Spark Schema

实验目的

掌握spark schema \RDD\Spark SQL的区别

实验内容

一、RDD广播变量、累加器、持久化、

1、Broadcast广播变量

共享变量（Shared variable）可用于节省内存与运行时间，提高并行处理的执行效率，共享变量包括 - Broadcast–广播变量 - accumulator–累加器

创建 kvFruit

```
kvFruit = sc.parallelize([(1,"apple"),(2,"orange"),(3,"banana"),(4,"grape")])
```

创建fruitMap字典

```
fruitMap = kvFruit.collectAsMap()
print("dict:"+str(fruitMap))
```

```
dict:{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

创建fruitIDs

```
fruitIDs = sc.parallelize([2,4,1,3])
print("fruitIDs:"+str(fruitIDs.collect()))
```

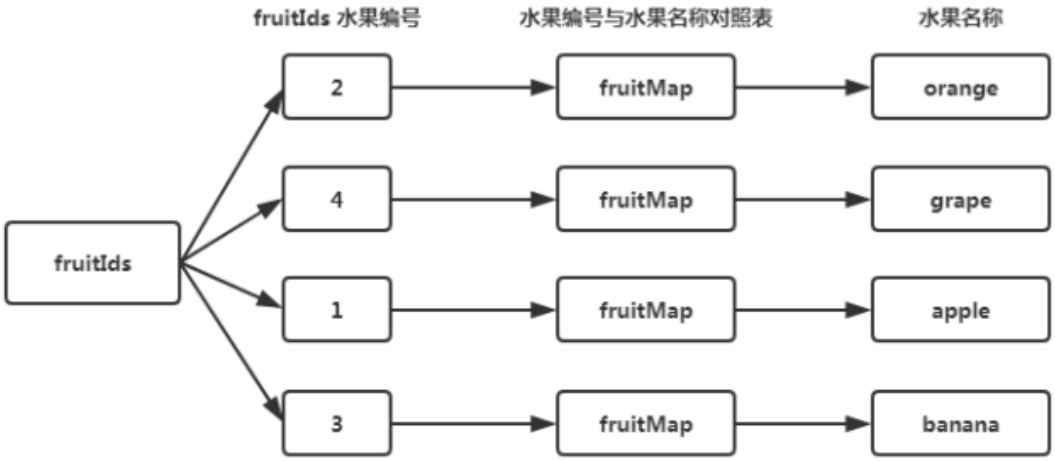
```
fruitIDs:[2, 4, 1, 3]
```

使用fruitMap字典进行切换

```
fruitNames = fruitIDs.map(lambda x : fruitMap[x]).collect()
print("fruitNames:"+str(fruitNames))
```

```
fruitNames:['orange', 'grape', 'apple', 'banana']
```

以上的例子执行起来虽然没问题，但是在并行处理中每执行一次转换都必须将 fruitIDs 与 fruitMap 传到 Worker 节点，才能够执行转换。如果字典 fruitMap（对照表）很大，而且需要转换的 fruitIDs 水果编号 RDD 也很大，就会消耗很多内存与时间



为了解决这个问题，可以使用 Broadcast 广播变量。
Broadcast 广播变量的使用规则如下：

- 可以使用 SparkContext.broadcast([初始值]) 创建。
- 使用.value 的方法来读取广播变量的值
- Broadcast 广播变量被创建后不能修改
  下列使用 Broadcast 广播变量的例子与之前的例子类似，不同之处是使用 sc.broadcast 传入 fruitMap 作为参数，创建 bcFruitMap 广播变量，使用 bcFruitMap.value(x) 广播变量转换为 fruitNames 水果名称

创建 kvFruit

```
kvFruit = sc.parallelize([(1,"apple"),(2,"orange"),(3,"banana"),(4,"grape")])
```

创建 fruitMap 字典

使用 collectAsMap 创建 fruitMap 字典（水果编号与名称对照表）

```
fruitMap=kvFruit.collectAsMap()
print("dict:"+str(fruitMap))
```

```
dict:{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

将 fruitMap 字典转换为 bcFruitMap 广播变量

使用 sc.broadcast 传入 fruitMap 参数，创建 bcFruitMap 广播变量

```
bcFruitMap = sc.broadcast(fruitMap)
print("dict:"+str(bcFruitMap))
```

```
dict:<pyspark.broadcast.Broadcast object at 0x7f4684343ba8>
```

创建 fruitIDs

```
fruitIDs = sc.parallelize([2,4,3,1])
print("fruit:"+str(fruitIDs.collect()))
```

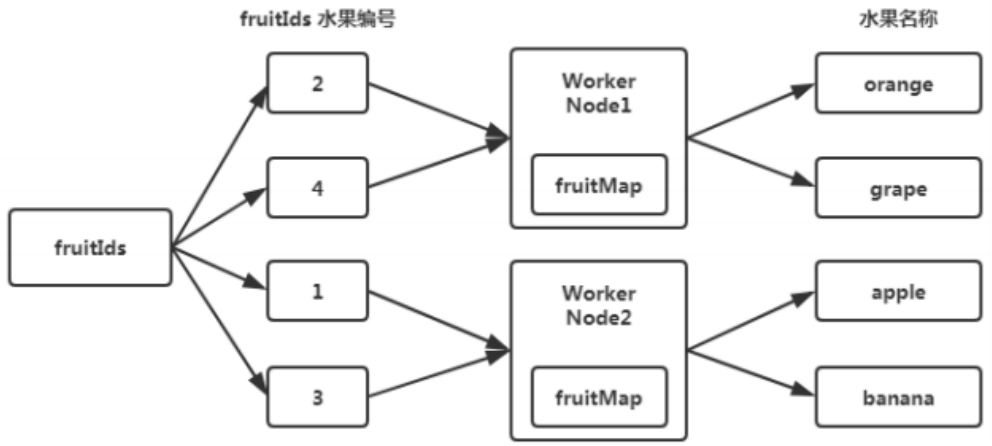
```
fruit:[2, 4, 3, 1]
```

使用 bcFruitMap.value[x] 广播变量将 fruitIDs 转换为 fruitNames

```
fruitNames = fruitIDs.map(lambda x:bcFruitMap.value[x]).collect()
print("fruitNames:"+str(fruitNames))
```

```
fruitNames:['orange', 'grape', 'banana', 'apple']
```

执行结果与之前的方法相同，在并行处理中 bcFruitMap 广播变量会传送到 Worker Node 机器，并且存储在内存中，后续在此 Worker Node 都可以使用这个 bcFruitMap 广播变量执行转换，这样就可以节省很多内存与传送时间。



2.accumulator累加器

计算总和是 MapReduce 常用的运算。为了方便并行处理，Spark 特别提供了 accumulator 累加器共享变量（Shared variable），使用规则如下：- accumulator 累加器可以使用 SparkContext.accumulator([初始值]) 来创建。- 使用.add() 进行累加 - 在 task 中，例如 foreach 循环中，不能读取累加器的值 - 只有驱动程序，也就是循环外，才可以使用.value 来读取累加器的值

```
intRDD = sc.parallelize([3,1,2,5,5])
```

创建 total 累加器，初始值使用 0.0，所以是 Double 的类型

```
total = sc.accumulator(0.0)
```

创建 num 累加器，初始值使用 0，所以是 Int 类型

```
num = sc.accumulator(0)
```

使用 foreach 传入参数 i，针对每一项数据执行，total 累加 intRDD 元素的值、num 累加 intRDD 元素的数量

```
intRDD.foreach(lambda i:[total.add(i),num.add(1)])
```

计算平均 = 求和/计数，并显示总和、数量

```
avg = total.value/num.value
print("total="+str(total.value)+"\tnum="+str(num.value)+"\tavg="+str(avg))
```

```
total=16.0  num=5   avg=3.2
```

3.RDD Persistence持久化

Spark RDD 持久化机制可以用于将需要重复运算的 RDD 存储在内存中，以便大幅提升运算效率。
Spark RDD 持久化使用方法如下：- RDD.persist(存储的等级)——可以指定存储等级，默认是MEMORY_ONLY，也就是存储在内存中 - RDD.unpersist()——取消持久化

创建 intRddMemory

```
intRddMemory = sc.parallelize([3,1,2,5,5])
```

使用RDD.persist()将intRddMemory()进行持久化

```
intRddMemory.persist()
```

```
ParallelCollectionRDD[2] at parallelize at PythonRDD.scala:195
```

查看是否已经缓存

```
intRddMemory.is_cached
```

```
True
```

```
intRddMemory.unpersist()
```

```
ParallelCollectionRDD[2] at parallelize at PythonRDD.scala:195
```

二、 PysparkSQL 、RDD、DataFrame比较

1 创建RDD

读取HDFS上的buyer_favorite文件，创建RDD并查看文件内数据行数

```
RawUserRDD = sc.textFile("/input/wordcount/buyer_favorite")
RawUserRDD.count()
```

```
30
```

查看前5行数据

```
RawUserRDD.take(5)
```

```
['10181\t1000481\t2010-04-04 16:54:31',
 '20001\t1001597\t2010-04-07 15:07:52',
 '20001\t1001560\t2010-04-07 15:08:27',
 '20042\t1001368\t2010-04-08 08:20:30',
 '20067\t1002061\t2010-04-08 16:45:33']
```

以 Tab 为分隔符，获取前 5 行的每个字段

```
userRDD = RawUserRDD.map(lambda line:line.split("\t"))
userRDD.take(5)
```

```
[['10181', '1000481', '2010-04-04 16:54:31'],
 ['20001', '1001597', '2010-04-07 15:07:52'],
 ['20001', '1001560', '2010-04-07 15:08:27'],
 ['20042', '1001368', '2010-04-08 08:20:30'],
 ['20067', '1002061', '2010-04-08 16:45:33']]
```

2.创建DataFrame

通过 userRDD 创建 DataFrame, 导入 Row 模块，定义 DataFrames 的每一个字段名与数据类型

```
from pyspark.sql import Row
user_Rows = userRDD.map(lambda
p:Row(buyer_id=int(p[0]),good_id=int(p[1]),dt=p[2]))
user_Rows.take(5)
```

```
[Row(buyer_id=10181, dt='2010-04-04 16:54:31', good_id=1000481),
 Row(buyer_id=20001, dt='2010-04-07 15:07:52', good_id=1001597),
 Row(buyer_id=20001, dt='2010-04-07 15:08:27', good_id=1001560),
 Row(buyer_id=20042, dt='2010-04-08 08:20:30', good_id=1001368),
 Row(buyer_id=20067, dt='2010-04-08 16:45:33', good_id=1002061)]
```

创建了 user_Rows 之后，使用 sqlContext.createDataFrame() 方法传入 user_Rows 数据，创建 DataFrame，然后使用 printSchema() 方法查看 DataFrames 的 Schema

```
user_df = spark.createDataFrame(user_Rows)
user_df.printSchema()
```

```
root
 |-- buyer_id: long (nullable = true)
 |-- dt: string (nullable = true)
 |-- good_id: long (nullable = true)
```

```
user_df.show(5)
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   10181|2010-04-04 16:54:31|1000481|
|   20001|2010-04-07 15:07:52|1001597|
|   20001|2010-04-07 15:08:27|1001560|
|   20042|2010-04-08 08:20:30|1001368|
|   20067|2010-04-08 16:45:33|1002061|
+--------+-------------------+-------+
only showing top 5 rows
```

也可以使用.alias() 方法来为 DadaFrame 创建别名，例如 user_df.alias(“df”)，后续我们就可以使用 df 这个别名执行命令

```
df = user_df.alias("df")
df.show(5)
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   10181|2010-04-04 16:54:31|1000481|
|   20001|2010-04-07 15:07:52|1001597|
|   20001|2010-04-07 15:08:27|1001560|
|   20042|2010-04-08 08:20:30|1001368|
|   20067|2010-04-08 16:45:33|1002061|
+--------+-------------------+-------+
only showing top 5 rows
```

3.创建Spark SQL

使用 registerDataFrameAsTable 方法将 DataFrame 转成 table,不可以使用sparkSession

```
sqlContext.registerDataFrameAsTable(df,"buyer_table")
```

使用 sqlContext.sql() 输入 sql 语句，使用 select 关键字查询文件内容行数，并使用 from 关键字指定要查询的表，最后使用 show() 方法显示查询结果

```
sqlContext.sql("select count(*) as counts from buyer_table").show()
```

```
+------+
|counts|
+------+
|    30|
+------+
```

为避免输入的 sql 语句过长，我们可以使用三个引号 “””，来将 sql 拆分成多行

```
sqlContext.sql("""
select count(*) as counts
from buyer_table
""").show()
```

```
+------+
|counts|
+------+
|    30|
+------+
```

4.查询部分字段

4.1 使用RDD查询部分数据

当我们使用 RDD 查询部分字段时，因为没有 Schema，未定义字段名，所以只能指定位置，这里我们查询 buyer_id、good_ids 和 dt 字段

```
userRDDnew = userRDD.map(lambda x:(x[0],x[1],x[2]))
userRDDnew.take(5)
```

```
[('10181', '1000481', '2010-04-04 16:54:31'),
 ('20001', '1001597', '2010-04-07 15:07:52'),
 ('20001', '1001560', '2010-04-07 15:08:27'),
 ('20042', '1001368', '2010-04-08 08:20:30'),
 ('20067', '1002061', '2010-04-08 16:45:33')]
```

4.2 使用DataFrame查询部分数据

当使用 DataFrame 时，因为已经定义了 Schema，所以可以使用 select 方法输入字段名，使用 DataFrame 查询部分数据时，有 4 种语句，执行结果一样

4.2.1 select 字段名查询

```
user_df.select("buyer_id","good_id","dt").show(5)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   10181|1000481|2010-04-04 16:54:31|
|   20001|1001597|2010-04-07 15:07:52|
|   20001|1001560|2010-04-07 15:08:27|
|   20042|1001368|2010-04-08 08:20:30|
|   20067|1002061|2010-04-08 16:45:33|
+--------+-------+-------------------+
only showing top 5 rows
```

4.2.2 select(dataframe.字段名)查询

```
user_df.select(user_df.buyer_id,user_df.good_id,user_df.dt).show(5)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   10181|1000481|2010-04-04 16:54:31|
|   20001|1001597|2010-04-07 15:07:52|
|   20001|1001560|2010-04-07 15:08:27|
|   20042|1001368|2010-04-08 08:20:30|
|   20067|1002061|2010-04-08 16:45:33|
+--------+-------+-------------------+
only showing top 5 rows
```

4.2.3 select (df别名.字段名)查询

```
df.select(df.buyer_id,df.good_id,df.dt).show(5)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   10181|1000481|2010-04-04 16:54:31|
|   20001|1001597|2010-04-07 15:07:52|
|   20001|1001560|2010-04-07 15:08:27|
|   20042|1001368|2010-04-08 08:20:30|
|   20067|1002061|2010-04-08 16:45:33|
+--------+-------+-------------------+
only showing top 5 rows
```

4.3.4 使用[] 查询部分数据

```
df[df['buyer_id'],df['good_id'],df['dt']].show(5)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   10181|1000481|2010-04-04 16:54:31|
|   20001|1001597|2010-04-07 15:07:52|
|   20001|1001560|2010-04-07 15:08:27|
|   20042|1001368|2010-04-08 08:20:30|
|   20067|1002061|2010-04-08 16:45:33|
+--------+-------+-------------------+
only showing top 5 rows
```

4.3 使用Spark SQL查询部分数据

```
sqlContext.sql("select buyer_id,good_id,dt from buyer_table").show(5)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   10181|1000481|2010-04-04 16:54:31|
|   20001|1001597|2010-04-07 15:07:52|
|   20001|1001560|2010-04-07 15:08:27|
|   20042|1001368|2010-04-08 08:20:30|
|   20067|1002061|2010-04-08 16:45:33|
+--------+-------+-------------------+
only showing top 5 rows
```

5.增加计算问题

当我们查询数据时，有些字段需要经过计算，现在我们使用 RDD、DataFrame、Spark SQL 三种方式，对 buyer_id 字段进行增加 100000 操作

5.1 使用RDD增加计算字段

```
userRDDnew = userRDD.map(lambda x:(x[0],x[1],x[2],int(x[0])+100000))
userRDDnew.take(5)
```

```
[('10181', '1000481', '2010-04-04 16:54:31', 110181),
 ('20001', '1001597', '2010-04-07 15:07:52', 120001),
 ('20001', '1001560', '2010-04-07 15:08:27', 120001),
 ('20042', '1001368', '2010-04-08 08:20:30', 120042),
 ('20067', '1002061', '2010-04-08 16:45:33', 120067)]
```

```
df.select("buyer_id","good_id","dt",df.buyer_id+100000).show(5)
```

```
+--------+-------+-------------------+-------------------+
|buyer_id|good_id|                 dt|(buyer_id + 100000)|
+--------+-------+-------------------+-------------------+
|   10181|1000481|2010-04-04 16:54:31|             110181|
|   20001|1001597|2010-04-07 15:07:52|             120001|
|   20001|1001560|2010-04-07 15:08:27|             120001|
|   20042|1001368|2010-04-08 08:20:30|             120042|
|   20067|1002061|2010-04-08 16:45:33|             120067|
+--------+-------+-------------------+-------------------+
only showing top 5 rows
```

还可以使用 alias() 方法为计算字段取一个别名，这里我们取名为 new_buyer_id

```
df.select("buyer_id","good_id","dt",
(df.buyer_id+100000).alias("new_buyer_id")).show(5)
```

```
+--------+-------+-------------------+------------+
|buyer_id|good_id|                 dt|new_buyer_id|
+--------+-------+-------------------+------------+
|   10181|1000481|2010-04-04 16:54:31|      110181|
|   20001|1001597|2010-04-07 15:07:52|      120001|
|   20001|1001560|2010-04-07 15:08:27|      120001|
|   20042|1001368|2010-04-08 08:20:30|      120042|
|   20067|1002061|2010-04-08 16:45:33|      120067|
+--------+-------+-------------------+------------+
only showing top 5 rows
```

5.3 使用Spark SQL增加计算字段

```
sqlContext.sql("""
select buyer_id,good_id,dt,buyer_id+100000
from buyer_table
""").show(5)
```

```
+--------+-------+-------------------+-----------------------------------+
|buyer_id|good_id|                 dt|(buyer_id + CAST(100000 AS BIGINT))|
+--------+-------+-------------------+-----------------------------------+
|   10181|1000481|2010-04-04 16:54:31|                             110181|
|   20001|1001597|2010-04-07 15:07:52|                             120001|
|   20001|1001560|2010-04-07 15:08:27|                             120001|
|   20042|1001368|2010-04-08 08:20:30|                             120042|
|   20067|1002061|2010-04-08 16:45:33|                             120067|
+--------+-------+-------------------+-----------------------------------+
only showing top 5 rows
```

6 筛选数据

6.1 使用RDD筛选数据

```
userRDD.filter(lambda r:r[0]=='20056').take(5)
```

```
[['20056', '1003289', '2010-04-12 10:50:55'],
 ['20056', '1003290', '2010-04-12 11:57:35'],
 ['20056', '1003292', '2010-04-12 12:05:29'],
 ['20056', '1002420', '2010-04-15 11:24:49'],
 ['20056', '1003066', '2010-04-15 11:43:01']]
```

6.2 使用DataFrame筛选数据

使用filter()方法筛选数据

```
df.filter("buyer_id=20056").show()
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   20056|2010-04-12 10:50:55|1003289|
|   20056|2010-04-12 11:57:35|1003290|
|   20056|2010-04-12 12:05:29|1003292|
|   20056|2010-04-15 11:24:49|1002420|
|   20056|2010-04-15 11:43:01|1003066|
|   20056|2010-04-15 11:43:06|1003055|
|   20056|2010-04-15 11:45:24|1010183|
|   20056|2010-04-15 11:45:49|1002422|
|   20056|2010-04-15 11:45:54|1003100|
|   20056|2010-04-15 11:45:57|1003094|
|   20056|2010-04-15 11:46:04|1003064|
|   20056|2010-04-15 16:15:20|1010178|
+--------+-------------------+-------+
```

使用（dataframe. 字段名）筛选

```
df.filter(df.buyer_id=="20056").show()
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   20056|2010-04-12 10:50:55|1003289|
|   20056|2010-04-12 11:57:35|1003290|
|   20056|2010-04-12 12:05:29|1003292|
```

```
|   20056|2010-04-15 11:24:49|1002420|
|   20056|2010-04-15 11:43:01|1003066|
|   20056|2010-04-15 11:43:06|1003055|
|   20056|2010-04-15 11:45:24|1010183|
|   20056|2010-04-15 11:45:49|1002422|
|   20056|2010-04-15 11:45:54|1003100|
|   20056|2010-04-15 11:45:57|1003094|
|   20056|2010-04-15 11:46:04|1003064|
|   20056|2010-04-15 16:15:20|1010178|
+--------+-------------------+-------+
```

使用中括号 [] 筛选

```
df.filter(df["buyer_id"]=="20056").show()
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   20056|2010-04-12 10:50:55|1003289|
|   20056|2010-04-12 11:57:35|1003290|
|   20056|2010-04-12 12:05:29|1003292|
|   20056|2010-04-15 11:24:49|1002420|
|   20056|2010-04-15 11:43:01|1003066|
|   20056|2010-04-15 11:43:06|1003055|
|   20056|2010-04-15 11:45:24|1010183|
|   20056|2010-04-15 11:45:49|1002422|
|   20056|2010-04-15 11:45:54|1003100|
|   20056|2010-04-15 11:45:57|1003094|
|   20056|2010-04-15 11:46:04|1003064|
|   20056|2010-04-15 16:15:20|1010178|
+--------+-------------------+-------+
```

6.3 使用Spark SQL筛选数据

```
sqlContext.sql("""
select
*
from
buyer_table
where buyer_id = "20056"
""").show(5)
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   20056|2010-04-12 10:50:55|1003289|
|   20056|2010-04-12 11:57:35|1003290|
|   20056|2010-04-12 12:05:29|1003292|
|   20056|2010-04-15 11:24:49|1002420|
|   20056|2010-04-15 11:43:01|1003066|
+--------+-------------------+-------+
only showing top 5 rows
```

7 数据排序

7.1 RDD 按单个字段给数据排序

- 在 RDD 中可以使用 takeOrdered(num,key=None) 方法对数据进行排序：
- 参数解释：
- num：要显示的项数
- key：使用 lambda 语句设置要排序的字段
  使用 RDD 按 buyer_id 的升序给数据排序

```
userRDD.takeOrdered(10,key=lambda x:int(x[0]))
```

```
[['10181', '1000481', '2010-04-04 16:54:31'],
 ['20001', '1001597', '2010-04-07 15:07:52'],
 ['20001', '1001560', '2010-04-07 15:08:27'],
 ['20042', '1001368', '2010-04-08 08:20:30'],
 ['20054', '1002420', '2010-04-14 15:24:12'],
 ['20054', '1010675', '2010-04-14 15:23:53'],
 ['20054', '1002429', '2010-04-14 17:52:45'],
 ['20054', '1003326', '2010-04-20 12:54:44'],
 ['20054', '1003103', '2010-04-15 16:40:14'],
 ['20054', '1003100', '2010-04-15 16:40:16']]
```

使用 RDD 按 buyer_id 的降序给数据排序

```
userRDD.takeOrdered(10,key=lambda x:-1*int(x[0]))
```

```
[['20076', '1002427', '2010-04-14 19:35:39'],
 ['20076', '1003101', '2010-04-15 16:37:27'],
 ['20076', '1003103', '2010-04-15 16:37:05'],
 ['20076', '1003100', '2010-04-15 16:37:18'],
 ['20076', '1003066', '2010-04-15 16:37:31'],
 ['20067', '1002061', '2010-04-08 16:45:33'],
 ['20064', '1002422', '2010-04-15 11:35:54'],
 ['20056', '1003289', '2010-04-12 10:50:55'],
 ['20056', '1003290', '2010-04-12 11:57:35'],
 ['20056', '1003292', '2010-04-12 12:05:29']]
```

7.2 DataFrame按单个字段给数据排序

使用.orderBy() 方法来进行排序，因为默认为升序，所以我们不需要注明 ascending

```
user_df.select('buyer_id','good_id','dt').orderBy('buyer_id').show(10)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   10181|1000481|2010-04-04 16:54:31|
|   20001|1001560|2010-04-07 15:08:27|
|   20001|1001597|2010-04-07 15:07:52|
|   20042|1001368|2010-04-08 08:20:30|
|   20054|1003103|2010-04-15 16:40:14|
|   20054|1010675|2010-04-14 15:23:53|
|   20054|1003326|2010-04-20 12:54:44|
|   20054|1002420|2010-04-14 15:24:12|
|   20054|1002429|2010-04-14 17:52:45|
|   20054|1003100|2010-04-15 16:40:16|
+--------+-------+-------------------+
only showing top 10 rows
```

使用 DataFrame 按 buyer_id 的降序给数据排序，我们需要使用.desc() 方法或者指定 ascending=0

```
user_df.select("buyer_id","good_id","dt").orderBy(user_df.buyer_id.desc()).show(
10)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   20076|1003066|2010-04-15 16:37:31|
|   20076|1002427|2010-04-14 19:35:39|
|   20076|1003100|2010-04-15 16:37:18|
|   20076|1003101|2010-04-15 16:37:27|
|   20076|1003103|2010-04-15 16:37:05|
|   20067|1002061|2010-04-08 16:45:33|
|   20064|1002422|2010-04-15 11:35:54|
|   20056|1003290|2010-04-12 11:57:35|
|   20056|1003066|2010-04-15 11:43:01|
|   20056|1003292|2010-04-12 12:05:29|
```

```
+--------+-------+------------------+
only showing top 10 rows
```

```
user_df.select('buyer_id','good_id','dt').orderBy('buyer_id',ascending=0).show(10)
```

```
+--------+-------+------------------+
|buyer_id|good_id|                dt|
+--------+-------+------------------+
|   20076|1003066|2010-04-15 16:37:31|
|   20076|1002427|2010-04-14 19:35:39|
|   20076|1003100|2010-04-15 16:37:18|
|   20076|1003101|2010-04-15 16:37:27|
|   20076|1003103|2010-04-15 16:37:05|
|   20067|1002061|2010-04-08 16:45:33|
|   20064|1002422|2010-04-15 11:35:54|
|   20056|1003290|2010-04-12 11:57:35|
|   20056|1003066|2010-04-15 11:43:01|
|   20056|1003292|2010-04-12 12:05:29|
+--------+-------+------------------+
only showing top 10 rows
```

7.3 Spark sql按单个字段给数据排序

使用spark sql按buyer_id的升序给数据排序

```
sqlContext.sql("""
select
buyer_id,
good_id,
dt
from
buyer_table
order by buyer_id
""").show(10)
```

```
+--------+-------+------------------+
|buyer_id|good_id|                dt|
+--------+-------+------------------+
|   10181|1000481|2010-04-04 16:54:31|
|   20001|1001560|2010-04-07 15:08:27|
|   20001|1001597|2010-04-07 15:07:52|
|   20042|1001368|2010-04-08 08:20:30|
|   20054|1003103|2010-04-15 16:40:14|
|   20054|1010675|2010-04-14 15:23:53|
|   20054|1003326|2010-04-20 12:54:44|
|   20054|1002420|2010-04-14 15:24:12|
|   20054|1002429|2010-04-14 17:52:45|
|   20054|1003100|2010-04-15 16:40:16|
+--------+-------+------------------+
only showing top 10 rows
```

使用 Spark SQL 按 buyer_id 的降序给数据排序

```
sqlContext.sql("""
select
buyer_id,
good_id,
dt
from
buyer_table
order by buyer_id desc
""").show(10)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   20076|1003066|2010-04-15 16:37:31|
|   20076|1002427|2010-04-14 19:35:39|
|   20076|1003100|2010-04-15 16:37:18|
|   20076|1003101|2010-04-15 16:37:27|
|   20076|1003103|2010-04-15 16:37:05|
|   20067|1002061|2010-04-08 16:45:33|
|   20064|1002422|2010-04-15 11:35:54|
|   20056|1003290|2010-04-12 11:57:35|
|   20056|1003066|2010-04-15 11:43:01|
|   20056|1003292|2010-04-12 12:05:29|
+--------+-------+-------------------+
only showing top 10 rows
```

8 按多个字段给数据排序

8.1 RDD 按多个字段给数据排序

```
userRDD.takeOrdered(10,key=lambda x:(x[0],-int(x[1])))
```

```
[['10181', '1000481', '2010-04-04 16:54:31'],
 ['20001', '1001597', '2010-04-07 15:07:52'],
 ['20001', '1001560', '2010-04-07 15:08:27'],
 ['20042', '1001368', '2010-04-08 08:20:30'],
 ['20054', '1010675', '2010-04-14 15:23:53'],
 ['20054', '1003326', '2010-04-20 12:54:44'],
 ['20054', '1003103', '2010-04-15 16:40:14'],
 ['20054', '1003100', '2010-04-15 16:40:16'],
 ['20054', '1002429', '2010-04-14 17:52:45'],
 ['20054', '1002420', '2010-04-14 15:24:12']]
```

8.2 DataFrame 按多个字段给数据排序

使用 orderBy([“buyer_id”,“good_id”],ascending=[0,1]) 按多个字段给数据排序
参数解释：- 第一个置要排序的字段：[“buyer_id”,“good_id”] - 第二个数：设置排序字段的升序/降序：ascending=[1,0]，其中第一个 buyer_id 设置为 1，表示升序；第二个字段 good_id 设置为 0，表示降序

```
user_df.orderBy(["buyer_id","good_id"],ascending=[1,0]).show(10)
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   10181|2010-04-04 16:54:31|1000481|
|   20001|2010-04-07 15:07:52|1001597|
|   20001|2010-04-07 15:08:27|1001560|
|   20042|2010-04-08 08:20:30|1001368|
|   20054|2010-04-14 15:23:53|1010675|
|   20054|2010-04-20 12:54:44|1003326|
|   20054|2010-04-15 16:40:14|1003103|
|   20054|2010-04-15 16:40:16|1003100|
|   20054|2010-04-14 17:52:45|1002429|
|   20054|2010-04-14 15:24:12|1002420|
+--------+-------------------+-------+
only showing top 10 rows
```

使用.desc() 方法表示降序

```
user_df.orderBy(user_df.buyer_id,user_df.good_id.desc()).show(10)
```

```
+--------+-------------------+-------+
|buyer_id|                 dt|good_id|
+--------+-------------------+-------+
|   10181|2010-04-04 16:54:31|1000481|
|   20001|2010-04-07 15:07:52|1001597|
|   20001|2010-04-07 15:08:27|1001560|
|   20042|2010-04-08 08:20:30|1001368|
|   20054|2010-04-14 15:23:53|1010675|
|   20054|2010-04-20 12:54:44|1003326|
|   20054|2010-04-15 16:40:14|1003103|
|   20054|2010-04-15 16:40:16|1003100|
|   20054|2010-04-14 17:52:45|1002429|
|   20054|2010-04-14 15:24:12|1002420|
+--------+-------------------+-------+
only showing top 10 rows
```

8.3 Spark SQL 按多个字段给数据排序

```
sqlContext.sql("""
select
buyer_id,
good_id,
dt
from buyer_table
order by good_id desc,buyer_id
""").show(10)
```

```
+--------+-------+-------------------+
|buyer_id|good_id|                 dt|
+--------+-------+-------------------+
|   20054|1010675|2010-04-14 15:23:53|
|   20056|1010183|2010-04-15 11:45:24|
|   20056|1010178|2010-04-15 16:15:20|
|   20054|1003326|2010-04-20 12:54:44|
|   20056|1003292|2010-04-12 12:05:29|
|   20056|1003290|2010-04-12 11:57:35|
|   20056|1003289|2010-04-12 10:50:55|
|   20054|1003103|2010-04-15 16:40:14|
|   20076|1003103|2010-04-15 16:37:05|
|   20076|1003101|2010-04-15 16:37:27|
+--------+-------+-------------------+
only showing top 10 rows
```

9 查询不重复的数据

9.1 RDD 查询不重复数据

使用.distinct() 方法查询 buyer_id 不重复数据

```
userRDD.map(lambda x:x[0]).distinct().collect()
```

```
['20001',
 '20067',
 '20056',
 '20076',
 '20064',
 '10181',
 '20042',
 '20054',
 '20055']
```

查询 buyer_id 和 good_id 都不重复的数据，并限制查看 10 条

```
userRDD.map(lambda x:(x[0],x[1])).distinct().take(10)
```

```
[('20001', '1001597'),
 ('20056', '1003292'),
 ('20054', '1010675'),
 ('20056', '1002420'),
 ('20056', '1003066'),
 ('20056', '1010183'),
 ('20056', '1003100'),
 ('20056', '1003094'),
 ('20056', '1003064'),
 ('20056', '1010178')]
```

9.2 DataFrame 查询不重复数据

使用.distinct() 方法查询 buyer_id 不重复数据

```
user_df.select("buyer_id").distinct().show()
```

```
+--------+
|buyer_id|
+--------+
|   20064|
|   20056|
|   20042|
|   20001|
|   10181|
|   20067|
|   20055|
|   20076|
|   20054|
+--------+
```

查询 buyer_id 和 good_id 都不重复的数据，并限制查看 10 条

```
user_df.select("buyer_id","good_id").distinct().show(10)
```

```
+--------+-------+
|buyer_id|good_id|
+--------+-------+
|   20054|1003100|
|   20055|1001679|
|   20056|1003055|
|   20076|1003066|
|   20056|1002422|
|   20056|1002420|
|   20056|1003100|
|   20064|1002422|
|   20056|1003094|
|   20054|1003103|
+--------+-------+
```

```
only showing top 10 rows
```

9.3 Spark SQL 查询不重复数据

使用 distinct 关键字查询 buyer_id 不重复数据

```
sqlContext.sql("""
select
distinct buyer_id
from
buyer_table
""").show()
```

```
+--------+
|buyer_id|
+--------+
|   20064|
|   20056|
|   20042|
|   20001|
|   10181|
|   20067|
|   20055|
|   20076|
|   20054|
+--------+
```

使用 distinct 关键字查询 buyer_id 和 good_id 不重复数据，并限制查看 10 条

```
sqlContext.sql("""
select
distinct buyer_id,good_id
from
buyer_table
""").show(10)
```

```
+--------+-------+
|buyer_id|good_id|
+--------+-------+
|   20054|1003100|
|   20055|1001679|
|   20056|1003055|
|   20076|1003066|
|   20056|1002422|
|   20056|1002420|
|   20056|1003100|
|   20064|1002422|
|   20056|1003094|
|   20054|1003103|
```

```
+--------+-------+
only showing top 10 rows
```

10 分组统计数据

10.1 RDD 分组查询

按照 buyer_id 分组统计数据，我们需要用到 map/reduce，此方法可用作 wordcount

```
userRDD.map(lambda x:(x[0],1)).reduceByKey(lambda x,y:x+y).collect()
```

```
[('20001', 2),
 ('20067', 1),
 ('20056', 12),
 ('20076', 5),
 ('20064', 1),
 ('10181', 1),
 ('20042', 1),
 ('20054', 6),
 ('20055', 1)]
```

10.2 DataFrame 分组查询

使用 groupby() 方法和 count() 方法对 buyer_id 进行分组查询

```
user_df.select("buyer_id").groupby("buyer_id").count().show()
```

```
+--------+-----+
|buyer_id|count|
+--------+-----+
|   20064|    1|
|   20056|   12|
|   20042|    1|
|   20001|    2|
|   10181|    1|
|   20067|    1|
|   20055|    1|
|   20076|    5|
|   20054|    6|
+--------+-----+
```

10.3 Spark SQL 分组查询

```
sqlContext.sql("""
select
buyer_id,count(*) counts
from buyer_table
group by buyer_id
""").show()
```

```
+--------+------+
|buyer_id|counts|
+--------+------+
|   20064|     1|
|   20056|    12|
|   20042|     1|
|   20001|     2|
|   10181|     1|
|   20067|     1|
|   20055|     1|
|   20076|     5|
|   20054|     6|
+--------+------+
```

11 Join 连接数据

```
buyer_log = sc.textFile("file:/data/buyer_log")
buyer_log.take(5)
```

```
['462\t10262\t2010-03-26 19:55:10\t123.127.164.252\t1',
 '463\t20001\t2010-03-29 14:28:02\t221.208.129.117\t2',
 '464\t20001\t2010-03-29 14:28:02\t221.208.129.117\t1',
 '465\t20002\t2010-03-30 10:56:35\t222.44.94.235\t2',
 '466\t20002\t2010-03-30 10:56:35\t222.44.94.235\t1']
```

以 Tab 为分隔符，分隔每个字段

```
userRDD2=buyer_log.map(lambda line:line.split("\t"))
userRDD2.first()
```

```
['462', '10262', '2010-03-26 19:55:10', '123.127.164.252', '1']
```

```
from pyspark.sql import Row
user_Rows2 = userRDD2.map(lambda
p:Row(id=int(p[0]),buyer_id=int(p[1]),dt=p[2],ip=p[3],opt_type=p[4]))
user_Rows2.take(5)
```

```
[Row(buyer_id=10262, dt='2010-03-26 19:55:10', id=462, ip='123.127.164.252',
opt_type='1'),
 Row(buyer_id=20001, dt='2010-03-29 14:28:02', id=463, ip='221.208.129.117',
opt_type='2'),
 Row(buyer_id=20001, dt='2010-03-29 14:28:02', id=464, ip='221.208.129.117',
opt_type='1'),
 Row(buyer_id=20002, dt='2010-03-30 10:56:35', id=465, ip='222.44.94.235',
opt_type='2'),
 Row(buyer_id=20002, dt='2010-03-30 10:56:35', id=466, ip='222.44.94.235',
opt_type='1')]
```

创建了 user_Rows2 之后，使用 sqlContext.createDataFrame() 方法传入 user_Rows2 数据，创建 DataFrame，然后使用 printSchema() 方法查看 DataFrames 的 Schema

```
buyerlog_df = sqlContext.createDataFrame(user_Rows2)
buyerlog_df.printSchema()
```

```
root
 |-- buyer_id: long (nullable = true)
 |-- dt: string (nullable = true)
 |-- id: long (nullable = true)
 |-- ip: string (nullable = true)
 |-- opt_type: string (nullable = true)
```

11.1 DataFrame 联接

user_df 通过 buyer_id 左外联接 buyerlog_df，联接结果将会创建另外一个 DataFrame（joined_df）

```
joined_df = user_df.join(buyerlog_df,user_df.buyer_id ==
buyerlog_df.buyer_id,'left_outer')
joined_df.printSchema()
```

```
root
 |-- buyer_id: long (nullable = true)
 |-- dt: string (nullable = true)
 |-- good_id: long (nullable = true)
 |-- buyer_id: long (nullable = true)
 |-- dt: string (nullable = true)
 |-- id: long (nullable = true)
 |-- ip: string (nullable = true)
 |-- opt_type: string (nullable = true)
```

代码解释:
joined_df = user_df.join(buyerlog_df： user_df 联接 buyerlog_df 创建 joined_df
user_df.buyer_id == buyerlog_df.buyer_id： 设置联接条件
'left_outer'： 设置联接方式
joined_df.printSchema()： 打印 joined_df 的 Schema

```
joined_df.show(20)
```

```
+--------+-------------------+-------+--------+-------------------+----+--------
-----+--------+
|buyer_id|                 dt|good_id|buyer_id|                 dt|  id|
  ip|opt_type|
+--------+-------------------+-------+--------+-------------------+----+--------
-----+--------+
|   20064|2010-04-15 11:35:54|1002422|    null|               null|null|
null|    null|
|   20056|2010-04-12 10:50:55|1003289|    null|               null|null|
null|    null|
|   20056|2010-04-12 11:57:35|1003290|    null|               null|null|
null|    null|
|   20056|2010-04-12 12:05:29|1003292|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:24:49|1002420|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:43:01|1003066|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:43:06|1003055|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:45:24|1010183|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:45:49|1002422|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:45:54|1003100|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:45:57|1003094|    null|               null|null|
null|    null|
|   20056|2010-04-15 11:46:04|1003064|    null|               null|null|
null|    null|
|   20056|2010-04-15 16:15:20|1010178|    null|               null|null|
null|    null|
|   20042|2010-04-08 08:20:30|1001368|   20042|2010-04-08 08:14:10|
512|218.9.124.214|       2|
|   20042|2010-04-08 08:20:30|1001368|   20042|2010-04-08 08:14:11|
513|218.9.124.214|       1|
```

```
|   20042|2010-04-08 08:20:30|1001368|   20042|2010-04-08 08:25:38|
514|218.9.124.214|       5|
|   20042|2010-04-08 08:20:30|1001368|   20042|2010-04-08 08:26:30|
515|218.9.124.214|       4|
|   20042|2010-04-08 08:20:30|1001368|   20042|2010-04-08 08:31:28|
516|218.9.124.214|       1|
|   20042|2010-04-08 08:20:30|1001368|   20042|2010-04-08 08:31:43|
517|218.9.124.214|       5|
|   20042|2010-04-08 08:20:30|1001368|   20042|2010-04-08 08:46:09|
518|218.9.124.214|       1|
+--------+-------------------+-------+--------+-------------------+----+--------
-----+--------+
only showing top 20 rows
```

```
joined_df.filter("ip='123.127.164.252'").show()
```

```
+--------+-------------------+-------+--------+-------------------+---+---------
------+--------+
|buyer_id|                 dt|good_id|buyer_id|                 dt| id|
    ip|opt_type|
+--------+-------------------+-------+--------+-------------------+---+---------
------+--------+
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-03-31
16:48:43|481|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-01
17:35:05|482|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-02
10:34:20|483|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-06
13:39:37|490|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-07
10:02:08|502|123.127.164.252|       1|
+--------+-------------------+-------+--------+-------------------+---+---------
------+--------+
```

11.2 Spark SQL Join 操作

之前我们创建了 buyer_table 表，现在我们使用 registerDataFrameAsTable() 方法来将 buyerlog_df 转换为 buyerlog_table

```
sqlContext.registerDataFrameAsTable(buyerlog_df, "buyerlog_table")
```

```
sqlContext.sql("select count(*) as counts from buyerlog_table").show()
```

```
+------+
|counts|
+------+
|    62|
+------+
```

查看 buyerlog_table 有多少行记录

```
sqlContext.sql("select count(*) as counts from buyerlog_table").show()
```

```
+------+
|counts|
+------+
|    62|
+------+
```

使用 spark sql 将 buyer_table 和 buyerlog_table 进行左连接，并查询 ip 为“123.127.164.252“的数据

```
sqlContext.sql("""
select
b.*,l.*
from buyer_table b
left join buyerlog_table l on b.buyer_id = l.buyer_id
where l.ip='123.127.164.252'
""").show()
```

```
+--------+-------------------+-------+--------+-------------------+---+---------
------+--------+
|buyer_id|                 dt|good_id|buyer_id|                 dt| id|
    ip|opt_type|
+--------+-------------------+-------+--------+-------------------+---+---------
------+--------+
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-03-31
16:48:43|481|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-01
17:35:05|482|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-02
10:34:20|483|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-06
13:39:37|490|123.127.164.252|       1|
|   10181|2010-04-04 16:54:31|1000481|   10181|2010-04-07
10:02:08|502|123.127.164.252|       1|
+--------+-------------------+-------+--------+-------------------+---+---------
------+--------+
```