> 想象可以使抽象具体化

SparkSQL之数据加载和保存

通用加载和保存方法

1.加载数据

（1）read直接加载数据

```
scala> spark.read.
csv  jdbc  json  orc  parquet textFile… …
```

注意：加载数据的相关参数需写到上述方法中。如：textFile需传入加载数据的路径，jdbc需传入JDBC相关参数。

（2）format指定加载数据类型

```
scala> spark.read.format("…")[.option("…")].load("…")
```

用法详解：

（1）format("..."）：指定加载的数据类型，包括"csv"、"jdbc"、"json"、"orc"、"parquet"和"textFile"。

（2）load("..."）：在"csv"、"orc"、"parquet"和"textFile"格式下需要传入加载数据的路径。

（3）option("..."）：在"jdbc"格式下需要传入JDBC相应参数，url、user、password和dbtable

2.保存数据

（1）write直接保存数据

```
scala>  df.write.
csv  jdbc  json  orc  parquet textFile
```

注意：保存数据的相关参数需写到上述方法中。如：textFile需传入加载数据的路径，jdbc需传入JDBC相关参数。

（2）format指定保存数据类型

```
scala>  df.write.format()[.option()].save()
```

用法详解：

（1）format()：指定保存的数据类型，包括"csv"、"jdbc"、"json"、"orc"、"parquet"和"textFile"。

（2）save ()：在"csv"、"orc"、"parquet"和"textFile"格式下需要传入保存数据的路径。

（3）option()：在"jdbc"格式下需要传入JDBC相应参数，url、user、password和dbtable

（3）文件保存选项

```
df.write.mode(SaveMode.Append).save()
```

可以采用SaveMode执行存储操作，SaveMode定义了对数据的处理模式。SaveMode是一个枚举类，其中的常量包括：

（1）Append：当保存路径或者表已存在时，追加内容；

（2）Overwrite：当保存路径或者表已存在时，覆写内容；

（3）ErrorIfExists：当保存路径或者表已存在时，报错；

（4）Ignore：当保存路径或者表已存在时，忽略当前的保存操作。

读取JSON

> SparkSQL能够**自动推测JSON数据集的结构**，并将它加载为一个Dataset[Row].可以通过SparkSession.read.json()去加载一个 一个JSON文件。

注意：这个JSON文件不是一个传统的JSON文件，每一行都得是一个JSON串。格式如下：

{"id":1，"name":"Tom"}

{"id":2，"name":"Jack"}

{"id":3，"name":"Marry"}

（1）导入隐式转换

```
import spark.implicits._
```

（2）加载JSON文件

```
val studentDF = spark.read.json("student.json")
```

（3）创建临时表

```
studentDF.createOrReplaceTempView("student")
```

（4）数据查询

```
val teenagerNamesDF = spark.sql("SELECT name FROM student
WHERE age BETWEEN 13 AND 19").show()
```

读取Parquent文件

> Spark SQL的**默认数据源**为Parquet格式。数据源为Parquet文件时，Spark SQL可以方便的执行所有的操作。修改配置项spark.sql.sources.default，可修改默认数据源格式。

（1）加载数据

```
val df = spark.read.load("/data/student.parquet")
```

（2）保存数据

```
df.select("id", " name").write.save("student.parquet")
```

读取Mysql数据

> 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。

1）启动spark-shell

```
bin/spark-shell --master spark://hadoop102:7077 [--jars mysql-connector-java-5.1.27-bin.jar]
```

2）定义JDBC相关参数配置信息

```
val connectionProperties = new Properties()
connectionProperties.put("user", "root")
connectionProperties.put("password", "123456")
```

3）使用read.jdbc加载数据

```
val jdbcDF2 = spark.read.jdbc("jdbc:mysql://hadoop102:3306/rdd", "rddtable", connectionProperties)
```

4）使用format形式加载数据

```
val jdbcDF = spark.read.format("jdbc").option("url", "jdbc:mysql://hadoop102:3306/rdd").option("dbtable", " rddtable").option("user", "root").option("password", "000000").load()
```

5）使用write.jdbc保存数据\

```
jdbcDF2.write.jdbc("jdbc:mysql://hadoop102:3306/mysql", "db", connectionProperties)
```

6）使用format形式保存数据'

```
jdbcDF.write
.format("jdbc")
.option("url", "jdbc:mysql://hadoop102:3306/rdd")
.option("dbtable", "rddtable3")
.option("user", "root")
.option("password", "123456")
.save()
```

读取Hive数据

> ApacheHive是Hadoop上的SQL引擎，SparkSQL编译时可以包含Hive支持，也可以不包含。包含Hive支持的SparkSQL可以支持Hive表访问、UDF(用户自定义函数)以及Hive查询语言(HiveQL/HQL)等。需要强调的一点是，如果要在SparkSQL中包含Hive的库，**并不需要事先安装Hive**。一般来说，最好还是在编译SparkSQL时引入Hive支持，这样就可以使用这些特性了。如果你下载的是二进制版本的Spark，它应该已经在编译时添加了Hive支持。

> 若要把SparkSQL连接到一个部署好的Hive上，你必须**把hive-site。xml复制到Spark的配置文件目录中($SPARK_HOME/conf)**。即使没有部署好Hive，SparkSQL也可以运行。需要注意的是，如果你没有部署好Hive，SparkSQL会在当前的工作目录中创建出自己的Hive元数据仓库，叫作**metastore_db**。此外，如果你尝试使用HiveQL中的CREATETABLE(并非CREATEEXTERNALTABLE)语句来创建表，这些表会被放在你默认的文件系统中的/user/hive/warehouse目录中(如果你的classpath中有配好的**hdfs-site.xml**，默认的文件系统就是HDFS，否则就是本地文件系统)。

1.使用内嵌的Hive

> 如果使用Spark内部的Hive则什么不用做直接使用spark.sql()即可。Hive的元数据存储在derby中，仓库地址:$SPARK_HOME/spark-warehouse

```
scala> val sql = spark.sql("show tables")
sql: org.apache.spark.sql.DataFrame = [database: string, tableName: string ... 1 more field]

scala> sql.show
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
+--------+---------+-----------+
```

从上图我们知道调用spark.sql()去执行sql语句，可以返回一个DataFrame，这样就可以通过使用DataFrame中的API对数据进行处理。现来来创建一张表：

```
scala> spark.sql("create table person(name string)")
19/11/07 09:25:52 WARN metastore.HiveMetaStore: Location: file:/root/spark-warehouse/person s
rson
res20: org.apache.spark.sql.DataFrame = []

scala> spark.sql("show tables").show
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
| default|   person|      false|
+--------+---------+-----------+
```

往表中添加数据：

```
scala> spark.sql("load data local inpath '/opt/data/person.txt' into table person")
res2: org.apache.spark.sql.DataFrame = []
```

```
scala> spark.sql("select * from person")
res3: org.apache.spark.sql.DataFrame = [name: string]

scala> spark.sql("select * from person").show
+-----+
| name|
+-----+
| Jack|
|Marry|
|  Tom|
+-----+
```

但是在实际使用中，几乎没有任何人会使用内置的Hive

2.使用外置的Hive

（1）Spark要接管Hive需要把hive-site.xml复制到conf/目录下。

（2）把Mysql的驱动copy到jars/目录下。

（3）如果访问不到hdfs，则需要把core-site.xml和hdfs-site.xml拷贝到conf/目录下。

打开spark-shell，查看student表的数据

```
scala> spark.sql("show tables").show
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
| default|  student|      false|
+--------+---------+-----------+


scala> spark.sql("select * from student").show
+----+---------+
|  id|     name|
+----+---------+
|1001|zhangshan|
|1002|    lishi|
|1003|  zhaoliu|
|1001|zhangshan|
|1002|    lishi|
|1003|  zhaoliu|
+----+---------+
```