# # day03 [Spark SQL]

## ## 今日内容

- Spark SQL概述

- Spark SQL原理

- 三大弹性分布式数据集之DataFrame/DataSet数据结构和使用方式

- Spark SQL完成计算任务

## 教学目标

- 掌握Spark SQL的原理

- 掌握DataFrame/DataSet数据结构和使用方式

- 熟练使用Spark SQL完成计算任务

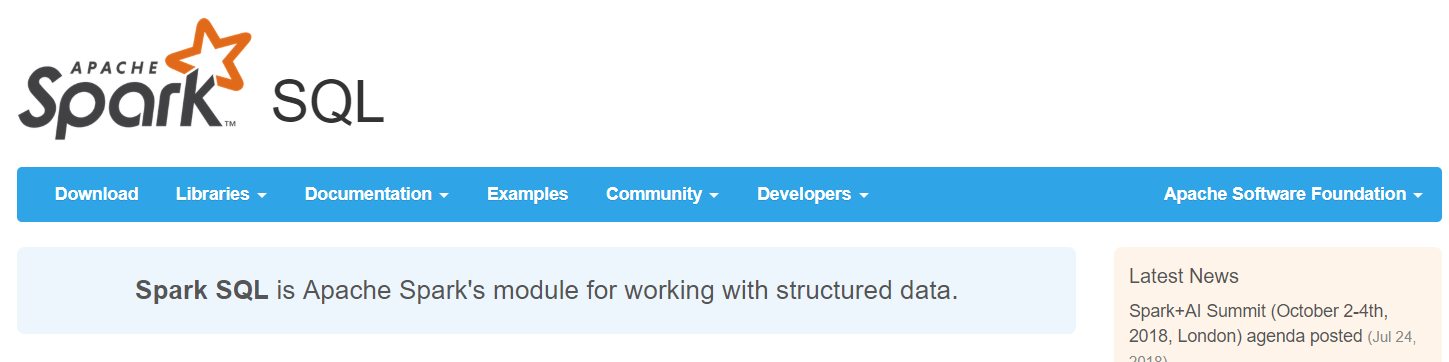
# # 第一章 Spark SQL概述

## ## 1.1 Spark SQL的历史

Shark是一个为Spark设计的大规模数据仓库系统，它与Hive兼容。Shark建立在Hive的代码基础上，并通过将Hive的部分物理执行计划交换出来。这个方法使得Shark的用户可以加速Hive的查询，但是Shark继承了Hive的大且复杂的代码使得Shark很难优化和维护，同时Shark依赖于Spark的版本。随着我们遇到了性能优化的上限，以及集成SQL的一些复杂的分析功能，我们发现Hive的MapReduce设计的框架限制了Shark的发展。在2014年7月1日的Spark Summit上，Databricks宣布终止对Shark的开发，将重点放到Spark SQL上。

## ## 1.2 什么是Spark SQL

![](img\Spark SQL官网简介.png)



Spark SQL是Spark用来处理结构化数据的一个模块，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。

相比于Spark RDD API，Spark SQL包含了对结构化数据和在其上运算的更多信息，Spark SQL使用这些信息进行了额外的优化，使对结构化数据的操作更加高效和方便。

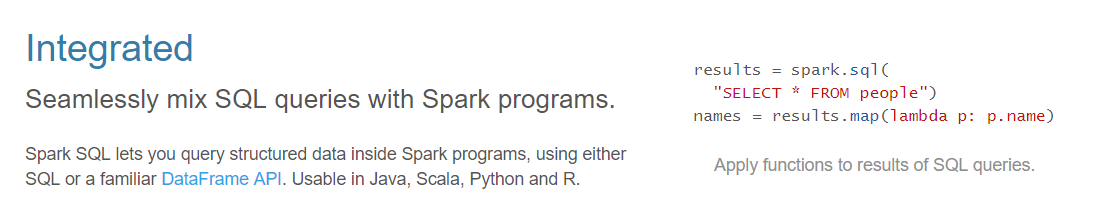
有多种方式去使用Spark SQL，包括SQL、DataFrames API和Datasets API。但无论是哪种API或者是编程语言，它们都是基于同样的执行引擎，因此你可以在不同的API之间随意切换，它们各有各的特点，看你喜欢那种风格。

## ## 1.3 Spark SQL的特点

我们已经学习了Hive，它是将Hive SQL转换成MapReduce然后提交到集群中去执行，大大简化了编写MapReduce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应运而生，它是将Spark SQL转换成RDD，然后提交到集群中去运行，执行效率非常快！

### ### 1.3.1 易整合

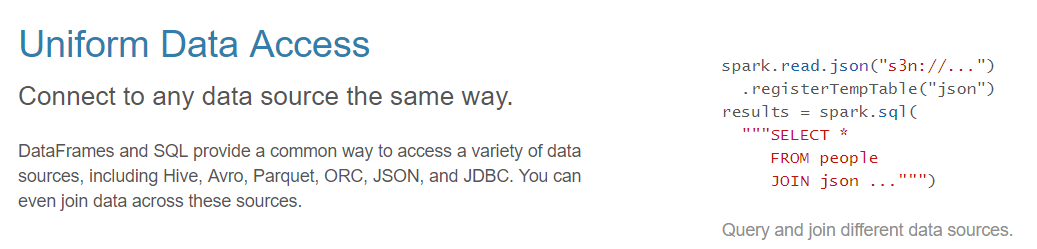
![](img\Spark SQL特点一\_易整合.png)



将sql查询与spark程序无缝混合，可以使用java、scala、python、R等语言的API操作。

### ### 1.3.2 统一的数据访问

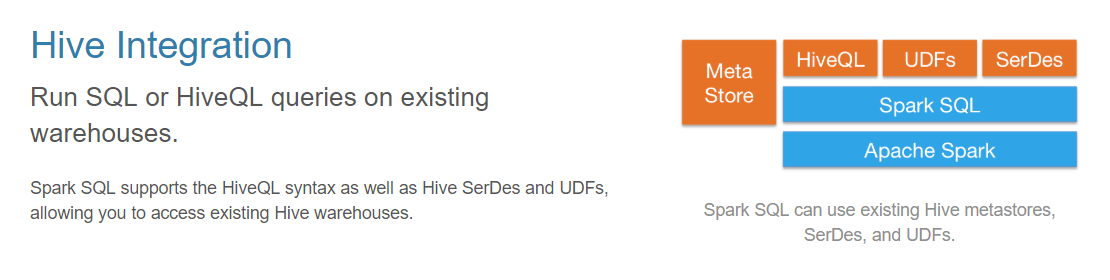
![](img\Spark SQL特点二\_统一的数据访问.png)



以相同的方式连接到任何数据源

### ### 1.3.3 兼容Hive

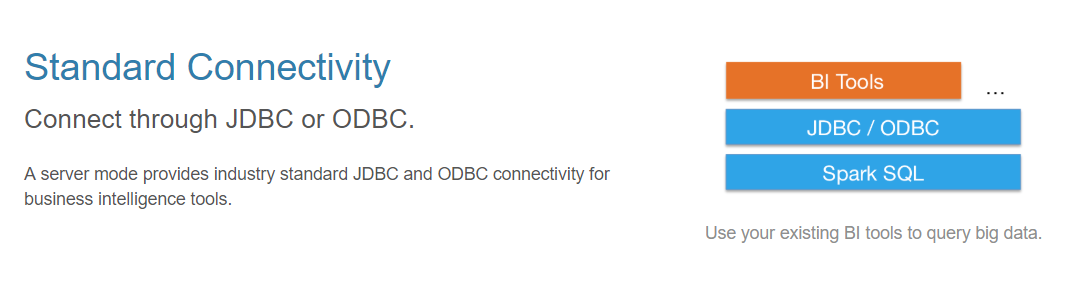
![](img\Spark SQL特点三\_兼容Hive.png)



支持hiveSQL的语法。

### ### 1.3.4 标准的数据连接

![](img\Spark SQL特点四\_标准的数据库连接.png)



可以使用行业标准的JDBC或ODBC连接。

# # 第二章 弹性分布式数据集之DataFrame

## ## 2.1 什么是DataFrame

DataFrame的前身是SchemaRDD，从Spark 1.3.0开始SchemaRDD更名为DataFrame。与SchemaRDD的主要区别是：DataFrame不再直接继承自RDD，而是自己实现了RDD的绝大多数功能。你仍旧可以在DataFrame上调用rdd方法将其转换为一个RDD。

在Spark中，DataFrame是一种以RDD为基础的分布式数据集，类似于传统数据库的二维表格，DataFrame带有Schema元信息，即DataFrame所表示的二维表数据集的每一列都带有名称和类型，但底层做了更多的优化。DataFrame可以从很多数据源构建，比如：已经存在的RDD、结构化文件、外部数据库、Hive表。

## ## 2.2 DataFrame与RDD的区别

RDD可看作是分布式的对象的集合，Spark并不知道对象的详细模式信息，DataFrame可看作是分布式的Row对象的集合，其提供了由列组成的详细模式信息，使得Spark SQL可以进行某些形式的执行优化。DataFrame和普通的RDD的逻辑框架区别如下所示：

![](img\DataFrame和RDD区别.png)



上图直观地体现了DataFrame和RDD的区别。左侧的RDD[Person]虽然以Person为类型参数，但Spark框架本身不了解Person类的内部结构。

而右侧的DataFrame却提供了详细的结构信息，使得Spark SQL可以清楚地知道该数据集中包含哪些列，每列的名称和类型各是什么，DataFrame多了数据的结构信息，即schema。这样看起来就像一张表了，DataFrame还配套了新的操作数据的方法，DataFrame API（如df.select())和SQL(select id, name from xx\_table where ...)。

此外DataFrame还引入了off-heap,意味着JVM堆以外的内存,这些内存直接受操作系统管理（而不是JVM）。Spark能够以二进制的形式序列化数据(不包括结构)到off-heap中,当要操作数据时,就直接操作off-heap内存.由于Spark理解schema,

所以知道该如何操作。

RDD是分布式的Java对象的集合。DataFrame是分布式的Row对象的集合。DataFrame除了提供了比RDD更丰富的算子以外，更重要的特点是提升执行效率、减少数据读取以及执行计划的优化。

有了DataFrame这个高一层的抽象后，我们处理数据更加简单了，甚至可以用SQL来处理数据了，对开发者来说，易用性有了很大的提升。

不仅如此，通过DataFrame API或SQL处理数据，会自动经过Spark优化器（Catalyst）的优化，即使你写的程序或SQL不高效，也可以运行的很快。

## ## 2.3 DataFrame与RDD的优缺点

### ### 2.3.1 RDD的优缺点

优点:

- 编译时类型安全

- 编译时就能检查出类型错误

- 面向对象的编程风格

- 直接通过对象调用方法的形式来操作数据

缺点:

- 序列化和反序列化的性能开销

- 无论是集群间的通信,还是IO操作都需要对对象的结构和数据进行序列化和反序列化.

- GC的性能开销

- 频繁的创建和销毁对象,势必会增加GC

### ### 2.3.2 DataFrame的优缺点

DataFrame通过引入schema和off-heap（不在堆里面的内存，指的是除了不在堆的内存，使用操作系统上的内存），解决了RDD的缺点,Spark通过schame就能够读懂数据,因此在通信和IO时就只需要序列化和反序列化数据,而结构的部分就可以省略了；通过off-heap引入，可以快速的操作数据，避免大量的GC。但是却丢了RDD的优点，DataFrame不是类型安全的,API也不是面向对象风格的。

## ## 2.4 创建DataFrame

### ### 2.4.1 读取文本文件创建DataFrame

在spark2.0版本之前，Spark SQL中SQLContext是创建DataFrame和执行SQL的入口，可以利用hiveContext通过hive sql语句操作hive表数据，兼容hive操作，并且hiveContext继承自SQLContext。在spark2.0之后，这些都统一于SparkSession，SparkSession封装了SparkContext，SqlContext，通过SparkSession可以获取SparkConetxt,SqlContext对象。

#### #### 2.4.1.1 添加pom配置

在pom.xml中需要添加Spark SQL的配置

```xml

<!--添加Spark SQL的依赖-->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql\_${scala.binary.version}</artifactId>

<version>${spark.version}</version>

</dependency>

```

#### #### 2.4.1.1 Java语言实现

```java

/\*\*

\* @author JackieZhu

\* @date 2018-08-21 16:13:50

\* @description 我是传智JackieZhu, 我是不一样的JackieZhu

\*/

public class CreateDataFrameWithJava {

public static void main(String[] args) {

//1:创建一个SparkSession对象,从Spark2.0版本以后,都被整合进入了SparkSession

SparkSession spark = SparkSession

.builder()

.appName("CreateDataFrameWithJava")

.master("local")

.getOrCreate();

//2:读取本地文件,这里返回的是一个Dataset对象,DataFrame本身也是一个Dataset对象

Dataset<String> df = spark.read().textFile("f:\\data\\people.txt");

//3:打印DataFrame的基本信息

df.show();

//4:释放资源

spark.close();

}

}

```

#### #### 2.4.1.2 Scala语言实现

```scala

/\*\*

\* @author JackieZhu

\* @note 2018-08-21 16:45:25

\* @desc 我是传智JackieZhu 我是不一样的JackieZhu

\*/

object CreateDataFrameWithScala {

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

//1:创建一个SparkSession对象

val spark = SparkSession

.builder()

.appName("Spark SQL basic example")

.master("local")

.getOrCreate()

//2:读取本地文件

val df: Dataset[String] = spark.read.textFile("f:\\data\\people.txt")

//3:打印信息

df.show()

//4:释放资源

spark.close()

}

}

```

### ### 2.4.2 使用spark-shell完成DataFrame的测试

#### #### 2.4.2.1 在本地创建一个文本文件,并上传HDFS

在本地创建一个文件，有三列，分别是id、name、age，用空格分隔，然后上传到hdfs上。person.txt内容为：

```text

1 zhangsan 20

2 lisi 23

3 wangwu 30

4 zhaoliu 21

5 tianqi 22

6 wangba 18

```

将上述文本文件上传至HDFS文件系统中

```shell

hdfs dfs -put person.txt /

```

#### #### 2.4.2.2 读取数据

```scala

val lineRDD = sc.textFile("/person.txt").map(line=>line.split(" "))

```

在spark-shell命令行中读取一行数据,将每一行的数据使用空格分隔符进行分割.

#### #### 2.4.2.3 定义case class(相当于表的schema)

```scala

case class Person(id:Int,name:String,age:Int)

```

#### #### 2.4.2.4 将RDD于case class关联

```scala

val personRDD = lineRDD.map(x=>Person(x(0).toInt,x(1),x(2).toInt))

```

#### #### 2.4.2.5 将RDD转换成DataFrame

```scala

val personDF = personRDD.toDF

```

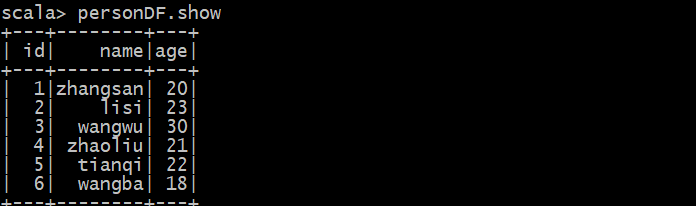
#### #### 2.4.2.6 对DataFrame进行处理

```scala

personDF.show

```

![](img\show.png)

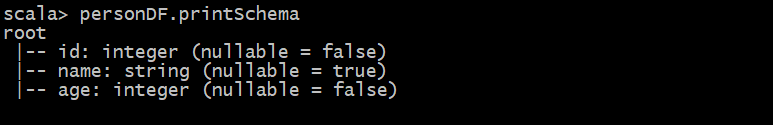


```scala

personDF.printSchema

```

![](img\printSchema.png)



#### #### 2.4.2.7 使用SparkSession构建DataFrame

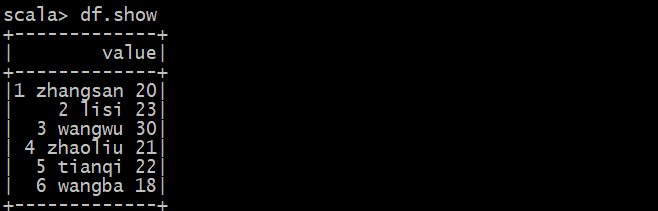
使用spark-shell中已经初始化好的SparkSession对象spark生成DataFrame

```scala

val df = spark.read.text("/person.txt")

```

![](img\show1.png)



### ### 2.4.3 读取json文件创建DataFrame

#### #### 2.4.3.1 读取数据文件people.json

数据文件我们使用spark安装包提供的examples包下的/opt/modules/spark-2.2.0/examples/src/main/resources/people.json文件.

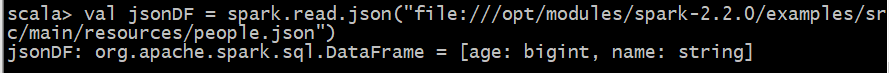
#### #### 2.4.3.2 在spark-shell中执行下面命令,读取数据

```scala

val jsonDF = spark.read.json("file:///opt/modules/spark-2.2.0/examples/src/main/resources/people.json")

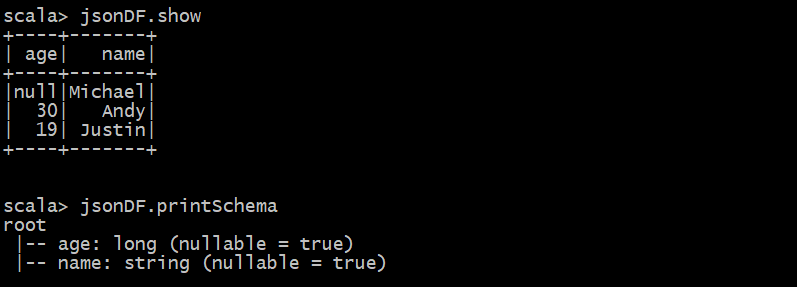
```

![](img\jsonDF.png)



#### #### 2.4.3.3 使用DataFrame的函数进行操作

![](img\jsonDF.show&printSchema.png)



### ### 2.4.4 读取parquet列式存储格式文件创建DataFrame

#### #### 2.4.4.1 读取数据文件users.parquet

数据文件我们使用spark安装包提供的examples包下的/opt/modules/spark-2.2.0/examples/src/main/resources/users.parquet文件.

#### #### 2.4.4.2 使用spark-shell读取文件

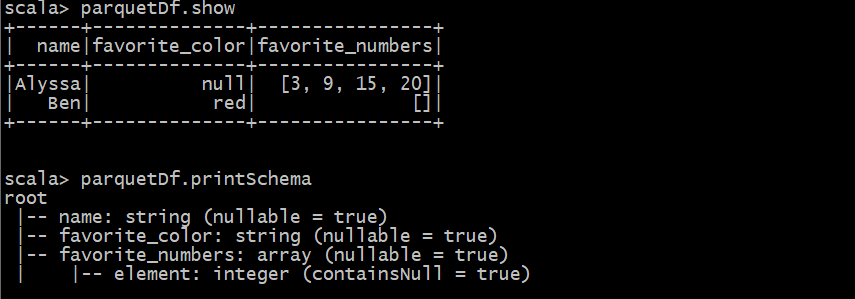
```scala

val parquetDf = spark.read.parquet("file:///opt/modules/spark-2.2.0/examples/src/main/resources/users.parquet")

```

#### #### 2.4.4.3 使用DataFrame的函数操作

![](img\parquetDf.png)



# # 第三章 DataFrame常用操作

## ## 3.1 DSL风格语法

DataFrame提供了一个特定领域语言(DSL)来操作结构化数据.接下来式DataFrame的一些案例

### ### 3.1.1 show

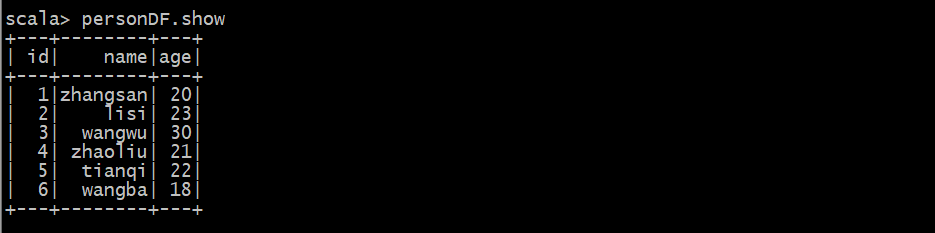
查看DataFrame中的内容,通过调用show方法

```scala

personDF.show

```

![](img\personDF.show.png)



### ### 3.1.2 查看某一列内容

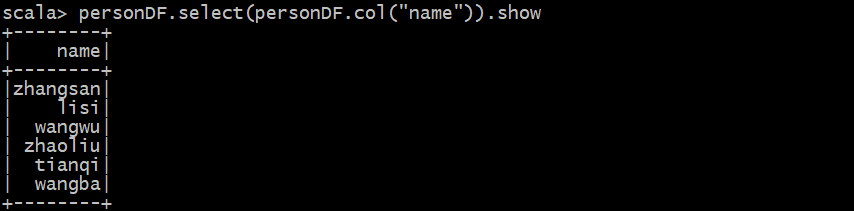
查看DataFrame部分列中的内容,查看name字段的数据

```scala

personDF.select(personDF.col("name")).show

```

![](img\查看一个字段内容.png)



查看name字段的另一种写法

```scala

personDF.select("name").show

```

查看name字段和age字段

```scala

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

```

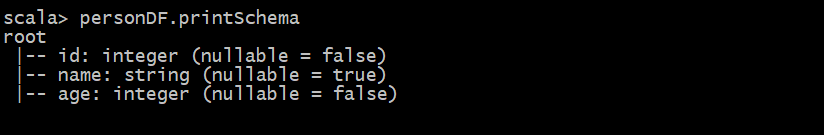
### ### 3.1.3 打印DataFrame的Schema信息

```scala

personDF.printSchema

```

![](img\打印Schema信息.png)



### ### 3.1.4 查询所有的name和age,并将age+1

```scala

//第一种方式

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

//第二种方式

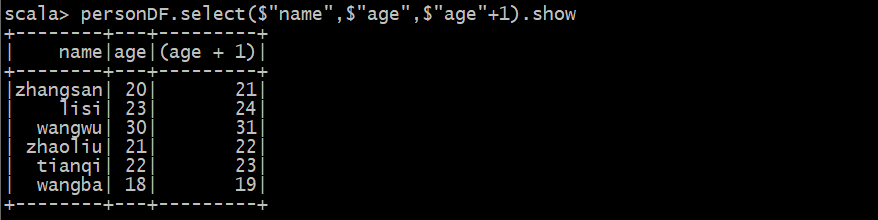
personDF.select($"name",$"age",$"age"+1).show

//第三种方式

personDF.select(personDF("name"),personDF("age"),personDF("age")+1).show

```

![](img\查询某些字段并进行计算.png)



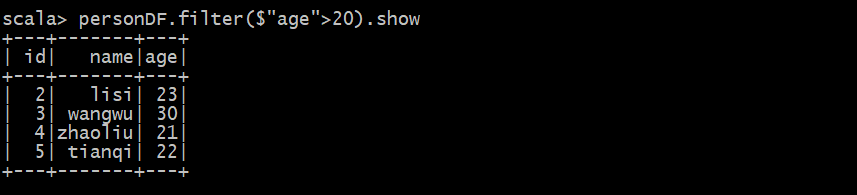
### ### 3.1.5 过滤age大于等于20的.

```scala

personDF.filter($"age">20).show

```

![](img\年龄大于20.png)



### ### 3.1.6 统计年龄大于20的人数

```scala

personDF.filter($"age">20).count()

```

![](img\统计大于20的人数.png)



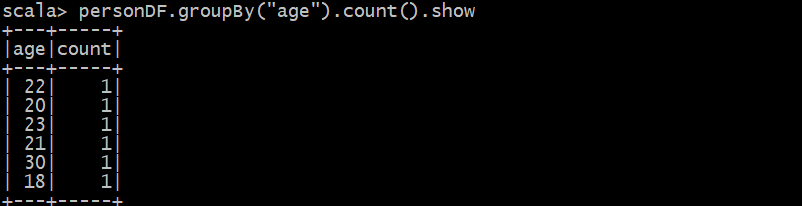
### ### 3.1.7 按年龄进行分组并统计相同年龄的人数

```scala

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

```

![](img\按照年龄进行分组.png)



## ## 3.2 SQL风格语法

DataFrame的一个强大之处就是我们可以将它看作是一个关系型数据表，然后可以通过在程序中使用spark.sql() 来执行SQL语句查询，结果返回一个DataFrame。

如果想使用SQL风格的语法，需要将DataFrame注册成表,采用如下的方式：

```scala

//已经过时了

personDF.registerTempTable("tb\_person")

//推荐使用这种方式

personDF.createOrReplaceTempView("tb\_person")

createOrReplaceTempView的意思是若数据库中已经存在这个名字的视图的话，就替代它，若没有则创建视图

off-heap堆外内存，与之对应的是堆内内存

堆是内存中动态分配对象的存在的地方，如果使用new一个对象就会分配到堆内存上面。这个相对于stack而言，如果有一个局部变量它是位于stack内存空间。

一般情况下，java虚拟机分配的非空对象都有垃圾回收器管理，也称为堆内内存（on-heap memory）,虚拟机会定期对垃圾内存进行回收，会在特定的时间进行一次彻底的回收。回收时，垃圾回收器会对所有分配的堆内内存进行完整的扫描。这意味着这样一次来及回收对java应用造成的影响，跟堆的大小是成正比的，过大的堆影响java应用的性能。

off-heap有一下特点

1.对于大内存有良好的伸缩性

2.堆垃圾回收的改善可以明显感受到

3.在进程之间可共享，减少虚拟机之间的复制

应用场景：

1.Session会话缓存，保存不激活的用户session，比如用户没有正常退出，我们也无法确定他会不会短时间内再回来，将其会话存到堆外内存。一旦再次登录，无需访问数据库可再次激活。

2。计算结果的缓存，大量查询结果等，击中率比较低的都可以迁移到堆外

```

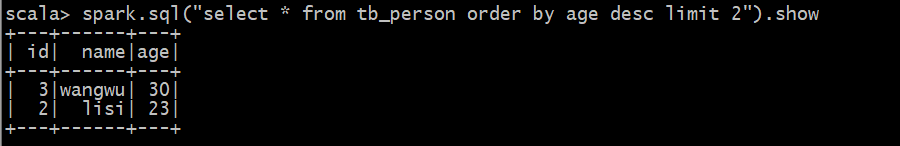
### ### 3.2.1 查询年龄最大的前两名

```scala

spark.sql("select \* from tb\_person order by age desc limit 2").show

```

![](img\sparksql查询前两名.png)



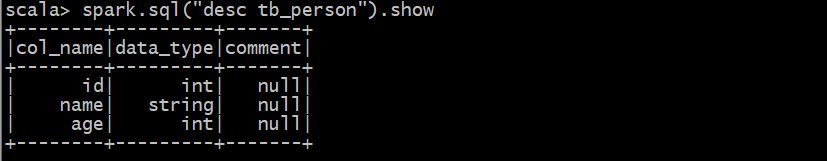
### ### 3.2.2 显示表的Schema信息

```scala

spark.sql("desc tb\_person").show

```

![](img\sparksql查看schema信息.png)



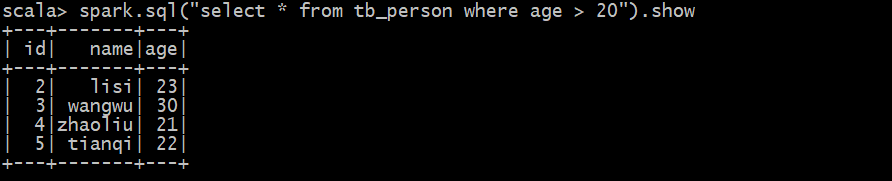
### ### 3.2.3 查询年龄大于20的人的信息

```scala

spark.sql("select \* from tb\_person where age > 20").show

```

![](img\sparksql查看年龄大于30的人的信息.png)



# # 第四章 DataSet

## ## 4.1 什么是DataSet

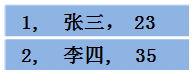
DataSet是分布式的数据集合，Dataset提供了强类型支持，也是在RDD的每行数据加了类型约束。DataSet是在Spark1.6中添加的新的接口。它集中了RDD的优点（强类型和可以用强大lambda函数）以及使用了Spark

SQL优化的执行引擎。DataSet可以通过JVM的对象进行构建，可以用函数式的转换（map/flatmap/filter）进行多种操作。

## ## 4.2 三大弹性分布式数据的区别

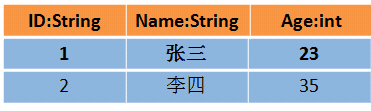
假设RDD中的两行数据长这样:

![](img\rdd的数据.png)



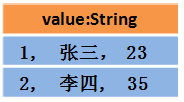
那么DataFrame中的数据长这样:

![](img\DataFrame中的数据.png)



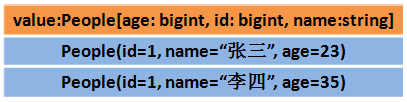
那么DataSet中的数据长这样:

![](img\DataSet中的数据.png)



或者长这样(每行数据是个Object)

![](img\DataSet中的数据1.png)



总结:

DataSet包含了DataFrame的功能，Spark2.0中两者统一，DataFrame表示为DataSet[Row]，即DataSet的子集。DataSet可以在编译时检查类型,并且是面向对象的编程接口.相比DataFrame,DataSet提供了编译时类型检查,对于分布式程序来讲,提交一次作业太费劲了(要编译,打包,上传,运行).到提交到集群运行时才发现错误.这会浪费大量的时间,这也是引入Dataset的一个重要原因.

## ## 4.3 DataFrame与DataSet互相转换

DataFrame和DataSet可以互相转换.

### ### 4.3.1 DataFrame转为DataSet

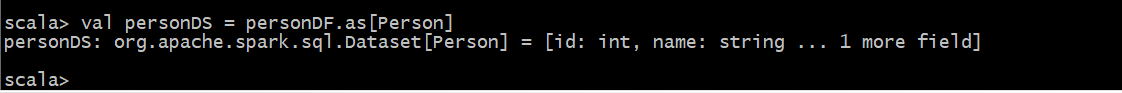
df.as[ElementType]这样可以把DataFrame转化为DataSet

```scala

val personDS = personDF.as[Person]

```

![](img\DataFrame转换为DataSet.png)



### ### 4.3.2 DataSet转为DataFrame

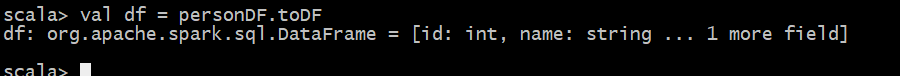
ds.toDF()这样可以把DataSet转换为DataFrame.

```scala

val df = personDF.toDF

```

![](img\DataSet转换为DataFrame.png)



## ## 4.4 创建DataSet

### ### 4.4.1 通过spark.createDataset创建

使用程序中已经存在的一个集合进行创建

```scala

//使用createDataset方法创建一个Dataset集合

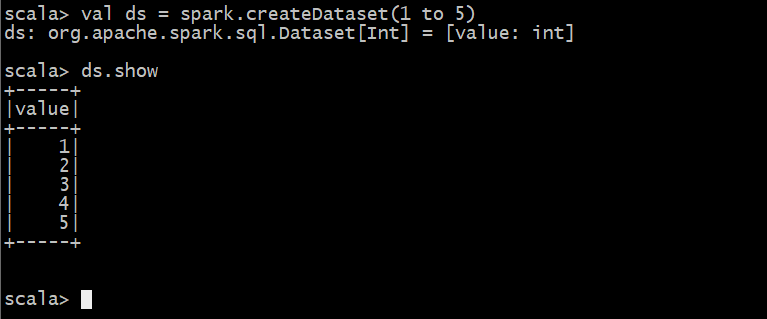
val ds = spark.createDataset(1 to 5)

//调用ds的show方法查看内容

ds.show()

```

![](img\createDataset.png)



读取一个配置文件,使用一个RDD创建一个Dataset

```scala

//读取HDFS系统上的文件

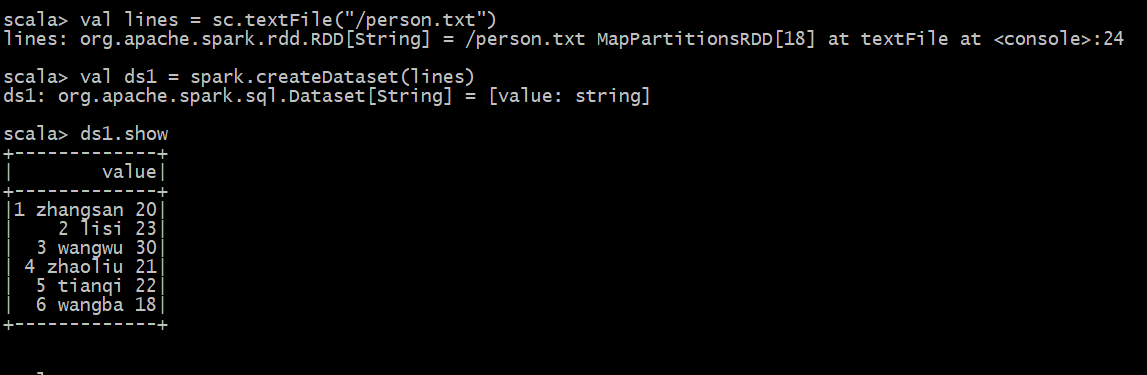
val lines = sc.textFile("/person.txt")

//使用rdd创建一个Dataset

val ds1 = spark.createDataset(lines)

```

![](img\createDataSetWithRDD.png)



### ### 4.4.2 通过toDS方法生成DataSet

```scala

//定义一个样例类

case class Person(name:String,age:Long)

//准备一个集合,集合里面存放Person对象

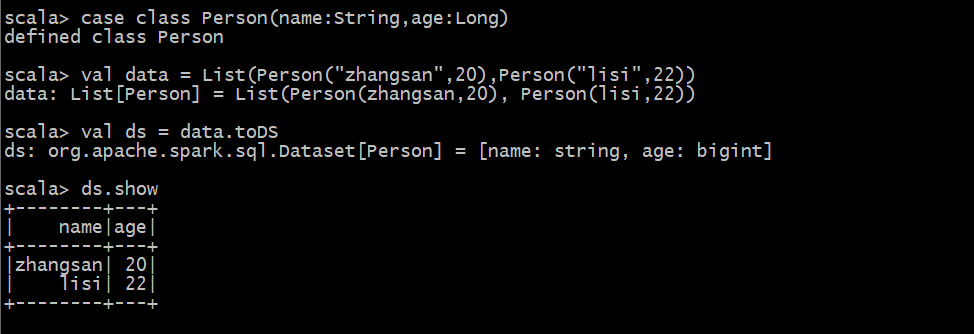
val data = List(Person("zhangsan",20),Person("lisi",22))

//将集合转换为Dataset

val ds = data.toDS

```

![](img\toDS.png)



### ### 4.4.3 通过DataFrame转换生成

使用as[类型]转换为Dataset

```scala

//定义样例类

case class Person(name:String,age:Long)

//读取文件

val jsonDF = spark.read.json("file:///opt/modules/spark-2.2.0/examples/src/main/resources/people.json")

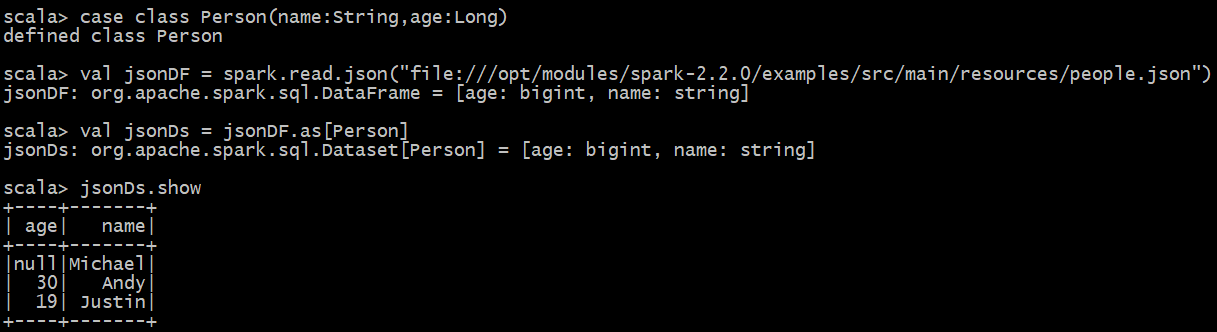
//DataFrame转换为Dataset

val jsonDs = jsonDF.as[Person]

jsonDs.show

```

![](img\dftods.png)



更多DataSet操作API地址：

http://spark.apache.org/docs/2.2.0/api/scala/index.html#org.apache.spark.sql.Dataset

# # 第五章 以编程的方式执行Spark SQL查询

## ## 5.1 编写Spark SQL程序实现RDD转换成DataFrame

前面我们学习了如何在Spark Shell中使用SQL完成查询，现在我们通过IDEA编写Spark SQL查询程序。

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，推导包含某种类型的RDD，通过反射将其转换为指定类型的DataFrame，适用于提前知道RDD的schema。第二种方法通过编程接口与RDD进行交互获取schema，并动态创建DataFrame，在运行时决定列及其类型。

首先在maven项目的pom.xml中添加Spark SQL的依赖。

```scala

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql\_2.11</artifactId>

<version>2.2.0</version>

</dependency>

```

## ## 5.2 通过反射推断Schema

Scala支持使用case class类型导入RDD转换为DataFrame，通过case class创建schema，case class的参数名称会被利用反射机制作为列名。这种RDD可以高效的转换为DataFrame并注册为表。

### ### 5.2.1 Java版本

```java

/\*\*

\* @author JackieZhu

\* @date 2018-08-21 16:13:50

\* @description 我是传智JackieZhu, 我是不一样的JackieZhu

\*/

public class CreateDataFrameWithJava {

public static void main(String[] args) {

//1:创建一个SparkSession对象

SparkSession spark = SparkSession

.builder()

.appName("CreateDataFrameWithJava")

.master("local")

.getOrCreate();

//2:读取本地文件,并创建一个RDD

JavaRDD<String> lines = spark.read().textFile("f:\\data\\people.txt").javaRDD();

//3:将每一行内容转换为Perple对象

JavaRDD<People> peopleRDD = lines.map(new Function<String, People>() {

@Override

public People call(String v1) throws Exception {

String[] arr = v1.split(" ");

return new People(Integer.parseInt(arr[0]), arr[1], Integer.parseInt(arr[2]));

}

});

//将RDD转换为DataFrame DataFrame是DataSet的子集

Dataset<Row> peopleDF = spark.createDataFrame(peopleRDD, People.class);

//===================================DSL语法start===========================

//打印DataFrame的基本信息

peopleDF.show();

//打印peopleDF的Schema信息

peopleDF.printSchema();

//查询某个字段username的信息

peopleDF.select("username").show();

//显示DataFrame的Schema信息

for (String column : peopleDF.columns()) {

System.out.println(column);

}

//显示DataFrame的总的记录数

System.out.println(peopleDF.count());

//查询age字段的内容,并对age字段的值加1

peopleDF.select(new Column("age"),new Column("age").$plus(1)).show();

//过滤出DataFrame中年龄大于25的记录

peopleDF.filter(new Column("age").$greater(25)).show();

//统计DataFrame中年龄大于25的人数

System.out.println(peopleDF.filter(new Column("age").$greater(25)).count());

//统计DataFrame中按照年龄进行分组，求每个组的人数

peopleDF.groupBy("age").count().show();

//===================================DSL语法end===========================

//===================================SQL语法start===========================

peopleDF.createOrReplaceTempView("tb\_people");

spark.sql("select \* from tb\_people").show();

spark.sql("select \* from tb\_people where username='张三'").show();

spark.sql("select \* from tb\_people order by age desc").show();

//===================================SQL语法end===========================

//4:释放资源

spark.close();

}

}

```

### ### 5.2.2 Scala版本

```scala

/\*\*

\* @author JackieZhu

\* @note 2018-08-21 16:45:25

\* @desc 我是传智JackieZhu 我是不一样的JackieZhu

\*/

case class Person(id:Int,username:String,age:Int)

object CreateDataFrameWithScala {

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

//1:创建一个SparkSession对象

val spark = SparkSession

.builder()

.appName("Spark SQL basic example")

.master("local")

.getOrCreate()

//2:从SparkSession中获取SparkContext对象

val sc: SparkContext = spark.sparkContext

sc.setLogLevel("WARN")

//3:读取本地文件,加载数据

val lines: RDD[String] = sc.textFile("f:\\data\\people.txt")

//4:将每一行数据进行切分,并转换成People对象

val peopleRDD: RDD[Person] = lines.map(line=>line.split(" ")).map(attr=>Person(Integer.parseInt(attr(0)),attr(1),Integer.parseInt(attr(2))))

//手动导入隐式转换

import spark.implicits.\_

//5:使用toDF方法,将peopleRDD转换成DataFrame

val peopleDF: DataFrame = peopleRDD.toDF()

//==============================DSL语法====================================

//显示DataFrame中的内容

peopleDF.show()

//2、显示DataFrame的schema信息

peopleDF.printSchema()

//3、显示DataFrame记录数

println(peopleDF.count())

//4、显示DataFrame的所有字段

peopleDF.columns.foreach(println)

//5、取出DataFrame的第一行记录

println(peopleDF.head())

//6、显示DataFrame中name字段的所有值

peopleDF.select("username").show()

//7、过滤出DataFrame中年龄大于25的记录

peopleDF.filter($"age" > 25).show()

//8、统计DataFrame中年龄大于25的人数

println(peopleDF.filter($"age">25).count())

//9、统计DataFrame中按照年龄进行分组，求每个组的人数

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

//==============================DSL语法====================================

//==============================SQL语法====================================

peopleDF.createOrReplaceTempView("tb\_people")

spark.sql("select \* from tb\_people").show()

spark.sql("select \* from tb\_people where username='张三'").show()

spark.sql("select \* from tb\_people order by age desc").show()

//==============================SQL语法====================================

//4:释放资源

spark.close()

}

}

```

## ## 5.3 通过StructType直接指定Schema

当case class不能提前定义好时，可以通过以下三步创建DataFrame

- 将RDD转为包含Row对象的RDD

- 基于StructType类型创建schema，与第一步创建的RDD相匹配

- 通过sparkSession的createDataFrame方法对第一步的RDD应用schema创建DataFrame

### ### 5.3.1 Java版本

```java

/\*\*

\* @author JackieZhu

\* @date 2018-08-23 20:43:39

\* @description 我是传智JackieZhu, 我是不一样的JackieZhu

\*/

public class CreateDataFrameWithStructTypeJava {

public static void main(String[] args) {

//1:创建一个SparkSession对象

SparkSession spark = SparkSession

.builder()

.appName("CreateDataFrameWithJava")

.master("local")

.getOrCreate();

//2:通过SparkSession对象获取SparkContext对象

SparkContext sc = spark.sparkContext();

sc.setLogLevel("WARN");

//3:读取本地文件,加载数据

JavaRDD<String> peopleRDD = sc.textFile("f:\\data\\people.txt", 1)

.toJavaRDD();

//4:定义schema信息

String schemaString = "id username age";

//5:创建结构化字段

List<StructField> fields = new ArrayList<>();

String[] arr = schemaString.split(" ");

for (String fieldName : arr) {

StructField structField = DataTypes.createStructField(fieldName, DataTypes.StringType, true);

fields.add(structField);

}

//创建StructType

StructType schema = DataTypes.createStructType(fields);

JavaRDD<Row> rowRDD = peopleRDD.map(new Function<String, Row>() {

@Override

public Row call(String v1) throws Exception {

String[] attributes = v1.split(" ");

return RowFactory.create(attributes[0],attributes[1],attributes[2]);

}

});

//将schema应用到RDD上

Dataset<Row> dataFrame = spark.createDataFrame(rowRDD, schema);

dataFrame.show();

//将dataFrame注册成一张临时表

dataFrame.createOrReplaceTempView("tb\_people");

spark.sql("select \* from tb\_people").show();

spark.sql("select count(\*) from tb\_people").show();

}

}

```

### ### 5.3.2 Scala语言

```scala

/\*\*

\* @author JackieZhu

\* @note 2018-08-23 21:02:06

\* @desc 我是传智JackieZhu 我是不一样的JackieZhu

\*/

object CreateDataFrameWithStructTypeScala {

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

//1:创建一个SparkSession对象

val spark = SparkSession

.builder()

.appName("Spark SQL basic example")

.master("local")

.getOrCreate()

//2:从SparkSession中获取SparkContext对象

val sc: SparkContext = spark.sparkContext

sc.setLogLevel("WARN")

//3:读取文件,创建一个RDD

val peopleRDD: RDD[String] = sc.textFile("f:\\data\\people.txt")

//4:指定schema

val schemaString = "id username age"

//5:根据schemaString创建结构化信息

val fields = schemaString.split(" ")

.map(fieldName => StructField(fieldName, StringType, nullable = true))

//6:将fields转换成schema

val schema: StructType = StructType(fields)

val rowRDD = peopleRDD

.map(\_.split(" "))

.map(attributes => Row(attributes(0), attributes(1),attributes(2)))

//7:创建DataFrame

val frame: DataFrame = spark.createDataFrame(rowRDD,schema)

//8:将DataFrame注册成一个临时表

frame.createOrReplaceTempView("people")

spark.sql("select \* from people").show()

}

}

```

## ## 5.4 编写Spark SQL程序操作HiveContext

HiveContext是对应spark-hive这个项目,与hive有部分耦合,

支持hql,是SqlContext的子类，在Spark2.0之后，HiveContext和SqlContext在SparkSession进行了统一，可以通过操作SparkSession来操作HiveContext和SqlContext。

### ### 5.4.1 添加pom依赖

```xml

<!--添加spark hive的支持-->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-hive\_2.11</artifactId>

<version>2.2.0</version>

</dependency>

```

### ### 5.4.2 代码实现

#### #### 5.4.2.1 Java版本

```java

/\*\*

\* @author JackieZhu

\* @date 2018-08-23 21:15:35

\* @description 我是传智JackieZhu, 我是不一样的JackieZhu

\*/

public class HiveSupportWithJava {

public static void main(String[] args) {

//创建一个SparkSession对象,开启对hive的支持

SparkSession spark = SparkSession

.builder()

.appName("Java Spark Hive Example")

.master("local")

.config("spark.sql.warehouse.dir","f:\\spark-warehouse") //指定spark仓库地址

.enableHiveSupport() //开启对hive的支持

.getOrCreate();

//操作sql语句

spark.sql("CREATE TABLE IF NOT EXISTS person(id int,name string,age int) row format delimited fields terminated by ' '");

//加载数据

spark.sql("LOAD DATA LOCAL INPATH './data/student.txt' INTO TABLE person");

//查询结果

spark.sql("select \* from person").show();

//释放资源

spark.stop();

}

}

```

#### #### 5.4.2.2 Scala版本

```scala

object HiveSupportWithScala {

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

//创建一个SparkSession对象

val spark:SparkSession = SparkSession

.builder()

.appName("Spark Hive Example")

.master("local")

//指定Spark的仓库地址

.config("spark.sql.warehouse.dir", "f:\\data\\spark-warehouse")

//开启对hive的支持

.enableHiveSupport()

.getOrCreate()

//设置日志级别

spark.sparkContext.setLogLevel("WARN")

//操作sql语句

spark.sql("CREATE TABLE IF NOT EXISTS person (id int, name string, age int) row format delimited fields terminated by ' '")

spark.sql("LOAD DATA LOCAL INPATH './data/student.txt' INTO TABLE person")

//查询

spark.sql("select \* from person order by age desc").show()

//释放资源

spark.stop()

}

}

```

# # 第六章 数据源

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。

## ## 6.1 使用Spark SQL从MySQL中加载数据

### ### 6.1.1 编写程序

#### #### 6.1.1.1 Java版本

```java

public class DataFromMySQLWithJava {

public static void main(String[] args) {

//1:创建一个SparkSession对象

SparkSession spark = SparkSession

.builder()

.appName("DataFromMySQLWithJava")

.master("local")

.getOrCreate();

//2:加载数据

/\*Properties connectionProperties = new Properties();

connectionProperties.put("user", "hive");

connectionProperties.put("password", "hive");

Dataset<Row> iplocation = spark.read().jdbc("jdbc:mysql://spark-node01.itheima.com:3306/test", "iplocation", connectionProperties);

\*/

Dataset<Row> jdbcDF = spark.read()

.format("jdbc")

.option("url", "jdbc:mysql://spark-node01.itheima.com:3306/test")

.option("dbtable", "iplocation")

.option("user", "hive")

.option("password", "hive")

.load();

//将数据打印到控制台

jdbcDF.show();

//释放资源

spark.stop();

}

}

```

#### #### 6.1.1.2 Scala版本

```scala

object DataFromMySQLWithScala {

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

//1:创建一个SparkSession对象

val spark = SparkSession

.builder()

.appName("Spark SQL basic example")

.master("local")

.getOrCreate()

//2:从SparkSession中获取SparkContext对象

val sc: SparkContext = spark.sparkContext

sc.setLogLevel("WARN")

/\*val connectionProperties = new Properties()

connectionProperties.put("user", "hive")

connectionProperties.put("password", "hive")

val jdbcDF2 = spark.read

.jdbc("jdbc:mysql://spark-node01.itheima.com:3306/test", "iplocation", connectionProperties)\*/

val jdbcDF = spark.read

.format("jdbc")

.option("url", "jdbc:mysql://spark-node01.itheima.com:3306/test")

.option("dbtable", "iplocation")

.option("user", "hive")

.option("password", "hive")

.load()

//打印结果

jdbcDF.show()

spark.stop()

}

}

```

## ## 6.2 使用spark-shell从mysql中加载数据

### ### 6.2.1 执行spark-shell命令

```she&#39;l&#39;l

bin/spark-shell --master spark://spark-node01.itheima.com:7077 --executor-memory 1g --total-executor-cores 2 --jars /opt/modules/hive-0.13.1-bin/lib/mysql-connector-java-5.1.17.jar --driver-class-path /opt/modules/hive-0.13.1-bin/lib/mysql-connector-java-5.1.17.jar

```

### ### 6.2.2 从mysql中加载数据

```scala

val jdbcDF = spark.read

.format("jdbc")

.option("url", "jdbc:mysql://spark-node01.itheima.com:3306/test")

.option("dbtable", "iplocation")

.option("user", "hive")

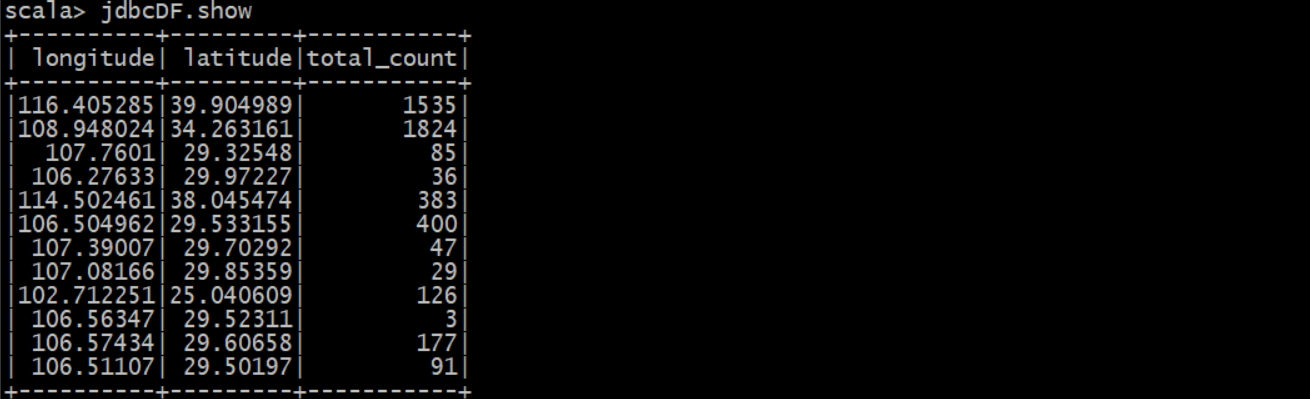
.option("password", "hive")

.load()

```

### ### 6.2.3 查看结果

![](img\查看结果.png)



## ## 6.3 Spark SQL将数据写入到MySQL中

### ### 6.3.1 代码实现

#### #### 6.3.1.1 Java版本

```java

public class Data2MysqlWithJava {

public static void main(String[] args) {

//1:创建一个SparkSession对象

SparkSession spark = SparkSession

.builder()

.appName("DataFromMySQLWithJava")

.master("local")

.getOrCreate();

//2:读取数据

RDD<String> lines = spark.sparkContext().textFile("f:\\data\\people.txt", 1);

//3:将RDD转换成JavaRDD

JavaRDD<String> linesJavaRDD = lines.toJavaRDD();

//4:切分每一行数据

JavaRDD<String[]> attributes = linesJavaRDD.map(new Function<String, String[]>() {

@Override

public String[] call(String v1) throws Exception {

return v1.split(" ");

}

});

//5:与样例类关联

JavaRDD<P> peopleRDD = attributes.map(new Function<String[], P>() {

@Override

public P call(String[] v1) throws Exception {

return new P(Integer.parseInt(v1[0]), v1[1], Integer.parseInt(v1[2]));

}

});

//6:将RDD转换成DataFrame

Dataset<Row> dataFrame = spark.createDataFrame(peopleRDD, P.class);

//7:将DataFramme注册成表

dataFrame.createOrReplaceTempView("people");

Dataset<Row> result = spark.sql("select \* from people order by age desc");

//8:将结果保存到mysql中

Properties connectionProperties = new Properties();

connectionProperties.put("user","hive");

connectionProperties.put("password","hive");

result.write().jdbc("jdbc:mysql://spark-node01.itheima.com:3306/test","people",connectionProperties);

result.write()

.format("jdbc")

.option("url", "jdbc:mysql://spark-node01.itheima.com:3306/test")

.option("dbtable", "people1")

.option("user", "hive")

.option("password", "hive")

.save();

}

}

```

#### #### 6.3.1.2 Scala版本

```scala

object Data2MysqlWithScala {

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

//1:创建一个SparkSession对象

val spark = SparkSession

.builder()

.appName("Spark SQL basic example")

.master("local")

.getOrCreate()

//2:从SparkSession中获取SparkContext对象

val sc: SparkContext = spark.sparkContext

sc.setLogLevel("WARN")

//3:读取文件

val lines: RDD[String] = sc.textFile("f:\\data\\people.txt")

//4:将每一行数据切分

val arrRDD: RDD[Array[String]] = lines.map(line=>line.split(" "))

//5:将RDD与Student进行关联

val studentRDD: RDD[Student] = arrRDD.map(x=>Student(Integer.parseInt(x(0)),x(1),Integer.parseInt(x(2))))

//导入隐式转换

import spark.implicits.\_

//6:将RDD转换成DataFrame

val studentDF: DataFrame = studentRDD.toDF()

//7:将DataFrame注册成一张表

studentDF.createOrReplaceTempView("student")

//8:操作sql

val resultDF: DataFrame = spark.sql("select \* from student order by age desc")

// val prop =new Properties()

// prop.setProperty("user","hive")

// prop.setProperty("password","hive")

// resultDF.write.jdbc("jdbc:mysql://spark-node01.itheima.com:3306/test","student",prop)

resultDF.write

.format("jdbc")

.option("url", "jdbc:mysql://spark-node01.itheima.com:3306/test")

.option("dbtable", "student1")

.option("user", "hive")

.option("password", "hive")

.save()

//释放资源

spark.stop()

}

}

case class Student(id:Int,username:String,age:Int)

```