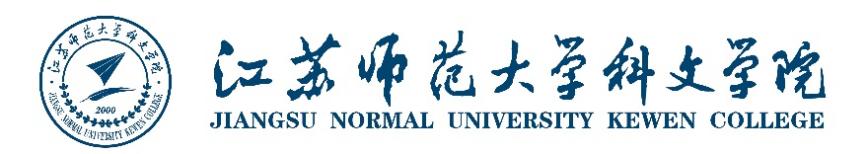
****

**人工智能与软件学院**

实 验 报 告

**课 程 大数据分析与智能决策**

**学 号 7200767121**

**姓 名 彭敏捷**

**班 级 20智科**

**教 师 胡元发**

2023年 4 月至 2023 年 4 月

**一、实验名称**

MapReduce编程初级实践

**二、实验目的和要求**

1、编程实现文件合并和去重操作：对于两个输入文件，即文件A和文件B，编写MapReduce程序，对两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件C。

2、编写程序实现对输入文件的排序：现在有多个输入文件，每个文件中的每行内容均为一个整数。要求读取所有文件中的整数，进行升序排序后，输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数。

**三、实验环境**

1.VMware Workstation Pro

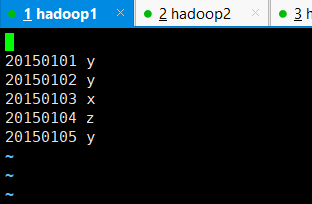
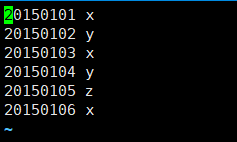
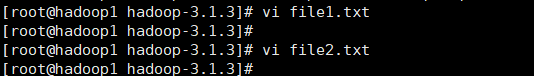
2.Xshell

1. **实验步骤**
2. 编程实现文件合并和去重操作

(1)创建/目录下的input文件: hadoop fs -mkdir -p input。

创建file1.txt和file2.txt。vi file1.txt ; vi file2.txt 。

2

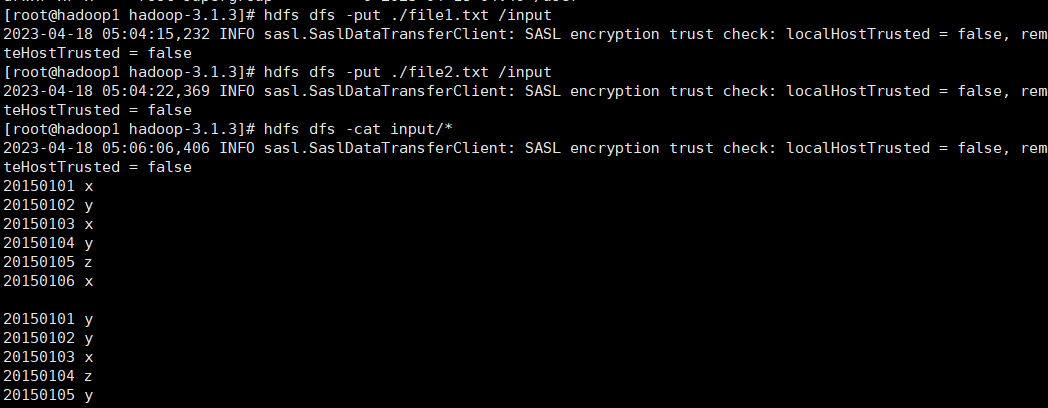


1. 将file1.txt和file2.txt传入到input文件中，并查看input文件。

hdfs dfs -put ./file1.txt input

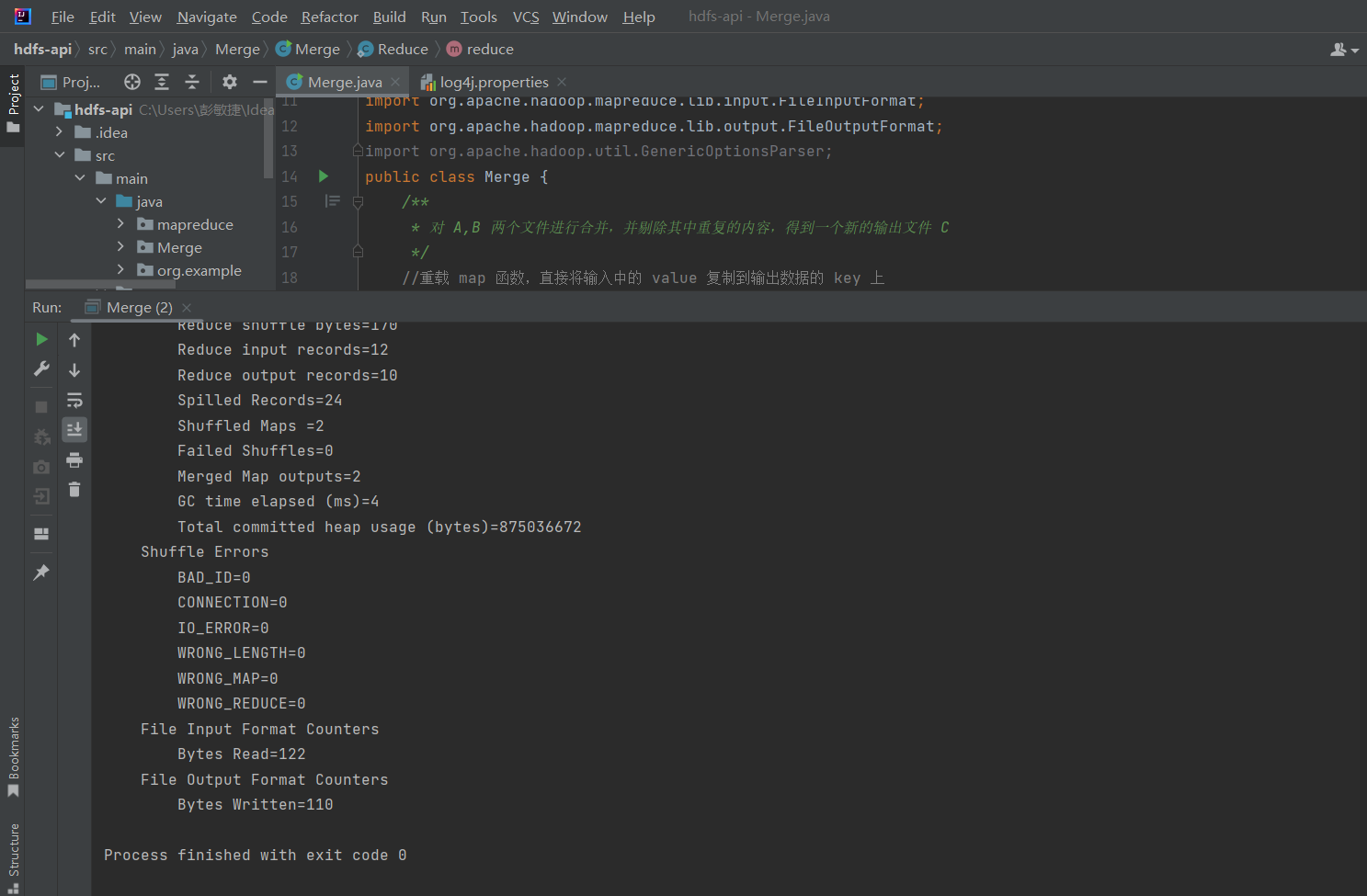
hdfs dfs -put ./file2.txt input

hdfs dfs -cat input/\*

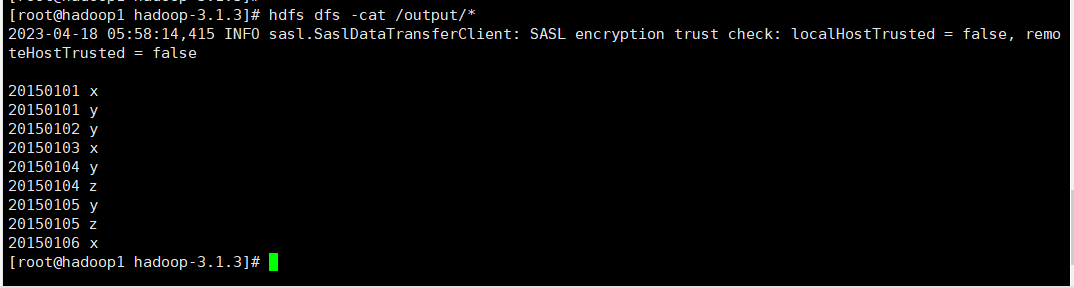


(3)编写代码，在src/main/java下创建Merge包，Merge Java类。

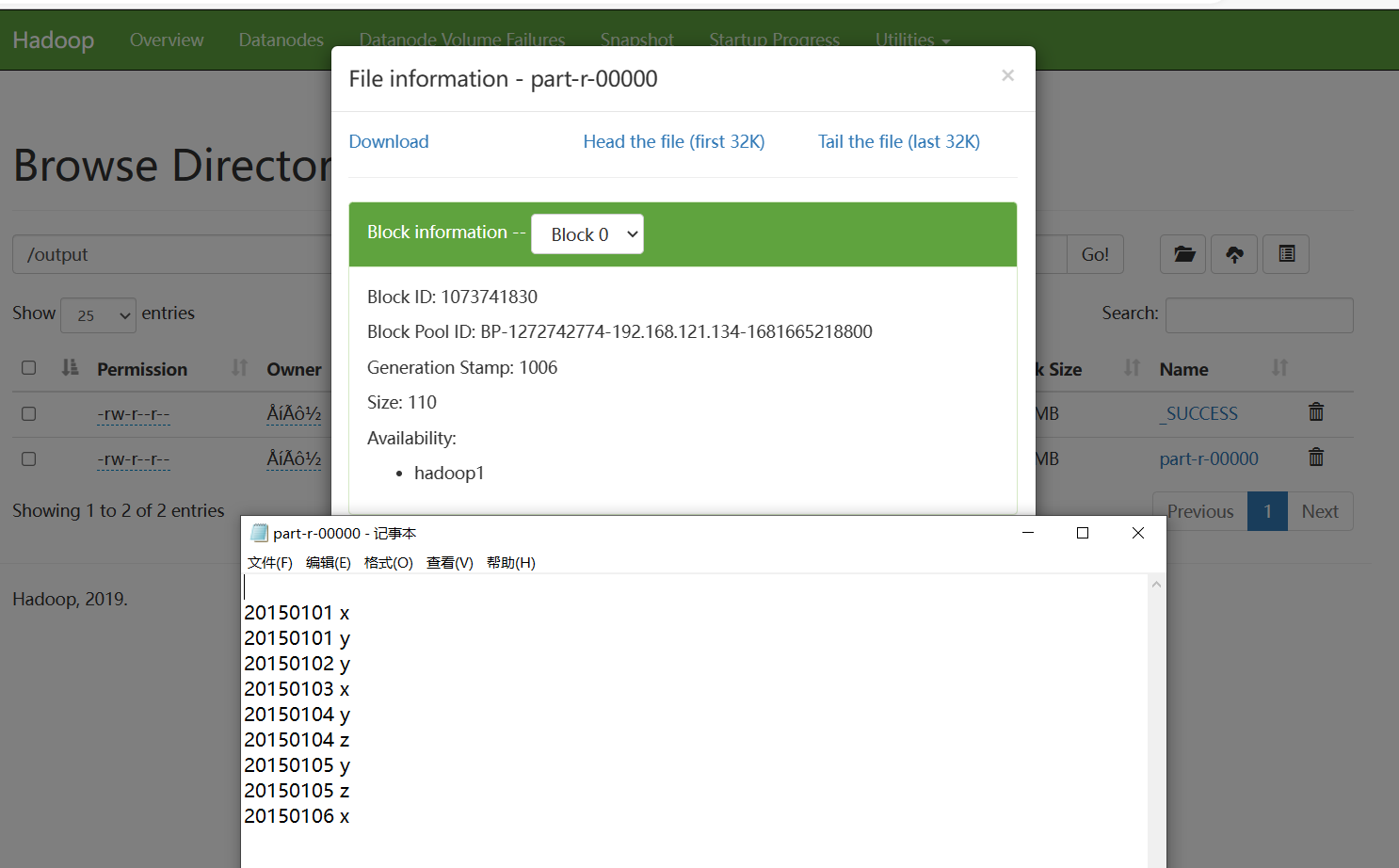
package Merge;  
import java.io.IOException;  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.util.GenericOptionsParser;  
public class Merge {  
 /\*\*  
 \* 对 A,B 两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件 C  
 \*/  
 //重载 map 函数，直接将输入中的 value 复制到输出数据的 key 上  
 public static class Map extends Mapper<Object, Text, Text, Text>{  
 private static Text text = new Text();  
 public void map(Object key, Text value, Context context) throws  
 IOException,InterruptedException{  
 text = value;  
 context.write(text, new Text(""));  
 } }  
  
 //重载 reduce 函数，直接将输入中的 key 复制到输出数据的 key 上  
 public static class Reduce extends Reducer<Text, Text, Text, Text>{  
 public void reduce(Text key, Iterable<Text> values, Context context ) throws IOException,InterruptedException{  
 context.write(key, new Text(""));  
 }  
 }  
  
 public static void main(String[] args) throws Exception{  
 // TODO Auto-generated method stub  
 Configuration conf = new Configuration();  
 conf.set("fs.default.name","hdfs://hadoop1:9000");  
 String[] otherArgs = new String[]{"hdfs://hadoop1:9000/input","hdfs://hadoop1:9000/output"}; /\* 直接设置输入参数  
 \*/  
 if (otherArgs.length != 2) {  
 System.err.println("Usage: wordcount <in><out>");  
 System.exit(2);  
 }  
 Job job = Job.getInstance(conf,"Merge and duplicate removal");  
 job.setJarByClass(Merge.class);  
 job.setMapperClass(Map.class);  
 job.setCombinerClass(Reduce.class);  
 job.setReducerClass(Reduce.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(Text.class);  
 FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }  
}

(4)运行结果。

(5)查看是否输出到output文件hdfs dfs -cat /output/\*



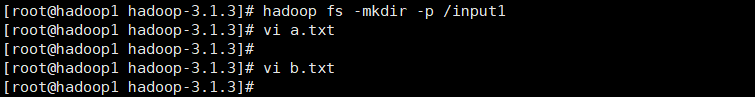
(6)生成part-r-00000文件，查看为去重后的文件。



1. 编写程序实现对输入文件的排序

(1)创建input1文件。 hadoop fs -mkdir -p /input1

创建a.txt和b.txt文件。 vi a.txt ; vi b.txt

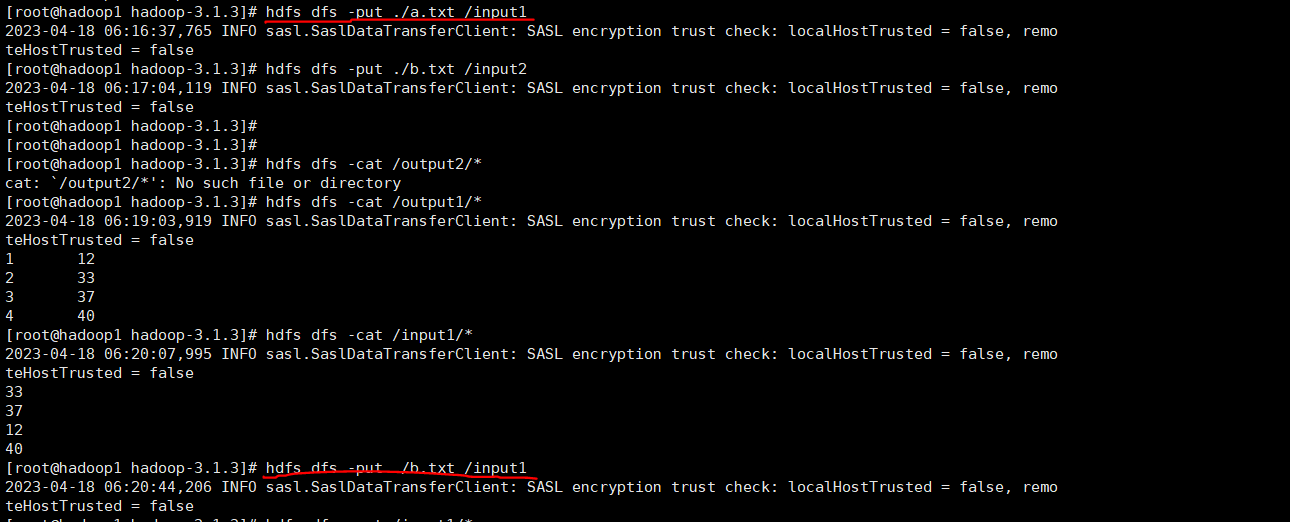


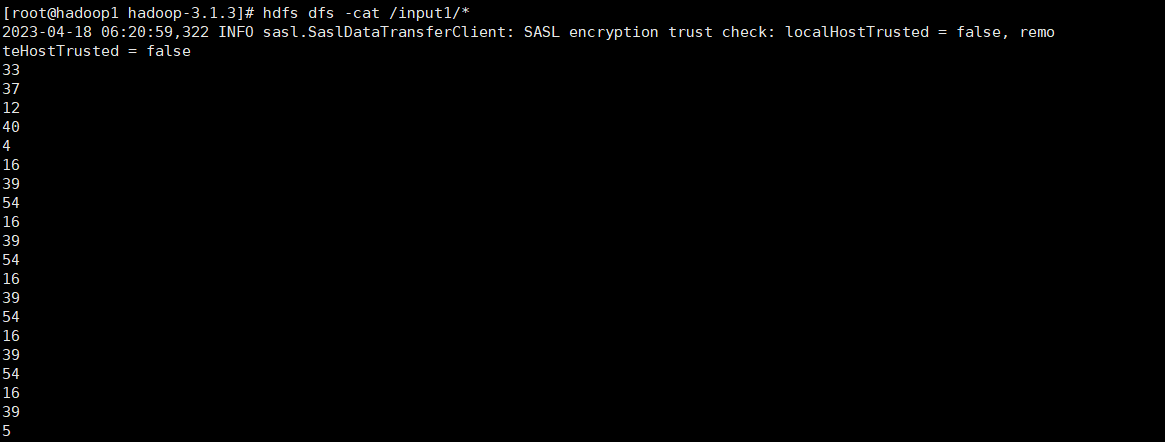
(2)将a.txt和b.txt传入到input1文件中，并查看input1文件。

hdfs dfs -put ./file1.txt input1

hdfs dfs -put ./file2.txt input1

hdfs dfs -cat input1/\*

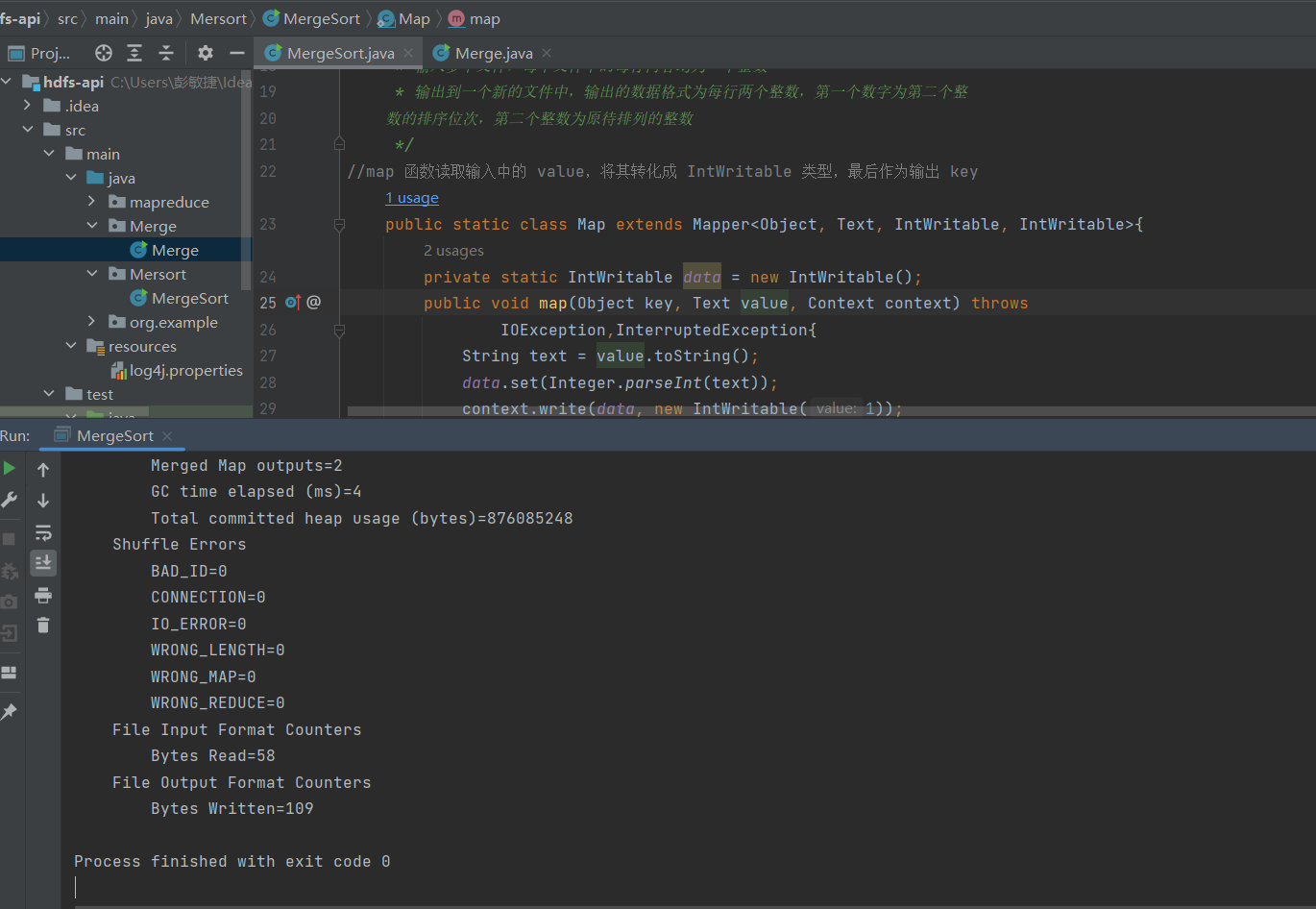




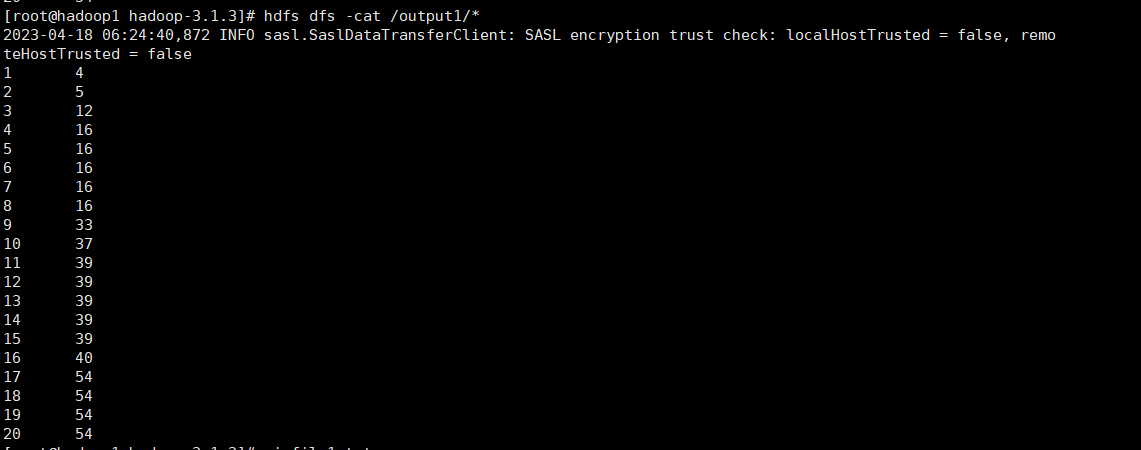
1. 编写代码，在src/main/java下创建Mersort包,MergeSort Java类。

package Mersort;  
import java.io.IOException;  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Partitioner;  
import org.apache.hadoop.mapreduce.Reducer;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.util.GenericOptionsParser;  
  
public class MergeSort {  
 /\*\*  
 \* 输入多个文件，每个文件中的每行内容均为一个整数  
 \* 输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整  
 数的排序位次，第二个整数为原待排列的整数  
 \*/  
//map 函数读取输入中的 value，将其转化成 IntWritable 类型，最后作为输出 key  
 public static class Map extends Mapper<Object, Text, IntWritable, IntWritable>{  
 private static IntWritable data = new IntWritable();  
 public void map(Object key, Text value, Context context) throws  
 IOException,InterruptedException{  
 String text = value.toString();  
 data.set(Integer.parseInt(text));  
 context.write(data, new IntWritable(1));  
 } }  
 //reduce 函数将 map 输入的 key 复制到输出的 value 上，然后根据输入的 value-list  
 //中元素的个数决定 key 的输出次数,定义一个全局变量 line\_num 来代表 key 的位次  
 public static class Reduce extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable>{  
 private static IntWritable line\_num = new IntWritable(1);  
 public void reduce(IntWritable key, Iterable<IntWritable> values, Context context) throws IOException,InterruptedException{  
 for(IntWritable val : values){  
 context.write(line\_num, key);  
 line\_num = new IntWritable(line\_num.get() + 1);  
 } } }  
 //自定义 Partition 函数，此函数根据输入数据的最大值和 MapReduce 框架中Partition 的数量获取将输入数据按照大小分块的边界，然后根据输入数值和边界的关系返回对应的 Partiton ID  
 public static class Partition extends Partitioner<IntWritable, IntWritable>{  
 public int getPartition(IntWritable key, IntWritable value, int num\_Partition){  
 int Maxnumber = 65223;//int 型的最大数值  
 int bound = Maxnumber/num\_Partition+1;  
 int keynumber = key.get();  
 for (int i = 0; i<num\_Partition; i++){  
 if(keynumber<bound \* (i+1) && keynumber>=bound \* i){  
 return i;  
 }  
 }  
 return -1;  
 } }  
 public static void main(String[] args) throws Exception{  
 // TODO Auto-generated method stub  
 Configuration conf = new Configuration();  
 conf.set("fs.default.name","hdfs://hadoop1:9000");  
 String[] otherArgs = new String[]{"hdfs://hadoop1:9000/input1","hdfs://hadoop1:9000/output1"}; /\* 直接设置输入参数  
 \*/  
 if (otherArgs.length != 2) {  
 System.err.println("Usage: wordcount <in><out>");  
 System.exit(2);  
 }  
 Job job = Job.getInstance(conf,"Merge and sort");  
 job.setJarByClass(MergeSort.class);  
 job.setMapperClass(Map.class);  
 job.setReducerClass(Reduce.class);  
 job.setPartitionerClass(Partition.class);  
 job.setOutputKeyClass(IntWritable.class);  
 job.setOutputValueClass(IntWritable.class);  
 FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 } }

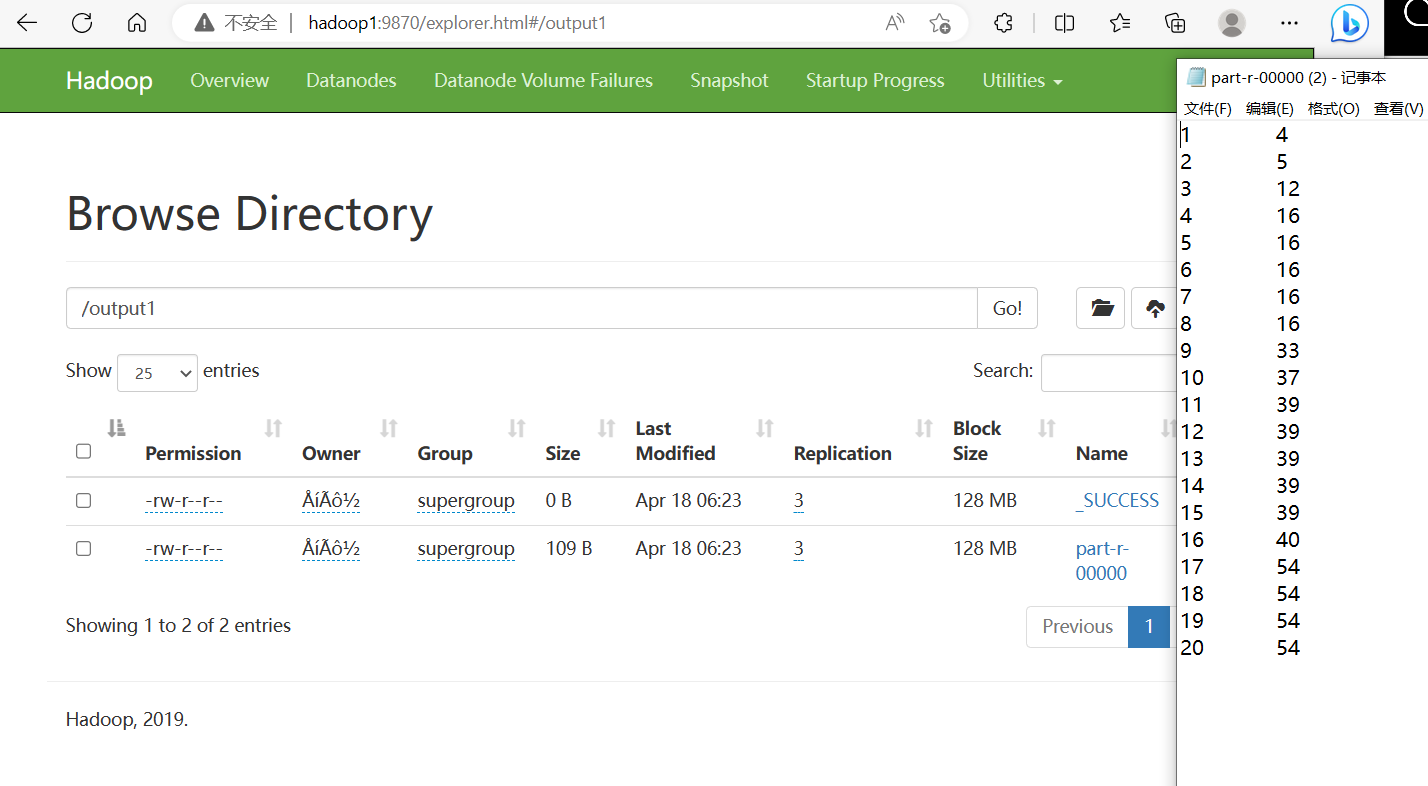
(4)运行结果



1. 查看output1文件

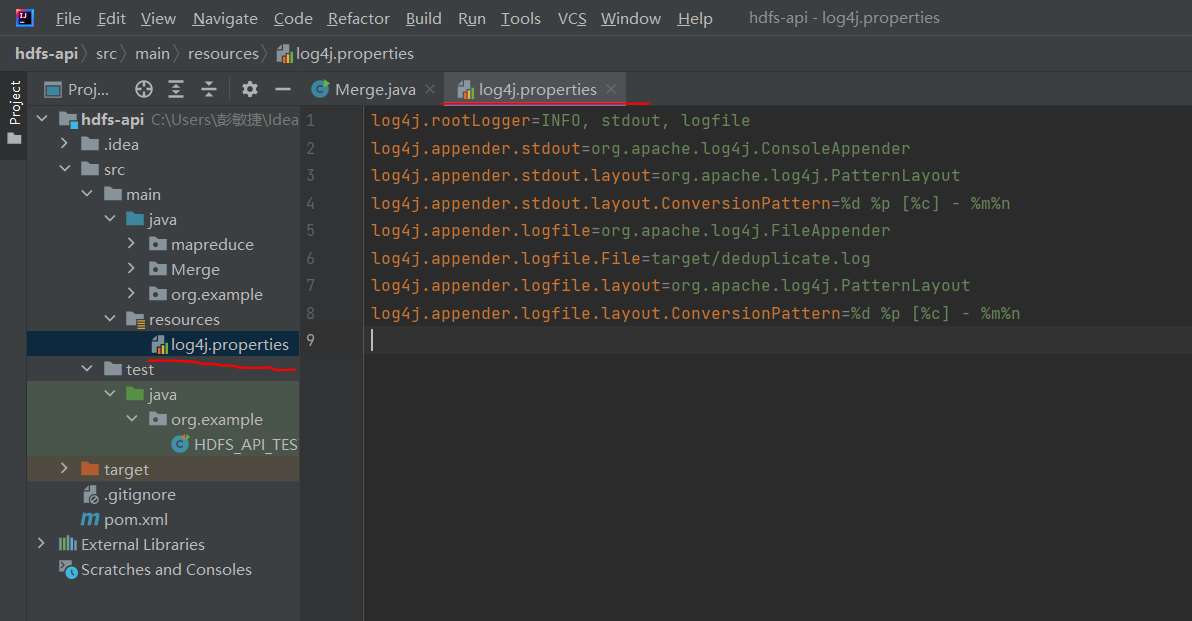


1. 在/output1生成part-r-00000文件

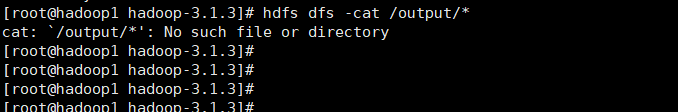


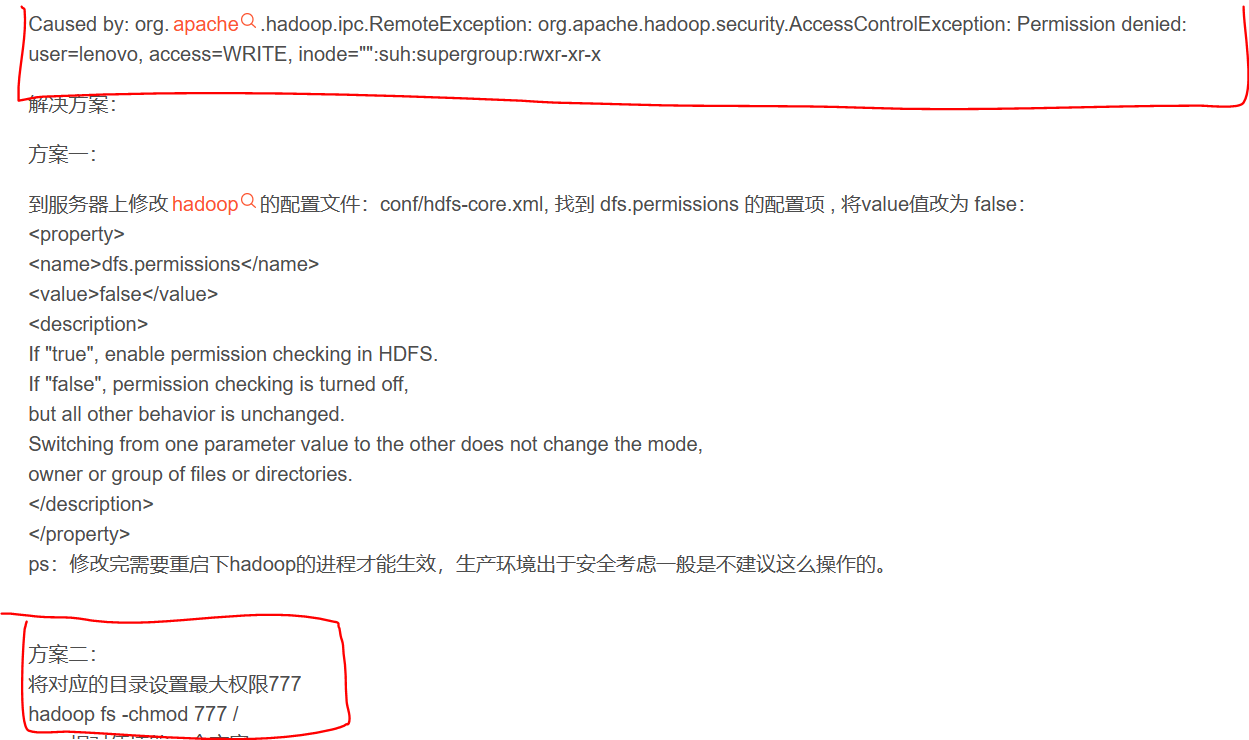
1. **实验总结**

1.在resoures中添加log4j。(不影响结果，但是可以看到报错内容)



1. error:查看不到output文件：hadoop fs -chmod 777 /

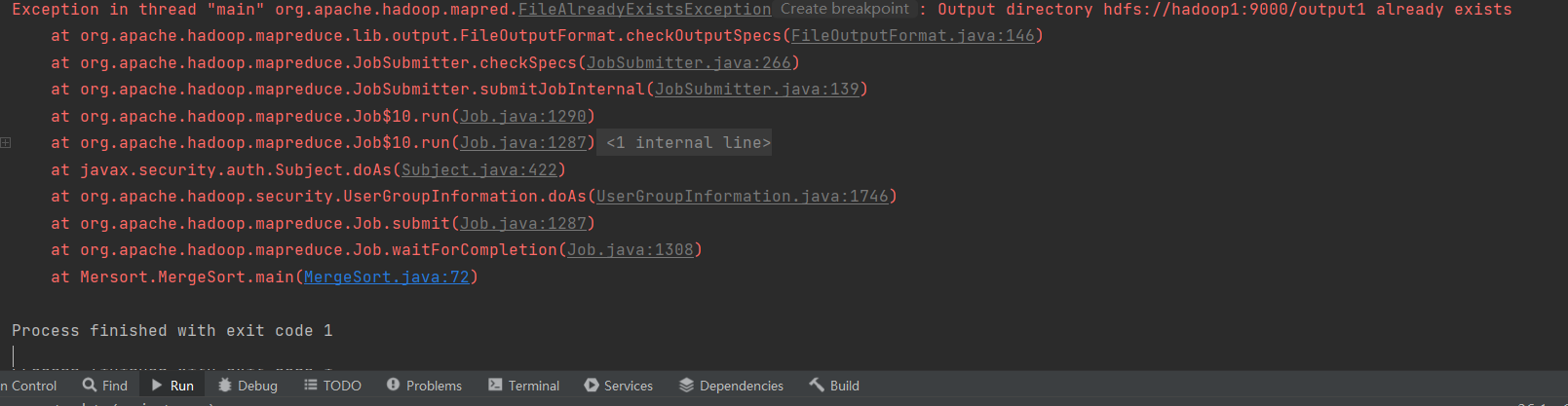




error2

1. Output 已存在：每运行程序一次，都会产生output。

删除output： hdfs dfs -rm -r /output1



error11

5.input的路径要和创建时的路径对应。

