# 案例分析与编程实现

## 案例一

### a. 案例描述

提起 Word Count(词频数统计)，相信大家都不陌生，就是统计一个或者多个文件中单词出现的次数。本文将此作为一个入门级案例，由浅入深的开启使用 Scala 编写 Spark 大数据处理程序的大门。

### b．案例分析

对于词频数统计，用 Spark 提供的算子来实现，我们首先需要将文本文件中的每一行转化成一个个的单词, 其次是对每一个出现的单词进行记一次数，最后就是把所有相同单词的计数相加得到最终的结果。

对于第一步我们自然的想到使用 flatMap 算子把一行文本 split 成多个单词，然后对于第二步我们需要使用 map 算子把单个的单词转化成一个有计数的 Key-Value 对，即 word -> (word,1). 对于最后一步统计相同单词的出现次数，我们需要使用 reduceByKey 算子把相同单词的计数相加得到最终结果。  
c. 编程实现

##### 清单 1.SparkWordCount 类源码

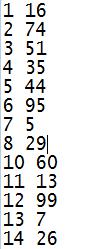
import org.apache.spark.{SparkConf, SparkContext}  
object HelloWorld{  
 def main(args:Array[String]){  
 val path1:String = args(0)  
 val path2:String = args(1)  
 //val master:String = args(2)  
 val conf = new SparkConf().setAppName("HelloWorld")  
 val sc = new SparkContext(conf)  
 val rdd1=sc.textFile(path1)  
 val rdd2 = rdd1.flatMap(\_.split(" ")).map(x=>(x,1)).reduceByKey(\_+\_)  
 //println(rdd2.collect().mkString(","))  
 rdd2.saveAsTextFile(path2)  
 sc.stop()  
 }  
}

## 案例二

a. 案例描述

该案例中，我们将假设我们需要统计一个 1000 万人口的所有人的平均年龄，当然如果您想测试 Spark 对于大数据的处理能力，您可以把人口数放的更大，比如 1 亿人口，当然这个取决于测试所用集群的存储容量。假设这些年龄信息都存储在一个文件里，并且该文件的格式如下，第一列是 ID，第二列是年龄。

##### 图 7. 案例二测试数据格式预览



现在我们需要用 Scala 写一个生成 1000 万人口年龄数据的文件，源程序如下：

##### 清单 3. 年龄信息文件生成类源码

|  |
| --- |
| **import** java.io.FileWriter **import** java.io.File **import** scala.util.Random  **object** SampleDataFileGenerator {   **def** main(args:Array[String]) {  **val** writer = **new** FileWriter(**new** File(**"D:\\甲骨文资料\\Spark师资培训\\data\\input\\sample\_age\_data.txt"**),**false**)  **val** rand = **new** Random()  **for** ( i <- 1 to 10000) {  writer.write( i + **" "** + rand.nextInt(100))  writer.write(System.*getProperty*(**"line.separator"**))  }  writer.flush()  writer.close()  } } |

b. 案例分析

要计算平均年龄，那么首先需要对源文件对应的 RDD 进行处理，也就是将它转化成一个只包含年龄信息的 RDD，其次是计算元素个数即为总人数，然后是把所有年龄数加起来，最后平均年龄=总年龄/人数。

对于第一步我们需要使用 map 算子把源文件对应的 RDD 映射成一个新的只包含年龄数据的 RDD，很显然需要对在 map 算子的传入函数中使用 split 方法，得到数组后只取第二个元素即为年龄信息；第二步计算数据元素总数需要对于第一步映射的结果 RDD 使用 count 算子；第三步则是使用 reduce 算子对只包含年龄信息的 RDD 的所有元素用加法求和；最后使用除法计算平均年龄即可。

由于本例输出结果很简单，所以只打印在控制台即可。

c. 编程实现

##### 清单 4.AvgAgeCalculator 类源码

|  |
| --- |
| **import** org.apache.spark.{SparkConf, SparkContext}  **object** AvgAgeCalculator {   **def** main(args: Array[String]): Unit = {  **if**(args.length!=3){  System.*out*.println(**"Usage:"**+**this**.getClass+**"<input><Appname><master>"**)  System.*out*.println(1)  }  **val** input = args(0)  **val** appname = args(1)  **val** master = args(2)  **val** conf = **new** SparkConf().setAppName(appname).setMaster(master)  **val** sc = **new** SparkContext(conf)  **val** text = sc.textFile(input,2)  **val** count = text.count()  *//val sum\_text = text.map(\_.split(" ")(1).toInt).reduce(\_+\_)* **val** sum\_text = text.map(\_.split(**" "**)(1).toInt).sum() *// val agv\_text = sum\_text/count* **val** text1 = text.map(x=>x.split(**" "**)).map(x=>(1,x(1).toInt)).reduce((a,b)=>(a.\_1+b.\_1,a.\_2+b.\_2))  **val** agv\_text = text1.\_2/text1.\_1   *println*(agv\_text)  }  } |

d. 提交到集群执行

要执行本实例的程序，需要将刚刚生成的年龄信息文件上传到 HDFS 上，假设您刚才已经在目标机器上执行生成年龄信息文件的 Scala 类，并且文件被生成到了/home/fams 目录下。

那么您需要运行一下 HDFS 命令把文件拷贝到 HDFS 的/user/fams 目录。

##### 清单 5. 年龄信息文件拷贝到 HDFS 目录的命令

hdfs dfs –copyFromLocal /home/fams /user/fams

##### 清单 6.AvgAgeCalculator 类的执行命令

./spark-submit \

--class com.ibm.spark.exercise.basic.AvgAgeCalculator \

--master spark://hadoop036166:7077 \

--num-executors 3 \

--driver-memory 6g \

--executor-memory 2g \

--executor-cores 2 \

/home/fams/sparkexercise.jar \

hdfs://hadoop036166:9000/user/fams/inputfiles/sample\_age\_data.txt

e. 监控执行状态

在控制台您可以看到如下所示信息：

##### 图 8. 案例二输出结果

IMG_259

我们也可以到 Spark Web Console 去查看 Job 的执行状态

##### 图 9. 案例二完成状态

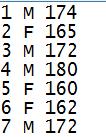


## 案例三

a. 案例描述

本案例假设我们需要对某个省的人口 (1 亿) 性别还有身高进行统计，需要计算出男女人数，男性中的最高和最低身高，以及女性中的最高和最低身高。本案例中用到的源文件有以下格式, 三列分别是 ID，性别，身高 (cm)。

##### 图 10. 案例三测试数据格式预览



我们将用以下 Scala 程序生成这个文件，源码如下：

##### 清单 7. 人口信息生成类源码

|  |
| --- |
| **object** PeopleInfoFileGenerator {  **def** main(args:Array[String]) {  **val** writer = **new** FileWriter(**new** File(**"D:\\甲骨文资料\\Spark师资培训\\data\\input\\sample\_people\_info.txt"**),**false**)  **val** rand = **new** Random()  **for** ( i <- 1 to 100) {  **var** height = rand.nextInt(220)  **if** (height < 50) {  height = height + 50  }  **var** gender = *getRandomGender* **if** (height < 100 && gender == **"M"**)  height = height + 100  **if** (height < 100 && gender == **"F"**)  height = height + 50  writer.write( i + **" "** + *getRandomGender* + **" "** + height)  writer.write(System.*getProperty*(**"line.separator"**))  }  writer.flush()  writer.close()  *println*(**"People Information File generated successfully."**)  }   **def** getRandomGender() :String = {  **val** rand = **new** Random()  **val** randNum = rand.nextInt(2) + 1  **if** (randNum % 2 == 0) {  **"M"** } **else** {  **"F"** }  } } |

b. 案例分析

对于这个案例，我们要分别统计男女的信息，那么很自然的想到首先需要对于男女信息从源文件的对应的 RDD 中进行分离，这样会产生两个新的 RDD，分别包含男女信息；其次是分别对男女信息对应的 RDD 的数据进行进一步映射，使其只包含身高数据，这样我们又得到两个 RDD，分别对应男性身高和女性身高；最后需要对这两个 RDD 进行排序，进而得到最高和最低的男性或女性身高。

对于第一步，也就是分离男女信息，我们需要使用 filter 算子，过滤条件就是包含”M” 的行是男性，包含”F”的行是女性；第二步我们需要使用 map 算子把男女各自的身高数据从 RDD 中分离出来；第三步我们需要使用 sortBy 算子对男女身高数据进行排序。

c. 编程实现

在实现上，有一个需要注意的点是在 RDD 转化的过程中需要把身高数据转换成整数，否则 sortBy 算子会把它视为字符串，那么排序结果就会受到影响，例如 身高数据如果是：123,110,84,72,100，那么升序排序结果将会是 100,110,123,72,84，显然这是不对的。

##### 清单 8.PeopleInfoCalculator 类源码

|  |
| --- |
| **import** org.apache.spark.{SparkConf, SparkContext}  **object** PeopleInfoCalculator {   **def** main(args: Array[String]): Unit = {   **if**(args.length!=2){  System.*out*.println(**"Usage:"**+**this**.getClass.getName+**"<input><master>"**)  System.*exit*(1)  }  **val** input = args(0)  **val** master = args(1)  *//var maxtemp = 0  //var mintemp =0* **val** conf = **new** SparkConf().setAppName(**this**.getClass.getName).setMaster(master)  **val** sc = **new** SparkContext(conf)  **val** lines = sc.textFile(input,2)  **val** maledata = lines.map(x=>x.split(**" "**))  **val** maledata1=maledata.map(x=>(x(1),x(2).toInt))    **val** mindata=maledata1.reduceByKey((a,b)=>  {**if**(a > b) a **else** b})  **val** maxdata =maledata1.reduceByKey((a,b)=>{  **if**(a<b) a **else** b})  *println*(mindata.collect().mkString(**","**)+**"\n"**+maxdata.collect().mkString(**","**)) *// val mandata=maledata.filter(\_.contains("M")).map(\_(2).toInt) // val womandata=maledata.filter(\_.contains("F")).map(\_(2).toInt) // val manMax = mandata.max() // val manMin = mandata.min() // // val womanMax = womandata.max() // val womenMin = womandata.min() // println("M:"+manMax+" "+manMin+"\n"+"F:"+womanMax+" "+womenMin)* }  } |

d. 提交到集群执行

在提交该程序到集群执行之前，我们需要将刚才生成的人口信息数据文件上传到 HDFS 集群，具体命令可以参照上文。

##### 清单 9.PeopleInfoCalculator 类的执行命令

./spark-submit \

--class com.ibm.spark.exercise.basic.PeopleInfoCalculator \

--master spark://hadoop036166:7077 \

--num-executors 3 \

--driver-memory 6g \

--executor-memory 3g \

--executor-cores 2 \

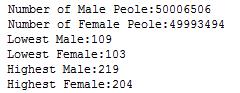
/home/fams/sparkexercise.jar \

hdfs://hadoop036166:9000/user/fams/inputfiles/sample\_people\_info.txt

e. 监控执行状态

对于该实例，如程序中打印的一样，会在控制台显示如下信息：

##### 图 11. 案例三输出结果



在 Spark Web Console 里可以看到具体的执行状态信息

##### 图 12. 案例三完成状态

