第5章Spark SQL

(源自:https://biglab.site)

(版本:Ver1.7-20230828)

学习目标

- 了解Spark SQL基本概念
- 掌握Spark SQL与Shell交互
- 掌握创建DataFrame对象的方法
- 掌握DataFrame查看数据的方法
- 掌握DataFrame的查询及输出操作

任务背景

基于大数据技术对房价进行分析和预测，是科学精准调控，促进房地产市场平稳健康发展，实现“房住不炒”的重要手段。现有一份房屋销售数据文件house.csv，记录了某地区的房屋销售情况，包含销售价格、销售日期、房屋评分等共14个数据字段，字段说明如下。（1英尺=0.3048米）

字段名称	说明
selling_price	销售价格（单位：美元）
bedrooms_num	卧室数
bathroom_num	浴室数
housing_area	房屋面积（单位：平方英尺）
parking_area	停车区面积（单位：平方英尺）
floor_num	楼层数
sales_data	销售日期
built_area	建筑面积（单位：平方英尺）
basement_area	地下室面积（单位：平方英尺）
year_bulit	修建年份
year_repair	修复年份
latitude	纬度
longitude	经度
housing_rating	房屋评分

在进行房价数据分析之前，由于无法直接判断出各个数据字段之间的关系，因此需要先对数据进行基础的探索，探索各个数据字段间的关系并加以分析。

本章将使用Spark SQL即席查询框架解决房价数据探索分析的问题。

1. 首先介绍Spark SQL框架及其编程数据模型DataFrame的基本概念，并对Spark SQL相关环境进行配置。
2. 其次介绍DataFrame的查询、输出操作API的用法。
3. 最后结合房价数据探索分析实例，帮助读者掌握DataFrame的基础操作。

学习重点

理论学习

- （1）认识Spark SQL。
- （2）Spark SQL CLI配置。
- （3）Spark SQL与Shell交互。
- （4）DataFrame基础操作方法

实验学习

- （1）配置Spark SQL CLI。
- （2）掌握DataFrame基础操作。
- （3）探索分析房屋售价数据。
- （4）统计分析各季度房屋销量和销售额。
- （5）探索分析房屋评分。

数据准备

上传spark自带的example到hdfs

```
 hdfs dfs -mkdir -p /user/myname/sparkSql
 hdfs dfs -put /usr/local/spark/examples/src/main/resources/users.parquet /user/myname/sparkSql/
 hdfs dfs -put /usr/local/spark/examples/src/main/resources/people.json /user/myname/sparkSql/
 hdfs dfs -put /usr/local/spark/examples/src/main/resources/people.txt /user/myname/sparkSql/
```

下载测试数据文件spark_data.tar.gz（若前面已下载过，可跳过）

```
mkdir /root/spark/
cd /root/spark/
wget http://biglab.site/b59510spark/file/spark_data.tar.gz
tar -xvzf ./spark_data.tar.gz
hdfs dfs -mkdir /user/myname
```

上传测试数据到hdfs

```
cd /root/spark
hdfs dfs -put ./spark_data/movies.dat /user/myname/sparkSql/
hdfs dfs -put ./spark_data/users.dat /user/myname/sparkSql/
hdfs dfs -put ./spark_data/ratings.dat /user/myname/sparkSql/
```

上传实训数据到hdfs

```
wget -c https://biglab.site/b46488/file/train5data.tar.gz
tar -xzvf ./train5data.tar.gz
hdfs dfs -put -d ./train5data /user/myname/sparkSql/
hdfs dfs -ls /user/myname/sparkSql/train5data
```

环境准备

版本配套(了解)

Hadoop 3.3.1

```
[hadoop@master jars]$ hadoop version
 Hadoop 3.3.1
 Source code repository https://github.com/apache/hadoop.git -r
a3b9c37a397ad4188041dd80621bdeefc46885f2
 Compiled by ubuntu on 2021-06-15T05:13Z
 Compiled with protoc 3.7.1
 From source with checksum 88a4ddb2299aca054416d6b7f81ca55
 This command was run using /usr/local/hadoop-
3.3.1/share/hadoop/common/hadoop-common-3.3.1.jar
```

Spark 3.1.3

```
[hadoop@master ~]$ spark-shell
 22/08/16 14:30:44 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
 22/08/16 14:31:02 INFO SignalUtils: Registering signal handler for INT
 22/08/16 14:35:16 INFO SparkContext: Running Spark version 3.1.3
```

Scala 2.12.16

```
[hadoop@master bin]$ scala -version
 Scala code runner version 2.12.16 -- Copyright 2002-2022, LAMP/EPFL and
Lightbend, Inc.
```

Hive 3.1.3

```
[hadoop@master bin]$ scala -version
 Scala code runner version 2.12.16 -- Copyright 2002-2022, LAMP/EPFL and Lightbend, Inc.
 [hadoop@master bin]$ hive -version
 which: no hbase in (/usr/local/hive/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr/local/hadoop-3.3.1/sbin:/usr/local/hadoop-3.3.1/bin:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.el7_9.x86_64/jre/bin:/usr/local/scala/bin:/usr/local/spark/bin:/usr/local/spark/sbin:/home/hadoop/.local/bin:/home/hadoop/bin)
 Hive Session ID = 1551baf4-0917-496f-987f-57b7c21ae61a

 Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-3.1.3.jar!/hive-log4j2.properties Async: true
```

软件包准备

```
cd /usr/local/spark/jars/
 wget http://home.hddly.cn:90/soft/mysql/mysql-connector-java-5.1.45-bin.jar
 (或 wget http://bigdata.hddly.cn/b46488/file/chap6/mysql-connector-java-5.1.45-bin.jar)
 scp  /usr/local/spark/jars/mysql-connector-java-5.1.45-bin.jar slave1:/usr/local/spark/jars/
 scp  /usr/local/spark/jars/mysql-connector-java-5.1.45-bin.jar slave2:/usr/local/spark/jars/
```

(备注: 若有指定ssh端口号，可参考:
scp -P 8022 /usr/local/spark/jars/mysql-connector-java-5.1.45-bin.jar c24:/usr/local/spark/jars/)

任务5.1认识Spark SQL

任务描述

使用Spark SQL探索分析房价数据前，需要先了解Spark SQL是什么，有什么作用。

本节的任务如下。

1. 了解Spark SQL框架、Spark SQL的编程模型DataFrame和Spark SQL的运行过程。
2. 对Spark SQL相关环境进行配置。

Spark SQL简介

- 是一个用来处理结构化数据的Spark组件
- 是分布式的SQL查询引擎，并且提供一个DataFrame可编程抽象数据模型
- 前身是Shark,由于Shark需要依赖Hive,Spark SQL摆脱了对Hive依赖

- Spark SQL在数据兼容性好
  - 可直接处理RDD
  - 可处理Parquet文件或JSON文件
  - 可处理外部数据库数据，及Hive中存在的表
  - 可统一处理RDD和关系表
- Spark SQL最核心的

  编程抽象是DataFrame
  - DataFrame是一个分布式的Row对象的数据集合
  - 它本身实现了RDD绝大多数功能
- 基本处理流程
  - Spark SQL通常从外部数据源加载数据为DataFrame，
  - 然后通过DataFrame上丰富的API进行查询、转换，
  - 最后将结果进行展现或存储

Mysql元数据库安装和配置

删除已有mysql安装新mysql

```
rm -rf /usr/lib/mysql
rm -rf /usr/include/mysql
rm -rf /etc/my.cnf
rm -rf /var/lib/mysql
rm -rf /usr/share/mysql
cd /root
yum remove -y mariadb-libs
yum install -y net-tools
yum install -y perl
yum remove mysql mysql-server
yum install -y mariadb-libs
rpm -ivh mysql-community-* --force --nodeps
cp /root/mysql-connector-java-5.1.32-bin.jar  /usr/local/hive/lib/
```

启mysql服务

```
systemctl start mysqld
systemctl enable mysqld
service mysqld restart
```

查看临时密码

```
[root@master ~]# more /var/log/mysqld.log  |grep password |grep generated
2023-11-08T03:46:49.898572Z 1 [Note] A temporary password is generated for root@localhost: K6UdV+XTmoAB
[root@master ~]
```

使用临时密码登陆：

```
mysql -uroot -p'K6UdV+XTmoAB'
```

-p后的临时密码需要替换成本机查询出来的密码。

修改mysql的root密码

进入mysql命令后，修改mysql的root密码为:123456

```
alter user 'root'@'localhost' identified by '@Root_123456';
set global validate_password_policy=0;
set global validate_password_length=6;
set password for 'root'@'localhost'=password('123456');
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;
FLUSH PRIVILEGES;
CREATE SCHEMA `hive` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_bin ;
grant all on *.* to hive@'%' identified by 'hive';
grant all on *.* to hive@'localhost' identified by 'hive';
grant all on *.* to hive@'master' identified by 'hive';
grant all privileges on *.* to 'hive'@'%' identified by 'hive' with grant option;
grant all privileges on *.* to 'hive'@'master' identified by 'hive' with grant option;
flush privileges;
create database hive;
show databases;
exit;
```

Spark Hive 安装

在master主机上安装hive:

```
cd /root
wget https://repo.huaweicloud.com/apache/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz --no-check-certificate
tar -xzvf ./apache-hive-3.1.3-bin.tar.gz
mv ./apache-hive-3.1.3-bin /usr/local/hive
```

Spark SQL配置

软件包准备

下载mysql-connector-java-5.1.49-bin.jar到 /usr/local/spark/jars目录下

```
wget https://repo.huaweicloud.com/mysql/Downloads/Connector-J/mysql-connector-java-5.1.49.tar.gz
tar -xzvf ./mysql-connector-java-5.1.49.tar.gz
cp ./mysql-connector-java-5.1.49/mysql-connector-java-5.1.49-bin.jar /usr/local/spark/jars/
cd /usr/local/spark/jars/
ls mysql-connector-java *
```

如果ls有显示mysql-connector-java-5.1.49-bin.jar文件，说明mysql包已完成准备。

(还可以从mysql官网下载connector： https://downloads.mysql.com/archives/c-j/)

替换hive-site.xml

```
sed -i 's/home.hddly.cn:53306/master:3306/g' \
/usr/local/hive/conf/hive-site.xml
ln -s /usr/local/hive/conf/hive-site.xml \
/usr/local/spark/conf/hive-site.xml
```

核对hive-site.xml

```
vi /usr/local/hive/conf/hive-site.xml
```

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://master:3306/hive?useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8</value>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
<description>Driver class name for a JDBC metastore</description>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>hive</value>
<description>Username to use against metastore database</description>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>hive</value>
<description>password to use against metastore database</description>
</property>
<property>
<name>hive.metastore.event.db.notification.api.auth</name>
<value>false</value>
</property>
<property>
<name>hive.metastore.schema.verification</name>
<value>false</value>
</property>
```

初始化元数据库

```
cd /usr/local/hive/bin
./schematool -dbType mysql -initSchema
```

运行后会出现如下completed的内容，说明元数据库初始化完成

```
Initialization script completed
schemaTool completed
[root@master bin]#
```

#####

启动hive测试下

```
hive
```

配置spark

修改配置spark-env.sh,添加或修改行

```
export SPARK_CLASSPATH=/usr/local/spark/jars/mysql-connector-java-5.1.45-bin.jar
```

分发到从机

```
cd /usr/local/spark/conf
scp ./* slave1:/usr/local/spark/conf/
scp ./* slave2:/usr/local/spark/conf/
```

修改日志级别

```
echo '' > /usr/local/hive/conf/log4j.properties
echo 'log4j.rootLogger = ERROR,stdout' >> /usr/local/hive/conf/log4j.properties
echo 'log4j.appender.stdout=org.apache.log4j.ConsoleAppender' >> /usr/local/hive/conf/log4j.properties
echo 'log4j.appender.stdout.Target=System.out' >> /usr/local/hive/conf/log4j.properties
echo 'log4j.appender.stdout.layout=org.apache.log4j.PatternLayout' >> /usr/local/hive/conf/log4j.properties
echo 'log4j.appender.stdout.layout.ConversionPattern=%d{yyyy/MM/dd HH:mm:ss,SSS}- %c{1}: %m%n' >> /usr/local/hive/conf/log4j.properties
```

核对log4j.properties 修改后如：

```
log4j.rootLogger = ERROR,stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy/MM/dd HH:mm:ss,SSS}-
%c{1}: %m%n
```

分发日志配置

```
\cp /usr/local/hive/conf/log4j.properties
/usr/local/spark/conf/log4j.properties
```

启动Spark集群

```
/usr/local/spark/sbin/start-all.sh
```

启动Spark客户端

观察从机jps是否有worker进程，如没有则运行从机上的worker程序，如果有则不用运行:

```
/usr/local/spark/sbin/start-worker.sh master:7077
```

启动spark-sql

```
spark-sql
```

使用HQL创建表

```
show databases;
 create database myname;
 use myname;
 create table students(id int,name string,score double, classes string);
```

Spark SQL与Shell交互

特性

- Spark SQL已经集成在spark-shell中
- 在Spark-shell中执行SQL需使用SQLcontext对象来调用sql()方法
- Spark SQL 查询分成SQLContext和HiveContext
- SQLContext和HiveContext区别
    - HiveContext继承了SQLContext
        - 过时，已被SparkSession代替
        - HiveContext对象
            - val hiveContext=new org.apache.spark.sql.hive.HiveContext(sc)

    - SQLContext只支持SQL语法解析器
        - 过时，已被SparkSession代替
        - SQLContext对象
            - val sqlContext= new org.apache.spark.sql.SQLContext(sc)
    - HiveContext同时支持SQL语法解析器和HiveQL语法解析器
        - 过时，已被SparkSession代替
- SparkSession
    - spark2后SQLContext和HiveContext合并为SparkSession,启动spark-shell后为spark对象
    - spark-shell启动过程会初始化SparkSession对象为spark,它同时支持SQL语法解析器和HiveQL语法解析器

启动spark-shell

- spark-shell
- SparkSession对象
    - 进入spark-shell后可直接使用spark对象：
      Spark session available as 'spark'.

使用spark测试:

```
val dbs= spark.sql("show databases;")
dbs.collect;
```

运行结果:

```
scala> val dbs= spark.sql("show databases;");
dbs: org.apache.spark.sql.DataFrame = [namespace: string]

scala> dbs.collect;
res0: Array[org.apache.spark.sql.Row] = Array([default], [huangmm], [law],
[limm], [myname], [zhangmm])

scala>
```

任务5.2掌握DataFrame基础操作

任务描述

- DataFrame由SehemaRDD发展来
- Spark1.3.0开始，SchemaRDD更名为DataFrame
- 不是直接继承自RDD
- 它实现了RDD的绝大多数功能
- 可以理解为一个分布式Row对象的数据集合

创建DataFrame对象

通过结构化数据文件，Hive中的表，外部数据库，Spark计算过程生成RDD

结构化数据文件创建DataFrame

load()方法将HDFS上的格式化文件转换为DataFrame

load默认导入的文件格式是Parquet

Parquet文件

上传测试文件(如果前面数据准备已完成，这步可忽略)：

```
 hdfs dfs -mkdir -p /user/myname/sparkSql
 hdfs dfs -put /usr/local/spark/examples/src/main/resources/users.parquet /user/myname/sparkSql/
 hdfs dfs -put /usr/local/spark/examples/src/main/resources/people.json /user/myname/sparkSql/
 hdfs dfs -put /usr/local/spark/examples/src/main/resources/people.txt /user/myname/sparkSql/
```

进入spark-shell, 创建DataFrame

```
val dfUsers=spark.read.load("/user/myname/sparkSql/users.parquet")
dfUsers.collect;
```

运行结果如：

```

scala> val dfUsers=spark.read.load("/user/myname/sparkSql/users.parquet")
dfUsers: org.apache.spark.sql.DataFrame = [name: string, favorite_color: string ... 1 more field]

scala> dfUsers.collect;
res0: Array[org.apache.spark.sql.Row] = Array([Alyssa,null,WrappedArray(3, 9, 15, 20)], [Ben,red,WrappedArray()])

scala>
```

通过format加载json

文件并创建DataFrame

```
val dfPeople =spark.read.format("json").load("/user/myname/sparkSql/people.json")
dfPeople.collect;
```

运行结果如：

```
scala> val dfPeople
=spark.read.format("json").load("/user/myname/sparkSql/people.json")
dfPeople: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> dfPeople.collect;
res1: Array[org.apache.spark.sql.Row] = Array([null,Michael], [30,Andy],
[19,Justin])

scala>
```

通过json方法将Json

文件创建DataFrame

```
val dfPeople=spark.read.json("/user/myname/sparkSql/people.json")
dfPeople.collect;
```

运行结果如：

```

scala> val dfPeople=spark.read.json("/user/myname/sparkSql/people.json")
dfPeople: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> dfPeople.collect;
res2: Array[org.apache.spark.sql.Row] = Array([null,Michael], [30,Andy],
[19,Justin])
```

外部数据创建DataFrame

还可以从外部数据库(mysql,oracle)中创建DataFrame

需通过JDBC或ODBC连接的方式访问数据库

将mysql中的test中people表数据转换成DataFrame

```
val url="jdbc:mysql://home.hddly.cn:53306/test?useSSL=false"
 val jdbcDF=spark.read.format("jdbc").options(Map("url"->url,"user"-
>"test","password"->"test","dbtable"->"class")).load()
 jdbcDF.show()
```

运行结果如：

```
scala> val url="jdbc:mysql://home.hddly.cn:53306/test?useSSL=false"
url: String = jdbc:mysql://home.hddly.cn:53306/test?useSSL=false

scala>  val jdbcDF=spark.read.format("jdbc").options(Map("url"->url,"user"-
>"test","password"->"test","dbtable"->"class")).load()
jdbcDF: org.apache.spark.sql.DataFrame = [id: int, name: string ... 2 more
fields]

```

```
scala>  jdbcDF.show()
+---+-----+--------------------------------+------+

| id| name|                            text|  stud|
+---+-----+--------------------------------+------+
|  1|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
|  2|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
|  3|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
|  4|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
|  5|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
|  6|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
|  7|tipdm|大数据产品 - 泰迪科技-专注于大...|包思瑶|
|  8|tipdm|大数据产品 - 泰迪科技-专注于大...|薛伟杰|
|  9|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
| 10|tipdm|大数据产品 - 泰迪科技-专注于大...|汪睿鹏|
| 11|tipdm|大数据产品 - 泰迪科技-专注于大...|  林维|
| 12|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
| 13|tipdm|大数据产品 - 泰迪科技-专注于大...|  林维|
| 14|tipdm|大数据产品 - 泰迪科技-专注于大...|  林维|
| 15|tipdm|大数据产品 - 泰迪科技-专注于大...|汪睿鹏|
| 16|tipdm|大数据产品 - 泰迪科技-专注于大...|包思瑶|
| 17|tipdm|大数据产品 - 泰迪科技-专注于大...|汪睿鹏|
| 18|tipdm|大数据产品 - 泰迪科技-专注于大...|  李洪|
| 19|tipdm|大数据产品 - 泰迪科技-专注于大...|  张三|
| 20|tipdm|大数据产品 - 泰迪科技-专注于大...|陈建良|
+---+-----+--------------------------------+------+
only showing top 20 rows

```

RDD创建DataFrame

利用反射机制推断RDD

例子

```
case class Person(name:String,age:Int)
 val data=sc.textFile("/user/myname/sparkSql/people.txt").map(_.split(","))
 val people=data.map(p=>Person(p(0),p(1).trim.toInt)).toDF
 people.collect
 people.show()
```

运行结果如：

```
scala> case class Person(name:String,age:Int)
defined class Person

scala>  val
data=sc.textFile("/user/myname/sparkSql/people.txt").map(_.split(","))
data: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[24] at map
at <console>:24

scala>  val people=data.map(p=>Person(p(0),p(1).trim.toInt)).toDF
people: org.apache.spark.sql.DataFrame = [name: string, age: int]
```

```
scala>  people.collect
res4: Array[org.apache.spark.sql.Row] = Array([Michael,29], [Andy,30],
[Justin,19])

scala>  people.show()
+-------+---+
|   name|age|
+-------+---+
|Michael| 29|
|   Andy| 30|
| Justin| 19|
+-------+---+


scala>
```

采用编程方式指定Schema的方式将RDD转换为DataFrame

1,从原来的RDD创建一个元组或列表的RDD

2,用StructType创建一个和步骤1中创建的RDD
中元组或列表的结构相匹配的Schema

3,通过spark提供的createDataFrame方法
将Schema应用到RDD上

例如

```
val people=sc.textFile("/user/myname/sparkSql/people.txt")
 val schemaString ="name age"
 import org.apache.spark.sql.Row
 import org.apache.spark.sql.types.{StructType,StructField,StringType}
 val schema=StructType(schemaString.split("
").map(fieldName=>StructField(fieldName,StringType,true)))
 val rowRDD=people.map(_.split(",")).map(p=>Row(p(0),p(1).trim))
 val peopleDataFrame=spark.createDataFrame(rowRDD,schema)
 peopleDataFrame.show
```

运行结果：

```
scala> val people=sc.textFile("/user/myname/sparkSql/people.txt")
people: org.apache.spark.rdd.RDD[String] = /user/myname/sparkSql/people.txt
MapPartitionsRDD[1] at textFile at <console>:24

scala>  val schemaString ="name age"
schemaString: String = name age

scala>  import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala>  import org.apache.spark.sql.types.
{StructType,StructField,StringType}
import org.apache.spark.sql.types.{StructType, StructField, StringType}


```

```
scala>  val schema=StructType(schemaString.split(" ").map(fieldName=>StructField(fieldName,StringType,true)))
schema: org.apache.spark.sql.types.StructType = StructType(StructField(name,StringType,true), StructField(age,StringType,true))

scala>  val rowRDD=people.map(_.split(",")).map(p=>Row(p(0),p(1).trim))
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[3] at map at <console>:27

scala>  val peopleDataFrame=spark.createDataFrame(rowRDD,schema)
peopleDataFrame: org.apache.spark.sql.DataFrame = [name: string, age: string]

scala>  peopleDataFrame.show
+-------+---+

|   name|age|
+-------+---+
|Michael| 29|
|   Andy| 30|
| Justin| 19|
+-------+---+


```

Hive中的表创建DataFrame

查询Hive中的表,转换成DataFrame

在hive中准备表，在hive中执行：

```
show databases;
create database myname;
use myname;
create table students(id int,name string,score double, classes string);
insert into students(id,name,score,classes) values(1,'limm',90,'class1');
```

(参考常见问题1处理insert失败的问题)

在spark-shell中执行

```
spark.sql("use myname")
val people=spark.sql("select * from students")
people.show
```

运行结果：

```
scala> spark.sql("use myname")
res0: org.apache.spark.sql.DataFrame = []

scala> val people=spark.sql("select * from students")

```

```
people: org.apache.spark.sql.DataFrame = [id: int, name: string ... 2 more fields]

scala> people.show
+---+------+-----+-------+

| id|  name|score|classes|
+---+------+-----+-------+
| 10|limm10| 90.0| class1|
| 12|limm12| 90.0| class1|
+---+------+-----+-------+


scala>
```

查看DataFrame数据

查看及获取数据的常用函数或方法

方法	描述
printSchema	打印数据模式
show	查看数据
first/head/take/takeAsList	获取若干行数据
collect/collectAsList	获取所有数据

数据准备

上传测试文件(如果前面数据准备已完成，这步可忽略):

```
cd /root/spark
hdfs dfs -put ./spark_data/movies.dat /user/myname/sparkSql/
hdfs dfs -put ./spark_data/users.dat /user/myname/sparkSql/
hdfs dfs -put ./spark_data/ratings.dat /user/myname/sparkSql/
```

DataFrame常用操作函数或方法

创建DataFrame对象movies

```
case class Movie(moveieId:Int,titile:String,Genres:String)
val data=sc.textFile("/user/myname/sparkSql/movies.dat").map(_.split("::"))
val movies=data.map(m=>Movie(m(0).trim.toInt,m(1),m(2))).toDF()
```

运行结果如:

```
scala> case class Movie(moveieId:Int,titile:String,Genres:String)
defined class Movie

scala> val
data=sc.textFile("/user/myname/sparkSql/movies.dat").map(_.split("::"))
data: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[15] at map
at <console>:24

scala> val movies=data.map(m=>Movie(m(0).trim.toInt,m(1),m(2))).toDF()
movies: org.apache.spark.sql.DataFrame = [moveieId: int, titile: string ...
1 more field]

scala>
```

printSchema:打印数据模式

```
movies.printSchema
```

运行结果：

```
scala> movies.printSchema
root
 |-- moveieId: integer (nullable = false)
 |-- titile: string (nullable = true)
 |-- Genres: string (nullable = true)


scala>
```

show查看数据

方法	解释
show()	显示前20条记录
show(numRows:Int)	显示numRows条
show(truncate:Boolean)	是否最多只显示20个字符，默认为true
show(numRows:Int,truncate:Boolean)	显示numRows条记录并设置过长字符串的显示格式

```
movies.show()
movies.show(false)
```

运行结果：

```
scala> movies.show()
+--------+--------------------+--------------------+
|moveieId|              titile|              Genres|
+--------+--------------------+--------------------+
|       1|    Toy Story (1995)|Animation|Childre...|
```

```
|       2|      Jumanji (1995)|Adventure|Childre...|
|       3|Grumpier Old Men ...|      Comedy|Romance|
|       4|Waiting to Exhale...|        Comedy|Drama|
|       5|Father of the Bri...|              Comedy|
|       6|         Heat (1995)|Action|Crime|Thri...|
|       7|      Sabrina (1995)|      Comedy|Romance|
|       8| Tom and Huck (1995)|Adventure|Children's|
|       9| Sudden Death (1995)|              Action|
|      10|    GoldenEye (1995)|Action|Adventure|...|
|      11|American Presiden...|Comedy|Drama|Romance|
|      12|Dracula: Dead and...|       Comedy|Horror|
|      13|        Balto (1995)|Animation|Children's|
|      14|        Nixon (1995)|               Drama|
|      15|Cutthroat Island ...|Action|Adventure|...|
|      16|       Casino (1995)|      Drama|Thriller|
|      17|Sense and Sensibi...|       Drama|Romance|
|      18|   Four Rooms (1995)|            Thriller|
|      19|Ace Ventura: When...|              Comedy|
|      20|  Money Train (1995)|              Action|
+--------+--------------------+--------------------+
only showing top 20 rows


scala> movies.show(false)
+--------+-----------------------------------+-----------------------------
+
|moveieId|titile                             |Genres
|
+--------+-----------------------------------+-----------------------------
+
|1       |Toy Story (1995)                   |Animation|Children's|Comedy
|
|2       |Jumanji (1995)
|Adventure|Children's|Fantasy|
|3       |Grumpier Old Men (1995)            |Comedy|Romance
|
|4       |Waiting to Exhale (1995)           |Comedy|Drama
|
|5       |Father of the Bride Part II (1995) |Comedy
|
|6       |Heat (1995)                        |Action|Crime|Thriller
|
|7       |Sabrina (1995)                     |Comedy|Romance
|
|8       |Tom and Huck (1995)                |Adventure|Children's
|
|9       |Sudden Death (1995)                |Action
|
|10      |GoldenEye (1995)                   |Action|Adventure|Thriller
|
|11      |American President, The (1995)     |Comedy|Drama|Romance
|
|12      |Dracula: Dead and Loving It (1995) |Comedy|Horror
|
```

```
|13      |Balto (1995)                          |Animation|Children's
|
|14      |Nixon (1995)                          |Drama
|
|15      |Cutthroat Island (1995)               |Action|Adventure|Romance
|
|16      |Casino (1995)                         |Drama|Thriller
|
|17      |Sense and Sensibility (1995)          |Drama|Romance
|
|18      |Four Rooms (1995)                     |Thriller
|
|19      |Ace Ventura: When Nature Calls (1995)|Comedy
|
|20      |Money Train (1995)                    |Action
|
+--------+--------------------------------------+----------------------------
+
only showing top 20 rows


scala>
```

first/head/take/takeAsList获取若干行数据

```
movies.first()
movies.head(3)
movies.take(3)
movies.takeAsList(3)
```

运行结果：

```
scala> movies.first()
res6: org.apache.spark.sql.Row = [1,Toy Story
(1995),Animation|Children's|Comedy]

scala> movies.head(3)
res7: Array[org.apache.spark.sql.Row] = Array([1,Toy Story
(1995),Animation|Children's|Comedy], [2,Jumanji
(1995),Adventure|Children's|Fantasy], [3,Grumpier Old Men
(1995),Comedy|Romance])

scala> movies.take(3)
res8: Array[org.apache.spark.sql.Row] = Array([1,Toy Story
(1995),Animation|Children's|Comedy], [2,Jumanji
(1995),Adventure|Children's|Fantasy], [3,Grumpier Old Men
(1995),Comedy|Romance])

scala> movies.takeAsList(3)
res9: java.util.List[org.apache.spark.sql.Row] = [[1,Toy Story
(1995),Animation|Children's|Comedy], [2,Jumanji
(1995),Adventure|Children's|Fantasy], [3,Grumpier Old Men
(1995),Comedy|Romance]]

scala>
```

collect/collectAsList获取所有数据

```
movies.collect()
movies.collectAsList()
```

运行结果：

```
scala> movies.collect()
res10: Array[org.apache.spark.sql.Row] = Array([1,Toy Story
(1995),Animation|Children's|Comedy], [2,Jumanji
(1995),Adventure|Children's|Fantasy], [3,Grumpier Old Men
(1995),Comedy|Romance], [4,Waiting to Exhale (1995),Comedy|Drama], [5,Father
of the Bride Part II (1995),Comedy], [6,Heat (1995),Action|Crime|Thriller],
[7,Sabrina (1995),Comedy|Romance], [8,Tom and Huck
(1995),Adventure|Children's], [9,Sudden Death (1995),Action], [10,GoldenEye
(1995),Action|Adventure|Thriller], [11,American President, The
(1995),Comedy|Drama|Romance], [12,Dracula: Dead and Loving It
(1995),Comedy|Horror], [13,Balto (1995),Animation|Children's], [14,Nixon
(1995),Drama], [15,Cutthroat Island (1995),Action|Adventure|Romance],
[16,Casino (1995),Drama|Thriller], [17,Sense and Sensibil...

scala> movies.collectAsList()
res11: java.util.List[org.apache.spark.sql.Row] = [[1,Toy Story
(1995),Animation|Children's|Comedy], [2,Jumanji
(1995),Adventure|Children's|Fantasy], [3,Grumpier Old Men
(1995),Comedy|Romance], [4,Waiting to Exhale (1995),Comedy|Drama], [5,Father
of the Bride Part II (1995),Comedy], [6,Heat (1995),Action|Crime|Thriller],
[7,Sabrina (1995),Comedy|Romance], [8,Tom and Huck
(1995),Adventure|Children's], [9,Sudden Death (1995),Action], [10,GoldenEye
(1995),Action|Adventure|Thriller], [11,American President, The
(1995),Comedy|Drama|Romance], [12,Dracula: Dead and Loving It
(1995),Comedy|Horror], [13,Balto (1995),Animation|Children's], [14,Nixon
(1995),Drama], [15,Cutthroat Island (1995),Action|Adventure|Romance],
[16,Casino (1995),Drama|Thriller], [17,Sense and Sens...

scala>
```

掌握DataFrame查询操作

1，方法一将DataFrame注册成为临时表，然后通过SQL语句进行查询

```
val people=sc.textFile("/user/myname/sparkSql/people.txt")
val schemaString ="name age"
import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{StructType,StructField,StringType}
val schema=StructType(schemaString.split("
").map(fieldName=>StructField(fieldName,StringType,true)))
val rowRDD=people.map(_.split(",")).map(p=>Row(p(0),p(1).trim))
val peopleDataFrame=spark.createDataFrame(rowRDD,schema)
peopleDataFrame.registerTempTable("peopleTempTab")
val personsRDD=spark.sql("select name,age from peopleTempTab where age
>20").rdd
personsRDD.collect
```

运行结果如：

```
scala> val people=sc.textFile("/user/myname/sparkSql/people.txt")
people: org.apache.spark.rdd.RDD[String] = /user/myname/sparkSql/people.txt
MapPartitionsRDD[42] at textFile at <console>:24
```

```
scala> val schemaString ="name age"
schemaString: String = name age

scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> import org.apache.spark.sql.types.{StructType,StructField,StringType}
import org.apache.spark.sql.types.{StructType, StructField, StringType}

scala> val schema=StructType(schemaString.split("
").map(fieldName=>StructField(fieldName,StringType,true)))
schema: org.apache.spark.sql.types.StructType =
StructType(StructField(name,StringType,true),
StructField(age,StringType,true))

scala> val rowRDD=people.map(_.split(",")).map(p=>Row(p(0),p(1).trim))
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] =
MapPartitionsRDD[44] at map at <console>:27

scala> val peopleDataFrame=spark.createDataFrame(rowRDD,schema)
peopleDataFrame: org.apache.spark.sql.DataFrame = [name: string, age:
string]

scala> peopleDataFrame.registerTempTable("peopleTempTab")
warning: there was one deprecation warning (since 2.0.0); for details,
enable `:setting -deprecation' or `:replay -deprecation'

scala> val personsRDD=spark.sql("select name,age from peopleTempTab where
age >20").rdd
personsRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] =
MapPartitionsRDD[49] at rdd at <console>:25

scala> personsRDD.collect
res13: Array[org.apache.spark.sql.Row] = Array([Michael,29], [Andy,30])


scala>
```

2，方法二直接在DataFrame对象上进行查询

DataFrame提供了很多查询的方法，是Transformation操作,是懒操作

方法	描述
where	条件查询
select/selectExpr/col/apply	查询指定字段的数据信息
limit	查询前n行记录
order by	排序查询
group by	分组查询
join	连接查询

创建DataFrame对象rating和user

```
case class Rating(userId:Int,movieId:Int,rating:Int,timestamp:Long)
var
ratingData=sc.textFile("/user/myname/sparkSql/ratings.dat").map(_.split("::"))
val
rating=ratingData.map(r=>Rating(r(0).trim.toInt,r(1).trim.toInt,r(2).trim.toIn
t,r(3).trim.toLong)).toDF()
case class User(userId:Int,gender:String,age:Int,occupation:Int,zip:String)
val userData=sc.textFile("/user/myname/sparkSql/users.dat").map(_.split("::"))
val
user=userData.map(u=>User(u(0).trim.toInt,u(1),u(2).trim.toInt,u(3).trim.toInt
,u(4))).toDF()
```

运行结果：

```
scala> case class Rating(userId:Int,movieId:Int,rating:Int,timestamp:Long)
defined class Rating

scala> var
ratingData=sc.textFile("/user/myname/sparkSql/ratings.dat").map(_.split("::"
))
ratingData: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[52]
at map at <console>:26

scala> val
rating=ratingData.map(r=>Rating(r(0).trim.toInt,r(1).trim.toInt,r(2).trim.to
Int,r(3).trim.toLong)).toDF()
rating: org.apache.spark.sql.DataFrame = [userId: int, movieId: int ... 2
more fields]

scala> case class
User(userId:Int,gender:String,age:Int,occupation:Int,zip:String)
defined class User

scala> val
userData=sc.textFile("/user/myname/sparkSql/users.dat").map(_.split("::"))
userData: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[56] at
map at <console>:26

```

```
scala> val user=userData.map(u=>User(u(0).trim.toInt,u(1),u(2).trim.toInt,u(3).trim.toInt,u(4))).toDF()
user: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 3 more fields]

scala>
```

查询条件:where查询&filter

where查询

```
val userWhere=user.where("gender='F' and age=18")
userWhere.show(3)
```

结果:

```
scala> val userWhere=user.where("gender='F' and age=18")
userWhere: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId: int, gender: string ... 3 more fields]

scala> userWhere.show(3)
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|    18|     F| 18|         3|95825|
|    34|     F| 18|         0|02135|
|    38|     F| 18|         4|02215|
+------+------+---+----------+-----+
only showing top 3 rows


scala>
```

filter

```
val userFilter=user.filter("gender='F' and age=18")
userFilter.show(3)
```

结果:

```

scala> val userFilter=user.filter("gender='F' and age=18")
userFilter: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId: int, gender: string ... 3 more fields]

scala> userFilter.show(3)
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|    18|     F| 18|         3|95825|
|    34|     F| 18|         0|02135|
|    38|     F| 18|         4|02215|
+------+------+---+----------+-----+
```

```
only showing top 3 rows


scala>
```

查询指定字段的数据信息select&selectExpr&col&apply

select:获取指定字段值

```
val userSelect=user.select("userId","gender")
userSelect.show
```

运行结果：

```
scala> val userSelect=user.select("userId","gender")
userSelect: org.apache.spark.sql.DataFrame = [userId: int, gender: string]

scala> userSelect.show
+------+------+
|userId|gender|
+------+------+
|     1|     F|
|     2|     M|
|     3|     M|
|     4|     M|
|     5|     M|
|     6|     F|
|     7|     M|
|     8|     M|
|     9|     M|
|    10|     F|
|    11|     F|
|    12|     M|
|    13|     M|
|    14|     M|
|    15|     M|
|    16|     F|
|    17|     M|
|    18|     F|
|    19|     M|
|    20|     M|
+------+------+
only showing top 20 rows


scala>
```

selectExpr:对指定字段进行特殊处理

```
spark.udf.register("replace",(x:String)=>
 {x match{
 case "M"=>0
 case "F"=>1
 }})
 val userSelectExpr=user.selectExpr("userId","replace(gender) as sex","age")
 userSelectExpr.show(3)
```

运行结果：

```
scala>  val userSelectExpr=user.selectExpr("userId","replace(gender) as sex","age")
userSelectExpr: org.apache.spark.sql.DataFrame = [userId: int, sex: int ... 1 more field]

scala>  userSelectExpr.show(3)
+------+---+---+
|userId|sex|age|
+------+---+---+
|     1|  1|  1|
|     2|  0| 56|
|     3|  0| 25|
+------+---+---+
only showing top 3 rows


scala>
```

col/apply

获取DataFrame指定字段,但是只能获取一个字段，并且返回对象为Column类型

```
val userCol=user.col("zip")
user.select(userCol).collect
val userApply=user.apply("zip")
user.select(userApply).collect
```

运行结果：

```
scala> val userCol=user.col("zip")
userCol: org.apache.spark.sql.Column = zip

scala> user.select(userCol).collect
res19: Array[org.apache.spark.sql.Row] = Array([48067], [70072], [55117],
[02460], [55455], [55117], [06810], [11413], [61614], [95370], [04093],
[32793], [93304], [60126], [22903], [20670], [95350], [95825], [48073],
[55113], [99353], [53706], [90049], [10023], [01609], [23112], [19130],
[14607], [33407], [19143], [06840], [19355], [55421], [02135], [02482],
[94123], [66212], [02215], [61820], [10543], [15116], [24502], [60614],
[98052], [94110], [75602], [94305], [92107], [77084], [98133], [10562],
[72212], [96931], [56723], [55303], [60440], [30350], [30303], [55413],
[72118], [95122], [98105], [54902], [53706], [55803], [57706], [60181],
[53706], [02143], [53703], [95008], [55122], [53706], [94530], [01748],
[55413], [15321], [98029], [98103], [49327], [606...

scala> val userApply=user.apply("zip")
userApply: org.apache.spark.sql.Column = zip

scala> user.select(userApply).collect
res20: Array[org.apache.spark.sql.Row] = Array([48067], [70072], [55117],
[02460], [55455], [55117], [06810], [11413], [61614], [95370], [04093],
[32793], [93304], [60126], [22903], [20670], [95350], [95825], [48073],
[55113], [99353], [53706], [90049], [10023], [01609], [23112], [19130],
[14607], [33407], [19143], [06840], [19355], [55421], [02135], [02482],
[94123], [66212], [02215], [61820], [10543], [15116], [24502], [60614],
[98052], [94110], [75602], [94305], [92107], [77084], [98133], [10562],
[72212], [96931], [56723], [55303], [60440], [30350], [30303], [55413],
[72118], [95122], [98105], [54902], [53706], [55803], [57706], [60181],
[53706], [02143], [53703], [95008], [55122], [53706], [94530], [01748],
[55413], [15321], [98029], [98103], [49327], [606...

```

limit

获取指定DataFrame的前n行记录

```
val userLimit=user.limit(3)
userLimit.show
```

运行结果：

```
scala> val userLimit=user.limit(3)
userLimit: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId:
int, gender: string ... 3 more fields]

scala> userLimit.show
+------+------+---+----------+-----+

|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|     1|     F|  1|        10|48067|

```

```
|     2|     M| 56|        16|70072|
|     3|     M| 25|        15|55117|
+------+------+---+----------+-----+


scala>
```

orderBy/sort

orderBy排序方法

```
val userOrderBy1=user.orderBy(desc("UserId"))
userOrderBy1.show(3)
val userOrderBy2=user.orderBy($"UserId".desc)
userOrderBy2.show(3)
val userOrderBy3=user.orderBy(-user("userId"))
userOrderBy3.show(3)
```

运行结果：

```
scala> val userOrderBy1=user.orderBy(desc("UserId"))
userOrderBy1: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] =
[userId: int, gender: string ... 3 more fields]

scala> userOrderBy1.show(3)
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|  6041|     M| 25|         7|11107|
|  6040|     M| 25|         6|11106|
|  6039|     F| 45|         0|01060|
+------+------+---+----------+-----+
only showing top 3 rows


scala> val userOrderBy2=user.orderBy($"UserId".desc)
userOrderBy2: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] =
[userId: int, gender: string ... 3 more fields]

scala> userOrderBy2.show(3)
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|  6041|     M| 25|         7|11107|
|  6040|     M| 25|         6|11106|
|  6039|     F| 45|         0|01060|
+------+------+---+----------+-----+
only showing top 3 rows


scala> val userOrderBy3=user.orderBy(-user("userId"))
userOrderBy3: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] =
[userId: int, gender: string ... 3 more fields]

scala> userOrderBy3.show(3)
```

```
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|  6041|     M| 25|         7|11107|
|  6040|     M| 25|         6|11106|
|  6039|     F| 45|         0|01060|
+------+------+---+----------+-----+
only showing top 3 rows


scala>
```

sort方法

```
val userSort1=user.sort(asc("userId"))
userSort1.show(3)
val userSort2=user.sort($"userId".asc)
userSort2.show(3)
val userSort3=user.sort(user("userId"))
userSort3.show(3)
```

运行结果：

```
scala> val userSort1=user.sort(asc("userId"))
userSort1: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId:
int, gender: string ... 3 more fields]

scala> userSort1.show(3)
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|     1|     F|  1|        10|48067|
|     2|     M| 56|        16|70072|
|     3|     M| 25|        15|55117|
+------+------+---+----------+-----+
only showing top 3 rows


scala> val userSort2=user.sort($"userId".asc)
userSort2: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId:
int, gender: string ... 3 more fields]

scala> userSort2.show(3)
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|     1|     F|  1|        10|48067|
|     2|     M| 56|        16|70072|
|     3|     M| 25|        15|55117|
+------+------+---+----------+-----+
only showing top 3 rows


scala> val userSort3=user.sort(user("userId"))
```

```
userSort3: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [userId: int, gender: string ... 3 more fields]

scala> userSort3.show(3)
+------+------+---+----------+-----+
|userId|gender|age|occupation|  zip|
+------+------+---+----------+-----+
|     1|     F|  1|        10|48067|
|     2|     M| 56|        16|70072|
|     3|     M| 25|        15|55117|
+------+------+---+----------+-----+
only showing top 3 rows


```

groupBy

两种分组方法:传入String类型和传入Column类型

groupBy返回的是GroupData对象:

方法	描述
max(colNames:String)	获取分组中指定字段或者所有的数值类型字段的最大值
min(colNames:String)	获取分组中指定字段或者所有的数字类型字段的最小值
mean(colNames:String)	获取分组中指定字段或者所有的数字类型字段的平均值
sum(colNames:String)	获取分组中指定字段或者所有的数字类型字段和值
count()	获取分组中的元素个数

```
val userGroupBy1 = user.groupBy("gender")
val userGroupBy2 = user.groupBy(user("gender"))
val userGroupByCount = user.groupBy(user("gender")).count
userGroupByCount.show()
```

运行结果:

```
scala> val userGroupBy1 = user.groupBy("gender")
userGroupBy1: org.apache.spark.sql.RelationalGroupedDataset = RelationalGroupedDataset: [grouping expressions: [gender: string], value: [userId: int, gender: string ... 3 more fields], type: GroupBy]

scala> val userGroupBy2 = user.groupBy(user("gender"))
userGroupBy2: org.apache.spark.sql.RelationalGroupedDataset = RelationalGroupedDataset: [grouping expressions: [gender: string], value: [userId: int, gender: string ... 3 more fields], type: GroupBy]
scala> val userGroupByCount = user.groupBy(user("gender")).count
userGroupByCount: org.apache.spark.sql.DataFrame = [gender: string, count: bigint]

```

```
scala> userGroupByCount.show()
+------+-----+

|gender|count|
+------+-----+
|     F| 1709|
|     M| 4332|
+------+-----+


scala>
```

join

DataFrame提供了三种join方法用于连接两个表

方法	描述
join(right:DataFrame)	两个表做笛卡尔积
join(right:DataFrame,joinExprs:Column)	根据两表中相同的某个字段进行连接
join(right:DataFrame,joinExprs:Column,joinType:String)	根据两表中相同的某个字段进行连接并指定连接类型

连接查询

```
val dfJoin = user.join(rating,"userId")
dfJoin.show(3)
val dfJoin2 = dfJoin.join(user,Seq("userId","gender"),"left_outer")
dfJoin2.show(3)
```

运行结果

```
scala> val dfJoin = user.join(rating,"userId")
dfJoin: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 6 more fields]

scala> dfJoin.show(3)
+------+------+---+----------+-----+-------+------+---------+
|userId|gender|age|occupation|  zip|movieId|rating|timestamp|
+------+------+---+----------+-----+-------+------+---------+
|   148|     M| 50|        17|57747|   2987|     5|979576038|
|   148|     M| 50|        17|57747|   2989|     4|977333611|
|   148|     M| 50|        17|57747|    647|     3|977352859|
+------+------+---+----------+-----+-------+------+---------+
only showing top 3 rows

scala> val dfJoin2 = dfJoin.join(user,Seq("userId","gender"),"left_outer")
dfJoin2: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 9 more fields]

scala> dfJoin2.show(3)
```

```
+------+------+---+----------+-----+-------+------+---------+---+----------
+-----+
|userId|gender|age|occupation|  zip|movieId|rating|timestamp|age|occupation|
 zip|
+------+------+---+----------+-----+-------+------+---------+---+----------
+-----+
|    71|     M| 25|        14|95008|    648|     4|977875582| 25|
 14|95008|
|    71|     M| 25|        14|95008|   1250|     4|977875378| 25|
 14|95008|
|    71|     M| 25|        14|95008|   2997|     3|977875912| 25|
 14|95008|
+------+------+---+----------+-----+-------+------+---------+---+----------
+-----+
only showing top 3 rows


scala>
```

掌握DataFrame输出操作

方法一

1，创建Map对象，存储数据，如保存路径，json文件头信息

val saveOptions=Map("header"->"true","path"->"/user/myname/sparkSql/copyOfUser.json")

2，从DataFrame对象中选择出userId,gender和age

val copyOfUser=user.select("userId","gender","age")

3, 调用save函数保存2中的

DataFrame数据到文件中

import org.apache.spark.sql.SaveMode
copyOfUser.write.format("json").mode(SaveMode.Overwrite).options(saveOptions).save()

4，查看保存结果

hdfs上查看目录：/user/myname/sparkSql/copyOfUser.json

源码

```
import org.apache.spark.sql.SaveMode
case class User(userId:Int,gender:String,age:Int,occupation:Int,zip:String)
val userData=sc.textFile("/user/myname/sparkSql/users.dat").map(_.split("::"))
val
user=userData.map(u=>User(u(0).trim.toInt,u(1),u(2).trim.toInt,u(3).trim.toInt
,u(4))).toDF()
val saveOptions=Map("header"->"true","path"-
>"/user/myname/sparkSql/copyOfUser.json")
val copyOfUser=user.select("userId","gender","age")
copyOfUser.write.format("json").mode(SaveMode.Overwrite).options(saveOptions).
save()
```

运行结果

```
scala> import org.apache.spark.sql.SaveMode
import org.apache.spark.sql.SaveMode

scala> case class
User(userId:Int,gender:String,age:Int,occupation:Int,zip:String)
defined class User

scala> val
userData=sc.textFile("/user/myname/sparkSql/users.dat").map(_.split("::"))
userData: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[32] at
map at <console>:25

scala> val
user=userData.map(u=>User(u(0).trim.toInt,u(1),u(2).trim.toInt,u(3).trim.toI
nt,u(4))).toDF()
user: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 3
more fields]

scala> val saveOptions=Map("header"->"true","path"-
>"/user/myname/sparkSql/copyOfUser.json")
saveOptions: scala.collection.immutable.Map[String,String] = Map(header ->
true, path -> /user/myname/sparkSql/copyOfUser.json)

scala> val copyOfUser=user.select("userId","gender","age")
copyOfUser: org.apache.spark.sql.DataFrame = [userId: int, gender: string
... 1 more field]

scala>
copyOfUser.write.format("json").mode(SaveMode.Overwrite).options(saveOptions
).save()


scala>
```

方法二

1，直接调用save方法

```
val dfPeople =spark.read.json("/user/myname/sparkSql/people.json")
dfPeople.show()
import org.apache.spark.sql.SaveMode
dfPeople.write.mode(SaveMode.Overwrite).json("/user/myname/sparkSql/people_cop
y")
```

说明：

DataFrameReader.json读文件,返回DataFrame ， 所以dfPeople是DataFrame

dfPeople.write, 返回DataFrameWriter对象， DataFrameWriter.json保存文件

运行结果：

```
scala> import org.apache.spark.sql.SaveMode
import org.apache.spark.sql.SaveMode

scala> case class
User(userId:Int,gender:String,age:Int,occupation:Int,zip:String)
defined class User

scala> val
userData=sc.textFile("/user/myname/sparkSql/users.dat").map(_.split("::"))
userData: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[32] at
map at <console>:25

scala> val
user=userData.map(u=>User(u(0).trim.toInt,u(1),u(2).trim.toInt,u(3).trim.toI
nt,u(4))).toDF()
user: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 3
more fields]

scala> val saveOptions=Map("header"->"true","path"-
>"/user/myname/sparkSql/copyOfUser.json")
saveOptions: scala.collection.immutable.Map[String,String] = Map(header ->
true, path -> /user/myname/sparkSql/copyOfUser.json)

scala> val copyOfUser=user.select("userId","gender","age")
copyOfUser: org.apache.spark.sql.DataFrame = [userId: int, gender: string
... 1 more field]

scala>
copyOfUser.write.format("json").mode(SaveMode.Overwrite).options(saveOptions
).save()


scala>
```

2，saveAsTable方法保存到表

```
case class User(userId:Int,gender:String,age:Int,occupation:Int,zip:String)
val userData=sc.textFile("/user/myname/sparkSql/users.dat").map(_.split("::"))
val
user=userData.map(u=>User(u(0).trim.toInt,u(1),u(2).trim.toInt,u(3).trim.toInt
,u(4))).toDF()

val copyOfUser=user.select("userId","gender","age")
copyOfUser.write.saveAsTable("copyUser")
spark.sql("select * from copyUser").show(5)
```

运行结果：

```
scala> case class
User(userId:Int,gender:String,age:Int,occupation:Int,zip:String)
defined class User


```

```
scala> val userData=sc.textFile("/user/myname/sparkSql/users.dat").map(_.split("::"))
userData: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[6] at map at <console>:24

scala> val user=userData.map(u=>User(u(0).trim.toInt,u(1),u(2).trim.toInt,u(3).trim.toInt,u(4))).toDF()
user: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 3 more fields]

scala>
     | val copyOfUser=user.select("userId","gender","age")
copyOfUser: org.apache.spark.sql.DataFrame = [userId: int, gender: string ... 1 more field]

scala> copyOfUser.write.saveAsTable("copyUser")


scala> spark.sql("select * from copyUser").show(5)
+------+------+---+

|userId|gender|age|
+------+------+---+
|     1|     F|  1|
|     2|     M| 56|
|     3|     M| 25|
|     4|     M| 45|
|     5|     M| 25|
+------+------+---+
only showing top 5 rows


```

任务5.3探索分析房屋售价数据

任务描述

从购房者角度出发，影响用户购房的主要因素有房屋价格、房屋属性。从地产公司角度出发，除了房屋本身的居住属性外，地产公司也会考虑什么样的房屋属性在市场上更具有价值。本节的任务如下。

使用Spark SQL实现房价数据的探索分析，从数据中的房屋售价、房屋评分和房屋销售日期等维度进行探索分析，获得房价波动的内在原因。

获取数据

创建表并导入数据

准备数据

```
hdfs dfs -put /root/spark/spark_data/house.csv /user/myname/sparkSql/
hdfs dfs -ls /user/myname/sparkSql/house.csv
```

打开hive,运行如下创建表和load数据

```
create database house;
use house;
create table king_county_house(
selling_price double,
bedrooms_num double,
bathroom_num double,
housing_area double,
parking_area double,
floor_num double,
housing_rating double,
built_area double,
basement_area double,
year_built int,
year_repair int,
latitude double,
longitude double,
sale_data string
)
row format delimited fields terminated by ',';
load data local inpath '/root/spark/spark_data/house.csv' overwrite into
table king_county_house;
```

运行结果：

```
hive> use house;
OK
Time taken: 0.089 seconds
hive> create table king_county_house(
    > selling_price double,
    > bedrooms_num double,
    > bathroom_num double,
    > housing_area double,
    > parking_area double,
    > floor_num double,
    > housing_rating double,
    > built_area double,
    > basement_area double,
    > year_built int,
    > year_repair int,
    > latitude double,
    > longitude double,
    > sale_data string
    > )
    > row format delimited fields terminated by ',';
OK
Time taken: 1.354 seconds
```

```
hive> load data local inpath '/root/spark/spark_data/house.csv' overwrite into table king_county_house;
Loading data to table house.king_county_house
79774 [cd973e9f-ac65-4d34-96d5-409feef08b62 main] WARN org.apache.hadoop.hive.metastore.ObjectStore  - datanucleus.autoStartMechanismMode is set to unsupported value null . Setting it to value: ignored
80836 [cd973e9f-ac65-4d34-96d5-409feef08b62 main] WARN org.apache.hadoop.hive.metastore.ObjectStore  - datanucleus.autoStartMechanismMode is set to unsupported value null . Setting it to value: ignored
OK
Time taken: 1.528 seconds
hive>
```

在spark-shell中读取数据

```
val house = spark.table("house.king_county_house")
house.take(3)
```

运行结果：

```
scala> val house = spark.table("house.king_county_house")
house: org.apache.spark.sql.DataFrame = [selling_price: double, bedrooms_num: double ... 12 more fields]

scala> house.take(3)
res6: Array[org.apache.spark.sql.Row] = Array([null,null,null,null,null,null,null,null,null,null,null,null,null,sale_data],
[545000.0,3.0,2.25,1670.0,6240.0,1.0,8.0,1240.0,430.0,1974,0,47.6413,-122.113,20200302],
[785000.0,4.0,2.5,3300.0,10514.0,2.0,10.0,3300.0,0.0,1984,0,47.6323,-122.036,20200211])

scala>
```

任务实现

源码实现：

```
package main.java

import org.apache.spark.sql.types.DataTypes
import org.apache.spark.sql.{ColumnName, DataFrame, SparkSession}

object process {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .master("local")
      .appName("process")
      .config("spark.some.config.option", "some-value")
```

```
      .enableHiveSupport()
      .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    import org.apache.spark.sql.functions._
    import spark.implicits._
    // 读取数据集
    val house = spark.table("house.king_county_house")
    val dataColumnName = house.columns.toList
    for (i <- dataColumnName) {
      if (i == "sale_data" || i == "year_built" || i == "year_repair") {
        println(i + ":")
        null_count(house, i)
        println("-" * 20)
      }
      else {
        max_min_mean_std(house, i)
      }
    }
    // 转换时间格式
    val houseDate = house.na.drop()
      .withColumn("date", to_date(col("sale_data"), "yyyyMMdd"))
    // 划分季度
    val houseQuarter = houseDate.withColumn("quarter", quarter(col("date")))
    houseQuarter.show()
    // 统计每个季度的房屋销量
    houseQuarter.groupBy("quarter").count().sort("quarter").show()
    // 统计每个季度的房屋销售总额
    houseQuarter.groupBy(
      "quarter").sum("selling_price").sort("quarter").show()
    houseQuarter.selectExpr("mean(selling_price)").show(100)

    // 房屋评分分布统计

houseQuarter.groupBy("housing_rating").count().sort(desc("count")).show()

    // 房屋评分与单位售价的关系
    houseQuarter.groupBy("housing_rating").agg(
      avg(col("selling_price") / col("housing_area"))).sort(
      "housing_rating").show()
    val houseAgeTitleUdf = udf((x: Int) => AgeTitle(x))
    val houseRepair = houseQuarter.filter(
      "year_repair != 0").withColumn(
      "houseAge", -(col("year_built").cast(DataTypes.IntegerType) - 2020))
    val houseAgeTitle =houseRepair.withColumn(
      "houseAgeTitle", houseAgeTitleUdf(col("houseAge")))
    houseAgeTitle.groupBy("houseAgeTitle").count().show()
  }

  // 求字段的最大值、最小值、均值及标准差
  def max_min_mean_std(data: DataFrame, columnName: String): Unit = {
    println(columnName + ":")
    data.selectExpr("max(" + columnName + ") as max")
      .foreach(line => println("max:" + line.toString()))
    data.selectExpr("min(" + columnName + ") as min")
      .foreach(line => println("min:" + line.toString()))
```

```
    data.selectExpr("mean(" + columnName + ") as mean")
      .foreach(line => println ("mean:" + line.toString()))
    data.selectExpr("stddev(" + columnName + ") as std")
      .foreach(line => println("std:" + line.toString()))
    null_count(data, columnName)
    println("-" * 20)
  }

  def null_count(data: DataFrame, columnName: String): Unit = {
    println(columnName + "缺失值数：" + (data.count() -
data.na.drop().count()))
  }

  def AgeTitle(age: Int) = {
    if (age >= 30) {
      "old"
    }
    else if (age <= 10) {
      "new"
    }
    else {
      "middle"
    }
  }
}
```

打包上传

使用sftp上传word.jar，例如：其中D:\yuxm\wordcount\out\artifacts\word要根据本机情况修改

```
lcd D:\yuxm\wordcount\out\artifacts\word
cd /root/spark/
put word.jar
```

结果如：

```
sftp> lcd D:\yuxm\wordcount\out\artifacts\word
sftp> cd /root/spark/
sftp> put word.jar
Uploading word.jar to /root/spark/word.jar
  100% 29KB     29KB/s 00:00:00
D:/yuxm/wordcount/out/artifacts/word/word.jar: 30327 bytes transferred in 0
seconds (29 KB/s)
sftp>

```

运行结果：

切换到shell命令行执行脚本

```
spark-submit --master local --class main.java.process /root/spark/word.jar
```

运行结果：

```
[root@master ~]# spark-submit --master local --class main.java.process /root/spark/word.jar
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/spark/jars/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.3.1/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
selling_price:
max:[6885000.0]
min:[75000.0]
mean:[542874.9288]
std:[372925.76587803545]
selling_price缺失值数：1
--------------------
bedrooms_num:
max:[10.0]
min:[0.0]
mean:[3.3676]
std:[0.8931685255322745]
bedrooms_num缺失值数：1
--------------------
bathroom_num:
max:[7.75]
min:[0.0]
mean:[2.1168]
std:[0.7740995950252791]
bathroom_num缺失值数：1
--------------------
housing_area:
max:[9890.0]
min:[390.0]
mean:[2082.4884]
std:[922.8789159010951]
housing_area缺失值数：1
--------------------
parking_area:
max:[1651359.0]
min:[572.0]
mean:[15352.7339]
std:[45776.22980321974]
parking_area缺失值数：1
--------------------
floor_num:
max:[3.5]
min:[1.0]
mean:[1.50285]
std:[0.5436418233940561]
floor_num缺失值数：1
--------------------
housing_rating:
```

```
max:[13.0]
min:[3.0]
mean:[7.6648]
std:[1.173873399424226]
housing_rating缺失值数：1
--------------------
built_area:
max:[8860.0]
min:[390.0]
mean:[1791.4749]
std:[829.449437281676]
built_area缺失值数：1
--------------------
basement_area:
max:[4820.0]
min:[0.0]
mean:[291.0135]
std:[446.64133907009403]
basement_area缺失值数：1
--------------------
year_built:
year_built缺失值数：1
--------------------
year_repair:
year_repair缺失值数：1
--------------------
latitude:
max:[47.7776]
min:[47.1593]
mean:[47.5606288899999]
std:[0.13857015670482747]
latitude缺失值数：1
--------------------
longitude:
max:[-121.315]
min:[-122.519]
mean:[-122.21584520000049]
std:[0.13973856419611894]
longitude缺失值数：1
--------------------
sale_data:
sale_data缺失值数：1
--------------------
+-------------+------------+------------+------------+------------+--------
-+-------------+----------+-------------+----------+----------+--------+-
--------+---------+----------+-------+
|selling_price|bedrooms_num|bathroom_num|housing_area|parking_area|floor_nu
m|housing_rating|built_area|basement_area|year_built|year_repair|latitude|l
ongitude|sale_data|      date|quarter|
+-------------+------------+------------+------------+------------+--------
-+-------------+----------+-------------+----------+----------+--------+-
--------+---------+----------+-------+
|     545000.0|         3.0|        2.25|      1670.0|      6240.0|
 1.0|           8.0|    1240.0|        430.0|      1974|          0|
47.6413| -122.113| 20200302|2020-03-02|      1|
```

```
|     785000.0|         4.0|         2.5|      3300.0|     10514.0|
 2.0|          10.0|    3300.0|          0.0|      1984|          0|
47.6323| -122.036| 20200211|2020-02-11|      1|
|     765000.0|         3.0|        3.25|      3190.0|      5283.0|
 2.0|           9.0|    3190.0|          0.0|      2007|          0|
47.5534| -122.002| 20200107|2020-01-07|      1|
|     720000.0|         5.0|         2.5|      2900.0|      9525.0|
 2.0|           9.0|    2900.0|          0.0|      1989|          0|
47.5442| -122.138| 20191103|2019-11-03|      4|
|     449500.0|         5.0|        2.75|      2040.0|      7488.0|
 1.0|           7.0|    1200.0|        840.0|      1969|          0|
47.7289| -122.172| 20190603|2019-06-03|      2|
|     248500.0|         2.0|         1.0|       780.0|     10064.0|
 1.0|           7.0|     780.0|          0.0|      1958|          0|
47.4913| -122.318| 20200506|2020-05-06|      2|
|     675000.0|         4.0|         2.5|      1770.0|      9858.0|
 1.0|           8.0|    1770.0|          0.0|      1971|          0|
47.7382| -122.287| 20200305|2020-03-05|      1|
|     730000.0|         2.0|        2.25|      2130.0|      4920.0|
 1.5|           7.0|    1530.0|        600.0|      1941|          0|
 47.573| -122.409| 20190701|2019-07-01|      3|
|     311000.0|         2.0|         1.0|       860.0|      3300.0|
 1.0|           6.0|     860.0|          0.0|      1903|          0|
47.5496| -122.279| 20190807|2019-08-07|      3|
|     660000.0|         2.0|         1.0|       960.0|      6263.0|
 1.0|           6.0|     960.0|          0.0|      1942|          0|
47.6646| -122.202| 20191204|2019-12-04|      4|
|     435000.0|         2.0|         1.0|       990.0|      5643.0|
 1.0|           7.0|     870.0|        120.0|      1947|          0|
47.6802| -122.298| 20200227|2020-02-27|      1|
|     350000.0|         3.0|         1.0|      1240.0|     10800.0|
 1.0|           7.0|    1240.0|          0.0|      1959|          0|
47.5233| -122.185| 20190904|2019-09-04|      3|
|     385000.0|         3.0|        2.25|      1630.0|      1598.0|
 3.0|           8.0|    1630.0|          0.0|      2008|          0|
47.6904| -122.347| 20190902|2019-09-02|      3|
|     235000.0|         2.0|         1.0|       930.0|     10505.0|
 1.0|           6.0|     930.0|          0.0|      1930|          0|
47.4337| -122.329| 20200413|2020-04-13|      2|
|     350000.0|         3.0|         1.0|      1300.0|     10236.0|
 1.0|           6.0|    1300.0|          0.0|      1971|          0|
47.5028|  -121.77| 20190930|2019-09-30|      3|
|    1350000.0|         4.0|        1.75|      2000.0|      3728.0|
 1.5|           9.0|    1820.0|        180.0|      1926|          0|
 47.643| -122.299| 20200507|2020-05-07|      2|
|     459900.0|         3.0|        1.75|      2580.0|     11000.0|
 1.0|           7.0|    1290.0|       1290.0|      1951|          0|
47.5646| -122.181| 20190530|2019-05-30|      2|
|     430000.0|         6.0|         3.0|      2630.0|      8800.0|
 1.0|           7.0|    1610.0|       1020.0|      1959|          0|
47.7166| -122.293| 20190723|2019-07-23|      3|
|     718000.0|         5.0|        2.75|      2930.0|      7663.0|
 2.0|           9.0|    2930.0|          0.0|      2013|          0|
47.5308| -122.184| 20191003|2019-10-03|      4|
```

```
|     220000.0|         4.0|         1.0|      1440.0|      8250.0|
 1.0|           7.0|    1440.0|          0.0|      1959|          0|
47.4325| -122.291| 20191215|2019-12-15|      4|
+-------------+------------+------------+------------+------------+--------
-+--------------+----------+-------------+----------+-----------+--------+-
--------+---------+----------+-------+
only showing top 20 rows

+-------+-----+
|quarter|count|
+-------+-----+
|      1| 1906|
|      2| 3139|
|      3| 2759|
|      4| 2196|
+-------+-----+

+-------+------------------+
|quarter|sum(selling_price)|
+-------+------------------+
|      1|     1.006865855E9|
|      2|     1.770422075E9|
|      3|     1.474271351E9|
|      4|     1.177190007E9|
+-------+------------------+

+--------------------+
|mean(selling_price)|
+--------------------+
|        542874.9288|
+--------------------+

+--------------+-----+
|housing_rating|count|
+--------------+-----+
|           7.0| 4162|
|           8.0| 2828|
|           9.0| 1212|
|           6.0|  925|
|          10.0|  514|
|          11.0|  192|
|           5.0|  100|
|          12.0|   45|
|           4.0|   14|
|          13.0|    6|
|           3.0|    2|
+--------------+-----+

+--------------+-----------------------------------+
|housing_rating|avg((selling_price / housing_area))|
+--------------+-----------------------------------+
|           3.0|                289.3034825870647|
|           4.0|                421.6765988891822|
|           5.0|                 244.872904522352|
|           6.0|                274.3597415143096|
```

```
|           7.0|                252.72629268682468|
|           8.0|                 260.1858918866025|
|           9.0|                274.36099140045263|
|          10.0|                 311.8355441252322|
|          11.0|                 337.7554628229812|
|          12.0|                407.72963099333407|
|          13.0|                495.34880449365454|
+--------------+-----------------------------------+

+-------------+-----+
|houseAgeTitle|count|
+-------------+-----+
|          old|  425|
|       middle|    2|
+-------------+-----+

[root@master ~]#
```

实训作业

统计分析某公司每年的产品销售量及销售额

训练要点

- 1，RDD转化为DataFrame的方法
- 2，DataFrame的查询操作

需求说明

以某公司的交易数据，该数据包含3个.txt文档数据，分别为日期数据tbData.txt，订单表头数据tbStock.txt和订单明细tbStockDetail.txt.

将tbData.txt,tbStock.txt,tbStockDetail.txt上传到 HDFS,通过sparkContext读取该数据得到一个RDD,将该RDD数据转换成DataFrame进行查询分析。

分析目标如：

- 1，计算所有订单中每年的销售单数，销售总额
- 2，计算所有订单每年最大金额订单的销售额
- 3，计算所有订单中每年最畅销货品

实现思路及步骤

- 1，将数据上传到HDFS
- 2，使用SparkContext读取HDFS上的数据
- 3，将2中得到的RDD数据转为DataFrame，并注册成为临时表
- 4，针对分析目标1，使用join连接3个临时表进行查询
- 5，针对分析目标2，首先求出每份订单的销售额以其发生时间，然后以第1步的查询结果和tbData连接，求出每年最大金额订单的销售订单的销售额
- 6，针对分析目标3，首先求出每年每个货品的销售利息额，然后求出每年单品销售的最大金额，最后求出每年与销售额最大相符的货品就是最畅销货品

作业要求

- 1，提供分析过程的所有脚本
- 2，提供分析目标1,2,3的运行过程截图
- 3，提供分析目标1,2,3的运行结果截图

参考

1,准备数据

上传实训数据到hdfs

```
wget -c http://bigdata.hddly.cn/b46488/file/train5data.tar.gz
 tar -xzvf ./train5data.tar.gz
 hdfs dfs -put -d ./train5data /user/myname/sparkSql/
 hdfs dfs -ls /user/myname/sparkSql/train5data
```

2,加载数据

```
case class tbStock(ordernumber:String,locationid:String,dateid:String) extends Serializable
 val tbStockRdd = spark.sparkContext.textFile("/user/myname/sparkSql/train5data/tbStock.txt")
 val tbStockDS = tbStockRdd.map(_.split(",")).map(attr=> tbStock(attr(0),attr(1),attr(2))).toDS
 tbStockDS.show

 case class tbStockDetail(ordernumber:String, rownum:Int, itemid:String, number:Int, price:Double, amount:Double) extends Serializable
 val tbStockDetailRdd = spark.sparkContext.textFile("/user/myname/sparkSql/train5data/tbStockDetail.txt")
 val tbStockDetailDS = tbStockDetailRdd.map(_.split(",")).map(attr=> tbStockDetail(attr(0),attr(1).trim().toInt,attr(2),attr(3).trim().toInt,attr(4).trim().toDouble, attr(5).trim().toDouble)).toDS
 tbStockDetailDS.show()

 case class tbDate(dateid:String, years:Int, theyear:Int, month:Int, day:Int, weekday:Int, week:Int, quarter:Int, period:Int, halfmonth:Int) extends Serializable
 val tbDateRdd = spark.sparkContext.textFile("/user/myname/sparkSql/train5data/tbDate.txt")
 val tbDateDS = tbDateRdd.map(_.split(",")).map(attr=> tbDate(attr(0),attr(1).trim().toInt, attr(2).trim().toInt,attr(3).trim().toInt, attr(4).trim().toInt, attr(5).trim().toInt, attr(6).trim().toInt, attr(7).trim().toInt, attr(8).trim().toInt, attr(9).trim().toInt)).toDS
 tbDateDS.show()
```

3,注册表

```
tbStockDS.createOrReplaceTempView("tbStock")
 tbDateDS.createOrReplaceTempView("tbDate")
 tbStockDetailDS.createOrReplaceTempView("tbStockDetail")
```

4,查询统计

计算所有订单中每年的销售单数、销售总额

```
spark.sql("SELECT c.theyear, COUNT(DISTINCT a.ordernumber), SUM(b.amount) FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid = c.dateid GROUP BY c.theyear ORDER BY c.theyear").show
```

计算所有订单每年最大金额订单的销售额

```
spark.sql("SELECT a.dateid, a.ordernumber, SUM(b.amount) AS SumOfAmount FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber GROUP BY a.dateid, a.ordernumber").show
 spark.sql("SELECT theyear, MAX(c.SumOfAmount) AS SumOfAmount FROM (SELECT a.dateid, a.ordernumber, SUM(b.amount) AS SumOfAmount FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber GROUP BY a.dateid, a.ordernumber ) c JOIN tbDate d ON c.dateid = d.dateid GROUP BY theyear ORDER BY theyear DESC").show
```

计算所有订单中每年最畅销货品

```
spark.sql("SELECT c.theyear, b.itemid, SUM(b.amount) AS SumOfAmount FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid = c.dateid GROUP BY c.theyear, b.itemid").show

 spark.sql("SELECT d.theyear, MAX(d.SumOfAmount) AS MaxOfAmount FROM (SELECT c.theyear, b.itemid, SUM(b.amount) AS SumOfAmount FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid = c.dateid GROUP BY c.theyear, b.itemid ) d GROUP BY d.theyear").show

 spark.sql("SELECT DISTINCT e.theyear, e.itemid, f.maxofamount FROM (SELECT c.theyear, b.itemid, SUM(b.amount) AS sumofamount FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid = c.dateid GROUP BY c.theyear, b.itemid ) e JOIN (SELECT d.theyear, MAX(d.sumofamount) AS maxofamount FROM (SELECT c.theyear, b.itemid, SUM(b.amount) AS sumofamount FROM tbStock a JOIN tbStockDetail b ON a.ordernumber = b.ordernumber JOIN tbDate c ON a.dateid = c.dateid GROUP BY c.theyear, b.itemid ) d GROUP BY d.theyear ) f ON e.theyear = f.theyear AND e.sumofamount = f.maxofamount ORDER BY e.theyear").show
```

常见问题

问题1：hive 插入数据，insert后报错 FAILED: Execution Error, return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask

hive 插入数据的时候，不能直接运行，报错

问题现象

```
hive> insert into students1(id,name,score,classes)
values(10,'limm10',90,'class1');
401282 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.hive.ql.session.SessionState  - METASTORE_FILTER_HOOK will
be ignored, since hive.security.authorization.manager is set to instance of
HiveAuthorizerFactory.
401475 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null .
Setting it to value: ignored
401931 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null .
Setting it to value: ignored
413409 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null .
Setting it to value: ignored
414744 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.hive.ql.Driver  - Hive-on-MR is deprecated in Hive 2 and
may not be available in the future versions. Consider using a different
execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20231007070737_e1b391bf-7584-43fd-a78f-5037068cb77f
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
420632 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.mapreduce.JobResourceUploader  - Hadoop command-line
option parsing not performed. Implement the Tool interface and execute your
application with ToolRunner to remedy this.
Starting Job = job_1696262549114_0004, Tracking URL =
http://master:8088/proxy/application_1696262549114_0004/
Kill Command = /usr/local/hadoop-3.3.1/bin/mapred job  -kill
job_1696262549114_0004
Hadoop job information for Stage-1: number of mappers: 0; number of
reducers: 0
499586 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN  mapreduce.Counters
- Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use
org.apache.hadoop.mapreduce.TaskCounter instead
2023-10-07 07:09:18,189 Stage-1 map = 0%,  reduce = 0%
```

```
499621 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN  mapreduce.Counters - Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use org.apache.hadoop.mapreduce.TaskCounter instead
Ended Job = job_1696262549114_0004 with errors
499674 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] ERROR org.apache.hadoop.hive.ql.exec.Task  - Ended Job = job_1696262549114_0004 with errors
Error during job, obtaining debugging information...
499687 [Thread-545] ERROR org.apache.hadoop.hive.ql.exec.Task  - Error during job, obtaining debugging information...
FAILED: Execution Error, return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask
499956 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] ERROR org.apache.hadoop.hive.ql.Driver  - FAILED: Execution Error, return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask
MapReduce Jobs Launched:
500076 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN  mapreduce.Counters - Group FileSystemCounters is deprecated. Use org.apache.hadoop.mapreduce.FileSystemCounter instead
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 0 FAIL
Total MapReduce CPU Time Spent: 0 msec
```

错误原因：

namenode内存空间不够，JVM剩余内存空间不够新job运行所致

解决办法：

在hive> 下，输入

```
set hive.exec.mode.local.auto=true;
```

通过命令修改的话在重启hive后会失效，要长期有效，修改文件：

```
vi /usr/local/hive/conf/hive-site.xml
```

修改hive.exec.mode.local.auto的value为true,如下：

```
  <property>
    <name>hive.exec.mode.local.auto</name>
    <value>true</value>
    <description>Let Hive determine whether to run in local mode automatically</description>
  </property>
```

再次验证

再次运行insert语句，结果如：

```
hive> set hive.exec.mode.local.auto=true;
hive> insert into students1(id,name,score,classes) values(10,'limm10',90,'class1');
```

```
1351659 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.hive.metastore.ObjectStore  -
datanucleus.autoStartMechanismMode is set to unsupported value null .
Setting it to value: ignored
Automatically selecting local only mode for query
1361460 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.hive.ql.Driver  - Hive-on-MR is deprecated in Hive 2 and
may not be available in the future versions. Consider using a different
execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20231007072328_132fb4ba-e5bd-40af-8b4c-84998b9da947
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
1363677 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.metrics2.impl.MetricsSystemImpl  - JobTracker metrics
system already initialized!
1363722 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.mapreduce.JobResourceUploader  - Hadoop command-line
option parsing not performed. Implement the Tool interface and execute your
application with ToolRunner to remedy this.
1365196 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.fs.FileUtil  - Command 'ln -s /usr/local/hadoop-
3.3.1/tmp/mapred/local/job_local1065285107_0001_dc86f705-a9c4-4b66-84e2-
276eb725b6b7/libjars /root/libjars/*' failed 1 with: ln: failed to create
symbolic link ‘/root/libjars/*’: No such file or directory

1365196 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN
org.apache.hadoop.mapred.LocalDistributedCacheManager  - Failed to create
symlink: /usr/local/hadoop-
3.3.1/tmp/mapred/local/job_local1065285107_0001_dc86f705-a9c4-4b66-84e2-
276eb725b6b7/libjars <- /root/libjars/*
Job running in-process (local Hadoop)
1371900 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN  mapreduce.Counters
- Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use
org.apache.hadoop.mapreduce.TaskCounter instead
2023-10-07 07:23:50,513 Stage-1 map = 0%,  reduce = 0%
1373201 [LocalJobRunner Map Task Executor #0] WARN
org.apache.hadoop.hive.serde2.objectinspector.StandardStructObjectInspector
- Trying to access 5 fields inside a list of 6 elements: [Long, 10, 10, 0,
48 4c 4c a0 01 01 83 c5 91 8e 03, null]
1373202 [LocalJobRunner Map Task Executor #0] WARN
org.apache.hadoop.hive.serde2.objectinspector.StandardStructObjectInspector
- ignoring similar errors.
1373202 [LocalJobRunner Map Task Executor #0] WARN
org.apache.hadoop.hive.serde2.objectinspector.StandardStructObjectInspector
- Trying to access 6 fields inside a list of 7 elements: [String, 6, 6, 1,
0, 48 4c 4c a0 01 01 c1 ae df ef 05, null]
```

```
1373202 [LocalJobRunner Map Task Executor #0] WARN org.apache.hadoop.hive.serde2.objectinspector.StandardStructObjectInspector - ignoring similar errors.
1373202 [LocalJobRunner Map Task Executor #0] WARN org.apache.hadoop.hive.serde2.objectinspector.StandardStructObjectInspector - Trying to access 5 fields inside a list of 6 elements: [Double, 90.0, 90.0, 0, 48 4c 4c a0 01 01 82 9b 88 e1 06, null]
1373202 [LocalJobRunner Map Task Executor #0] WARN org.apache.hadoop.hive.serde2.objectinspector.StandardStructObjectInspector - ignoring similar errors.
1373437 [pool-19-thread-1] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl  - JobTracker metrics system already initialized!
2023-10-07 07:23:52,581 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1065285107_0001
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://master:9864/user/hive/warehouse/myname.db/students1/.hive-staging_hive_2023-10-07_07-23-29_497_8368754191379137565-1/-ext-10000
Loading data to table myname.students1
1374415 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN org.apache.hadoop.hive.metastore.ObjectStore  - datanucleus.autoStartMechanismMode is set to unsupported value null . Setting it to value: ignored
1375519 [c1fc6cdd-cfa6-4959-963d-ac46ea866976 main] WARN org.apache.hadoop.hive.metastore.ObjectStore  - datanucleus.autoStartMechanismMode is set to unsupported value null . Setting it to value: ignored
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 659769448 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 27.088 seconds
hive>
```

查看表信息，可查记录数：

```
hive> desc formatted students1;
OK
# col_name              data_type               comment
id                      int
name                    string
score                   double
classes                 string

# Detailed Table Information
Database:               myname
OwnerType:              USER
Owner:                  root
CreateTime:             Sat Oct 07 04:56:20 EDT 2023
LastAccessTime:         UNKNOWN
Retention:              0
```

```
Location:
hdfs://master:9864/user/hive/warehouse/myname.db/students1
Table Type:             MANAGED_TABLE
Table Parameters:
        numFiles                2
        spark.sql.create.version        3.1.3
        spark.sql.sources.schema.numParts       1
        spark.sql.sources.schema.part.0 {\"type\":\"struct\",\"fields\":
[{\"name\":\"id\",\"type\":\"integer\",\"nullable\":true,\"metadata\":{}},
{\"name\":\"name\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},
{\"name\":\"score\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},
{\"name\":\"classes\",\"type\":\"string\",\"nullable\":true,\"metadata\":
{}}]}
        totalSize               44
        transient_lastDdlTime   1696678532

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        serialization.format    1
Time taken: 0.127 seconds, Fetched: 33 row(s)
hive>
```

由结果看出：numFiles： 2 表示已录入两条。

版本历史

- Ver1.1-20221007
  - 更新脚本sqlcontent为spark
  - 增加随堂练习
  - 增加实训作业
- Ver1.2-20221011
  - 先更新/usr/local/spark/conf/hive-site.xml为链接方式，后删除了软链接,因为spark 和hive有着天然的集成关系
- Ver1.3-20221015
  - 增加随堂和实训的视频学习
- Ver1.4-20221016
  - 增加理论视频学习
- Ver1.5-20221030
  - 增加理论视频学习2
- Ver1.7-20230828
  - 增加markdown版本