《金融大数据处理技术》作业5报告

191870068 嵇泽同

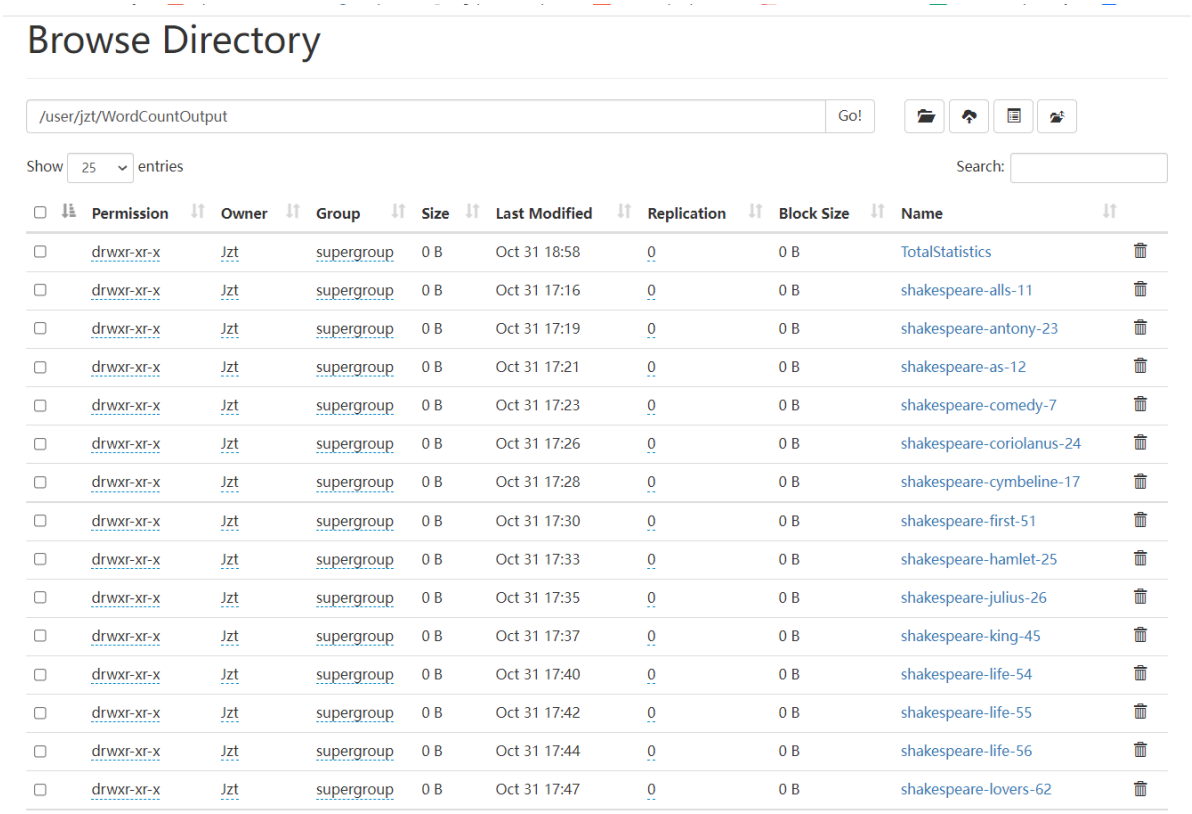
一、实验结果

yarn web截图：

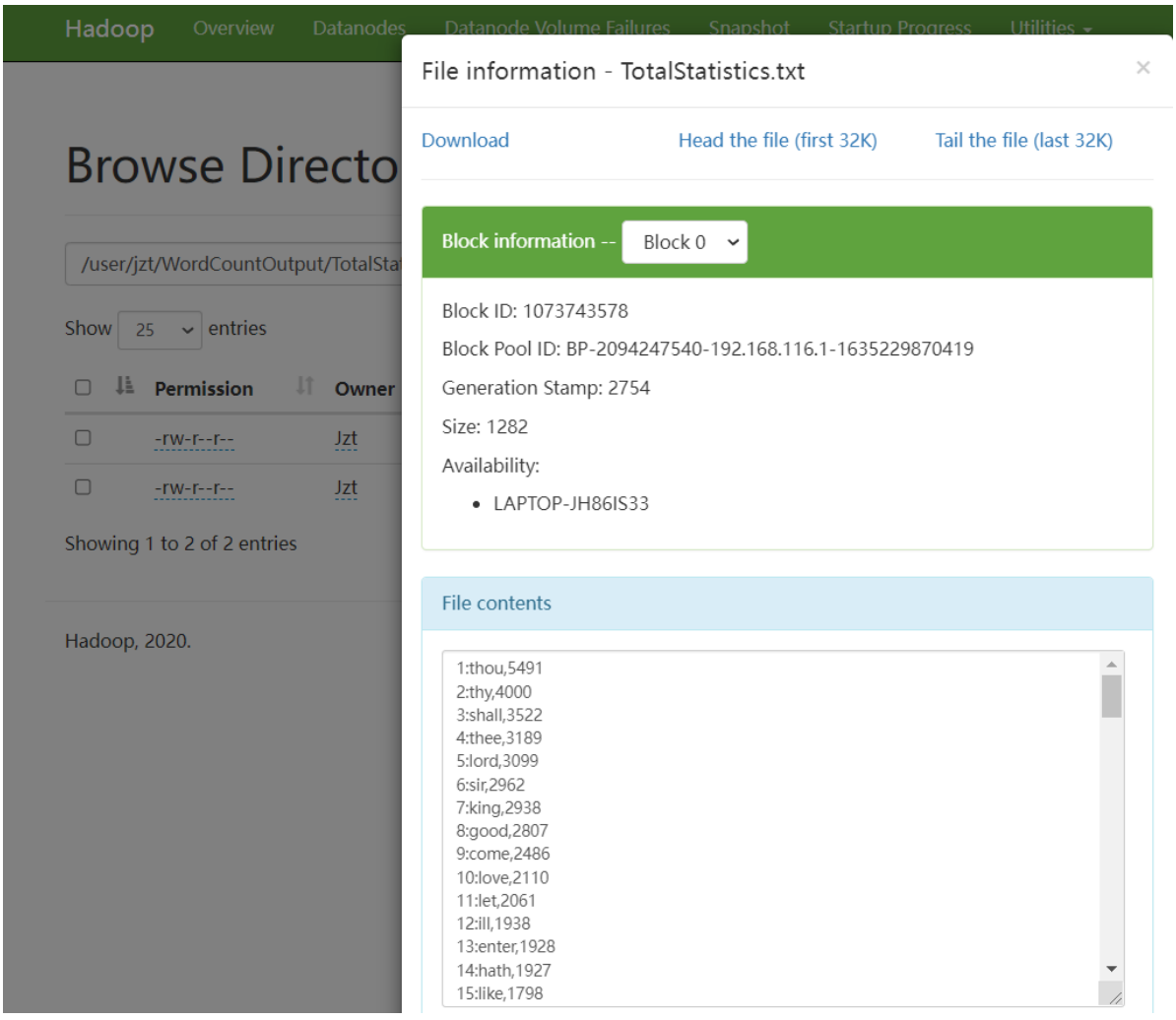
ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tr
application_1635591297624_0140	Jzt	final wordcount2	MAPREDUCE		default	0	Sun Oct 31 18:57:06 +0800 2021	Sun Oct 31 18:57:10 +0800 2021	Sun Oct 31 18:58:12 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0139	Jzt	final wordcount	MAPREDUCE		default	0	Sun Oct 31 18:47:58 +0800 2021	Sun Oct 31 18:48:03 +0800 2021	Sun Oct 31 18:57:02 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0138	Jzt	word count2	MAPREDUCE		default	0	Sun Oct 31 18:46:48 +0800 2021	Sun Oct 31 18:46:53 +0800 2021	Sun Oct 31 18:47:54 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0137	Jzt	word count	MAPREDUCE		default	0	Sun Oct 31 18:45:34 +0800 2021	Sun Oct 31 18:45:39 +0800 2021	Sun Oct 31 18:46:45 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0136	Jzt	word count2	MAPREDUCE		default	0	Sun Oct 31 18:44:25 +0800 2021	Sun Oct 31 18:44:29 +0800 2021	Sun Oct 31 18:45:31 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0135	Jzt	word count	MAPREDUCE		default	0	Sun Oct 31 18:43:15 +0800 2021	Sun Oct 31 18:43:20 +0800 2021	Sun Oct 31 18:44:21 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0134	Jzt	word count2	MAPREDUCE		default	0	Sun Oct 31 18:42:07 +0800 2021	Sun Oct 31 18:42:11 +0800 2021	Sun Oct 31 18:43:12 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0133	Jzt	word count	MAPREDUCE		default	0	Sun Oct 31 18:40:55 +0800 2021	Sun Oct 31 18:41:00 +0800 2021	Sun Oct 31 18:42:04 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0132	Jzt	word count2	MAPREDUCE		default	0	Sun Oct 31 18:39:47 +0800 2021	Sun Oct 31 18:39:52 +0800 2021	Sun Oct 31 18:40:53 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0131	Jzt	word count	MAPREDUCE		default	0	Sun Oct 31 18:38:35 +0800 2021	Sun Oct 31 18:38:40 +0800 2021	Sun Oct 31 18:39:44 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0130	Jzt	word count2	MAPREDUCE		default	0	Sun Oct 31 18:37:27 +0800 2021	Sun Oct 31 18:37:32 +0800 2021	Sun Oct 31 18:38:32 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0129	Jzt	word count	MAPREDUCE		default	0	Sun Oct 31 18:36:16 +0800 2021	Sun Oct 31 18:36:21 +0800 2021	Sun Oct 31 18:37:24 +0800 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His
application_1635591297624_0128	Jzt	word	MAPREDUCE		default	0	Sun Oct 31	Sun Oct 31	Sun Oct 31	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		His

（不是很清楚“提交作业运行成功的WEB页面截图”是啥意思，个人感觉可能是这个页面……）

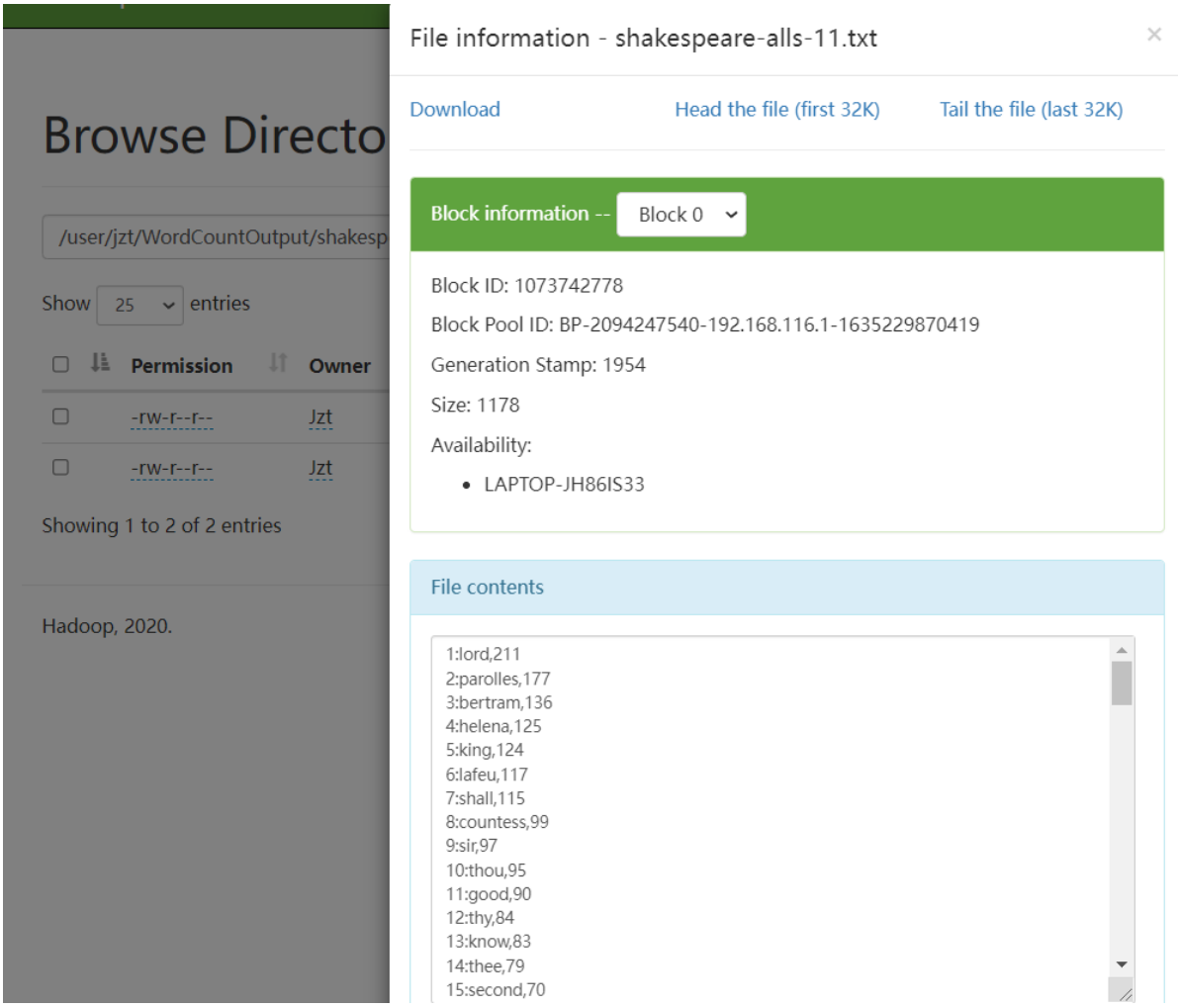
HDFS上的输出文件目录：



汇总词频的统计结果：



部分文档的个别统计结果（以shakespeare-alls-11.txt为例）：



二、设计思路

本次实现的基础是传统WordCount算法，在此基础上新增的功能有：

①忽略大小写、忽略标点、忽略停词、忽略数字、忽略长度小于3的单词；

②对结果单词按照词频从高到低进行排序；

③保留单词排序结果的前100个；

④输出格式为“<排名>：<单词>，<次数>”，而非默认的“单词\t次数”或“次数\t单词"；

⑤既能对每个txt文件分开统计，也能对所有txt文件整体统计；

⑥输出文件名符合规律（比如和输入文件名一致）；

下面逐一陈述对于以上每一点，本人的实现思路：

①忽略大小写、忽略标点、忽略停词、忽略数字、忽略长度小于3的单词

忽略大小写和标点的实现方式课堂上已经讲过（即WordCount2.0），即添加cache文件、并在map阶段将读取到的每一行文字转换成小写，同时读取cache文件，记录cache文件中存在的标点符号类型，并且对读取到的每一行文字都进行replace替换（将每一行中出现的符合条件的标点替换成空字符）即可。

而对于停词则不能采用上述忽略标点的方法。因为停词（比如you）可能是另一个非停词（如young）的一部分，如果直接盲目替换，则会导致读入不合法单词“ng”。而且如果像标点一样，对于读取到的每一行数据，都for循环遍历停词表并逐一替换，那么性能损耗是很大的（标点表很小，可以遍历，但是停词数量是相对较多的）。因此，对于停词，我采取的方式是：读取停词表并存入内存，对于

读取到的每一行，进行分词后，对于得到的每一个单词，用contains()方法判断该单词是否在停词表内，若是，则忽略该单词，若不是，则进行后续context.write()操作。与此同时，在判断每个单词是否是停词的同时，也可以同时判断其是否是数字（用正则表达式判断）、是否长度小于3，一旦有其中一个条件不满足，就忽略该单词。

```
StringTokenizer itr = new StringTokenizer(line);  //Java的字符串分解类，默认分隔符"空格"、"制表符
while (itr.hasMoreTokens()) {  //循环条件表示返回是否还有分隔符。
    String wo = itr.nextToken();
    if (!WordToSkip.contains(wo) && !wo.matches( regex: "^[0-9]*$") && wo.length() >= 3) {
```

②对结果单词按照词频从高到低进行排序

该部分在本人的实现中主要分为两个点：一是要排序，二是要输出结果从高到低。

关于排序，用一个Map-Reduce过程本人感觉较难实现，因为如果存在多个Reducer节点的话，在Partitioner分配key时是不知道每个词的词频的，而后续即使在shuffle过程中通过combiner或者在reduce过程中统计出词频，也无法再次重新通过Partitioner分配键值对，因此每个Reducer节点得到的数据在词频上可能有很大差异，无法实现输出的词频有序性。而如果设置成只有一个Reducer节点，那么同样，在reduce执行完统计完每个词的词频后，reduce的任务已经完成了，无法让其再额外实现排序。因此考虑采用链式MapReduce过程。

在一个传统的WordCount的Map-Reduce程序执行完毕之后，我们得到的输出文件格式是每一行都是"单词\t词频"的形式，且并不按照词频排序。注意到，在MapReduce的shuffle过程中，程序会自动对每个partition内的键值对按照key进行sort排序。因此，如果我们在第一次Map-Reduce过程结束后、已经得到乱序词频表后，再追加一个Map-Reduce过程，且设置成只有一个Reducer节点，并且将key设置成词频而非单词，那么Reducer得到的就是已经按照词频排序好的键值对序列，并且reduce过程不需要再做其他操作，直接将得到的已排序好的键值对序列写入输出文件即可，不用我们再手动设计排序算法了！因此，为了实现这一点，我们可以在第一次Map-Reduce过程中设置输出为"词频\t单词"的形式，也可以在第二次Map-Reduce过程的map阶段将读到的"单词\t词频"形式的键值对按照"词频\t单词"的形式输出到下一阶段，总之这一过程非常容易实现。

至此我们已经能够得到排好序的输出文件。但是值得注意的是，此时默认的结果（如果采用IntWritable作为词频的类的话）是按照词频从小到大进行排序。而我们要求的结果是词频从大到小进行排序，虽然可以通过后续再追加文件读写、修改操作来进行修改，但是总感觉这样显得有些"low"，而且万一输出文件非常大的话，传统的文件读入内存再修改的方式也难以实现。这里就涉及到为什么默认情况（IntWritable）下是从小到大排序了：在sort阶段，程序对key（即词频）进行比较，而IntWritable类实例间的比较就是正常的比较，1比2小，10比9大。而sort则按照比较的结果，将比出来较小的放在前面，较大的放在后面。那么，我们岂不是可以自己重写一个类，让实际比较小的数比出来反而大，这样比较的结果就和实际大小颠倒了，sort将比较出来较小的放在前面，也就是实际较大的放在前面了！为此，我们看一下IntWritable类的源码：

```
public int compareTo(IntWritable o) {
    int thisValue = this.value;
    int thatValue = o.value;
    return thisValue < thatValue ? -1 : (thisValue == thatValue ? 0 : 1);
}
```

而我们只需要自己定义一个类（比如我定义的是ReverseIntWritable类），基本照抄IntWritable类的源码，除了在compareTo方法中略作修改，将上图中框出来的"<"改成">"即可！如此之后，我们就可以得到按词频从大到小排序的输出结果了！

```java
public static class ReverseIntWritable implements WritableComparable<ReverseIntWritab
    private int value;

    public ReverseIntWritable() {}

    public ReverseIntWritable(int value) { this.set(value); }

    public void set(int value) { this.value = value; }

    public int get() { return this.value; }

    public void readFields(DataInput in) throws IOException {...}

    public void write(DataOutput out) throws IOException {...}

    public boolean equals(Object o) {...}

    public int hashCode() { return this.value; }

    public int compareTo(ReverseIntWritable o) {
        int thisValue = this.value;
        int thatValue = o.value;
        return thisValue > thatValue ? -1 : (thisValue == thatValue ? 0 : 1);
    }

    public String toString() { return Integer.toString(this.value); }

}
```

③保留单词排序结果的前100个

这个就比较容易实现了，在第二次Map-Reduce过程的Reduce类中设置一个变量记录已经接收并输出的键值对数量即可。因为如上所说，此时Reducer节点接收到的键值对序列已经按照词频从大到小排序好了，只要接收并输出前100个，并忽略之后其他的所有键值对即可。

```java
for (Text word : values) {
    if (occupied < 100) {
        RankWord rankword = new RankWord();
        rankword.set(occupied + 1, word.toString());
        context.write(rankword, key);
        occupied++;
    }
```

④输出格式为“<排名>：<单词>，<次数>”，而非默认的“单词\t次数”或“次数\t单词"

同样，这个可以采用后续追加文件读写、修改操作来实现，但是同样显得比较low而且面对大文件时无力。我在思考这个问题时，将“<排名>：<单词>，<次数>”分为两部分，“<排名>：<单词>”作为第一部分，而“<次数>”作为第二部分。注意到reduce输出是按照“键\t值”的形式，传统情况下是将键设置为单词，类型为Text。那么，如果我们自定义一个实现Writable接口的类（如我自己定义的是RankWord类），含义是既包含排名，也包含单词，并且自己编写有关于该类实例输出格式的方法，岂不是就能实现输出的键是“<排名>：<单词>”格式而非一般的“<单词>”格式（Text类）了？经过不断的探索，该方法确实是可行的，不过本人在探索途中也确实踩了不少坑，费了挺大功夫……

我们已经将键设为RankWord类，值不需变动，正常输出次数即可。此时剩下的问题就是，默认情况下的分隔符是制表符\t，而非逗号。要实现逗号分割，除了后续追加文件操作外，还有什么办法呢？我在网上找到了一些所谓的通过设置某一属性就能直接修改分隔符的方法，但是不知道是因为年代久远、版本不同还是什么原因，统统不奏效。联想到课上讲过的“高级操作”，我就开始思考通过自定义OutputFormat类的方法来实现。不得不说，对于我这个初学者（无论是对于MapReduce而言还是对于Java来言）来说，自定义输出类、重写相关方法相对而言确实更难一些，没有那么简洁直观。不过经过反复不断的探索，最终还是成功通过“自定义RecordWriter、自定义继承自FileOutputFormat类的OutputFormat类并重写该类的getRecordWriter方法”的方法成功实现了该目标，并且还实现了额外的功能——自定义输出路径（在RecordWriter中设置分割符为逗号，在自定义的输出类中自定义输出路径）。

```
public static class CommaRecordWriter extends RecordWriter<RankWord, ReverseIntWritable> {
    private FSDataOutputStream out;
    private String separator = ",";

    public CommaRecordWriter() {}

    public CommaRecordWriter(FSDataOutputStream fileOut) { this.out = fileOut; }

    public void write(RankWord key, ReverseIntWritable value) throws IOException, InterruptedException {
        try{
            out.write((key.toString() + separator + value.toString()).getBytes(StandardCharsets.UTF_8));
            this.out.write("\n".getBytes(StandardCharsets.UTF_8));
        } catch (Exception exc) {
            System.out.println("err");
            System.out.println(exc);
        }
    }

    public void close(TaskAttemptContext context) throws IOException, InterruptedException {...}
}
```

```
public static class SPOutputFormat extends FileOutputFormat<RankWord, ReverseIntWritable> {

    public SPOutputFormat() {}

    @Override
    public RecordWriter<RankWord, ReverseIntWritable> getRecordWriter(TaskAttemptContext job) throws IOException,
            InterruptedException {
//...
        Path outputDir = FileOutputFormat.getOutputPath(job);
        String[] sArray = outputDir.toString().split( regex: "\\/");
        String outputPath = outputDir.toString()+'/'+sArray[sArray.length-1]+".txt";
        Path path = new Path(outputPath);
//...
        URI output = URI.create(outputPath);

        Configuration conf = job.getConfiguration();
        FileSystem fs = FileSystem.get(output, conf);
        FSDataOutputStream out = fs.create(path);
        RecordWriter<RankWord, ReverseIntWritable> commaRecordWriter = new CommaRecordWriter(out)
        return commaRecordWriter;
    }
}
```

⑤既能对每个txt文件分开统计，也能对所有txt文件整体统计

以上部分可以实现对于单个txt文本进行统计。为了能自动对所有txt文本都自动执行该统计而不用手动对于每个txt文件都重新运行一遍程序，我的实现方式是编写for循环，同一时刻只有一个txt文本被处理，当一个文本被处理完毕后对下一个文本执行同样的操作。这样的方式非常直观，也很简洁易实现。

而至于对所有txt文件进行词频总排行，其实和单个文件比较类似，完全可以通过和单个文件一样的方式来进行，即从所有原始文件中重新读取数据，再次执行两次Map-Reduce过程。这样的话唯一的区别就在于输入文件是多个，而非一个。但是这样的话性能消耗是比较大的：我们之间已经对于每个文档都统计出了词频，（如果还没删掉的话）完全可以借助这些已有结果，而不需要重新去提取词频了。在之前的代码中，由于涉及两次Map-Reduce过程，我将第一次Map-Reduce过程的结果全部输出到一个中

间文件夹，并且在代码中一直没有删除该文件夹。因此，在统计全部词频时，只需要从这个中间文件夹中读取每个文档分别的词频信息，然后仿照之前对于单个文件的操作，再次运行一遍Map-Reduce过程即可。唯一需要注意的就是还需要对Mapper和Reducer类进行一些小的修改，不过都很基础，在这里就不细说了。

⑥输出文件名符合规律（比如和输入文件名一致）

对于这个问题，我一开始考虑的是能否自定义输出类，并在其中设置输出文件名。后来上网查找了相关资料后发现没有必要，只需要在最后一次reduce过程中做一些修改，让结果直接输出到输出文件而非调用context.write()方法，并且在该过程中可以指定输出文件名。如果还想指定输出文件名和输入文件名一致的话，该如何在reduce过程中获取输入文件名呢？我采取的方法是在一开始job配置时（此时能够获取到输入文件路径）就采用set()方法将文件路径存储到conf里，这样在reduce过程中就可以通过context获取到当初存入的文件名了：

```
public static class IntSumReducer extends Reducer<Text, IntWritable, IntWritable, Text> {    //继承泛型类Reducer
    private IntWritable result = new IntWritable();   //实例化IntWritable
    private MultipleOutputs<IntWritable, Text> mos;

    protected void setup(Context context) throws IOException, InterruptedException {
        mos = new MultipleOutputs(context);
    }

    protected void cleanup(Context context) throws IOException, InterruptedException {
        super.cleanup(context);
        mos.close();
    }
```

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
    Configuration conf = context.getConfiguration();
    String filePath = conf.get("mapreduce.inputpath");

    Path tmpFilePath = new Path(filePath);
    String fileName = tmpFilePath.getName();
    try {
        int sum = 0;
        for (IntWritable val : values)
            sum += val.get();
        result.set(sum);
          context.write(result, key);
        mos.write(result, key, fileName);
    } catch (Exception exc) {
        System.out.println(exc.getStackTrace());
    }
}
```

三、实验过程中遇到的问题和困难

由于本人在虚拟机方面经常遇到问题，让人头大，而且来回切换又经常感到麻烦，虚拟机用起来也没有主机方便舒服，加上听闻有些同学在此次作业中用了windows系统来跑，因此此次作业我也是全程在windows10系统上运行的，也算是一种尝试。在此次作业中，也一如既往地遇到了不少问题，下面我对其中主要的部分进行一下陈述：

①JAVA版本问题

在win10系统上，我此前已经安装了高版本的JAVA，考虑到可能存在不兼容问题，因此又去下载了jdk1.8。但是即使在修改了环境变量之后，在控制台中“java -version”指令依然提示出错：

```
C:\Users\Jzt>java -version
Error: could not open `F:\java\lib\amd64\jvm.cfg'
```

但是“javac -version”指令却可以查看到jdk1.8版本：

```
C:\Users\Jzt>javac -version
javac 1.8.0_60
```

个人感觉可能还是环境变量配置问题，上网查找发现众说纷纭，只能逐一慢慢踩坑排雷，最后终于成功解决：

1、出现原因

我之前安装的jdk是1.6；后来又装成1.8，把变量名JAVA_HOME改成了1.8的文件路径，最终导致java -version和javac -version显示的版本不一致。

2、错误结果

java -version　显示的是最新安装版本的java
javac -version 显示的是你配置环境变量版本的java

3、修改方法

把 %JAVA_HOME% 放在Path的头部；

1. 变量名：JAVA_HOME
2. 变量值：C:\Program Files\Java\jdk1.8.0
3. 变量名：CLASSPATH
4. 变量值：%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;
5. 变量名：Path
6. 变量值：%JAVA_HOME%\bin;

这些变量都是配置在系统变量里面的（见下图）

②win10系统下安装hadoop并运行

据说windows系统和hadoop的“相性”没有linux那么好，因此在安装配置hadoop的时候我格外小心，上网多方查找资料，最终发现一个对于我来说比较实用、比较好的教程：https://zhuanlan.zhihu.com/p/375066643。按照教程配置完成后，测试运行grep样例，成功：

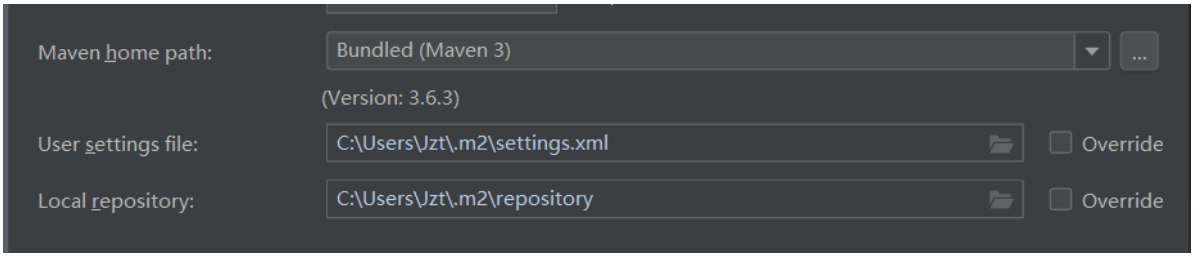
```
C:\Windows\System32\cmd.exe
2021-10-26 15:05:28,513 INFO mapreduce.Job:  map 87% reduce 28%
2021-10-26 15:05:36,622 INFO mapreduce.Job:  map 87% reduce 29%
2021-10-26 15:06:14,120 INFO mapreduce.Job:  map 90% reduce 29%
2021-10-26 15:06:26,248 INFO mapreduce.Job:  map 90% reduce 30%
2021-10-26 15:06:34,354 INFO mapreduce.Job:  map 97% reduce 30%
2021-10-26 15:06:35,369 INFO mapreduce.Job:  map 97% reduce 32%
2021-10-26 15:06:36,383 INFO mapreduce.Job:  map 100% reduce 32%
2021-10-26 15:06:41,472 INFO mapreduce.Job:  map 100% reduce 100%
2021-10-26 15:06:42,507 INFO mapreduce.Job: Job job_1635230022161_0002 completed successfully
2021-10-26 15:06:42,910 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=56
                FILE: Number of bytes written=8505662
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=112252
                HDFS: Number of bytes written=142
                HDFS: Number of read operations=98
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=4
                Launched map tasks=34
                Launched reduce tasks=1
                Data-local map tasks=34
                Total time spent by all maps in occupied slots (ms)=2248407
                Total time spent by all reduces in occupied slots (ms)=311306
                Total time spent by all map tasks (ms)=2248407
```

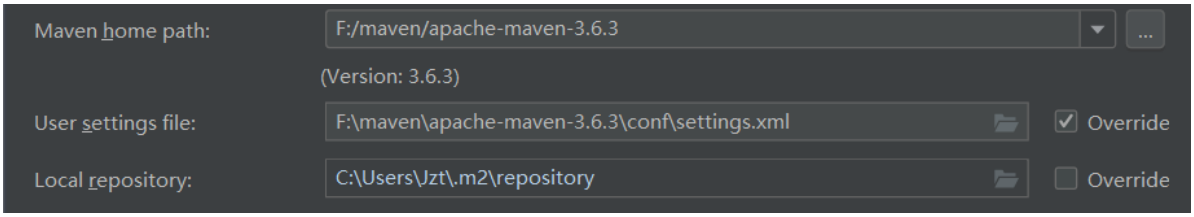
③IDEA+Maven配置

关于IDEA上hadoop的配置，我在网上找了很久，也尝试了很多教程，但是都不让人满意，也可能是由于机器配置不一样等问题。最终，还是找到了一篇比较适合我的教程：https://www.cnblogs.com/ziyaziya/p/12456586.html。其中，关于pom.xml的配置也是众说纷纭，我在参考了网上的教程后发现如下的配置始终无法成功：

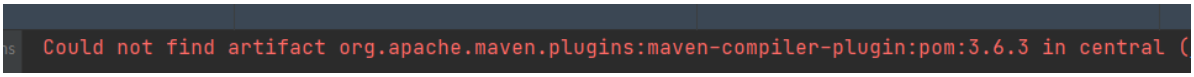
```
pom.xml (myhadoop3)
39
40      <build>
41          <plugins>
42              <!-- Java 1.8 -->
43              <plugin>
44                  <artifactId>maven-compiler-plugin</artifactId>
45                  <version>3.6.3</version>
46                  <configuration>
47                      <source>1.8</source>
48                      <target>1.8</target>
49                  </configuration>
50              </plugin>
51          </plugins>
52      </build>
53
54  </project>

project › build › plugins › plugin › version

7 sec, 534 ms  Cannot resolve plugin org.apache.maven.plugins:maven-compiler-plugin:3.6.3
3 sec, 777 ms
ugins:maven-compiler-p
```

尝试过的解决方法包括但不限于修改文件路径、

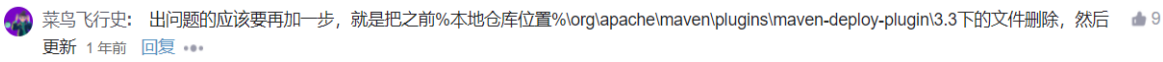




换源、



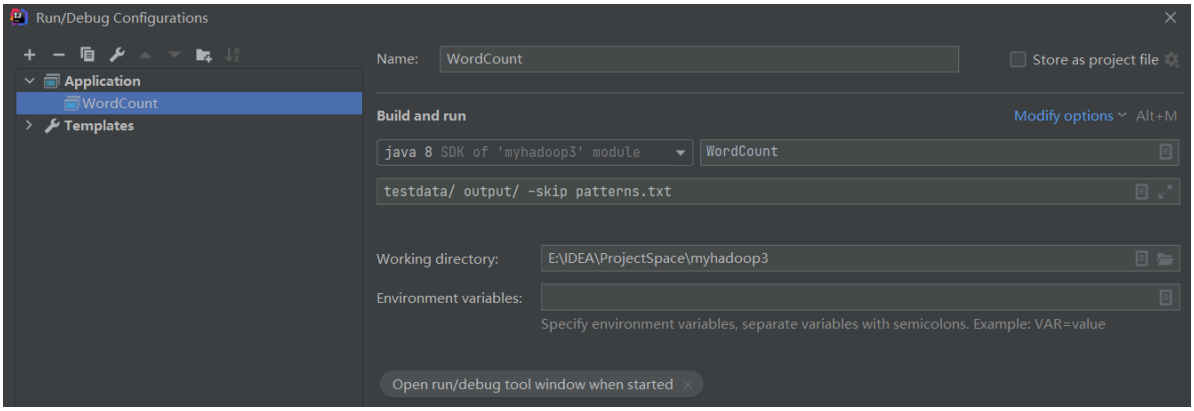
修改文件



等等，但还是怎么都搞不好。上网查了这个插件（maven-compiler-plugin）的作用后发现是用于指定JDK版本，我觉得不需要似乎也行，所以就试着直接删除了该项配置，发现后续顺利，似乎没什么影响。

④IDEA本地运行和控制台运行结果不一致

在配置完IDEA后，同学传授说不用打成jar包在命令行里运行，直接在IDEA里就可以运行，速度很快，也方便DEBUG，配置好后右键运行程序即可：



这样确实是非常方便，但需要注意的是和在控制台中运行不一样，在IDEA中运行是在本地进行操作，不涉及HDFS（除非在代码中设计指定），因此在IDEA中顺利运行后，直接打包照搬到控制台中运行会产生非常不一样的结果和报错，原因就在于在控制台中运行涉及HDFS，而在IDEA中本地运行是不涉及的，本地文件的路径、读写遍历操作等和HDFS上的操作在代码实现上很多时候是有区别的，因此报错也是非常正常的。要想在控制台和HDFS上运行，必须对代码进行修改使其适配HDFS，比如在路径的前面加上“hdfs://127.0.0.1:8900"（这是我本人的hdfs路径）等。我一开始就是在IDEA本地运行非常成功，最后却发现无法照搬到控制台运行，要修改代码的时候又很匆忙，而且还踩了很多坑，搞得非常狼狈。具体关于在java中如何实现HDFS文件的读取、下载、上传等操作，具体细节在此就不细说了。

⑤修改reduce输出格式后报错

在我修改reduce的输出格式从<Text, IntWritable>到<IntWritable, Text>后，map中context.write()报错：Type mismatch in key from map: expected org.apache.hadoop.io.IntWritable, received org.apache.hadoop.io.IntWritable。

但是我的map的输出类型和reduce的输入类型一致，为什么会报这个错？让我百思不得其解。

后来反复上网查找、尝试，发现需要添加如下两行，显式指定map的输出格式：

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
```

不添加这两行的话，即使在Mapper类和Reducer类的参数中指定一致的输出输入格式也会报错，网上有解释是当map方法中context.write()和reduce方法中context.write()输入参数类型不相同时，需要在job中设置每个方法的参数类型。

⑥用Chain实现链式MapReduce出错

上文提到我采用了两个Map-Reduce过程，并且是用的顺序链式。其实本来是打算参考上课所讲的内容、用Chain的，但是在API参数上遇到了问题。官网给出文档如下：

Method Detail

addMapper

```
public static <K1,V1,K2,V2> void addMapper(JobConf job,
                                           Class<? extends Mapper<K1,V1,K2,V2>> klass,
                                           Class<? extends K1> inputKeyClass,
                                           Class<? extends V1> inputValueClass,
                                           Class<? extends K2> outputKeyClass,
                                           Class<? extends V2> outputValueClass,
                                           boolean byValue,
                                           JobConf mapperConf)
```

Adds a Mapper class to the chain job's JobConf.

It has to be specified how key and values are passed from one element of the chain to the next, by value or by reference. If a Mapper leverages the assumed semantics that the key and values are not modified by the collector 'by value' must be used. If the Mapper does not expect this semantics, as an optimization to avoid serialization and deserialization 'by reference' can be used.

For the added Mapper the configuration given for it, `mapperConf`, have precedence over the job's JobConf. This precedence is in effect when the task is running.

IMPORTANT: There is no need to specify the output key/value classes for the ChainMapper, this is done by the addMapper for the last mapper in the chain

Parameters:

`job` - job's JobConf to add the Mapper class.

`klass` - the Mapper class to add.

`inputKeyClass` - mapper input key class.

`inputValueClass` - mapper input value class.

`outputKeyClass` - mapper output key class.

`outputValueClass` - mapper output value class.

`byValue` - indicates if key/values should be passed by value to the next Mapper in the chain, if any.

`mapperConf` - a JobConf with the configuration for the Mapper class. It is recommended to use a JobConf without default values, using the JobConf(boolean loadDefaults) constructor with FALSE.

PPT：

- MapReduce前处理和后处理步骤的链式执行
  - 设有一个完整的MapReduce作业，由Map1，Map2，Reduce，Map3，Map4构成。

```
Configuration conf = new Configuration();
Job job = new Job(conf);
job.setJobName("ChainJob");
job.setInputFormat(TextInputFormat.class);
job.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(job, in);
FileOutputFormat.setOutputPath(job, out);
JobConf  map1Conf = new JobConf(false);
ChainMapper.addMapper(job, Map1.class, LongWritable.class, Text.class,
                      Text.class, Text.class, true, map1Conf);
JobConf map2Conf = new JobConf(false);
ChainMapper.addMapper(job, Map2.class, Text.class, Text.class, LongWritable.class,
                      Text.class, true, map2Conf);
```

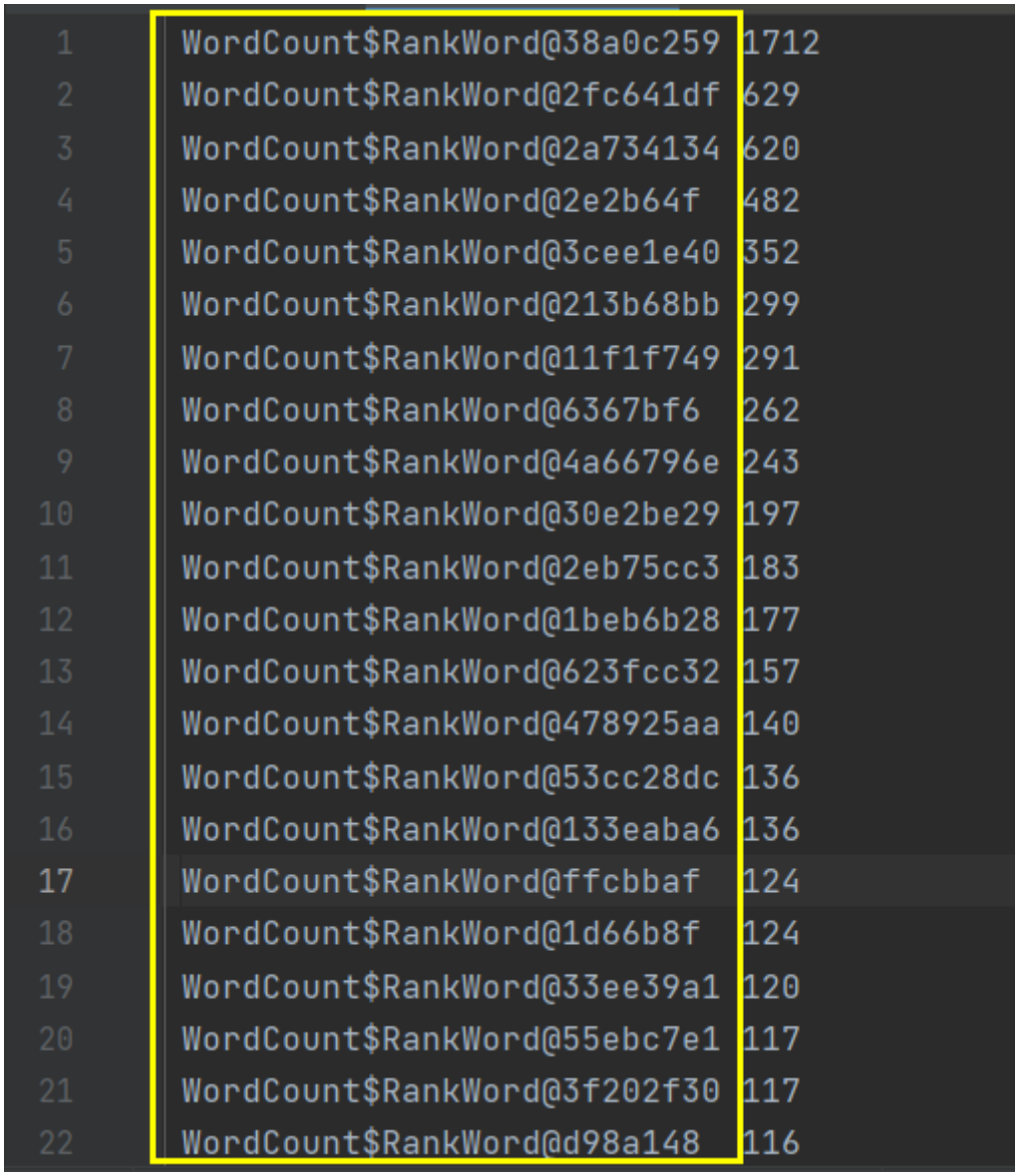
官网中第一个参数是JobConf类型，而教程中是Job类型，可能是由于版本不一致，我的代码中一开始有的也是Job类型的实例，没有JobConf类型的实例，也不知道该如何生成。不断尝试后又发现实在不知道如何从Job类的实例job获得对应的JobConf类实例，所以该参数不知道填什么……我安装的hadoop是3.3.0版本，很多以前的方法也都被弃用了（版本太新了可能也不好），上网也没查到解决方案，所以换了另一种实现方式，采用多个job顺序执行的方式（代价是增加很多IO操作，效率不高），正如上文中所描述的那样。

⑦自定义RankWord类实现Writable接口，在输出时出错

上文中提到，为了使输出符合 “排名：单词，词频”的格式，我自定义了一个RankWord类来实现Writable接口，意图是想输出RankWord类实例并使得每个RankWord类的实例都对应“排名+单词”的形式。上网查阅资料后，结合自身理解，我重写了write()方法和readFields()方法：

```
public static class RankWord implements Writable{
    private int rank;
    private String word;
    public RankWord() {
    }
    public RankWord(int r, String w) {
        this.rank = r;
        this.word = w;
    }
    public void set(int r, String w) {
        this.rank = r;
        this.word = w;
    }
    public void readFields(DataInput in) throws IOException{
        this.rank = in.readInt();
        this.word = in.readUTF();
    }
    public void write(DataOutput out) throws IOException{
        out.writeInt(this.rank);
        out.writeUTF(this.word);
    }
```

我想当然write()方法就是指定在输出文件中的输出格式。但是此时运行程序，得到的结果却不符合预期，词频能正常显示，而RankWord类的实例却没有正常写入：



上网搜索也找不到相关资料，经过自身反复不断地摸索，费了很大的功夫，最终才发现是因为还需要实现一个toString()方法！

```
public String toString() {
    return this.rank + ":" + this.word;
}
```

实现这个方法后终于可以正常显示了，但是本人还是有所不理解：按常理来说，写入文件时所遵循的格式不是应该由write()方法定义吗？如果是由toString()方法定义的话，那write()方法起的作用又是什么呢？上网查阅之后略得知涉及序列化和反序列化，但还是不甚理解，时间较为紧迫也没来得及去细究。总之，write()这个名字实在太具有误导性，搞得我想破了头也想不出来为什么输出不遵循write()方法定义的格式……

四、不足和可改进之处

此次作业，由于时间较为紧张，虽然在很多地方我都去思考了较好的解决方法，但还是有些地方受限于技术等原因，没有实现地很好。主要包括：

①链式MapReduce性能消耗大

上文中提到，我一开始是打算用Chain以降低性能消耗，但是出于技术上的原因没有实现，最终转而采用了多个MapReduce叠加的方式。这样带来的代价是增加了很多IO操作，降低了性能，实际运行中也提现了出来，这点仍然有待改进。

②for循环处理多个文件效率低

上文中提到，为了对所有文档都进行词频统计，我采用了for循环的方法，一次只有一个文档被处理，这样带来的代价是效率低，再加上采用了叠加MapReduce的方法，跑起来真是跟龟速一样。我目前想到的可能的更好的处理办法是同时输入多个文档，同时指定每一个文档由哪些Mapper节点和Reducer节点处理，以至于不同文档的词频统计不会混合起来。但是这样实现感觉技术上对本人而言有些挑战，受限于时间最终没有能够实现，仍然有待改进。

③无法传参，灵活性不够

一开始在IDEA中本地运行时，我是设置的是停词表、标点符号表、输入输出路径、中间文件路径等都是由用户传入参数控制，最终也成功实现了。但是在打包至控制台运行时发现参数传不进去，比如输入-skip xxx，结果却会把xxx识别成输出路径。由于时间紧张，本人也比较着急，所以一时就采取了将一些路径在代码中固定的做法，最终能够成功运行，但是这也是一个有待改进的缺陷。

五、参考资料/教程

https://blog.csdn.net/KingJin_CSDN_/article/details/53667319?spm=1001.2101.3001.6650.5&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EOPENSEARCH%7Edefault-5.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EOPENSEARCH%7Edefault-5.no_search_link

https://zhuanlan.zhihu.com/p/375066643

https://www.cnblogs.com/ziyaziya/p/12456586.html

https://www.cnblogs.com/yaohaitao/p/5607798.html

https://blog.csdn.net/andrewgb/article/details/50589665