191870294-朱云佳-作业5

说明：**1st jar**是第一题的**jar**包，**2nd jar**是第二题的**jar**包

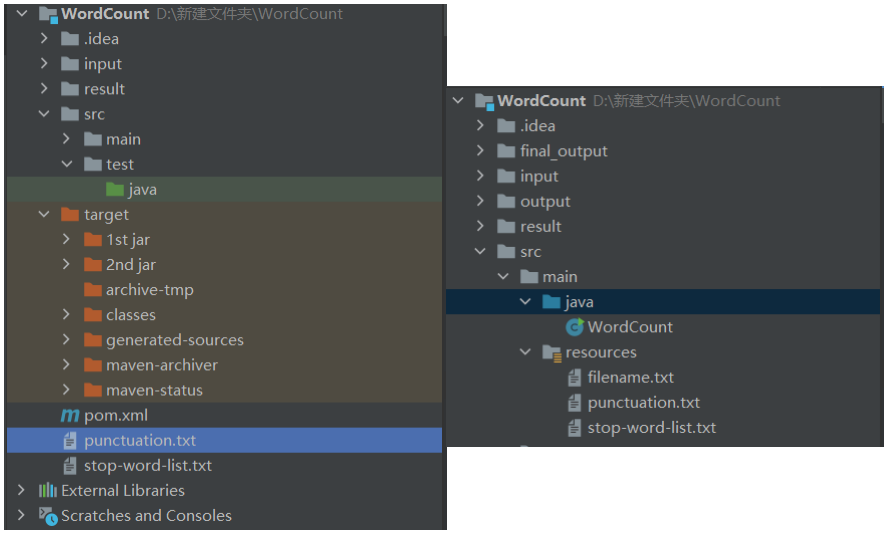
0.Intellij环境配置

Gradle or Maven?

- 在班级群公布教程之前，曾经是直接在`file->project structure`内直接导入所需要的包。但是之后与同学交流后，发现用gradle 或maven打jar包较为方便，于是调整
- gradle方面，参考了https://cloud.tencent.com/developer/article/1435044，试图创建gradle项目--但是初始创建的gradle项目并没有自行生成src文件夹（至今困惑，不知道当时为何会出现这个问题）
- 也参考了maven项目的创建教程--但苦于pom.xml的配置文件时常报错，即使install和刷新之后这些报错仍然没能消失
- 班级群教程出现之后，同学大多采用maven管理依赖包，遂转移到maven。参考https://zhuanlan.zhihu.com/p/89617163教程，将hadoop-core这个远古包注释掉，解决了一直困扰的配置文件报错问题
- 以上对于环境的探索，就持续了整整一周😥...尝试屡屡愈挫

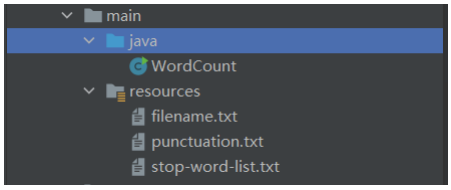
最终配置情况

最终的项目结构如下：



（右侧的output是中间结果，最终结果记录在final_output中）

上述项目结构中input文件夹是存放shakespeare的所有.txt文件的（仅用于本地intellij调试，到linux hdfs环境下会改成hdfs中的路径）



只有WordCount一个类用于实现功能，resources文件夹中存放了标点和停词文件，还有记录文件名的txt，这主要是为了方便打成jar包之后利用xx.class.getClassLoader().getResource()接口快速调取文件

其他说明

值得一提的是pom.xml的此处（完整版直接查看文件）：

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.2.0</version>
        <configuration>
            <descriptorRefs>
                <descriptorRef>
                    jar-with-dependencies
                </descriptorRef>
            </descriptorRefs>
        </configuration>
        <executions>
            <execution>
                <id>make-assembly</id>
                <phase>package</phase>
                <goals>
                    <goal>single</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
```

为了导出jar时包含plugin，参考教程加入了以上这一段，无论修改version为3.2.0或3.3.0（教程3.2.0，本机hadoop 3.3.0）都显示红色报错。但可以正常在intellij中运行，也可以打成jar包在分布式环境中运行。

1.解题思路

因为涉及到调整后排序，所以整体思路是启动两个job：第一个进行去除标点、停词、忽略大小写和wordcount计数，第二个专门做排序。第二个job的input路径是第一个job的输出路径

去标点延续课件中parseSkipFile的思路--读入命令行参数punctuation.txt，忽略大小写--用`tmpword.toLowerCase()`变小写然后写入,去除数字--直接用正则表达式搞定，如下所示：

```java
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException{
    String line=(caseSensitive)?
            value.toString():value.toString().toLowerCase();
    for (String pattern : patternsToSkip) {
        line = line.replaceAll(pattern, replacement: " ");
    }
    line=line.replaceAll( regex: "[0-9][0-9]*", replacement: " ");
    StringTokenizer itr=new StringTokenizer(line);
    while (itr.hasMoreTokens()){
        String tmpword= itr.nextToken();
        if(!stoplist.contains(tmpword.toLowerCase())&&(tmpword.length()>=3)){
            FileSplit inputSplit = (FileSplit) context.getInputSplit();
            String inputFileName = inputSplit.getPath().getName();
//            System.out.println(inputFileName);
            word.set(tmpword.toLowerCase());
            context.write(new Text( string: word.toString()+"-"+inputFileName),one);
//              context.write(new Text(word),one);
            Counter counter = context.getCounter(CountersEnum.class.getName(), CountersEnum.INPUT_WORDS.toString());
            counter.increment(1);
        }
    }
}
```

第一个job的map基本上就完成了上述三件事（当然还过滤了单词长度，并为第一、第二题不同的处理做铺垫，下面会说）

第一个job的reduce是简单计数，第一题和第二题的差别主要体现在第二个job

第一题

对于第一题，统计全部作品的单词频数，map中会仅仅以`context.write(new Text(word),one)`写入，不标记作品名称。第二个job中只需对第一个job得到的output进行统计，生成最终的final_output

关于排序

参照教程，自定义myclass类

```java
public static class myclass
            implements WritableComparable<myclass>{
        public int x;
        public String y;
        public int getX(){
            return x;
        }
        public String getY(){
            return y;
        }
        public void readFields(DataInput in) throws IOException{
            x = in.readInt();
            y = in.readUTF();
```

```java
        }
        public void write(DataOutput out) throws IOException {
            out.writeInt(x);
            out.writeUTF(y);
        }
        public int compareTo(myclass p) {
            if (this.x > p.x) {
                return -1;
            } else if (this.x < p.x) {
                return 1;
            } else {
                if (this.getY().compareTo(p.getY()) < 0) {
                    return -1;
                } else if (this.getY().compareTo(p.getY()) > 0) {
                    return 1;
                } else {
                    return 0;
                }

            }
        }
    }
```

第二个job中map输出的key类型是myclass，其中x和y属性分别存储出现频次和单词。通过自定义compareTo函数，无需reverse类即可将单词按出现频次--字典序进行排列

```java
public static class TokenizerMapperNew
        extends Mapper<Object, Text, myclass, IntWritable> {
    private final IntWritable valueInfo = new IntWritable();
    @Override
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
//          StringTokenizer itr = new StringTokenizer(value.toString());
        String[] word = value.toString().split( regex: "\\s+");
        myclass keyInfo = new myclass();
        keyInfo.x = Integer.parseInt(word[word.length - 1]);
        keyInfo.y = word[word.length - 2];
        valueInfo.set(Integer.parseInt(word[word.length - 1]));
        context.write(keyInfo, valueInfo);
        System.out.println(keyInfo.y);
    }
}
```

在最后的reduce输出上，只输出前100个，所以做如下的设计

```
if (!title){
    context.write(new Text("总计: "),new IntWritable());
    title=true;
}
keyInfo.set(key.y);
valueInfo.set(key.x);
if (sortcounter<=99) {
    context.write(new Text(String.valueOf(sortcounter+1)+" "+keyInfo),valueInfo);
    sortcounter+=1;
}
```

title为boolean变量，判断是否已经写入标题，其余定义sortcounter进行计数--当输出达到100个则停止

第二题

第二问的与第一问的不同之处在于，在第一个job写入时，需要加入文件名属性：

```
        while (itr.hasMoreTokens()){
            String tmpword= itr.nextToken();
            if(!stoplist.contains(tmpword.toLowerCase())&&(tmpword.length()>=3)){
                FileSplit inputSplit = (FileSplit) context.getInputSplit();
                String inputFileName = inputSplit.getPath().getName();
//              System.out.println(inputFileName);
                word.set(tmpword.toLowerCase());
                context.write(new Text( string: word.toString()+"-"+inputFileName),one);
```

在处理第一个job得到的结果时，需要依据输出属性，对首个"-"后面的文件名称进行解析。定义WordCountPartioner对单词进行分区

```
public static class WordCountPartitioner extends Partitioner<myclass, IntWritable> {
    @Override
    public int getPartition(myclass keyInfo, IntWritable value, int numPartitions) {

        String raw_word=keyInfo.y.toString();
        String need_word=raw_word.substring(0,raw_word.indexOf("-"));
        String file_name=raw_word.substring(raw_word.indexOf("-")+1);
        int res=0;
        for(int i=0;i< filenamelist.size();i++){
            if(file_name.equals(filenamelist.get(i))){
                res=i;
                break;
            }
        }
        return res;
    }
}
```

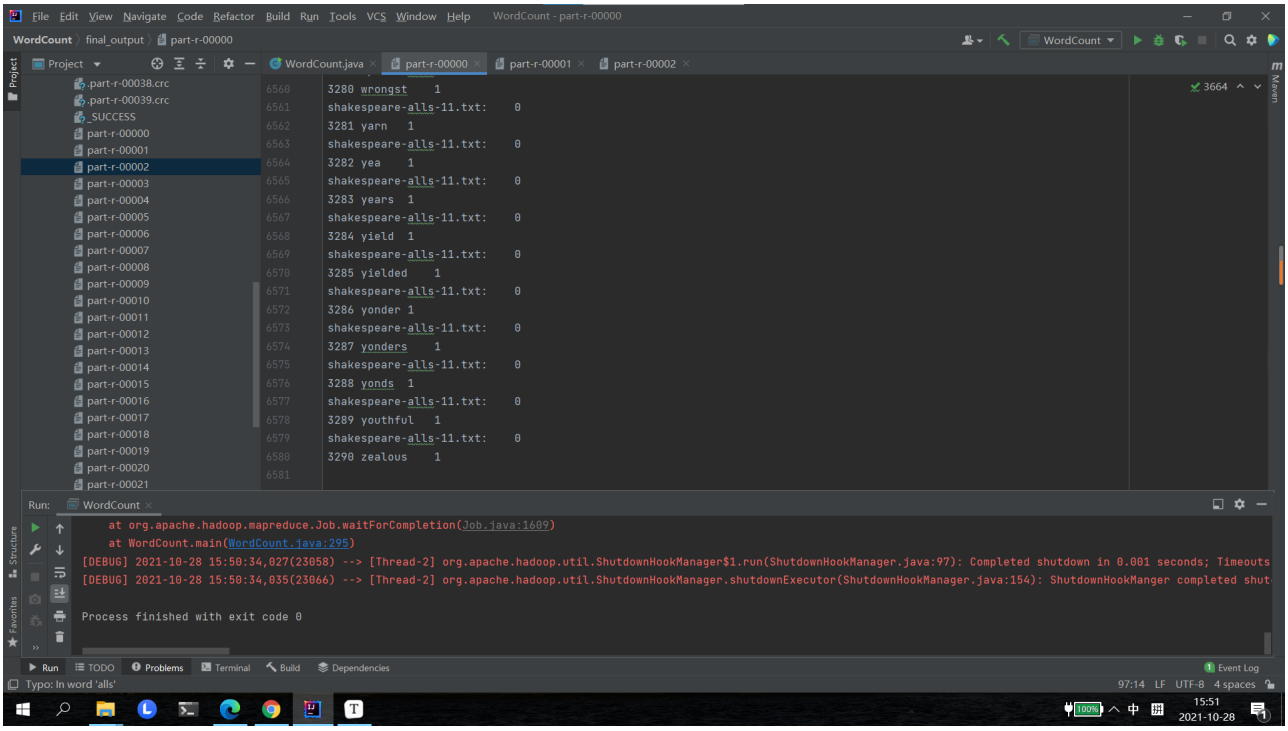
这里用到的filename是读取的文件名list，回顾setup函数：

```
if(!readfilename){
    //读取input文件中的文件名，应该也要做修改
      File inputfile=new File("input");
      File[] inputfilelist=inputfile.listFiles();
      assert inputfilelist != null;
      for (File file : inputfilelist) {
          filenamelist.add(file.getName());
      }
    InputStream is_file= Objects.requireNonNull(WordCount.class.getClassLoader().getResource( name: "filename.txt")).open
    Reader reader = new InputStreamReader(is_file);
    BufferedReader stopbuffer=new BufferedReader(reader);
    String tmp=null;
    while ((tmp=stopbuffer.readLine())!=null){
        String[] ss =tmp.split( regex: " ");
        filenamelist.addAll(Arrays.asList(ss));
    }
    stopbuffer.close();
    readfilename=true;
}
```

分区完毕后进入reduce：仍然像partitioner中一样，解析得到need_word（真正需要统计的单词）

```
    for(int i=0;i<filenamelist.size();i++){
        if(file_name.equals(filenamelist.get(i))){
            cur_process_num=i;
            break;
        }
    }
    if(cur_process_num==should_process_num){
        if (!title){
              context.write(new Text("总计: "),new IntWritable());
            context.write(new Text( string: file_name+":"),new IntWritable());
            title=true;
        }
        keyInfo.set(key.y);
        valueInfo.set(key.x);
        if (sortcounter<=99) {
            context.write(new Text( string: String.valueOf(sortcounter+1)+" "+need_word),valueInfo);
            sortcounter+=1;
        }
        else {
            should_process_num+=1;
            sortcounter=0;
            title=false;
        }
```

这里定义了cur_process_num（目前遍历的单词位于哪一个文件）和should_process_num（应该输出的单词位于哪一个文件），当两者保持一致时才会输出。这是基于我在debug时的一个发现：



分区后，对文件单词的统计似乎是逐一进行的（通过println序号判断），由此产生了上面的想法

reduce过后写入结果，两题的解答到此也就完成了！

2.实验结果

本地

本地intellij实验的结果位于result文件夹，以下选取截图：

```
shakespeare-alls-11.txt:    0
1 lord  213
2 parolles  177
3 bertram   137
4 king  136
5 helena    125
6 lafeu 117
7 shall 115
8 thou  100
9 countess  99
10 sir  97
11 good 90
12 know 87
13 thy  84
14 thee 79
15 second   70
16 clown    67
17 love 65
18 say  62
19 diana    58
20 enter    57
21 tis  53
22 hath 51
23 let  50
24 make 49
25 soldier  49
26 come 47
```

WordCount.java | part-r-00000 | part-r-00000-total | pom.xml (WordCount)

```
总计:    0
1 thou  5589
2 thy   4004
3 shall 3536
4 thee  3204
5 lord  3134
6 king  3101
7 sir   2976
8 good  2837
9 come  2492
10 let  2317
11 love 2285
12 enter    2257
13 man  1977
14 hath 1931
15 like 1893
16 know 1764
17 say  1698
18 make 1676
19 did  1670
20 tis  1392
21 speak    1189
22 time 1181
23 tell 1086
24 heart    1083
25 henry    1076
26 duke 1075
```

WordCount.java | part-r-00000 | part-r-00000-total

```
79 fortune 19
80 france 19
81 truth 19
82 virginity 19
83 art 18
84 comes 18
85 death 18
86 fair 18
87 farewell 18
88 gone 18
89 heart 18
90 noble 18
91 old 18
92 sweet 18
93 bring 17
94 business 17
95 court 17
96 himself 17
97 lordship 17
98 majesty 17
99 master 17
100 till 17
```

WordCount.java | part-r-00000 | part-r-00000-total | pom.

```
79 anthony 562
80 grace 561
81 bear 559
82 house 556
83 dead 551
84 gloucester 535
85 richard 534
86 live 515
87 thing 514
88 wife 513
89 brutus 509
90 eye 506
91 word 506
92 mark 505
93 peace 505
94 head 504
95 little 500
96 john 498
97 hamlet 494
98 fool 493
99 madam 488
100 thine 485
```

分布式环境

第一题

zyj@LAPTOP-T1OKOQBM: ~/ha

```
                INPUT_WORDS=422310
        File Input Format Counters
                Bytes Read=5020327
        File Output Format Counters
                Bytes Written=245764
2021-10-30 16:54:09,360 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2021-10-30 16:54:09,369 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement
the Tool interface and execute your application with ToolRunner to remedy this.
2021-10-30 16:54:09,376 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
zyj/.staging/job_1635432422647_0024
2021-10-30 16:54:09,540 INFO input.FileInputFormat: Total input files to process : 1
2021-10-30 16:54:09,578 INFO mapreduce.JobSubmitter: number of splits:1
2021-10-30 16:54:09,619 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1635432422647_0024
2021-10-30 16:54:09,619 INFO mapreduce.JobSubmitter: Executing with tokens: []
2021-10-30 16:54:09,642 INFO impl.YarnClientImpl: Submitted application application_1635432422647_0024
2021-10-30 16:54:09,648 INFO mapreduce.Job: The url to track the job: http://LAPTOP-T1OKOQBM.localdomain:8088/proxy/appl
ication_1635432422647_0024/
2021-10-30 16:54:09,648 INFO mapreduce.Job: Running job: job_1635432422647_0024
2021-10-30 16:54:21,739 INFO mapreduce.Job: Job job_1635432422647_0024 running in uber mode : false
2021-10-30 16:54:21,740 INFO mapreduce.Job:  map 0% reduce 0%
2021-10-30 16:54:25,761 INFO mapreduce.Job:  map 100% reduce 0%
2021-10-30 16:54:29,777 INFO mapreduce.Job:  map 100% reduce 100%
2021-10-30 16:54:29,783 INFO mapreduce.Job: Job job_1635432422647_0024 completed successfully
2021-10-30 16:54:29,807 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=452078
                FILE: Number of bytes written=1432525
                FILE: Number of read operations=0
```

Browse Directory

/user/zyj/final_output Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	zyj	supergroup	0 B	Oct 30 16:54	1	128 MB	_SUCCESS
-rw-r--r--	zyj	supergroup	1.26 KB	Oct 30 16:54	1	128 MB	part-r-00000

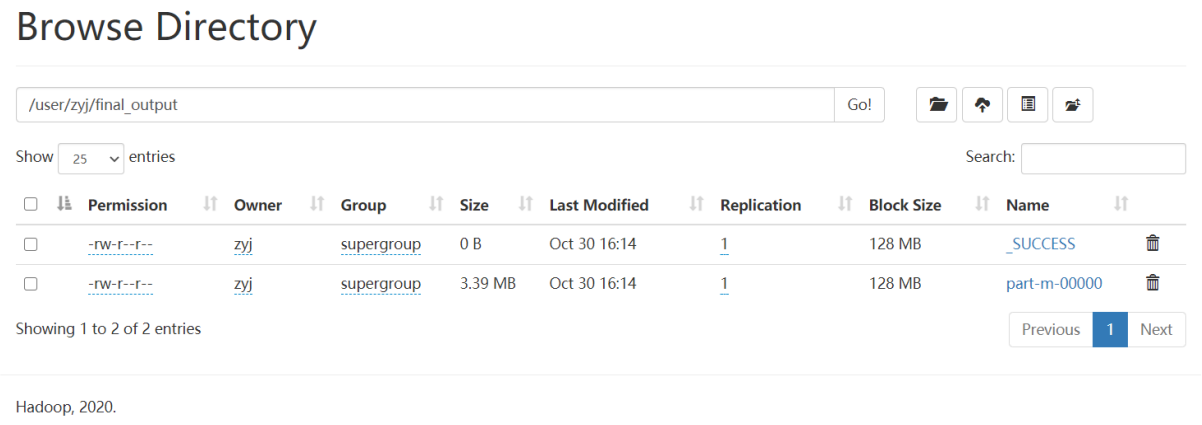
Showing 1 to 2 of 2 entries Previous 1 Next

Hadoop, 2020.

```
zyj@LAPTOP-T1OKOQBM:~/hadoop/hadoop-3.3.0/bin$ ./hdfs dfs -cat ./final_output/part-r-00000
总计:   0
1 thou  5589
2 thy   4004
3 shall 3536
4 thee  3204
5 lord  3134
6 king  3101
7 sir   2976
8 good  2837
9 come  2492
10 let  2317
11 love 2285
12 enter        2257
13 man  1977
14 hath 1931
15 like 1893
16 know 1764
17 say  1698
18 make 1676
19 did  1670
20 tis  1392
21 speak        1189
22 time 1181
23 tell 1086
24 heart        1083
25 henry        1076
26 duke 1075
```

第二题

```
zyj@LAPTOP-T1OKOQBM: ~/ha
2021-10-30 16:14:01,272 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2021-10-30 16:14:01,607 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
zyj/.staging/job_1635432422647_0021
2021-10-30 16:14:01,912 INFO input.FileInputFormat: Total input files to process : 40
2021-10-30 16:14:01,971 INFO mapreduce.JobSubmitter: number of splits:40
2021-10-30 16:14:02,057 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1635432422647_0021
2021-10-30 16:14:02,057 INFO mapreduce.JobSubmitter: Executing with tokens: []
2021-10-30 16:14:02,180 INFO conf.Configuration: resource-types.xml not found
2021-10-30 16:14:02,180 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2021-10-30 16:14:02,237 INFO impl.YarnClientImpl: Submitted application application_1635432422647_0021
2021-10-30 16:14:02,265 INFO mapreduce.Job: The url to track the job: http://LAPTOP-T1OKOQBM.localdomain:8088/proxy/appl
ication_1635432422647_0021/
2021-10-30 16:14:02,266 INFO mapreduce.Job: Running job: job_1635432422647_0021
2021-10-30 16:14:08,343 INFO mapreduce.Job: Job job_1635432422647_0021 running in uber mode : false
2021-10-30 16:14:08,344 INFO mapreduce.Job:  map 0% reduce 0%
2021-10-30 16:14:15,417 INFO mapreduce.Job:  map 15% reduce 0%
2021-10-30 16:14:19,445 INFO mapreduce.Job:  map 22% reduce 0%
2021-10-30 16:14:20,450 INFO mapreduce.Job:  map 30% reduce 0%
2021-10-30 16:14:23,480 INFO mapreduce.Job:  map 32% reduce 0%
2021-10-30 16:14:24,488 INFO mapreduce.Job:  map 38% reduce 0%
2021-10-30 16:14:25,494 INFO mapreduce.Job:  map 43% reduce 0%
2021-10-30 16:14:26,499 INFO mapreduce.Job:  map 45% reduce 0%
2021-10-30 16:14:27,503 INFO mapreduce.Job:  map 50% reduce 0%
2021-10-30 16:14:29,513 INFO mapreduce.Job:  map 55% reduce 0%
2021-10-30 16:14:30,518 INFO mapreduce.Job:  map 57% reduce 0%
2021-10-30 16:14:31,523 INFO mapreduce.Job:  map 63% reduce 0%
2021-10-30 16:14:33,532 INFO mapreduce.Job:  map 68% reduce 0%
2021-10-30 16:14:34,537 INFO mapreduce.Job:  map 70% reduce 0%
2021-10-30 16:14:35,541 INFO mapreduce.Job:  map 75% reduce 0%
2021-10-30 16:14:36,550 INFO mapreduce.Job:  map 75% reduce 25%
2021-10-30 16:14:37,556 INFO mapreduce.Job:  map 80% reduce 25%
2021-10-30 16:14:38,561 INFO mapreduce.Job:  map 82% reduce 25%
2021-10-30 16:14:39,567 INFO mapreduce.Job:  map 88% reduce 25%
2021-10-30 16:14:41,576 INFO mapreduce.Job:  map 93% reduce 25%
2021-10-30 16:14:42,580 INFO mapreduce.Job:  map 95% reduce 30%
2021-10-30 16:14:43,584 INFO mapreduce.Job:  map 100% reduce 30%
2021-10-30 16:14:44,588 INFO mapreduce.Job:  map 100% reduce 100%
2021-10-30 16:14:44,594 INFO mapreduce.Job: Job job_1635432422647_0021 completed successfully
```



正确的！与本地相符，泪目！

3.实验遇到的问题及解决方法

本地实验遇到的问题

-

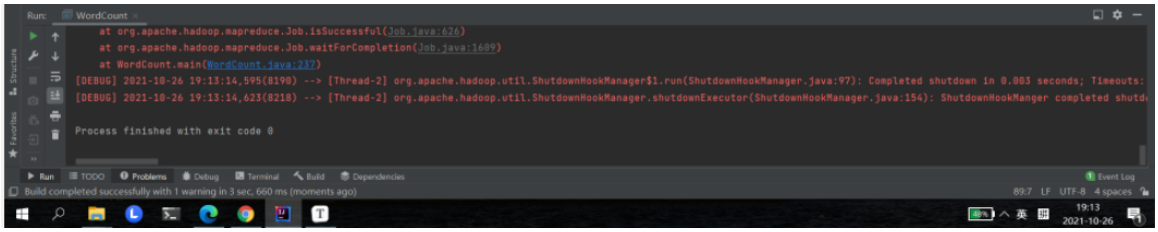
2013年的教程在疯狂传阅。他们的配置文件为：

```
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-core</artifactId>
<version>1.0.3</version>
</dependency>

<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.4</version>
<scope>test</scope>
</dependency>
```

针对这种hadoop-core为1.0.3版本，并且去maven repository 查看也是在13年就从未更新的jar包。想想还是不用为好。

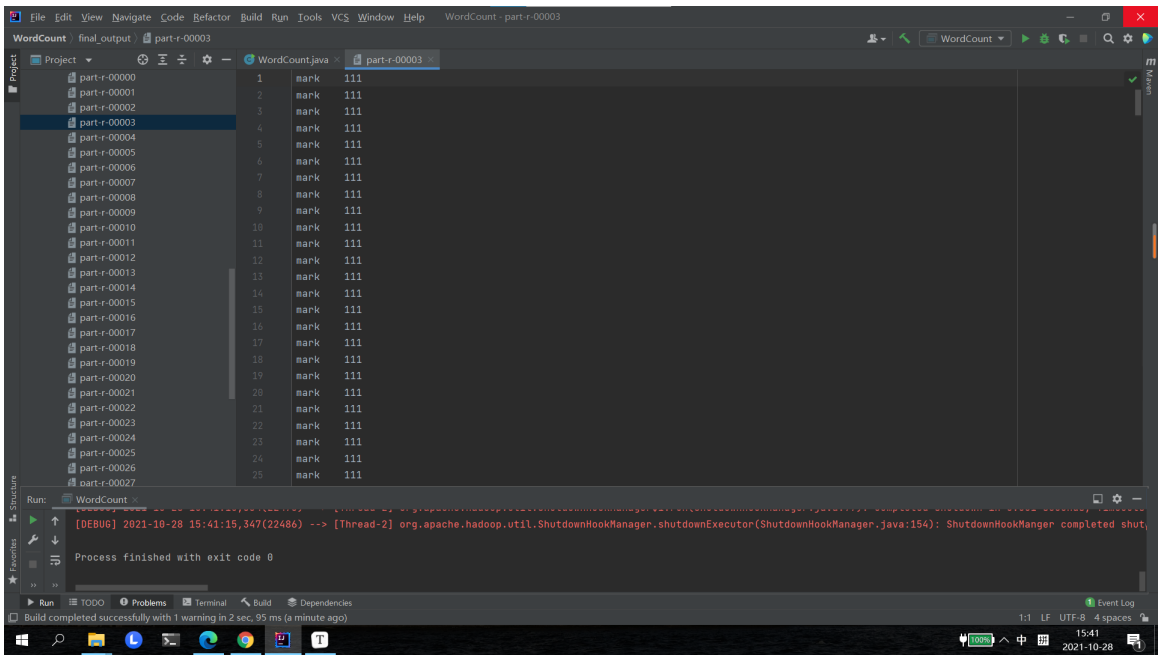
配置intellij环境的问题--上面已提及，此处不多赘述，耗费了接近一周的时间调试环境，惨痛教训！

- 

  我的log4j开始没有.properties文件，每次warning，按照网上教程https://www.jianshu.com/p/ccafda45bcea配置过后，变成一条条的细致记录输出（所以debug还是主要依靠println，目前这个问题应该是暂时没有接近--也没有发现遇到相近问题的同学）

- 开始发现分区过后，只能写入part-r-00000这一个文件的前100，其他统统为空

  思索良久--推测是100的个数限制导致的，随便写入了一些内容，确实验证了想法



解决办法就是进行partitioner输出实验，发现逐个文件计数的原理后，设计成到100个就停--强行切换到下个文件

```
//          得到现在在第几个
            for(int i=0;i<filenamelist.size();i++){
                if(file_name.equals(filenamelist.get(i))){
                    cur_process_num=i;
                    break;
                }
            }
            if(cur_process_num==should_process_num){
                if (!title){
 //                   context.write(new Text("总计: "),new IntWritable());
                    context.write(new Text( string: file_name+":"),new IntWritable());
                    title=true;
                }
                keyInfo.set(key.y);
                valueInfo.set(key.x);
                if (sortcounter<=99) {
                    context.write(new Text( string: String.valueOf(sortcounter+1)+" "+need_word),valueInfo);
                    sortcounter+=1;
                }
                else {
                    should_process_num+=1;
```

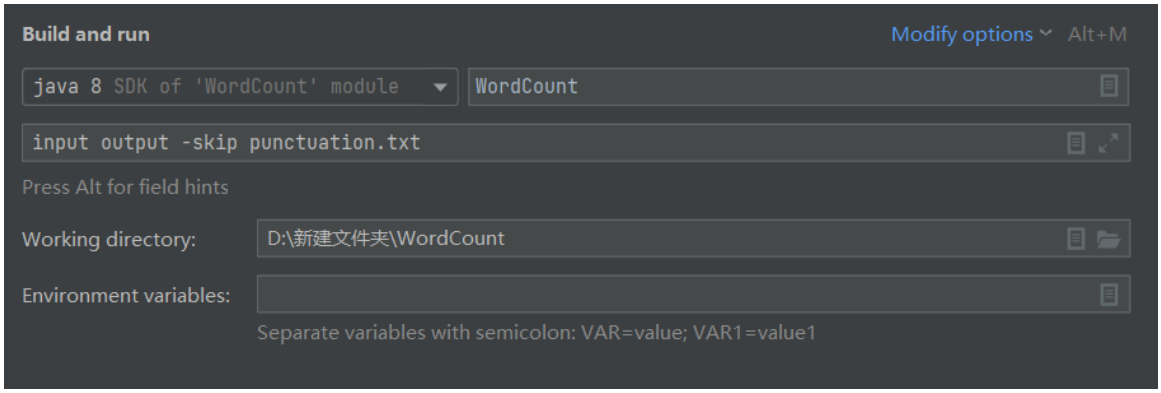
分布式遇到的问题

- 运行jar的时候，多次报错---已经检查了路径名称，和实验2是类似的操作

```
zyj@LAPTOP-T1OKOQBM:~/hadoop/hadoop-3.3.0/bin$ ./hadoop jar /home/zyj/HW5/WordCount-1.0-SNAPSHOT.jar /local/input output
 -skip /local/punctuation.txt
Exception in thread "main" java.lang.ClassNotFoundException: /local/input
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:348)
        at org.apache.hadoop.util.RunJar.run(RunJar.java:316)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:236)
```

  解决方法：增加了主类名称，已解决问题

- 实现读取stop-word-list，原因是我在IDE中写死了这个参数，只有以下是命令行传入的



  解决方法：通过xx.class.getClassLoader().getResource()接口快速调取文件，同时把要用的文件放到resources文件夹下

- 在运行时又爆出的错误：

```
Error: java.lang.RuntimeException: java.lang.ClassNotFoundException: Class WordCount$TokenizerMapper1 not found
        at org.apache.hadoop.conf.Configuration.getClass(Configuration.java:2665)
        at org.apache.hadoop.mapreduce.task.JobContextImpl.getMapperClass(JobContextImpl.java:187)
        at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:759)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:347)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:178)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1845)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:172)
Caused by: java.lang.ClassNotFoundException: Class WordCount$TokenizerMapper1 not found
```

且发现此时中间文件output已经可以显示！

```
zyj@LAPTOP-T1OKOQBM: ~/ha
zeal-shakespeare-merchant-5.txt 1
zeal-shakespeare-much-3.txt     1
zeal-shakespeare-second-52.txt  3
zeal-shakespeare-third-53.txt   1
zeal-shakespeare-timon-49.txt   2
zeal-shakespeare-titus-50.txt   1
zeal-shakespeare-tragedy-57.txt 2
zeal-shakespeare-tragedy-58.txt 3
zeal-shakespeare-troilus-22.txt 2
zeal-shakespeare-two-18.txt     1
zeal-shakespeare-winters-19.txt 1
zealous-shakespeare-alls-11.txt 1
zealous-shakespeare-life-56.txt 2
zealous-shakespeare-loves-8.txt 1
zealous-shakespeare-sonnets-59.txt      1
zealous-shakespeare-tragedy-58.txt      1
zeals-shakespeare-timon-49.txt  1
zed-shakespeare-king-45.txt     1
zelous-shakespeare-sonnets.txt  1
zenelophon-shakespeare-loves-8.txt      1
zenith-shakespeare-tempest-4.txt        1
zephyrs-shakespeare-cymbeline-17.txt    1
zir-shakespeare-king-45.txt     2
zodiac-shakespeare-titus-50.txt 1
zodiacs-shakespeare-measure-13.txt      1
zone-shakespeare-hamlet-25.txt  1
zounds-shakespeare-first-51.txt 10
zounds-shakespeare-life-56.txt  1
zounds-shakespeare-othello-47.txt       3
zounds-shakespeare-romeo-48.txt 2
zounds-shakespeare-titus-50.txt 1
zounds-shakespeare-tragedy-58.txt       4
zwaggered-shakespeare-king-45.txt       1
zyj@LAPTOP-T1OKOQBM:~/hadoop/hadoop-3.3.0/bin$
```

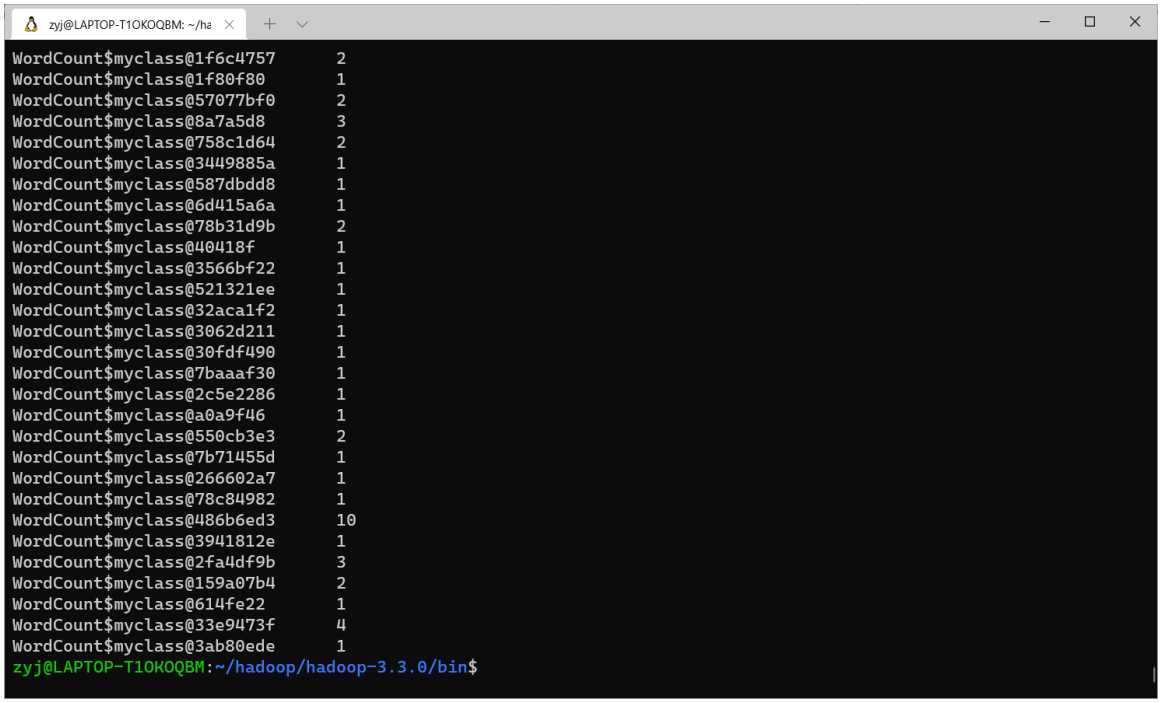
解决方法：

```java
if(job.waitForCompletion( verbose: true)){
    Configuration conf1 = new Configuration();
    Job job2 = Job.getInstance(conf1,  jobName: "sort");
    job2.setJarByClass(WordCount.class);
    job2.setMapperClass(TokenizerMapperNew.class);
    job2.setPartitionerClass(WordCountPartitioner.class);
    job2.setNumReduceTasks(filenamelist.size());
    job2.setReducerClass(IntSumReducer1.class);
    job2.setMapOutputKeyClass(myclass.class);
    job2.setMapOutputValueClass(IntWritable.class);
    job2.setOutputKeyClass(Text.class);
```

4.可能的改进之处

- 了解到部分同学采用的是提高排序函数的功能，实现在一个文件中写入所有分作品的词频统计，我觉得十分值得尝试--避免了partitioner分区，也就不会生成40个统计文件，极大地节约了hdfs资源（在运行的时候还爆出了如下错误，怀疑是资源不

足所致，最后修改了yarn-site.xml文件）



- 此次实验的排序算法可以改进，曾经看到有的教程提出用reversemapper类或treemap算法实现，但最后没能一一尝试
- 在文件读取方面，虽然最后采用了getResource()方法解决了问题（思考到几乎最后时刻...），但有同学是采用了命令行读取参数的方法，更加灵活--应该反思，改进读取文件的方式（因为最后代码主体已经完成了，就不太敢对setup等处读取的方式做修改了）

总结

本次作业是发下来就开始研究的--前前后后做了有2周时间，现在甚至于要卡ddl才能完成😖，暂时得出如下的体会吧：

- 最先开始探索总是不易的，因为可以参考的不多，要一个人摸索--和同学的有效交流能够避免走一个人的弯路
- 要提升资料搜集能力--这也是解决问题能力的一部分（发现一周后用bing搜比之前搜到了更加满意的答案）
- 需要投入整块的时间思考问题，否则不容易有思路

接下来的策略应该要有所调整了。