# Spark SQL and DataFrame

# 1. 课程目标

## 1.1. 掌握 Spark SQL 的原理

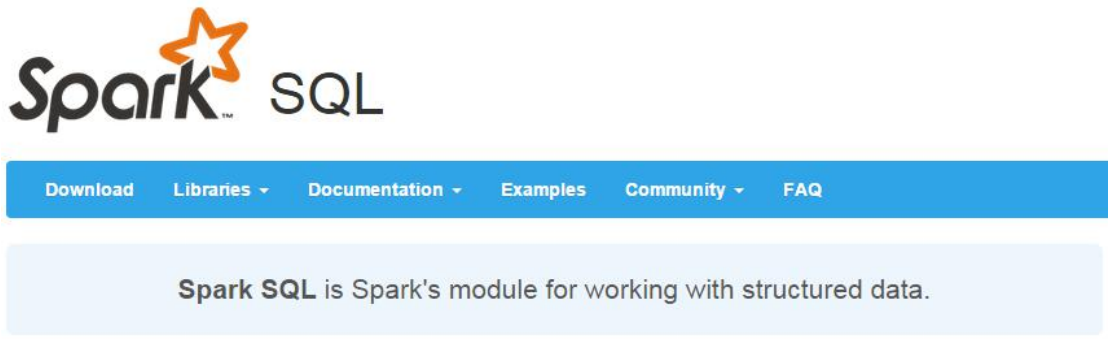
## 1.2. 掌握 DataFrame 数据结构和使用方式

## 1.3. 熟练使用 Spark SQL 完成计算任务

# 2. Spark SQL

## 2.1. Spark SQL 概述

### 2.1.1. 什么是 Spark SQL

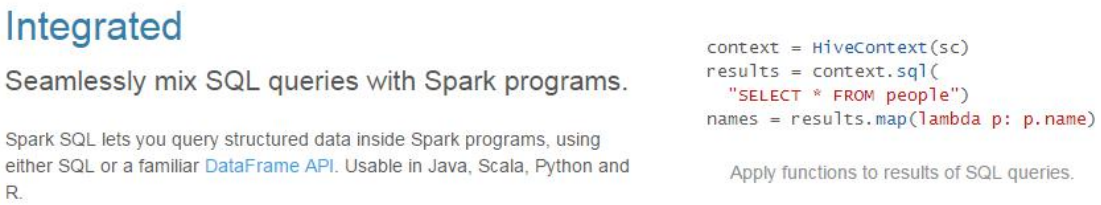



Spark SQL 是 Spark 用来处理结构化数据的一个模块，它提供了一个编程抽象叫做 DataFrame 并且作为分布式 SQL 查询引擎的作用。

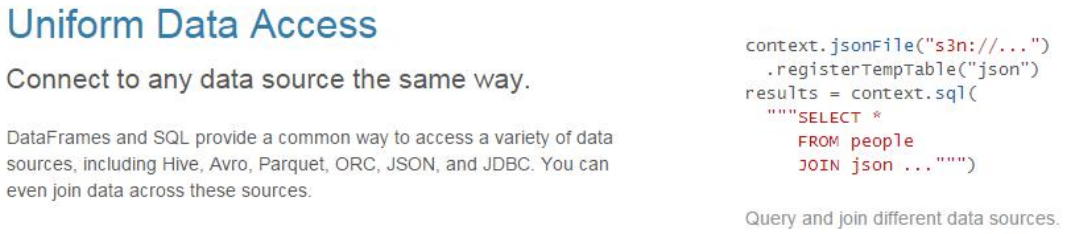
### 2.1.2. 为什么要学习 Spark SQL

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！

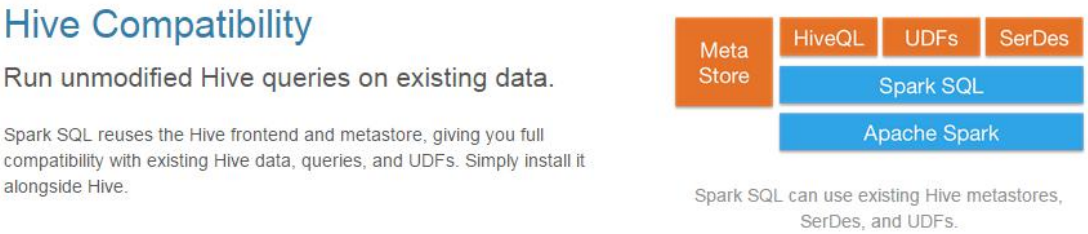
1.易整合



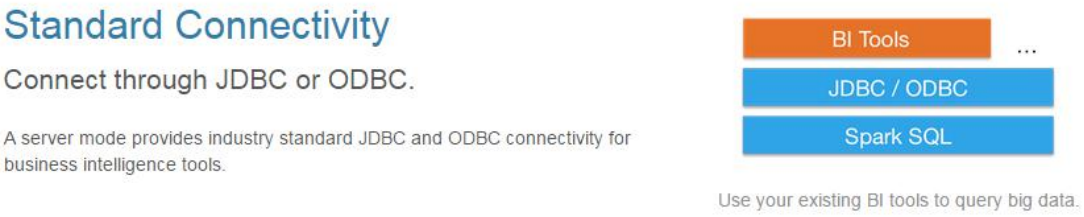

2.统一的数据访问方式




3.兼容 Hive



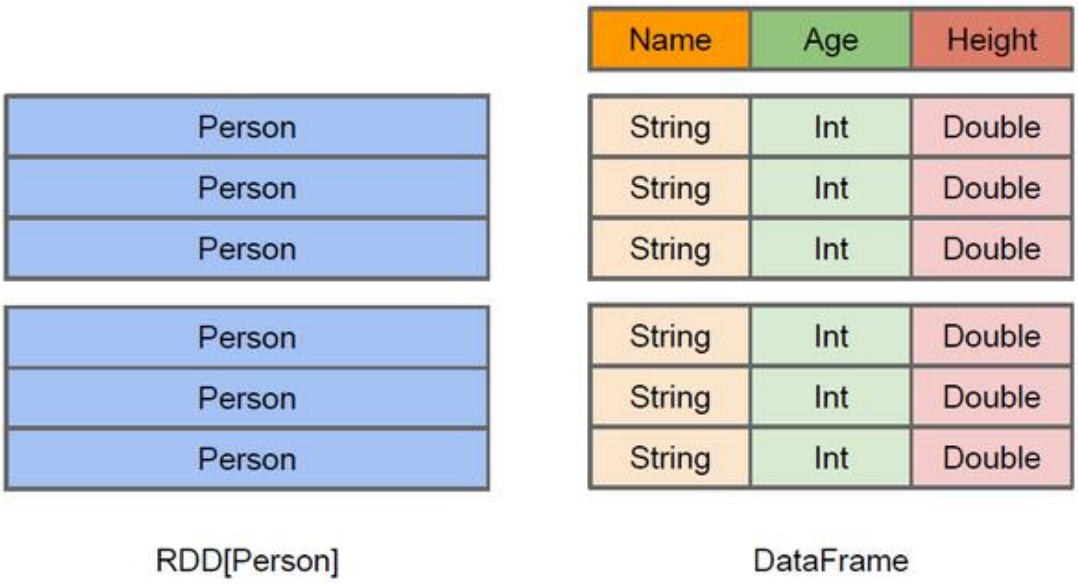

4.标准的数据连接




# 2.2. DataFrames

## 2.2.1. 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。




## 2.2.2. 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**

```
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 1.5.2
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_45)
Type in expressions to have them evaluated.
Type :help for more information.
15/12/11 10:30:44 WARN MetricsSystem: Using default name DAGScheduler for source because spark.app.id is not set.
Spark context available as sc.
15/12/11 10:30:48 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
15/12/11 10:30:49 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
15/12/11 10:30:54 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification
 is not enabled so recording the schema version 1.2.0
15/12/11 10:30:54 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
15/12/11 10:30:56 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-j
ava classes where applicable
15/12/11 10:30:56 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
15/12/11 10:30:56 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
SQL context available as 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)
```

4.将 RDD 和 case class 关联

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

5.将 RDD 转换成 DataFrame

*val personDF = personRDD.toDF*

6.对 DataFrame 进行处理

*personDF.show*

```
scala> val personRDD = lineRDD.map(x => Person(x(0).toInt, x(1), x(2).toInt))
personRDD: org.apache.spark.rdd.RDD[Person] = MapPartitionsRDD[44] at map at <console>:25

scala> personDF.show
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  1|laozhao| 18|
|  2|laoduan| 30|
|  3|    tom| 10|
|  4|  jerry|  8|
|  5|  kitty| 18|
+---+-------+---+
```

# 2.3. DataFrame 常用操作

## 2.3.1. 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*

```
scala> personDF.select(col("id"), col("name"), col("age") + 1).show
+---+-------+---------+
| id|   name|(age + 1)|
+---+-------+---------+
|  1|laozhao|       19|
|  2|laoduan|       31|
|  3|    tom|       11|
|  4|  jerry|        9|
|  5|  kitty|       19|
+---+-------+---------+


scala> personDF.select(personDF("id"), personDF("name"), personDF("age") + 1).show
+---+-------+---------+
| id|   name|(age + 1)|
+---+-------+---------+
|  1|laozhao|       19|
|  2|laoduan|       31|
|  3|    tom|       11|
|  4|  jerry|        9|
|  5|  kitty|       19|
+---+-------+---------+
```

//过滤 age 大于等于 18 的

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

```
scala> personDF.filter(col("age") >= 18).show
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  1|laozhao| 18|
|  2|laoduan| 30|
|  5|  kitty| 18|
+---+-------+---+
```

//按年龄进行分组并统计相同年龄的人数

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

```
scala> personDF.groupBy("age").count().show()
+---+-----+
|age|count|
+---+-----+
|  8|    1|
| 10|    1|
| 18|    2|
| 30|    1|
+---+-----+
```

## 2.3.2. SQL 风格语法

如果想使用 SQL 风格的语法，需要将 DataFrame 注册成表

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

```
scala> sqlContext.sql("select * from t_person order by age desc limit 2").show
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  2|laoduan| 30|
|  1|laozhao| 18|
+---+-------+---+
```

//显示表的 Schema 信息

sqlContext.sql("desc t_person").show

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

# 3. 以编程方式执行 Spark SQL 查询

## 3.1. 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

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.10</artifactId>
    <version>1.5.2</version>
</dependency>
```

### 3.1.1. 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，如果不到人无法将 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-*
```

```
[root@node1 ~]#
[root@node1 ~]#
[root@node1 ~]#
[root@node1 ~]# hdfs dfs -cat hdfs://node1.itcast.cn:9000/out/part-r-*
{"id":2,"name":"laoduan","age":30}
{"id":1,"name":"laozhao","age":18}
[root@node1 ~]#
```

## 3.1.2. 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-*
```

```
[root@node1 ~]# hdfs dfs -cat  hdfs://node1.itcast.cn:9000/out1/part-r-*
{"id":2,"name":"laoduan","age":30}
{"id":5,"name":"kitty","age":18}
{"id":1,"name":"laozhao","age":18}
{"id":3,"name":"tom","age":10}
[root@node1 ~]#
```

# 4. 数据源

## 4.1. 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。

### 4.1.1. 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

```
/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
```

2.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()
```

3.执行查询

```
jdbcDF.show()
```

```
scala> jdbcDF.show()
+---+-----+---+
| id| name|age|
+---+-----+---+
|  3|kitty| 30|
|  2|jerry| 20|
+---+-----+---+
```

### 4.1.2. 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}
```

```scala
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()
  }
}
```

2.用 maven 将程序打包

3.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
```