Hive应用开发

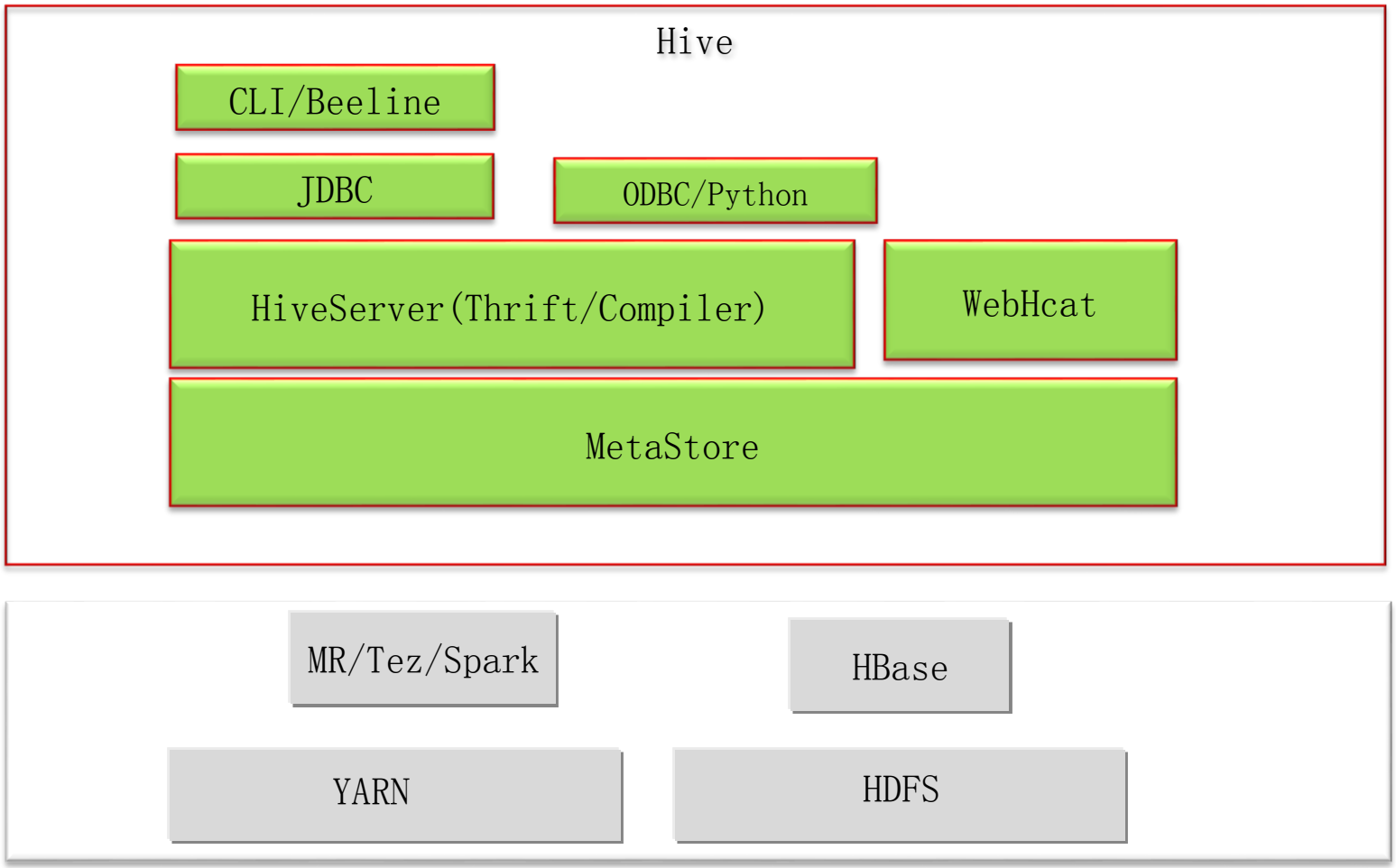
www.huawei.com

目标

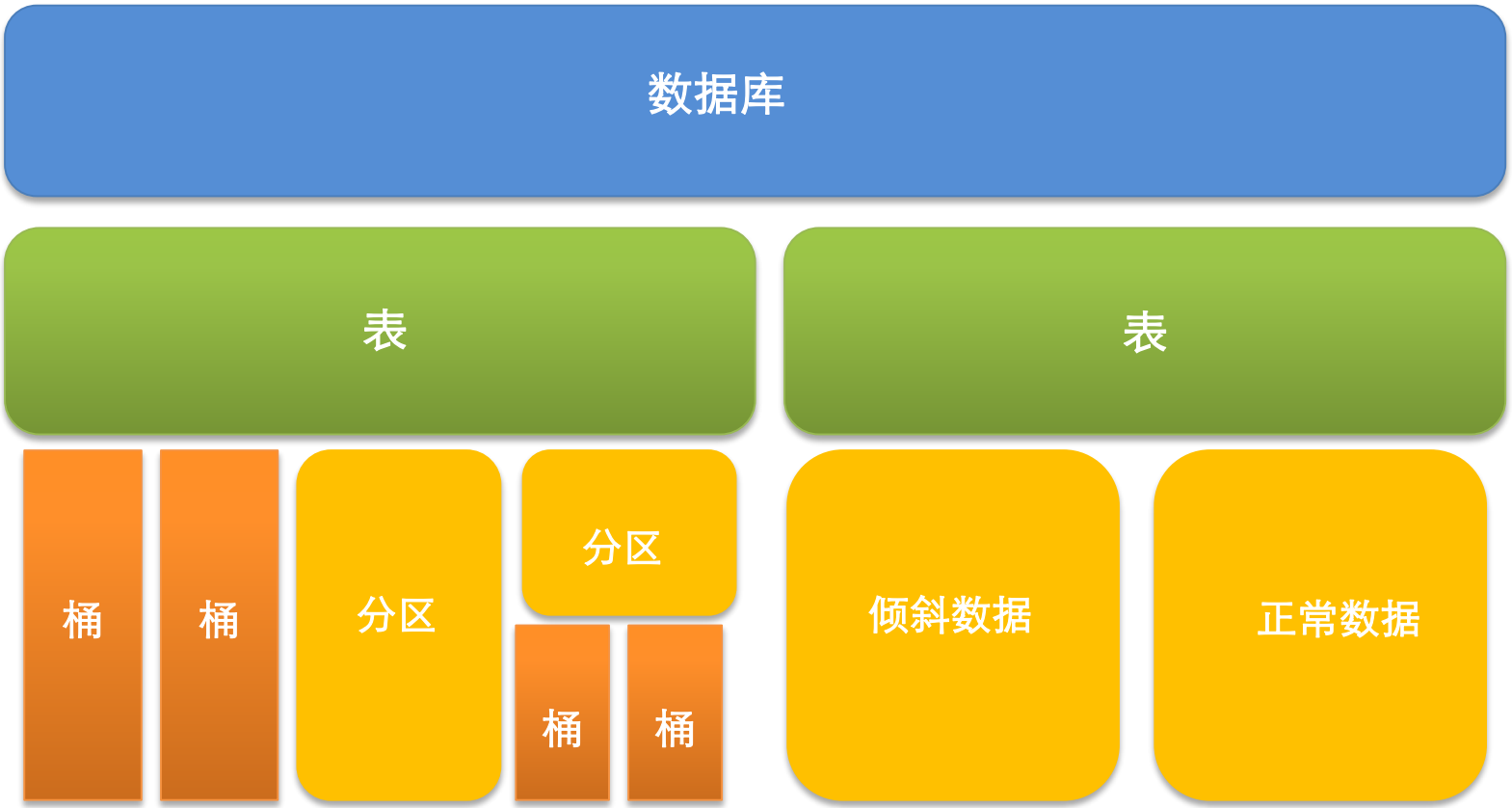
- 学完本课程后，您将能够:
  - 了解**Hive**的基本架构原理
  - 掌握**JDBC**客户端开发流程
  - 了解**ODBC**客户端开发流程
  - 了解**Python**客户端开发流程
  - 了解**Hcatalog/WebHcat**开发接口
  - 掌握**Hive**开发规则

目录

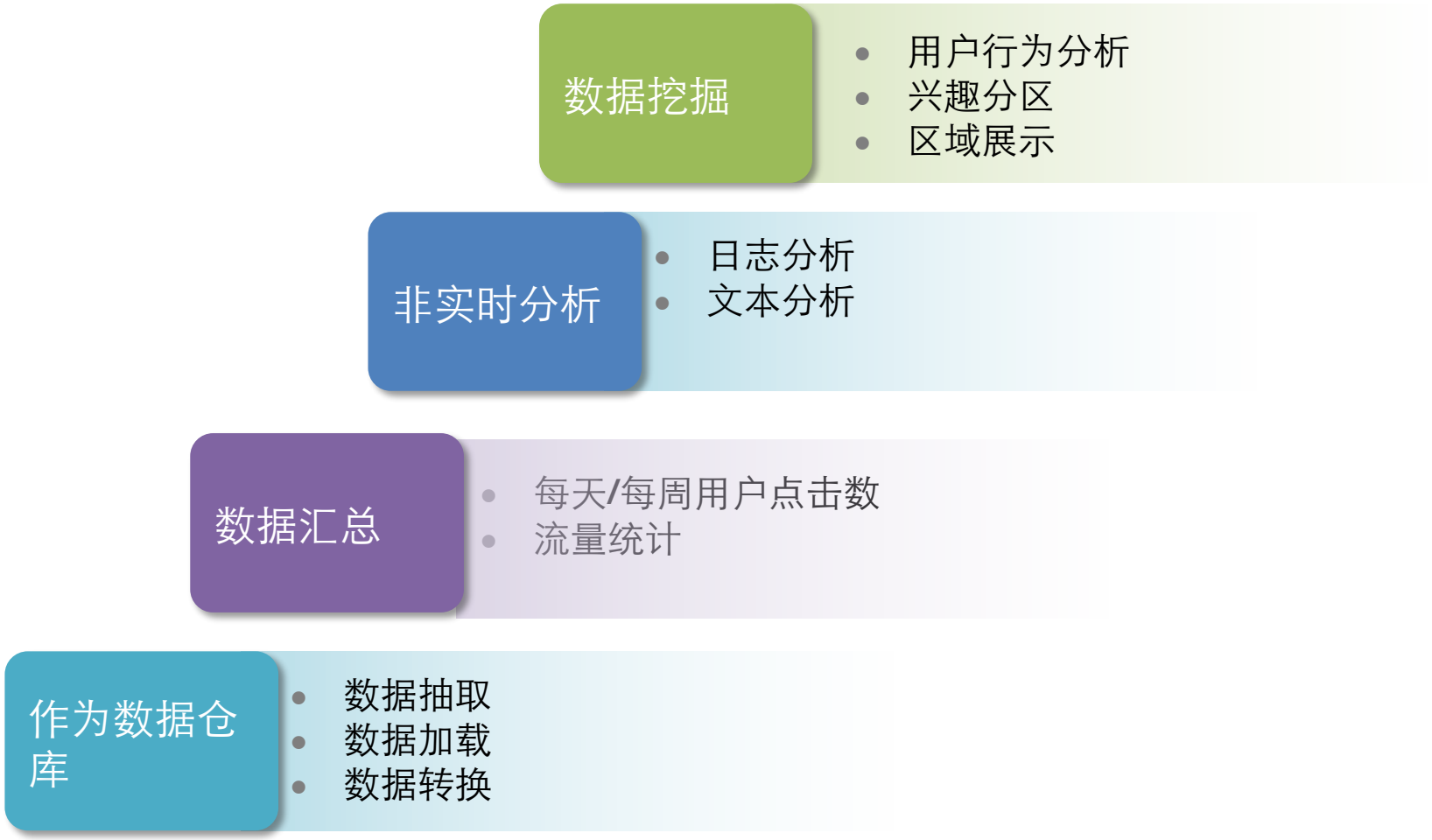
Hive的架构原理



Hive数据模型



Hive的应用场景



本章小结

本章主要介绍了**Hive**的架构原理，数据模型以及应用场景。通过学习，可以对**hive**有一个宏观的了解。

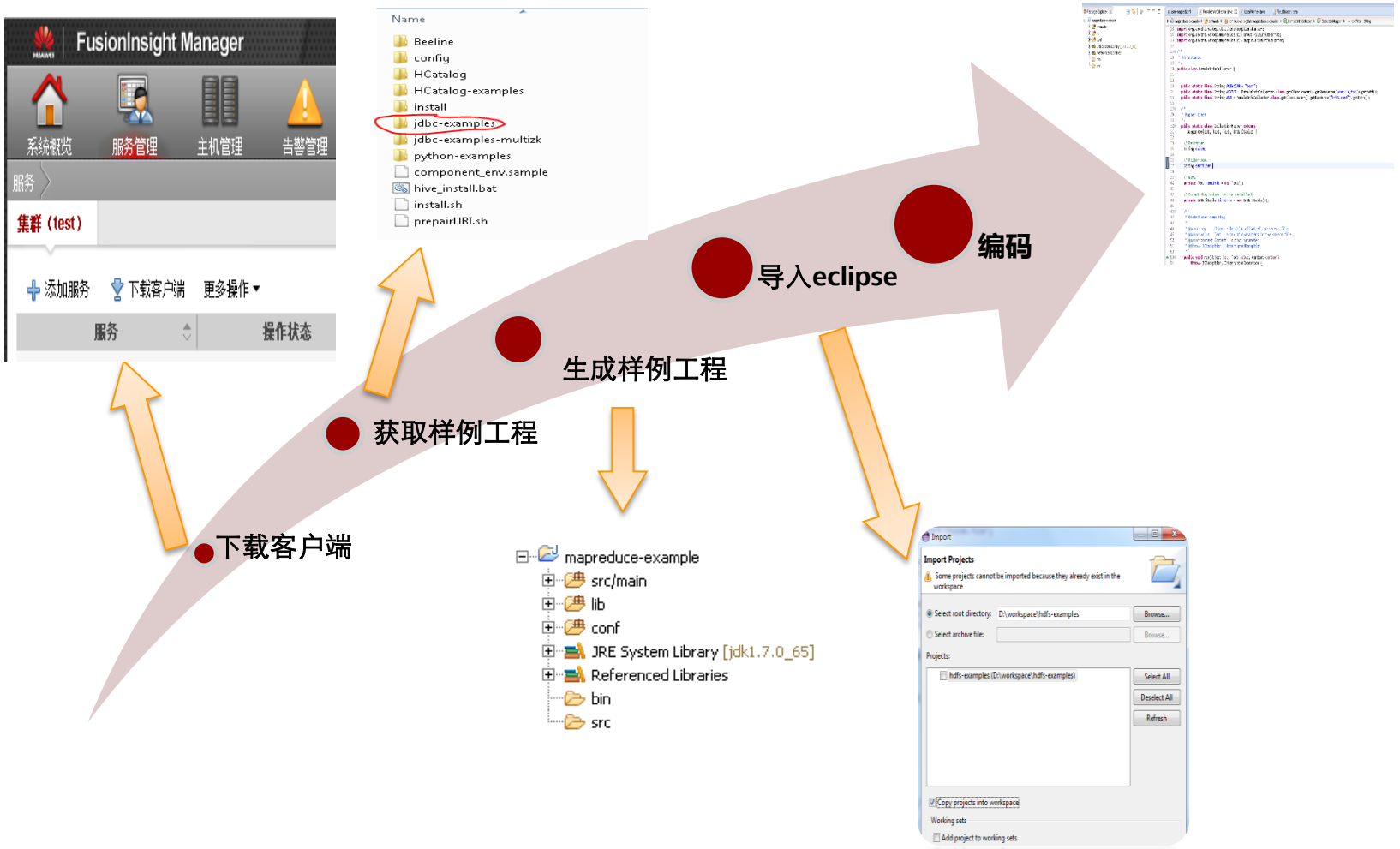
目录

HUAWEI

Hive二次开发—总体介绍

- **JDBC**接口：
  Java标准接口，适合基于**Java**语言的应用程序开发
- **ODBC**接口：
  适合基于**C/C++**语言的程序开发
- **Python**接口：
  适合基于**Python**语言访问**Hive**的场景
- **WebHCat**接口：
  适合基于**HTTP**、**HTTPS**访问**Hive**的场景，返回格式为**JSON**
- **HCatlog**：
  WebHcat接口的底层实现接口

JDBC开发-搭建开发环境



HUAWEI

JDBC开发-参数初始化

设置**ZooKeeper**地址

获取**krb5**文件路径

设置**JAAS**配置

配置**ZooKeeper Principal**

执行登录

```
private static void init() throws IOException{
    CONF = new Configuration();
    zkQuorum =
"xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002,xxx.xxx.xxx.xxx:24002";
    USER_NAME = "xxx";
    if (isSecureVerson) {
      String userdir = System.getProperty("user.dir") + File.separator
              + "conf" + File.separator;
      USER_KEYTAB_FILE = userdir + "hive.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);
      LoginUtil.login(USER_NAME, USER_KEYTAB_FILE, KRB5_FILE, CONF);
    } }}
```

JDBC开发-拼接URL

JDBC前缀设置

服务发现模式

安全配置：
qop,auth,principal

非安全配置

```
StringBuilder sBuilder = new StringBuilder(
        "jdbc:hive2://").append(zkQuorum).append("/");
if (isSecureVerson) {
  sBuilder.append(";serviceDiscoveryMode=")
          .append("zooKeeper")
          .append(";zooKeeperNamespace=")
          .append("hiveserver2;sasl.qop=auth-
conf;auth=KERBEROS;principal=hive/hadoop.hadoop.com@HADOOP.COM")
          .append(";");
} else {
  sBuilder.append(";serviceDiscoveryMode=")
          .append("zooKeeper")
          .append(";zooKeeperNamespace=")
          .append("hiveserver2;auth=none");
} String url = sBuilder.toString();
```

JDBC开发-执行SQL

加载驱动类

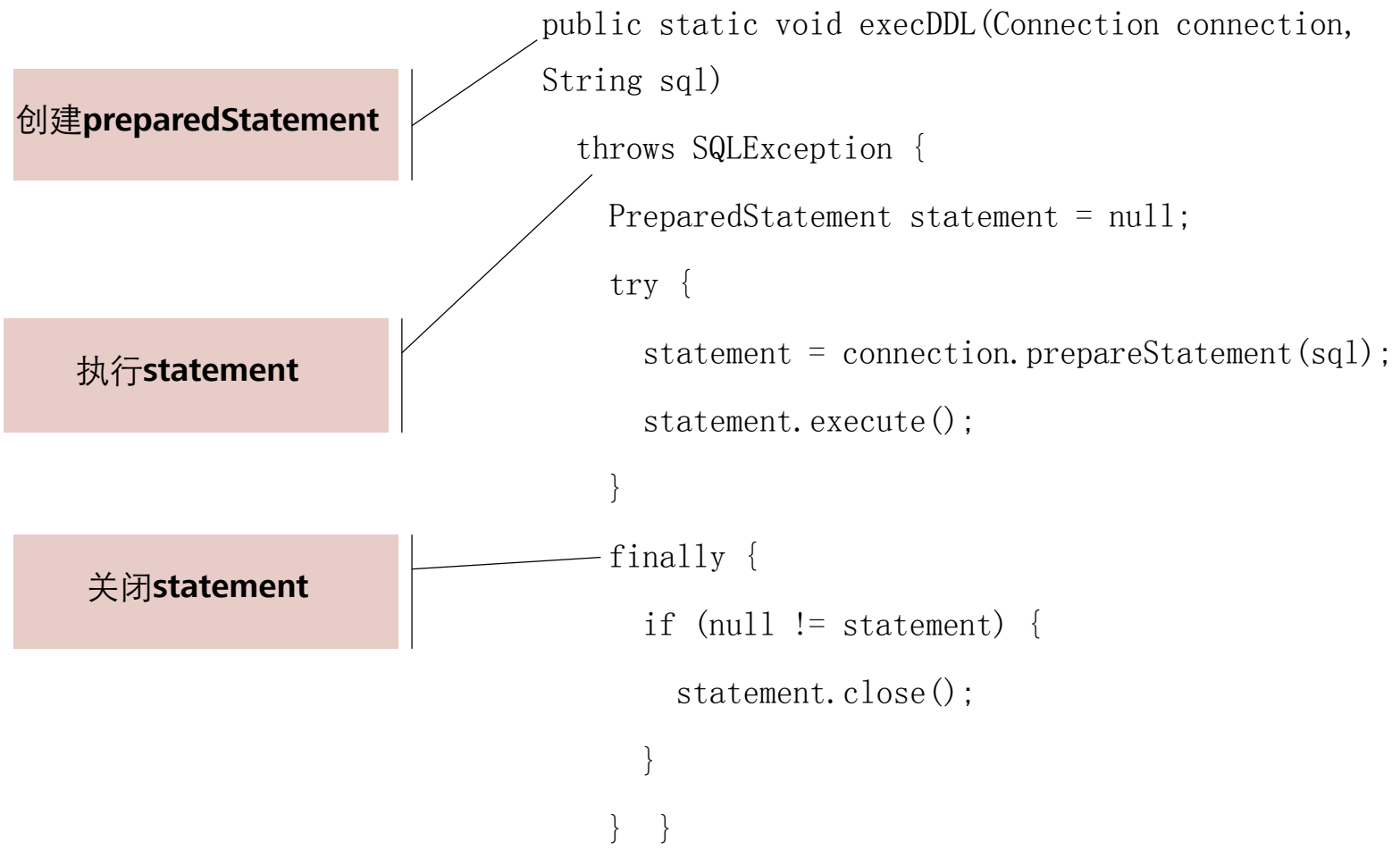
建立连接

执行SQL

关闭连接

```
Class.forName(HIVE_DRIVER);
Connection connection = null;
try {
  connection = DriverManager.getConnection(url, "", "");
  execDDL(connection,sqls[0]);
  System.out.println("Create table success!");
  execDML(connection,sqls[1]);
  execDDL(connection,sqls[2]);
  System.out.println("Delete table success!");
}
finally {
  if (null != connection) {
    connection.close();
  }
}
```

JDBC开发-SQL实现



JDBC开发– 示例程序运行

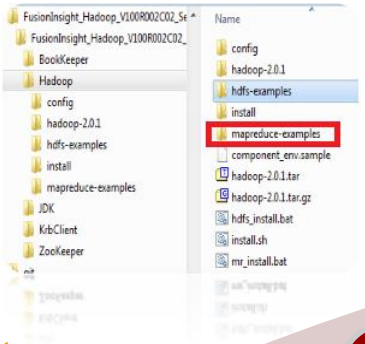
- 安装**Eclipse**。
- 打开**HiveJDBCExample**工程。
- 双击**JDBCExample**类。
- 选中**main**函数，右键点击**Run As →Java Application**。
- 成功后会打印出语句的执行结果。

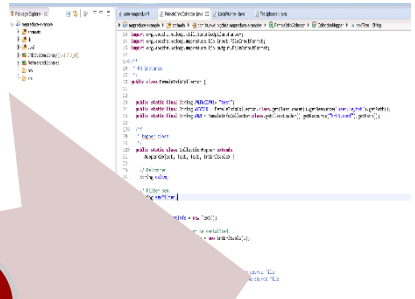
ODBC开发—标准与目录

- **FusionInsight Hive ODBC驱动标准：**
  - **ODBC 3.5**标准
  - 不兼容**ODBC1.x**和**ODBC 2.x**标准
  - **Windows**下支持微软标准
  - **Linux**下支持**unix ODBC**标准。
- **FusionInsight Hive ODBC示例工程目录（获取方法见下页）：**

```
FusionInsight_Hive_ODBC_Driver
|--Linux
   |--Fusioninsight-hive-odbc-1.0.0-1.el6.x86_64.rpm
   |--Fusioninsight-hive-odbc-1.0.0-1.x86_64.rpm
|--Windows
   |--hiveodbc.dll
   |--libeay32.dll
   |--libsasl.dll
   |--odbcreg.exe
   |--ssleay32.dll
   |--template.dsn
   |-- jars/
|-- HiveODBCExample/
```

ODBC开发-搭建开发环境





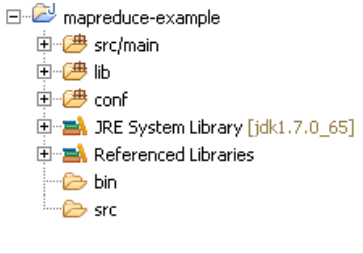
编码

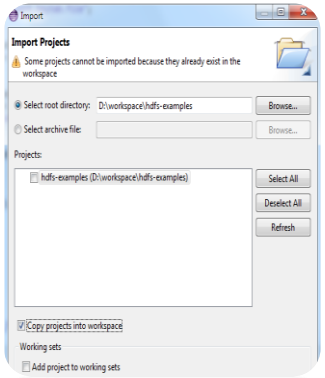
VisualStadio打开工程

其他环境配置

安装ODBC驱动

下载客户端



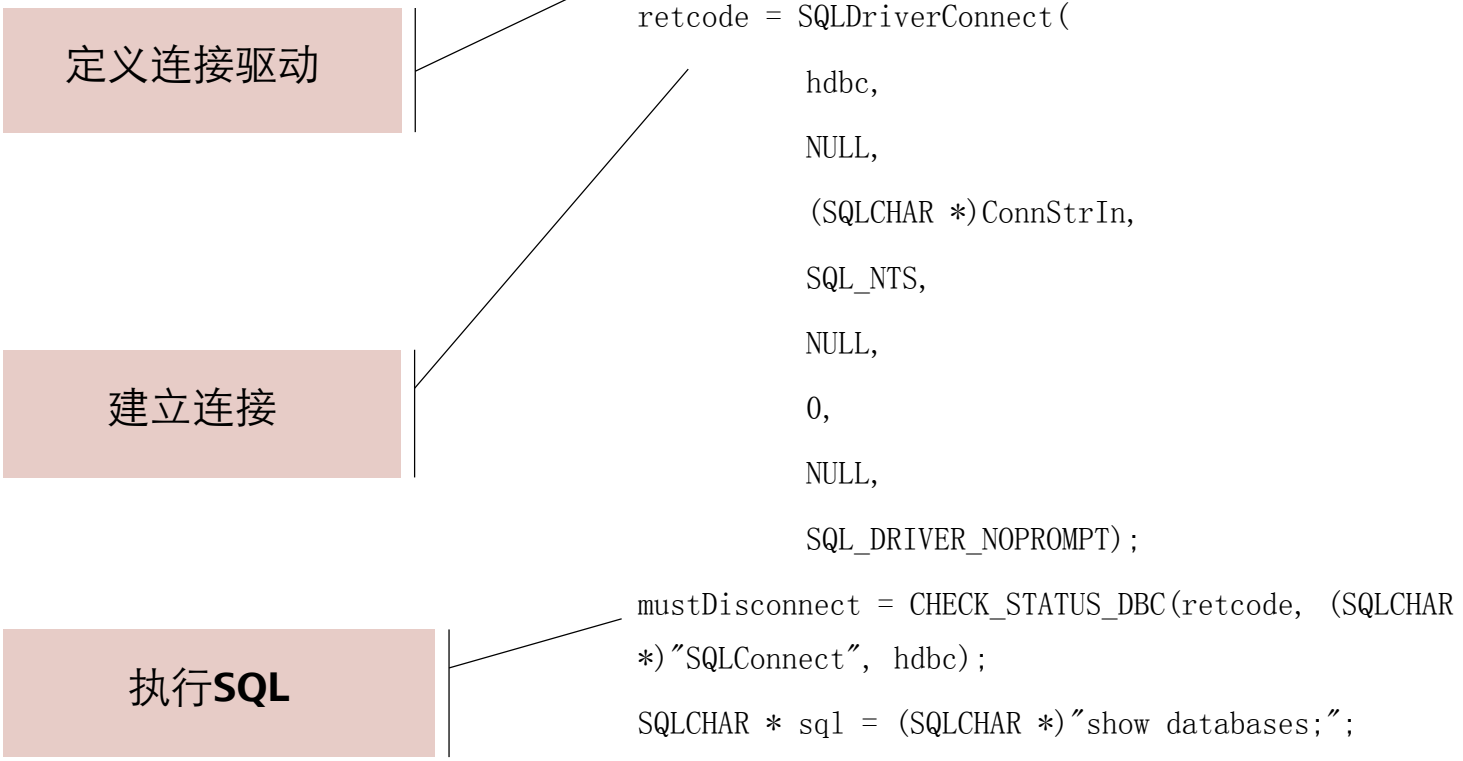


HUAWEI

ODBC开发- 环境准备

- **FusionInsight Hive ODBC**驱动安装：
  - **MIT Kerberos** 安装
  - **JDK1.8**安装
  - 驱动注册
  - 数据源配置
  - 环境变量配置

ODBC开发-示例程序



ODBC开发-示例程序

定义返回字段字符串名字和长度

分配**statement**句柄

```
void viewFetchresults(SQLHDBC hdbc,SQLCHAR * SQL) {
    SQLRETURN sr;
    SQLHSTMT hstmt;
    // Column Date Variables
    struct {
        SQLCHAR tab_name[51];
        SQLINTEGER nameLength;
    } row;
    char message[4096];
    strcpy (message, "Fetch Results:\n"); //Initialize
    // Allocate Statement Handle
    sr = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if(sr != SQL_SUCCESS && sr != SQL_SUCCESS_WITH_INFO)
       printf("Error in allocating statement in
OnViewfetchresults");
```

ODBC开发-示例程序

执行SQL

```
// Execute SQL statement
    sr = SQLExecDirect(hstmt, SQL, SQL_NTS);
if(sr != SQL_SUCCESS && sr != SQL_SUCCESS_WITH_INFO)
    printf("Error executing statement in
OnViewfetchresults");
```

绑定结构体到statement句柄

```
// Bind each column
  sr = SQLBindCol(hstmt, 1, SQL_C_CHAR,
    row.tab_name, sizeof(row.tab_name),
&row.nameLength);
  if(sr != SQL_SUCCESS && sr != SQL_SUCCESS_WITH_INFO)
    printf("Error in Binding 1 in OnViewfetchresults");
```

ODBC开发-示例程序

获取结果

释放句柄

```
// Start fetching records
    while (SQLFetch(hstmt) == SQL_SUCCESS) {
        sprintf(message,
          "%s\t%s \n",
          message,
          row.tab_name);
    }
    printf(message);
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
}
```

ODBC开发– 示例程序运行

- **FusionInsight Hive ODBC**驱动安装：
  - 安装**Visual Studio 2012**。
  - 打开**HiveODBCExample**。
  - 双击**HiveODBCExample.sln**，在**Visual Studio 2012**中打开**HiveODBCExample**工程。
  - 在左边的导航栏中选中**HiveODBCExample**工程，按“**F5**”键，编译并运行示例代码。
  - 成功后会打印出所连的**Hive**中的所有数据库名称。

Python开发-环境准备

- **使用Python进行二次开发前请确认以下事项：**
  - 确认**FusionInsight HD**产品**Hive**组件已经安装，并且正常运行。
  - 客户端机器必须安装有**Python**，其版本不低于**2.6.6**，最高不能超过**2.7.0**。
  - 客户端机器必须安装有**setuptools**，其版本不低于**5.0**。
  - 客户端机器的时间与**FusionInsight HD**集群的时间要保持一致，时间差要小于**5**分钟。
  - **Hive**客户端已经成功下载到本地。（下载过程请参考**JDBC**开发-环境搭建章节）。

Python开发-工程搭建

- **Python客户端示例程序搭建步骤：**
  - 解压客户端端安装包，进入解压后的目录（以下简称解压目录）。
  - 进入“**Hive**”子目录。
  - 将“**python-examples**”文件夹拷贝到客户端机器。
  - 进入客户端机器的“**python-examples**”文件夹。
  - 在客户端机器的命令行终端执行**python setup.py install**。
  - 输出以下关键内容表示安装**Python**客户端成功。
  - **Finished processing dependencies for pyhs2==0.5.0** 。

Python开发-代码示例

定义要连接的HiveServer地址，可以是多个

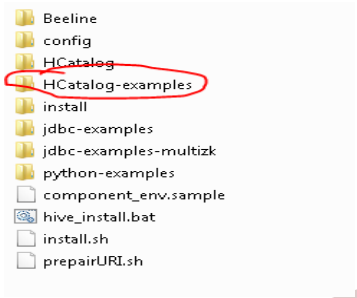
建立连接

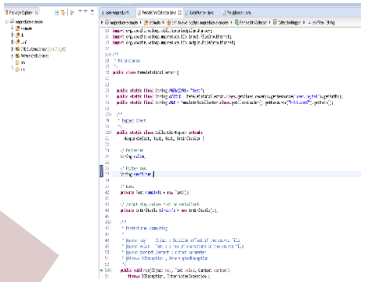
执行语句

打印结果

```
hosts = ["xxx.xxx.xxx.xxx", "xxx.xxx.xxx.xxx"]
conf = {"krb_host":"hadoop.hadoop.com",
"krb_service":"hive"}
try:
        with HAConnection(
        hosts = hosts, port = 21066, authMechanism =
        "KERBEROS", configuration = conf) as haConn:
        with haConn.getConnection() as conn:
        with conn.cursor() as cur:
        print cur.getDatabases()
        cur.execute("show tables")
        print cur.getSchema() for i in cur.fetch():
        print i
except Exception,
e: print e
```

HCatalog -工程搭建





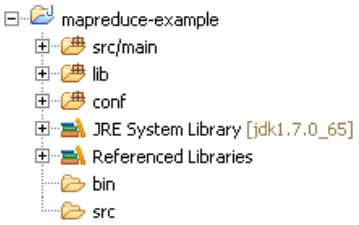
下载客户端

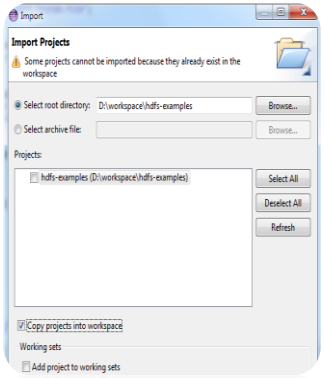
获取样例工程

生成样例工程

导入eclipse

编码





HUAWEI

HCatalog开发讲解

- HCatInputFormat: HCatInputFormat 用于提供接口给MapReduce程序，用于直接从Hive表里读取数据。
  - setInput
  - setOutputSchemasetInput
  - setOutputSchema
- HCatOutputFormat: HCatInputFormat 用于提供接口给MapReduce程序，用于将数据写入Hive表。
  - setOutput
  - setSchema
  - getTableSchema

HCatalog开发讲解

Map实现

```
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));
        }    }
```

HCatalog开发讲解

Reduce实现

```
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,
              IntWritable, HCatRecord>.Context context)
              throws IOException, InterruptedException {
          int sum = 0;
          Iterator<IntWritable> iter = values.iterator();
```

HCatalog开发讲解

Reduce实现

```
while (iter.hasNext()) {
                sum++;
                iter.next();
            }
            HCatRecord record = new DefaultHCatRecord(2);
            record.set(0, key.get());
            record.set(1, sum);
            context.write(null, record);
        }
    }
```

HCatalog开发讲解

设置**InputFormat**

设置**OutputValueClass**

```
        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);
```

HCatalog开发讲解

Job提交

```
HCatSchema schema =
    outputjobInfo.getOutputSchema();
        HCatOutputFormat.setSchema(job,
    schema);

    job.setOutputFormatClass(HCatOutputFormat.c
    lass);
        return (job.waitForCompletion(true) ?
    0 : 1);
```

WebHCat开发讲解1

向Hive提交查询示例

```
% curl -s -d execute="select+*+from+pokes;" \ -d
    statusdir="pokes.output" \ 'http:// 10.64.35.143:21055
    /templeton/v1/hive?user.name=ekoifman'
```

返回结果是JSON格式的

```
{
"id": "job_201111111311_0005",
"info": {
        "stdout": "templeton-job-id:job_201111111311_0005 ",
         "stderr": "",
         "exitcode": 0
          }
 }
```

WebHCat开发讲解2

安全配置：
krb5,keytab,principle

```
System.setProperty("java.security.krb5.conf",
    "D:/krb/krb5.conf");
System.setProperty("java.security.krb5.keytab“,"D:/krb/hive.ke
    ytab");
System.setProperty("java.security.krb5.principal“,"hive/hadoop
    .hadoop.com@HADOOP.COM");
```

获取**cookie**

```
AuthenticatedURL.Token newToken = new
    AuthenticatedURL.Token();
WebHCatAuthenticator authenticator = new
    WebHCatAuthenticator();
URL url = new
    URL("http://ip:21055/templeton/v1/ddl/database/default/ta
    ble/t1?format=extended");
```

WebHCat开发讲解2

使用**cookie**进行连接

```
conn = new
    AuthenticatedURL(authenticator).openConnection(url,
    newToken);
conn.connect();
```

建立连接后，读取数据

```
StringBuilder sb = new StringBuilder();
    is = conn.getInputStream();
    reader = new BufferedReader(new InputStreamReader(is,
    "utf-8"));
    String line = null;
```

WebHCat开发讲解2

拼接结果

```
while ((line = reader.readLine()) != null) {
    sb.append(line);
}
System.out.println(sb);
```

打印结果

WebHCat开发- 示例程序运行

- 安装**Eclipse**。
- 创建**Eclipse**工程并引入相应的**jar**。
- 编写**WebHCat Java**类。
- 选中**main**函数，右键点击**Run As →Java Application**。
- 成功后会打印出语句的执行结果。

本章小结

本章主要学习了使用不同的**Hive**接口调用**Hive**服务。这些接口包括**JDBC**，**ODBC**，**Python**客户端，**Hcatlog**，**WebHcat**等。通过学习，可以基本掌握使用这些接口进行开发调试。

目录

HQL示例-建表

```
-- 创建外部表employees_info.
CREATE TABLE IF NOT EXISTS employees_info
( id INT,
name STRING,
usd_flag STRING,
salary DOUBLE,
deductions MAP<STRING, DOUBLE>,
address STRING,
entrytime STRING )

-- 使用CREATE Like创建表.
CREATE TABLE employees_like LIKE employees_info;

-- 使用DESCRIBE查看表结构.
DESCRIBE employees_info;
DESCRIBE employees_like;
```

HQL示例-加载数据

-- 从本地文件系统/opt/hive_examples_data/目录下将employee_info.txt加载进employees_info表中.

```
LOAD DATA LOCAL INPATH '/opt/hive_examples_data/employee_info.txt'
OVERWRITE INTO TABLE employees_info;
```

-- 从HDFS上/user/hive_examples_data/employee_info.txt加载进employees_info表中.

```
LOAD DATA INPATH '/user/hive_examples_data/employee_info.txt'
OVERWRITE INTO TABLE employees_info;
```

HQL示例-查询

```
-- 查看薪水支付币种为美元的雇员联系方式.
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';

-- 查询入职时间为2014年的雇员编号、姓名等字段，并将查询结果加载进表employees_info_extended中的入职时间为2014的分区中.
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';
```

UDF示例

UDF:用户自定义函数。其创建使用过程如下：

1. 编写**UDF**，并打包成**jar**。
2. 将**jar**上传至**HDFS**，如“**/user/hive_examples_jars/**”。
3. 由管理员在**Hive**系统中创建永久**UDF**。
4. 用户使用**UDF**。

```
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;
    // 处理逻辑部分.
    for (int i = 0; i < a.length; i++)
      if (a[i] != null)
        total += a[i];
    return total;
  }
}
```

常用客户端参数

- 引擎选择
  hive.execution.engine 可选**mr**或**spark**

- 并行度设定
  hive.exec.reducers.bytes.per.reducer 默认**256 * 1000 * 1000**
  hive.exec.reducers.max 默认 **999**

- 动态分区
  hive.exec.dynamic.partition.mode 默认**false**
  hive.exec.max.dynamic.partitions 默认**1000**
  hive.exec.max.dynamic.partitions.pernode 默认**100**

- 压缩

hive.exec.compress.intermediate 默认**false**
hive.intermediate.compression.codec 建议选择**Snappy**即
org.apache.hadoop.io.compress.SnappyCodec

规则建议

场景	规则
开发调试	在开发程序时，可通过使用**Hive**的客户端**Beeline**先进行调试，检验语句与结果正确性，再部署基于**JDBC**等的应用程序。
获取数据库连接	Hive 的数据库URL在拼接时已经经过安全认证，所以Hive数据库的用户名和密码为null或者空。 connection = DriverManager.getConnection(url, "", "");
JDBC超时限制	Hive提供的JDBC实现有超时限制，默认是5分钟，用户可以通过java.sql.DriverManager.setLoginTimeout(int seconds)设置，seconds的单位为秒。

规则建议

场景	规则
执行HQL	在JAVA JDBC应用开发中，拼装HQL语句，注意HQL语句不能以“;”结尾，以下为正确示例: String sql = "SELECT COUNT(*) FROM employees_info"
HQL语法规则之判空	判断字段是否为“空”，即没有值，使用“is null”；判断不为空，即有值，使用“is not null”。 要注意的是，在HQL中String类型的字段若是空字符串，即长度为0，那么对它进行IS NULL的判断结果是False。此时应该使用col = '' "来判断空字符串。

规则建议

场景	规则
UDF的管理	建议由管理员创建永久**UDF**，避免每次使用时都去**add jar**，和重新定义**UDF**。示例： **CREATE FUNCTION [db_name.]function_name AS class_name** **[USING JAR|FILE|ARCHIVE 'file_uri' [, JAR|FILE|ARCHIVE 'file_uri']];**
UDF的注解	Hive的UDF会有一些默认属性，比如deterministic 默认为true(同一个输入会返回同一个结果)，stateful(是否有状态，默认为true)。当用户实现的自定义UDF内部实现了汇总等，需要在类上加上相应的注解，比如如下类 @UDFType(deterministic = false) Public class MyGenericUDAFEvaluator implements Closeable {

规则建议

场景	规则

使用分区表	当数据量较大，且经常需要按天统计时，建议使用分区表，按天存放数据。
	分区表创建示例：
	`create table store_sales`
	`(`
	`ss_sold_time_sk int,`
	`…`
	`)`
	`partitioned by (ss_sold_date_sk int)`

动态分区表	为了避免在插入动态分区数据的过程中，产生过多的小文件，在执行插入时，在分区字段上加上distribute by.
	示例：
	`insert overwrite table store_sales partition (ss_sold_date_sk)`
	`select ss.ss_sold_date_sk`
	`from ${SOURCE}.store_sales ss`
	`distribute by ss.ss_sold_date_sk;`

规则建议

场景	规则
文件格式选择	Hive支持多种存储格式，比如TextFile，RCFile，ORC，Sequence，Parquet等。为了节省存储空间，或者大部分时间只查询其中的一部分字段时，可以在建表时使用列式存储(比如ORC文件)。使用方法如下 create table store_sales (ss_sold_time_sk　int, …) stored as orc

规则建议

场景	规则

HQL编写之隐式类型转换	查询语句使用字段的值做过滤时，不建议通过Hive自身的隐式类型转换来编写HQL。因为隐式类型转换不利于代码的阅读和移植。
	建议示例：
	select * from default.tbl_src where id = 10001;
	select * from default.tbl_src where name = 'HuangShuang';
	不建议示例：
	select * from default.tbl_src where id = '10001';
	select * from default.tbl_src where name = HuangShuang;

本章小结

本章主要介绍了如何实现业务开发，比如如何开发**HQL**，如何实现**UDF**，同时介绍了一些开发规则以及客户端常用的参数。

通过这些学习，可以掌握**Hive**业务开发的常用语句，以及通用技巧。

思考题

- 问答

1. 在哪里可以获取**Hive**的**JDBC**二次开发示例?
2. **Hive**的**UDF**是否需要每次使用时都重新创建?
3. 请列举出至少**3**种**Hive**支持的文件格式。
4. 如果用户的**UDF**实现中，做了汇总累加等操作，开发这种**UDF**需要做什么特殊处理?

习题

- 选择

  1. 以下哪些是**Hive**支持的文件格式？ **A. RCFile B. ORC C. HFile D. TextFile**
  2. 以下哪些是**Hive**支持的表类型？ **A.** 分区表 **B.**分桶表 **C.**倾斜表 **D.**外部表

Thank You