Spark

一、尝试使用多个Transformations和Actions函数

对一些Transformations和Actions函数进行使用，了解其用法和效果。

1. 创建RDD

```
scala> val textFile = sc.textFile("hdfs://mycluster/wc/paper_large.txt")
```

2. 统计数据条数

```
scala> textFile.count()
res1: Long = 87021873
```

3. 取出第n条数据，这里n=2

```
scala> textFile.take(2)
res10: Array[String] = Array(Geoinformatica (2015) 19:525C565, DOI
10.1007/s10707-014-0220-8)
```

4. 统计包含2015的数据条数

```
scala> textFile.filter(line => line.contains("2015")).count()
res14: Long = 424736
```

二、实现WordCount功能

输入输出

输入：hdfs://mycluster/wc/paper_large.txt

输出：hdfs://mycluster/user/2017210879/spark-output

操作步骤

进入spark shell

```
$ spark-shell
```

读取paper_large.txt创建RDD，先将RDD每行根据空格且分开，再把切分开的单词map为(word,1)的形式，最后根据每个key（也就是word）将其value个数累加得到该word的总出现次数

```
scala> val textFile = sc.textFile("hdfs://mycluster/wc/paper_large.txt")
scala> val wordCounts = textFile.flatMap(line => line.split(" ")).map(word
=> (word, 1)).reduceByKey((a, b) => a + b)
scala> wordCounts.saveAsTextFile("hdfs://mycluster/user/2017210879/spark-
output")
```

结果对比

首先将spark的wordcount结果下载到服务器中，再使用cat将其全部合并，合并后再排序生成最终结果

```
hadoop fs -get spark-output
cd spark-output
cat part* > output
sort output > Sparkout
```

同理，对于上次实验hadoop的结果合并后，也将其排序

```
sort part-00000 > hout
```

由于hadoop的wordcount结果的格式为**key value**，而spark的结果格式为**(key, value)**

格式不同无法对比，所以使用如下命令，将hadoop的结果格式转化为与spark的一致

```
awk -F "\t" '{print "("$1","$2")"}' hout > Hadoopout
```

最后使用diff命令对两个结果进行对比，没有输出，证明两文件结果完全一致，结果正确

```
diff Hadoopout Sparkout
```

时间对比

由于使用spark-shell，不知道或者没有对命令运行计时的方法，所以没有对spark运行wordcount精确地计时，由我个人的感觉来说，很明显使用spark速度快很多，2.4G的文件wordcount大概只需要十几秒，而使用Hadoop用时则是spark的几十倍。

原因在于spark使用内存处理一切数据，而hadoop从磁盘读取数据，计算完毕后都要把数据存放到磁盘上，内存的读写速度比磁盘要快很多。

三、另三种方法实现WordCount功能

方法一

首先读取文件创建RDD

```
scala> val textFile = sc.textFile("hdfs://mycluster/wc/paper_large.txt")
```

将RDD根据空格切分开，然后将每个单词map为（word,1）的形式

```
scala> val wordlist = textFile.flatMap(line => line.split(" ")).map(word =>
(word, 1))
```

将（word，1）这样的元组根据第一个元素（也就是word）使用groupBy合并，形成（word，list{1,1,1,....}）这样的元组，再使用map将每个元组中的list{1,1,1,....}替换为list.size也就是出现的次数，得到了每个单词的计数。

```
scala> val wordcount = wordlist.groupBy(_._1).map { case (word, list) =>
(word, list.size)  }
```

剩下的保存到文件saveAsTextFile，排序，格式化数据，使用diff对比等一系列操作同上，最后结果正确

方法二

首先读取文件创建RDD

```
scala> val textFile = sc.textFile("hdfs://mycluster/wc/paper_large.txt")
```

将RDD根据空格切分开，切分开的每一条数据是一个单词

```
scala> val wordlist = textFile.flatMap(line => line.split(" "))
```

使用countByValue函数对相同的单词计数，得到每个单词的计数结果

```
scala> val wordcount = wordlist.countByValue()
```

最终结果wordcount中内容如下，只是格式不同

```
ACM -> 8168, chosen -> 16336, Master, -> 187864, possibility -> 8168, user-
location-activity -> 16336,....
```

剩下的保存到文件saveAsTextFile，排序，格式化数据，使用diff对比等一系列操作同上，最后结果正确

方法三

首先读取文件创建RDD

```
scala> val textFile = sc.textFile("hdfs://mycluster/wc/paper_large.txt")
```

将RDD根据空格切分开，再将切分开的每个单词map为（word,1）的形式

```
scala> val wordlist = textFile.flatMap(line => line.split(" ")).map(word =>
(word, 1))
```

使用countByKey函数对（word,1）这样的元组根据Key（也就是word）来计数，得到每个单词计数结果

```
scala> val wordcount = wordlist.countByKey()
```

最终结果wordcount中内容如下，格式同方法二

```
45 -> 653440, P(r) -> 16336, Sissel, -> 8168, Much -> 8168, termination. ->
8168, abstraction -> 24504,....
```

剩下的保存到文件saveAsTextFile，排序，格式化数据，使用diff对比等一系列操作同上，最后结果正确