大数据技术基础

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

第四章 大数据处理
—Spark SQL

鄂海红 计算机学院 教授
ehaihong@bupt.edu.cn 微信: 87837001 QQ: 3027581960

大数据处理—Spark框架

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

大数据处理—Spark框架

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

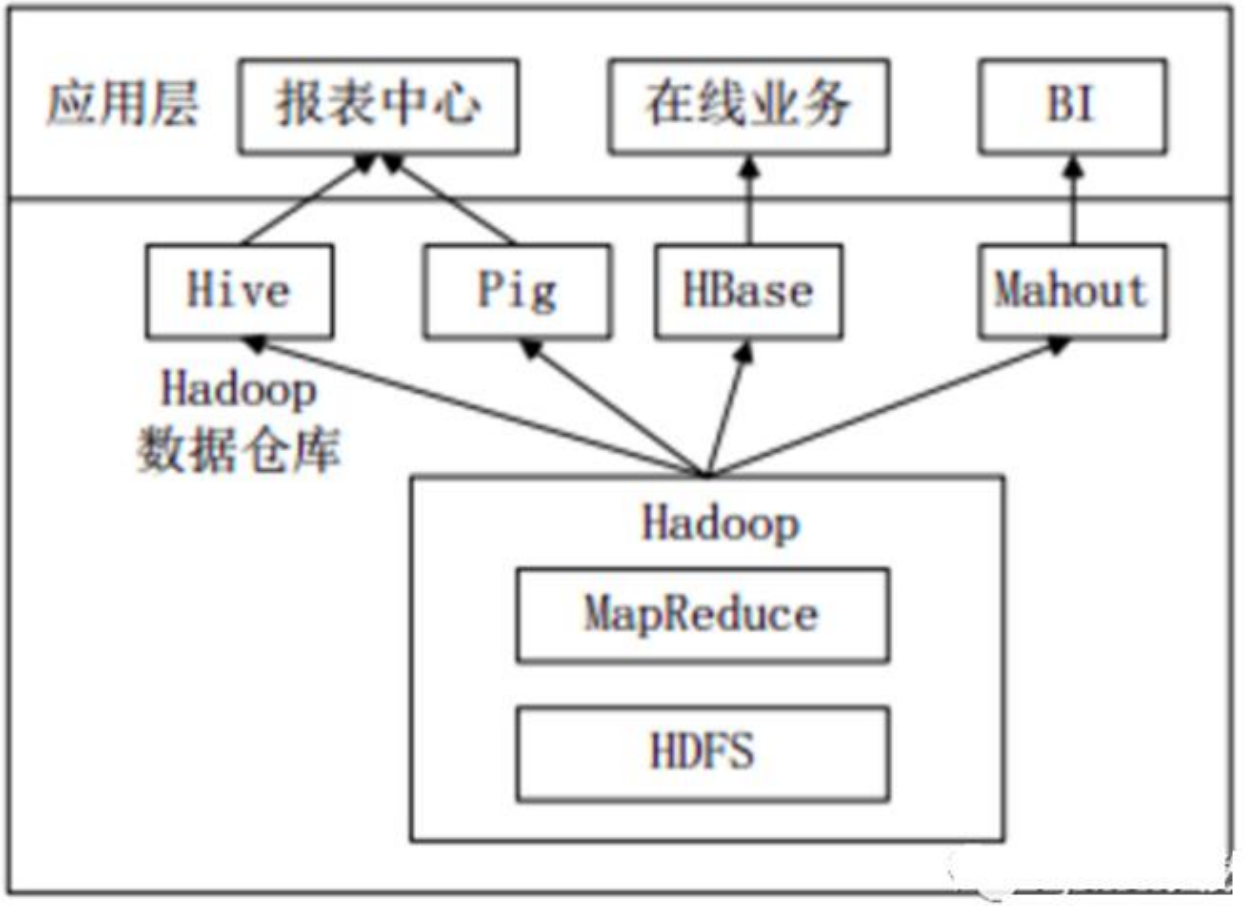
Spark SQL简介

从Hive和Shark说起

Hive: SQL-on-Hadoop

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

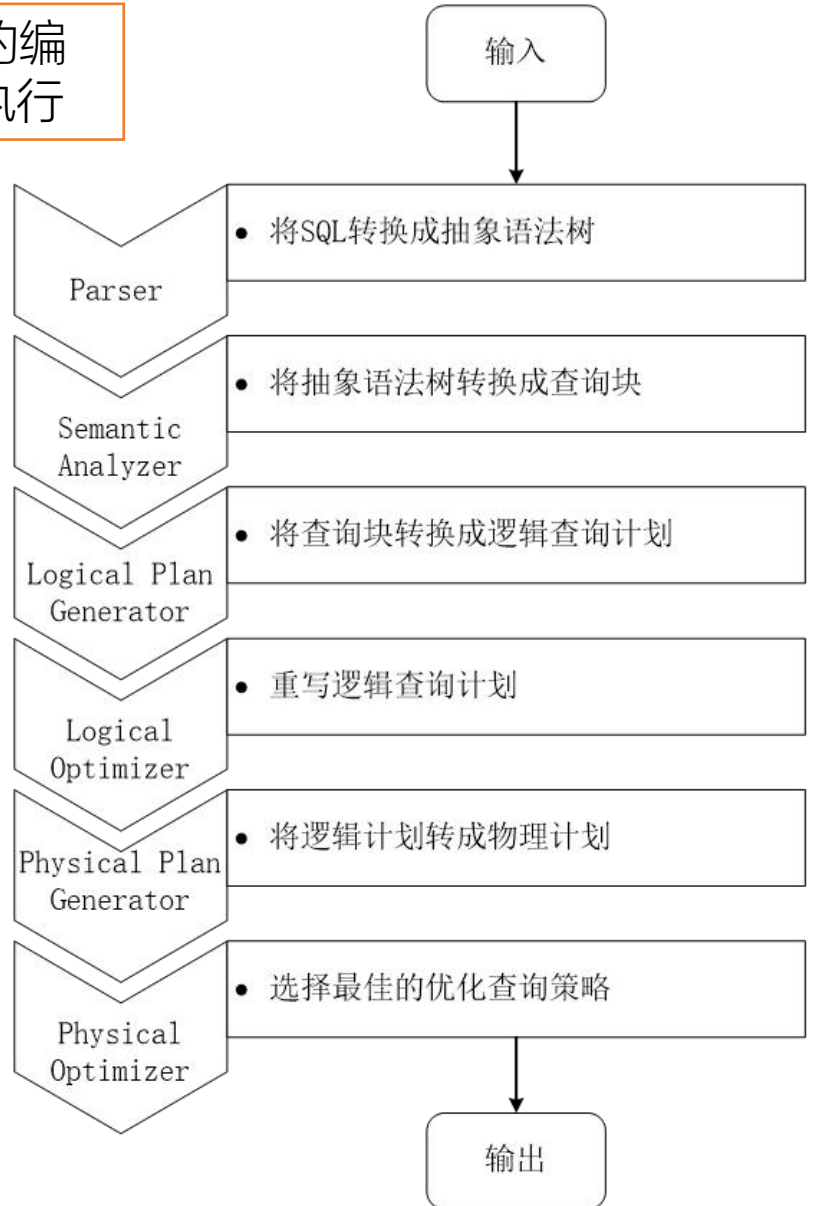
- Hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供完整的 SQL查询功能，可以将SQL语句转换为 MapReduce 任务进行运行
- 其优点是学习成本低，可以通过类 SQL 语句快速实现简单的 MapReduce 统计，不必开发专门的 MapReduce 应用，十分适合数据仓库的统计分析
- Spark未出现之前，企业中部署的大数据分析平台，组合使用了Hadoop的HDFS、MR、Hive、Pig、HBase、Mahout，从而满足不同业务场景需求



Hive中SQL查询转化成MapReduce作业过程

当Hive接收到一条HQL语句后，HQL首先进入驱动模块，由驱动模块中的编译器解析编译，并由优化器对该操作进行优化计算，然后交给执行器去执行

1. 由驱动模块中的编译器，对用户输入的SQL语句进行词法和语法解析，将HQL语句转换成抽象语法树（AST Tree）的形式
2. 因为AST结构复杂，不方便直接翻译成MR算法程序。语法分析器遍历抽象语法树，转化成QueryBlock查询单元。QueryBlock是一条最基本的SQL语法组成单元，包括输入源、计算过程、和输入三个部分
3. 逻辑计划生成器遍历QueryBlock，生成OperatorTree（操作树），它由很多逻辑操作符组成，如TableScanOperator、SelectOperator、FilterOperator、JoinOperator、GroupByOperator和ReduceSinkOperator等，它们在MR阶段完成某一特定操作
4. Hive驱动模块中的逻辑优化器对OperatorTree进行优化，变换OperatorTree的形式，合并多余的操作符，减少MR任务数、以及Shuffle阶段的数据量
5. Hive驱动模块中的物理计划生成器，遍历优化后的OperatorTree，根据OperatorTree中的逻辑操作符生成需要执行的MR任务
6. 启动Hive驱动模块中的物理优化器，对生成的MR任务进行优化，生成最终的MR任务执行计划
7. 最后由Hive驱动模块中的执行器，对最终的MR任务执行输出

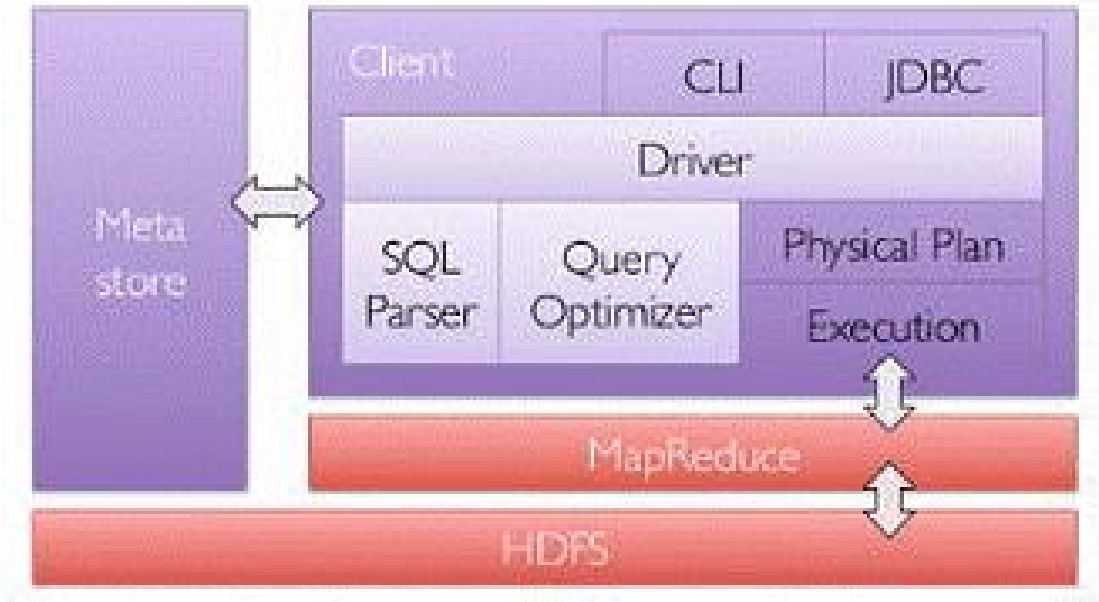


Shark

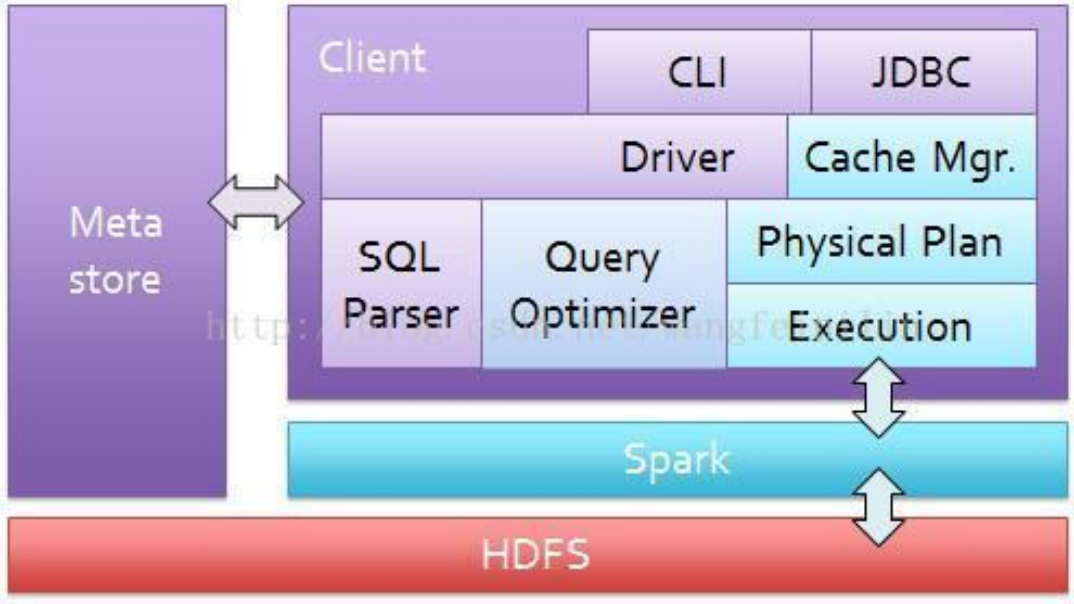
Shark：Hive on Spark

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- Shark提供了类似Hive的功能，与Hive不同的是Shark把SQL语句转化成Spark作业
- Shark为了实现Hive兼容，在HQL方面重用了Hive中HQL的解析、逻辑执行计划翻译、执行计划优化等逻辑
- 可以近似认为仅将物理执行计划从MR作业替换成了Spark作业（辅以内存列式存储等各种和Hive关系不大的优化），也就是通过Hive的HiveSQL解析功能，把HSQL翻译成Spark上的RDD操作
- Shark的出现，使得SQL-on-Hadoop的性能比Hive有了10~100倍的提高



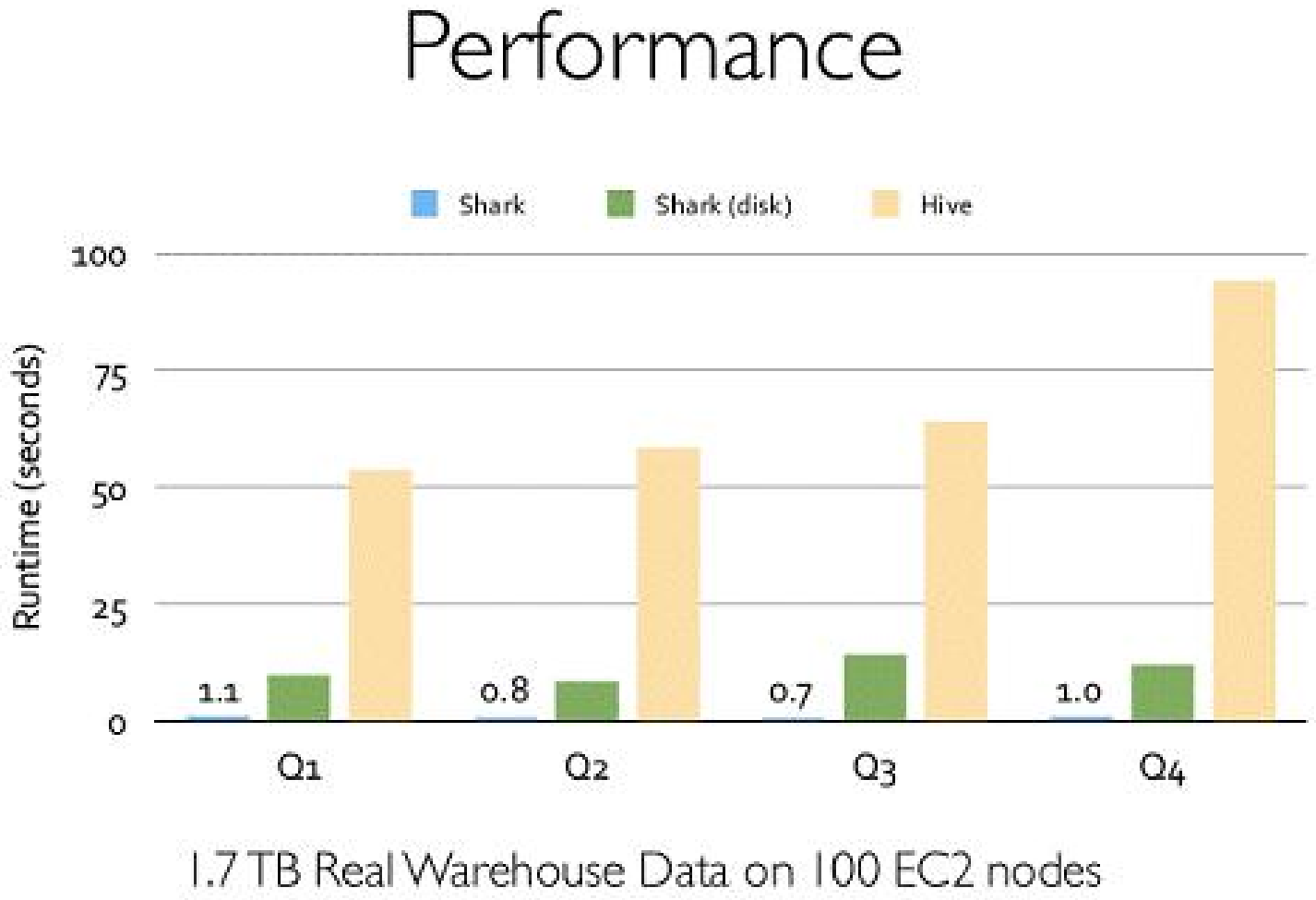
Hive体系结构



Shark体系结构

从Shark说起

- Shark的出现，使得SQL-on-Hadoop的性能比Hive有了10-100倍的提高

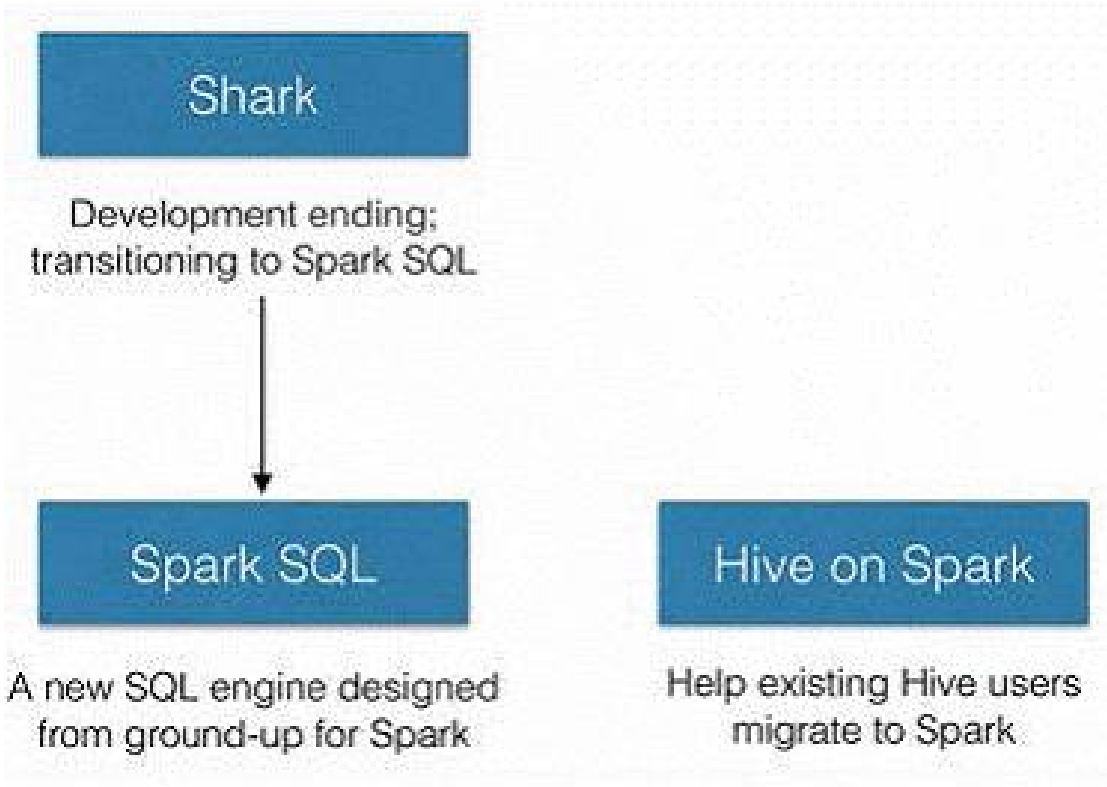


从Shark说起

- Shark的设计导致了两个问题：
  - 一是执行计划优化完全依赖于Hive，不方便添加新的优化策略
  - 二是因为Spark是线程级并行，而MapReduce是进程级并行，因此，Spark在兼容Hive的实现上存在线程安全问题，导致Shark不得不使用另外一套独立维护的打了补丁的Hive源码分支

从Shark说起

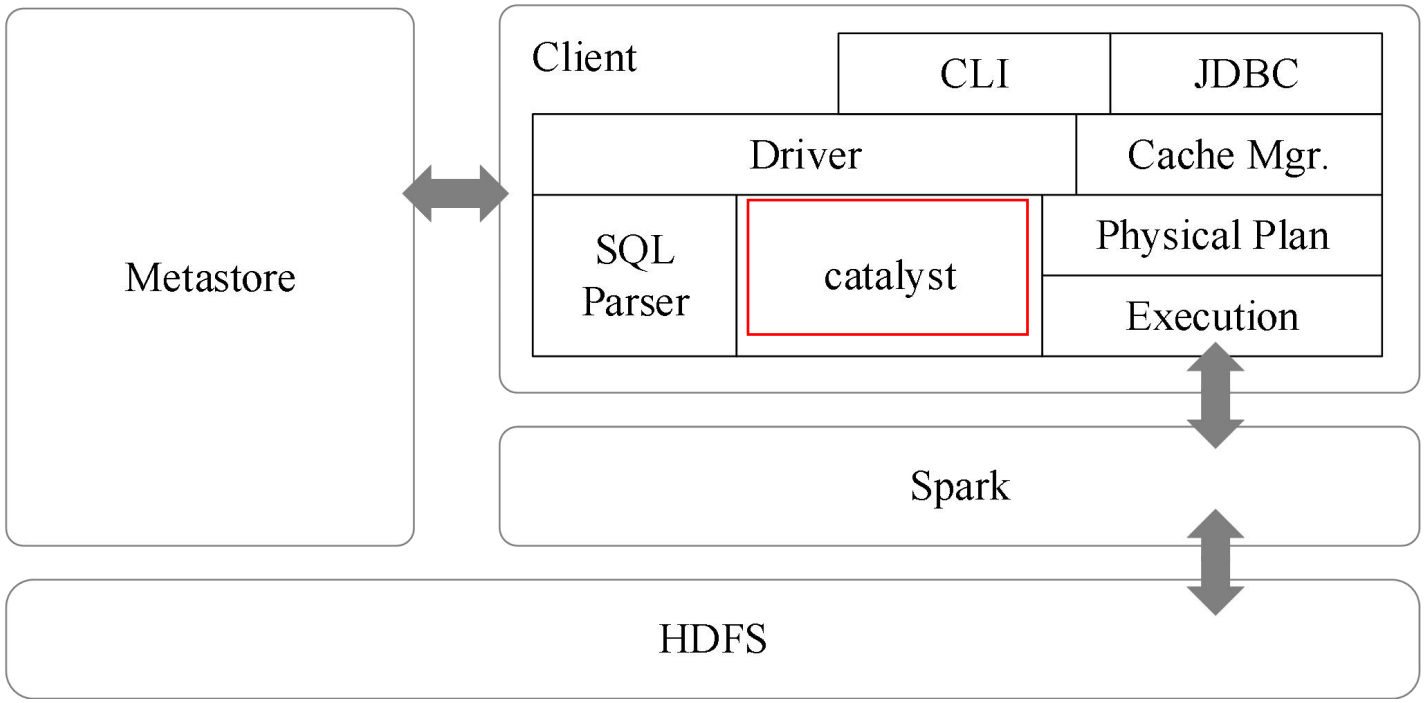
- 2014年6月1日Shark项目和Spark SQL项目的主持人Reynold Xin宣布：停止对Shark的开发，团队将所有资源放在Spark SQL项目上，至此，Shark的发展画上了句话，但也因此发展出两个直线：Spark SQL和Hive on Spark
- Spark SQL作为Spark生态的一员继续发展，而不再受限于Hive，只是兼容Hive
- Hive on Spark是一个Hive的发展计划，该计划将Spark作为Hive的底层引擎之一，也就是说，Hive将不再受限于一个引擎，可以采用Map-Reduce、Spark等引擎



Spark SQL架构

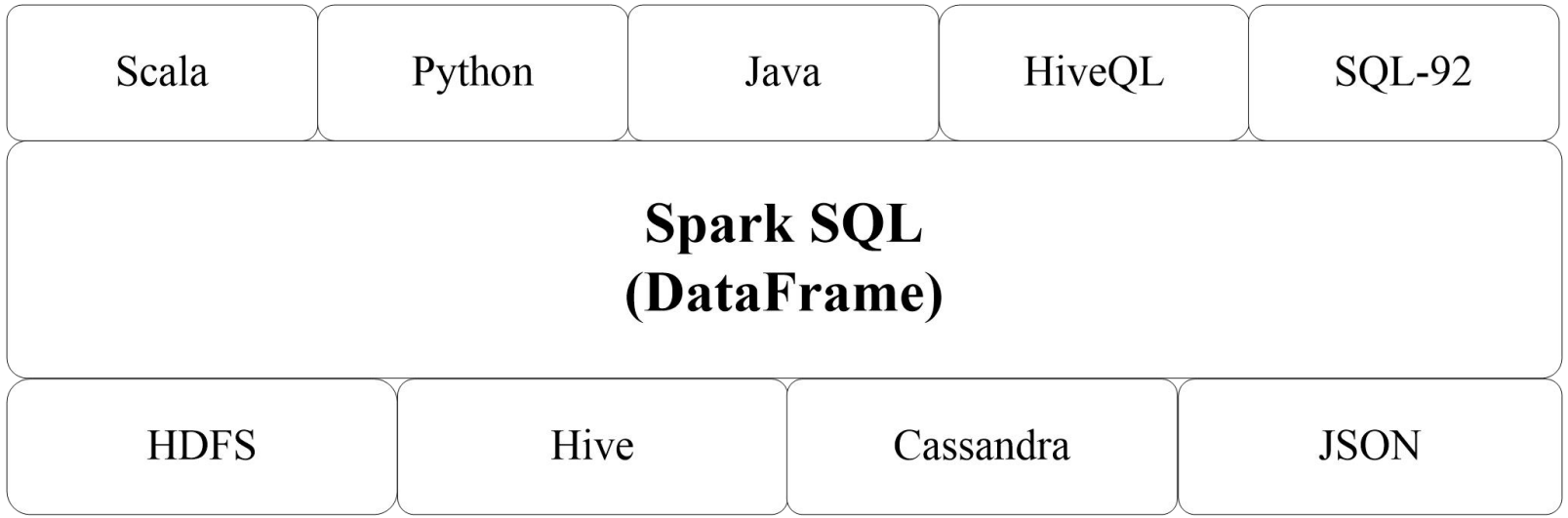
北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- Spark SQL在Hive兼容层面仅依赖HiveQL解析、Hive元数据
- 也就是说，从HQL被解析成抽象语法树（AST）起，就全部由Spark SQL接管了
- Spark SQL执行计划生成和优化都由Catalyst（函数式关系查询优化框架）负责



Spark SQL设计

- Spark SQL增加了DataFrame（即带有Schema信息的RDD），使用户可以在Spark SQL中执行SQL语句
- 数据既可以来自RDD，也可以是Hive、HDFS、Cassandra等外部数据源，还可以是JSON格式的数据
- Spark SQL目前支持Scala、Java、Python三种语言，支持SQL-92规范



Spark SQL支持的数据格式和编程语言

为什么推出Spark SQL

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Spark SQL: Relational Data Processing in Spark

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†], Xiangrui Meng[†], Tomer Kaftan[‡], Michael J. Franklin[†‡], Ali Ghodsi[†], Matei Zaharia[†*]

[†]Databricks Inc. [*]MIT CSAIL [‡]AMPLab, UC Berkeley

While the popularity of relational systems shows that users often prefer writing declarative queries, the relational approach is insufficient for many big data applications. First, users want to perform ETL to and from various data sources that might be semi- or unstructured, requiring custom code. Second, users want to perform advanced analytics, such as machine learning and graph processing, that are challenging to express in relational systems. In practice, we have observed that most data pipelines would ideally be expressed with a combination of both relational queries and complex procedural algorithms. Unfortunately, these two classes of systems—relational and procedural—have until now remained largely disjoint,

为什么推出Spark SQL

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Spark SQL: Relational Data Processing in Spark

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†], Xiangrui Meng[†], Tomer Kaftan[†], Michael J. Franklin[†‡], Ali Ghodsi[†], Matei Zaharia[†*]

Spark SQL bridges the gap between the two models through two contributions. First, Spark SQL provides a *DataFrame API* that can perform relational operations on both external data sources and Spark's built-in distributed collections. This API is similar to the widely used data frame concept in R [32], but evaluates operations lazily so that it can perform relational optimizations. Second, to support the wide range of data sources and algorithms in big data, Spark SQL introduces a novel extensible optimizer called *Catalyst*. Catalyst makes it easy to add data sources, optimization rules, and

为什么推出Spark SQL

- 关系数据库已经很流行，但关系数据库在大数据时代已经不能满足要求
  - 用户需要从不同数据源执行各种操作，包括结构化、半结构化和非结构化数据
  - 用户需要执行高级分析，比如机器学习和图像处理
- 在实际大数据应用中，经常需要融合关系查询和复杂分析算法（比如机器学习或图像处理），但是，缺少这样的系统
- Spark SQL填补了这个鸿沟：
  - 可以提供DataFrame API，可以对内部和外部各种数据源执行各种关系型操作
  - 可以支持大数据中的大量数据源和数据分析算法
  - Spark SQL可以融合：传统关系数据库的结构化数据管理能力和机器学习算法的数据处理能力

大数据处理—Spark框架

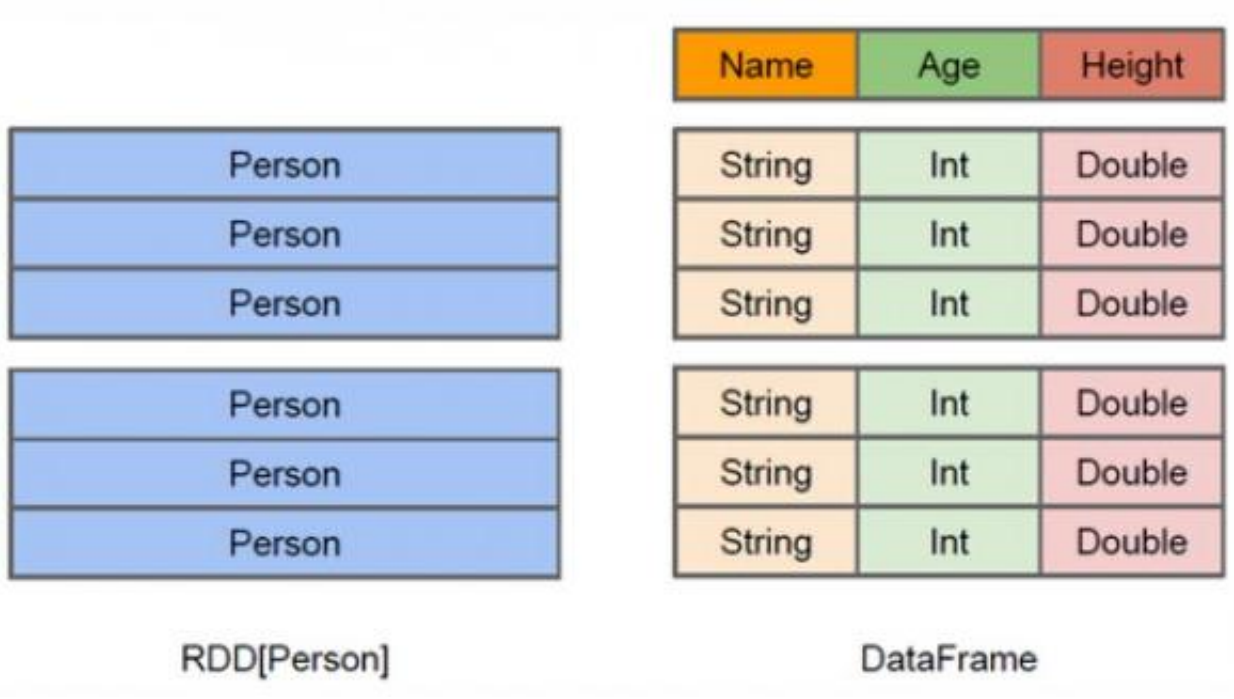
北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

DataFrame介绍及常用操作

DataFrame概述

- DataFrame的推出，让Spark具备了处理大规模结构化数据的能力，不仅比原有的RDD转化方式更加简单易用，而且获得了更高的计算性能
- Spark能够轻松实现从MySQL到DataFrame的转化，并且支持SQL查询
- RDD是分布式的 Java对象的集合，但是，对象内部结构对于RDD而言却是不可知的
- DataFrame是一种以RDD为基础的分布式数据集，提供了详细的结构信息

DataFrame与RDD的区别



DataFrame的创建

- 从Spark2.0以上版本开始，Spark使用全新的SparkSession接口替代Spark1.6中的SQLContext及HiveContext接口来实现其对数据加载、转换、处理等功能。
- SparkSession实现了SQLContext及HiveContext所有功能
- SparkSession支持从不同的数据源加载数据，并把数据转换成DataFrame，并且支持把DataFrame转换成SQLContext自身中的表，然后使用SQL语句来操作数据
- SparkSession亦提供了HiveQL以及其他依赖于Hive的功能的支持
- 可以通过如下语句创建一个SparkSession对象

```
scala> import org.apache.spark.sql.SparkSession
scala> val spark=SparkSession.builder().getOrCreate()
```

DataFrame的创建

- 在创建DataFrame之前，为了支持RDD转换为DataFrame及后续的SQL操作，需要通过import语句（即import spark.implicits._）导入相应的包，启用隐式转换
- 在创建DataFrame时，可以使用spark.read操作，从不同类型的文件中加载数据创建DataFrame，例如：
  - spark.read.json("people.json")：读取people.json文件创建DataFrame；在读取本地文件或HDFS文件时，要注意给出正确的文件路径；或spark.read.format(“json").load(“people. json”)
  - spark.read.parquet("people.parquet")：读取people.parquet文件创建DataFrame；或spark.read.format(“parquet") .load(“people.parquet”)
  - spark.read.csv(“people.csv”)：读取people.csv文件创建DataFrame 或spark.read.format(“csv").load(“people.csv”)

DataFrame的创建

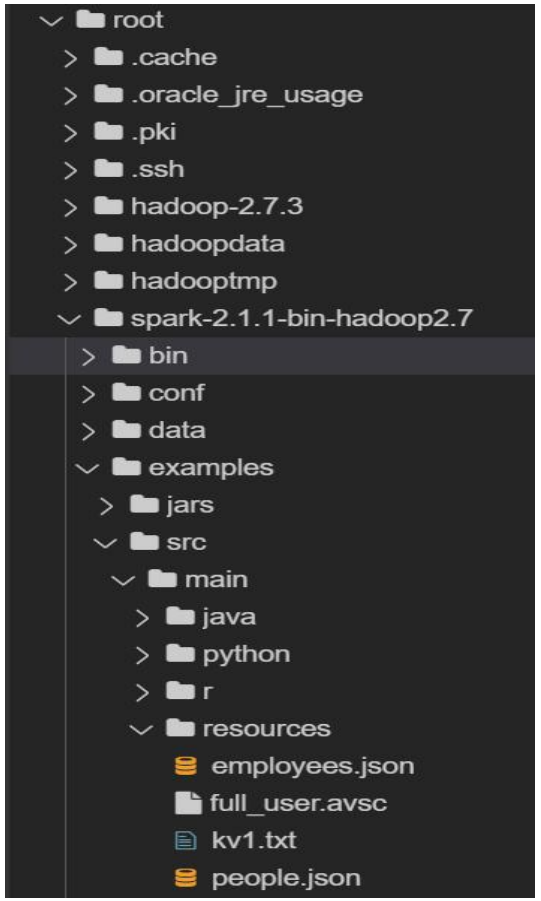
- **一个实例**
- 在"/root/spark-2.1.1-bin-hadoop2.7/examples/src/main/resources/"这个目录下，这个目录下有两个样例数据people.json和people.txt。

people.json文件的内容如下：

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

people.txt文件的内容如下：

```
Michael, 29
Andy, 30
Justin, 19
```



DataFrame的创建


```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> val spark=SparkSession.builder().getOrCreate()
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@2bdab835

//导入包，支持把一个RDD隐式转换为一个DataFrame
scala> import spark.implicits._
import spark.implicits._

scala> val df = spark.read.json("file:///root/spark-2.1.1-bin-hadoop2.7/examples/src/main/resources/people.json")
df: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> df.show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

DataFrame的保存

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 可以使用spark.write操作，把一个DataFrame保存成不同格式的文件，例如，把一个名称为df的DataFrame保存到不同格式文件中，方法如下：
- df.write.json("people.json")
- df.write.parquet("people.parquet")
- df.write.csv("people.csv") 或 df.write.format("csv").save("people.csv")
- 下面从示例文件people.json中创建一个DataFrame，然后保存成csv格式文件，代码如下：

```
scala> val peopleDF = spark.read.format("json").
| load("file:///root/spark-2.1.1-bin-hadoop2.7/examples/src/main/resources/people.json")
scala> peopleDF.select("name", "age").write.format("csv").
| save("file:///root/spark-2.1.1-bin-hadoop2.7/spark/mycode/sql/newpeople.csv")
```

DataFrame的常用操作

- printSchema()打印DataFrame的模式（Schema）信息
- select()从DataFrame中选取部分列的数据，还可同时修改列名，对列值做操作
- filter()对DataFrame做条件查询，找到满足条件要求的记录

```
// 打印模式信息
scala> df.printSchema()
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)

// 选择多列
scala> df.select(df("name"),df("age")+1).show()
+-------+---------+
|   name|(age + 1)|
+-------+---------+
|Michael|     null|
|   Andy|       31|
| Justin|       20|
+-------+---------+

// 条件过滤
scala> df.filter(df("age") > 20 ).show()
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

DataFrame的常用操作

- groupBy()操作用于对记录进行分组；结合count()做分组计数
- sort()对记录进行排序
- df .sort(df("age").desc)表示对age字段进行降序排序；

```
// 分组聚合
scala> df.groupBy("age").count().show()
+----+-----+
| age|count|
+----+-----+
|  19|    1|
|null|    1|
|  30|    1|
+----+-----+

// 排序
scala> df.sort(df("age").desc).show()
+----+-------+
| age|   name|
+----+-------+
|  30|   Andy|
|  19| Justin|
|null|Michael|
+----+-------+
```

DataFrame的常用操作

- df .sort(df(“age”).desc, df(“name”).asc)表示对age字段进行降序排序，当age相同时再根据name字段进行升序排序
- select(df(“name”).as(“username”)，从DataFrame中选取name列数据，并修改列名为username

```
//多列排序
scala> df.sort(df("age").desc, df("name").asc).show()
+----+-------+
| age|   name|
+----+-------+
|  30|   Andy|
|  19| Justin|
|null|Michael|
+----+-------+

//对列进行重命名
scala> df.select(df("name").as("username"),df("age")).show()
+--------+----+
|username| age|
+--------+----+
| Michael|null|
|    Andy|  30|
|  Justin|  19|
+--------+----+
```

大数据处理—Spark框架

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

从RDD转换得到DataFrame

从RDD转换得到DataFrame及模式

- Spark官网提供了两种方法来实现从RDD转换得到DataFrame
- 第一种方法是，利用反射机制推断RDD模式
  - 利用反射来推断包含特定类型对象的RDD的schema，适用对已知数据结构的RDD转换
- 第二种方法是，使用编程方式定义RDD模式
  - 使用编程接口，构造一个schema并将其应用在已知的RDD上

利用反射机制推断RDD模式-1

- 在“/usr/local/spark/examples/src/main/resources/”目录下，有个Spark安装时自带的样例数据people.txt，其内容如下：
- 现在要把people.txt加载到内存中生成一个DataFrame，并查询其中的数据

```
Michael, 29
Andy, 30
Justin, 19
```

具体代码讲解：

- 1、首先通过import语句导入所需的包

```
scala> import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
scala> import org.apache.spark.sql.Encoder
import org.apache.spark.sql.Encoder
scala> import spark.implicits._  //导入包，支持把一个RDD隐式转换为一个DataFrame
import spark.implicits._
```

剩余代码
见下一页

利用反射机制推断RDD模式-2

- 2、定义了一个名称为Person的case class；在利用反射机制推断RDD模式时，需要首先定义一个case class；因为，只有case class才能被Spark隐式地转换为DataFrame

```
Michael, 29
Andy, 30
Justin, 19
```

```
scala> case class Person(name: String, age: Long)  //定义一个case class
defined class Person
scala> val peopleDF = spark.sparkContext.
| textFile("file:///root/spark-2.1.1-bin-hadoop2.7/examples/src/main/resources/people.txt").
| map(_.split(",")).
| map(attributes => Person(attributes(0), attributes(1).trim.toInt)).toDF()
peopleDF: org.apache.spark.sql.DataFrame = [name: string, age: bigint]
```

- 3、spark.sparkContext.textFile()执行后，系统会把people.txt文件加载到内存中生成一个RDD，每个RDD都是string类型，3个元素分别为“Michael, 29“、”Andy, 30“、”Justin, 19“
- 然后这个RDD调用.map(_.split(“,”))方法得到一个新的RDD，这个RDD中的3个元素分别是Array(“Michael“, “29”)、 Array(”Andy“, “30”)、 Array(”Justin“, “19”)
- 对RDD执行map(attributes => Person(attributes(0), attributes(1).trim.toInt))操作，得到新的RDD，每个元素是一个Person对象，3个元素分别是Person(” Michael”, 29) Person(” Andy”, 30) Person(” Justin”, 19)
- 在这个RDD上执行.toDF()操作，把RDD转换为DataFrame；执行后返回结果可以看到，新生成名称为peopleDF的DataFrame，每条记录的模式信息是[name: string, age: bigint]

剩余代码见下一页

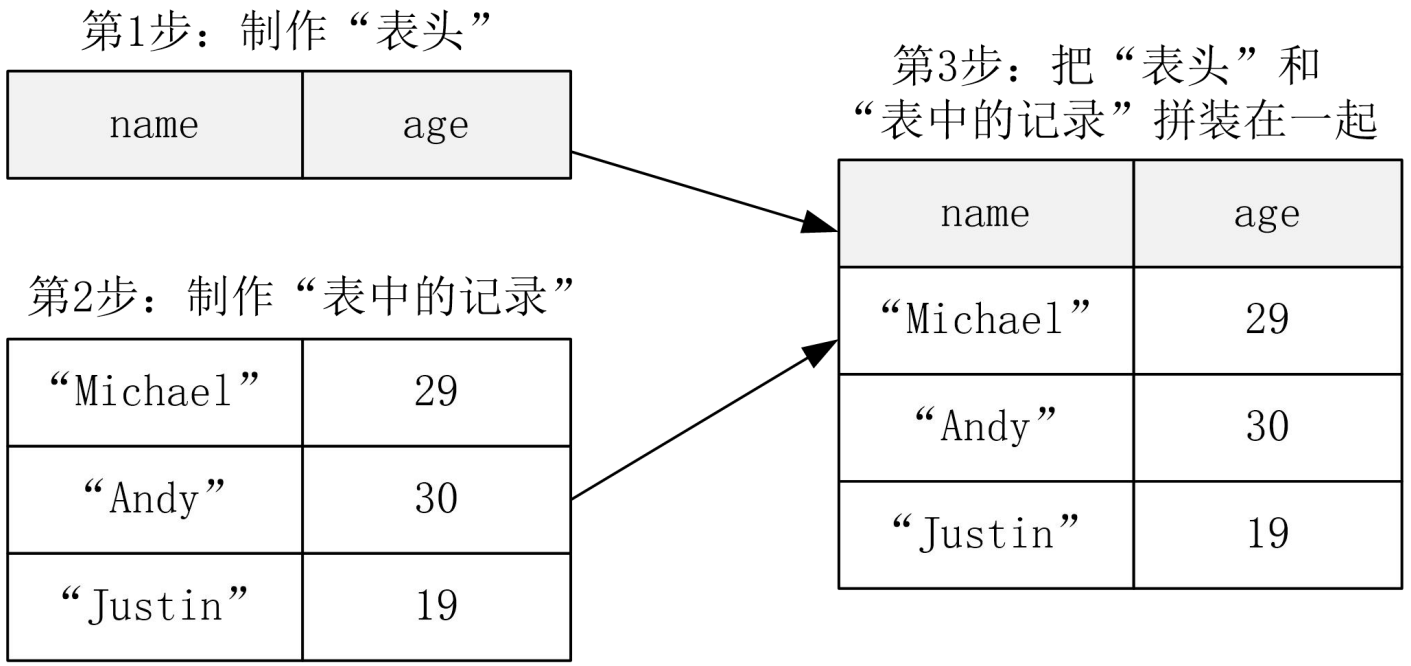
利用反射机制推断RDD模式-3

- 4、生成DataFrame之后，可以进行SQL查询。Spark要求必须把DataFrame注册为临时表才能供后面的查询使用
- peopleDF.createOrReplaceTempView("people")，把系统会把peopleDF注册为临时表people
- 执行查询语句spark.sql("select name,age from people where age > 20")，得到返回值personsRDD是一个DataFrame
- 最后通过personsRDD.map(t => "Name: "+t(0)+ ","+"Age: "+t(1)).show() 操作，把其中元素格式化输出

```
scala> peopleDF.createOrReplaceTempView("people") //必须注册为临时表才能供下面的查询使用
scala> val personsRDD = spark.sql("select name,age from people where age > 20")
//最终生成一个DataFrame，下面是系统执行返回的信息
personsRDD: org.apache.spark.sql.DataFrame = [name: string, age: bigint]
scala> personsRDD.map(t => "Name: "+t(0)+ ","+"Age: "+t(1)).show()
//DataFrame中的每个元素都是一行记录，包含name和age两个字段，分别用t(0)和t(1)来获取值
//下面是系统执行返回的信息
+-----------------+
| value|
+-----------------+
|Name:Michael,Age:29|
| Name:Andy,Age:30|
+-----------------+
```

使用编程方式定义RDD模式-1

- 当无法提前定义case class时，就需要采用编程方式定义RDD模式。
- 比如，现在需要通过编程方式把people.txt加载进来生成DataFrame，并完成SQL查询



通过编程方式定义RDD模式的实现过程

- “表头”是表的schema模式，需要包含字段名称、字段类型和是否允许空值等信息
- SparkSQL提供StructField(name,datatype,nullable)是用来表示表的字段信息的，名称、字段数据类型、是否允许为空，例如：
  StructField("name",StringType,true)
  StructField("age",IntegerType,true)
- SparkSQL提供StructType(fields:Seq[StructField])类来表示表的模式信息
- 生成一个StructType对象时，需要提供fields作为参数，例如：
  val schema = StructType(fields)
- Fields是一个集合类型，里面的每个集合元素都是StructField类型

使用编程方式定义RDD模式-2

第1步：制作“表头”

name	age

```
scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._
scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row
//生成字段
scala> val fields = Array(StructField("name",StringType,true),
StructField("age",IntegerType,true))
fields: Array[org.apache.spark.sql.types.StructField] =
Array(StructField(name,StringType,true),
StructField(age,IntegerType,true))
scala> val schema = StructType(fields)
schema: org.apache.spark.sql.types.StructType =
StructType(StructField(name,StringType,true), StructField(age,
IntegerType,true))
//从上面信息可以看出，schema描述了模式信息，模式中包含name和age
两个字段
//shcema就是“表头”
```

剩余代码见下一页

使用编程方式定义RDD模式-3

- 表中的记录每条都要封装到Row对象中，并把所有记录的Row对象一起保存在RDD中

第2步：制作“表中的记录”

“Michael”	29
“Andy”	30
“Justin”	19

```
//下面加载文件生成RDD
scala> val peopleRDD = spark.sparkContext.
| textFile("file:///root/spark-2.1.1-bin-hadoop2.7/examples/src/main/resources/people.txt")
peopleRDD: org.apache.spark.rdd.RDD[String] =
file:///usr/local/spark/examples/src/main/resources/people.txt MapPartitionsRDD[1] at
textFile at <console>:26

//对peopleRDD 这个RDD中的每一行元素都进行解析
scala> val rowRDD = peopleRDD.map(_.split(",")).
|  map(attributes => Row(attributes(0), attributes(1).trim.toInt))
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[3]
at map at <console>:29

//上面得到的rowRDD就是“表中的记录”
```

剩余代码见下一页

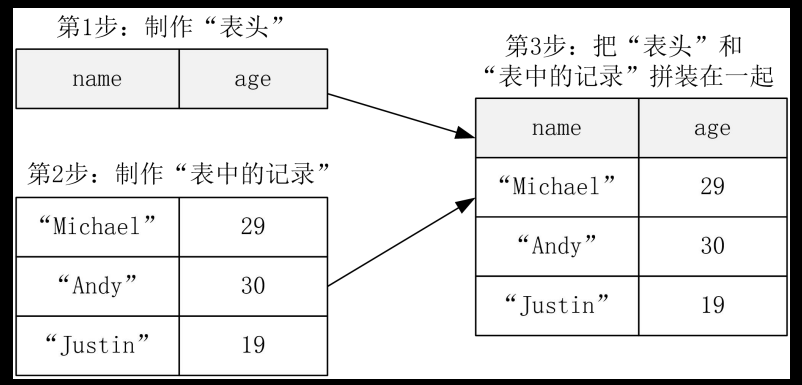
使用编程方式定义RDD模式-4

- 通过spark.createDataFrame(rowRDD, schema)语句，把记录拼装在一起，得到一个DataFrame，用于后续SQL查询

```
//下面把“表头”和“表中的记录”拼装起来
 scala> val peopleDF = spark.createDataFrame(rowRDD, schema)
peopleDF: org.apache.spark.sql.DataFrame = [name: string, age: int]

//必须注册为临时表才能供下面查询使用
scala> peopleDF.createOrReplaceTempView("people")
scala> val results = spark.sql("SELECT name,age FROM people")
results: org.apache.spark.sql.DataFrame = [name: string, age: int]

scala> results.
|  map(attributes => "name: " + attributes(0)+","+"age:"+attributes(1)).
|  show()
+-------------------+
| value|
+-------------------+
|name: Michael,age:29|
| name: Andy,age:30|
| name: Justin,age:19|
+-------------------+
```



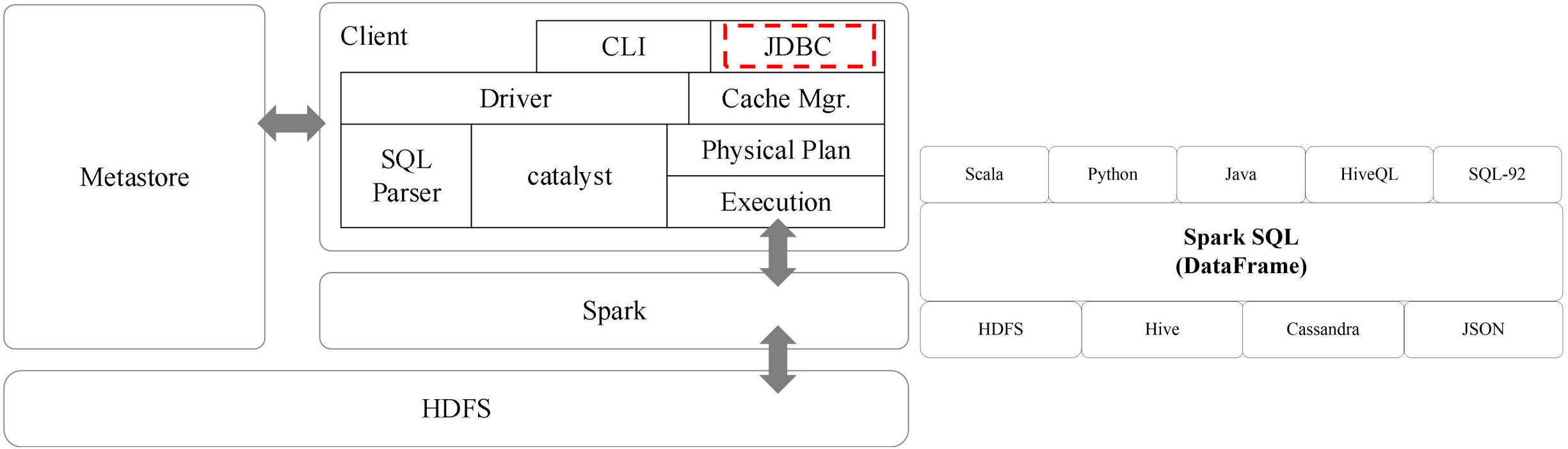
大数据处理—Spark框架

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

使用Spark SQL读写数据库

使用Spark SQL读写数据库

- Spark SQL可以支持Parquet、JSON、Hive等数据源，并且可以通过JDBC连接外部数据源



通过JDBC连接数据库

- 1. 准备工作
- 2. 读取MySQL数据库中的数据
- 3. 向MySQL数据库写入数据

•在Linux中启动MySQL数据库

```
$ service mysql start
$ mysql -u root -p
#屏幕会提示你输入密码
```

输入下面SQL语句完成数据库和表的创建：

```
mysql> create database spark;
mysql> use spark;
mysql> create table student (id int(4), name char(20), gender char(4), age int(4));
mysql> insert into student values(1,'Xueqian','F',23);
mysql> insert into student values(2,'Weiliang','M',24);
mysql> select * from student;
```

通过JDBC连接数据库

- 下载MySQL的JDBC驱动程序，比如mysql-connector-java-8.0.23.tar.gz
- 把该驱动程序拷贝到spark的安装目录” /root/spark-2.1.1-bin-hadoop2.7/”下
- 启动一个spark-shell，启动Spark Shell时，必须指定mysql连接驱动jar包

```
$ cd /usr/local/spark
$ ./bin/spark-shell  \
--jars /root/spark-2.1.1-bin-hadoop2.7/mysql-connector-java-8.0.23/mysql-connector-java-8.0.23-bin.jar \
--driver-class-path /root/spark-2.1.1-bin-hadoop2.7/mysql-connector-java-8.0.23/mysql-connector-java-8.0.23-bin.jar
```

通过JDBC连接数据库

- 2. 读取MySQL数据库中的数据
- 执行以下命令连接数据库，读取数据，并显示：

```
scala> val jdbcDF = spark.read.format("jdbc").
| option("url","jdbc:mysql://localhost:3306/spark").
| option("driver","com.mysql.jdbc.Driver").
| option("dbtable", "student").
| option("user", "root").
| option("password", "hadoop").
| load()
scala> jdbcDF.show()
+---+--------+------+---+
| id| name|gender|age|
+---+--------+------+---+
| 1| Xueqian| F| 23|
| 2|Weiliang| M| 24|
+---+--------+------+---+
```

通过JDBC连接数据库

- 3. 向MySQL数据库写入数据
- 在MySQL数据库中创建了一个名称为spark的数据库，并创建了一个名称为student的表
- 创建后，查看一下数据库内容：

```
mysql>  use spark;
Database changed

mysql> select * from student;
//上面命令执行后返回下面结果
+------+----------+--------+------+
| id   | name     | gender | age  |
+------+----------+--------+------+
|    1 | Xueqian  | F      |   23 |
|    2 | Weiliang | M      |   24 |
+------+----------+--------+------+
2 rows in set (0.00 sec)
```

通过JDBC连接数据库

- 现在开始在spark-shell中编写程序，往spark.student表中插入两条记录

```
import java.util.Properties
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row

//下面我们设置两条数据表示两个学生信息
val studentRDD = spark.sparkContext.parallelize(Array("3 Rongcheng M 26","4
Guanhua M 27")).map(_.split(" "))

//下面要设置模式信息
val schema = StructType(List(StructField("id", IntegerType,
true),StructField("name", StringType, true),StructField("gender", StringType,
true),StructField("age", IntegerType, true)))
```

剩余代码见下一页

通过JDBC连接数据库


```
//下面创建Row对象，每个Row对象都是rowRDD中的一行
val rowRDD = studentRDD.map(p => Row(p(0).toInt, p(1).trim, p(2).trim, p(3).toInt))

//建立起Row对象和模式之间的对应关系，也就是把数据和模式对应起来
val studentDF = spark.createDataFrame(rowRDD, schema)

//下面创建一个prop变量用来保存JDBC连接参数
val prop = new Properties()
prop.put("user", "root") //表示用户名是root
prop.put("password", "hadoop") //表示密码是hadoop
prop.put("driver","com.mysql.jdbc.Driver") //表示驱动程序是com.mysql.jdbc.Driver

//下面就可以连接数据库，采用append模式，表示追加记录到数据库spark的student表中
studentDF.write.mode("append").jdbc("jdbc:mysql://localhost:3306/spark",
"spark.student", prop)
```

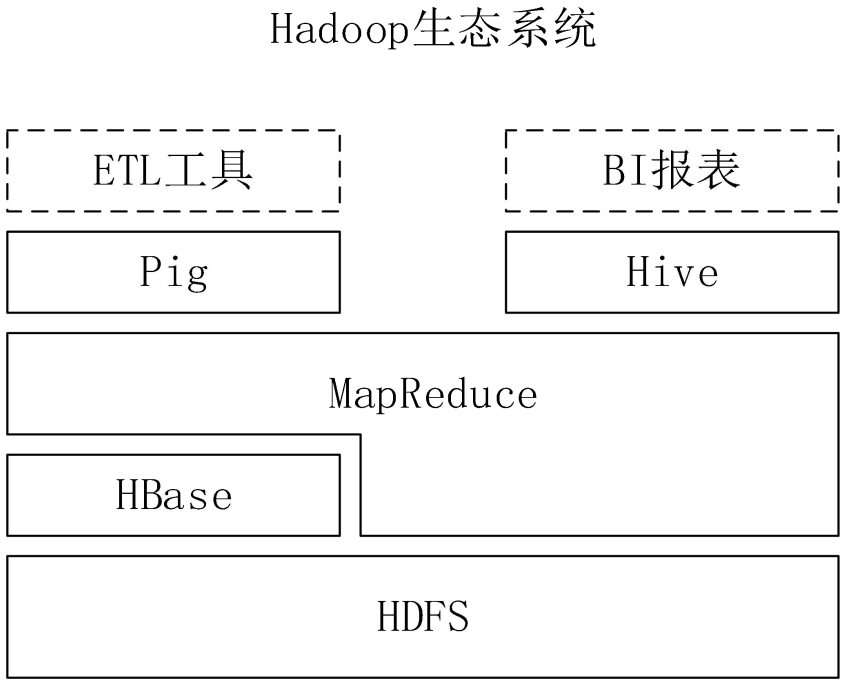
通过JDBC连接数据库

可以看一下效果，看看MySQL数据库中的spark.student表数据进行更新

```
mysql> select * from student;
+------+----------+--------+------+
| id | name | gender | age |
+------+----------+--------+------+
| 1 | Xueqian | F | 23 |
| 2 | Weiliang | M | 24 |
| 3 | Rongcheng | M | 26 |
| 4 | Guanhua | M | 27 |
+------+----------+--------+------+
4 rows in set (0.00 sec)
```

Spark SQL对Hive数据仓库的数据读写

- Hive是一个构建于Hadoop顶层的数据仓库工具
- 支持大规模数据存储、分析，具有良好的可扩展性
- 定义了简单的类似SQL 的查询语言—HiveQL
- 用户可以通过编写的HiveQL语句运行MapReduce任务
- 可以很容易把原来构建在关系数据库上的数据仓库应用程序移植到Hadoop平台上
- 是一个可以提供有效、合理、直观组织和使用数据的分析工具
- Spark SQL兼容对Hive数据仓库的访问



Hive与Hadoop生态系统中其他组件的关系（注：早期Hadoop组件的使用方案，故图中没有spark等组件）

连接Hive读写数据

1.测试Spark版本是否支持Hive

- 为了让Spark能够访问Hive，必须为Spark添加Hive支持
- Spark官方提供的预编译版本，通常是不包含Hive支持的，需要在Spark官网选择Source Code版本类型下载，进行源码编译，编译得到一个包含Hive支持的Spark版本

(1) 测试一下自己电脑上已经安装的Spark版本是否支持Hive

启动进入了spark-shell，如果不支持Hive，会显示如下信息：

```
scala> import org.apache.spark.sql.hive.HiveContext
<console>:25: error: object hive is not a member of package org.apache.spark.sql
         import org.apache.spark.sql.hive.HiveContext
                                         ^
```

如果你当前电脑上的Spark版本包含Hive支持，那么应该显示下面的正确信息：

```
scala> import org.apache.spark.sql.hive.HiveContext
import org.apache.spark.sql.hive.HiveContext
```

连接Hive读写数据

- **（2）采用源码编译方法得到支持Hive的Spark版本**
- 到Spark官网下载源码
- http://spark.apache.org/downloads.html

连接Hive读写数据

- 解压文件

```
$ cd /home/hadoop/下载 //spark-2.1.0.tgz就在这个目录下面
$ ls #可以看到刚才下载的spark-2.1.0.tgz文件
$ sudo tar -zxf ./spark-2.1.0.tgz -C /home/hadoop/
$ cd /home/hadoop
$ ls #这时可以看到解压得到的文件夹spark-2.1.0
```

在编译时，需要给出电脑上之前已经安装好的Hadoop的版本

```
$ hadoop version
```

查看已安装的Hadoop的版本

运行如下编译命令，对Spark源码进行编译

```
$ cd /home/hadoop/spark-2.1.0
$ ./dev/make-distribution.sh —tgz —name h27hive -Pyarn -Phadoop-2.7 -
Dhadoop.version=2.7.1 -Phive -Phive-thriftserver -DskipTests
```

编译成功后会得到文件名“spark-2.1.0-bin-h27hive.tgz”，这个就是包含Hive支持的Spark安装文件

连接Hive读写数据

2.在Hive中创建数据库和表

- 假设已经完成了Hive的安装，并且使用的是MySQL数据库来存放Hive的元数据
- 需要借助于MySQL保存Hive的元数据，首先启动MySQL数据库：

```
$ service mysql start
```

由于Hive是基于Hadoop的数据仓库，使用HiveQL语言撰写的查询语句，最终都会被Hive自动解析成MapReduce任务由Hadoop去具体执行，因此，需要启动Hadoop，然后再启动Hive

启动Hadoop：

```
$ cd /usr/local/hadoop
$ ./sbin/start-all.sh
```

Hadoop启动成功以后，可以再启动Hive：

```
$ cd /usr/local/hive
$ ./bin/hive
```

在Hive中创建数据库，并写入初始数据

- 进入Hive，新建一个数据库sparktest，并在这个数据库下面创建一个表student，并录入两条数据

```
hive> create database if not exists sparktest;//创建数据库sparktest
hive> show databases; //显示一下是否创建出了sparktest数据库

//下面在数据库sparktest中创建一个表student
hive> create table if not exists sparktest.student(
> id int,
> name string,
> gender string,
> age int);
hive> use sparktest; //切换到sparktest
hive> show tables; //显示sparktest数据库下面有哪些表
hive> insert into student values(1,'Xueqian','F',23); //插入一条记录
hive> insert into student values(2,'Weiliang','M',24); //再插入一条记录
hive> select * from student; //显示student表中的记录
```

连接Hive读写数据

- 为了让Spark顺利访问Hive
- 需要修改“/usr/local/sparkwithhive/conf/spark-env.sh”这个配置文件，具体修改后的配置文件内容：

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib
export SCALA_HOME=/usr/local/scala
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
export HIVE_CONF_DIR=/usr/local/hive/conf
export SPARK_CLASSPATH=$SPARK_CLASSPATH:/usr/local/hive/lib/mysql-connector-java-5.1.40-bin.jar
```

从Hive中读取数据

- 安装好包含Hive支持的spark版本后，启动进入Spark-shell
- 在spark-shell（中执行以下命令从Hive中读取数据：

```
Scala> import org.apache.spark.sql.Row
Scala> import org.apache.spark.sql.SparkSession
Scala> case class Record(key: Int, value: String)
// warehouseLocation points to the default location for managed databases and tables
Scala> val warehouseLocation = "spark-warehouse”
Scala> val spark = SparkSession.builder().appName("Spark Hive Example").config("spark.sql.warehouse.dir", warehouseLocation).enableHiveSupport().getOrCreate()
Scala> import spark.implicits._
Scala> import spark.sql
```

```
//下面是运行结果
scala> sql("SELECT * FROM sparktest.student").show()
+---+--------+------+---+
| id| name|gender|age|
+---+--------+------+---+
| 1| Xueqian| F| 23|
| 2|Weiliang| M| 24|
+---+--------+------+---+
```

向Hive写数据-1

- 编写程序向Hive数据库的sparktest.student表中插入两条数据
- 在插入数据之前，先在Hive里查看一下已有的2条数据

```
hive> use sparktest;
OK
Time taken: 0.016 seconds

hive> select * from student;
OK
1    Xueqian F    23
2    Weiliang     M    24
Time taken: 0.05 seconds, Fetched: 2 row(s)
```

向Hive写数据-2

- 编写Spark程序向Hive数据库的sparktest.student表中插入两条数据：
- 1）先准备待写入Hive的数据，创建RDD，下面创建了studentRDD

```
scala> import java.util.Properties
scala> import org.apache.spark.sql.types._
scala> import org.apache.spark.sql.Row
//下面我们设置两条数据表示两个学生信息
scala> val studentRDD =
spark.sparkContext.parallelize(Array("3 Rongcheng M
26","4 Guanhua M 27")).map(_.split(" "))
```

剩余代码见下一页

向Hive写数据-3

- 2）创建构建DataFrame所需要的模式信息schema和行数据Row对象
- 通过准备好的rowRDD和schema，创建出studentDF

```
//下面要设置模式信息
scala> val schema = StructType(List(StructField("id", IntegerType, true),StructField("name", StringType, true),StructField("gender", StringType, true),StructField("age", IntegerType, true)))

//下面创建Row对象，每个Row对象都是rowRDD中的一行
scala> val rowRDD = studentRDD.map(p => Row(p(0).toInt, p(1).trim, p(2).trim, p(3).toInt))

//建立起Row对象和模式之间的对应关系，也就是把数据和模式对应起来
scala> val studentDF = spark.createDataFrame(rowRDD, schema)
```

剩余代码见下一页

向Hive写数据-4

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- studentDF创建成功，可以使用show()操作查看
- 使用DataFrame注册临时表
- 将临时表的数据插入Hive

```
//查看studentDF
scala> studentDF.show()
+---+---------+------+---+
| id| name|gender|age|
+---+---------+------+---+
| 3|Rongcheng| M| 26|
| 4| Guanhua| M| 27|
+---+---------+------+---+
//下面注册临时表
scala> studentDF.registerTempTable("tempTable")
 //下面执行向Hive中插入记录的操作
scala> sql("insert into sparktest.student select * from tempTable")
```

向Hive写数据-5

- 在Hive下，可以输入以下命令查看Hive数据库内容的变化：

```
hive> select * from student;
OK
1   Xueqian F   23
2   Weiliang    M   24
3   Rongcheng   M   26
4   Guanhua M   27
Time taken: 0.049 seconds, Fetched: 4 row(s)
```

插入数据操作执行成功

Spark SQL 小结

- 在大数据的处理框架上提供SQL支持，一方面可以简化开发人员的编程工作，另一方面可以用 大数据技术实现对结构化数据的高效复杂分析
- Spark SQL的数据模型DataFrame，它是一个由多个列组成的结构化的分布式数据集 合，相当于关系数据库中的一张表
- DataFrame是Spark SQL中的最基本的概念，可以通过多种方式创建：如结构化的数据集、Hive表、外部数据库或者是RDD等
- DataFrame创建后，可以执行一些常用的 DataFrame 操作，包括 printSchema()、select()、filter()、groupBy ()和 sort ()等
- 从 RDD 转换得到DataFrame，有时候可以实现自动的隐式转换；但是，有时候需要通过编程的方式实现转换， 主要有两种方式：即利用反射机制推断RDD模式和使用编程方式定义RDD模式
- 在Spark中通过JDBC实现对MySQL数据库连接；Spark也支持连接Hive数据库等进行读写数据

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

谢谢聆听
批评指正

ehaihong@bupt.edu.cn