第3章Spark编程实训

CH03实训1：使用spark-shell进行词频统计

训练要点

1. 掌握RDD创建方法，读数数据文件和存储文件的方法。
2. 掌握flatMap操作、map操作、reduceByKey操作、filter操作方法。

需求说明

数据文件words.txt如下文所示，文件中包含了多行句子，现在要求对文档中的单词计数，并把单词计数超过3的结果存储到HDFS上

---

words.txt

WHat is going on there?

I talked to John on email. We talked about some computer stuff that's it.

I went bike riding in the rain, it was not that cold.

We went to the museum in SF yesterday it was $3 to get in and they had free food. At the same time was a SF Giants game, when we got done we had to take the train with all the Giants fans, they are 1/2 drunk.

可以在linux下使用wget获取该文件：

wget http://bigdata.hddly.cn/b46488/file/chap3/words.txt

实现思路及步骤

1. 通过textFile的方法读取数据。
2. 通过flatMap将字符串切分成单词。
3. 通过map将单词转化为(单词，1)的形式。
4. 通过reduceByKey将同一一个单词的所有值相加。
5. 通过filter将单词计数大于3的结果过滤出来。
6. 通过saveAsTextFile将结果写人到HDFS

作业要求

1. 将测试文件words.txt上传到hdfs上的/user/myname/目录下,并截图(myname要改为自已名字全拼)
2. 通过crt进入spark-shell命令窗口，执行词频分析脚本，其中输出结果文件存于/user/myname/output_wordcount，并截图运行过程；
3. 提交hdfs上生成的结果文件: /user/myname/output_wordcount 的内容截图，图中需显示完整的文件路径。

实现参考

环境准备参考

```
//启动hadoop集群
start-all.sh
//启动spark集群
cd /usr/local/spark/sbin
./start-all.sh
```

使用jps检查master主机进程

```
[root@master sbin]# jps
2421 SecondaryNameNode
2134 NameNode
2710 ResourceManager
3150 Master
3902 Jps
[root@master sbin]
```

数据准备

在slave1下执行

```
cd /root/spark
wget https://biglab.site//b5951Ospark/file/spark_t_data.tar.gz
tar -xzvf ./spark_t_data.tar.gz
hdfs dfs -rm /user/myname/words.txt
hdfs dfs -rm -r /user/myname/output_wordcount
hdfs dfs -put ./spark_t_data/words.txt /user/myname/
```

在spark-shell上执行

```
spark-shell
```

```
val worddata=sc.textFile("/user/myname/words.txt").flatMap(x=>x.split(" ")).map(x=>(x,1)).reduceByKey((x,y)=>x+y)

val worddata_3=worddata.filter(x=>x._2>=2).map(x=>x._1)

worddata_3.repartition(1).saveAsTextFile("/user/myname/output_wordcount")
```

spark-shell预期输出：

```
scala> val worddata=sc.textFile("/user/myname/words.txt").flatMap(x=>x.split(" ")).map(x=>(x,1)).reduceByKey((x,y)=>x+y)
worddata: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[16] at reduceByKey at <console>:24

scala>

scala> val worddata_3=worddata.filter(x=>x._2>3).map(x=>x._1)
worddata_3: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[18] at map at <console>:25

scala>

scala> worddata_3.repartition(1).saveAsTextFile("/user/myname/output_wordcount")

scala>
```

在hdfs上观察

```
/user/myname/output_wordcount
```

学习视频

学习视频

CH03实训2：通过spark编程统计某月份的客户总消费金额

训练要点

1. 掌握数据读取和存储的方法。
2. 掌握RDD基本操作

需求说明

大数据和人工智能技术的应用，使得我国广大科技工作者能够把握大势、抢占先机、直面问题、迎难而上，瞄准世界科技前沿，引领科技发展方向。

本任务中，某互联网企业，创建了线上购物平台，开拓了新的商品销售渠道。现有一份某电商2020年12月份的订单数据文件online retail.csv，记录了每位顾客每笔订单的购物情况，包含3个数据字段，字段说明如表3．7所示。因为该电商准备给重要的客户发放购物津贴作为福利回馈，提高顾客满意度，所以需要统计每位客户的总消费金额，并筛选出消费金额排在前50名的客户。

某电商的订单数据字段说明：

字段名称	说明
Invoice	订单编号
Price	订单价格（单位：元）
Customer ID	客户编号

实现思路及步骤

1. 读取数据并创建RDD.
2. 通过map()方法分割数据，选择客户编号和订单价格字段组成键值对数据.
3. 使用reduceByKey()方法计算每位客户的总消费金额.
4. 使用sortBy()方法对每位客户的总消费金额进行降序排序，取出前50条数据.

作业要求

提交源码和脚本运行过程截图

实现参考

数据准备

在slave1上运行：

```
cd /root/spark
wget https://biglab.site//b59510spark/file/spark_t_data.tar.gz
tar -xzvf ./spark_t_data.tar.gz
hdfs dfs -put ./spark_t_data/online_retail.txt /user/myname/
```

spark-shell脚本

```
//在Spark-shell中 读取上传到HDFS上的数据
val input = sc.textFile("/user/myname/online_retail.txt")

//去除csv数据文件的第一行
val cutinput = input.mapPartitionsWithIndex((ix, it) => {
   if (ix == 0) it.drop(1)
   it
})
cutinput.collect

val re = cutinput.map(line => {val data = line.split(",");(data(0),data(1).toDouble)}).reduceByKey(_+_)

val re_sort = re.filter(x=> !(x._1=="")).sortBy(x=>x._2,false)

re_sort.take(50)

```

预期输出

```
scala> //在Spark-shell中 读取上传到HDFS上的数据

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

scala>

scala> //去除csv数据文件的第一行

scala> val cutinput = input.mapPartitionsWithIndex((ix, it) => {
     |    if (ix == 0) it.drop(1)
     |    it
     | })
cutinput: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at
mapPartitionsWithIndex at <console>:25

scala> cutinput.collect
res0: Array[String] = Array(17850,2.55,536365, 17850,3.39,536365,
17850,2.75,536365, 17850,3.39,536365, 17850,3.39,536365, 17850,7.65,536365,
17850,4.25,536365, 17850,1.85,536366, 17850,1.85,536366, 13047,4.25,536368,
13047,4.95,536368, 13047,4.95,536368, 13047,4.95,536368, 13047,1.69,536367,
13047,2.1,536367, 13047,2.1,536367, 13047,3.75,536367, 13047,1.65,536367,
13047,4.25,536367, 13047,4.95,536367, 13047,9.95,536367, 13047,5.95,536367,
13047,5.95,536367, 13047,7.95,536367, 13047,7.95,536367, 13047,5.95,536369,
12583,3.75,536370, 12583,3.75,536370, 12583,3.75,536370, 12583,0.85,536370,
12583,0.65,536370, 12583,0.85,536370, 12583,1.25,536370, 12583,2.95,536370,
12583,2.95,536370, 12583,1.95,536370, 12583,1.95,536370, 12583,1.95,536370,
12583,0.85,536370, 1258...

scala>

scala> val re = cutinput.map(line => {val data = line.split(",");(data(0),
data(1).toDouble)}).reduceByKey(_+_)
re: org.apache.spark.rdd.RDD[(String, Double)] = ShuffledRDD[4] at
reduceByKey at <console>:25

scala>

scala> val re_sort = re.filter(x=> !(x._1=="")).sortBy(x=>x._2,false)
re_sort: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[10] at
sortBy at <console>:25

scala>

scala> re_sort.take(50)
```

```
res1: Array[(String, Double)] = Array((12748,1618.1500000000015),
(14911,1573.1600000000014), (17850,1176.2299999999982),
(17841,1073.1299999999997), (14606,828.5199999999996),
(16607,726.3199999999999), (14527,666.1399999999996),
(17340,613.5099999999999), (15311,582.7899999999997),
(15044,545.4599999999998), (13174,519.7500000000001),
(14667,506.5499999999998), (15727,456.44999999999993),
(17961,414.82000000000005), (14030,413.75), (18116,410.97999999999973),
(15039,404.6699999999998), (16873,401.4299999999999),
(18118,389.7099999999998), (15574,386.2499999999999),
(14180,382.24999999999994), (16713,377.51999999999987),
(18055,377.3799999999999), (14505,373.29999999999956),
(15498,369.66999999999996), (15808,366.2999999999998),
(15570,363.50999999999993), (12...

scala>
```

CH03实训3：通过Spark编程计算各城市的平均气温

训练要点

1. 掌握RDD创建方法
2. 掌握map()、groupBy()、mapvalues()、reduce()方法的使用

需求说明

天气预报每天都会显示各城市的温度，方便出行人士根据当天的温度穿上合适的衣物现有一份各城市的温度数据文件avgTemperature.txt,数据如表所示，记录了某段时间范围内各城市每天的温度，文件中每一行数据分别表示城市名和温度，现要求使用spark 编程计算出各城市的平均气温。

各城市温度部分数据

```
beiing  28.1
shanghai 28.7
guangzhou 32.0
shenzhen 33.1
beiing  27.3
shanghai 30.1
guangzhou 33.3
shenzhen 28.6
beljing 28.2
shanghai 29.1
guangzhou 32.0
shenzhen 32.1
```

实现思路及步骤

1. 通过textFile()方法读取数据创建RDD
2. 使用map()方法将输人数据按制表符(TAB)进行分割，并转化成（城市，温度）的形式。
3. 使用groupBy()方法按城市分组，得到每个城市对应的所有温度。
4. 使用mapvalues()和reduce()方法计算各城市平均气温

作业要求

提交源码和脚本运行过程截图

实现参考

数据准备

```
cd /root/spark
wget https://biglab.site//b59510spark/file/spark_t_data.tar.gz
tar -xzvf ./spark_t_data.tar.gz
hdfs dfs -put ./spark_t_data/avgTemperature.txt /user/myname/
```

在spark-shell上执行

```
val input = sc.textFile("/user/myname/avgTemperature.txt")
// 利用map算子转换数据类型
val mapRDD = input.map(
      data => {
      val datas = data.split("\t")
      (datas(0), datas(1).toDouble)
            }
          )
//按城市分组
val groupRDD = mapRDD.groupBy(_._1)
// 计算各城市平均气温
val avgTmp = groupRDD.mapValues(data => {
            val a = data.reduce((x, y) => ("", x._2 + y._2))
            a._2 / data.size
          })
avgTmp.collect
```

spark-shell预期输出：

```
scala> val
worddata=sc.textFile("/user/myname/words.txt").flatMap(x=>x.split("
")).map(x=>(x,1)).reduceByKey((x,y)=>x+y)
worddata: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[16] at
reduceByKey at <console>:24

scala>

scala> val worddata_3=worddata.filter(x=>x._2>3).map(x=>x._1)
```

```
worddata_3: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[18] at map at <console>:25

scala>

scala>
worddata_3.repartition(1).saveAsTextFile("/user/myname/output_wordcount")


scala> val input = sc.textFile("/user/myname/avgTemperature.txt")
input: org.apache.spark.rdd.RDD[String] = /user/myname/avgTemperature.txt MapPartitionsRDD[25] at textFile at <console>:24

scala> // 利用map算子转换数据类型

scala> val mapRDD = input.map(
     |       data => {
     |       val datas = data.split("\t")
     |       (datas(0), datas(1).toDouble)
     |             }
     |           )
mapRDD: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[26] at map at <console>:25

scala> //按城市分组

scala> val groupRDD = mapRDD.groupBy(_._1)
groupRDD: org.apache.spark.rdd.RDD[(String, Iterable[(String, Double)])] = ShuffledRDD[28] at groupBy at <console>:25

scala> // 计算各城市平均气温

scala> val avgTmp = groupRDD.mapValues(data => {
     |             val a = data.reduce((x, y) => ("", x._2 + y._2))
     |             a._2 / data.size
     |           })
avgTmp: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[29] at mapValues at <console>:25

scala> avgTmp.collect
res3: Array[(String, Double)] = Array((guangzhou,32.43333333333333), (beijing,27.86666666666667), (shenzhen,32.6), (shanghai,29.3))

scala>
```