因为没有使用Maven管理项目，就没照着老师给的pdf中要求的提交格式交了

output文件夹中是输出文件，code文件夹中是代码文件（包括.java和.jar）

0 运行流程介绍

0.1 数据集准备

先查看所有容器 `sudo docker ps`（获得容器id）

```
chenmiao@chenmiao-virtual-machine:~$ sudo docker ps
CONTAINER ID   IMAGE        COMMAND       CREATED      STATUS          PORTS
                                                                     NAMES
cb46a3dead24   newuhadoop   "/bin/bash"   8 days ago   Up 42 seconds
                                                                     h05
2758f3a05bf8   newuhadoop   "/bin/bash"   8 days ago   Up 42 seconds
                                                                     h04
e45c02dda942   newuhadoop   "/bin/bash"   8 days ago   Up 43 seconds
                                                                     h03
1d8cd276f450   newuhadoop   "/bin/bash"   8 days ago   Up 43 seconds
                                                                     h02
63f0da8c80fc   newuhadoop   "/bin/bash"   8 days ago   Up 44 seconds   0.0.0.0:8088->8088/t
cp, :::8088->8088/tcp, 0.0.0.0:9870->9870/tcp, :::9870->9870/tcp   h01
```

将数据集传输到docker容器（h01）

`sudo docker cp <需要传输的文件路径> <容器id>:<传输文件在容器中的存放位置>`

```
chenmiao@chenmiao-virtual-machine:~$ sudo docker cp ~/financial_big_data/homework5/analyst_
ratings.csv 63f0da8c80fc:/tmp
Successfully copied 52.5MB to 63f0da8c80fc:/tmp
chenmiao@chenmiao-virtual-machine:~$ sudo docker cp ~/financial_big_data/homework5/stop-wor
d-list.txt 63f0da8c80fc:/tmp
Successfully copied 4.1kB to 63f0da8c80fc:/tmp
```

将数据集放到hdfs上（需要先在hdfs新建一个input文件夹）然后在usr/local/hadoop目录下执行：

`./bin/hdfs dfs -put <数据集在docker中存放的位置> /input/`

可以通过`./bin/hdfs dfs -ls /input`查看是否放好了

```
root@h01:/usr/local/hadoop# ./bin/hdfs dfs -put /tmp/analyst_ratings.csv /input/
root@h01:/usr/local/hadoop# ./bin/hdfs dfs -ls /input
Found 1 items
-rw-r--r--   2 root supergroup   52462980 2024-10-22 13:53 /input/analyst_rating
s.csv
root@h01:/usr/local/hadoop# ./bin/hdfs dfs -put /tmp/stop-word-list.txt /input/
root@h01:/usr/local/hadoop# ./bin/hdfs dfs -ls /input
Found 2 items
-rw-r--r--   2 root supergroup   52462980 2024-10-22 13:53 /input/analyst_rating
s.csv
-rw-r--r--   2 root supergroup       2231 2024-10-22 13:55 /input/stop-word-list
.txt
root@h01:/usr/local/hadoop#
```

0.2 运行代码准备

同理，先把java代码文件放到docker（这里我直接放到了hadoop文件夹下）

```
chenmiao@chenmiao-virtual-machine:~$ sudo docker cp ~/financial_big_data/homework5/StockCou
nt.java 63f0da8c80fc:/usr/local/hadoop
[sudo] password for chenmiao:
Successfully copied 5.63kB to 63f0da8c80fc:/usr/local/hadoop
```

然后生成相应的.class文件

```
javac -classpath `hadoop classpath` StockCount.java
```

再生成相应的.jar

```
jar -cvf stockcount.jar StockCount*.class
```

0.3 MapReduce运行

```
# 把结果保存到/output/<新建文件夹>
./bin/hadoop jar stockcount.jar StockCount /input/analyst_ratings.csv /output/<文件夹名>
# 查看
./bin/hadoop fs -ls /output/<文件夹名>
# 打印输出
./bin/hadoop fs -cat <输出文件的路径>
```

0.4 把docker中的文件传回

先在容器中运行以下命令，将文件从HDFS下载到容器的本地目录（代码文件不需要这步，输出文件需要）

`./bin/hdfs dfs -get /output/stock_count/part-r-00000 <需要在docker中存放的路径>`

再将该文件从 Docker 容器传输到本机

`docker cp <容器id>:<文件在docker中存放的位置> <文件在本机需要存储的位置>`

0.5 遇到错误

生成.java文件相应的.class文件时，遇到多次报错。

错误1： `StockCount.java:19: error: unmappable character for encoding ASCII // ??????????????????????? ^`

原因：代码文件中包含非ASCII编码的字符，编译器默认使用 ASCII编码，无法识别这些字符。

解决方法：指定编码方式

```
javac -encoding UTF-8 -classpath `hadoop classpath`StockCount.java
```

错误2： `StockCount.java:2: error: package org.apache.hadoop.conf does not exist import org.apache.hadoop.conf.Configuration; ^`

原因：找不到 Hadoop的核心类库。

解决方法：

直接输入`/usr/local/hadoop/bin/hadoop classpath`，会得到输出hadoop的classpath，然后直接把它粘贴到命令行里即可，最终完整可用的命令如下

```
javac -encoding UTF-8 -classpath "/usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/*:/usr/local/hadoop/share/hadoop/common/*:/usr/local/hadoop/share/hadoop/hdfs:/usr/local/hadoop/share/hadoop/hdfs/lib/*:/usr/local/hadoop/share/hadoop/hdfs/*:/usr/local/hadoop/share/hadoop/mapreduce/*:/usr/local/hadoop/share/hadoop/yarn:/usr/local/hadoop/share/hadoop/yarn/lib/*:/usr/local/hadoop/share/hadoop/yarn/*" StockCount.java
```

1 StockCount

要求：在HDFS上加载上市公司热点新闻标题数据集（analyst_ratings.csv），该数据集收集了部分上市公司的热点财经新闻标题。编写MapReduce程序完成以下任务：统计数据集上市公司股票代码（“stock”列）的出现次数，按出现次数从大到小输出，输出格式为“<排名>：<股票代码>，<次数>”。

1.1 设计思路

最直观的想法是先将需要的股票代码（stock列）提取出来，维护一个含股票代码和出现次数两个字段的列表，对重复出现的股票代码进行次数累加。最后做排序和格式的修改。

- Mapper

  定义一个StockMapper类，逐行处理数据，使用split方法切片，提取第四个字段stock，然后将每个股票代码视为出现了1次，输出键值对`<股票代码，1>`。

- Reducer

  定义一个StockReducer类，接收来自Mapper的输出，累加相同股票代码的出现次数，得到每个股票代码的总出现次数；并维护一个列表存储每个股票代码及其出现次数。

  对列表按照股票代码出现次数进行降序排序，遍历排序后的列表，为每个股票代码生成一个排名，并按照指定的格式输出结果。

- main

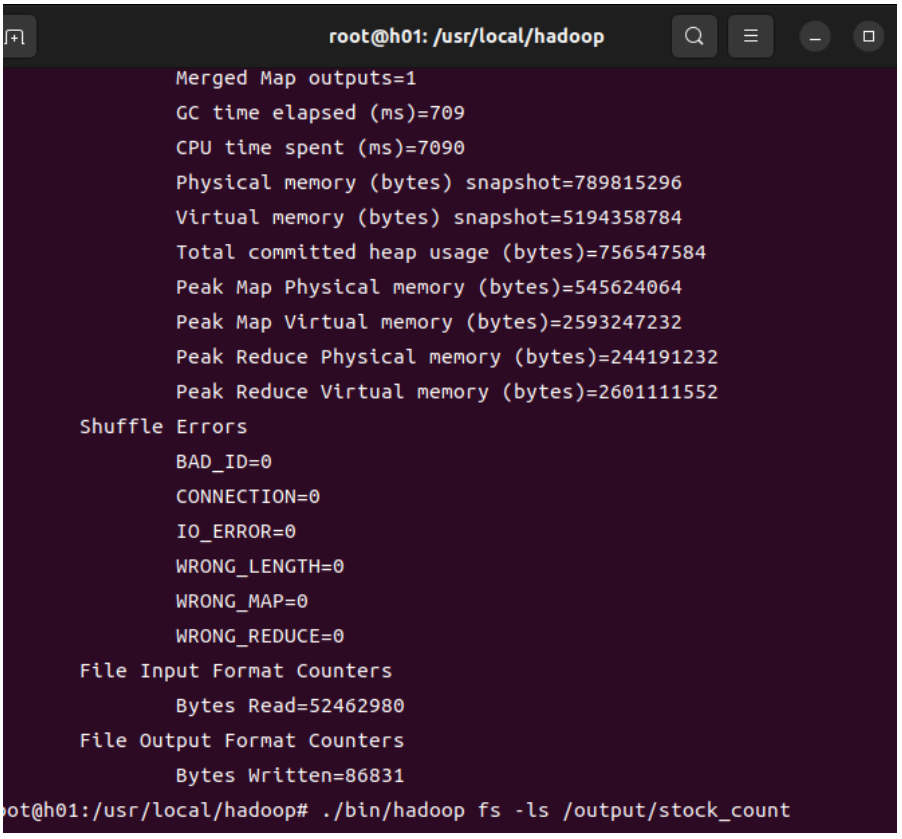
  设置MapReduce作业的配置：配置Mapper和Reducer类；指定输入、输出文件路径（args[0]和 args[1]）；启动作业并等待作业完成。

1.2 程序运行结果

- 终端输出结果

  运行MapReduce结果

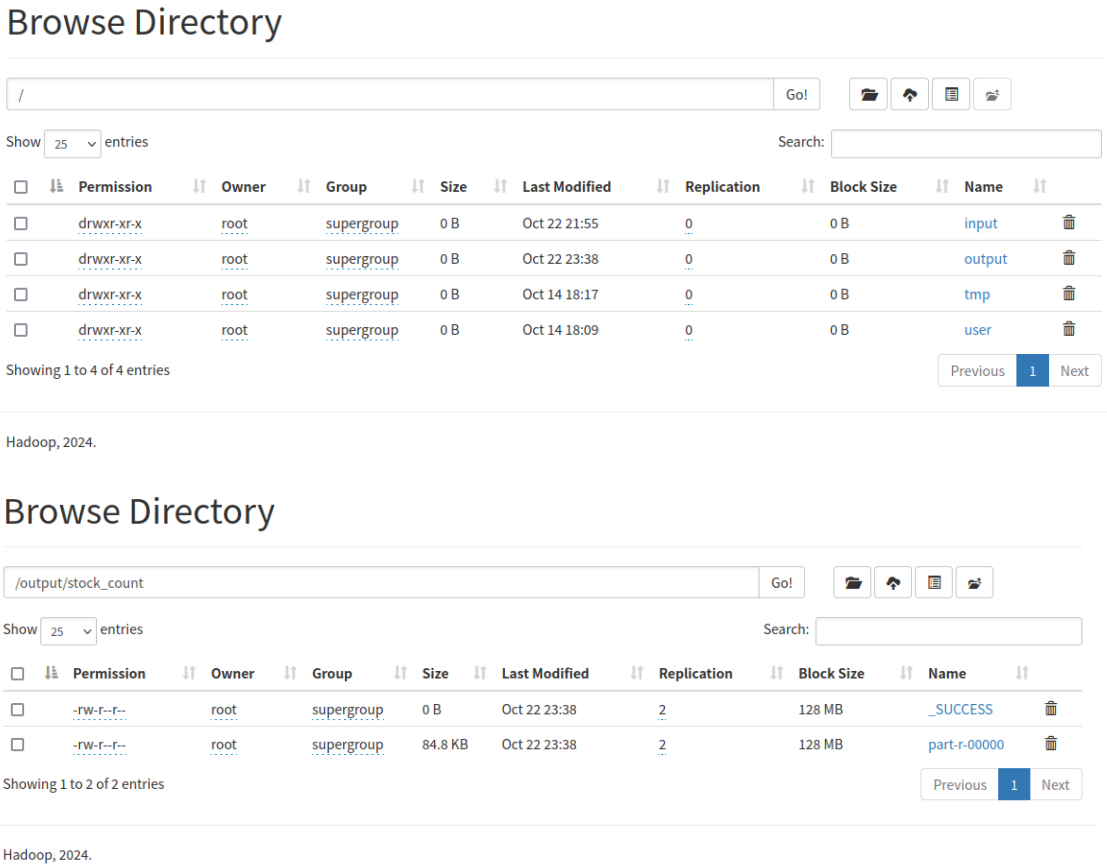
```
root@h01:/usr/local/hadoop# ./bin/hadoop jar stockcount.jar StockCount /input/a
nalyst_ratings.csv /output/stock_count
2024-10-22 15:38:13,787 INFO client.DefaultNoHARMFailoverProxyProvider: Connect
ing to ResourceManager at h01/172.18.0.2:8032
2024-10-22 15:38:14,501 WARN mapreduce.JobResourceUploader: Hadoop command-line
 option parsing not performed. Implement the Tool interface and execute your ap
plication with ToolRunner to remedy this.
2024-10-22 15:38:14,539 INFO mapreduce.JobResourceUploader: Disabling Erasure C
oding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1729604884725_0002
2024-10-22 15:38:15,021 INFO input.FileInputFormat: Total input files to proces
s : 1
2024-10-22 15:38:15,165 INFO mapreduce.JobSubmitter: number of splits:1
2024-10-22 15:38:15,364 INFO mapreduce.JobSubmitter: Submitting tokens for job:
 job_1729604884725_0002
2024-10-22 15:38:15,364 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-10-22 15:38:15,679 INFO conf.Configuration: resource-types.xml not found
2024-10-22 15:38:15,680 INFO resource.ResourceUtils: Unable to find 'resource-t
ypes.xml'.
2024-10-22 15:38:15,837 INFO impl.YarnClientImpl: Submitted application applica
tion_1729604884725_0002
2024-10-22 15:38:15,907 INFO mapreduce.Job: The url to track the job: http://h0
```



打印输出文件结果

```
root@h01:/usr/local/hadoop# ./bin/hadoop fs -ls /output/stock_count
Found 2 items
-rw-r--r--   2 root supergroup          0 2024-10-22 15:38 /output/stock_count/
_SUCCESS
-rw-r--r--   2 root supergroup      86831 2024-10-22 15:38 /output/stock_count/
part-r-00000
root@h01:/usr/local/hadoop# ./bin/hadoop fs -cat /output/stock_count/part-r-000
00
        1: MS, 726
        2: MRK, 704
        3: QQQ, 693
        4: BABA, 689
        5: EWU, 681
        6: GILD, 663
        7: JNJ, 663
        8: MU, 659
        9: NVDA, 655
        10: VZ, 648
        11: KO, 643
        12: QCOM, 636
        13: M, 635
        14: NFLX, 635
        15: EBAY, 621
        16: DAL, 605
```

- WEB页面截图



1.3 程序分析

进一步对性能、扩展性等方面存在的不足和可能改进之处进行分析。

1.3.1 Mapper输出量

不足：对每一行（股票代码不为空的）数据都输出`<股票代码，1>`，导致Mapper输出量太大。

改进分析：先在Mapper对股票代码的计数进行局部聚合，减少Mapper和Reducer之间的数据传输量。

1.3.2 Reducer运行内存

不足：使用List来存储所有股票代码和它们的出现次数，并对整个List进行排序，这对于大型数据集来说会占用大量内存，可能导致崩溃。

改进分析：可以将数据分成多个部分进行排序，然后通过归并排序的方法将结果合并，降低内存占用。

2 WordFrequency

要求：在HDFS上加载上市公司热点新闻标题数据集（analyst_ratings.csv），该数据集收集了部分上市公司的热点财经新闻标题。统计数据集热点新闻标题（“headline”列）中出现的前100个高频单词，按出现次数从大到小输出。要求忽略大小写，忽略标点符号，忽略停词（stop-word-list.txt）。输出格式为“<排名 >：<单词>，<次数>”。

2.1 设计思路

设计的思路与前一个任务基本一致，但多了几个处理：停词表的加载、对切片后字段的处理，

- Mapper

  将停词表加载到HashSet中，可以在后续操作快速查询是否为停词。

  逐行处理数据，使用split方法切片，提取第二个字段headline；去除标点符号，将文本转换为小写，并按空格分割成单词；检查每个单词是否为停词或空词，如果不是，则将其输出为键值对`<单词，1>`。

- Reducer

  接收来自Mapper的输出，累加相同单词的出现次数，得到每个单词的总出现次数；并维护一个map存储每个单词及其出现次数。

  cleanup方法在Reducer任务结束后将map转换为list，按照单词出现次数进行降序排序，输出出现次数最多的前100个单词，格式为“<排名>: <单词>, <次数>”。

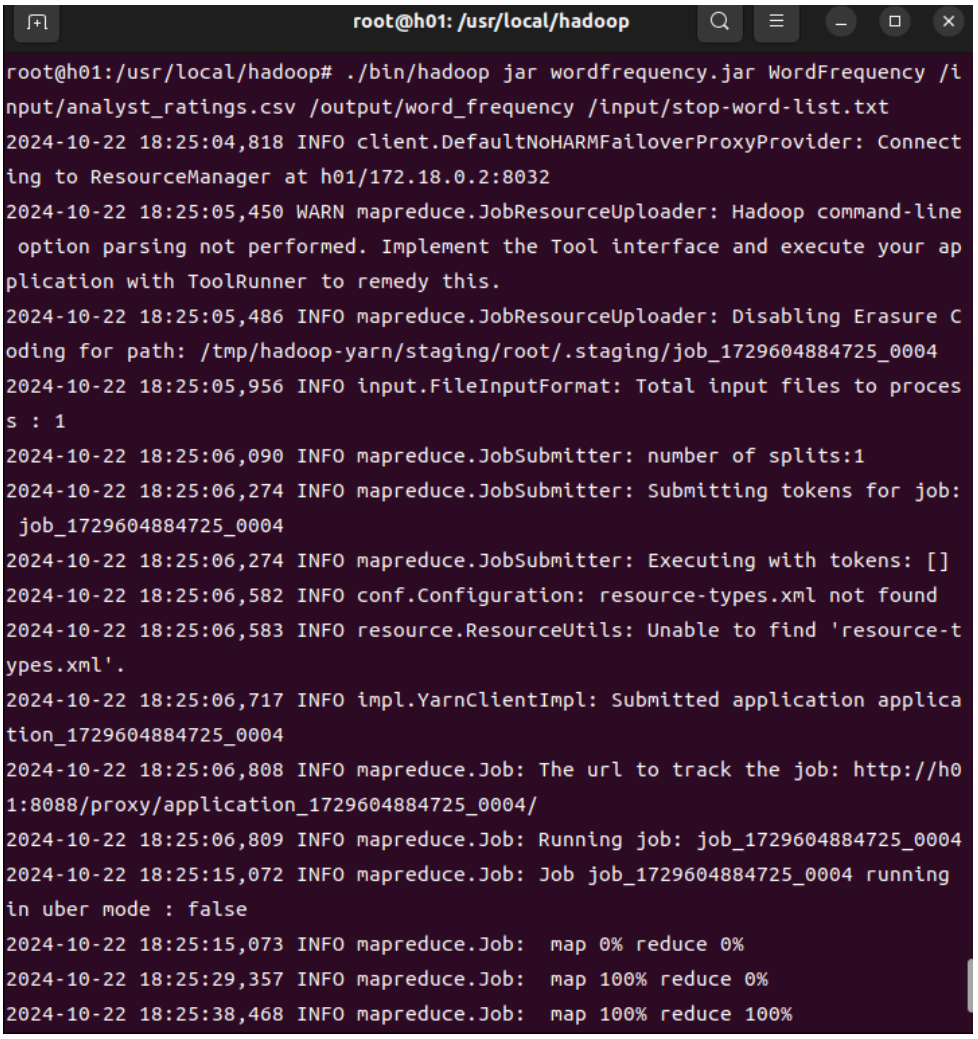
- main

  设置MapReduce作业的配置：配置Mapper和Reducer类；指定输入、输出文件路径（args[0]和 args[1]）；启动作业并等待作业完成。

2.2 程序运行结果

- 终端输出结果

  运行MapReduce结果



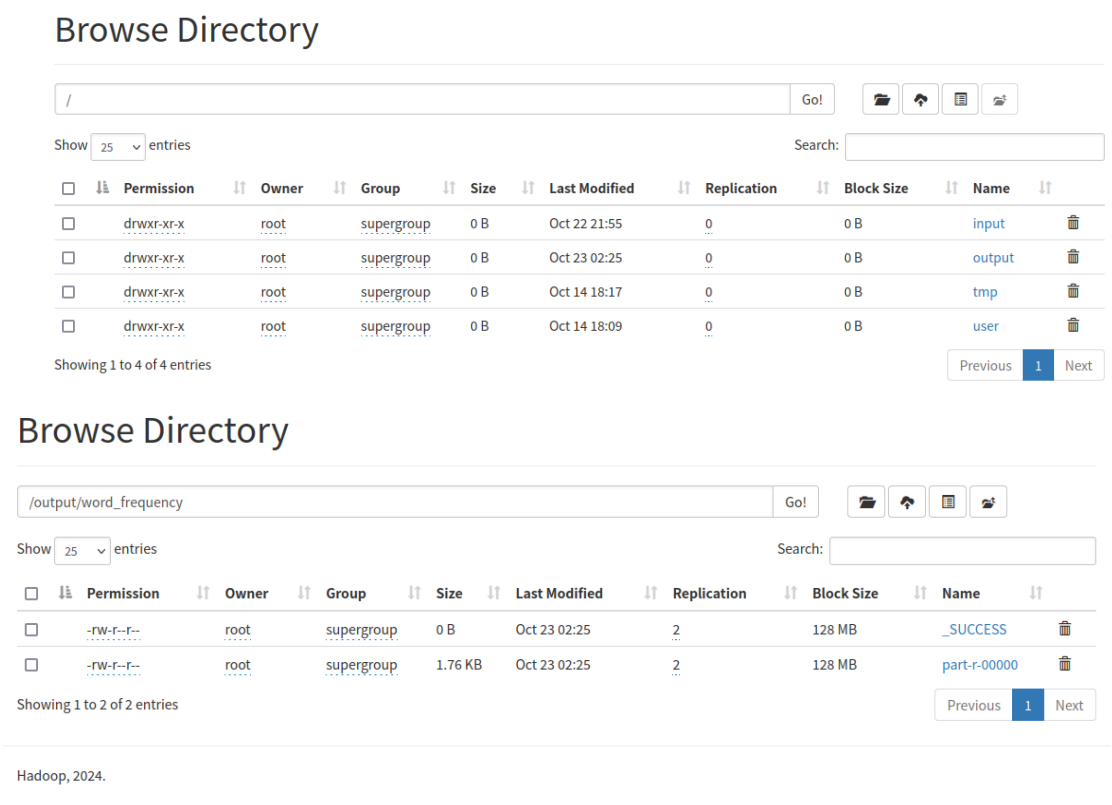

```
        Reduce output records=100
        Spilled Records=4267858
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=568
        CPU time spent (ms)=14130
        Physical memory (bytes) snapshot=834555904
        Virtual memory (bytes) snapshot=5201440768
        Total committed heap usage (bytes)=783286272
        Peak Map Physical memory (bytes)=590536704
        Peak Map Virtual memory (bytes)=2594750464
        Peak Reduce Physical memory (bytes)=244019200
        Peak Reduce Virtual memory (bytes)=2606690304
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=52462980
    File Output Format Counters
        Bytes Written=1799
```

打印输出文件结果

```
root@h01:/usr/local/hadoop# ./bin/hadoop fs -ls /output/word_frequency/
Found 2 items
-rw-r--r--   2 root supergroup          0 2024-10-22 18:25 /output/word_frequen
cy/_SUCCESS
-rw-r--r--   2 root supergroup       1799 2024-10-22 18:25 /output/word_frequen
cy/part-r-00000
root@h01:/usr/local/hadoop# ./bin/hadoop fs -cat /output/word_frequency/part-r-
00000
        1: stocks, 37669
        2: shares, 26843
        3: q, 25950
        4: vs, 24905
        5: m, 24538
        6: update, 23804
        7: market, 22458
        8: est, 20181
        9: reports, 19300
        10: eps, 18050
        11: session, 14337
        12: week, 14202
```

```
88: gathers, 3354
89: services, 3331
90: volume, 3264
91: ahead, 3201
92: midafternoon, 3190
93: etf, 3182
94: report, 3175
95: futures, 3173
96: wednesdays, 3163
97: thursdays, 3096
98: coverage, 3077
99: amid, 3049
100: ceo, 3007
```

- WEB页面截图



2.3 程序分析

进一步对性能、扩展性等方面存在的不足和可能改进之处进行分析。

2.3.1 Reducer内存

不足：WordReducer将所有的单词计数存储在一个HashMap中。如果输入数据量非常大，单词的种类和数量超过Reducer的内存限制，会导致内存溢出问题。

改进分析：先在Mapper对单词的计数进行局部聚合，减少Mapper和Reducer之间的数据传输量，以此减少Reducer阶段需要处理的数据量，从而减轻内存压力。

2.3.2 排序优化

不足：在Reducer的cleanup方法中，对所有单词进行排序，性能开销较大，尤其当单词总量较大时，排序时间会显著增长。

改进分析：可以先找到出现次数第100大的元素，然后剔除比该元素出现次数少的单词，对最高频100个元素进行排序，减少了不必要的全局排序开销。