# Spark SQL and DataFrame

# 课程目标

## 掌握Spark SQL的原理

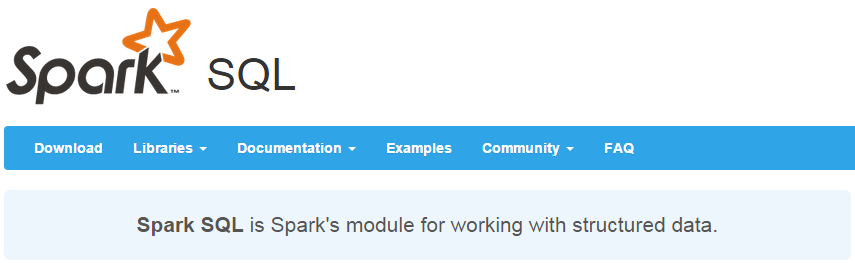
## 掌握DataFrame数据结构和使用方式

## 熟练使用Spark SQL完成计算任务

# Spark SQL

## Spark SQL概述

### 什么是Spark SQL



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。

### 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。所有Spark SQL的应运而生，它是将Spark SQL转换成RDD，然后提交到集群执行，执行效率非常快！

1.易整合



1. 统一的数据访问方式



1. 兼容Hive



1. 标准的数据连接



## DataFrames

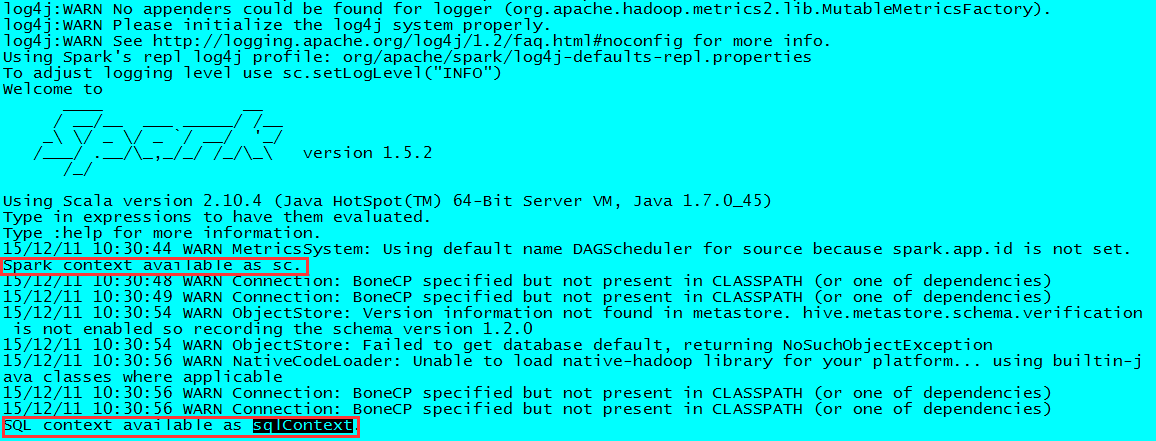
### 什么是DataFrames

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。



### 创建DataFrames

 在Spark SQL中SQLContext是创建DataFrames和执行SQL的入口，在spark-1.5.2中已经内置了一个**sqlContext**

****

1. 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

hdfs dfs -put person.txt /

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://node1.itcast.cn:9000/person.txt").map(\_.split(" "))

3.定义case class（相当于表的schema）

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

1. 将RDD和case class关联

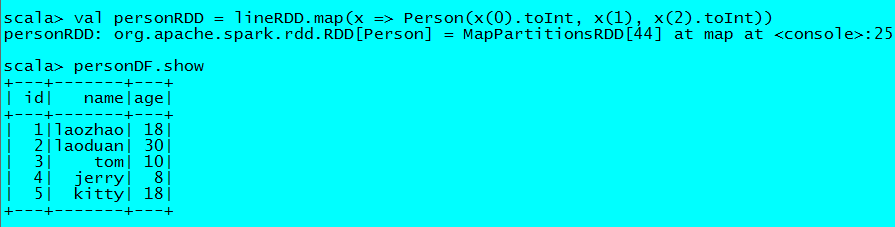
val personRDD = lineRDD.map(x => Person(x(0).toInt, x(1), x(2).toInt))

1. 将RDD转换成DataFrame

val personDF = personRDD.toDF

6.对DataFrame进行处理

personDF.show



## DataFrame常用操作

### DSL风格语法

//查看DataFrame中的内容

personDF.show

//查看DataFrame部分列中的内容

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

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

personDF.select("name").show

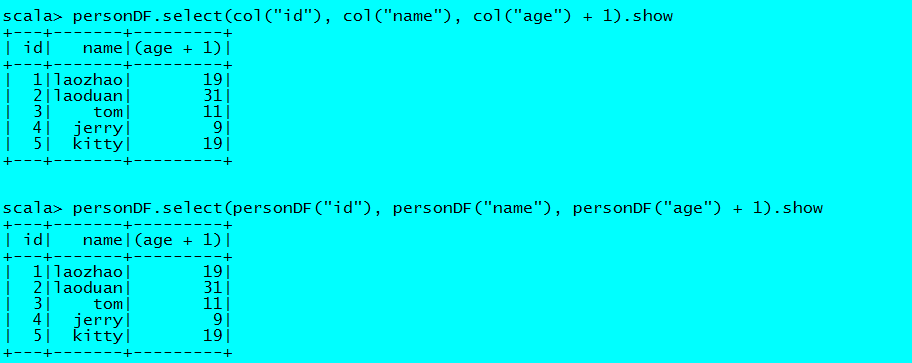
//打印DataFrame的Schema信息

personDF.printSchema

//查询所有的name和age，并将age+1

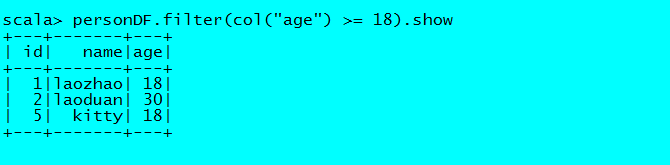
personDF.select(col("id"), col("name"), col("age") + 1).show

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



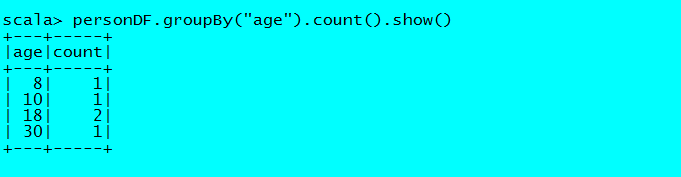
//过滤age大于等于18的

personDF.filter(col("age") >= 18).show



//按年龄进行分组并统计相同年龄的人数

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



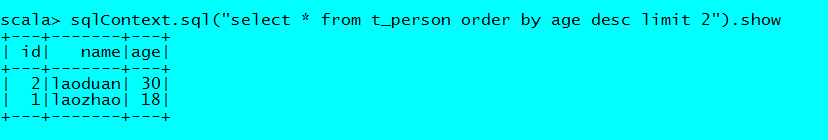
### SQL风格语法

如果想使用SQL风格的语法，需要将DataFrame注册成表

personDF.registerTempTable("t\_person")

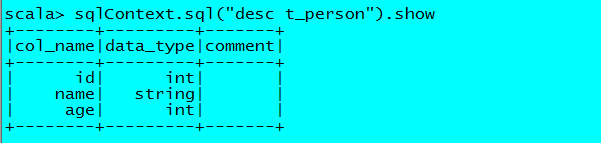
//查询年龄最大的前两名

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



//显示表的Schema信息

sqlContext.sql("desc t\_person").show



# 以编程方式执行Spark SQL查询

## 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

|  |
| --- |
| <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-sql\_2.10</**artifactId**>  <**version**>1.5.2</**version**> </**dependency**> |

### 通过反射推断Schema

创建一个object为cn.itcast.spark.sql.InferringSchema

|  |
| --- |
| **package** cn.itcast.spark.sql  **import** org.apache.spark.{SparkConf, SparkContext} **import** org.apache.spark.sql.SQLContext **object** InferringSchema {  **def** main(args: Array[String]) {   *//创建SparkConf()并设置App名称* **val** conf = **new** SparkConf().setAppName(**"SQL-1"**)  *//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** personRDD = lineRDD.map(x => *Person*(x(0).toInt, x(1), x(2).toInt))  *//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** personDF = personRDD.toDF  *//注册表* personDF.registerTempTable(**"t\_person"**)  *//传入SQL* **val** df = sqlContext.sql(**"select \* from t\_person order by age desc limit 2"**)  *//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()  } } *//case class一定要放到外面* **case class** Person(id: Int, name: String, age: Int) |

将程序打成jar包，上传到spark集群，提交Spark任务

/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-submit \

--class cn.itcast.spark.sql.InferringSchema \

--master spark://node1.itcast.cn:7077 \

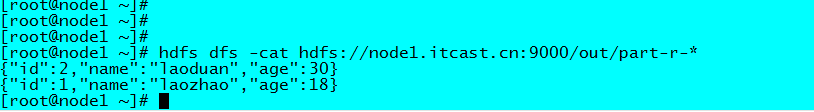
/root/spark-mvn-1.0-SNAPSHOT.jar \

hdfs://node1.itcast.cn:9000/person.txt \

hdfs://node1.itcast.cn:9000/out

查看运行结果

hdfs dfs -cat hdfs://node1.itcast.cn:9000/out/part-r-\*



### 通过StructType直接指定Schema

创建一个object为cn.itcast.spark.sql.SpecifyingSchema

|  |
| --- |
| **package** cn.itcast.spark.sql  **import** org.apache.spark.sql.{Row, SQLContext} **import** org.apache.spark.sql.types.\_ **import** org.apache.spark.{SparkContext, SparkConf}  */\*\*  \* Created by ZX on 2015/12/11.  \*/* **object** SpecifyingSchema {  **def** main(args: Array[String]) {  *//创建SparkConf()并设置App名称* **val** conf = **new** SparkConf().setAppName(**"SQL-2"**)  *//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*(**"age"**, IntegerType, **true**)  )  )  *//将RDD映射到rowRDD* **val** rowRDD = personRDD.map(p => *Row*(p(0).toInt, p(1).trim, p(2).toInt))  *//将schema信息应用到rowRDD上* **val** personDataFrame = sqlContext.createDataFrame(rowRDD, schema)  *//注册表* personDataFrame.registerTempTable(**"t\_person"**)  *//执行SQL* **val** df = sqlContext.sql(**"select \* from t\_person order by age desc limit 4"**)  *//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()  } } |

将程序打成jar包，上传到spark集群，提交Spark任务

/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-submit \

--class cn.itcast.spark.sql.InferringSchema \

--master spark://node1.itcast.cn:7077 \

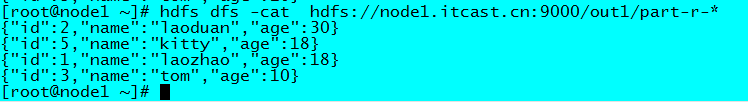
/root/spark-mvn-1.0-SNAPSHOT.jar \

hdfs://node1.itcast.cn:9000/person.txt \

hdfs://node1.itcast.cn:9000/out1

查看结果

hdfs dfs -cat hdfs://node1.itcast.cn:9000/out1/part-r-\*



# 数据源

## 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。

### 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

/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-shell \

--master spark://node1.itcast.cn:7077 \

--jars /usr/local/spark-1.5.2-bin-hadoop2.6/mysql-connector-java-5.1.35-bin.jar \

--driver-class-path /usr/local/spark-1.5.2-bin-hadoop2.6/mysql-connector-java-5.1.35-bin.jar

1. 从mysql中加载数据

val jdbcDF = sqlContext.read.format("jdbc").options(Map("url" -> "jdbc:mysql://192.168.10.1:3306/bigdata", "driver" -> "com.mysql.jdbc.Driver", "dbtable" -> "person", "user" -> "root", "password" -> "123456")).load()

1. 执行查询

jdbcDF.show()



### 将数据写入到MySQL中（打jar包方式）

1. 编写Spark SQL程序

|  |
| --- |
| **package** cn.itcast.spark.sql  **import** java.util.Properties **import** org.apache.spark.sql.{SQLContext, Row} **import** org.apache.spark.sql.types.{StringType, IntegerType, StructField, StructType} **import** org.apache.spark.{SparkConf, SparkContext}  **object** JdbcRDD {  **def** main(args: Array[String]) {  **val** conf = **new** SparkConf().setAppName(**"MySQL-Demo"**)  **val** sc = **new** SparkContext(conf)  **val** sqlContext = **new** SQLContext(sc)  *//通过并行化创建RDD* **val** personRDD = sc.parallelize(*Array*(**"1 tom 5"**, **"2 jerry 3"**, **"3 kitty 6"**)).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*(**"age"**, IntegerType, **true**)  )  )  *//将RDD映射到rowRDD* **val** rowRDD = personRDD.map(p => *Row*(p(0).toInt, p(1).trim, p(2).toInt))  *//将schema信息应用到rowRDD上* **val** personDataFrame = 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"**, **"123456"**)  *//将数据追加到数据库* personDataFrame.write.mode(**"append"**).jdbc(**"jdbc:mysql://192.168.10.1:3306/bigdata"**, **"bigdata.person"**, prop)  *//停止SparkContext* sc.stop()  } } |

1. 用maven将程序打包
2. 将Jar包提交到spark集群

/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-submit \

--class cn.itcast.spark.sql.JdbcRDD \

--master spark://node1.itcast.cn:7077 \

--jars /usr/local/spark-1.5.2-bin-hadoop2.6/mysql-connector-java-5.1.35-bin.jar \

--driver-class-path /usr/local/spark-1.5.2-bin-hadoop2.6/mysql-connector-java-5.1.35-bin.jar \

/root/spark-mvn-1.0-SNAPSHOT.jar