Day35 & Day36_Spark SQL

大数据-张军锋 Day35 Day36 Spark SQL

Spark SQL

官方文档：http://spark.apache.org/docs/latest/sql-programming-guide.html
它是对spark的扩展和优化，让我们增添了一种新的使用spark进行数据处理的方式

1. sql处理
2. linq api（Dataset）处理

Shark(hive on spark)
虽然利用了spark进行执行，但是对于hive的依赖性比较大
查看hive中sql的执行计划： `Explain select * from tableName`

解释器—>优化器—>执行器—>spark rdd执行完成对数据处理
兼容hive 的metastore、serder、udf等

scala项目的maven创建

new—>maven project–>不使用模板—>在pom中添加：
依赖：

```
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>2.11.8</version>
</dependency>
```

插件：

```
<build>
    <plugins>
        <plugin>
            <!-- see http://davidb.github.com/scala-maven-plugin --
>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.1.3</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>testCompile</goal>
                    </goals>
                    <configuration>
                        <args>
                            <arg>-dependencyfile</arg>
                            <arg>${project.build.directory}/.scal
a_dependencies</arg>
                        </args>
                    </configuration>
                </execution>
            </executions>
        </plugin>

        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.13</version>
            <configuration>
                <useFile>false</useFile>
                <disableXmlReport>true</disableXmlReport>
                <!-- If you have classpath issue like NoDefClassErr
or,... -->
                <!-- useManifestOnlyJar>false</useManifestOnlyJar -
->
                <includes>
                    <include>**/*Test.*</include>
                    <include>**/*Suite.*</include>
                </includes>
            </configuration>
        </plugin>
    </plugins>
</build>
```

在src下新建一个scala目录

projcet structure—>modules—>选中scala目录—>点击Sources按钮，把scala目录设置成源文件目录

spark编程模型

1.SparkConf–>SparkContext–>RDD–>RDD调用transformation和action
2.SparkSession–>DataSet/DataFrame –>直接调用sql后者调用DS/DF的 transformat和action linq模式取数据方法
3.SparkConf–>StreamingContext–>DStream –>对DStream调用transformation和action 以及窗口window计算函数

基本程序

```
package com.bd14.zjf

import org.apache.spark.sql.SparkSession

object WordCount {

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

    //构建SparkSession对象
    val spark = SparkSession.builder().master("local[*]").appNam
e("SparkSQL WordCount").getOrCreate()
    //引入隐式转换
    import spark.implicits._
    import spark.sql

    //构建rdd
    val rdd = spark.sparkContext.textFile("/user_info.txt")
    val ds = rdd.toDS()

    //打印当前ds的模式
    ds.printSchema()
    //对ds中的数据进行sql的形式来处理
    //把ds创建一个视图名，这个视图名就可以直接被当做成表名来使用
    ds.createOrReplaceTempView("line_str")
    val wcResult = sql(
      """
        |select word,
        |       count(1) as count
        |from(
        |     select explode(split(value,'\\s')) as word
        |     from line_str
        |)
        |group by word
      """.stripMargin
    )
    wcResult.show()
  }

}
```

Sparksql的文件格式

- SparkSql DateSet保存成text文件的时候DataSet中只能有1个字段，读取进来的时候也是只能读取一个
- 保存的时候不管ds字段叫什么名称，读取进来的时候名称都是value，都是字符串类型

csv

csv格式，只保留字段数量和字段之间的分割，字段名称没有保存，字段类型没有保存，文件是文本

```
val spark = SparkSession.builder().master("local[*]").appName("spar
k sql 存储文件").getOrCreate()

import spark.implicits._
import spark.sql

def saveCsv() = {
  val ds = spark.read.text("/orderdata/orders")
  ds.printSchema()
  val orderSplit = ds.map(x => {
    val infos = x.getString(0).split("\\|")
    (infos(0), infos(1), infos(2), infos(3), 1)
  })
  orderSplit.printSchema()
  orderSplit.createOrReplaceTempView("order")
  //获取订单状态为COMPLETE的订单信息
  val result = sql("select * from order where _4 = 'COMPLETE'")
  result.show()
  //根据核心数，默认启动2个，每个输出一个
  result.coalesce(1)
  result.write.csv("/bd14/spark/ordercsv")
}

def readCsv() = {
  val ds = spark.read.csv("/bd14/spark/ordercsv")
  ds.printSchema()
  //ds.show()
  ds.createOrReplaceTempView("orders")
  sql(
    """
      |select _c0,_c3
      |from orders
    """.stripMargin).show()
}
```

Parquet

parquet，保存字段数量和分割，保存字段名称，保存字段类型，文件是二进制文件

```
def writeParquer() = {
  val ds = spark.read.text("/orderdata/orders")
  ds.printSchema()
  val orderSplit = ds.map(x => {
    val infos = x.getString(0).split("\\|")
    (infos(0), infos(1), infos(2), infos(3), 1)
  })
  orderSplit.printSchema()
  orderSplit.write.parquet("/bd14/spark/orderparquer")
}

def readParquer() = {
  val ds = spark.read.parquet("/bd14/spark/orderparquer")
  ds.printSchema()
}
```

orc

orc，保存字段数量和分割，保存字段名称，保存字段类型，文件是二进制文件

```
def writeOrc() = {
  val ds = spark.read.text("/orderdata/orders")
  ds.printSchema()
  val orderSplit = ds.map(x => {
    val infos = x.getString(0).split("\\|")
    (infos(0), infos(1), infos(2), infos(3), 1)
  })
  orderSplit.printSchema()
  orderSplit.write.orc("/bd14/spark/orderorc")
}

def readOrc() = {
  val ds = spark.read.parquet("/bd14/spark/orderorc")
  ds.printSchema()
}
```

json

json,保存字段数量和分割，保存字段名称，保存字段类型，文件是文本文件(字段类型不丰富)

```scala
def writeJson() = {
  val ds = spark.read.text("/orderdata/orders")
  ds.printSchema()
  val orderSplit = ds.map(x => {
    val infos = x.getString(0).split("\\|")
    (infos(0), infos(1), infos(2), infos(3), 1)
  })
  orderSplit.printSchema()
  orderSplit.write.json("/bd14/spark/orderjson")
}

def readJson() = {
  val ds = spark.read.json("/bd14/spark/orderjson")
  ds.printSchema()
}
```

jdbc

jdbc，从关系型数据库读取或保存数据，会存储数据的字段名称，字段类型等信息完整保存

```scala
def readJdbc() = {
  val url = "jdbc:mysql://localhost:3306/bigdata14"
  val properties = new Properties()
  properties.put("user", "root")
  properties.put("password", "root")
  val ds = spark.read.jdbc(url, "hero", properties)
  ds.printSchema()
  ds.show()
}

def writeJdbc() = {
  val ds = spark.read.text("/orderdata/orders")
  val orderSplit = ds.map(x => {
    val infos = x.getString(0).split("\\|")
    (infos(0), infos(1), infos(2), infos(3), 1)
  })
  orderSplit.printSchema()
  orderSplit.createOrReplaceTempView("order")
  val complete = sql("select * from order where _4 = 'COMPLETE'")
  val url = "jdbc:mysql://localhost:3306/bigdata14"
  val properties = new Properties()
  properties.put("user", "root")
  properties.put("password", "root")
  complete.write.jdbc(url, "orderss", properties)
}
```

Rdd与ds、df之间的转换

- Rdd加载文件时，文件的每一行构成rdd的一个元素
- DataSet/DataFrame 加载文件时，文件的每一行构成DataSet/DataFrame的一条有模式的记录
- 使用sql的条件：模式 + 元组

Rdd–加模式—>DataSet/DataFrame

```scala
rdd.toDS()
rdd.toDF()
spark.createDataSet()
spark.createDataFrame()
```

DataSet/DataFrame –加模式—>RDD

```
ds.rdd
df.rdd
```

Dataset的保存文件的模式

- SaveMode:Overwrite: overwrite the existing data.
- SaveMode:Append: append the data.
- SaveMode:ignore: ignore the operation
- SaveMode:ErrorIfExists: default option,throw an exception at runtime

```
def saveMode() = {
  val ds = spark.read.text("/orderdata/orders")
  val orderSplit = ds.map(x => {
    val infos = x.getString(0).split("\\|")
    (infos(0), infos(1), infos(2), infos(3), 1)
  })
  orderSplit.createOrReplaceTempView("order")
  val complete = sql("select * from order where _4 = 'COMPLETE'")
  //mode(SaveMode.Append)枚举类型设置存储方式
  complete.write.mode(SaveMode.Overwrite).parquet("/bd14/spark/orde
rparquer")
}
```

使用模式的方法

使用sparksession的read方法加载数据文件时，结果中(DataSet/DataFrame)从文件中加载模式

自动加载模式

- 从文件中自动加载
- 从数据中自动加载

```
val spark = SparkSession.builder().master("local[*]").appName("add
Schema").getOrCreate()

import spark.implicits._
import spark.sql
import scala.collection.JavaConversions._

def addSchemaFromData() = {
  val list = List(1, 2, 3, 4, 5, 6)
  val rdd = spark.sparkContext.parallelize(list)
  val ds1 = rdd.toDS()
  ds1.printSchema()
  val ds2 = spark.createDataset(list)
  ds2.printSchema()
  val list2 = List((1, 2, 3), (4, 5, 6))
  val ds3 = spark.createDataset(list2)
  ds3.printSchema()
}
```

通过DataFrame添加模式

- 数据元素用Row类型来封装
- 模式用StructType来描述表结构，用StructField来描述字段（名称，类型，是否为空）
- 使用DataType的子类来描述字段的类型LongType，IntegerType，DoubleType，DecimalType，StringType，TimestampType，DateType

```
  //手动给元组加载模式
  def addSchemaByFrame() = {
    val list = List(1,2,3,4,5,6)
    val structType = StructType(StructField("column1",IntegerType,
true)::Nil)

    val ds1 = spark.createDataFrame(list.map(x => Row(x)), structTy
pe)
    ds1.printSchema()

    val students = List(Row("小张", "男", 18), Row("小李", "女", 16),
Row("小刘", "女", 17))
    val studentSchema = StructType(StructField("name", StringType,
true) ::
                                   StructField("gender", StringTyp
e, true) ::
                                   StructField("age", IntegerType,
true) :: Nil)

    // 方式二
    //val studentSchema = (new StructType())
    //  .add(StructField("name", StringType, true))
    //  .add("gender", StringType, true)
    //  .add(StructField("age", IntegerType, true))

    val ds2 = spark.createDataFrame(students, studentSchema)
    ds2.printSchema()
    ds2.createOrReplaceTempView("student")
    sql("select * from student where age >= 17").show()
  }
```

Case class添加模式

在数据自动加载模式的时候，如果每个元素的数据类型是一个case class，那么sparksql会自动把case class的字段模式添加到dataset数据集上，即每个case class的属性名称将会作为模式的字段名称，属性的类型将会作为模式的字段类型
Caseclass如果定义在Object里面，则相当于在object里面定义一个静态的内部类，类似于java中的public static class
如果定义在class里面，相当于定义一个内部类 public class

```
case class Student(name: String, gender: String, age: Int)

//case class的方式给数据添加模式
def caseClassSchema() = {
  val list = List(Student("小张", "男", 18), Student("小李", "女", 1
6), Student("小刘", "女", 17))
  val ds1 = spark.createDataset(list)
  ds1.printSchema()
  ds1.createOrReplaceTempView("student")
  sql("select * from student where age >= 17").show()
}
```

Dataset和DataFrame

dataFrame 是一种更加具体的dataset
Dataset中的每个元素可以是任意的类型
而dataFrame中的元素必须是Row的对象，Row在声明的时候初始化方法和Tuple一样，一个Row里面可以放多个数据，位置是有序的，多个数据之间是不同的数据类型，在取值的时候row和array、list类似，使用小括号以位置标识作为参数来取值

```
val row = Row(1,"a",true,3.14)
row (0)
row (1)
```

DataSet转换成DataFrame

DataSet 可以通过调用toDF方法转换成DataFrame
两个类型中都保存有模式

```
def caseClassSchemaRdd() = {
  val rdd = spark.sparkContext.textFile("/orderdata/orders")
  val ccRdd = rdd.map(x => {
    val info = x.split("\\|")
    Order(info(0), info(1), info(2), info(3))
  })
  val ds = ccRdd.toDS()
  ds.printSchema()
  ds.createOrReplaceTempView("order")
  //统计每个客户订单数
  sql(
    """
      |select customerId
      |       ,count(1)
      |from order
      |group by customerId
    """.stripMargin).show()
  //dateframe是每个元素都是row类型的dataset
  val df = ds.toDF()
  df.printSchema()
  df.take(10).foreach(println)
  ds.take(10).foreach(println)
}
```

获取ds、df的模式

要获取ds或df模式，使用schema方法，获取的对象是structty

```
val spark = SparkSession.builder().master("local[*]").appName("添加
模式").getOrCreate()

import spark.implicits._
import spark.sql
def dsSchemaType() = {
val list = List(1, 2, 3, 4, 5)
val structType = StructType(StructField("column1", IntegerType, tru
e) :: Nil)
import scala.collection.JavaConversions._
val ds1 = spark.createDataFrame(list.map(x =>Row(x)), structType)

val sc = ds1.schema.printTreeString()
  ds1.printSchema()
}
```

udf自定义函数

SparkSession.udf —> UDFRegistration

Spark.udf.register(“函数名称”,函数字面量)

在接下来的spark环境下就可以直接调用sql中 “函数名称”来使用函数了

```
val spark = SparkSession.builder().master("local[*]").appName("add
Schema").getOrCreate()

import spark.implicits._
import spark.sql

def udfTest() = {
  val rdd = spark.sparkContext.textFile("/orderdata/orders")
  val ds = rdd.map(x => {
    val infos = x.split("\\|")
    Order(infos(0), infos(1), infos(2), infos(3))
  }).toDS()

  //定义一个函数把2014-02-22 00:00:00 日期格式转换成2014-02-22格式
  spark.udf.register("my_dt_to_date", (dateTime: String) => {
    val sformat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
    val tformat = new SimpleDateFormat("yyyy-MM-dd")
    val date = sformat.parse(dateTime)
    tformat.format(date)
  })

  ds.createOrReplaceTempView("order")
  val useUdf = sql(
    """
      |select orderId,
      |       my_dt_to_date(orderDate) ndate,
      |       customerId,
      |       status
      |from order
    """.stripMargin)
  useUdf.printSchema()
  useUdf.show()
}
```

Spark读写hbase

读Hbase

- 使用sparkContext的newHadoopAPIDataSet，从Hbase中加载数据，生成RDD
- RDD[(ImmutableWritable,Result)]
- 调用RDD的map方法或flatMap方法。把每一行数据转换成class case类型
- 调用RDD的toDs()方法，获取DataSet

写Hbase

- 调用DataSet或者DataFrame的rdd方法将其转换成rdd
- 调用rdd的saveNewHadoopApiDataSet方法，将要保存的数据封装到一个Put对象中，然后在rdd的value上，保存到hbase中去

Spark sql操作Hive

构建sparksession的时候调用其builder的enableHiveSupport方法支持hive

1. 创建scala项目
2. 添加依赖

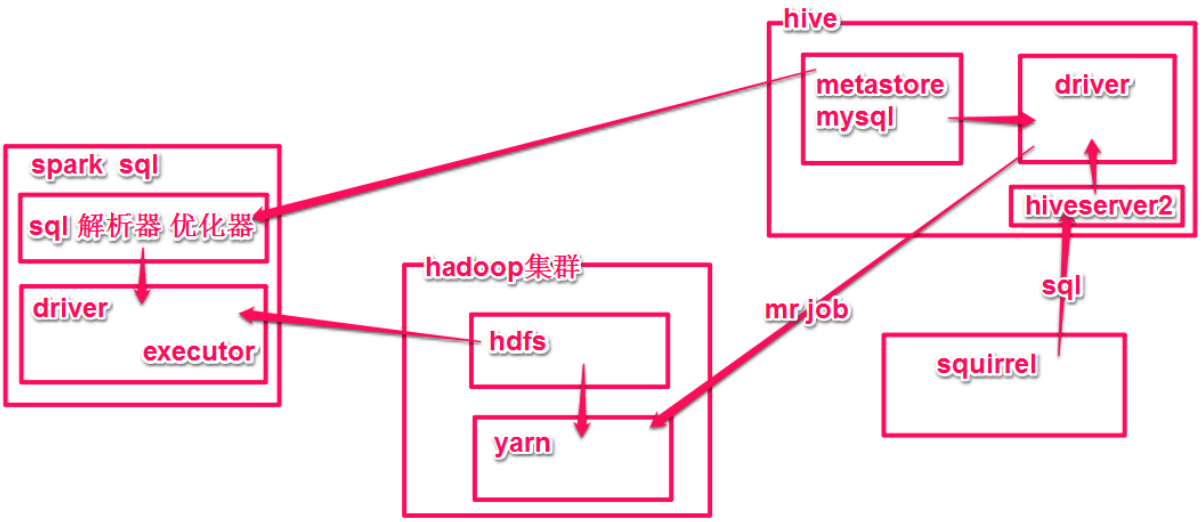
```
<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-core
-->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>2.2.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql
-->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>2.2.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java
-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.38</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-hive
-->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-hive_2.11</artifactId>
    <version>2.2.0</version>
</dependency>
```

3. 添加core-site.xml和hive.site.xml文件
   修改hive.site.xml

```
<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://master:3306/hive?createDatabaseIfNotExist=t
rue</value>
</property>
```

如果出现metastore 版本不一致，修改hive-site.xml文件

```
<property>
    <name>hive.metastore.schema.verification</name>
    <value>false</value>
</property>
```



spark的数仓文件目录位置
spark.sql.warehouse.dir

hive的数仓文件目录位置
hive.metastore.warehouse.dir – /user/hive/warehouse

sparksql 是支持hive 而不是依赖hive

- 如果不和hive集成，在insert数据到sparksql的数仓中的话需要指定spark.sql.warehouse.dir为某个hdfs上的路径
- 如果和hive集成使用，则spark.sql.warehouse.dir如果不填就会默认是hive.metastore.warehouse.dir配置的目录，保持文件存储位置就在hive的仓库目录下。

sparksql支持绝大部分hive中的sql语法，但不是全部
常用的只有hive的分桶sparksql不支持，其余的基本都支持