Spark SQL 核心知识

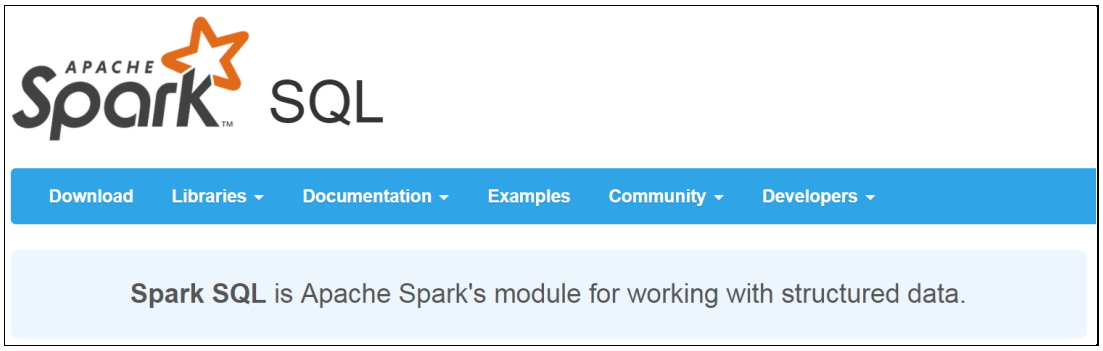
目录

1、Spark SQL 概述

1.1、什么是 Spark SQL

Spark SQL 是 Spark 用来处理**结构化数据**（结构化数据可以来自外部结构化数据源也可以通过 RDD 获取）的一个模块，它提供了一个编程抽象叫做 DataFrame 并且作为分布式 SQL 查询引擎的作用。

外部的结构化数据源包括 **JSON、Parquet(默认)、RMDBS、Hive** 等。当前 Spark SQL 使用 Catalyst 优化器来对 SQL 进行优化，从而得到更加高效的执行方案。并且可以将结果存储到外部系统。

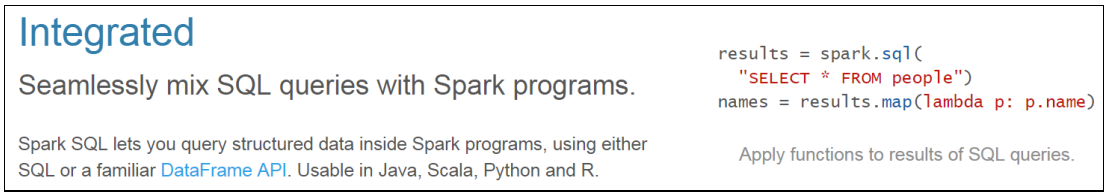


1.2、为什么要学习 Spark SQL

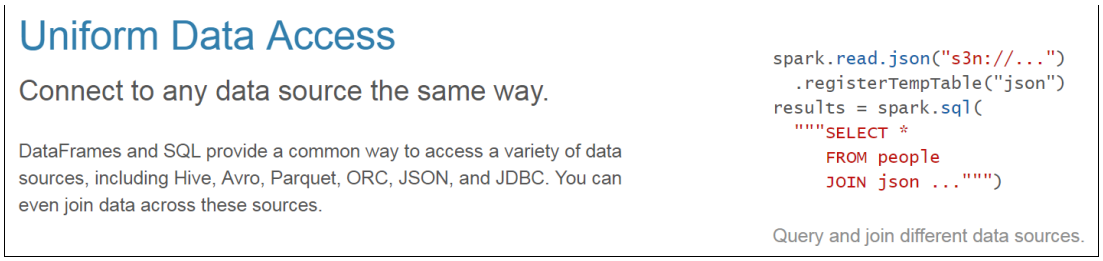
我们已经学习了 Hive，它是将 Hive SQL 转换成 MapReduce 然后提交到集群上执行，大大简化了编写 MapReduce 的程序的复杂性，由于 MapReduce 这种计算模型执行效率比较慢。所以 Spark SQL 就应运而生，它的工作机制是**将 Spark SQL 的 SQL 查询转换成 Spark Core 的应用程序**，然后提交到集群执行，执行效率非常快！

Spark SQL 的特点：

1、容易整合



2、统一的数据访问方式



3、兼容 Hive

4、标准的数据连接

1.3、SparkSQL 的版本迭代

SparkSQL 版本升级对应的新特性汇总

1、**SparkSQL 的前身是 Shark**。由于 Shark 自身的不完善，2014 年 6 月 1 日 Reynold Xin 宣布：停止对 Shark 的开发。SparkSQL 抛弃原有 Shark 的代码，汲取了 Shark 的一些优点，如内存列存储（In-Memory Columnar Storage）、Hive 兼容性等，重新开发 SparkSQL。

2、**Spark-1.1：2014 年 9 月 11 日，发布 Spark1.1.0。Spark 从 1.0 开始引入 SparkSQL（Shark 不再支持升级与维护）**。Spark1.1.0 变化较大是 SparkSQL 和 MLlib

3、Spark-1.3：增加 DataFrame 新 API

4、Spark-1.4：增加窗口分析函数

5、Spark 1.5：钨丝计划。Hive 中有 UDF 与 UDAF，Spark 中对 UDF 支持较早
UDAF：User Defined Aggregate Function 用户自定义聚合函数，直到 Spark 1.5.x 才引入的最新特性
关于“钨丝计划”：
https://blog.csdn.net/nysyxxg/article/details/56962247?locationNum=6&fps=1

6、spark-1.6：执行的 sql 中可以增加"--"注释，Spark-1.5/1.6 的新特性，引入 DataSet 的概念

7、Spark-2.x：SparkSQL+DataFrame+DataSet(正式版本)，Structured Streaming(DataSet)，引入 SparkSession 统一了 RDD，DataFrame，DataSet 的编程入口

2、SparkSession

SparkSession 是 Spark-2.0 引如的新概念。SparkSession 为用户提供了统一的切入点，来让用户学习 Spark 的各项功能。

在 Spark 的早期版本中，SparkContext 是 Spark 的主要切入点，由于 RDD 是主要的 API，我们通过 sparkContext 来创建和操作 RDD。对于每个其他的 API，我们需要使用不同的 context。例如：

对于 Spark Streaming，我们需要使用 StreamingContext

对于 Spark SQL，使用 SQLContext

对于 Hive，使用 HiveContext

但是随着 DataSet 和 DataFrame 的 API 逐渐成为标准的 API，就需要为他们建立接入点。所以在 Spark2.0 中，引入 SparkSession 作为 DataSet 和 DataFrame API 的切入点，SparkSession 封装了 SparkConf、SparkContext 和 SQLContext。为了向后兼容，SQLContext 和 HiveContext 也被保存下来。

SparkSession 实质上是 SQLContext 和 HiveContext 的组合，所以在 SQLContext 和 HiveContext 上可用的 API 在 SparkSession 上同样是可以使用的。SparkSession 内部封装了 SparkContext，所以计算实际上是由 SparkContext 完成的。

特点：

---- 为用户提供一个统一的切入点使用 Spark 各项功能
---- 允许用户通过它调用 DataFrame 和 Dataset 相关 API 来编写程序
---- 减少了用户需要了解的一些概念，可以很容易的与 Spark 进行交互
---- 与 Spark 交互之时不需要显示的创建 SparkConf、SparkContext 以及 SQlContext，这些对象已经封闭在 SparkSession 中
---- SparkSession 提供对 Hive 特征的内部支持：用 HiveQL 写 SQL 语句，访问 Hive UDFs，从 Hive 表中读取数据。

SparkSession 的创建：

1、如果是在 Spark-Shell 中

SparkSession 会被自动初始化一个对象叫做 spark，为了向后兼容，Spark-Shell 还提供了一个 SparkContext 的初始化对象，方便用户操作：

```
[hadoop@hadoop02 ~]$ ~/apps/spark-2.3.1-bin-hadoop2.7/bin/spark-shell \
> --master spark://hadoop02:7077 \
> --executor-memory 512m \
> --total-executor-cores 1
```

```
[hadoop@hadoop02 ~]$ ~/apps/spark-2.3.1-bin-hadoop2.7/bin/spark-shell \
> --master spark://hadoop02:7077 \
> --executor-memory 512m \
> --total-executor-cores 1
18/07/26 16:08:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-
Spark context Web UI available at http://hadoop02:4040
Spark context available as 'sc' (master = spark://hadoop02:7077, app id = app-20180726160844-0000).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

2、如果是在 IDEA 中创建 SparkSession

The entry point into all functionality in Spark is the `SparkSession` class. To create a basic `SparkSession`, just use `SparkSession.builder()`:

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

val spark = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()

// For implicit conversions like converting RDDs to DataFrames
import spark.implicits._
```

Find full example code at "examples/src/main/scala/org/apache/spark/examples/sql/SparkSQLExample.scala" in the Spark repo.

`SparkSession` in Spark 2.0 provides builtin support for Hive features including the ability to write queries using HiveQL, access to Hive UDFs, and the ability to read data from Hive tables. To use these features, you do not need to have an existing Hive setup.

3、RDD/DataFrame/DataSet

3.1、RDD 的局限性

RDD 仅表示数据集，RDD 没有元数据，也就是说没有字段语义定义
RDD 需要用户自己优化程序，对程序员要求较高
从不同数据源读取数据相对困难，读取到不同格式的数据都必须用户自己定义转换方式
合并多个数据源中的数据也较困难

3.2、什么是 DataFrame

首先回忆一下之前的 SparkCore 的 RDD 编程：

Spark Core 编程：
1）首先要找到程序入口（SparkContext）
2）通过程序入口构建一个 RDD（核心的抽象 RDD）
3）对写 RDD 进行 Transformation 或者 Action 的操作
4）对最后的结果进行处理（输出或者存入数据库等）

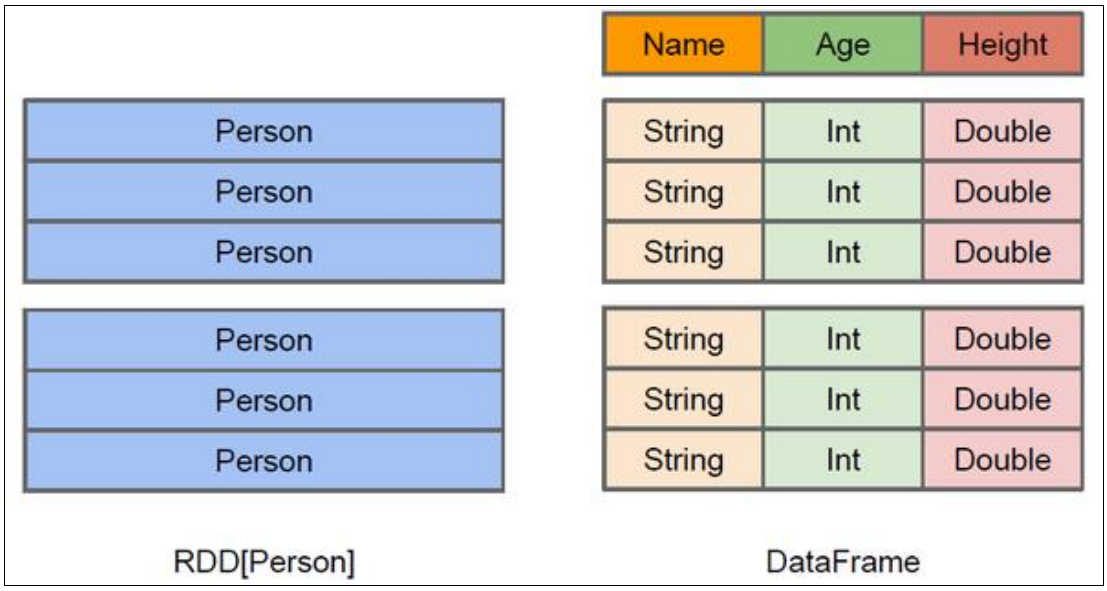
由于 RDD 的局限性，Spark 产生了 DataFrame。
其中 Schema 是就是元数据，是语义描述信息。
在 Spark1.3 之前，DataFrame 被称为 SchemaRDD。以行为单位构成的分布式数据集合，按照列赋予不同的名称。对 select、fileter、aggregation 和 sort 等操作符的抽象。
DataFrame = RDD+Schema = SchemaRDD

内部数据无类型，统一为 Row
DataFrame 是一种特殊类型的 Dataset，**DataSet[Row] = DataFrame**
DataFrame 自带优化器 Catalyst，可以自动优化程序
DataFrame 提供了一整套的 Data Source API

DataFrame 的官方解释：
http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes

A DataFrame is a Dataset organized into named columns.
DataFrame 是按列名的方式去组织的一个分布式的数据集（RDD）
It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood.
就像关系型数据库里面的一张表，（或者说好比是 R/Python 语言里面的 DataFrame），不过 SparkSQL 这儿的方法比 R/Python 语言里面的 DataFrame 提供的操作方法更丰富
DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.
DataFrame 的数据源有如下：结构化的文件，Hive 里面的表，外部的数据库（MySQL 等），已经存在的 RDD。
The DataFrame API is available in Scala, Java, Python, and R.
DataFrame 提供了 Scala，Java，Python，R 的编程 API
In Scala and Java, a DataFrame is represented by a Dataset of Rows.
在 Scala 或者 Java 编程中，一个 DataFrame 表示以行组织的 Rows 的数据集合
In the Scala API, DataFrame is simply a type alias of Dataset[Row].
在 Scala 的 API 中，DataFrame 就可以看做是 Dataset[Row]的另一种称呼
While, in Java API, users need to use Dataset<Row> to represent a DataFrame.
但是，在 Java 的 API 中，开发者必须使用 Dataset<Row>去表示一个 DataFrame

与 RDD 类似，DataFrame 也是一个分布式数据容器。然而 DataFrame 更像传统数据库的二维表格，除了数据以外，还记录数据的结构信息，即 Schema。同时，与 Hive 类似，DataFrame 也支持嵌套数据类型（struct、array 和 map）。从 API 易用性的角度上 看，DataFrame API 提供的是一套高层的关系操作，比函数式的 RDD API 要更加友好，门槛更低。由于与 R 和 Pandas 的 DataFrame 类似，Spark DataFrame 很好地继承了传统单机数据分析的开发体验。



DataFrame 特点：

1、Ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster（支持单机 KB 级到集群 PB 级的数据处理）

2、Support for a wide array of data formats and storage systems（支持多种数据格式和存储系统）

3、State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer（通过 Spark SQL Catalyst 优化器可以进行高效的代码生成和优化）

4、Seamless integration with all big data tooling and infrastructure via Spark（能够无缝集成所有的大数据处理工具）

5、APIs for Python, Java, Scala, and R (in development via SparkR)（提供 Python, Java, Scala, R 语言 API)

现在的 SparkSQL 的编程：

Spark SQL 编程：

1）首先要找到程序入口（SQLContext），新版本 Spark-2.x 之后寻找 SparkSession

2）通过程序入口构建一个 DataFrame(核心的抽象 DataFrame)

3）对 DataFrame 做各种操作。最重要就是编写 SQL 语句

4）对得到的结果数据进行处理（打印输出或者存入数据库等）

3.3、DataSet 的产生

由于 DataFrame 的数据类型统一是 Row，所以 DataFrame 也是有缺点的。

Row 运行时类型检查，比如 salary 是字符串类型，下面语句也只有运行时才进行类型检查。

```
dataframe.filter("salary>1000").show()
```

明显的缺点：

```
1、Row 不能直接操作 domain 对象
2、函数风格编程，没有面向对象风格的 API
```

所以，Spark SQL 引入了 Dataset，扩展了 **DataFrame API，提供了编译时类型检查，面向对象风格的 API**。

```
Dataset 可以和 DataFrame、RDD 相互转换。DataFrame=Dataset[Row]
可见 DataFrame 是一种特殊的 Dataset。
```

官网中的 DataSet 解释：

A Dataset is a distributed collection of data. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). The Dataset API is available in Scala and Java. Python does not have the support for the Dataset API. But due to Python's dynamic nature, many of the benefits of the Dataset API are already available (i.e. you can access the field of a row by name naturally row.columnName). The case for R is similar.

翻译：

一个 Dataset 是一个分布式的数据集合 Dataset 是在 Spark 1.6 中被添加的新接口，它提供了 RDD 的优点（强类型化，能够使用强大的 lambda 函数）与 Spark SQL 执行引擎的优点。一个 Dataset 可以从 JVM 对象来构造并且使用转换功能（map, flatMap, filter,等等）。Dataset API 在 Scala 和 Java 是可用的。Python 不支持 Dataset API。但是由于 Python 的动态特性，许多 Dataset API 的优点已经可用了 (也就是说，你可能通过 name 天生的 row.columnName 属性访问一行中的字段)。这种情况和 R 相似。

http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes.

3.4、为什么需要 DataFrame 和 Dataset

我们知道 Spark SQL 提供了两种方式操作数据：

```
SQL 查询
DataFrame 和 Dataset API
```

既然 Spark SQL 提供了 SQL 访问方式，那为什么还需要 DataFrame 和 Dataset 的 API 呢？

这是因为 SQL 语句虽然简单，但是 **SQL 的表达能力却是有限**的（所以 Oracle 数据库提供了 PL/SQL）。**DataFrame 和 Dataset 可以采用更加通用的语言（Scala 或 Python）来表达用户的**

查询请求。此外，Dataset 可以更快捕捉错误，因为 **SQL 是运行时捕获异常，而 Dataset 是编译时检查错误**。

3.5、Spark SQL 程序基本编写步骤

1、创建 SparkSession 对象

SparkSession 封装了 Spark SQL 执行环境信息，是所有 Spark SQL 程序唯一的入口。

```
// 构建 SparkSQL 程序的编程入口对象 SparkSession
val sparkSession:SparkSession = SparkSession.builder()
  .appName("MyFirstSparkSQL")
  .config("someKey", "someValue")
  .master("local")
  .getOrCreate()
```

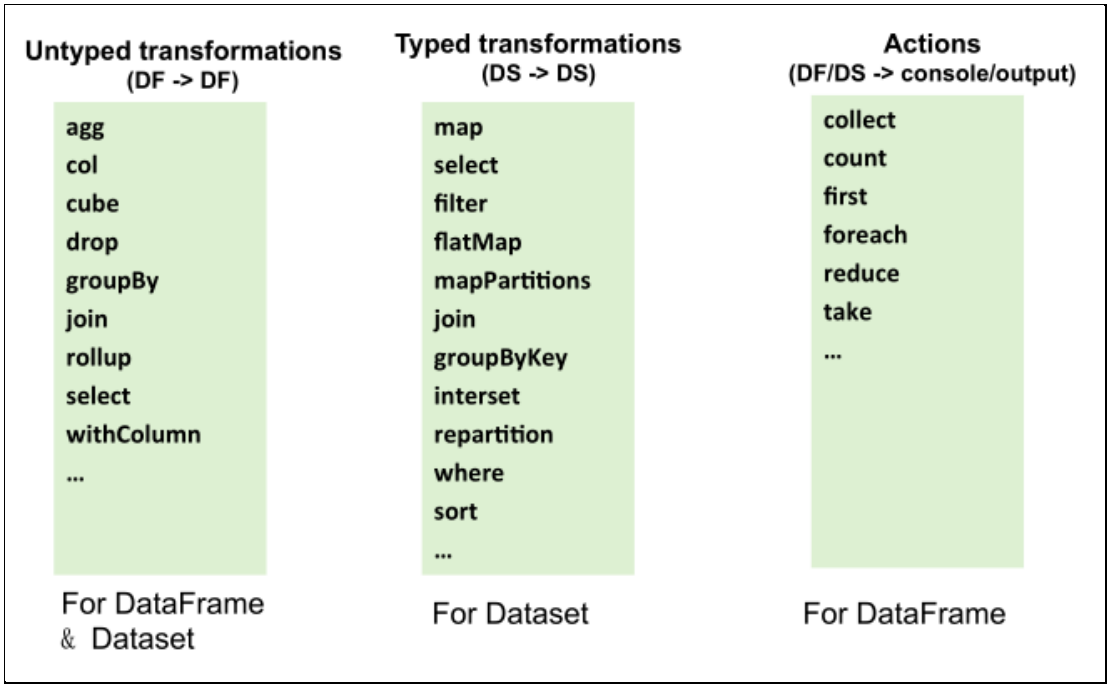
2、创建 DataFrame 或 Dataset

Spark SQL 支持多种数据源，包括常见的 JSON，JDBC，Parquet，HDFS，提供了读写各种格式数据的 API

```
scala> spark.
!=                      close              equals          ne              sharedState     time
##                      conf               experimental    newSession      sparkContext    toString
+                       createDataFrame    formatted       notify          sql             udf
->                      createDataset      getClass        notifyAll       sqlContext      version
==                      emptyDataFrame     hashCode        range           stop            wait
asInstanceOf            emptyDataset       implicits       read            streams         →
baseRelationToDataFrame ensuring           isInstanceOf    readStream      synchronized
catalog                 eq                 listenerManager sessionState    table
```

3、在 DataFrame 或 Dataset 之上进行转换和 Action

Spark SQL 提供了多钟转换和 Action 函数



4、返回结果

保存结果到 HDFS 中，或直接打印出来

3.6、创建 DataFrame

官网：http://spark.apache.org/docs/latest/sql-programming-guide.html#creating-dataframes

在 Spark SQL 中 SQLContext 是创建 DataFrames 和执行 SQL 的入口，在 spark-1.6.3 中已经内置了一个 sqlContext

```
[hadoop@hadoop02 ~]$ apps/spark-1.6.3-bin-hadoop2.6/bin/spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's repl log4j profile: org/apache/spark/log4j-defaults-repl.properties
To adjust logging level use sc.setLogLevel("INFO")
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.3
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
18/05/04 08:57:19 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:57:20 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:57:25 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verifica
18/05/04 08:57:25 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
18/05/04 08:57:27 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:57:28 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
SQL context available as sqlContext.

scala>
```

但是在 Spark-2.x 版本中，使用 SparkSession 作为统一的程序入口了。

```
[hadoop@hadoop05 ~]$ $SPARK_HOME/bin/spark-shell --master spark://hadoop02:7077 --executor-memory 512m --total-executor-cores 2
2018-05-03 17:49:42 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java cla
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://hadoop05:4040
Spark context available as 'sc' (master = spark://hadoop02:7077, app id = app-20180503174950-0009).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

1、在本地创建一个文件，有五列，分别是 id、name、sex、age、department，用空格分隔，然后上传到 HDFS 上

hdfs dfs -put student.txt /student

```
[hadoop@hadoop02 ~]$ hadoop fs -cat /student/student.txt
95002,刘晨,女,19,IS
95017,王风娟,女,18,IS
95018,王一,女,19,IS
95013,冯伟,男,21,CS
95014,王小丽,女,19,CS
95019,邢小丽,女,19,IS
95020,赵钱,男,21,IS
95003,王敏,女,22,MA
95004,张立,男,19,IS
95012,孙花,女,20,CS
95010,孔小涛,男,19,CS
95005,刘刚,男,18,MA
95006,孙庆,男,23,CS
95007,易思玲,女,19,MA
95008,李娜,女,18,CS
95021,周二,男,17,MA
95022,郑明,男,20,MA
95001,李勇,男,20,CS
95011,包小柏,男,18,MA
95009,梦圆圆,女,18,MA
95015,王君,男,18,MA[hadoop@hadoop02 ~]$
```

2、在 spark shell 执行下面命令，读取数据，将每一行的数据使用列分隔符分割

val lineRDD = sc.textFile("hdfs://myha01/student/student.txt").map(_.split(","))

3、定义 case class（相当于表的 schema）

case class Student(id:Int, name:String, sex:String, age:Int, department:String)

4、将 RDD 和 case class 关联

val studentRDD = lineRDD.map(x => Student(x(0).toInt, x(1), x(2), x(3).toInt, x(4)))

5、将 RDD 转换成 DataFrame

Spark-2.3 : val studentDF = spark.createDataFrame(studentRDD)

Spark-1.6 : val studentDF = studentRDD.toDF

6、对 DataFrame 进行处理

studentDF.show

studentDF.printSchema

7、结果验证

如果是 Spark-1.6.3 版本：

```
hadoop02 | hadoop02 (1) | hadoop03 | hadoop04 | hadoop05
18/05/04 09:07:30 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification
18/05/04 09:07:30 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
18/05/04 09:07:36 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 09:07:37 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
SQL context available as sqlContext.

scala> val lineRDD = sc.textFile("hdfs://myha01/student/student.txt").map(_.split(","))
lineRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at <console>:27

scala> case class Student(id:Int, name:String, sex:String, age:Int, department:String)
defined class Student

scala> val studentRDD = lineRDD.map(x => Student(x(0).toInt, x(1), x(2), x(3).toInt, x(4)))
studentRDD: org.apache.spark.rdd.RDD[Student] = MapPartitionsRDD[3] at map at <console>:31

scala> val stduentDF = studentRDD.toDF
stduentDF: org.apache.spark.sql.DataFrame = [id: int, name: string, sex: string, age: int, department: string]

scala> stduentDF.show
+-----+----+---+---+----------+
|   id|name|sex|age|department|
+-----+----+---+---+----------+
|95002|  刘晨|  女| 19|        IS|
|95017| 王风娟|  女| 18|        IS|
|95018|  王一|  女| 19|        IS|
|95013|  冯伟|  男| 21|        CS|
|95014| 王小丽|  女| 19|        CS|
|95019| 邢小丽|  女| 19|        IS|
|95020|  赵钱|  男| 21|        IS|
|95003|  王敏|  女| 22|        MA|
|95004|  张立|  男| 19|        IS|
|95012|  孙花|  女| 20|        CS|
|95010| 孔小涛|  男| 19|        CS|
|95005|  刘刚|  男| 18|        MA|
|95006|  孙庆|  男| 23|        CS|
|95007| 易思玲|  女| 19|        MA|
|95008|  李娜|  女| 18|        CS|
|95021|  周二|  男| 17|        MA|
|95022|  郑明|  男| 20|        MA|
|95001|  李勇|  男| 20|        CS|
|95011| 包小柏|  男| 18|        MA|
|95009| 梦圆圆|  女| 18|        MA|
+-----+----+---+---+----------+
only showing top 20 rows


scala>
```

如果使用 Spark-2.3.0 版本：

```
scala> val lineRDD = sc.textFile("hdfs://myha01/student/student.txt").map(_.split(","))
lineRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at <console>:24

scala> case class Student(id:Int, name:String, sex:String, age:Int, department:String)
defined class Student

scala> val studentRDD = lineRDD.map(x => Student(x(0).toInt, x(1), x(2), x(3).toInt, x(4)))
studentRDD: org.apache.spark.rdd.RDD[Student] = MapPartitionsRDD[3] at map at <console>:27

scala> val stduentDF = studentRDD.toDF
2018-05-04 09:34:49 WARN  ObjectStore:6666 - Version information not found in metastore. hive.metastore.schema.v
0
2018-05-04 09:34:49 WARN  ObjectStore:568 - Failed to get database default, returning NoSuchObjectException
2018-05-04 09:34:50 WARN  ObjectStore:568 - Failed to get database global_temp, returning NoSuchObjectException
stduentDF: org.apache.spark.sql.DataFrame = [id: int, name: string ... 3 more fields]

scala> stduentDF.show
+-----+----+---+---+----------+
|   id|name|sex|age|department|
+-----+----+---+---+----------+
|95002|  刘晨|  女| 19|        IS|
|95017| 王风娟|  女| 18|        IS|
|95018|  王一|  女| 19|        IS|
|95013|  冯伟|  男| 21|        CS|
|95014| 王小丽|  女| 19|        CS|
|95019| 邢小丽|  女| 19|        IS|
|95020|  赵钱|  男| 21|        IS|
|95003|  王敏|  女| 22|        MA|
|95004|  张立|  男| 19|        IS|
|95012|  孙花|  女| 20|        CS|
|95010| 孔小涛|  男| 19|        CS|
|95005|  刘刚|  男| 18|        MA|
|95006|  孙庆|  男| 23|        CS|
|95007| 易思玲|  女| 19|        MA|
|95008|  李娜|  女| 18|        CS|
|95021|  周二|  男| 17|        MA|
|95022|  郑明|  男| 20|        MA|
|95001|  李勇|  男| 20|        CS|
|95011| 包小柏|  男| 18|        MA|
|95009| 梦圆圆|  女| 18|        MA|
+-----+----+---+---+----------+
only showing top 20 rows


scala>
```

3.7、DataFrame 常用操作

3.7.1、DSL 风格语法

//打印 DataFrame 的 Schema 信息

studentDF.printSchema

```
scala> studentDF.printSchema
root
 |-- id: integer (nullable = false)
 |-- name: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- age: integer (nullable = false)
 |-- department: string (nullable = true)
```

// 查看 DataFrame 中的所有内容

studentDF.show

```
scala> studentDF.show
+-----+----+---+---+----------+
|   id|name|sex|age|department|
+-----+----+---+---+----------+
|95002|  刘晨|  女| 19|        IS|
|95017| 王风娟|  女| 18|        IS|
|95018|  王一|  女| 19|        IS|
|95013|  冯伟|  男| 21|        CS|
|95014| 王小丽|  女| 19|        CS|
|95019| 邢小丽|  女| 19|        IS|
|95020|  赵钱|  男| 21|        IS|
|95003|  王敏|  女| 22|        MA|
|95004|  张立|  男| 19|        IS|
|95012|  孙花|  女| 20|        CS|
|95010| 孔小涛|  男| 19|        CS|
|95005|  刘刚|  男| 18|        MA|
|95006|  孙庆|  男| 23|        CS|
|95007| 易思玲|  女| 19|        MA|
|95008|  李娜|  女| 18|        CS|
|95021|  周二|  男| 17|        MA|
|95022|  郑明|  男| 20|        MA|
|95001|  李勇|  男| 20|        CS|
|95011| 包小柏|  男| 18|        MA|
|95009| 梦圆圆|  女| 18|        MA|
+-----+----+---+---+----------+
only showing top 20 rows
```

//查看 DataFrame 部分列中的内容

studentDF.select("name", "age").show

studentDF.select(col("name"), col("age")).show

studentDF.select(studentDF.col("name"), studentDF.col("age")).show

```
scala> studentDF.select(studentDF.col("name"), studentDF.col("age")).show
+----+---+
|name|age|
+----+---+
|  刘晨| 19|
| 王风娟| 18|
|  王一| 19|
|  冯伟| 21|
| 王小丽| 19|
| 邢小丽| 19|
|  赵钱| 21|
|  王敏| 22|
|  张立| 19|
|  孙花| 20|
| 孔小涛| 19|
|  刘刚| 18|
|  孙庆| 23|
| 易思玲| 19|
|  李娜| 18|
|  周二| 17|
|  郑明| 20|
|  李勇| 20|
| 包小柏| 18|
| 梦圆圆| 18|
+----+---+
only showing top 20 rows
```

//查询所有的 name 和 age，并将 age+1

studentDF.select(col("id"), col("name"), col("age") + 1).show

studentDF.select(studentDF ("id"), studentDF ("name"), studentDF ("age") + 1).show

```
scala> studentDF.select(studentDF ("id"), studentDF ("name"), studentDF ("age") + 1).show
+-----+----+---------+
|   id|name|(age + 1)|
+-----+----+---------+
|95002|  刘晨|       20|
|95017| 王风娟|       19|
|95018|  王一|       20|
|95013|  冯伟|       22|
|95014| 王小丽|       20|
|95019| 邢小丽|       20|
|95020|  赵钱|       22|
|95003|  王敏|       23|
|95004|  张立|       20|
|95012|  孙花|       21|
|95010| 孔小涛|       20|
|95005|  刘刚|       19|
|95006|  孙庆|       24|
|95007| 易思玲|       20|
|95008|  李娜|       19|
|95021|  周二|       18|
|95022|  郑明|       21|
|95001|  李勇|       21|
|95011| 包小柏|       19|
|95009| 梦圆圆|       19|
+-----+----+---------+
only showing top 20 rows
```

// 过滤 age 大于等于 20 的

studentDF.filter(col("age") >= 20).show

```
scala> studentDF.filter(col("age") >= 20).show
+-----+----+---+---+----------+
|   id|name|sex|age|department|
+-----+----+---+---+----------+
|95013|  冯伟|  男| 21|        CS|
|95020|  赵钱|  男| 21|        IS|
|95003|  王敏|  女| 22|        MA|
|95012|  孙花|  女| 20|        CS|
|95006|  孙庆|  男| 23|        CS|
|95022|  郑明|  男| 20|        MA|
|95001|  李勇|  男| 20|        CS|
+-----+----+---+---+----------+
```

// 按年龄进行分组并统计相同年龄的人数

studentDF.groupBy("age").count().show()

```
scala> studentDF.groupBy("age").count().show()
+---+-----+
|age|count|
+---+-----+
| 17|    1|
| 18|    6|
| 19|    7|
| 20|    3|
| 21|    2|
| 22|    1|
| 23|    1|
+---+-----+
```

3.7.2、SQL 风格语法

如果想使用 SQL 风格的语法，需要将 DataFrame 注册成表

老版本写法：

studentDF.registerTempTable("t_student")

```
scala> studentDF.registerTempTable("t_student")
```

新版本写法

1、Session 范围内的临时表：studentDF.createOrReplaceTempView("t_student")

只在 Session 范围内有效，Session 结束临时表自动销毁

2、全局范围内的临时表：studentDF.createGlobalTempView("t_student")
所有 Session 共享

// 查询年龄最大的前五名
sqlContext.sql("select * from t_student order by age desc limit 5").show

```
scala> sqlContext.sql("select * from t_student order by age desc limit 5").show
+-----+----+---+---+----------+
|   id|name|sex|age|department|
+-----+----+---+---+----------+
|95006|  孙庆|  男| 23|        CS|
|95003|  王敏|  女| 22|        MA|
|95013|  冯伟|  男| 21|        CS|
|95020|  赵钱|  男| 21|        IS|
|95012|  孙花|  女| 20|        CS|
+-----+----+---+---+----------+
```

// 显示表的 Schema 信息
sqlContext.sql("desc t_student ").show

```
scala> sqlContext.sql("desc t_student ").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|        id|      int|       |
|      name|   string|       |
|       sex|   string|       |
|       age|      int|       |
|department|   string|       |
+----------+---------+-------+
```

// 统计学生数超过 6 个的部门和该部门的学生人数。并且按照学生的个数降序排序
sqlContext.sql("select department, count(*) as total from t_student group by department having total > 6 order by total desc").show

```
scala> sqlContext.sql("select department, count(*) as total from t_student group by department having total > 6 order by total desc").show
+----------+-----+
|department|total|
+----------+-----+
|        MA|    8|
|        CS|    7|
+----------+-----+
```

3.7.3、DataFrame 支持的操作

```
scala> val df = spark.sqlContext.read.json("file:///home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/examples/src/main/resources/people.json")
df: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> df.
agg                             dtypes              mapPartitions           stat
alias                           except              na                      storageLevel
apply                           explain             orderBy                 summary
as                              explode             persist                 take
cache                           filter              printSchema             takeAsList
checkpoint                      first               queryExecution          toDF
coalesce                        flatMap             randomSplit             toJSON
col                             foreach             randomSplitAsList       toJavaRDD
colRegex                        foreachPartition    rdd                     toLocalIterator
collect                         groupBy             reduce                  toString
collectAsList                   groupByKey          registerTempTable       transform
columns                         head                repartition             union
count                           hint                repartitionByRange      unionAll
createGlobalTempView            inputFiles          rollup                  unionByName
createOrReplaceGlobalTempView   intersect           sample                  unpersist
createOrReplaceTempView         isLocal             schema                  where
createTempView                  isStreaming         select                  withColumn
crossJoin                       javaRDD             selectExpr              withColumnRenamed
cube                            join                show                    withWatermark
describe                        joinWith            sort                    write
distinct                        limit               sortWithinPartitions    writeStream
drop                            localCheckpoint     sparkSession
dropDuplicates                  map                 sqlContext
```

4、以编程方式执行 Spark SQL 程序

4.1、编写 Spark SQL 查询程序

前面我们学习了如何在 Spark Shell 中使用 SQL 完成查询，现在我们来实现在自定义的程序中编写 Spark SQL 查询程序。首先在 maven 项目的 pom.xml 中添加 Spark SQL 的依赖

```
<spark.version>2.3.0</spark.version>

<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>${spark.version}</version>
</dependency>
```

```
11      <properties>
12          <maven.compiler.source>1.8</maven.compiler.source>
13          <maven.compiler.target>1.8</maven.compiler.target>
14          <encoding>UTF-8</encoding>
15          <scala.version>2.11.8</scala.version>
16          <spark.version>2.3.0</spark.version>
17          <hadoop.version>2.7.5</hadoop.version>
18          <scala.compat.version>2.11</scala.compat.version>
19      </properties>
20
21      <dependencies>
22
23          <dependency>
24              <groupId>org.apache.spark</groupId>
25              <artifactId>spark-sql_2.11</artifactId>
26              <version>${spark.version}</version>
27          </dependency>
```

Spark SQL 支持两种不同的方法用于转换已存在的 RDD 成为 Dataset

1、第一种方法是使用反射去推断一个包含指定的对象类型的 RDD 的 Schema。在你的 Spark 应用程序中当你已知 Schema 时这个基于方法的反射可以让你的代码更简洁
2、第二种用于创建 Dataset 的方法是通过一个允许你构造一个 Schema 然后把它应用到一个已存在的 RDD 的编程接口。然而这种方法更繁琐，当列和它们的类型知道运行时都是未知时它允许你去构造 Dataset

4.1.1、通过反射推断 Schema

创建一个名称叫做 com.mazh.spark.sql.StudentSparkSQL 的 Object

```
package com.mazh.spark.sql
```

```
import org.apache.spark.sql.{SQLContext, SparkSession}
import org.apache.spark.{SparkConf, SparkContext}

//case class 一定要事先放到外面定义好
case class Student(id: Int, name: String, sex: String, age: Int, department: String)

object StudentSparkSQL {

  def main(args: Array[String]) {

    //创建SparkConf()并设置App 名称
    val conf = new SparkConf().setAppName("FirstSparkSQLAPP--Student")
    //SQLContext 要依赖SparkContext
    val sc = new SparkContext(conf)
    //创建SQLContext
    val sqlContext = new SQLContext(sc)

    //从指定的地址创建RDD
    val lineRDD = sc.textFile(args(0)).map(_.split(","))

    //创建case class
    //将RDD 和case class 关联
    val studentRDD = lineRDD.map(x => Student(x(0).toInt, x(1), x(2), x(3).toInt,
x(4)))

    //导入隐式转换，如果不导入无法将RDD 转换成DataFrame
    //将RDD 转换成DataFrame
    import sqlContext.implicits._
    val studentDF = studentRDD.toDF

    //注册表
    studentDF.registerTempTable("t_student")
    //传入SQL
    val df = sqlContext.sql("select department, count(*) as total from t_student " +
      "group by department having total > 6 order by total desc")

    //将结果以JSON 的方式存储到指定位置
    df.write.json(args(1))
    //停止Spark Context
    sc.stop()
  }
}
```

先准备数据：student.txt

存储在 HDFS 中的/stuent 目录中
将程序打成 jar 包，上传到客户机，提高 Spark 任务：

```
// 先清空最终的输出结果的目录：
hadoop fs -rm -r hdfs://myha01/student/output_sparksql

// 提交 Spark 的任务：
$SPARK_HOME/bin/spark-submit \
--class com.mazh.spark.sql.StudentSparkSQL \
--master spark://hadoop02:7077,hadoop04:7077 \
/home/hadoop/Spark_SQL-1.0-SNAPSHOT.jar \
hdfs://myha01/student/student.txt \
hdfs://myha01/student/output_sparksql

// 查看 Spark 任务的结果：
hadoop fs -cat hdfs://myha01/student/output_sparksql/p*
```

最终结果：

```
[hadoop@hadoop05 ~]$ hadoop fs -cat hdfs://myha01/student/output_sparksql/p*
{"department":"MA","total":8}
{"department":"CS","total":7}
[hadoop@hadoop05 ~]$
```

4.1.2、通过 StructType 直接指定 Schema

创建一个名称为 com.mazh.spark.sql.StudentSparkSQL2 的 object

```
package com.mazh.spark.sql

import org.apache.spark.sql.{Row, SQLContext}
import org.apache.spark.sql.types._
import org.apache.spark.{SparkContext, SparkConf}


object StudentSparkSQL2 {

  def main(args: Array[String]) {
    //创建SparkConf()并设置App 名称
    val conf = new SparkConf().setAppName("StructType SparkSQL")
    //SQLContext 要依赖SparkContext
    val sc = new SparkContext(conf)
    //创建SQLContext
    val sqlContext = new SQLContext(sc)

    //从指定的地址创建RDD
```

```
    val personRDD = sc.textFile(args(0)).map(_.split(","))
    //通过 StructType 直接指定每个字段的 schema
    val schema = StructType(
      List(
        StructField("id", IntegerType, true),
        StructField("name", StringType, true),
        StructField("sex", StringType, true),
        StructField("age", IntegerType, true),
        StructField("department", StringType, true)
      )
    )
    //将 RDD 映射到 rowRDD
    val rowRDD = personRDD.map(p => Row(p(0).toInt, p(1).trim, p(2).trim, p(3).toInt,
p(4).trim))
    //将 schema 信息应用到 rowRDD 上
    val personDataFrame = sqlContext.createDataFrame(rowRDD, schema)
    //注册表
    personDataFrame.registerTempTable("t_student")
    //执行 SQL
    val df = sqlContext.sql("select department, count(*) as total from t_student " +
      "group by department having total > 6 order by total desc")
    //将结果以 JSON 的方式存储到指定位置
    df.write.json(args(1))
    //停止 Spark Context
    sc.stop()
  }
}
```

先准备数据：student.txt

存储在 HDFS 中的/stuent 目录中

将程序打成 jar 包，上传到客户机，提高 Spark 任务：

```
// 先清空最终的输出结果的目录：
hadoop fs -rm -r hdfs://myha01/student/output_sparksql2

// 提交 Spark 的任务：
$SPARK_HOME/bin/spark-submit \
--class com.mazh.spark.sql.StudentSparkSQL2 \
--master spark://hadoop02:7077,hadoop04:7077 \
/home/hadoop/Spark_SQL-1.0-SNAPSHOT.jar \
hdfs://myha01/student/student.txt \
hdfs://myha01/student/output_sparksql2

// 查看 Spark 任务的结果：
hadoop fs -cat hdfs://myha01/student/output_sparksql2/p*
```

最终结果：

```
[hadoop@hadoop05 ~]$ hadoop fs -cat hdfs://myha01/student/output_sparksql2/p*
{"department":"MA","total":8}
{"department":"CS","total":7}
[hadoop@hadoop05 ~]$
```

5、数据源

官网：http://spark.apache.org/docs/latest/sql-programming-guide.html#data-sources

5.1、通用的 Load 和 Save 功能

第一步：先准备数据

[hadoop@hadoop04 resources]$ **hadoop fs -mkdir -p /spark/sql/input/**

[hadoop@hadoop04 resources]$ **hadoop fs -put $SPARK_HOME/examples/src/main/resources/users.parquet /spark/sql/input/**

[hadoop@hadoop04 resources]$ **hadoop fs -ls /spark/sql/input/**

```
[hadoop@hadoop04 resources]$ hadoop fs -mkdir -p /spark/sql/input/
[hadoop@hadoop04 resources]$ hadoop fs -put $SPARK_HOME/examples/src/main/resources/users.parquet /spark/sql/input/
[hadoop@hadoop04 resources]$ hadoop fs -ls /spark/sql/input/
Found 1 items
-rw-r--r--   2 hadoop supergroup        615 2018-05-04 17:27 /spark/sql/input/users.parquet
[hadoop@hadoop04 resources]$
```

第二步：编写普通的 load 和 save 功能：

spark.read.load("hdfs://myha01/spark/sql/input/users.parquet").select("name", "favorite_color").write.save("hdfs://myha01/spark/sql/output")

```
[hadoop@hadoop04 ~]$ $SPARK_HOME/bin/spark-shell --master spark://hadoop02:7077,hadoop04:7077 --executor-memory 512m --total-executor-cores 2
2018-05-04 17:40:28 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://hadoop04:4040
Spark context available as 'sc' (master = spark://hadoop02:7077,hadoop04:7077, app id = app-20180504174037-0006).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.

scala> spark.read.load("hdfs://myha01/spark/sql/input/users.parquet").select("name", "favorite_color").write.save("hdfs://myha01/spark/sql/output")
2018-05-04 17:41:19 WARN  ObjectStore:568 - Failed to get database global_temp, returning NoSuchObjectException

scala>
```

第三步：验证结果

```
[hadoop@hadoop04 resources]$ hadoop fs -ls hdfs://myha01/spark/sql/output/
Found 2 items
-rw-r--r--   2 hadoop supergroup          0 2018-05-04 17:41 hdfs://myha01/spark/sql/output/_SUCCESS
-rw-r--r--   2 hadoop supergroup        590 2018-05-04 17:41 hdfs://myha01/spark/sql/output/part-00000-5a98b83e-dd4d-4ec9-a0b6-fc7e37b2ed21-c000.snappy.parquet
[hadoop@hadoop04 resources]$
```

第四步：指定 load 和 save 的特定文件格式

[hadoop@hadoop04 resources]$ hadoop fs -put people.json hdfs://myha01/spark/sql/input/

```
val peopleDF = spark.read.format("json").load("hdfs://myha01/spark/sql/input/people.json")
peopleDF.select("name", "age").write.format("csv").save("hdfs://myha01/spark/sql/csv")
```

```
scala> val peopleDF = spark.read.format("json").load("hdfs://myha01/spark/sql/input/people.json")
peopleDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> peopleDF.select("name", "age").write.format("csv").save("hdfs://myha01/spark/sql/csv")
```

最终结果：

```
hadoop02 | hadoop02 (1) | hadoop03 | hadoop04 | hadoop04 (1) | hadoop05
[hadoop@hadoop04 resources]$ hadoop fs -cat hdfs://myha01/spark/sql/input/people.json
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
[hadoop@hadoop04 resources]$ hadoop fs -cat hdfs://myha01/spark/sql/csv/p*
Michael,
Andy,30
Justin,19
[hadoop@hadoop04 resources]$
```

第五种：直接查询

val sqlDF = spark.sql("SELECT * FROM parquet.`hdfs://myha01/spark/sql/input/users.parquet`")

```
scala> val sqlDF = spark.sql("SELECT * FROM parquet.`hdfs://myha01/spark/sql/input/users.parquet`")
2018-05-04 17:56:48 WARN  ObjectStore:568 - Failed to get database parquet, returning NoSuchObjectException
sqlDF: org.apache.spark.sql.DataFrame = [name: string, favorite_color: string ... 1 more field]

scala> sqlDF.printSchema
root
 |-- name: string (nullable = true)
 |-- favorite_color: string (nullable = true)
 |-- favorite_numbers: array (nullable = true)
 |    |-- element: integer (containsNull = true)


scala> sqlDF.collect
res7: Array[org.apache.spark.sql.Row] = Array([Alyssa,null,WrappedArray(3, 9, 15, 20)], [Ben,red,WrappedArray()])

scala> sqlDF.count
res8: Long = 2

scala>
```

或者：

```
scala> val sqlDF = spark.sql("SELECT * FROM json.`hdfs://myha01/spark/sql/input/people.json`")
2018-05-04 17:59:30 WARN  ObjectStore:568 - Failed to get database json, returning NoSuchObjectException
sqlDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> sqlDF.printSchema
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)


scala> sqlDF.collect
res10: Array[org.apache.spark.sql.Row] = Array([null,Michael], [30,Andy], [19,Justin])

scala> sqlDF.count
res11: Long = 3
```

5.2、Save Modes

Save operations can optionally take a `SaveMode`, that specifies how to handle existing data if present. It is important to realize that these save modes do not utilize any locking and are not atomic. Additionally, when performing an `Overwrite`, the data will be deleted before writing out the new data.

Scala/Java	Any Language	Meaning
`SaveMode.ErrorIfExists` (default)	`"error"` or `"errorifexists"` (default)	When saving a DataFrame to a data source, if data already exists, an exception is expected to be thrown.
`SaveMode.Append`	`"append"`	When saving a DataFrame to a data source, if data/table already exists, contents of the DataFrame are expected to be appended to existing data.
`SaveMode.Overwrite`	`"overwrite"`	Overwrite mode means that when saving a DataFrame to a data source, if data/table already exists, existing data is expected to be overwritten by the contents of the DataFrame.
`SaveMode.Ignore`	`"ignore"`	Ignore mode means that when saving a DataFrame to a data source, if data already exists, the save operation is expected to not save the contents of the DataFrame and to not change the existing data. This is similar to a `CREATE TABLE IF NOT EXISTS` in SQL.

5.3、JDBC

Spark SQL 可以通过 JDBC 从关系型数据库中读取数据的方式创建 DataFrame，通过对 DataFrame 一系列的计算后，还可以将数据再写回关系型数据库中。

5.3.1、从 MySQL 中加载数据（Spark Shell 方式）

1、启动 Spark Shell，必须指定 mysql 连接驱动 jar 包

启动本机的单进程 Shell

```
$SPARK_HOME/bin/spark-shell \
--jars $SPARK_HOME/mysql-connector-java-5.1.40-bin.jar \
--driver-class-path $SPARK_HOME/mysql-connector-java-5.1.40-bin.jar
```

启动连接 Spark 集群的 Shell

```
$SPARK_HOME/bin/spark-shell \
--master spark://hadoop02:7077,hadoop04:7077 \
--jars $SPARK_HOME/mysql-connector-java-5.1.40-bin.jar \
--driver-class-path $SPARK_HOME/mysql-connector-java-5.1.40-bin.jar
```

2、从 mysql 中加载数据

```
val jdbcDF = sqlContext.read.format("jdbc").options(Map("url" -> "jdbc:mysql://hadoop02:3306/spider", "driver" -> "com.mysql.jdbc.Driver", "dbtable" -> "lagou", "user" -> "root", "password" -> "root")).load()
```

3、执行查询

```
jdbcDF.limit(3).show()
```

```
scala> jdbcDF.limit(3).show
+---+---------+------+--------------------+------+-------+-----+-------+---------+-------+
| id|    t_job|t_addr|               t_tag| t_com|t_money|t_edu|  t_exp|   t_type|t_level|
+---+---------+------+--------------------+------+-------+-----+-------+---------+-------+
|  1|  Java高级|  北苑|        信息安全,大数据,架构|  闲徕互娱|20k-40k|  本科|经验5-10年|       游戏|  不需要融资|
|  2|Java开发工程师|  金融街|                中级|深圳雁联技术| 8k-16k|  本科| 经验3-5年| 移动互联网,金融|  不需要融资|
|  3|Java开发工程师|北京大学|专家,总监,资深,高级,C/C++...|zilliz|15k-30k|  本科| 经验3-5年|企业服务,数据服务|     A轮|
+---+---------+------+--------------------+------+-------+-----+-------+---------+-------+
```

5.3.2、将数据写入 MySQL 中（Spark Submit 方式）

编写 SparkSQL 的写 JDBC 的程序：

```
package com.mazh.spark.sql

import java.util.Properties

import org.apache.spark.sql.types.{IntegerType, StringType, StructField,
StructType}
import org.apache.spark.sql.{Row, SQLContext}
import org.apache.spark.{SparkConf, SparkContext}

object SparkSQL_JDBC {

  def main(args: Array[String]) {

    val conf = new SparkConf().setAppName("SparkSQL_JDBC")
    val sc = new SparkContext(conf)
    val sqlContext = new SQLContext(sc)
    //通过并行化创建 RDD
//    val studentRDD = sc.parallelize(Array("1 huangbo 33", "2 xuzheng 44", "3
wangbaoqiang 55")).map(_.split(" "))
    //通过读取文件创建 RDD
    val studentRDD = sc.textFile(args(0)).map(_.split(","))
    //通过 StructType 直接指定每个字段的 schema
    val schema = StructType(
      List(
        StructField("id", IntegerType, true),
        StructField("name", StringType, true),
        StructField("sex", StringType, true),
        StructField("age", IntegerType, true),
        StructField("department", StringType, true)
      )
    )
    //将 RDD 映射到 rowRDD
    val rowRDD = studentRDD.map(p => Row(p(0).toInt, p(1).trim, p(2).trim, p(3).toInt,
p(4).trim))
    //将 schema 信息应用到 rowRDD 上
```

```
    val studentDataFrame = sqlContext.createDataFrame(rowRDD, schema)
    //创建 Properties 存储数据库相关属性
    val prop = new Properties()
    prop.put("user", "root")
    prop.put("password", "root")
    //将数据追加到数据库

studentDataFrame.write.mode("append").jdbc("jdbc:mysql://hadoop02:3306/spider",
"student", prop)
    //停止 SparkContext
    sc.stop()
  }
}
```

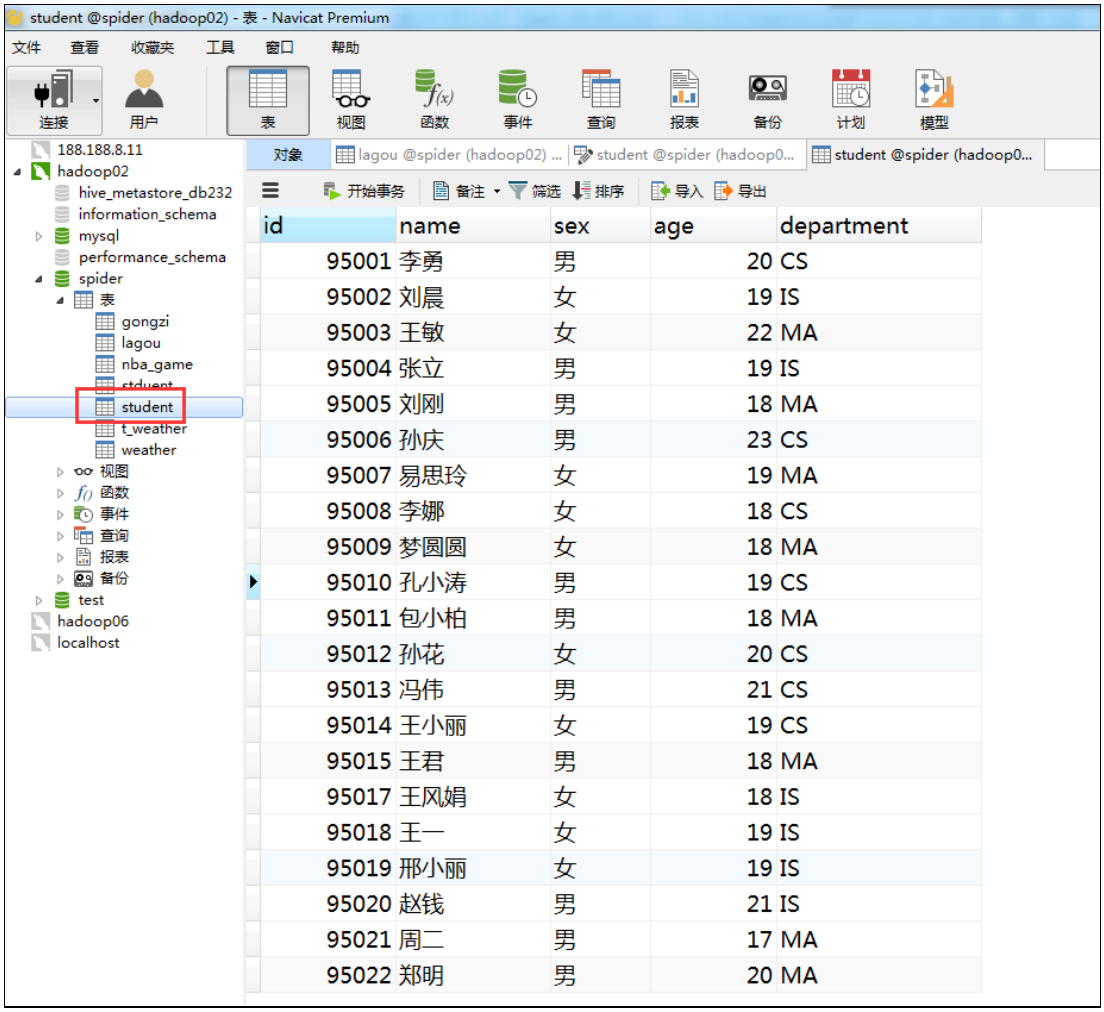
准备数据：student.txt 存储在 HDFS 上的/student 目录中

给项目打成 jar 包，上传到客户端

提交任务给 Spark 集群：

```
$SPARK_HOME/bin/spark-submit \
--class com.mazh.spark.sql.SparkSQL_JDBC \
--master spark://hadoop02:7077,hadoop04:7077 \
--jars $SPARK_HOME/mysql-connector-java-5.1.40-bin.jar \
--driver-class-path $SPARK_HOME/mysql-connector-java-5.1.40-bin.jar \
/home/hadoop/Spark_WordCount-1.0-SNAPSHOT.jar \
hdfs://myha01/student/student.txt
```

最终结果：



5.4、JSON

```
scala> import spark.implicits._
import spark.implicits._

scala> val parquetFileDF = spark.read.parquet("hdfs://myha01/spark/sql/input/people.parquet")
parquetFileDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> val peopleDF = parquetFileDF.write.json("hdfs://myha01/spark/sql/output/people_copy.json")
peopleDF: Unit = ()

scala> val jsonFileDF  = spark.read.json("hdfs://myha01/spark/sql/output/people_copy.json")
jsonFileDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> jsonFileDF.createOrReplaceTempView("json_table")

scala> val namesDF = spark.sql("SELECT name FROM json_table WHERE age BETWEEN 13 AND 19")
namesDF: org.apache.spark.sql.DataFrame = [name: string]

scala> namesDF.map(attributes => "Name: " + attributes(0)).show()
+------------+
|       value|
+------------+
|Name: Justin|
+------------+
```

代码：

```
package com.mazh.sparksql


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

```
/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  * 描述:  读取 JSON
  */
object TestSparkSQL_ReadJSON {

  def main(args: Array[String]): Unit = {

    // 构建 SparkSQL 程序的编程入口对象 SparkSession
    val sparkSession:SparkSession = SparkSession.builder()
      .appName("MyFirstSparkSQL")
      .config("someKey", "someValue")
      .master("local")
      .getOrCreate()

    // 方式 1
    val df1 = sparkSession.read.json("D:\\bigdata\\json\\people.json")

    // 方式 2
    val df2 = sparkSession.read.format("json").load("D:\\bigdata\\json\\people.json")

  }
}
```

5.5、Parquet Files

```
scala> val peopleDF = spark.read.json("hdfs://myha01/spark/sql/input/people.json")
peopleDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> import spark.implicits._
import spark.implicits._

scala> peopleDF.write.parquet("hdfs://myha01/spark/sql/input/people.parquet")

scala> val parquetFileDF = spark.read.parquet("hdfs://myha01/spark/sql/input/people.parquet")
parquetFileDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> parquetFileDF.createOrReplaceTempView("parquetFile")

scala> val namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN 13 AND 19")
namesDF: org.apache.spark.sql.DataFrame = [name: string]

scala> namesDF.map(attributes => "Name: " + attributes(0)).show()
+------------+
|       value|
+------------+
|Name: Justin|
+------------+


scala>
```

代码：

```
package com.mazh.sparksql
```

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

/**
  * 作者：  马中华：http://blog.csdn.net/zhongqi2513
  * 描述：  读取 JSON
  */
object TestSparkSQL_ReadParquet {

  def main(args: Array[String]): Unit = {

    // 构建 SparkSQL 程序的编程入口对象 SparkSession
    val sparkSession:SparkSession = SparkSession.builder()
      .appName("MyFirstSparkSQL")
      .config("someKey", "someValue")
      .master("local")
      .getOrCreate()

    // 方式 1
    val df1 = sparkSession.read.parquet("D:\\bigdata\\parquet\\people.parquet")

    // 方式 2
    val df2 = sparkSession.read.format("parquet")
      .load("D:\\bigdata\\json\\people.json")

  }
}
```