# Spark SQL

## Spark SQL的前世今生

### SQL是什么

结构化查询语言(Structured Query Language)简称SQL(发音：/ˈes kjuː ˈel/ "S-Q-L")，是一种特殊目的的编程语言，是一种数据库查询和[程序设计语言](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80" \t "https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E6%9F%A5%E8%AF%A2%E8%AF%AD%E8%A8%80/_blank)，用于存取数据以及查询、更新和管理[关系数据库系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E6%9F%A5%E8%AF%A2%E8%AF%AD%E8%A8%80/_blank)；同时也是[数据库脚本文件](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%84%9A%E6%9C%AC%E6%96%87%E4%BB%B6" \t "https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E6%9F%A5%E8%AF%A2%E8%AF%AD%E8%A8%80/_blank)的扩展名。

结构化查询语言是高级的非过程化编程语言，允许用户在高层[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84" \t "https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E6%9F%A5%E8%AF%A2%E8%AF%AD%E8%A8%80/_blank)上工作。它不要求用户指定对数据的存放方法，也不需要用户了解具体的数据存放方式，所以具有完全不同底层结构的不同[数据库系统](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E6%9F%A5%E8%AF%A2%E8%AF%AD%E8%A8%80/_blank), 可以使用相同的结构化查询语言作为数据输入与管理的接口。结构化查询语言语句可以嵌套，这使它具有极大的灵活性和强大的功能。

### **为什么需要SQL**

（1）SQL简单，学习成本低

（2）以前的数据工作者习惯了SQL

### **常用的SQL框架**

* Hive
* Impala
* Presto
* Shark
* Drill
* Phoenix

### **Spark SQL 概述**

Spark SQL使得运行SQL查询十分简单。Spark SQL 能够轻易地定位相应的表和元数据。Spark SQL 为Spark提供了查询结构化数据的能力，查询时既可以使用SQL也可以使用人们熟知的DataFrame API（RDD）。Spark SQL支持多语言编程包括Java、Scala、Python及R，开发人员可以根据自身喜好进行选择。

## **Spark SQL编程**

### Spark SQL编程入口

**SQL Context**

**Scala语言创建SQLContext实例代码如下：**

val sc:SparkContext//已经存在的SparkContext

val sqlContext = new org.apache.spark.sql.SQL(sc)

### **DataFrame**

DataFrame是一个由以命名列组成的分布式数据集，本身就是一个带schema的RDD。它在概念上等价于关系数据库中的表或R/Python中的data frame。从另一个角度看，DataFrame是一个新的RDD,它可以通过许多方式创建，如结构化的数据文件、Hive中的表、外部数据库或已有的RDD。DataFrame也提供了Scala、Java、Python及R语言的API接口。

### **DataFrame编程**

1. 基于json文件创建DataFrame

示例代码：

|  |
| --- |
| **import** org.apache.spark.sql.SQLContext **import** org.apache.spark.{SparkConf, SparkContext}  **object** SparkSql\_DataFrameApp {  **def** main(args: Array[String]): Unit = {  **val** conf = **new** SparkConf().setAppName(**"SparkSql\_DlataFrameApp"**).setMaster(**"local"**)  **val** sc = **new** SparkContext(conf)  **val** sqlc = **new** SQLContext(sc)  *////////////////////////////////////////////////////////////  /\* people.json 内容如下：  ｛"name":"Michael"｝  {"name":"Andy","age":30}  {"name":"Justin","age":19}  \*/* **val** df = sqlc.read.json(**"D:\\甲骨文资料\\Spark师资培训\\data\\people.json.txt"**)  df.select(**"name"**,**"age"**).show()  sc.stop  } } |

1. DataFram操作

DataFrame支持Scala、Java和Python的操作接口。下面是以Scala语言为例进行几个操作示例：

* DataFram常用操作

|  |
| --- |
| **import** org.apache.spark.sql.SQLContext **import** org.apache.spark.{SparkConf, SparkContext}  **object** SparkSql\_DataFrameApp {  **def** main(args: Array[String]): Unit = {  **val** conf = **new** SparkConf().setAppName(**"SparkSql\_DlataFrameApp"**).setMaster(**"local"**)  **val** sc = **new** SparkContext(conf)  **val** sqlc = **new** SQLContext(sc)  *////////////////////////////////////////////////////////////  /\* people.json 内容如下：  ｛"name":"Michael"｝  {"name":"Andy","age":30}  {"name":"Justin","age":19}  \*/* **val** df = sqlc.read.json(**"D:\\甲骨文资料\\Spark师资培训\\data\\people.json.txt"**)  df.show()  *///////////////////////////////////////////////////////////  //使用printSchema方法输出DataFrame的Schema信息* df.printSchema()  *//////////////////////////////////////////////////////////  //使用select方法来选择所需要的字段* df.select(**"name"**).show()  *//////////////////////////////////////////////////////////  //使用select 方法选择所需要的字段，并为age字段加1* df.select(df(**"name"**),df(**"age"**)+1).show()  */////////////////////////////////////////////////////////  //使用filter方法完成条件过滤* df.filter(df(**"age"**)>20).show()  */////////////////////////////////////////////////////////  //使用groupBy方法进行分组，求分组后的总数* df.groupBy(**"name"**).count().show()  sc.stop  } } |

* DataFrame SQL操作

Spark可以使用SQLContext的sql方法执行SQL查询操作，SQL方法返回的查询结果为DataFrame格式。代码如下：

|  |
| --- |
| **import** org.apache.spark.sql.SQLContext **import** org.apache.spark.{SparkConf, SparkContext}  **object** SparkSql\_DataFrameApp {  **def** main(args: Array[String]): Unit = {  **val** conf = **new** SparkConf().setAppName(**"SparkSql\_DlataFrameApp"**).setMaster(**"local"**)  **val** sc = **new** SparkContext(conf)  **val** sqlc = **new** SQLContext(sc)  *////////////////////////////////////////////////////////////  /\* people.json 内容如下：  ｛"name":"Michael"｝  {"name":"Andy","age":30}  {"name":"Justin","age":19}  \*/* **val** df = sqlc.read.json(**"D:\\甲骨文资料\\Spark师资培训\\data\\people.json.txt"**)  df.registerTempTable(**"people"**)  sqlc.sql(**"select name,age from people where age>21"**).show()  sc.stop  } } |

1. RDD与DataFrame互操作

在Spark SQL中有两种方式可以在DataFram和RDD进行转换，第一种是利用反射机制，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。

1. 使用反射获取RDD内的Schema

当已知类的Schema，使用这种基于反射的方法会让代码更加简洁而且效果更好。Scala支持使用case class类型导入RDD并转换为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。case class可以嵌套组合成Sequences或Array。这种RDD可以高效的转换为DataFrame并注册为表。

**样例数据：**

Michael,29

Andy,30

Justin,19

Tom,50

**样例代码:**

|  |
| --- |
| **import** org.apache.spark.sql.SQLContext **import** org.apache.spark.{SparkConf, SparkContext}   **object** SparkSql\_DFToRdd {  **case class** PEOPLE(name:String,age:Int)   **def** main(args: Array[String]): Unit = {  *method1* }  **def** method1():Unit={  **val** conf = **new** SparkConf().setAppName(**"SparkSql\_DFToRdd"**).setMaster(**"local[2]"**)  **val** sc = **new** SparkContext(conf)  **val** sqlc = **new** SQLContext(sc)  **import** sqlc.implicits.\_  *//将RDD转成DataFramval* **val** people = sc.textFile(**"D:\\甲骨文资料\\Spark师资培训\\data\\people.txt"**).map(x=>x.split(**","**)).map(x=>*PEOPLE*(x(0),x(1).trim.toInt)).toDF()  *//使用DataFrame API访问  // people.show()* people.registerTempTable(**"people"**)  **val** teenagers = sqlc.sql(**"select \* from people where age>20"**)  teenagers.show()   *//DataFrame 转换成RDD进行操作,根据索引号进行取值* teenagers.map(x=>**"Name: "**+x(0)).collect().foreach(*println*)  *//DataFrame 转换成RDD进行操作，根据字段名进行取值* teenagers.map(x=>**"Name: "**+x.getAs(**"name"**)).collect().foreach(*println*)  *//DataFrame转成RDD进行操作，一次返回多列的值* teenagers.map(\_.getValuesMap(*List*(**"name"**,**"age"**))).collect().foreach(*println*)     }  } |

1. 通过编程接口指定Schema

通过Spark SQL的接口创建RDD的Schema，这种方式会让代码比较繁琐。这种方法的好处是，在运行时才知道数据的列以及列的类型的情况下，可以动态生成Schema。可以通过以下三步创建DataFrame：第一步将RDD转为包含row对象的RDD;

第二步：基于structType类型创建Schema,与第一步创建的RDD相匹配;

第三步：通过SQLContext的createDataFrame方法对第一步的RDD应用Schema。

功能实现，样例代码：

|  |
| --- |
| **import** org.apache.spark.sql.SQLContext **import** org.apache.spark.{SparkConf, SparkContext} **import** org.apache.spark.sql.Row **import** org.apache.spark.sql.types.{StringType, StructField, StructType}  **object** SparkSql\_DfToRdd2 {  **def** main(args: Array[String]): Unit = {  **val** conf = **new** SparkConf().setAppName(**"SparkSql\_DfToRdd2"**).setMaster(**"local[2]"**)  **val** sc = **new** SparkContext(conf)  **val** sqls = **new** SQLContext(sc)  **import** sqls.implicits.\_  **val** people = sc.textFile(**"D:\\甲骨文资料\\Spark师资培训\\data\\people.txt"**)  *//定义Schema 信息* **val** schemaString = **"name,age"** *//根据自定义的字符串schema信息产生DataFrame的Schema* **val** schema = StructType(  schemaString.split(**","**).map(fieldName=>*StructField*(fieldName,StringType,**true**))  )  *//将RDD转换成Row* **val** rowRDD = people.map(\_.split(**","**)).map(x=>*Row*(x(0),x(1).trim))   *//将Schema作用到RDD上* **val** peopleDataFrame = sqls.createDataFrame(rowRDD,schema)   *//将DataFrame注册成临时表* peopleDataFrame.registerTempTable(**"people"**)  **val** results = sqls.sql(**"select \* from people where age>30"**)   results.map(x=>**"Name: "**+x(0)).collect().foreach(*println*)  sc.stop  }  } |