## 第九部分 SparkSql 编程

Spark SQL是用于处理结构化数据的Spark模块。与基本的Spark RDD API不同，Spark SQL提供的接口包含了有关数据结构和任务执行过程的更多信息。在Spark引擎内部，Spark SQL使用这些信息来执行额外的优化。有几种与Spark SQL交互的方法，包括SQL和Dataset API。在执行计算任务时，这几种交互方式使用相同的执行引擎。这种统一意味着开发人员可以轻松地在不同的API之间来回切换，提高了开发效率，又降低了使用成本。

### 1 任务概述

* 了解Spark-SQL开发流程
* 了解Spark DataFrame原理
* 掌握Spark SQL编程API

### 2 任务分析

要利用SparkSQL分析数据，需要掌握SparkSQLAPI以及核心对象DataFrame。SparkSQL是处理结构化数据，处理完毕后还需存入Mysql。

### 3 SparkSql概述

Spark SQL的前身是Shark，Shark在设计上完全兼容Hive。由于对Hive的过度依赖，因此Spark团队放弃了Shark，开发了更易用的Spark SQL。

Spark SQL拥有如下特点：

* 易整合：Spark SQL允许用户使用SQL或熟悉的DataFrame API在Spark程序中查询结构化数据。 支持使用Java，Scala，Python和R编程语言。
* 统一数据访问：DataFrames和SQL提供了访问各种数据源的常用方法，包括Hive，Avro，Parquet，ORC，JSON和JDBC。用户还可以在程序中交叉导入数据。
* 集成Hive：Spark SQL支持HiveQL语法以及Hive SerDes和UDF，允许用户访问现有的Hive仓库。
* 标准的数据连接：支持使用行业标准的JDBC和ODBC访问数据源。

Spark SQL提供了一个叫做DataFrame的数据类型，来表示结构化数据。从使用上看，DataFrame 的API提供了更加友好的API，使编程更为容易。另外DataFrame更像数据库中的表，具有行列信息。可以避免Spark引擎在执行的时候，过多的去推断数据类型，因此比直接使用普通的RDD效率上更高。与RDD相同的是，DataFrame也是“惰性的”，也分转换操作与行动操作，任务执行仍然需要生成DAG、划分阶段等。

### 4 使用SparkSql分析数据

#### **安装Mysql及配置**

步骤1：获取Mysql仓库源。

wget <http://dev.mysql.com/get/mysql57-community-release-el7-8.noarch.rpm>

步骤2：安装源。

yum localinstall mysql57-community-release-el7-8.noarch.rpm

步骤3：检查mysql源是否安装成功。

yum repolist enabled | grep "mysql.\*-community.\*"

显示Mysql信息表示安装成功。

步骤4：安装mysql。

yum install mysql-community-server

步骤5：启动MySQL服务。

systemctl start mysqld

步骤6：查找临时密码。

grep 'temporary password' /var/log/mysqld.log

步骤7：登录Mysql，输入临时密码。

mysql -u root -p

步骤8：修改Mysql密码并使之生效。

set password for 'root'@'localhost'=password('这里输入自己的密码');

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY '这里输入自己的密码';

#### **SparkSQL编程**

##### 数据源

创建一个json文件，录入以下内容：

{"product":"suit"}

{"product":"pants", "price":30}

{"product":"Shoes", "price":19}

##### 创建DataFrame

|  |
| --- |
| from pyspark.sql import SparkSession  spark = SparkSession.builder.getOrCreate()  loan\_info = spark.read.json("file:///hwadee/spark/testdata/04.json")  loan\_info.show() |

##### 选择列与列操作

|  |
| --- |
| loan\_info.select(loan\_info.product, loan\_info. price + 1).show() |

##### 数据过滤

|  |
| --- |
| loan\_info.filter(loan\_info. price > 25).show() |

##### 分组计算

|  |
| --- |
| loan\_info.groupBy("product").count().show() |

##### 排序

|  |
| --- |
| loan\_info.sort(loan\_info. price.desc()).show()  loan\_info.sort(loan\_info. price.desc(),loan\_info. product.desc()).show() |

##### 列重命名

|  |
| --- |
| loan\_info.select(loan\_info. product.alias("Name"), loan\_info. price).show() |

#### **RDD转换到DataFrame**

##### 数据源

pant1,12

Shoe1,13

pant2,14

suit3,16

suit1,17

suit2,120

##### 通过反射自动推断

|  |
| --- |
| from pyspark import SparkConf, SparkContext  from pyspark.sql import SparkSession  from pyspark.sql.types import Row  spark\_conf = SparkConf().setMaster("local").setAppName("RddTransDF")  sc = SparkContext(conf=spark\_conf)  spark = SparkSession.builder.getOrCreate()  loan\_info = sc.textFile("file:///hwadee/spark/testdata/05.json")  # 分割,获取键值  loan\_info\_list = loan\_info.map(lambda x: x.split(','))  def f(x):  data = {}  data["product"] = x[0]  data["price"] = x[1]  return data  # 调用外部函数创建模式  loan\_info\_df = loan\_info\_list.map(lambda x: Row(\*\*f(x))).toDF()  # 必须创建临时表，否则报错  loan\_info\_df.createOrReplaceTempView("ProInfo")  rows = spark.sql("select \*from ProInfo")  result = rows.rdd.map(lambda x: "product:" + x[0] + ",price:" + x[1])  def show\_data(x):  print(x)  result.foreach(lambda x: show\_data(x)) |

##### 构造schema应用到现有的RDD上

|  |
| --- |
| from pyspark import SparkConf, SparkContext  from pyspark.sql import SparkSession  from pyspark.sql.types import Row, StructField, StringType, StructType  spark\_conf = SparkConf().setMaster("local").setAppName("TestSchema")  sc = SparkContext(conf=spark\_conf)  spark = SparkSession.builder.getOrCreate()  loan\_info = sc.textFile("file:///hwadee/spark/testdata/05.json")  schema\_string = ["product", "price"]  """A field in :class:`StructType`.  :param name: string, name of the field.  :param dataType: :class:`DataType` of the field.  :param nullable: boolean, whether the field can be null (None) or not.  :param metadata: a dict from string to simple type that can be toInternald to JSON automatically  """  fields = list(map(lambda fieldName: StructField(fieldName, StringType(), nullable=True), schema\_string))  schema = StructType(fields)  # 将数据行分割  # 将分割后的数据构造成rdd  loan\_info\_row\_rdd = loan\_info.map(lambda line: line.split(',')).map(  lambda attributes: Row(attributes[0], attributes[1]))  # 将schema应用到rdd上  loan\_info\_df = spark.createDataFrame(loan\_info\_row\_rdd, schema)  # 创建临时表  loan\_info\_df.createOrReplaceTempView("LoanInfo")  data = spark.sql("SELECT \* FROM LoanInfo")  result = data.rdd.map(lambda x: "product: " + x[0] + "," + "price:" + x[1])  def show\_data(x):  print(x)  result.foreach(lambda x: show\_data(x)) |

#### **spark读写mysql**

##### 创建数据源

|  |
| --- |
| CREATE TABLE  IF NOT EXISTS `customerinfo` (  `num` VARCHAR (10),  `uname` VARCHAR (100),  `amount` VARCHAR (40),  PRIMARY KEY (`num`)  ) ENGINE = INNODB DEFAULT CHARSET = utf8;  insert into customerinfo values('NO1','老王','500000');  insert into customerinfo values('NO2','张三','600000');  insert into customerinfo values('NO3','老五','300000');  insert into customerinfo values('NO4','老大','400000'); |

##### 配置驱动

在配置hive的时候，已经下载了mysql-connector-java-5.1.40-bin.jar

拷贝到spark jars目录下：

cp /hwadee/mysql-connector-java-5.1.40-bin.jar ./

##### 查询数据

|  |
| --- |
| from pyspark.sql import SparkSession  spark = SparkSession.builder.getOrCreate()  mysql\_df = spark.read.format("jdbc").option("url", "jdbc:mysql://114.116.31.24:3306/hive").option("driver",  "com.mysql.jdbc.Driver").option(  "dbtable", "customerinfo").option("user", "root").option("password", "root").load()  mysql\_df.show() |

##### 插入数据

|  |
| --- |
| from pyspark.sql import SparkSession  from pyspark.sql.types import Row  from pyspark.sql.types import StructType  from pyspark.sql.types import StructField  from pyspark.sql.types import StringType  spark = SparkSession.builder.getOrCreate()  # 将数组初始化为rdd  customerinfo\_rdd = spark.sparkContext.parallelize(["NO5,黄飞鸿,25000"]).map(lambda line: line.split(","))  # 构建schema  schema = StructType([StructField("page\_type", StringType(), True),  StructField("count\_num", StringType(), True),  StructField("weights", StringType(), True)])  # 构建新的rdd  row\_rdd = customerinfo\_rdd.map(lambda x: Row(x[0].strip(), x[1].strip(), x[2].strip()))  # 应用schema  customerinfo\_df = spark.createDataFrame(row\_rdd, schema)  conn\_param = {}  conn\_param['user'] = 'root'  conn\_param['password'] = 'root'  conn\_param['driver'] = "com.mysql.jdbc.Driver"  # 'append' 指数据的插入模式  # mode: specifies the behavior of the save operation when data already exists.  #  # \* ``append``: Append contents of this :class:`DataFrame` to existing data.  # \* ``overwrite``: Overwrite existing data.  # \* ``ignore``: Silently ignore this operation if data already exists.  # \* ``error`` or ``errorifexists`` (default case): Throw an exception if data already \  # exists.  customerinfo\_df.write.jdbc("jdbc:mysql://114.116.31.24:3306/hive", 'customerinfo', 'append', conn\_param) |

### 5 拓展提高

在本章的示例中，创建Spark连接对象使用的是SparkSession。在Spark2.x版本之后，推荐使用SparkSession来连接Spark引擎。SparkSession对象已经封装了SparkContext、HiveContext、SQLContext、StreamingContext，使得针对不同场景的开发都可以统一使用一种编程方式。

同时，在示例程序中使用了createOrReplaceTempView方法。该方法是为了创建一个临时表。SparkSession还包含createOrReplaceGlobalTempView方法。createOrReplaceTempView创建的临时表只能在当前会话中使用，createOrReplaceGlobalTempView创建的临时表称为全局临时表，可以在不同会话中使用。

### 7 练习与实训

问题1：SparkSQL支持哪些数据源：\_\_\_\_\_\_\_\_\_\_\_

1. Mysql数据
2. Json数据
3. 文本数据
4. 视音频数据

问题2：以下说法正确的是：\_\_\_\_\_\_\_\_\_\_\_

1. DataFrame与RDD是一样的
2. DataFrame对象包含RDD对象
3. Spark SQL完全支持SQL标准语法
4. Spark SQL支持事务

问题3：如何在控制台输入DataFrame中的数据：\_\_\_\_\_\_\_\_\_\_\_

1. df.show()
2. df.show(false)
3. df.collect()
4. df.take(10)