**大数据分析技术与应用实验报告**

**实验一:Hadoop环境搭建测试**

目 录

[1 实验任务 1](#_Toc107163030)

[2 实验过程 1](#_Toc107163031)

[2.1 Hadoop 实现文件操作 1](#_Toc107163032)

[1. 向HDFS中上传任意文本文件，如果指定的文件在HDFS 中已经存在，由用户指定是追加到原有文件末尾还是覆盖原有的文件 1](#_Toc107163033)

[2. 从HDFS中下载指定文件，如果本地文件与要下载的文件名称相同，则自动对下载的文件重命名 5](#_Toc107163034)

[3. 将HDFS中指定文件的内容输出到终端中 7](#_Toc107163035)

[4. 显示HDFS中指定的文件的读写权限﹑大小﹑创建时间﹑路径等信息 8](#_Toc107163036)

[5. 删除HDFS中指定的文件 9](#_Toc107163037)

[6. 删除HDFS中指定的目录 10](#_Toc107163038)

[7. 在HDFS 中，将文件从源路径移动到目的路径 13](#_Toc107163039)

[2.2 Wordcount实例 14](#_Toc107163040)

[3 实验总结 19](#_Toc107163041)

# 1 实验任务

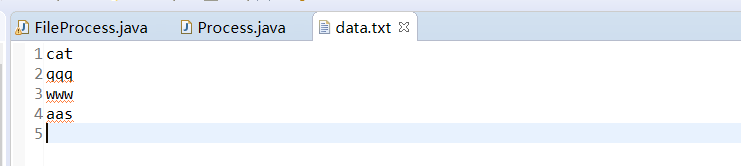
1. 熟悉常用的Hadoop命令
2. 运行WordCount实例
3. 搭建Eclipse编程环境
4. 编程实现Wordcount程序

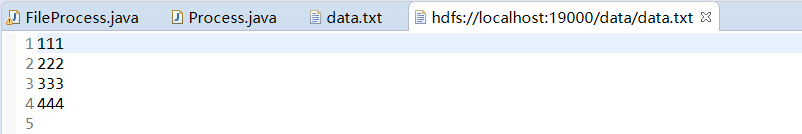
# 2 实验过程

## 2.1 Hadoop 实现文件操作

### 向HDFS中上传任意文本文件，如果指定的文件在HDFS 中已经存在，由用户指定是追加到原有文件末尾还是覆盖原有的文件

#### Eclipse编程实现

本地的data.txt文件

HDFS的data.txt文件

文件追加函数代码如下：

public static void addFile(Configuration conf, String choice) {

//本地路径

String localFilePath = "D:\\code\\eclipsej2ee\\DataProcess\\src\\fileprocess\\data.txt";

//HDFS路径

String remoteFilePath = "/data/data.txt";

try (FileSystem fs = FileSystem.get(conf)){

// 判断HDFS文件路径是否存在

if(fs.exists(new Path(remoteFilePath))) {

//用户指定文件处理方式

if(choice.equals("append")) { //选择追加到末尾

FileInputStream in = new FileInputStream(localFilePath);

FSDataOutputStream out = fs.append(new Path(remoteFilePath));

byte[] data = new byte[1024];

int read = -1;

while((read = in.read(data))>0) {

out.write(data,0,read);

}

out.close();

System.out.println("追加成功");

}else if(choice.equals("overwrite")){ //选择覆盖

//第一个参数表示是否删除源文件，第二个参数表示是否覆盖

fs.copyFromLocalFile(false, true, new Path(localFilePath), new Path(remoteFilePath));

System.out.println("覆盖成功");

}

}else {

//上传文件

fs.copyFromLocalFile(false, true, new Path(localFilePath), new Path(remoteFilePath));

}

fs.close();

}catch(IOException e) {

e.printStackTrace();

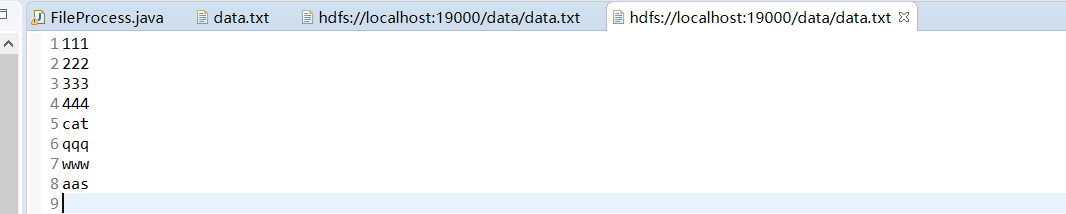
}

}

1. 在HDFS中追加data.txt文件内容

Configuration conf = new Configuration();

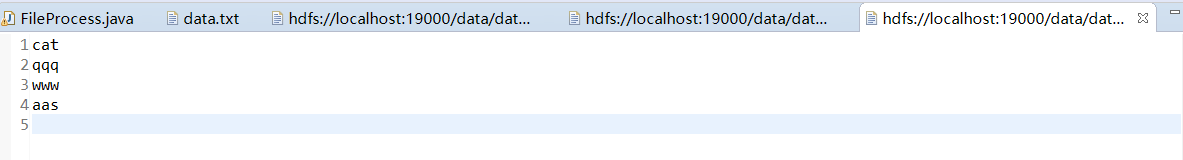
addFile(conf,"append");

运行后HDFS data.txt内容如下图所示

1. 在HDFS中覆盖data.txt文件内容

Configuration conf = new Configuration();

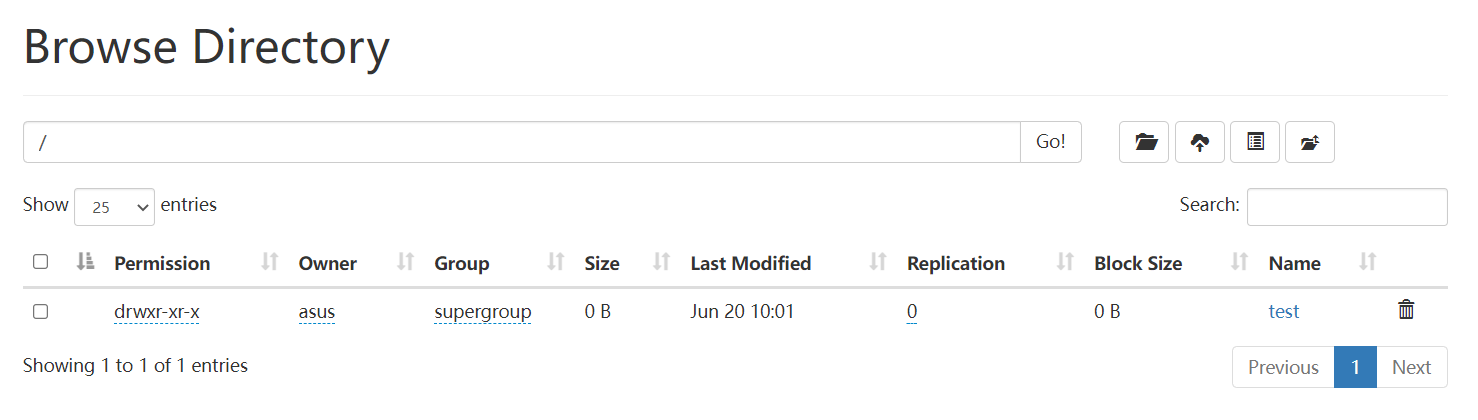
addFile(conf, "overwrite");

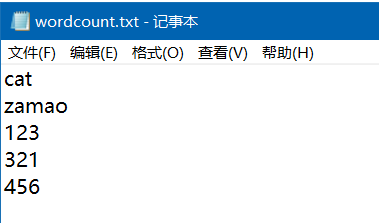
运行后HDFS data.txt内容如图所示:

#### Shell命令实现

1. 在hdfs中创建文件夹test

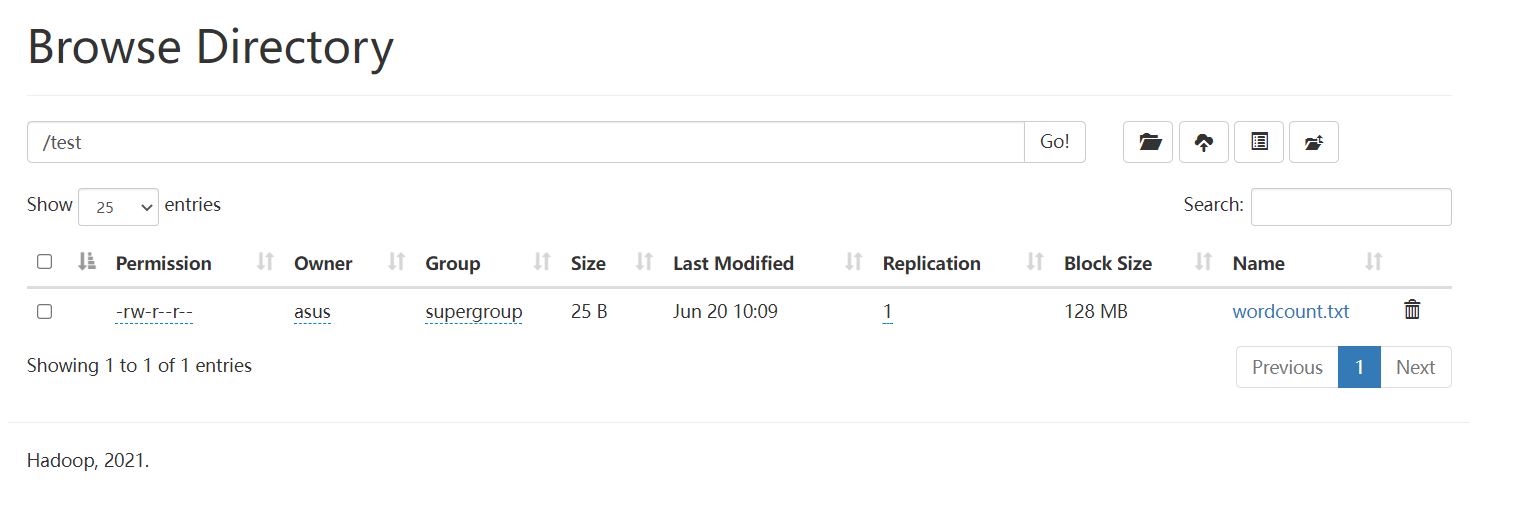
运行 hdfs dfs -mkdir /test

1. 向hdfs上传txt文件

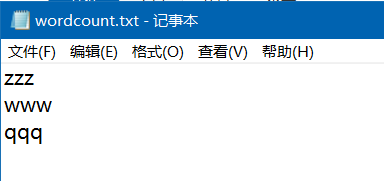
要上传的txt文件内容

运行 hdfs dfs -put ./wordcount.txt /test

-put:上传文件 第一个路径是要传的文件路径，第二个是目的地

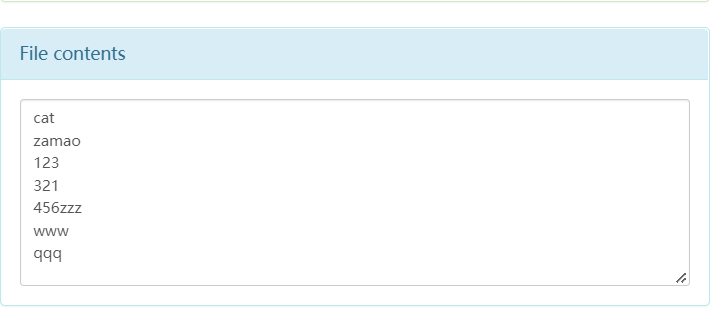
上传到hdfs中的wordcount

1. 上传相同文件名的wordcount.txt，并在原文件中追加

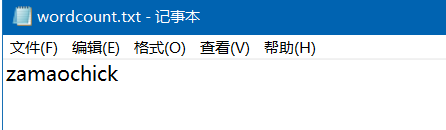
 要上传的txt文件内容

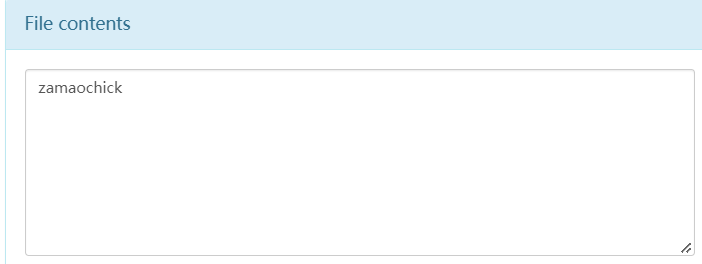
运行 hdfs dfs -appendToFile ./wordcount.txt /test/wordcount.txt

hdfs wordcount.txt内容

1. 上传相同文件名的wordcount.txt，并覆盖原文件

要上传的wordcount.txt文件

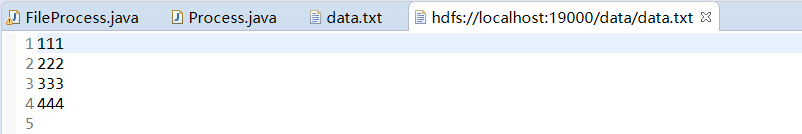
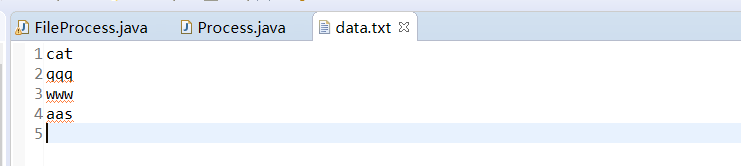
运行 hdfs dfs -copyFromLocal -f ./wordcount.txt /test/wordcount.txt

hdfs wordcount.txt文件内容

### 从HDFS中下载指定文件，如果本地文件与要下载的文件名称相同，则自动对下载的文件重命名

#### Eclipse编程实现

本地的data.txt文件

HDFS的data.txt文件

文件下载函数代码如下：

public static void downloadFile(Configuration conf) {

//本地路径

String localFilePath = "D:\\code\\eclipsej2ee\\DataProcess\\src\\fileprocess\\data.txt";

//HDFS路径

String remoteFilePath = "/data/data.txt";

try (FileSystem fs = FileSystem.get(conf)){

//判断本地文件是否存在

File f = new File(localFilePath);

if(f.exists()) {

//文件存在：自动重命名

System.out.println(localFilePath+"已存在");

Integer i = Integer.valueOf(0);

while(true) {

localFilePath = localFilePath+"\_"+i.toString();

f = new File(localFilePath);

if(!f.exists()) {

break;

}else {

i++;

continue;

}

}

}

System.out.println("将重新命名为："+localFilePath);

//下载文件到本地

fs.copyToLocalFile(new Path(remoteFilePath), new Path(localFilePath));

System.out.println("下载完成");

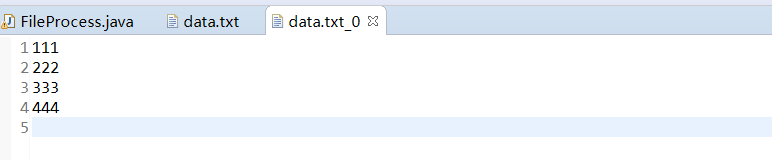
}catch(IOException e) {

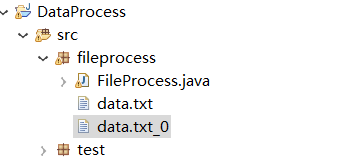
e.printStackTrace();

}

}

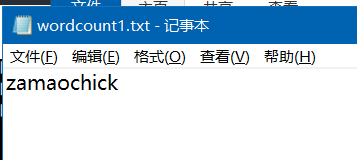
因为HDFS中的data.txt与本地的data.txt重名，所以HDFS的data.txt下载到本地时，重命名为data.txt\_0

运行后下载的data.txt\_0内容如图所示:

下载的data.txt\_0所处的目录如下:

#### Shell命令实现

运行 hdfs dfs -copyToLocal /test/wordcount.txt ./wordcount1.txt

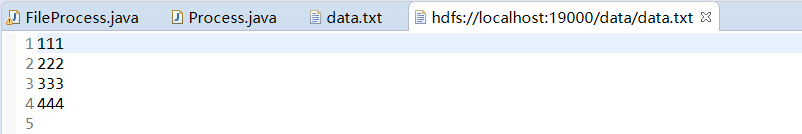
第一个路径是hdfs文件路径，第二个路径是本地存放文件位置

本地从hdfs下载文件内容

### 将HDFS中指定文件的内容输出到终端中

#### Eclipse编程实现

实现对HDFS的data.txt文件内容输出

HDFS的data.txt文件内容如下:

文件内容查看函数代码如下:

public static void watchFile(Configuration conf) {

//HDFS路径

String remoteFilePath = "/data/data.txt";

try(FileSystem fs = FileSystem.get(conf);

FSDataInputStream in = fs.open(new Path(remoteFilePath)); //打开HDFS上文件

BufferedReader d = new BufferedReader(new InputStreamReader(in));)

{

String line;

while((line=d.readLine())!=null) { //读取完整的一行字符串

System.out.println(line); //输出

}

}catch(IOException e) {

e.printStackTrace();

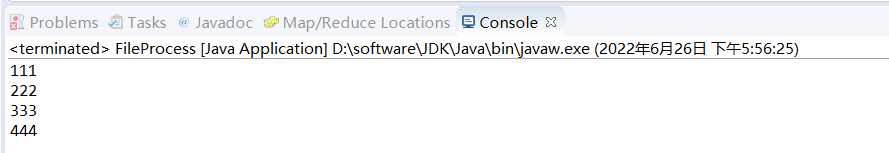
}

}

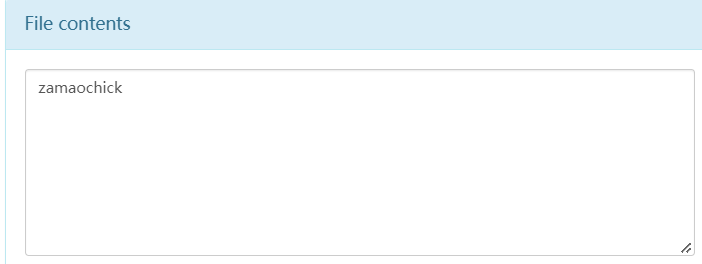
运行文件内容显示程序：

Configuration conf = new Configuration();

watchFile(conf);

运行结果如下:

#### Shell命令实现

hdfs wordcount.txt文件内容

运行 hdfs dfs -cat /test/wordcount.txt

wordcount.txt 文件输出结果



### 显示HDFS中指定的文件的读写权限﹑大小﹑创建时间﹑路径等信息

#### Eclipse编程实现

查看HDFS中的data/data.txt文件信息

查看文件信息函数代码如下:

public static void watchFileInfo(Configuration conf) {

String remoteFilePath = "/data/data.txt";

try(FileSystem fs = FileSystem.get(conf)){

//FileStatus对象通过listStatus()获取文件状态信息

FileStatus[] fileStatuses = fs.listStatus(new Path(remoteFilePath));

for(FileStatus s: fileStatuses) {

System.out.println("路径："+s.getPath().toString());

System.out.println("读写权限："+s.getPermission().toString());

System.out.println("文件大小："+s.getLen());

//返回的是时间戳，转化为时间日期格式

long timeStamp = s.getModificationTime();

SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String date = format.format(timeStamp);

System.out.println("创建时间："+date);

}

}catch(IOException e) {

e.printStackTrace();

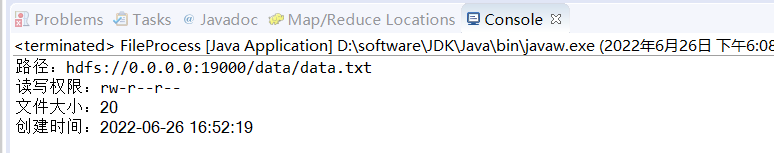
}

}

运行查看文件信息函数：

Configuration conf = new Configuration();

watchFileInfo(conf);

运行结果如下:

#### Shell命令实现

显示wordcount.txt的读写权限、大小、创建时间、路径等信息

运行 hdfs dfs -ls -h /test/wordcount.txt

### 删除HDFS中指定的文件

#### Eclipse编程实现

删除HDFS中data文件夹下的data.txt文件

文件删除函数代码如下:

public static void deleteFile(Configuration conf) {

String remoteFile = "/data/data.txt";

try(FileSystem fs = FileSystem.get(conf)){

if(fs.exists(new Path(remoteFile))) {

boolean b = fs.delete(new Path(remoteFile), false); //false表示不递归删除

System.out.println("删除文件："+remoteFile+" "+b);

}

}catch(IOException e) {

e.printStackTrace();

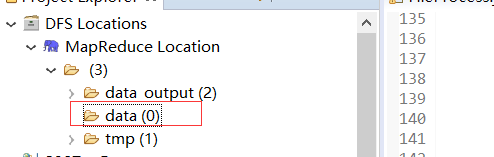
}

}

运行文件删除函数：

Configuration conf = new Configuration();

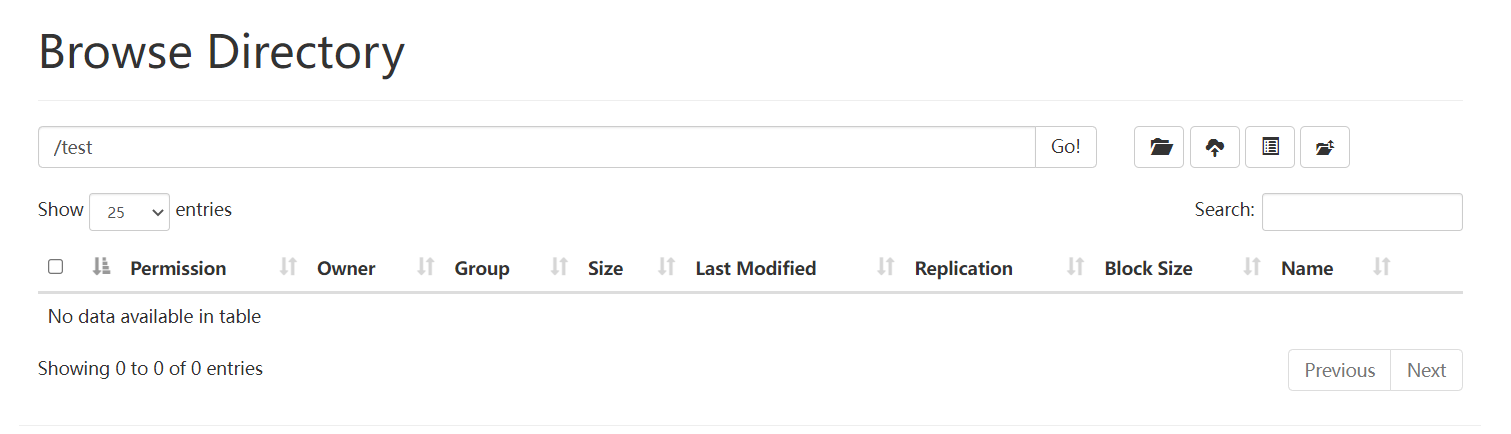
deleteFile(conf);

删除后data文件夹内容如下:

#### Shell命令实现

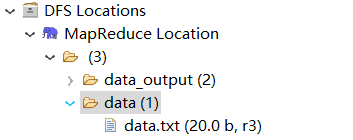
删除hdfs中test目录下的wordcount.txt文件

运行 hdfs dfs -rm /test/wordcount.txt

hdfs test下的wordcount.txt已经被删除

### 删除HDFS中指定的目录，由用户指定目录中如果存在文件时是否删除目录

#### Eclipse编程实现

HDFS data文件夹：

删除目录函数如下:

public static void deleteNo(Configuration conf,boolean deleteNotEmpty) {

String remotePath = "/data";

try(FileSystem fs = FileSystem.get(conf)){

//判断路径是否存在。

if(fs.exists(new Path(remotePath))) {

FileStatus[] listStatus = fs.listStatus(new Path(remotePath));

if(listStatus.length!=0) { //目录不为空

if(deleteNotEmpty) {

fs.delete(new Path(remotePath),true);

System.out.println("目录不为空，仍然删除"+remotePath.toString());

}else {

System.out.println("目录不为空，不删除"+remotePath.toString());

}

}else{ //目录为空，直接删除

fs.delete(new Path(remotePath));

System.out.println("删除"+remotePath);

}

}else {

//目录不存在

System.out.println("目录不存在"+remotePath);

}

}catch(IOException e) {

e.printStackTrace();

}

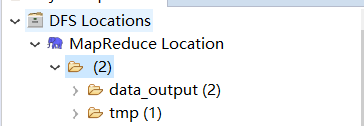
}

删除HDFS中的data文件夹，如果存在文件，仍然删除文件夹

运行删除目录函数:

Configuration conf = new Configuration();

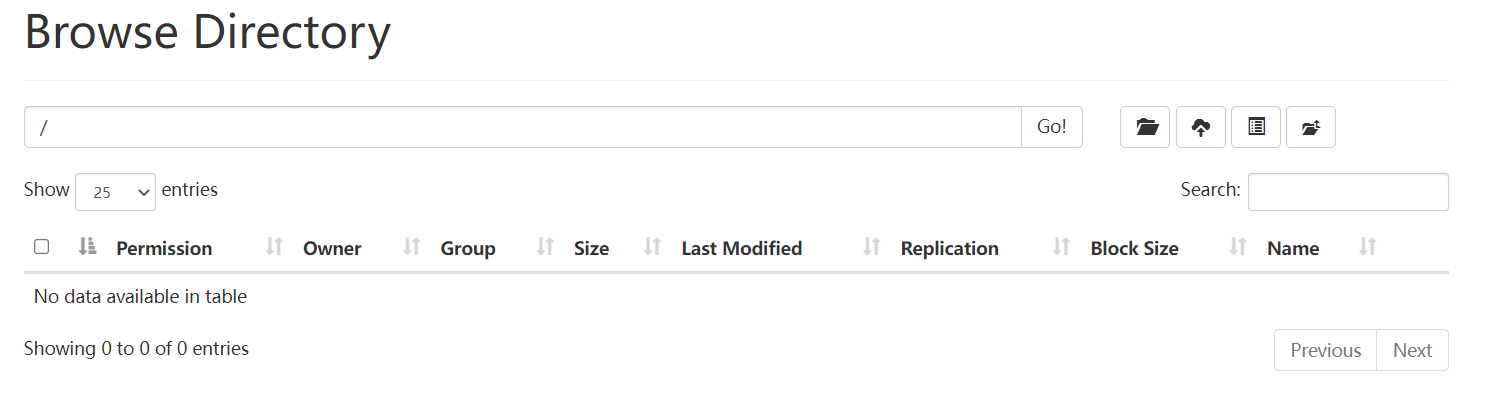
deleteNo(conf,true);

运行后HDFS变化情况:

#### Shell命令实现

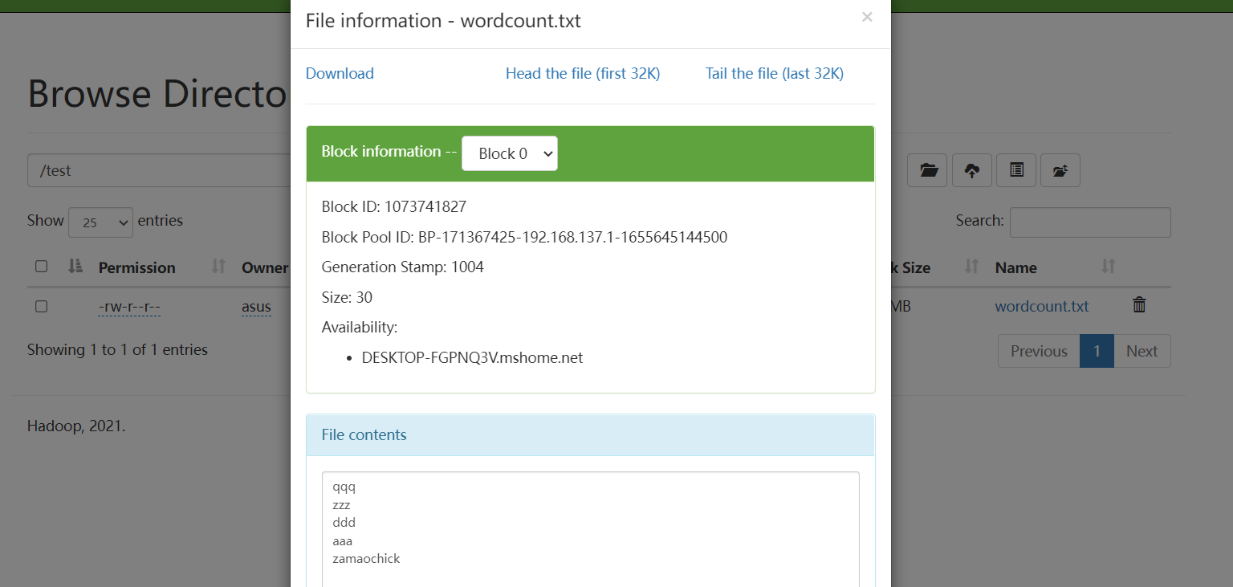
1. 删除空目录test

运行 hdfs dfs -rmdir /test

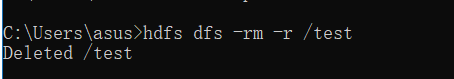
hdfs 中的test文件夹已被删除

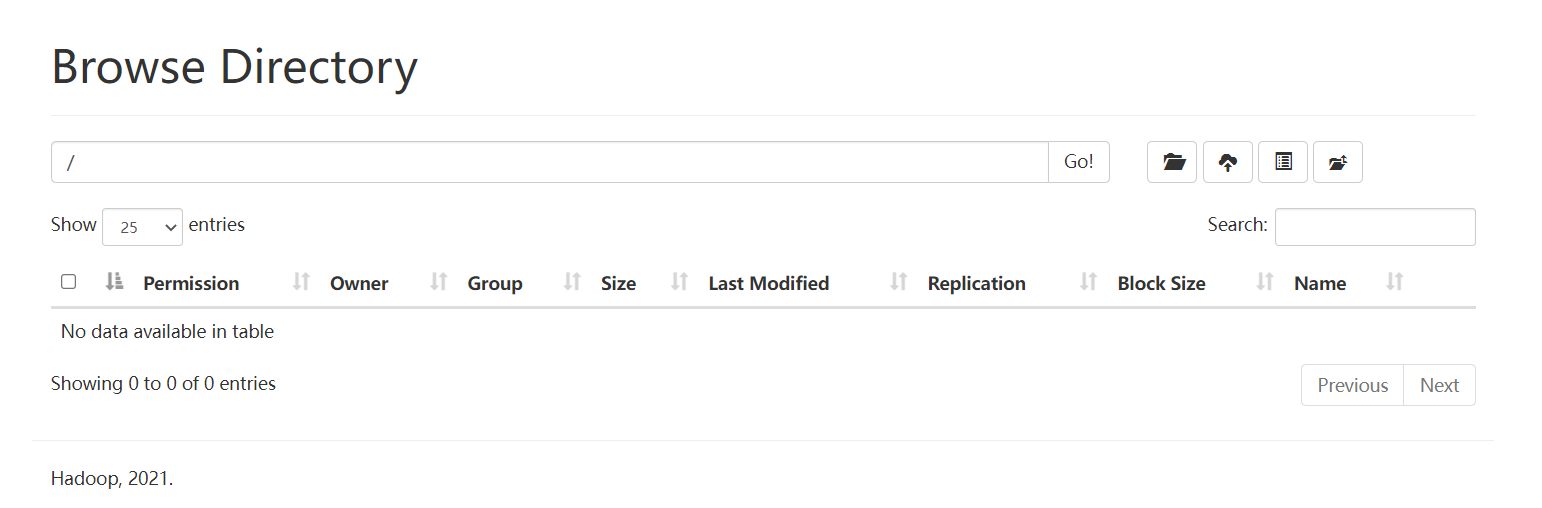
1. 存在文件时强制删除文件夹

创建/test/wordcount.txt文件

运行 hdfs dfs -rm -r /test

hdfs 中的 test文件夹以及文件夹的内容已被删除

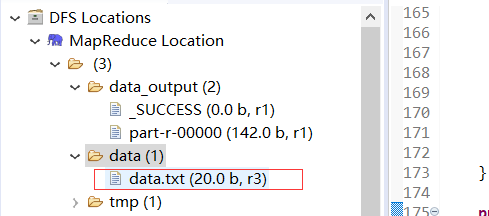




### 在HDFS 中，将文件从源路径移动到目的路径

#### Eclipse编程实现

将HDFS中的data文件夹下的data.txt移动到data\_output文件夹下

HDFS文件情况：

移动文件函数代码如下:

public static void removeFile(Configuration conf) {

String remoteSrcPath = "/data/data.txt";

String remoteDstPath = "/data\_output/data.txt";

try(FileSystem fs = FileSystem.get(conf)){

fs.rename(new Path(remoteSrcPath), new Path(remoteDstPath));

System.out.println("将文件"+remoteSrcPath+"移动到"+remoteDstPath);

}catch(IOException e) {

e.printStackTrace();

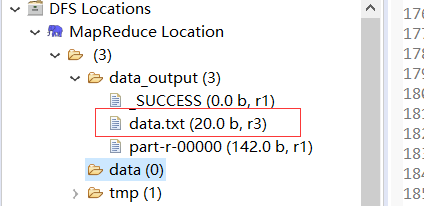
}

}

运行移动文件函数:

Configuration conf = new Configuration();

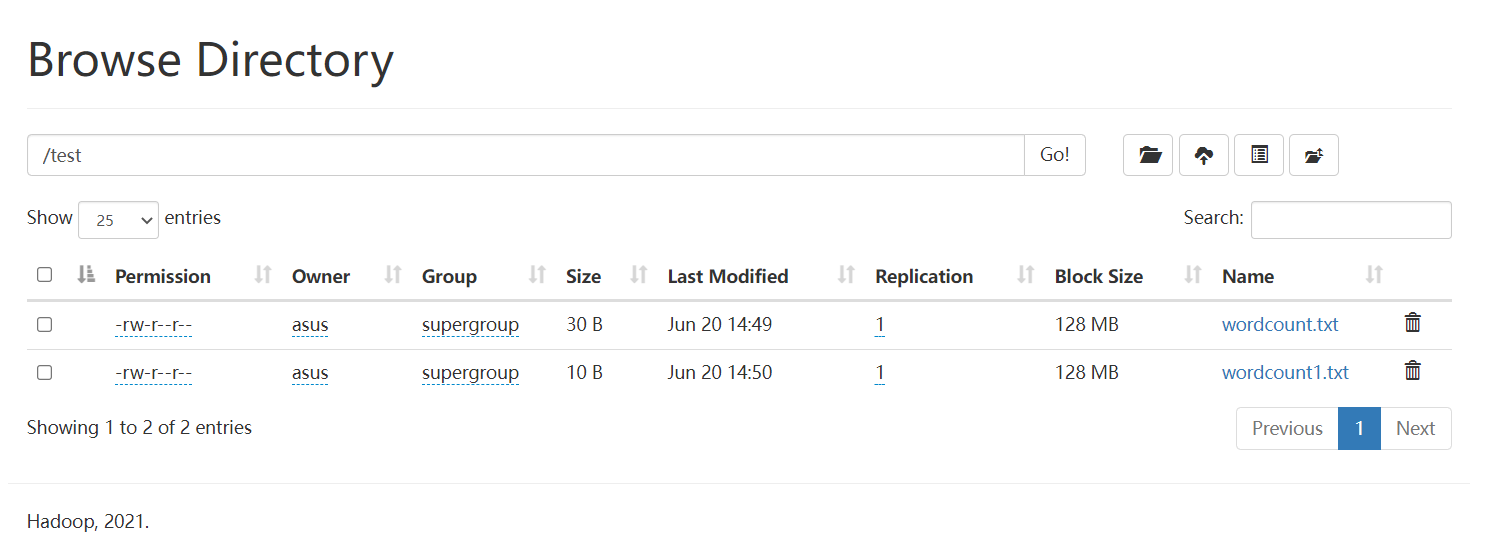
removeFile(conf);

运行结果如下:

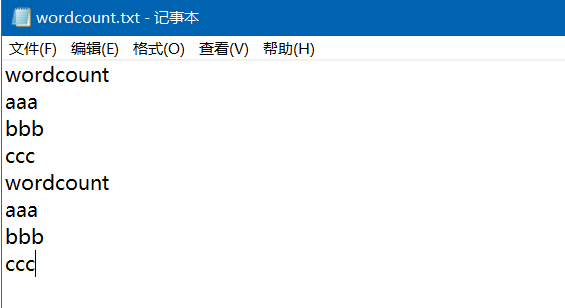
#### Shell命令实现

将/wordcount1.txt移动到/test下

运行 hdfs dfs -mv /wordcount1.txt /test/wordcount1.txt

wordcount1.txt已经移动到/test下

## 2.2 Wordcount实例

1. 使用系统命令实现wordcount程序
2. 在本地创建一个wordcount.txt文本文档
3. 启动hadoop

运行 %HADOOP\_HOME%\sbin\start-all.cmd

1. 将wordcount.txt存在hdfs中的/tmp文件夹下

hdfs创建/tmp文件夹

运行 hadoop fs -mkdir /tmp

将wordcount.txt上传到hdfs/tmp文件夹下

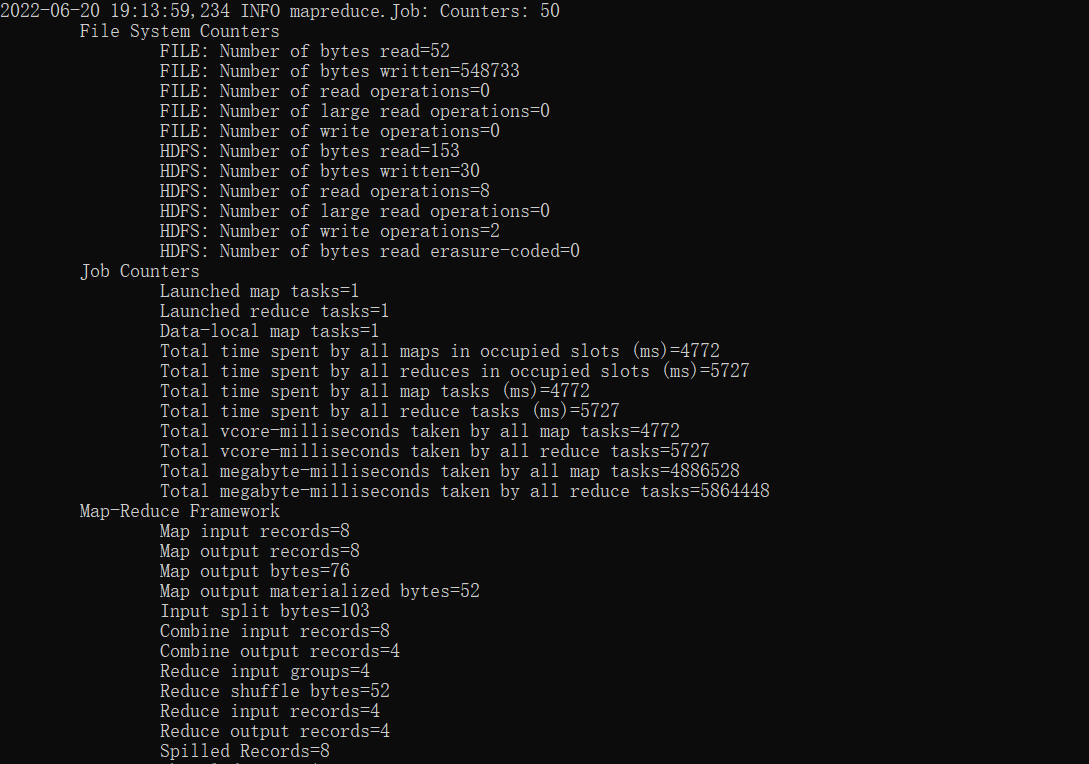
运行 hadoop fs -put ./wordcount.txt /tmp

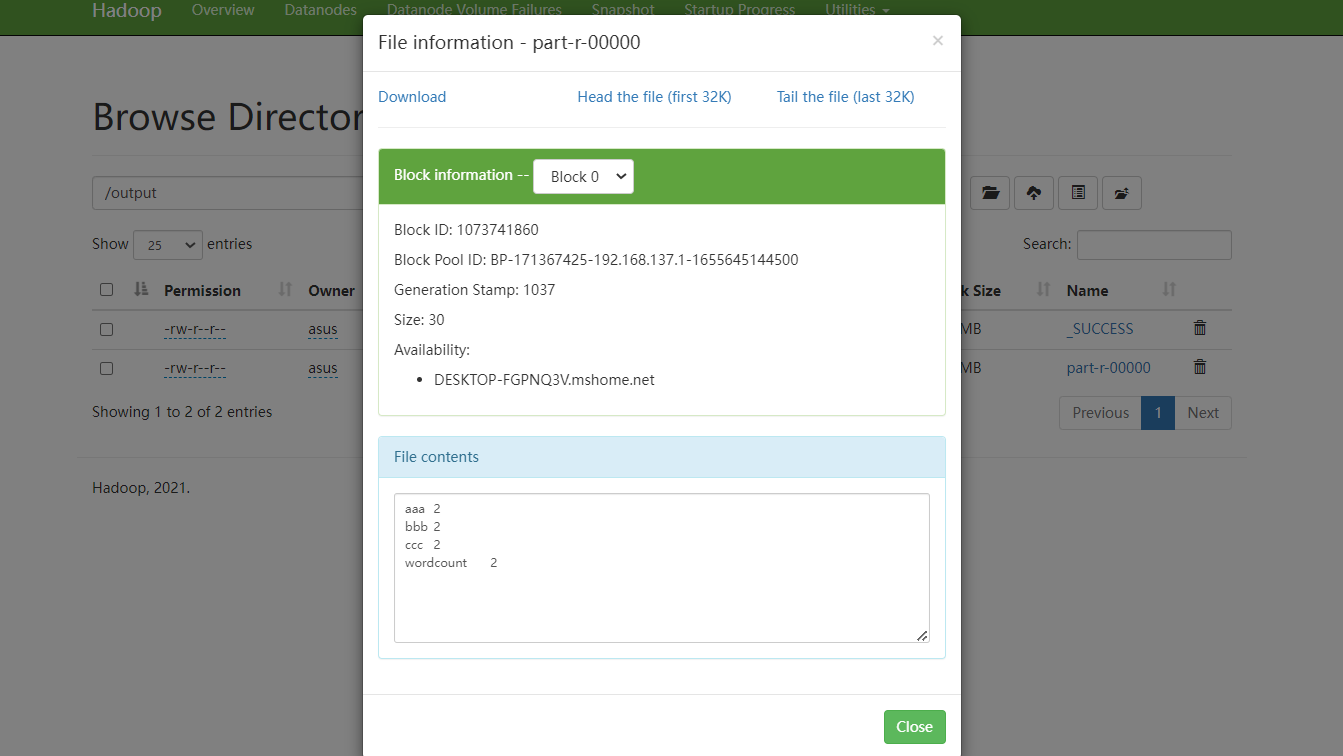
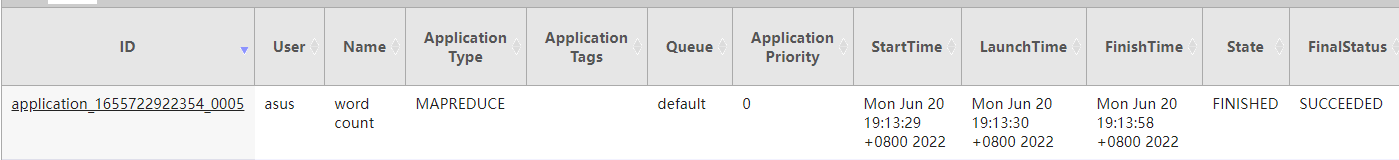
进入到mapreduce文件夹下，并运行hadoop-mapreduce-examples jar包

统计的是/tmp/wordcount.txt的单词数目，并将结果输出到output文件夹下

cd F:\hadoop\hadoop-3.3.1\share\hadoop\mapreduce

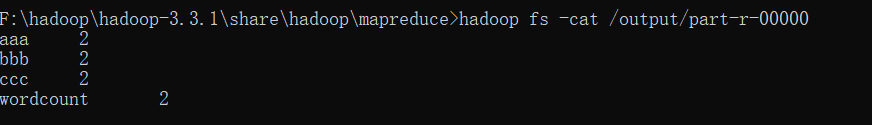
hadoop jar hadoop-mapreduce-examples-3.3.1.jar wordcount /tmp/wordcount.txt /output

运行截图如下：

hdfs output文件夹下统计结果

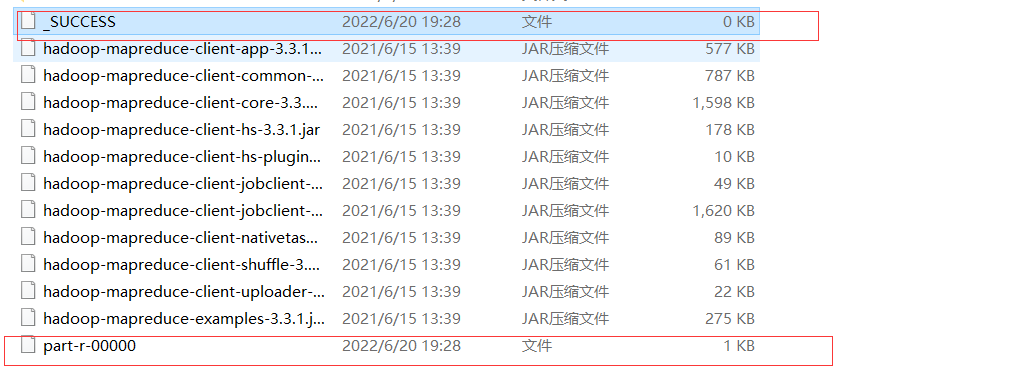
1. 在终端输出结果

运行 hadoop fs -cat /output/part-r-0000

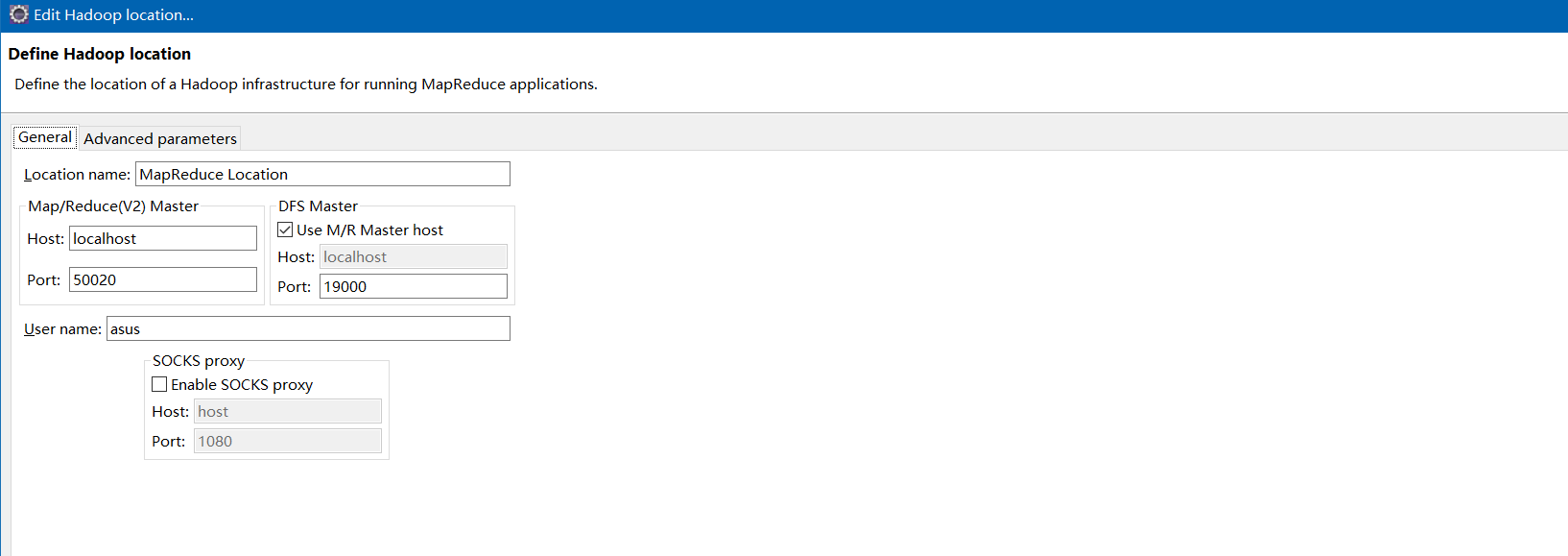


1. 把结果下载到本地文件系统

运行 hadoop fs -get /output/\* ./

第一个路径为hdfs的文件路径，第二个路径为本地相对路径

1. 搭建eclipse环境编程实现wordcount程序

在eclipse添加Map/Reduce,设置的端口与core-site配置的端口对应。

创建WordCount类，并将core-site.xml,hdfs-site.xml等文件复制到Map/Reduce Project中，WordCount类代码如下:

public class WordCount {

public WordCount() {

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingArgs();

if(otherArgs.length < 2) {

System.err.println("Usage: wordcount <in> [<in>...] <out>");

System.exit(2);

}

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(WordCount.TokenizerMapper.class);

job.setCombinerClass(WordCount.IntSumReducer.class);

job.setReducerClass(WordCount.IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

for(int i = 0; i < otherArgs.length - 1; ++i) {

FileInputFormat.addInputPath(job, new Path(otherArgs[i]));

}

FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length - 1]));

System.exit(job.waitForCompletion(true)?0:1);

}

public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

public IntSumReducer() {

}

public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {

int sum = 0;

IntWritable val;

for(Iterator i$ = values.iterator(); i$.hasNext(); sum += val.get()) {

val = (IntWritable)i$.next();

}

this.result.set(sum);

context.write(key, this.result);

}

}

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

private static final IntWritable one = new IntWritable(1);

private Text word = new Text();

public TokenizerMapper() {

}

public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {

StringTokenizer itr = new StringTokenizer(value.toString());

while(itr.hasMoreTokens()) {

this.word.set(itr.nextToken());

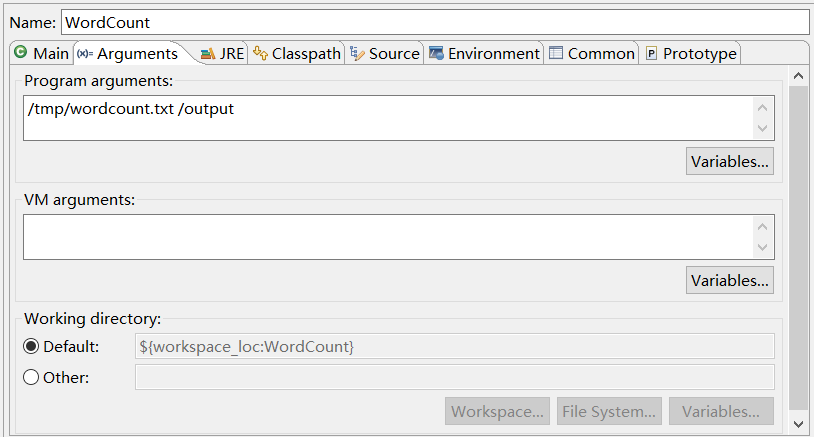
context.write(this.word, one);

}

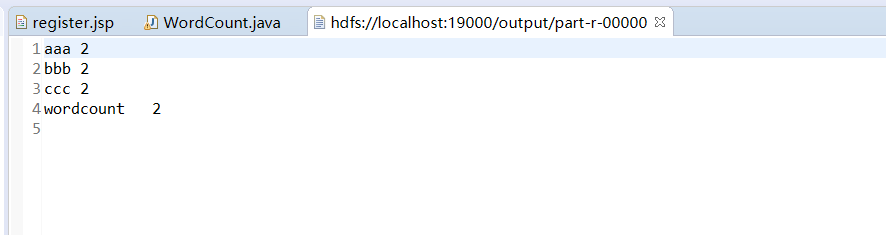
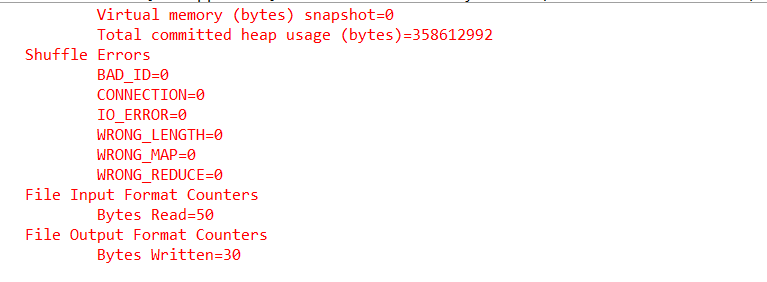
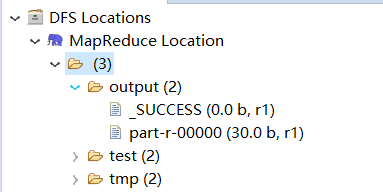
}

}

}

运行WordCount，设置的运行参数如下，第一个参数是要统计的单词信息的文件路径，第二个是输出结果地址。

运行结果如下：

生成的output文件夹在DFS中如图所示：

生成的统计结果文件如图所示：

# 3 实验总结

通过此次实验，我学到了很多新知识，收获颇丰。我学会了安装hadoop和java，学会了如何运行和结束hadoop。我学会了使用shell命令在hadoop上添加、删除、查看、下载文件。我学会了使用hadoop运行wordcount程序。我学会了在eclipse中配置hadoop环境，并创建MapReduce项目实现运行wordcount程序。老师提供的实验指导书内容详尽，给我做实验途中提供了许多帮助。