storage location comparison

Item	Stored in HDFS	Stored On Local Disks
Disadvantages	<p>The performance delivered when indexes are stored in HDFS is 30% to 50% lower than that delivered when indexes are stored on local disks.</p> <p>Real-time writing speed of a single node is no faster than 2 Mbyte/s.</p> <p>Data expansion ratio is slightly higher than if data are stored on local disks.</p>	<p>The disks need to be managed, for example, a RAID level needs to be selected.</p> <p>Data needs to be managed, Solr does not have the automatic balance function, and therefore the relationship between disk partitions and Solr instances needs to be appropriately planned to maximize disk usage efficiency.</p> <p>Multiple copies are needed for data reliability.</p>
Advantages	<p>Only one replica is configured, and data reliability is guaranteed by the HDFS Replica mechanism.</p> <p>Data is managed by HDFS, and data on each node is balanced and convenient to migrate.</p>	<p>Real-time writing speed of single node is between 2 Mbyte/s to 4 Mbyte/s.</p> <p>Storing indexes on local disks delivers higher performance than storing indexes in HDFS.</p>

## Suggestions on Optimizing Solr Insertion

You do not need to perform the commit operation each time the batch index insertion is complete. Use the automatic committing function of Solr.

### NOTE

The commit operation will forcibly write index data to disks; therefore, invoking Commit frequently will decrease insertion efficiency.

## Shard Division Policy

Each Shard can process index and query requests. When setting the number of Shards, consider the following two aspects:

- It is recommended that the number of Shards be an integer multiple of that of SolrServer instances. Ensure that each SolrServer instance is balanced.
- To achieve optimal performance, each Shard contains a maximum of 100 million data records.

### NOTE

Generally, the memory of a SolrServer instance is 16 GB. One SolrCore occupies about 64 MB memory. Therefore, a SolrServer instance can contain 256 SolrCores. In dual-copy mode, a Solr instance can contain 128 Shards. The number of Shards can be calculated based on the number of SolrServer instances in the current cluster.

## User-defined Analyzers and Databases

Applications can use user-defined analyzers and databases when the built-in databases of Solr cannot meet the requirements. When using user-defined databases, ensure that the definitions of database types are consistent with the definitions of index fields.

### NOTE

For how to configure user-defined analyzers, refer to the "Word Filter Customization" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.

## Suggestions on Optimizing Schema

Fixed field attributes:

- **name**: Field name
- **Type**: Field type
- **indexed**: Whether the field is used to create indexes (concerning search and sequencing)
- **stored**: Whether to store the original value. The original value is returned when the value is set to **true**
- **multiValued**: Whether the field contains multiple values in the document
- **omitNorms**: Whether to omit normalization to save memory
- **required**: Whether the field must have value when a document is added
- **docValues**: Whether the field is oriented to column storage. **uniqueKey** is generally set to **true**.

#### NOTE

- **omitNorms**: Whether the field is oriented to omit normalization to save memory. The value is set to **true** by default while full-text field or field whose privilege needs to be escalated during index are set to **false**
- **docValues** is used to record the field value and is especially suitable for scenarios in which field values are obtained based on document IDs. For example, set the value to **true** during sorting, statistics collection, and Facet query

Several attributes of full-text fields:

- **termVectors**: If it is set to **true**, term vector will be saved to accelerate MoreLikeThis and Highlighting, which will generate storage overheads.
- **termPositions**: Whether to save the address of term vector.
- **termOffsets**: Whether to save the offset of term vector.

**dynamicField**

- Dynamic field setting is used to user-define field and wildcard (\*) later.

**copyField**

- Integrate multiple fields to one field.
- If multiple sources exist in the dest of copyField, multiValued of the dest field should be set to **true**.

Property of ordinary FieldType:

- **name**: field type name.
- **class**: field type implementation class.
- **positionIncrementGap**: specifies a distance for multiValued, optimizing phrase search.
- **precisionStep**: should be set for ordinary value fields. It requires more space consumption while increasing range search speed.

## 3.18 Spark

### 3.18.1 Application Scenarios

Spark is a distributed batch processing framework. It provides analysis and mining and iterative memory computing capabilities and supports application development in multiple programming languages, including Scala, Java, and Python. Spark is applicable to the following scenarios:

- Data processing: Spark can process data quickly and has fault tolerance and scalability.
- Iterative computation: Spark supports iterative computation to meet the requirements of multi-step data processing logic.
- Data mining: Spark supports complex mining and analysis of massive data and supports multiple data mining and machine learning algorithms.
- Streaming Processing: Spark supports streaming processing with a second-level delay and supports multiple external data sources.
- Query Analysis: Spark supports standard SQL query analysis, provides the DSL (DataFrame), and supports multiple external input types.

## 3.18.2 Rules

### Import the Spark class in Spark applications

- Example in Java:

```
//Class imported when SparkContext is created.  
import org.apache.spark.api.java.JavaSparkContext  
//Class imported for the RDD operation.  
import org.apache.spark.api.java.JavaRDD  
//Class imported when SparkConf is created.  
import org.apache.spark.SparkConf
```

- Example in Scala:

```
//Class imported when SparkContext is created.  
import org.apache.spark.SparkContext  
//Class imported for the RDD operatin.  
import org.apache.spark.SparkContext._  
//Class imported when SparkConf is created.  
import org.apache.spark.SparkConf
```

### Pay attention to the parameter transfer between the Driver and Executor nodes in distributed cluster

When Spark is used for programming, certain code logic needs to be determined based on the parameter entered. Generally, the parameter is specified as a global variable and assigned a null value. The actual value is assigned before the SparkContext object is instantiated using the main function. However, in the distributed cluster mode, the jar package of the executable program will be sent to each Executor. If the global variable values are changed only for the nodes in the main function and are not sent to the functions executing tasks, an error of null pointer will be reported.

#### Correct:

```
object Test  
{  
    private var testArg: String = null;  
    def main(args: Array[String])  
    {  
        testArg = i;  
        val sc: SparkContext = new SparkContext(i);  
  
        sc.textFile(i)  
            .map(x => testFun(x, testArg));  
    }  
  
    private def testFun(line: String, testArg: String): String =  
    {  
        testArg.split(i);  
        return i;  
    }  
}
```

#### Incorrect:

```
//Define an object.  
object Test  
{  
    // Define a global variable and set it to null. Assign a value to this variable before the SparkContext object  
    // is instantiated using the main function.  
    private var testArg: String = null;  
    //main function  
    def main(args: Array[String])
```

```
{  
    pair  
    testArg = i;  
    val sc: SparkContext = new SparkContext(i);  
  
    sc.textFile(i)  
        .map(x => testFun(x));  
}  
  
private def testFun(line: String): String =  
{  
    testArg.split(...);  
    return i;  
}
```

No error will be reported in the local mode of Spark. However, in the distributed cluster mode, an error of null pointer will be reported. In the cluster mode, the jar package of the executable program is sent to each Executor for running. When the testFun function is executed, the system queries the value of testArg from the memory. The value of testArg, however, is changed only when the nodes of the main function are started and other nodes are unaware of the change. Therefore, the value returned by the memory is null, which causes an error of null pointer.

## SparkContext.stop must be added before an application program stops

When Spark is used in secondary development, SparkContext.stop() must be added before an application program stops.



When Java is used in application development, JavaSparkContext.stop() must be added before an application program stops.

When Scala is used in application development, SparkContext.stop() must be added before an application program stops.

The following use Scala as an example to describe correct and incorrect examples.

### Correct:

```
//Submit a spark job.  
val sc = new SparkContext(conf)  
  
//Specific task  
...  
  
//The application program stops.  
sc.stop()
```

### Incorrect:

```
//Submit a spark job.  
val sc = new SparkContext(conf)  
  
//Specific task  
...
```

If you do not add SparkContext.stop, the YARN page displays the failure information. In the same task, as shown in [Figure 3-1](#), the first program does not add SparkContext.stop(), while the second program adds SparkContext.stop.

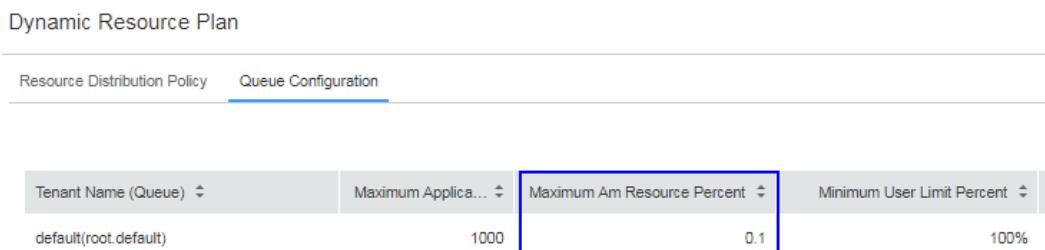
**Figure 3-1 Difference when SparkContext.stop() is added**

application_1417593322234_0019	root	YarnClientWithoutStop	SPARK	default	Wed, 3 Dec 2014 08:49:42 UTC	Wed, 3 Dec 2014 08:49:51 UTC	FINISHED	FAILED	
application_1417593322234_0018	root	YarnClientNormalStop	SPARK	default	Wed, 3 Dec 2014 08:48:59 UTC	Wed, 3 Dec 2014 08:49:12 UTC	FINISHED	SUCCEEDED	

## Appropriately plan the proportion of resources for AM

When there are many tasks and each task occupies few resources, the tasks may fail to start even if the cluster resources are sufficient and the tasks are submitted successfully. To address this issue, you can increase the value of **Max AM Resource Percent**.

**Figure 3-2 Modify Max AM Resource Percent**



## 3.18.3 Suggestions

### Persist the RDD that will be frequently used

The default RDD storage level is StorageLevel.NONE, which means that the RDD is not stored on disks or in memory. If an RDD is frequently used, persist the RDD as follows:

Call cache(), persist(), or persist(newLevel: StorageLevel) of spark.RDD to persist the RDD. The cache() and persist() functions set the RDD storage level to StorageLevel.MEMORY\_ONLY. The persist(newLevel: StorageLevel) function allows you to set other storage level for the RDD. However, before calling this function, ensure that the RDD storage level is StorageLevel.NONE or the same as the newLevel. That is, once the RDD storage level is set to a value other than StorageLevel.NONE, the storage level cannot be changed.

To unpersist an RDD, call unpersist(blocking: Boolean = true). The function can:

1. Remove the RDD from the persistence list. The corresponding RDD data becomes recyclable.
2. Set the storage level of the RDD to StorageLevel.NONE.

### Carefully select the the shuffle operator

This type of operator features wide dependency. That is, a partition of the parent RDD affects multiple partitions of the child RDD. The elements in an RDD are <key, value> pairs. During the execution process, the partitions of the RDD will be sequenced again. This operation is called shuffle.

Network transmission between nodes is involved in the shuffle operators. Therefore, for an RDD with large data volume, you are advised to extract

information as much as possible to minimize the size of each piece of data and then call the shuffle operators.

The following methods are often used:

- `combineByKey() : RDD[(K, V)] => RDD[(K, C)]`

This method is used to convert all the keys that have the same value in `RDD[(K, V)]` to a value with type of `C`.

- `groupByKey()` and `reduceByKey()` are two types of implementation of `combineByKey`. If `groupByKey` and `reduceByKey` cannot meet requirements in complex data aggregation, you can use customized aggregation functions as the parameters of `combineByKey`.

- `distinct(): RDD[T] => RDD[T]`

This method is used to remove repeated elements. The code is as follows:

```
map(x => (x, null)).reduceByKey((x, y) => x, numPartitions).map(_._1)
```

This process is time-consuming, especially when the data volume is high. Therefore, it is not recommended for the RDD generated from large files.

- `join() : (RDD[(K, V)], RDD[(K, W)]) => RDD[(K, (V, W))]`

This method is used to combine two RDDs through key.

If a key in `RDD[(K, V)]` has `X` values and the same key in `RDD[(K, W)]` has `Y` values, a total of  $(X * Y)$  data records will be generated in `RDD[(K, (V, W))]`.

## Use high-performance operators if the service permits

1. Using `reduceByKey/aggregateByKey` to replace `groupByKey`

The map-side pre-aggregation refers to that each local node performs the aggregation operation on the same key, which is similar to the local combiner in MapReduce. The map-side pre-aggregation ensures that each key on a node is unique. When a node is collecting the data of the same key in the processing results of the previous nodes, data that needs to be obtained will be significantly reduced, decreasing disk I/O and Internet transmission cost. Generally speaking, it is advised to replace `groupByKey` operator with `reduceByKey` or `aggregateByKey` operator if possible because they will pre-aggregate the local same key on each node by using user-defined functions. However, the `groupByKey` operator does not support pre-aggregation and delivers lower performance than `reduceByKey` or `aggregateByKey` because all data are distributed and transmitted on all the nodes.

2. Using `mapPartitions` to replace ordinary map operators

During a function invocation, `mapPartitions` operators will process all the data in a partition instead of only one piece of data, and therefore delivers higher performance than the ordinary map operators. However, `mapPartitions` may occasionally result in Out of Memory (OOM). If memory is insufficient, some objects cannot be recycled during memory recycling. Therefore, exercise caution when using `mapPartitions`.

3. Performing the coalesce operation after filtering

After filtering a large portion of data (for example, above 30%) by using the filter operator in an RDD, you are advised to manually decrease the number of partitions by using `coalesce` in order to compress the data in RDD to fewer partitions. This is because after filtering, much data in each partition is filtered out, leaving little data to be processed. If the computing is continued,

resources can be wasted. The task handling speed decreases as the number of tasks increases. Therefore, decreasing the number of partitions by using coalesce to compress the RDD data to fewer partitions can ensure that all the partitions are handled with fewer tasks. The performance can also be enhanced in some scenarios.

4. Using repartitionAndSortWithinPartitions to replace repartition and sort  
repartitionAndSortWithinPartitions is recommended by Spark official website. It is advised to use repartitionAndSortWithinPartitions for sorting after repartitioning. This operator can sort and shuffle repartitions at the same time, delivering higher performance.
5. Using foreachPartitions to replace foreach  
Similar to "Using mapPartitions to replace ordinary map operators", this mechanism handles all the data in a partition during a function invocation instead of one piece of data. In practice, foreachPartitions is proved to be helpful in improving performance. For example, the foreach function can be used to write all the data in RDD into MySQL. Ordinary foreach operators, write data piece by piece, and a database connection is established for each function invocation. Frequent connection establishments and destructions cause low performance. foreachPartitions, however, processes all the data in a partition at a time. Only one database connection is required for each partition. Batch insertion delivers higher performance.

## RDD Shared Variables

In application development, when a function is transferred to a Spark operation(such as map and reduce) and runs on a remote cluster, the operation is actually performed on the independent copies of all the variables involved in the function. These variables will be copied to each machine. In general, reading and writing shared variables across tasks is apparently inefficient. Spark provides two shared variables that are commonly used: broadcast variable and accumulator.

## Kryo can be used to optimize serialization performance in performance-demanding scenarios.

Spark offers two serializers:

**org.apache.spark.serializer.KryoSerializer**: high-performance but low compatibility

**org.apache.spark.serializer.JavaSerializer**: average performance and high compatibility

Method: conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")

### NOTE

The following are reasons why Spark does not use Kryo-based serialization by default:

Spark uses Java serialization by default, that is, uses the ObjectOutputStream and ObjectInputStream API to perform serialization and deserialization. Spark can also use Kryo serialization library, which delivers higher performance than Java serialization library. According to official statistics, Kryo-based serialization is 10 times more efficient than Java-based serialization. Kryo-based serialization requires the registration of all the user-defined types to be serialized, which is a burden for developers.

## Suggestions on Optimizing Spark Streaming Performance

1. Set an appropriate batch processing duration (batchDuration).
2. Set concurrent data receiving appropriately.
  - Set multiple receivers to receive data.
  - Set an appropriate receiver congestion duration.
3. Set concurrent data processing appropriately.
4. Use Kryo-based serialization.
5. Optimize memory.
  - Set the persistence level to reduce GC costs.
  - Use concurrent Mark Sweep GC algorithms to shorten GC pauses.

## 3.19 Yarn

### 3.19.1 Application Scenarios

YARN is a distributed resource management system that is used to improve resource usage in the distributed cluster environment. Resources include memory, I/O, network resources, and disk resources. YARN is developed to address the shortage of the original MapReduce framework. Initially, MapReduce's committers were able to modify the existing codes periodically. As codes increase and due to the problems in the design of the original MapReduce framework, the modification becomes increasingly difficult. Therefore, MapReduce's committers decides to redesign the framework with enhancements to scalability, availability, reliability, and compatibility, increasing the resource usage of next-generation MapReduce(MRv2/Yarn) and supporting more computing frameworks apart from MapReduce.

### 3.19.2 Rules

#### Use `YarnClient.createYarnClient()` to create a client

Do not directly use the protocol interface `ClientRMProxy.createRMProxy(config,ApplicationClientProtocol.class)` to create a client.

#### The Application Master uses the asynchronous interface `AMRMClientAsync.createAMRMClientAsync()` to interact with the ResourceManager

Do not directly use the protocol interface `ClientRMProxy.createRMProxy(config,ApplicationMasterProtocol.class)` to create a client used by the Application Master to interact with the ResourceManager.

## The Application Master uses the asynchronous interface **AMRMClientAsync.createAMRMClientAsync()** to interact with the NodeManager

Do not directly use the ContainerManagementProtocolProxy interface to create a client used by the Application Master to interact with the NodeManager.

### Multithread security login mode

If multiple threads are performing login operations, the relogin mode must be used for the subsequent logins of all threads after the first successful login of an application.

login example code:

```
private Boolean login(Configuration conf){  
    boolean flag = false;  
    UserGroupInformation.setConfiguration(conf);  
    try {  
        UserGroupInformation.loginUserFromKeytab(conf.get(PRINCIPAL), conf.get(KEYTAB));  
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "  
+UserGroupInformation.isLoginKeytabBased());  
        flag = true;  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    return flag;  
}
```

relogin example code:

```
public Boolean relogin(){  
    boolean flag = false;  
    try {  
        UserGroupInformation.getLoginUser().reloginFromKeytab();  
        System.out.println("UserGroupInformation.isLoginKeytabBased(): "  
+UserGroupInformation.isLoginKeytabBased());  
        flag = true;  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    return flag;  
}
```

# 4 Manager Management Development Guide

## 4.1 Overview

### 4.1.1 Application Development Overview

#### Intended Audience

This document is intended for users who need to access the FusionInsight Manager Rest APIs through HTTP basic authentication.

This document is intended for development personnel who are experienced in Java development.

#### Introduction to FusionInsight Manager

FusionInsight Manager, functioning as the O&M management system of cluster, provides services deployed in the cluster with a unified cluster management capability.

Manager provides functions such as installation and deployment, performance monitoring, alarms, user management, permission management, auditing, service management, health check and log collection.

### 4.1.2 Common Concepts

#### REST API

REST API is a set of APIs used to log in to the web server. REST APIs are executed through HTTP requests. Specifically, REST APIs receive the GET, PUT, POST, and DELETE requests and respond to the requests with JSON data.

The format of an HTTP request is as follows: `https://<Process IP>:<Process Port>/<Path>`

The *Process\_IP* indicates the IP address of the server node where the process resides, and *Process\_Port* indicates the process listening port.

For example, <https://10.162.181.57:32261/config>.

## Basic Authentication

In HTTP, basic authentication is designed to allow a web browser or other client programs to provide credentials in the form of a username and password when making a request.

Before a request is sent, a space is added by using Basic to identify basic authentication, and the username is appended with a colon and concatenated with the password. Then, the character string is encoded using the Base64 algorithm.

For example:

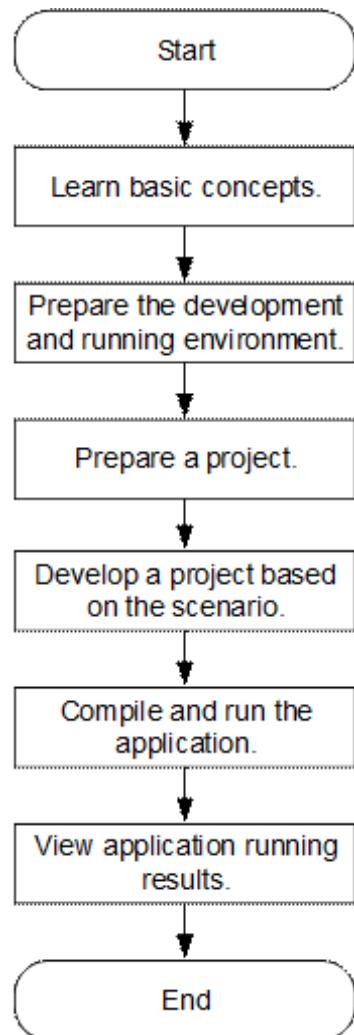
If the username is **admin** and the password is **Asd#smSisn\$123**, the combined character string is **admin:Asd#smSisn\$123**. After the character string is encoded using the Base64 algorithm, **YWRtaW46QWRtaW5AMTlz** is generated. Then the basic authentication flag is added to obtain **Basic YWRtaW46QWRtaW5AMTlz**. Finally, the encoded character string is sent out, and the receiver decodes it to obtain a character string of the username and password separated by a colon.

### 4.1.3 Development Process

This document describes FusionInsight Manager application development based on the Java API.

**Figure 4-1** shows and **Table 4-1** describes each phase of the development process.

**Figure 4-1** FusionInsight Manager application development process



**Table 4-1** Description of the FusionInsight Manager development process

Phase	Description	Reference
Learning basic concepts	Before application development, learn basic concepts of basic authentication, understand the scenario requirements, and design tables.	<a href="#">Common Concepts</a>
Preparing the development and running environment	The Java language is recommended for FusionInsight Manager REST API application development. The IntelliJ IDEA tool can be used.	<a href="#">Preparing Development and Running Environments</a>

Phase	Description	Reference
Preparing a project	FusionInsight Manager REST API provides example projects for different scenarios. You can import an example project to learn the application.	<a href="#">Configuring and Importing Sample Projects</a>
Developing a project based on the scenario	Provide a sample project based on the Java language, including typical scenarios such as adding users, searching for users, modifying users, deleting users, and exporting user lists.	<a href="#">Developing an Application</a>
Compiling and running the application	Compile the developed application and submit it for running.	<a href="#">Compiling and Running an Application</a>
Viewing application running results	Application running results are exported to a specified path. Users can also view the application running status on the UI.	<a href="#">Viewing Windows Commissioning Results</a>

## 4.2 Environment Preparation

### 4.2.1 Preparing Development and Running Environments

[Table 4-2](#) describes the development and running environment required for development.

**Table 4-2** Development and running environment

Item	Description
Operating system (OS)	Development environment: Windows OS. Windows 7 or later version is supported. The local development environment must interconnect with the cluster service-plane network.

Item	Description
JDK installation	<p>Basic configuration for the development and running environment. The JDK must meet the following requirements:</p> <p>The JDK version number must be the same as that used by the FusionInsight Manager to be accessed. For details about the version number, see the corresponding version document or contact the system administrator.</p> <p>For example, JDK of versions later than 1.8.x must be used if FusionInsight Manager 8.3.1 is used.</p>
IntelliJ IDEA installation and configuration	Tool used for developing HBase applications. The version must be <b>2019.1</b> or other compatible version.

## 4.2.2 Configuring and Importing Sample Projects

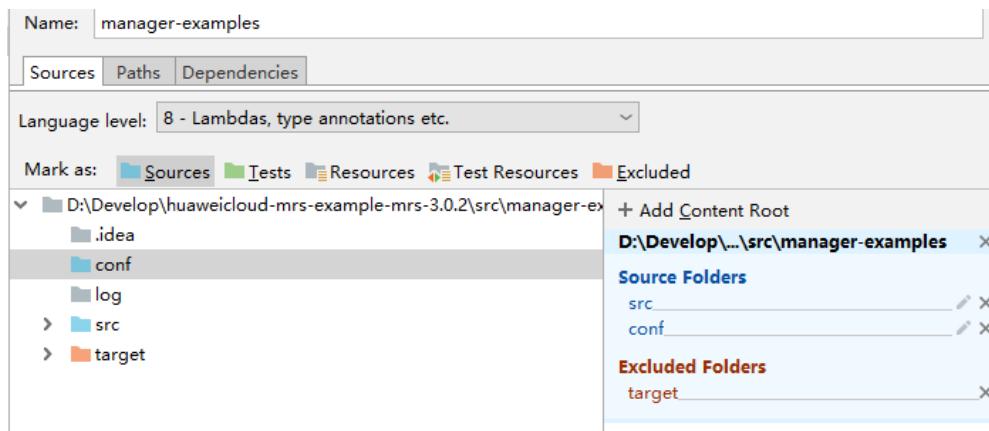
### Procedure

- Step 1** Obtain the sample project folder **manager-examples** in the **src** directory where the sample code is decompressed. For details, see [Obtaining Sample Projects from Huawei Mirrors](#).
- Step 2** In the application development environment, import the sample project to the IntelliJ IDEA development environment.
- Start IntelliJ IDEA, and then choose **File > New > Project from Existing Sources**. In the **Select File or Directory to Import** dialog box, select the sample code folder.
  - Select the **pom.xml** file from the sample project folder, choose **Import project from external model > Maven**, and click **Next** and then **Finish**.

#### NOTE

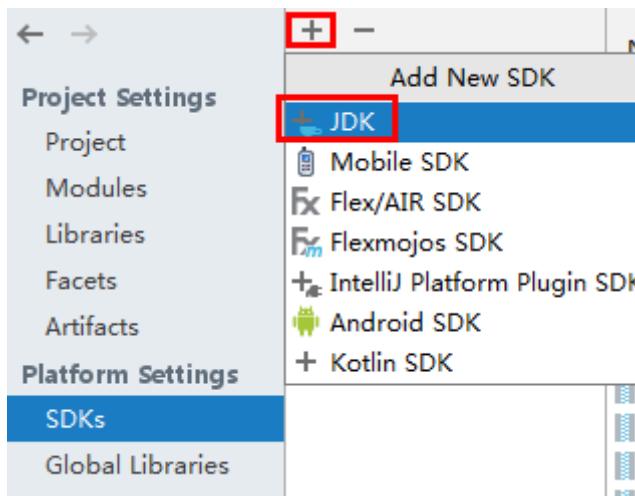
The sample code is a Maven project. You can adjust the project configuration based on the site requirements. The operations vary according to the IntelliJ IDEA version. The actual information displayed on the software interface prevails.

- Add the **src** and **conf** directories in the project to the source file path.  
After the project is imported, choose **File > Project Structure** on the menu bar of IntelliJ IDEA. In the displayed window, choose **Project Settings > Modules**.  
Select the current project, click the **src** folder, click **Sources** on the right of **Mark as**, click the **conf** folder, and click **Sources** on the right of **Mark as**. Click **Apply** and then **OK**.

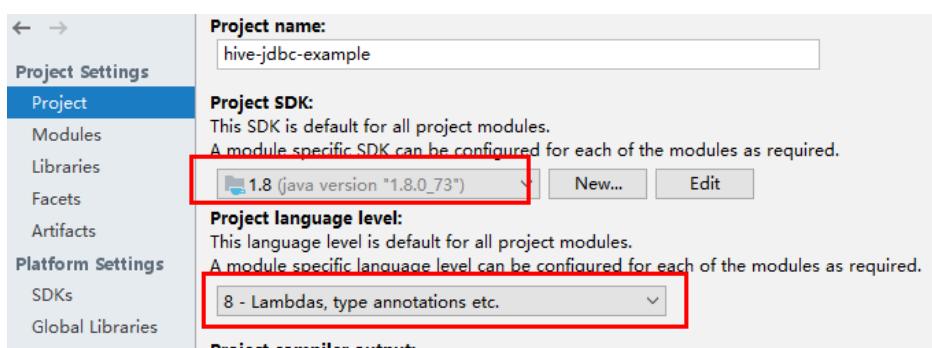


**Step 3** Set the JDK of the project.

1. On the IntelliJ IDEA menu bar, choose **File > Project Structure....** The **Project Structure** window is displayed.
2. Select **SDKs**, click the plus sign (+), and select **JDK**.

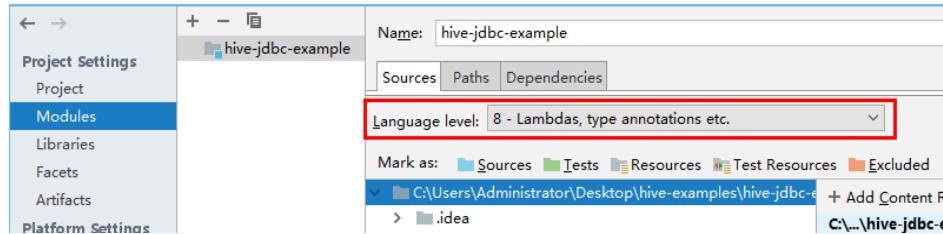


3. On the **Select Home Directory for JDK** page that is displayed, select the **JDK** directory and click **OK**.
4. After selecting the JDK, click **Apply**.
5. Select **Project**, select the JDK added in SDKs from the **Project SDK** drop-down list box, and select **8 - Lambdas, type annotations etc.** from the **Project language level** drop-down list box.

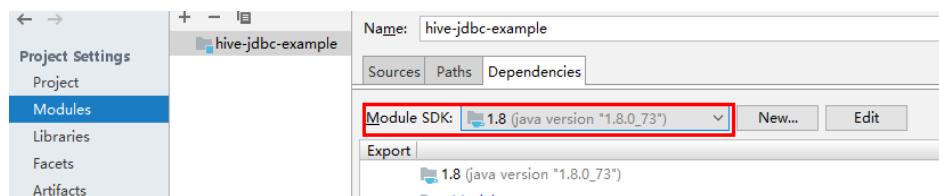


6. Click **Apply**.

7. Choose **Modules**. On the **Source** page, change the value of **Language level** to **8 - Lambdas, type annotations etc.**



On the **Dependencies** page, change the value of **Module SDK** to the JDK added in SDKs.

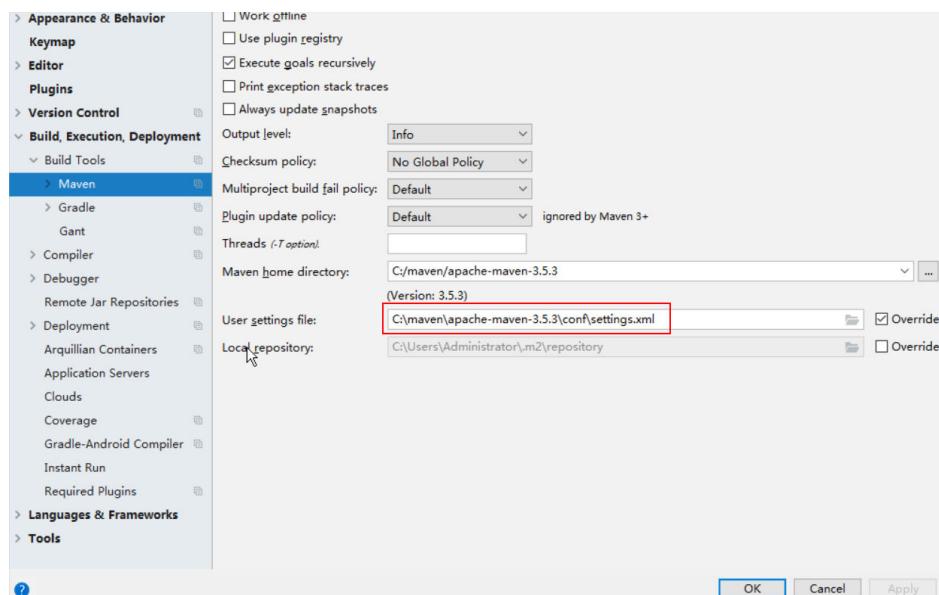


8. Click **Apply** and **OK**.

#### Step 4 Configure Maven.

1. Add the configuration information such as the address of the open-source image repository to the **setting.xml** configuration file of the local Maven by referring to [Configuring Huawei Open-Source Mirrors](#).
2. On the IntelliJ IDEA page, select **File > Settings > Build, Execution, Deployment > Build Tools > Maven**, select **Override** next to **User settings file**, and change the value of **User settings file** to the directory where the **settings.xml** file is stored. Ensure that the directory is **<Local Maven installation directory>\conf\settings.xml**.

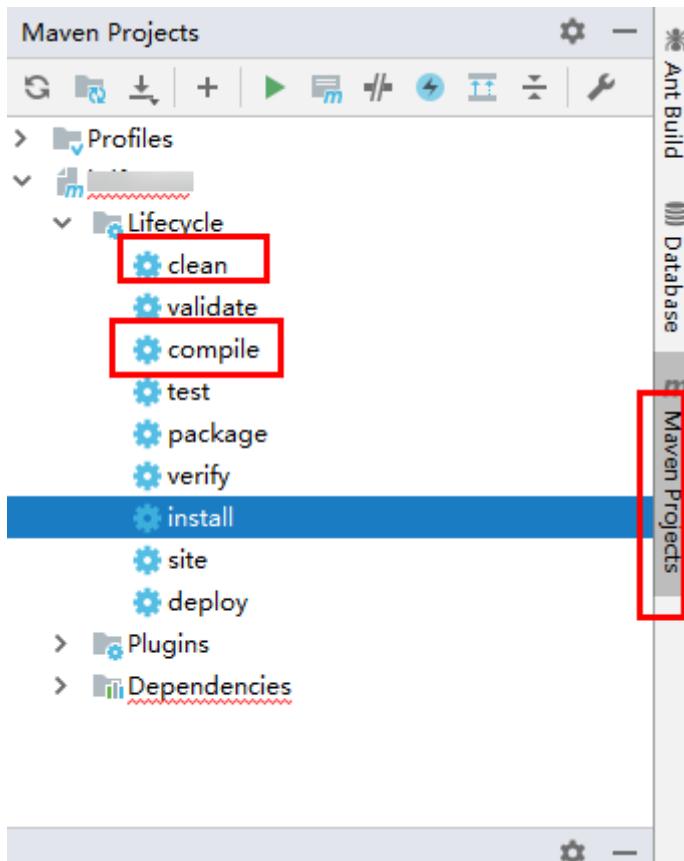
**Figure 4-2** Directory for storing the **settings.xml** file



3. Click the drop-down list next to **Maven home directory** and select the Maven installation directory.

4. Click **Apply**, and then click **OK**.
5. On the right of the IntelliJ IDEA home page, click **Maven Projects**. On the **Maven Projects** page, choose *Project name* > **Lifecycle** and run the **clean** and **compile** scripts.

Figure 4-3 Maven Projects page



----End

## 4.3 Developing an Application

### 4.3.1 Typical Scenario Description

This section describes the application development in a typical scenario, enabling users to quickly learn and master the FusionInsight Manager REST API development process and know key functions.

#### Typical Scenario

Users need to work with FusionInsight Manager in non-GUI mode. This way, an HTTP basic authentication-based application is required to provide the following functions:

- Login to FusionInsight Manager.

- Login to FusionInsight Manager for data querying, adding, or deleting.

## 4.3.2 Development Guideline

### Function Decomposition

Decompose the function based on the scenario, as listed in **Table 4-3**.

**Table 4-3** Functions to be developed in FusionInsight Manager

No.	Procedure	Code Implementation
1	Add users.	For details, see <a href="#">Adding Users</a> .
2	Search for users.	For details, see <a href="#">Searching for Users</a> .
3	Modify users.	For details, see <a href="#">Modifying Users</a> .
4	Delete users.	For details, see <a href="#">Deleting Users</a> .
5	Export a user list.	For details, see <a href="#">Exporting a User List</a> .

### Data Preparation

- Obtain the webUrl and the Url corresponding to the operation to be implemented from the REST API reference document of FusionInsight Manager.
- Obtain the JSON request body parameters and format required by the operation to be implemented from the REST API reference document of FusionInsight Manager, and create and save the corresponding JSON file.

## 4.3.3 Example Code Description

### 4.3.3.1 Login Authentication

#### Function

Implement basic authentication for login and return the HTTP client after login.

#### Prerequisites

User cluster information and login account have been configured.

- Configuration file: *Sample project folder\conf\UserInfo.properties*
- Parameters:
  - **userName**: username for logging in to FusionInsight Manager.
  - **password**: password of the username. Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.

- **webUrl:** address of the Manager home page.

In the following example, **IP\_Address** indicates the floating IP address of the cluster.

If you want to use another user to perform operations, you need log in to FusionInsight Manager to create a user.

```
userName= admin  
password= ExamplePassword  
webUrl= https://IP_Address:28443/web/  
#For ECS/BMS clusters , if the EIP is used, you can use the EIP to invoke the interface.  
#webUrl=https://eip:9022/mrsmanager/
```

## Example code

The following provides an example code of invoking the firstAccess interface to perform login authentication, which is included in the **main** method of the **rest.UserManager** class.

```
BasicAuthAccess authAccess = new BasicAuthAccess();  
HttpClient httpClient = authAccess.loginAndAccess(webUrl, userName, password, userTLSVersion);  
LOG.info("Start to access REST API.");  
HttpManager httpManager = new HttpManager();  
String operationName = "";  
String operationUrl = "";  
String jsonFilePath = "";
```

### 4.3.3.2 Adding Users

#### Function

Access the FusionInsight Manager interface to add users.

## Example code

The following provides an example code of adding users, which uses the **main** method of the **rest.UserManager** class.

#### NOTE

- Before running the code example, change **userName** in the **addUser** file to the actual user name and **password** to the corresponding password. After the code example is run, delete the user password in time.
- Passwords stored in plaintext pose security risks. Store them in ciphertext in configuration files or environment variables.

```
//Access the FusionInsight Manager interface to add users.  
operationName = "AddUser";  
operationUrl = webUrl + ADD_USER_URL;  
jsonFilePath = "./conf/addUser.json";  
httpManager.sendHttpPostRequest(httpClient, operationUrl, jsonFilePath, operationName)
```

### 4.3.3.3 Searching for Users

#### Function

Access the FusionInsight Manager interface to search for users.

## Example code

The following provides an example code of searching for users, which uses the **main** method of the **rest.UserManager** class.

```
//Access the FusionInsight Manager interface to search the user list.  
operationName = "QueryUserList";  
operationUrl = webUrl + QUERY_USER_LIST_URL;  
String responseLineContent = httpManager.sendHttpGetRequest(httpClient, operationUrl,  
operationName);  
LOG.info("The {} response is {}.", operationName, responseLineContent);
```

### 4.3.3.4 Modifying Users

#### Function

Access the FusionInsight Manager interface to modify users.

#### Example code

The following provides an example code of modifying users, which uses the **main** method of the **rest.UserManager** class.

```
//Access the FusionInsight Manager interface to modify users.  
operationName = "ModifyUser";  
String modifyUserName = "user888";  
operationUrl = webUrl + MODIFY_USER_URL + modifyUserName;  
jsonFilePath = "./conf/modifyUser.json";  
httpManager.sendHttpPutRequest(httpClient, operationUrl, jsonFilePath, operationName);
```

### 4.3.3.5 Deleting Users

#### Function

Access the FusionInsight Manager interface to delete users.

#### Example code

The following provides an example code of deleting users, which uses the **main** method of the **rest.UserManager** class.

```
//Access the FusionInsight Manager interface to delete users.  
operationName = "DeleteUser";  
String deleteJsonStr = "{\"userNames\":[\"user888\"]}";  
operationUrl = webUrl + DELETE_USER_URL;  
httpManager.sendHttpDeleteRequest(httpClient, operationUrl, deleteJsonStr, operationName);  
LOG.info("Exit main.");
```

### 4.3.3.6 Exporting a User List

#### Function

Invoke the export and download interfaces in sequence to export the user list. The output of the export interface is the input of the download interface.

## Example code

The following provides an example code of exporting a user list, which uses the **main** method of the **rest.ExportUsers** class.

```
String operationName = "ExportUsers";
String exportOperationUrl = webUrl + EXPORT_URL;
HttpManager httpManager = new HttpManager();
//Invoke the export interface.
String responseLineContent = httpManager
    .sendHttpPostRequestWithString(httpClient, exportOperationUrl, StringUtils.EMPTY, operationName);
//Invoke the download interface.
operationName = "DownloadUsers";
JSONObject jsonObj = JSON.parseObject(responseLineContent);
String fileName = jsonObj.getString("fileName");
String downloadOperationUrl = webUrl + DOWNLOAD_URL + fileName;
//Local path to which the file is downloaded. (The root directory of drive C is used as an example. Define
the root directory according to the actual situation.)
String downloadFilePath = "C:\\";
httpManager.sendDownloadRequest(httpClient, downloadOperationUrl, operationName,
    downloadFilePath + fileName);
```

## 4.4 Application Commissioning

### 4.4.1 Commissioning an Application in the Windows OS

#### 4.4.1.1 Compiling and Running an Application

##### Scenario

Run an application in the Windows OS after code development is complete.



##### NOTE

If the IBM JDK is used in the windows OS, applications cannot be directly run in the windows OS.

##### Prerequisites

Before commissioning the program, ensure that login authentication and user adding have been performed. For details, see [Login Authentication](#) and [Adding Users](#).

##### Procedure

In a development environment (such as IntelliJ IDEA), choose the following two projects separately and run the projects:

- Choose **UserManager.java** and right-click the project and choose **Run 'UserManager.main()'** from the shortcut menu to run the project.
- Choose **ExportUser.java** and right-click the project and choose **Run 'ExportUser.main()'** from the shortcut menu to run the project.

#### 4.4.1.2 Viewing Windows Commissioning Results

##### Scenario

After a FusionInsight Manager application is run, you can check the running result using either of the following methods:

- View the IntelliJ IDEA running result. Configure log printing information in the **log4j.properties** file in the **/conf** directory.
- Log in to the management node and view the system log **web.log** in the **/var/log/Bigdata/tomcat** directory.

##### Procedure

- Run the **UserManager** class. If the following log information is displayed, the operation is successful.

```
2020-10-19 14:22:52,111 INFO [main] Enter main. rest.UserManager.main(UserManager.java:43)
2020-10-19 14:22:52,113 INFO [main] Get the web info and user info from file .\conf
UserInfo.properties rest.UserManager.main(UserManager.java:56)
2020-10-19 14:22:52,113 INFO [main] The user name is : admin.
rest.UserManager.main(UserManager.java:63)
2020-10-19 14:22:52,113 INFO [main] The webUrl is : https://10.112.16.93:28443/web/.
rest.UserManager.main(UserManager.java:75)
2020-10-19 14:22:52,113 INFO [main] Begin to get httpclient and first access.
rest.UserManager.main(UserManager.java:84)
2020-10-19 14:22:52,117 INFO [main] Enter loginAndAccess.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:56)
2020-10-19 14:22:52,120 INFO [main] 1.Get http client for sending https request, username is admin,
webUrl is https://10.112.16.93:28443/web/.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:66)
2020-10-19 14:22:52,121 INFO [main] Enter getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:98)
2020-10-19 14:22:52,693 INFO [main] Exit getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:104)
2020-10-19 14:22:52,693 INFO [main] The new http client is:
org.apache.http.impl.client.DefaultHttpClient@66d2e7d9.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:70)
2020-10-19 14:22:52,693 INFO [main] 2.Construct basic authentication,username is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:76)
2020-10-19 14:22:52,694 INFO [main] the authentication is Basic YWRtaW46QmlnZGF0YV8yMDEz .
basicAuth.BasicAuthAccess.constructAuthentication(BasicAuthAccess.java:122)
2020-10-19 14:22:52,695 INFO [main] 3. Send first access request, usename is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:86)
2020-10-19 14:22:53,555 INFO [main] First access status is HTTP/1.1 200
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:162)
2020-10-19 14:22:53,556 INFO [main] Response content is
[{"infoType":1,"infoContent":"","alarmStat":[{"level":"Critical","num":0},{"level":"Major","num":6},
{"level":"Minor","num":0}, {"level":"Warning","num":0}],"timestamp":1603088492362,"timezoneOffset":-480}]
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:168)
2020-10-19 14:22:53,556 INFO [main] User admin first access success
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:174)
2020-10-19 14:22:53,556 INFO [main] Start to access REST API.
rest.UserManager.main(UserManager.java:88)
2020-10-19 14:22:53,557 INFO [main] Enter sendHttpPostRequest for userOperation AddUser.
basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:93)
2020-10-19 14:22:53,558 INFO [main] The json content =
{"userName":"user888","userType":"HM","password":"XXX","confirmPassword":"XXX","userGroups": ["supergroup"],"userRoles":[],"primaryGroup":"supergroup","description":"Add user"}.
basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:148)
2020-10-19 14:22:55,437 INFO [main] The AddUser status is HTTP/1.1 204 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:55,437 INFO [main] sendHttpPostRequest completely.
basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:178)
2020-10-19 14:22:55,437 INFO [main] Enter sendHttpGetRequest for userOperation QueryUserList.
```

```
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:48)
2020-10-19 14:22:55,437 INFO [main] The operationUrl is:https://10.112.16.93:28443/web/api/v2/
permission/users?limit=10&offset=0&filter=&order=ASC&order_by=userName
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:60)
2020-10-19 14:22:55,565 INFO [main] The QueryUserList status is HTTP/1.1 200 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:55,565 INFO [main] The response lineContent is {"users":
[{"userName":"admin","userType":"HM","description":"Administrator of FusionInsight
Manager.","password":"","createTime":"2020-09-30T10:31:44+08:00","defaultUser":true,"primaryGroup
":"compcommon","locked":false,"userRoles":["Manager_administrator"],"userGroups":
[],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":"MRS_MANAG
ER_USER","iamCustomPolicyUser":false},
 {"userName":"developuser","userType":"MM","description":"","password":"","createTime":"2020-10-15
T19:16:37+08:00","defaultUser":false,"primaryGroup":"elasticsearch","locked":false,"userRoles":
["System_administrator"],"userGroups":
["elasticsearch"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false},
 {"userName":"hue1","userType":"HM","description":"","password":"","createTime":"2020-10-09T17:39:5
7+08:00","defaultUser":false,"primaryGroup":"hive","locked":false,"userRoles":
["System_administrator"],"userGroups":
["hive","hadoop","supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO",
"userSource":"MRS_MANAGER_USER","iamCustomPolicyUser":false},
 {"userName":"user888","userType":"HM","description":"Add
user","password":"","createTime":"2020-10-19T14:21:32+08:00","defaultUser":false,"primaryGroup":"su
pergroup","locked":false,"userRoles":[],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false},
 {"userName":"yangtong","userType":"MM","description":"","password":"","createTime":"2020-10-19T10
:50:52+08:00","defaultUser":false,"primaryGroup":"supergroup","locked":false,"userRoles":
["System_administrator"],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false}],
 "totalCount":5}.
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:430)
2020-10-19 14:22:55,565 INFO [main] SendHttpGetRequest completely.
basicAuth.HttpManager.sendHttpGetRequest(HttpManager.java:69)
2020-10-19 14:22:55,565 INFO [main] The QueryUserList response is {"users":
[{"userName":"admin","userType":"HM","description":"Administrator of FusionInsight
Manager.","password":"","createTime":"2020-09-30T10:31:44+08:00","defaultUser":true,"primaryGroup
":"compcommon","locked":false,"userRoles":["Manager_administrator"],"userGroups":
[],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":"MRS_MANAG
ER_USER","iamCustomPolicyUser":false},
 {"userName":"developuser","userType":"MM","description":"","password":"","createTime":"2020-10-15
T19:16:37+08:00","defaultUser":false,"primaryGroup":"elasticsearch","locked":false,"userRoles":
["System_administrator"],"userGroups":
["elasticsearch"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false},
 {"userName":"hue1","userType":"HM","description":"","password":"","createTime":"2020-10-09T17:39:5
7+08:00","defaultUser":false,"primaryGroup":"hive","locked":false,"userRoles":
["System_administrator"],"userGroups":
["hive","hadoop","supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO",
"userSource":"MRS_MANAGER_USER","iamCustomPolicyUser":false},
 {"userName":"user888","userType":"HM","description":"Add
user","password":"","createTime":"2020-10-19T14:21:32+08:00","defaultUser":false,"primaryGroup":"su
pergroup","locked":false,"userRoles":[],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false},
 {"userName":"yangtong","userType":"MM","description":"","password":"","createTime":"2020-10-19T10
:50:52+08:00","defaultUser":false,"primaryGroup":"supergroup","locked":false,"userRoles":
["System_administrator"],"userGroups":
["supergroup"],"indepdtType":"NONE","domainUser":false,"synchroStatus":"SYNCHRO","userSource":
"MRS_MANAGER_USER","iamCustomPolicyUser":false}],
 "totalCount":5}.
rest.UserManager.main(UserManager.java:105)
2020-10-19 14:22:55,565 INFO [main] Enter sendHttpPutRequest for userOperation ModifyUser.
basicAuth.HttpManager.sendHttpPutRequest(HttpManager.java:239)
2020-10-19 14:22:55,566 INFO [main] The json content =
{"userName":"user888","userType":"HM","password":"XXX","confirmPassword":"XXX","userGroups":
["supergroup"],"primaryGroup":"supergroup","userRoles":
["Manager_administrator"],"description":"Modify user"}.
basicAuth.HttpManager.sendHttpPutRequest(HttpManager.java:293)
```

```
2020-10-19 14:22:56,299 INFO [main] The ModifyUser status is HTTP/1.1 204 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:56,299 INFO [main] sendHttpPutRequest completely.
basicAuth.HttpManager.sendHttpPutRequest(HttpManager.java:304)
2020-10-19 14:22:56,299 INFO [main] The operationUrl is:https://10.112.16.93:28443/web/api/v2/
permission/users basicAuth.HttpManager.sendHttpDeleteRequest(HttpManager.java:389)
2020-10-19 14:22:56,299 INFO [main] Enter sendHttpDeleteMessage for operation DeleteUser.
basicAuth.HttpManager.sendHttpDeleteRequest(HttpManager.java:390)
2020-10-19 14:22:57,463 INFO [main] The DeleteUser status is HTTP/1.1 204 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:425)
2020-10-19 14:22:57,463 INFO [main] sendHttpDeleteMessage for DeleteUser completely.
basicAuth.HttpManager.sendHttpDeleteRequest(HttpManager.java:406)
2020-10-19 14:22:57,463 INFO [main] Exit main. rest.UserManager.main(UserManager.java:120)
```

#### NOTE

As indicated in the log, when the main method of the UserManager class is executed, the loginAndAccess, sendHttpPostRequest, sendHttpGetRequest, sendHttpPutRequest, and sendHttpDeleteRequest methods are invoked in sequence to send the POST, GET, PUT, and DELETE requests for login authentication, adding users, searching for users, modifying users, and deleting users.

- Run the **ExportUsers** class. If the following log information is displayed, the operation is successful.

```
2023-07-11 20:12:41,172 INFO [main] Enter main. rest.ExportUsers.main(ExportUsers.java:33)
2023-07-11 20:12:41,176 INFO [main] The user name is : admin.
utils.PropertiesUtil.getUserName(PropertiesUtil.java:75)
2023-07-11 20:12:41,176 INFO [main] The webUrl is : https://192.168.20.60:28443/web/.
utils.PropertiesUtil.getWebUrl(PropertiesUtil.java:96)
2023-07-11 20:12:41,177 INFO [main] Begin to get httpclient and first access.
rest.ExportUsers.main(ExportUsers.java:59)
2023-07-11 20:12:41,183 INFO [main] Enter loginAndAccess.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:57)
2023-07-11 20:12:41,192 INFO [main] 1.Get http client for sending https request, username is admin,
webUrl is https://192.168.20.60:28443/web/.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:67)
2023-07-11 20:12:41,192 INFO [main] Enter getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:99)
2023-07-11 20:12:41,719 INFO [main] Exit getHttpClient.
basicAuth.BasicAuthAccess.getHttpClient(BasicAuthAccess.java:105)
2023-07-11 20:12:41,719 INFO [main] The new http client is:
org.apache.http.impl.client.DefaultHttpClient@33e5ccce.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:71)
2023-07-11 20:12:41,719 INFO [main] 2.Construct basic authentication,username is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:77)
2023-07-11 20:12:41,719 INFO [main] 3. Send first access request, usename is admin.
basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:87)
2023-07-11 20:12:42,909 INFO [main] First access status is 200
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:159)
2023-07-11 20:12:42,910 INFO [main] Response content is
[{"infoType":1,"infoContent":"","alarmStat":[{"level":"Critical","num":0},{"level":"Major","num":5},
{"level":"Minor","num":0},
{"level":"Warning","num":1}],"timestamp":1689077219149,"timezoneOffset":-480}]
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:165)
2023-07-11 20:12:42,910 INFO [main] User admin first access success
basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:171)
2023-07-11 20:12:42,910 INFO [main] Start to access REST API.
rest.ExportUsers.main(ExportUsers.java:63)
2023-07-11 20:12:42,913 INFO [main] Enter sendHttpPostRequest for userOperation ExportUsers.
basicAuth.HttpManager.sendHttpPostRequestWithString(HttpManager.java:245)
2023-07-11 20:12:43,402 INFO [main] The ExportUsers status is HTTP/1.1 200 .
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:477)
2023-07-11 20:12:43,402 INFO [main] The response lineContent is
{"fileName":"userInfo_2023-07-11-20-06-59.zip"}.
basicAuth.HttpManager.handleHttpResponse(HttpManager.java:482)
2023-07-11 20:12:43,402 INFO [main] Send HttpPostRequest completely.
basicAuth.HttpManager.sendHttpPostRequestWithString(HttpManager.java:268)
2023-07-11 20:12:43,468 INFO [main] Enter sendDownloadRequest for userOperation
DownloadUsers. basicAuth.HttpManager.sendDownloadRequest(HttpManager.java:97)
```

```
2023-07-11 20:12:43,468 INFO [main] The operationUrl is:https://192.168.20.60:28443/web/api/v2/permission/users/download?file_name=userInfo_2023-07-11-20-06-59.zip  
basicAuth.HttpManager.sendDownloadRequest(HttpManager.java:114)  
2023-07-11 20:12:43,651 INFO [main] The DownloadUsers status is HTTP/1.1 200 .  
basicAuth.HttpManager.handleDownloadResponse(HttpManager.java:513)  
2023-07-11 20:12:43,653 INFO [main] File C:\userInfo_2023-07-11-20-06-59.zip download successful.  
basicAuth.HttpManager.handleDownloadResponse(HttpManager.java:532)  
2023-07-11 20:12:43,653 INFO [main] SendDownloadRequest completely.  
basicAuth.HttpManager.sendDownloadRequest(HttpManager.java:123)  
2023-07-11 20:12:43,654 INFO [main] Exit main. rest.ExportUsers.main(ExportUsers.java:82)
```

#### NOTE

As indicated in the log, when the main method of the ExportUsers class is executed, the loginAndAccess, sendHttpPostRequestWithString, and sendHttpGetRequest methods are invoked in sequence to send the POST and GET requests for login authentication, exporting users, and downloading users.

- You can configure the log printing information in the **conf\log4j.properties** file to view the application running process and result.

The printing information is configured by default. The following provides an example.

```
##set log4j DEBUG < INFO < WARN < ERROR < FATAL  
log4j.logger.rest=INFO,A1,A2  
log4j.logger.basicAuth=INFO,A1,A2  
log4j.logger.org.apache.http=INFO,A1,A2  
  
#print to the console?A1  
log4j.appendер.A1=org.apache.log4j.ConsoleAppender  
log4j.appendер.A1.layout=org.apache.log4j.PatternLayout  
log4j.appendер.A1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%t] %m %l%n  
  
#log file  
log4j.appendер.A2=org.apache.log4j.DailyRollingFileAppender  
log4j.appendер.A2.File=../log/rest.log  
log4j.appendер.A2.Append = true  
log4j.appendер.A2.layout=org.apache.log4j.PatternLayout  
log4j.appendер.A2.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%t] %m %l%n
```

## 4.5 More Information

### 4.5.1 External Interfaces

#### 4.5.1.1 Java API

For details about FusionInsight Manager APIs, see *MapReduce Service (MRS) 3.3.1-LTS Manager Rest API Reference (for Huawei Cloud Stack 8.3.1)*.

#### Typical APIs

**Table 4-4** describes the typical methods of developing FusionInsight Manager REST APIs.

**Table 4-4** restApiDemo/src/rest.BasicAuthAcces

Method	Description
loginAndAccess (String webUrl, String userName, String password, String userTLSVersion)	<ul style="list-style-type: none"><li>• webUrl URL of the cluster home page, which is obtained from the <b>UserInfo.properties</b> configuration file.</li><li>• userName Username for logging in to the FusionInsight system, which is obtained from the <b>UserInfo.properties</b> configuration file.</li><li>• password Password of the account userName, which is obtained from the <b>UserInfo.properties</b> configuration file.</li><li>• userTLSVersion TSL version.</li></ul> <p>Return type: HttpClient Return: HttpClient</p>

 NOTE

- The Java API implements basic authentication for login and returns the HttpClient after login. This way, only one API is invoked during the login, simplifying the usage process.
- The API input parameters are obtained from the configuration file **UserInfo.properties**. Users need to set the parameters in the file. The Java API also invokes multiple methods of the BasicAuthAccess class.

**Table 4-5** restApiDemo/src/rest.BasicAuthAcces.HttpManager

Method	Description
sendHttpGetRequest(HttpClient httpClient, String operationUrl, String operationName)	<ul style="list-style-type: none"><li>• HttpClient httpClient: result returned after the login authentication is complete.</li><li>• operationUrl URL of the httpGet operation.</li><li>• operationName The operation name.</li></ul>
sendHttpPostRequest(HttpClient httpClient, String operationUrl, String jsonFilePath, String operationName)	<ul style="list-style-type: none"><li>• HttpClient httpClient: result returned after the login authentication is complete.</li><li>• operationUrl URL of the httpPost operation.</li><li>• jsonFilePath JSON file corresponding to the httpPost operation.</li><li>• operationName The operation name.</li></ul>

Method	Description
sendHttpPostRequestWithString(HttpClient httpClient, String operationUrl, String jsonString, String operationName)	<ul style="list-style-type: none"><li>• HttpClient httpClient: result returned after the login authentication is complete.</li><li>• operationUrl URL of the httpPost operation.</li><li>• jsonString JSON string format corresponding to the httpPost operation.</li><li>• operationName The operation name.</li></ul>
sendHttpPutRequest(HttpClient httpClient, String operationUrl, String jsonFilePath, String operationName)	<ul style="list-style-type: none"><li>• HttpClient httpClient: result returned after the login authentication is complete.</li><li>• operationUrl URL of the httpPut operation.</li><li>• jsonFilePath JSON file corresponding to the httpPut operation.</li><li>• operationName The operation name.</li></ul>
sendHttpPutRequestWithString(HttpClient httpClient, String operationUrl, String jsonString, String operationName)	<ul style="list-style-type: none"><li>• HttpClient httpClient: result returned after the login authentication is complete.</li><li>• operationUrl URL of the httpPut operation.</li><li>• jsonString JSON string format corresponding to the httpPut operation.</li><li>• operationName The operation name.</li></ul>
sendHttpDeleteRequest(HttpClient httpClient, String operationUrl, String jsonString, String operationName)	<ul style="list-style-type: none"><li>• HttpClient httpClient: result returned after the login authentication is complete.</li><li>• operationUrl URL of the httpDelete operation.</li><li>• jsonString JSON string format corresponding to the httpDelete operation.</li><li>• operationName The operation name.</li></ul>

Method	Description
sendDownloadRequest(HttpClient httpClient, String operationUrl, String operationName, String fileName)	<ul style="list-style-type: none"><li>• HttpClient httpClient: result returned after the login authentication is complete.</li><li>• operationUrl URL of the httpDelete operation.</li><li>• jsonString JSON string format corresponding to the httpDelete operation.</li><li>• operationName The operation name.</li><li>• fileName Name of the downloaded file.</li></ul>

#### NOTE

- APIs in the table are used to send HTTP requests. To invoke the APIs, users only need to provide the URLs corresponding to each type of operations, and JSON files corresponding to the operations or JSON string formats. In this way, no intermediate execution code is required, reducing the code compilation workload and simplifying operation procedures.
- The APIs return the command ID corresponding to the requests. Based on the command IDs, users can query the command execution progress.

## 4.5.2 FAQ

### 4.5.2.1 JDK1.6 Fails to Connect to the FusionInsight System Using JDK1.8

#### Question

The MRS system uses JDK1.8 and supports TLSv1.2 and TLSv1.2. Some users can only use JDK1.6 to connect to the MRS system, and the connection fails.

#### Answer

Solution: Add an SSL of an earlier version.

**Step 1** Add an SSL of an earlier version, for example, TLSv1, to the **server.xml** file in the **\${BIGDATA\_HOME}/om-server/tomcat/conf** directory. In the file, there are two contexts to be modified.

First context:

```
<Connector port="20009" address="10.52.0.35"  
protocol="com.omm.huawei.tomcat.http11.Http11Protocol" SSLEnabled="true"  
maxHttpHeaderSize="8192" maxPostSize="10240"  
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"  
enableLookups="false" disableUploadTimeout="true"  
acceptCount="100" scheme="https" secure="true"  
clientAuth="false" sslProtocol="TLS" sslEnabledProtocols = "TLSv1.1,TLSv1.2,TLSv1"
```

Second context:

```
<Connector port="28443" address="10.52.0.35"
protocol="com.omm.huawei.tomcat.http11.Http11Protocol" SSLEnabled="true"
scheme="https"
secure="true"
maxThreads="500"
acceptCount="500"
connectionTimeout="20000"
maxPostSize="10240"
clientAuth="false"
allowTrace="false"
sslProtocol="TLS" sslEnabledProtocols = "TLSv1.1,TLSv1.2,TLSv1"
```

- Step 2** Set the **userTLSVersion** parameter in the main method of the commissioning source file to be the same as that added in the **server.xml** file, for example, TLSv1.

```
String webUrl = resourceBundle .getString("webUrl");
LOG .info("The webUrl is:{}" ,webUrl);
if(password == null || password .isEmpty())
{
    LOG .error("The password is empty.");
}
//userTLSVersion is mandatory, which is an important parameter used for connecting the JDK1.6
server and the JDK1.8 server. If the JDK1.8 server is used, set this parameter to a null character string.
String userTLSVersion = "TLSv1" ;
//Invoke the firstAccess interface to perform the login authentication.
LOG .info("Begin to get httpclient and access.");
BasicAuthAccess authAccess = new BasicAuthAccess();
HttpClient httpclient = authAccess .firstAccess(webUrl , userName , password , userTLSVersion);
```

- Step 3** The other steps are the same as those for the JDK1.8 server. For details, see [Example Code Description](#).



#### NOTE

This solution can resolve the connection problem. However, the SSL of earlier versions may cause security risks to the system. Remind users to use JDK1.8.

----End

### 4.5.2.2 Authentication Fails, Error Code "401" Is Returned, and Logs Displayed as "Authorize Failed"

#### Question

An application execution fails, the error code 401 is returned. [Authentication Fails, Error Code "401" Is Returned, and Logs Displayed as "Authorize Failed"](#) shows the log information.

**Figure 4-4** Log indicating operation failure

```
2023-07-11 20:18:57,258 INFO [main] 3. Send first access request, username is admin. basicAuth.BasicAuthAccess.loginAndAccess(BasicAuthAccess.java:87)
2023-07-11 20:18:58,106 INFO [main] First access status is 401 basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:159)
2023-07-11 20:18:58,106 INFO [main] Response content is {"state":"FAILED", "errorCode":401,"errorDescription":"AUTH_FAILED","totalProgress":0.0,"id":-1} basicAuth.BasicAuthAccess.firstAccessResp(BasicAuthAccess.java:165)
2023-07-11 20:18:58,107 ERROR [main] Export Users running exception:Authorize failed! rest.ExportUsers.main(ExportUsers.java:85)
```

#### Answer

The mapping between error description and failure cause is as follows. Rectify the fault based on the cause.

Error Description	Failure Causes and Handling Methods
USER_FIRST_LOGIN	This is the first login. You need to change the password before logging in to the system.
PASSWORD_EXPIRED	The current user password has expired. You need to change the password before logging in to the system.
LOCKED_OUT	The user is locked. Contact the user administrator to unlock the user and log in again.
AUTH_FAILED	Authentication failed. Check whether the values of <b>username</b> and <b>password</b> in the <b>UserInfo.properties</b> file are correct.

#### 4.5.2.3 An Operation Fails and "log4j:WARN No appenders could be found for logger(basicAuth.Main)" Is Displayed in Logs

##### Question

An application execution fails. Figure 1-6 shows the log information.

**Figure 4-5** Log indicating operation failure

```
log4j:WARN No appenders could be found for logger (rest.UserManager).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

##### Answer

Check whether the compiled **log4j.properties** file exists in the **bin** directory of the project. If it does not, add the compilation path.

**Step 1** In IntelliJ IDEA, choose **File > Project Structure > Modules**, and then add the **conf** folder that contains **log4j.properties** and **UserInfo.properties** as **Source Folders**.

**Step 2** Compile the file again.

----End

#### 4.5.2.4 An Operation Fails and "illegal character in path at index 57" Is Displayed in Logs

##### Question

An application execution fails. **Figure 4-6** shows the log information.

**Figure 4-6** Log indicating operation failure

```
java.lang.IllegalArgumentException: Illegal character in path at index 57: https://189.120.205.139:28443/web/api/v2/permission/users
at java.net.URI.create(Unknown Source)
at org.apache.http.client.methods.HttpPost.<init>(HttpPost.java:76)
at basicAuth.HttpManager.sendHttpPostRequest(HttpManager.java:156)
at rest.UserManager.main(UserManager.java:101)
Caused by: java.net.URISyntaxException: Illegal character in path at index 57: https://189.120.205.139:28443/web/api/v2/permission/users
at java.net.URI$Parser.fail(Unknown Source)
at java.net.URI$Parser.checkChars(Unknown Source)
at java.net.URI$Parser.parseHierarchical(Unknown Source)
at java.net.URI$Parser.parse(Unknown Source)
at java.net.URI.<init>(Unknown Source)
... 4 more
```

## Answer

The URL may contain spaces. As a result, the server cannot identify the URL.

Delete the spaces in the URL.

### 4.5.2.5 Run the curl Command to Access REST APIs

#### Description

You can use the curl tool to access the Manager REST API. Before running the curl command, run the **openssl version** command to check the OpenSSL version of the current operating system. If the version is earlier than OpenSSL 1.0.1, install the OpenSSL of a later version for the operating system. to support interaction with the cluster using TLSv1.1 and TLSv1.2.

For details about how to install and use curl, see related official documents.

#### Operation Example

- Send the GET method to access the user list query interface and save cookie to the `/tmp/jsessionId.txt` file.

```
curl -k -i --basic -u <user name><password> -c /tmp/jsessionId.txt 'https://Manager floating IP address:28443/web/api/v2/permission/users?limit=10&offset=0&filter=&order=ASC&order_by=userName'
```

##### NOTE

`<user name>`: `<password>` indicates the user name and password that have the Manager operation rights.

- Use cookie to access the user group interface to add, modify, and delete user groups.

- Send the POST method to add a user.

```
curl -k -i -b /tmp/jsessionId.txt -X POST -HContent-type:application/json -d '{"userName":"user888","userType":"HM","password":"xxx","confirmPassword":"xxx","userGroups":["supergroup"],"userRoles":[],"primaryGroup":"supergroup","description":"Add user"}' 'https://Manager floating IP address:28443/web/api/v2/permission/users'
```

- Send the PUT method to modify the user information.

```
curl -k -i -b /tmp/jsessionId.txt -X PUT -HContent-type:application/json -d '{"userName":"user888","userType":"HM","password":"","confirmPassword":"","userGroups":["supergroup","hadoopmanager"],"primaryGroup":"supergroup","userRoles":[],"description":"Modify user"}' 'https://Manager floating IP address:28443/web/api/v2/permission/users/user888'
```

- Send a DELETE method to delete a user.

```
curl -k -i -b /tmp/jsessionId.txt -X DELETE -HContent-type:application/json -d '{"userNames":["user888"]}' 'https://Manager floating IP address:28443/web/api/v2/permission/users'
```

3. For the download type interface

- Exporting users

```
curl -k -i -b /tmp/jsessionId.txt -X POST 'https://Manager floating IP  
address:28443/web/api/v2/permission/users/operations/export?  
filter=&format=txt'
```

After the command is executed, a user file is returned, for example,  
**userInfo\_2023-05-10-11-08-07.zip**.

- Download user

```
curl -k -i -b /tmp/jsessionId.txt 'https://Manager floating IP  
address:28443/web/api/v2/permission/users/download?  
file_name='Name of the exported user file' --output 'Name of the  
exported user file'
```

**Name of the exported user file** is the name of the file returned after  
the export user command is executed, for example,  
**userInfo\_2023-05-10-11-08-07.zip**.

 NOTE

For ECS/BMS clusters, if the EIP is used, the EIP can be invoked. In this case, you need to  
replace the following parts of the command:

- Replace *Manager floating IP address:28443* with *EIP address:9022*.
- Replace **web** with **/mrsmanager**.

For example, to add a user by using the POST method, run the following command:

```
curl -k -i -b /tmp/jsessionId.txt -X POST -HContent-type:application/json -d  
'{"userName":"user888","userType":"HM","password":"xxx","confirmPassword":"xxx","u  
serGroups":["supergroup"],"userRoles":  
[],"primaryGroup":"supergroup","description":"Add user"}' 'https://EIP address:9022/  
mrsmanager/api/v2/permission/users'
```

MRS 3.2.0 and later versions does not provide the **gateway/mrs/mrsmanager** interface (in  
versions earlier than mrs 3.x). Replace it with **/mrsmanager**.

# 5 Appendix

## 5.1 Accessing FusionInsight Manager of an MRS Cluster

### Scenario

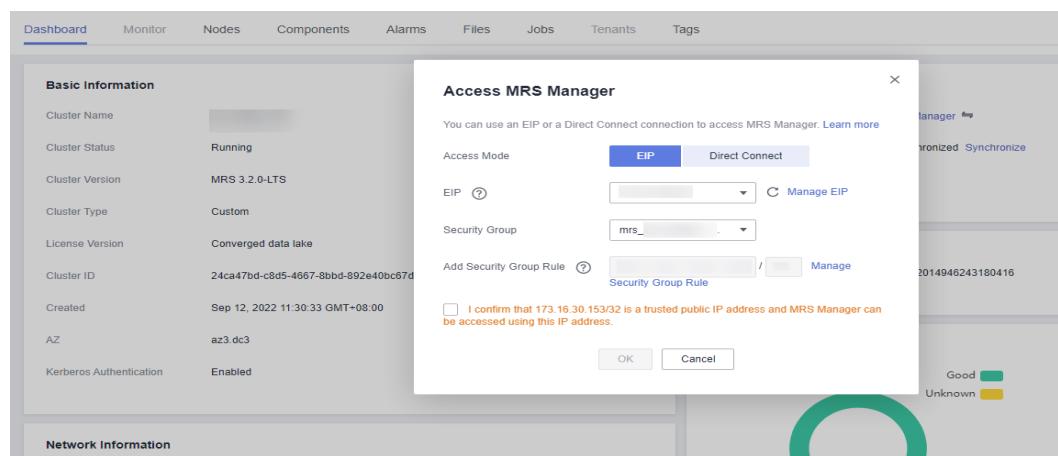
FusionInsight Manager is used to monitor, configure, and manage clusters. After a cluster is installed, you can use an account to log in to FusionInsight Manager.

For ECS or BMS clusters, you can access FusionInsight Manager in the following methods:

- [Accessing FusionInsight Manager Using an EIP](#)
- [Accessing FusionInsight Manager Using a Direct Connect Connection](#)
- [Accessing FusionInsight Manager from an ECS](#)

You can switch access methods between EIP and Direct Connect by performing the following steps:

Log in to the MRS management console and click the target MRS cluster in the cluster list on the **Active Clusters** page. On the **Dashboard** tab page that is displayed, in the **Basic Information** area, click  next to **Access Manager** to switch access methods.



 NOTE

If you cannot log in to the web UI of the cluster, access FusionInsight Manager by referring to [Accessing FusionInsight Manager from an ECS](#).

## Accessing FusionInsight Manager As a VDC User Created on ManageOne

After you create an MRS cluster through the console, you can log in to FusionInsight Manager as a VDC user.

### Accessing FusionInsight Manager Using an EIP

**Step 1** Log in to the MRS management console.

**Step 2** On the **Active Clusters** page that is displayed by default, click the target cluster in the cluster list to access its details page.

**Step 3** On the displayed **Dashboard** tab page, click **Access Manager** next to **MRS Manager**. In the **Access MRS Manager** dialog box that is displayed, select **EIP** for **Access Mode** and configure the EIP information.

1. If an MRS cluster is not bound with an EIP during creation, select an EIP from the drop-down list next to **EIP**. If an EIP has been bound, go to [Step 3.2](#).

 NOTE

If no EIPs are available in the drop-down list, click **Manage EIP** to create one. Then, select the created EIP from the drop-down list.

2. In the **Security Group** area, select the security group to which the security group rule to be added belongs. The security group is configured when the cluster is created.
3. Add a security group rule. The rule for accessing the EIP is automatically used by default. To enable multiple trusted IP address ranges to access FusionInsight Manager, see steps [Step 6](#) to [Step 9](#). To view, modify, or delete a security group rule, click **Manage Security Group Rule**.
4. Confirm the information and click **OK**.

 NOTE

Click  next to **Access Manager** to change the FusionInsight Manager access mode. For details about how to access FusionInsight Manager using a Direct Connect connection, see [Accessing FusionInsight Manager Using a Direct Connect Connection](#).

**Step 4** Click **OK**.

**Step 5** On the FusionInsight login page that is displayed, enter the default username **admin** and the password configured during cluster creation, and click **Log In**.

**Step 6** On the MRS management console, choose **Clusters > Active Clusters**. Click the target cluster in the cluster list to access its details page.

 NOTE

To grant other users the permission to access FusionInsight Manager, perform [Step 6](#) to [Step 9](#) to add the users' public IP addresses to the trusted IP address range.

**Step 7** Click **Add Security Group Rule** next to EIP.

**Step 8** On the **Add Security Group Rule** page, add the IP address range for the user to access the public network and select **I confirm that Public IP address/Port number is a trusted public IP address. I understand that using 0.0.0.0/0. poses security risks.**

By default, the IP address used for accessing the public network is used. You can change the IP address range as needed. To enable multiple trusted IP address ranges, repeat steps **Step 6** to **Step 9**. To view, modify, or delete a security group rule, click **Manage Security Group Rule**.

**Step 9** Click OK.

----End

## Accessing FusionInsight Manager Using a Direct Connect Connection

You can access FusionInsight Manager using a Direct Connect connection if Direct Connect has been enabled in the environment.

**Step 1** Log in to the MRS management console.

**Step 2** On the **Active Clusters** page that is displayed by default, click the target cluster in the cluster list to access its details page.

**Step 3** On the **Dashboard** tab page that is displayed, click **Access Manager** next to **MRS Manager**. In the **Access MRS Manager** dialog box that is displayed, select **Direct Connect** for **Access Mode**.

**Step 4** Click OK.

**Step 5** On the FusionInsight login page that is displayed, enter the default username **admin** and the password configured during cluster creation, and click **Log In**.

----End

## Accessing FusionInsight Manager from an ECS

**Step 1** Log in to the MRS management console.

**Step 2** Choose **Active Clusters**. On this page, click the target cluster in the cluster list to access its details page.

On the **Dashboard** tab page that is displayed, record the values of **AZ**, **VPC**, **MRS Manager**, and **Security Group** of the cluster.

**Step 3** On the homepage of the Huawei Cloud management console, choose **Service List > Compute > Elastic Cloud Server** to switch to the ECS management console and create an ECS.

- Set **AZ**, **VPC**, and **Security Group** of the ECS to the same values that you have recorded.
- Select a Windows public image, for example, **Windows Server 2012 R2 Standard 64bit(40GB)**.
- For details about how to configure other parameters, see the ECS operation guide.

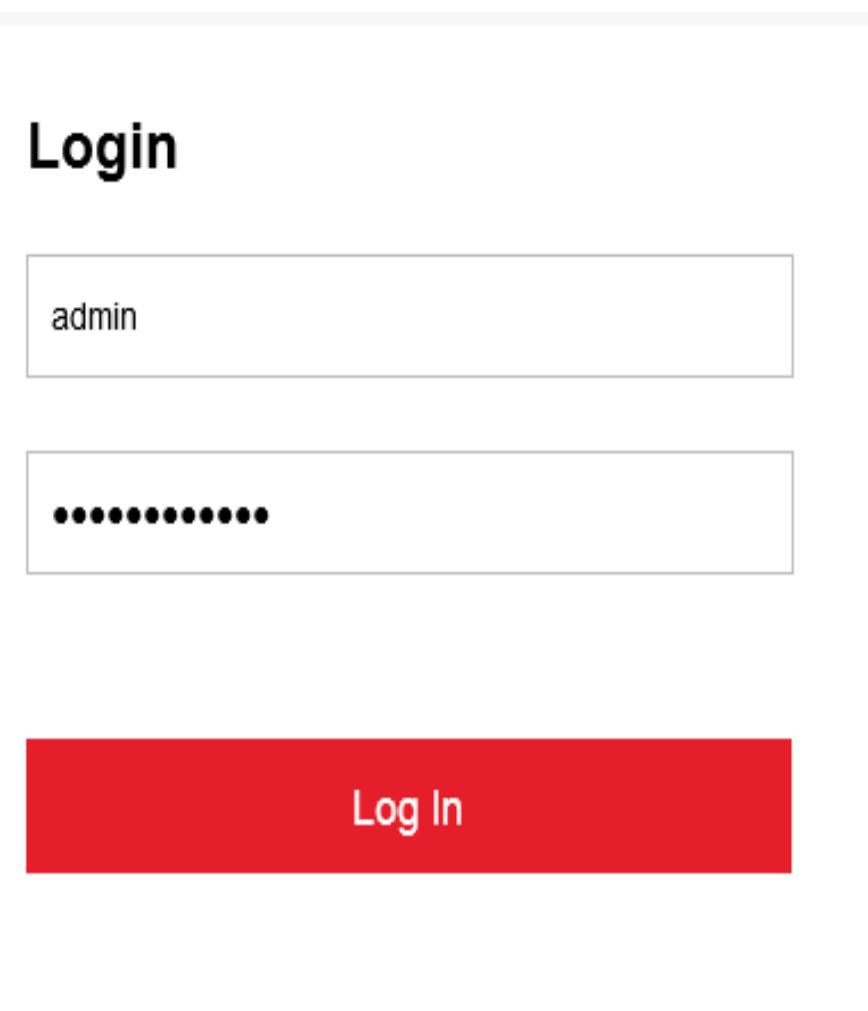
**Step 4** On the VPC management console, apply for an EIP and bind it to the ECS.

**Step 5** Log in to the ECS.

The Windows system account, password, EIP, and security group rules are required for logging in to the ECS.

**Step 6** On the Windows remote desktop, use your browser to access FusionInsight Manager.

The URL for accessing FusionInsight Manager is the value of **MRS Manager** that you have recorded. On the login page, enter the username and password for accessing the cluster, for example, log in as user **admin**.



 **NOTE**

- If you access FusionInsight Manager as another user, change the password upon your first login. The new password must meet the password strength requirements of the current user. For details, contact the administrator.
- By default, a user will be locked after 5 consecutive login failures, but will be automatically unlocked 5 minutes later.

**Step 7** Log out of FusionInsight Manager by moving your cursor over  in the upper right corner and clicking **Log Out**.

----End

## 5.2 Installing a Client

### Scenario

Install the clients of all services, except Flume, in the MRS cluster. MRS provides shell scripts for different services so that maintenance personnel can log in to related maintenance clients and implement maintenance operations.

#### NOTE

- Reinstall the client after server configuration is modified on FusionInsight Manager or after the system is upgraded. Otherwise, the versions of the client and server will be inconsistent.
- To install the Flume client, see "Installing the Flume Client on Clusters" in *MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1)* in the *MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1)*.
- Open JDK 1.8.0\_392 is recommended for the Kunpeng server.
- In openSUSE Leap 15.4, the shared link library files on the HDFS client are incompatible with curl commands. To run curl commands, do not load HDFS client variables.

### Prerequisites

- A client installation directory will be automatically created if it does not exist. If the directory exists, it must be empty. The directory cannot contain any space.
- If a server outside the cluster is used as the client node, the node can communicate with the cluster service plane. Otherwise, client installation will fail.
- The client must have the NTP service enabled and synchronized time with the NTP server. Otherwise, client installation will fail.
- If clients of all components are downloaded, HDFS and MapReduce are installed in the same directory (*Client directory/HDFS/*).
- You can install and use the client as any user whose username and password have been obtained from the system administrator. This section uses **user\_client** as an example. Ensure that user **user\_client** is the owner of the server file directory (for example, `/opt/Bigdata/hadoopclient`) and client installation directory (for example, `/opt/hadoopclient`). The permission for the two directories is **755**.
- You have obtained the component service username (a default user or new user) and password from the system administrator.
- When you install the client as a user other than **omm** or **root**, and the `/var/tmp/patch` directory already exists, you have changed the permission for the directory to **777** and changed the permission for the logs in the directory to **666**.

- Ensure that port 20029 is enabled. Otherwise, client registration fails.

## Procedure

### Step 1 Obtain the required software packages.

Log in to FusionInsight Manager.

In the upper right corner of **Homepage**, click **Download Client**. The **Download Cluster Client** page is displayed.

**Figure 5-1** Downloading a client

#### Download Cluster Client

Download the **MRS Cluster** client. The cluster client provides all services.

The screenshot shows a configuration interface for downloading a cluster client. At the top, there are two tabs: 'Complete Client' (which is selected) and 'Configuration Files Only'. Below this, under 'Select Platform Type', 'aarch64' is selected. Under 'Select Download Location', 'Server' is selected. A red asterisk indicates a required field for 'Save to Path', which is set to '/tmp/FusionInsight-Client/'. There is also a question mark icon for help.

#### NOTE

If you only need to install the client of a service in the cluster, choose **Cluster > Services**, click a service name, click **More**, and select **Download Client**. The **Download Client** page is displayed.

### Step 2 Set Select Client Type to Complete Client.

**Configuration Files Only** is to download client configuration files in the following scenario: After a complete client is downloaded and installed and the system administrator modifies server configurations on Manager, developers need to update the configuration files during application development.

- **x86\_64**: indicates the client software package that can be deployed on the x86 servers. This option is unavailable for Kunpeng clusters. To use it, register the x86 software packages with the cluster by referring to "Registering Cross-Platform or Operating System Software Packages" in *MRS Installation Guide in Huawei Cloud Stack 8.3.1 Software Installation Guide for gPaaS & AI DaaS Services*.
- **aarch64**: indicates the client software package that can be deployed on the Kunpeng servers. This option is unavailable for x86 clusters. To use it, register the Kunpeng software packages with the cluster by referring to "Registering Cross-Platform or Operating System Software Packages" in *MRS Installation Guide in Huawei Cloud Stack 8.3.1 Software Installation Guide for gPaaS & AI DaaS Services*.

 NOTE

- If multiple types of OSs are deployed in a cluster and multiple OS component packages are registered, the platform type is displayed by OS. For example, **redhat-x86\_64** and **suse-aarch64**. Otherwise, only the parameters **x86\_64** and **aarch64** are displayed.
- If the client is installed on a node outside the cluster, the operating system platform type of the client you want to download must be the same as that of the node outside the cluster, and the operating system version must be supported by the MRS cluster. For details about the supported operating system versions, see "Preparing the OS" in the MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1).

**Step 3** Select the path for saving the downloaded client file.

You can directly download the client file to the node where the client is to be installed, or download the file to the active OMS node or local computer and copy it to the node where the client is to be installed.

- **Server:** Download the file to the active OMS node of the cluster.

The generated file is stored in the **/tmp/FusionInsight-Client** directory on the active OMS node by default. You can also store the client file in other directories, and user **omm** has the read, write, and execute permissions on the directory. If the client file already exists in the path, the existing client file will be replaced.

 NOTE

When a cluster has many services installed, the cluster client file becomes quite large. Additionally, decompressing this file during installation can consume significant disk space. It's recommended to download the client files to a different directory that has ample space, or to promptly remove unnecessary files from the client download directory after installation. Doing so helps avoid exhausting the **/tmp** directory's disk space, which could interrupt the normal operation of the cluster nodes.

After the file is generated, copy the obtained package to another directory, for example, **/opt/Bigdata/hadoopclient**, as user **omm** or client installation user.

- **Browser:** Download the file to the local computer.
- **Remote node:** Download the file to a node other than the active OMS node. If you select this option, you need to set the following parameters:

**Table 5-1** Parameters

Parameter	Description	Example Value
Save to Path	Path for storing client files. If there is already a client file in the path, it will be overwritten. For a remote node, write permission for the path is required.	/tmp/FusionInsight-Client-Remote/

Parameter	Description	Example Value
Host IP Address	IP address of the remote node. <b>NOTE</b> The platform type of the remote node must be the same as that of the downloaded client. Otherwise, the client may fail to be installed.	x.x.x.x
Host Port	Host port of the remote node.	22
Username	Username for logging in to the remote node. For a remote node, write permission for the path is required.	xxx
Authentication Method	You can choose one of the following methods: <ul style="list-style-type: none"><li>- <b>Password:</b> Use the password for login.</li><li>- <b>SSH private keys:</b> Use SSH private keys for login.</li><li>- <b>None:</b> To use this method, passwordless login needs to be enabled for the node.</li></ul>	Password
Password	This parameter is mandatory when <b>Authentication Method</b> is set to <b>Password</b> . This parameter indicates the password used for login.	xxx
SSH Private Keys	This parameter is mandatory when <b>Authentication Method</b> is set to <b>SSH private keys</b> . Click <b>Select File</b> and select a local file to upload.	-

Parameter	Description	Example Value
Auto Deployment	<p>Whether to enable auto deployment. This parameter is mandatory when <b>Select Client Type</b> is set to <b>Complete Client</b>.</p> <ul style="list-style-type: none"><li>- If you set this parameter to <b>yes</b>, the client is automatically installed and deployed on the current node.</li><li>- If you set this parameter to <b>no</b>, the client will not be automatically installed and deployed. You need to manually install the client after it is downloaded.</li></ul>	Yes
Deployment Path	<p>This parameter is mandatory when <b>Auto Deployment</b> is set to <b>Yes</b>. If only the configuration file is downloaded, this parameter will not be displayed.</p> <p>The deployment path must be empty if it already exists on the remote node. Otherwise, it will be created automatically. The path also requires operate and write permissions.</p>	/opt/testclient

Copy the obtained software package to the file directory (for example, **/tmp/FusionInsight-Client**) of the server where the client is to be installed as the user (for example, **user\_client**) who is preparing to install the client.

The name of the client software package is in the following format:  
**FusionInsight\_Cluster\_<Cluster ID>\_Services\_Client.tar**.

The following steps and sections use **FusionInsight\_Cluster\_1\_Services\_Client.tar** as an example.

 NOTE

The host where the client is to be installed can be a node inside or outside the cluster. If the node is a server outside the cluster, it must be able to communicate with the service plane network of the cluster, and the NTP service must be enabled to ensure that the time is the same as that on the server.

For example, you can configure the same NTP clock source for external servers as that of the cluster. After the configuration, you can run the **ntpq -np** command to check whether the time is synchronized.

- If there is an asterisk (\*) before the IP address of the NTP clock source in the command output, the synchronization is normal. For example:

```
=====
=
*10.10.10.162 .LOCL. 1 u 1 16 377 0.270 -1.562 0.014
```

- If there is no asterisk (\*) before the IP address of the NTP clock source and the value of **refid** is **.INIT.**, or if the command output is abnormal, the synchronization is abnormal. Contact technical support.

```
=====
=
remote refid st t when poll reach delay offset jitter
=====
=
10.10.10.162 .INIT. 1 u 1 16 377 0.270 -1.562 0.014
```

You can also configure the same chrony clock source for external servers as that for the cluster. After the configuration, run the **chronyc sources** command to check whether the time is synchronized.

- In the command output, if there is an asterisk (\*) before the IP address of the chrony service on the active OMS node, the synchronization is normal. For example:

```
MS Name/IP address      Stratum Poll Reach LastRx Last sample
```

```
=====
=
^* 10.10.10.162          10 10 377 626 +16us[ +15us] +/- 308us
```

- In the command output, if there is no asterisk (\*) before the IP address of the NTP service on the active OMS node, and the value of **Reach** is **0**, the synchronization is abnormal.

```
MS Name/IP address      Stratum Poll Reach LastRx Last sample
```

```
=====
=
^? 10.1.1.1              0 10 0   - +0ns[ +0ns] +/- 0ns
```

**Step 4** Log in to the server where the client software package is located as user **user\_client**.

**Step 5** Decompress the software package.

Go to the directory where the installation package is stored, such as **/tmp/FusionInsight-Client**. Run the following command to decompress the installation package to a local directory:

```
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
```

**Step 6** Verify the software package.

Run the following command to verify the decompressed file and check whether the command output is consistent with the information in the **sha256** file:

```
sha256sum -c FusionInsight_Cluster_1_Services_ClientConfig.tar.sha256
```

```
FusionInsight_Cluster_1_Services_ClientConfig.tar: OK
```

**Step 7** Decompress the obtained installation file.

```
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
```

**Step 8** Configure network connections for the client.

1. Ensure that the host where the client is installed can communicate with the hosts listed in the **hosts** file in the decompression directory (for example, `/tmp/FusionInsight-Client/FusionInsight_Cluster_<Cluster ID>_Services_ClientConfig/hosts`).
2. If the host where the client is installed is not a host in the cluster, you need to set the mapping between the host name and the service plane IP address for each cluster node in **/etc/hosts**, as user **root**. Each host name uniquely maps an IP address. You can perform the following steps to import the domain name mapping of the cluster to the **hosts** file:
  - a. Switch to user **root** or a user who has the permission to modify the **hosts** file.  
**su - root**
  - b. Go to the directory where the client package is decompressed.  
**cd /tmp/FusionInsight-Client/FusionInsight\_Cluster\_1\_Services\_ClientConfig**
  - c. Run the **cat realm.ini >> /etc/hosts** command to import the domain name mapping to the **hosts** file.

 **NOTE**

- If the host where the client is installed is not a node in the cluster, configure network connections for the client to prevent errors when you run commands on the client.
- If Spark tasks are executed in yarn-client mode, add the **spark.driver.host** parameter to the file *Client installation directory/Spark/spark/conf/spark-defaults.conf* and set the parameter to the client IP address.
- If the yarn-client mode is used, you need to configure the mapping between the IP address and host name of the client in the **hosts** file on the active and standby Yarn nodes (ResourceManager nodes in the cluster) to make sure that the Spark web UI is properly displayed.

**Step 9** Go to the directory where the installation package is stored, and run the following command to install the client to a specified directory (an absolute path), for example, **/opt/hadoopclient**. The client installation directory can contain only uppercase letters, lowercase letters, digits, and underscores (\_).

**cd /tmp/FusionInsight-Client/FusionInsight\_Cluster\_1\_Services\_ClientConfig**

Run the **./install.sh /opt/hadoopclient** command to install the client. The client is successfully installed if information similar to the following is displayed:

ALL component client is installed successfully

 NOTE

- If the **/opt/hadoopclient** directory has been used by existing service clients, you need to use another directory in this step when installing other service clients.
- You must delete the client installation directory when uninstalling a client.
- To ensure that an installed client can only be used by the installation user (for example, **user\_client**), add parameter **-o** during the installation. That is, run the **./install.sh /opt/hadoopclient -o** command to install the client.
- If the server where the client is to be installed is in the cluster, you do not need to specify the NTP server mode. You can run the **./install.sh /opt/hadoopclient** command to install the client.
- If the server where the client is to be installed is outside the cluster and the NTP server mode is **chrony**, run the **./install.sh /opt/hadoopclient -o chrony** command to install the client. If the NTP server mode is not specified, the NTP server mode in the cluster is used by default.
- If you do not need to verify clock synchronization during client installation, run the **./install.sh /opt/hadoopclient -u** command to install the client.
- If you do not need to verify clock synchronization during client installation and the installed client can be used by the installation user only (for example, **user\_client**), add the **-ou** parameter during the installation, that is, run the **./install.sh /opt/hadoopclient -ou** command to install the client.
- If the client node is a server outside the cluster and cannot communicate with the service plane IP address of the active OMS node or cannot access port 20029 of the active OMS node, the client can be successfully installed but cannot be registered with the cluster or displayed on the UI.

**Step 10** Log in to the client to check whether the client is successfully installed.

1. Run the **cd /opt/hadoopclient** command to go to the client installation directory.
2. Run the **source bigdata\_env** command to configure environment variables for the client.
3. Run related commands based on the cluster mode.
  - For a cluster in normal mode, directly run commands related to the component client, for example, **hdfs**.
  - For a cluster in security mode, run the following command to set **kinit** authentication and enter the password for logging in to the client. For a cluster in normal mode, user authentication is not required.

**kinit admin**

Password for xxx@HADOOP.COM: #Enter the login password of user **admin** (same as the user password for logging in to the cluster).

Run the **klist** command to query and confirm authentication details.

Ticket cache: FILE:/tmp/krb5cc\_0  
Default principal: xxx@HADOOP.COM

Valid starting Expires Service principal  
04/09/2021 18:22:35 04/10/2021 18:22:29 krbtgt/HADOOP.COM@HADOOP.COM

 NOTE

- When kinit authentication is used, the ticket is stored in the `/tmp/krb5cc_uid` directory by default.  
*uid* indicates the ID of the user who logs in to the OS. For example, if the UID of user **root** is 0, the ticket generated for kinit authentication after user **root** logs in to the system is stored in the `/tmp/krb5cc_0` directory.  
If the current user does not have the read/write permission for the `/tmp` directory, the ticket cache path is changed to **Client installation directory**/`tmp/krb5cc_uid`. For example, if the client installation directory is `/opt/hadoopclient`, the kinit authentication ticket is stored in `/opt/hadoopclient/tmp/krb5cc_uid`.
- If the same user is used to log in to the OS for kinit authentication, there is a risk that tickets are overwritten. You can set the `-c cache_name` parameter to specify the ticket cache path or set the **KRB5CCNAME** environment variable to avoid this problem.

**Step 11** After the cluster is reinstalled, the previously installed client is no longer available. Perform the following operations to deploy the client again:

1. Log in to the node where the client is deployed as user **root**.
2. Run the following command to view the directory where the client is located: (In the following example, `/opt/hadoopclient` is the directory where the client is located.)

**ll /opt**

```
drwxr-x---. 6 root root 4096 Dec 11 19:00 hadoopclient
drwxr-xr-x. 3 root root 4096 Dec  9 02:04 godi
drwx-----. 2 root root 16384 Nov  6 01:03 lost+found
drwxr-xr-x. 2 root root 4096 Nov  7 09:49 rh
```

3. Run the following command to delete the files in the folder (for example, `/opt/hadoopclient`) where all client programs are located:

**mv /opt/hadoopclient /tmp/clientbackup**

4. Reinstall the client.

----End

## 5.3 Open Source Application Reconstruction Guide

MRS provides enterprise-level big data storage, query, and analysis. It helps enterprises quickly build a massive data processing system and easily run big data components and applications such as Hadoop, Spark, HBase, and Kafka.

This document provides common application development examples and specifications for developers to develop service applications in MRS clusters.

MRS encapsulates and enhances open source components and is compatible with open source APIs of each component. Upper-layer applications that have been developed on other big data platforms can run on MRS smoothly only through simple adaptation.

- For details about the security authentication principles and authentication API sample code of MRS clusters in security mode, see [Security Authentication](#) and development guides of each component in security mode.

- For details about the APIs of each component, see *MapReduce Service (MRS) 3.3.1-LTS API Reference (for Huawei Cloud Stack 8.3.1)* and the corresponding sections in the development guide of each component. For example, for details about HDFS APIs, see [Common API Introduction](#).
- You can use Loader of MRS clusters and Cloud Data Migration (CDM) to migrate data from other big data platforms or systems to MRS clusters. For details about the migration scenarios and operations, see "Component Operation Guide" > "Using Loader" in MapReduce Service (MRS) 3.3.1-LTS User Guide (for Huawei Cloud Stack 8.3.1) in the MapReduce Service (MRS) 3.3.1-LTS Usage Guide (for Huawei Cloud Stack 8.3.1) and "User Guide" > "DataArts Migration" in DataArts Studio 2.10.1 User Guide (for Huawei Cloud Stack 8.3.1).