# 金融大数据作业5

施宇 191250119

## 题目要求

在HDFS上加载莎士比亚文集的数据文件（shakespeare-txt.zip解压后目录下的所有文件），编写MapReduce程序进行词频统计，并按照单词出现次数从大到小排列，输出

1. 每个作品的前100个高频单词；
2. 所有作品的前100个高频单词，要求忽略大小写，忽略标点符号（punctuation.txt），忽略停词（stop-word-list.txt），忽略数字，单词长度>=3。输出格式为"<排名>：<单词>，<次数>“，输出可以根据作品名称不同分别写入不同的文件，也可以合并成一个文件。

## 基本思路

由于需要输出每个作品的前100个高频单词，于是选择对每个作品单独进行处理，通过循环依次执行每个作品的词频统计和排序工作。对于每个作品，先进行词频统计，再进行排序。

对于词频统计任务，在map阶段，将输入的一行文本转为小写后按空格划分为单词，然后依次剔除数字，停词，标点符号，然后判断单词长度是否大于等于3，最后输出键值对。在reduce操作阶段，为了记录作品前100个高频单词的结果，使用TreeMap记录每次reduce的结果，始终保持TreeMap中记录的键值对个数不超过100个，由于TreeMap会自动将键值对按照Key的值进行排序，因此最后TreeMap的结果即为该作品前100个高频单词。由于TreeMap中不能存储相同的Key，因此对原本的Key进行一定改动，将单词和词频一同作为Key，Key与Key之间的比较先按照词频进行比较，若词频相同，再按照单词进行比较。这样便可以通过TreeMap完成词频的排序和获取前100个词频的任务。待reduce工作全部完成后，在cleanup阶段，将TreeMap中存储的100个词频数据通过MultipleOutputs类按照要求的格式输出即可。

词频统计任务和排序任务分别通过一个类完成，通过调用该类的run函数来启动任务，在run函数中进行job的相关配置和执行。再通过一个启动类来读取所有作品名称并通过训练依次进行每个作品的词频统计和排序。当所有作品都完成后，通过一个合并任务，将所有作品的词频统计结果合并到一起，再进行排序后，得到所有作品的前100个高频单词。

因此，整个任务的执行逻辑可以代码表示如下：

```
// textList为所有作品的作品名列表
for(String textName:textList){
    //先执行词频统计任务，等到词频统计完成后，再执行排序任务
    if(Count.run(in + textName, wordCount + textName)){
        Sort.run(wordCount + textName, sort + textName, topK + textName);
    }
}
// 对所有词频统计结果合并并排序，得到所有文本的词频统计+排序结果
Combiner.run(wordCount + "*\\", topK + "rawCountAll");
Sort.run(topK + "rawCountAll", sort + "all", topK + "all");
```

## 代码

通过7个类实现整个词频统计任务，每个类的作用如下：

| 类名 | 功能 |
| --- | --- |
| WordCount | 程序启动入口 |
| WCMapper | 词频统计中map部分的实现 |
| WCReducer | 词频统计中reduce部分的实现 |
| Count | 词频统计启动入口 |
| Sort | 排序任务启动入口 |
| WCString | 作为TreeMap中Key的数据结构，以完成词频的排序 |
| Combiner | 实现所有作品词频统计结果的合并 |

具体代码内容见代码文件。

# 运行程序

通过IDEA将该程序打包成jar包后，可以跨平台运行程序：

配置好后直接编译构建jar包，这里为hdfs-api-exise.jar。

在Windows下单机运行该程序，打开cmd启动Hadoop，然后运行jar包：

```
hadoop jar D:\Hadoop\code\test_hadoop\out\artifacts\hdfs_api_exise_jar\hdfs-api-exise.jar test.WordCount /user/86137/input /user/hadoop/output
```

运行结果：

运行结果保存在output文件夹中，包含以下内容：

| 文件 | 内容 |
| --- | --- |
| rawCount | 文件夹，存放每个作品的词频统计结果，每个作品的结果都在以该作品名命名的文件夹中 |
| rawCountAll | 文件夹，存放所有作品词频统计结果合并后的结果 |
| sort | 文件夹，存放每个作品词频统计结果的排序结果，每个作品的排序结果都在以该作品名命名的文件夹中 |
| all-r-00000 | 文件，所有作品的前100个高频单词 |
| xxx.txt-r-00000 | 文件，作品xxx的前100个高频单词 |

由于Windows下单机运行程序，Hadoop的Web端并不能监控到任务的提交与运行状态，因此在之前搭建的Linux集群中运行该程序。先将需要读取的input文件和jar包复制到Linux中，再复制到docker容器中，然后再docker中将input文件上传到hdfs上。通过如下命令执行程序：

```
hadoop jar hdfs_api_exise_jar/hdfs-api-exise.jar test.WordCount /user/86137/input /user/86137/output
```

程序运行结果可以通过Web端查看，由于一共有40个作品，每个作品通过一个词频统计和一个排序任务得到结果，所有作品的高频词需要通过一个合并任务和一个排序任务得到结果，所以共有40*2+2=82个任务：

# 性能与可拓展性分析

由于程序将每个作品的词频统计和排序任务分成两个任务处理，并且每个作品的词频统计及排序也是分别通过不同的任务处理的，因此需要多次IO操作，读取大量的文件，影响了程序的性能。另外，通过比对标点符号文件中所提供的标点符号的方式来剔除词频统计中的标点符号，也较为耗时，每个节点都需要读取标点符号文件并进行循环来完成该过程。

因为程序将每个任务分成单独的job去处理，且停词和标点符号的处理均通过读取相应的包含停词和标点符号的文件来完成的，因此程序的可扩展性较高，每个部分的相应组件可以拿出来独立运行，模块间的耦合度较低。

# 改进

对于标点符号的剔除，可以通过正则表达式完成：

```
String token=token.replaceAll("[\\pP\\p{Punct}]", "");
```

通过该正则表达式可以提出字符串中的所有标点符号，只留下字母和数字。

另外，对于停词文件的读取，可以通过缓存加载来提高读取的速度：

在setup函数中读取停词

```
URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
String patternsFileName = patternsPath.getName().toString();
parseSkipFile(patternsFileName);
```

```
private void parseSkipFile(String fileName){
    try{
        fis = new BufferedReader(new FileReader(fileName));
        String pattern = null;
        while((pattern = fis.readLine()) != null){
            patternsToSkip.add(pattern);
        }
    }catch(IOException ioe){
        System.err.println("Caught exception while parsing the cached file '"
                           + StringUtils.stringifyException(ioe));
    }
}
```

在map函数中剔除单词中的停词：

```
for(String pattern: patternsToSkip){
    line = line.replaceAll(pattern, "");
}
```

# 附录：程序打包到Hadoop集群上运行

集群上的java版本和程序打包成jar包时的java版本要对应，即执行时的java版本不能小于打包时的java版本，否则程序执行时会出现类似"Unsupported major.minor version 52.0"的报错。

在IDEA上编写好程序后，打开File->Project Structure，点击左上方的加号->JAR->From modules with dependencies...

Main Class设置为程序的入口类，JAR files from libiaries选择第二项，然后点击OK。

打开Build->Build Artifacts，然后Build刚刚创建的Artifact：

Build完成后，可以在项目的out目录下找到打包好的jar包：

然后将jar包复制到Linux中，在启动Linux上的Hadoop集群后，将jar包和程序执行需要的input文件夹复制到docker的主节点上：

```
docker cp 本地文件路径 容器ID:容器路径
```

容器路径要写绝对路径，否则可能复制不成功，并且也没有报错信息。可以通过`docker ps -a`查看容器ID

然后进入主节点，将input文件夹上传到hdfs的相应位置上：

```
hdfs dfs -put 本地文件路径 要拷贝到hdfs上的位置
```

然后执行jar包：

```
hadoop jar hdfs_api_exise_jar/hdfs-api-exise.jar test.WordCount /user/86137/input /user/86137/output
```