

**数学与信息学院学生实验报告**

**实验课程名称：** 大数据技术概论 **教师：** \_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **MapReduce初级编程实践** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** |  | **年级专业班级** |  |
| **小组成员** | **无** | | | **实验日期** | **2019年 11月** |

# 1. 实验目的和要求

## 1.1 实验目的

* 通过实验掌握基本的MapReduce编程方法；
* 掌握用MapReduce解决一些常见的数据处理问题，包括数据去重、数据排序和数据挖掘等。

## 1.2 实验软硬件环境

* 操作系统：Linux（建议Ubuntu18.04）
* JDK版本：open-jdk-1.11
* Hadoop版本：3.2.1
* IDE：eclipse-4.3

## 1.3 实验步骤

**（一）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，请编写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。下面是输入文件和输出文件的一个样例供参考。

输入文件A的样例如下：

|  |
| --- |
| 20170101 x  20170102 y  20170103 x  20170104 y  20170105 z  20170106 x |

输入文件B的样例如下：

|  |
| --- |
| 20170101 y  20170102 y  20170103 x  20170104 z  20170105 y |

根据输入文件A和B合并得到的输出文件C的样例如下：

|  |
| --- |
| 20170101 x  20170101 y  20170102 y  20170103 x  20170104 y  20170104 z  20170105 y  20170105 z  20170106 x |

**（二）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。下面是输入文件和输出文件的一个样例供参考。

输入文件1的样例如下：

|  |
| --- |
| 33  37  12  40 |

输入文件2的样例如下：

|  |
| --- |
| 4  16  39  5 |

输入文件3的样例如下：

|  |
| --- |
| 1  45  25 |

根据输入文件1、2和3得到的输出文件如下：

|  |
| --- |
| 1 1  2 4  3 5  4 12  5 16  6 25  7 33  8 37  9 39  10 40  11 45 |

**（三）对给定的表格进行信息挖掘**

下面给出一个child-parent的表格，要求挖掘其中的父子辈关系，给出祖孙辈关系的表格。

输入文件内容如下：

|  |
| --- |
| child parent  Steven Lucy  Steven Jack  Jone Lucy  Jone Jack  Lucy Mary  Lucy Frank  Jack Alice  Jack Jesse  David Alice  David Jesse  Philip David  Philip Alma  Mark David  Mark Alma |

输出文件内容如下：

|  |
| --- |
| grandchild grandparent  Steven Alice  Steven Jesse  Jone Alice  Jone Jesse  Steven Mary  Steven Frank  Jone Mary  Jone Frank  Philip Alice  Philip Jesse  Mark Alice  Mark Jesse |

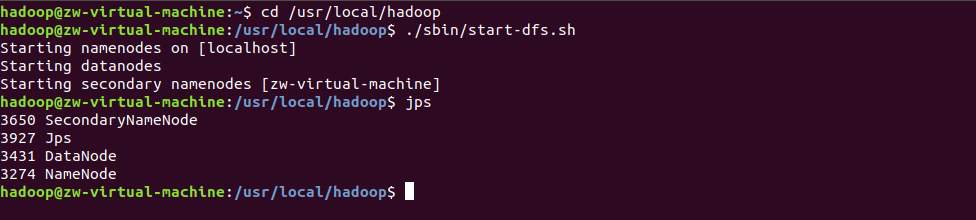
# 2. 实验记录

依次输入以下命令，跳转至Hadoop的相关目录，之后输入命令启动Hadoop。最后，输入命令检查三种进程：NameNode、SecondaryNameNode、DataNode是否都已启动成功。

**cd /usr/local/hadoop**

**./sbin/start-dfs.sh**

**jps**



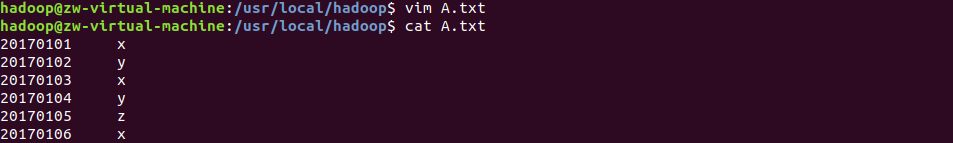
## 2.1 编程实现文件合并与去重操作

### **2.1.1创建输入文件**

·依次输入以下命令，按照题目给出的样例文件分别创建A.txt以及B.txt。之后，查看其中的内容是否已完成写入或是否有误。A.txt和B.txt中的内容即为与题目要求一致的测试内容，接下来，可以开始实验。

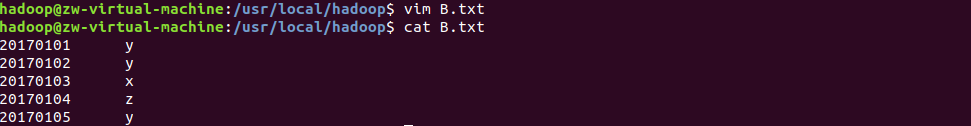
**vim A.txt**

**cat A.txt**



**vim B.txt**

**cat B.txt**



### **2.1.2将输入文件上传至HDFS**

·输入以下命令，创建测试文件夹lab5。

**./bin/hadoop fs -mkdir lab5**



·之后，可以通过浏览器发现要求的路径已创建完毕。



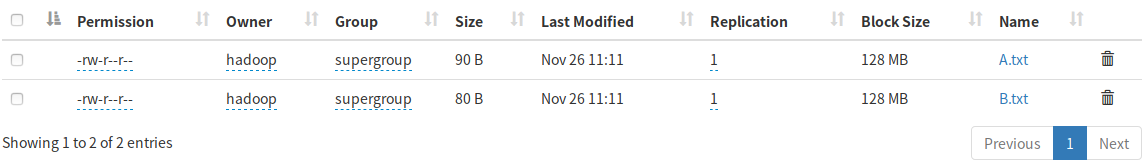
·输入以下命令，分别将输入文件A.txt以及B.txt上传至HDFS中的lab5文件夹中。

**/bin/hadoop fs -put A.txt ./lab5/**

**./bin/hadoop fs -put B.txt ./lab5/**

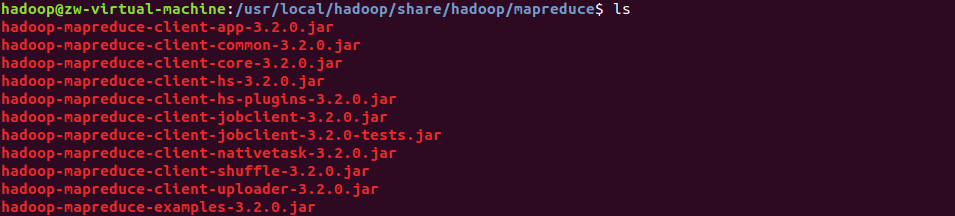


·通过浏览器可查看是否已经上传完成。可以看到，本地文件A.txt以及B.txt已上传至HDFS中的lab5路径下。



### **2.1.3 MapReduce Java编程**

·根据Hadoop的版本，导入相应的mapreduce的jar包，所需包在Hadoop文件包中的mapreduce文件路径下。其中，一个比较关键的jar包：**hadoop-mapreduce-client-core-3.2.0.jar**，后续编程所需的一些重要的类在此包中。



·开始编写MapReduce程序，主要步骤如下：编写Map处理逻辑、编写Reduce处理逻辑、编写main方法、编译打包代码以及运行程序。

**·一些关键类的介绍：**

## Class Job

该类允许用户配置作业、提交作业、控制作业的执行并查询状态。其中，set方法只在提交作业之前有效，之后会抛出一个IllegalStateException。通常情况下, 通常，用户创建应用程序，通过Job描述作业的各个方面，然后提交作业并监视其进度。

## Class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

Mapper将输入键/值对映射到一组中间键/值对。映射是将输入记录转换为中间记录的相互独立的任务群。Hadoop MapReduce框架为作业的InputFormat生成的每个InputSplit生成一个map任务。

总的来说，mapper实现是通过job.setmapperclass (Class)方法传递给作业的。然后，框架为该任务的InputSplit中的每个键/值对调用map(WritableComparable,Writable,Context)。

转换后的中间记录不要求与输入记录具有相同的类型。一个给定的输入对可以映射到零个或多个输出对。通过调用context.write(WritableComparable, Writable)来收集输出对。

## Class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

Reducer合并一组共享同一个键的中间值，让它们的数量变得更少。作业的reduce数量由用户通过方法Job.setNumReduceTasks(int)设置。

总的来说，Reducer实现是通过Job.setReducerClass(Class)方法传递Job给作业，并且可以覆盖它来进行初始化。然后，框架为分组输入中的每个 <key, (list of values)>对调用reduce(WritableComparable, Iterable<Writable>, Context) 方法。

Reducer有三个主要阶段：Shuffle，Sort和Reduce。

**·思路：**map阶段的<key,value>情况

每一行都以值为key，例：

|  |  |
| --- | --- |
| **Key** | **Value** |
| 20170101 x |  |
| 20170102 y |  |
| 20170103 x |  |
| 20170104 y |  |

之后，经过shuffle，这些键值对会自动根据key值合并到达去重的效果，因此直接写入即可。

**·代码段：**

**import** java.io.IOException;

**import** java.util.Scanner;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**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;

**public** **class** Merge {

//Map函数

**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(""));

}

}

//Reduce函数

**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 {

Configuration conf = **new** Configuration();

conf.set("fs.defaultFS", "hdfs://localhost:9000");

String[] otherArgs = **new** String[]{"lab5/A.txt", "lab5/B.txt", "output\_lab501"};

//给定的输入输出路径少于2

**if** (otherArgs.length < 2)

{

System.***err***.println("Usage:Join <in> <out>");

System.*exit*(2);

}

Path path = **new** Path(otherArgs[2]);

FileSystem fileSystem = path.getFileSystem(conf);

//判断给定的输出路径是否存在

**if** (fileSystem.exists(path))

{

Scanner scan = **new** Scanner(System.***in***);

System.***err***.print("The output directory already exits!Delete it?(yes/no)");

String mark = scan.nextLine();

scan.close();

//删除

**if**(mark.equalsIgnoreCase("yes"))

{

fileSystem.delete(path, **true**);

}

//退出

**else**

{

System.***out***.println("exited.");

System.*exit*(0);

}

}

Job job = Job.getInstance(conf, "Merge"); //设置环境参数

job.setJarByClass(Merge.**class**); //设置整个程序的类名

job.setMapperClass(Map.**class**); //添加Mapper类

job.setReducerClass(Reduce.**class**); //添加Reducer类

job.setOutputKeyClass(Text.**class**); //设置输出类型

job.setOutputValueClass(Text.**class**); //设置输出类型

FileInputFormat.addInputPath(job, **new** Path(otherArgs[0])); //设置输入文件

FileInputFormat.addInputPath(job, **new** Path(otherArgs[1])); //设置输入文件

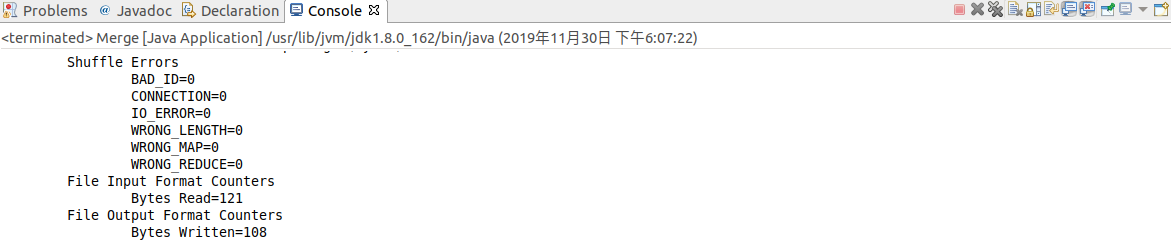
FileOutputFormat.setOutputPath(job, **new** Path(otherArgs[2])); //设置输出文件

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

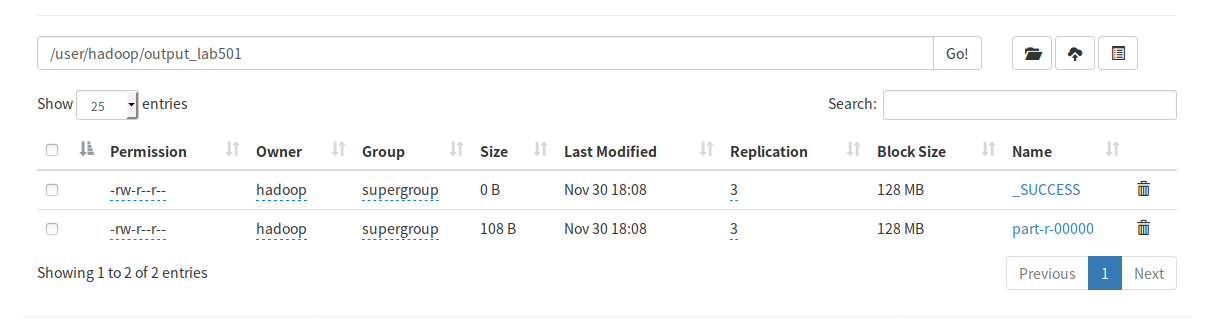
}

}

·**运行结果：**配置完log4j.properties后会显示执行的过程，最后显示输出文件的大小，这里即为108B。



·运行成功后，可以通过浏览器发现output\_lab501文件夹已成功创建，生成了两个文件\_SUCCESS和part-r-00000。另外，可以发现文件part-r-00000的大小为108B，与Eclipse终端输出结果相一致。

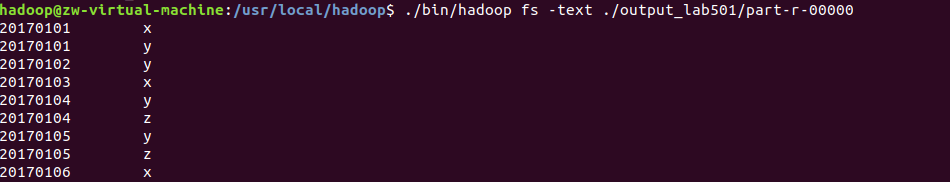


**问：生成的文件\_SUCCESS和文件part-r-00000分别是什么？**

答：在成功完成作业后，MapReduce运行时将在输出目录中创建一个\_SUCCESS文件。因此，这对于需要通过检查HDFS来查看结果集是否完整的应用程序非常有用。输出文件默认名为part-x-yyyyy，其中：x是'm'还是'r'，取决于作业是map作业还是reduce作业；yyyyy是map或reduce任务编号（基于零）。因此，一个有32个reducer的工作将会有名为part-r-00000到part-r-00031的文件，每个reducer任务一个编号。

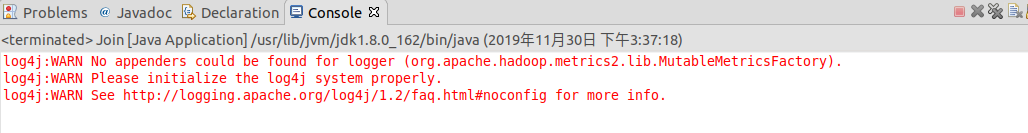
·之后，在终端输入以下命令来查看生成的part-r-00000文件中的内容，可以发现与题目要求的结果相一致，达到了去重的目标。

**./bin/hadoop fs -text ./output\_lab501/part-r-00000**

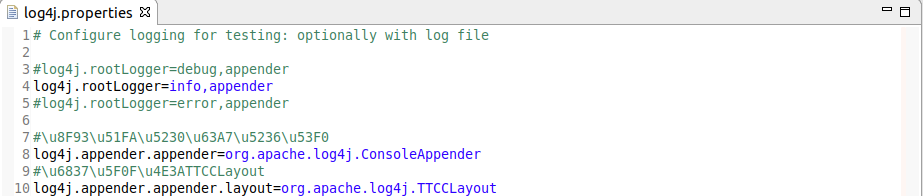


**问：log4j警告是怎么回事？**

答：缺少了log4j.properties文件，无法找到日志文件，日志记录无法正常显示。解决方法，在当前文件目录src下添加log4j.properties文件即可（终端的解决办法）。

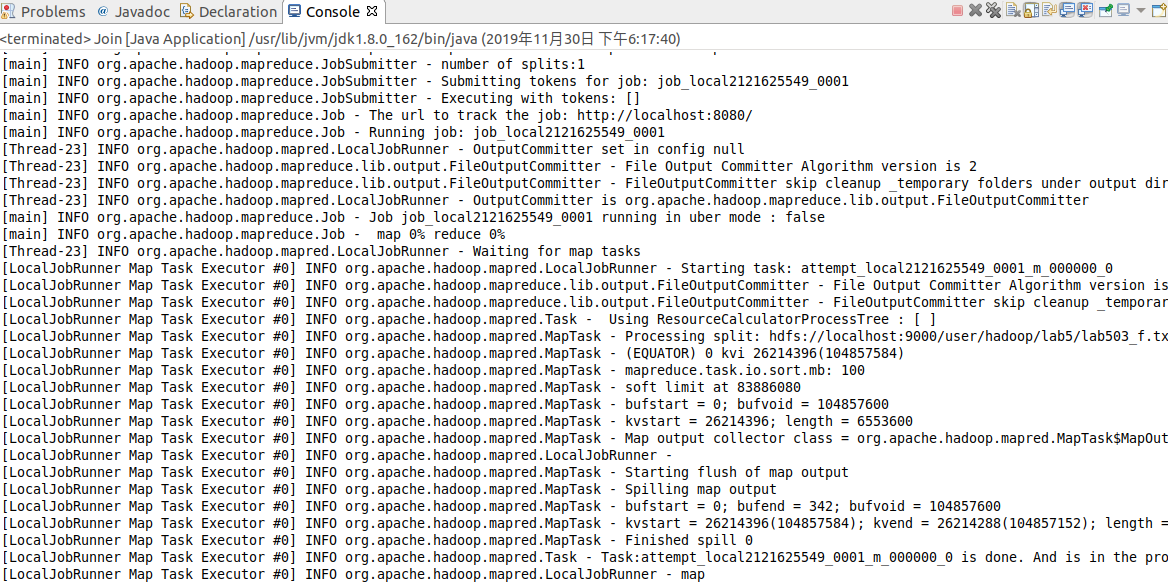


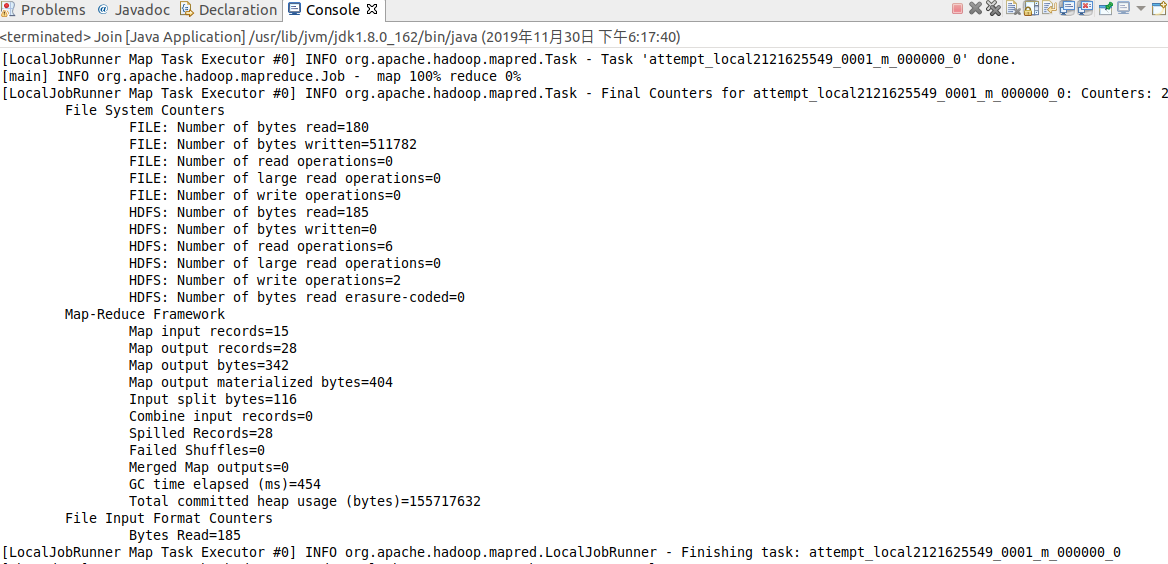
附：log4j.properties文件中的内容



**问：只是警告未报错，为什么还要配置log4j.properties文件呢？**

答：配置log4j.properties文件后，程序运行的详细过程能在控制台终端显示出来，包括map、shuffle、reduce的执行情况，若程序结果出错，我们可以通过日志记录来分析原因。部分日志内容如下：

****

****

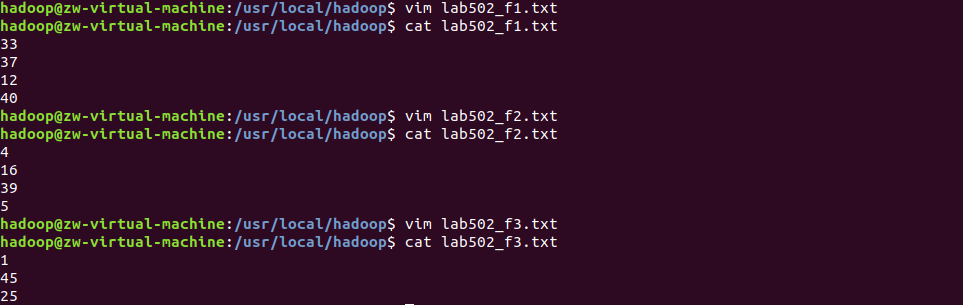
## 2.2 编程实现对输入文件的排序

### **2.2.1 创建输入文件**

·依次输入以下命令，按照题目给出的样例文件分别创建lab502\_f1.txt、lab502\_f2.txt以及lab502\_f3.txt。之后，查看其中的内容是否已完成写入或是否有误。可以发现文件lab502\_f1.txt、lab502\_f2.txt以及lab502\_f3.txt中的内容与题目要求相一致，接下来，可以开始实验。

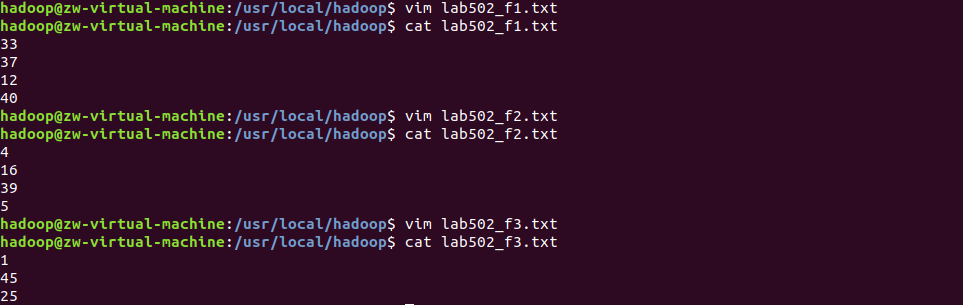
**vim lab502\_f1.txt**

**cat lab502\_f1.txt**



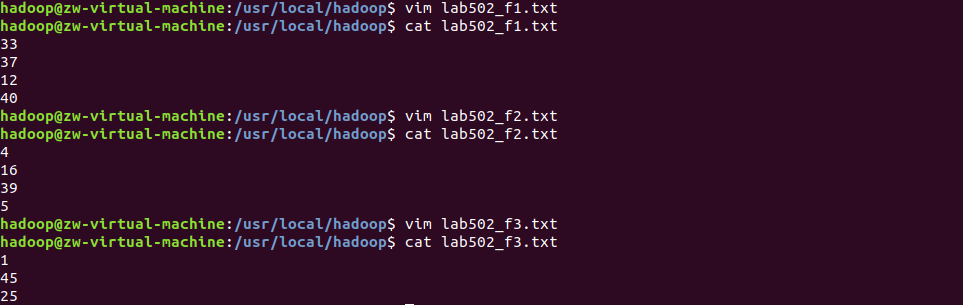
**vim lab502\_f2.txt**

**cat lab502\_f2.txt**



**vim lab502\_f3.txt**

**cat lab502\_f3.txt**



### **2.2.2将输入文件上传至HDFS**

·输入以下命令，分别将输入文件lab502\_f1.txt、lab502\_f2.txt以及lab502\_f3.txt上传至HDFS中的lab5文件夹中。

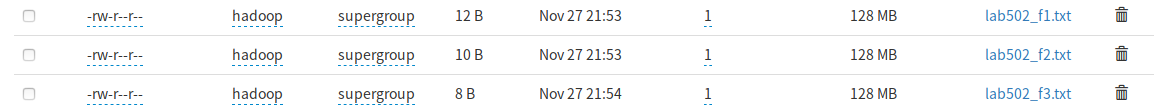
**./bin/hadoop fs -put lab502\_f1.txt ./lab5/**

**./bin/hadoop fs -put lab502\_f2.txt ./lab5/**

**./bin/hadoop fs -put lab502\_f3.txt ./lab5/**



·通过浏览器可查看是否已经上传完成。可以看到，本地文件lab502\_f1.txt、lab502\_f2.txt以及lab502\_f3.txt已上传至HDFS中的lab5路径下。



### **2.2.3 MapReduce Java编程**

·开始编写MapReduce程序，主要步骤如下：编写Map处理逻辑、编写Reduce处理逻辑、编写main方法、编译打包代码以及运行程序。

**·思路：**map阶段的<key,value>情况

每一行都以值为key，例：

|  |  |
| --- | --- |
| **Key** | **Value** |
| 33 | 1 |
| 37 | 1 |
| 12 | 1 |
| 40 | 1 |

之后，经过shuffle，这些键值对会按key排好序，只需获取当前行数即可知道在序列中的顺序。

**·代码段：**

**import** java.io.IOException;

**import** java.util.Scanner;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FileSystem;

**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;

**public** **class** Sort {

//Map函数

**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 line = value.toString();

*data*.set(Integer.*parseInt*(line));

context.write(*data*, **new** IntWritable(1));

}

}

//Reduce函数

**public** **static** **class** Reduce **extends** Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {

**private** **static** IntWritable *linenum* = **new** IntWritable(1);

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

**for** (IntWritable val : values)

{

context.write(*linenum*, key);

*linenum* = **new** IntWritable(*linenum*.get() + 1);

}

}

}

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

Configuration conf = **new** Configuration();

conf.set("fs.defaultFS", "hdfs://localhost:9000");

String[] otherArgs = **new** String[]{"lab5/lab502\_\*.txt", "output\_lab502"};

//给定的输入输出路径少于2或多于2

**if** (otherArgs.length != 2)

{

System.***err***.println("Usage: Sort <in> <out>");

System.*exit*(2);

}

Path path = **new** Path(otherArgs[1]);

FileSystem fileSystem = path.getFileSystem(conf);

//判断给定的输出路径是否存在

**if** (fileSystem.exists(path))

{

Scanner scan = **new** Scanner(System.***in***);

System.***err***.print("The output directory already exits!Delete it?(yes/no)");

String mark = scan.nextLine();

scan.close();

//删除

**if**(mark.equalsIgnoreCase("yes"))

{

fileSystem.delete(path, **true**);

}

//退出

**else**

{

System.***out***.println("exited.");

System.*exit*(0);

}

}

Job job = Job.getInstance(conf, "Sort"); //设置环境参数

job.setJarByClass(Sort.**class**); //设置整个程序的类名

job.setMapperClass(Map.**class**); //添加Mapper类

job.setReducerClass(Reduce.**class**); //添加Reducer类

job.setOutputKeyClass(IntWritable.**class**); //设置输出类型

job.setOutputValueClass(IntWritable.**class**); //设置输出类型

FileInputFormat.addInputPath(job, **new** Path(otherArgs[0])); //设置输入文件

FileOutputFormat.setOutputPath(job, **new** Path(otherArgs[1])); //设置输出文件

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

}

}

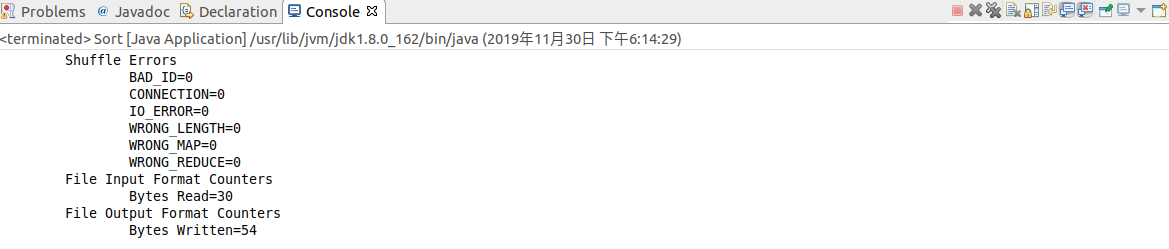
**问：MapReduce中的IntWritable是指什么？**

IntWritable是 Hadoop 中实现的用于封装 Java 数据类型的类，它的原型是public IntWritable(int value)和public IntWritable()两种。所以new IntWritable(1)创建了这个类的一个对象，而数值1是参数。在Hadoop中，它相当于Java中Integer整型变量，为这个变量赋值为1。

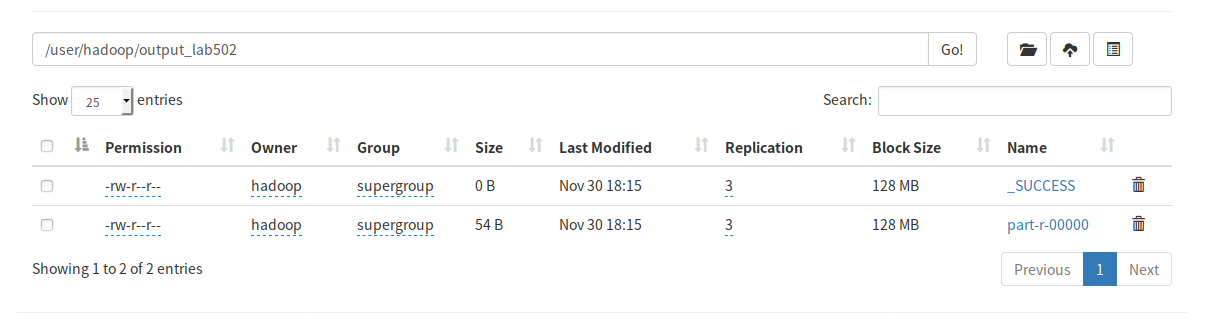
·get():Return the value of this IntWritable.

·set():Set the value of this IntWritable.

·**运行结果：**配置完log4j.properties后会显示执行的过程，最后显示输出文件的大小，这里即为54B。

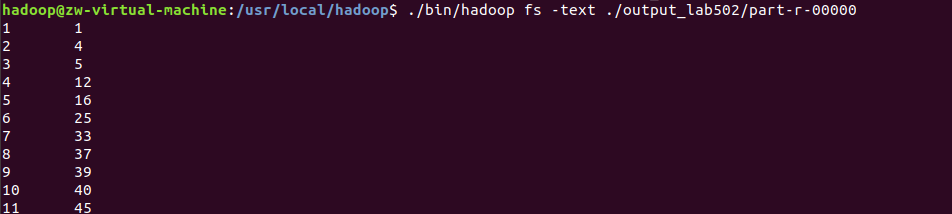


·运行成功后，可以通过浏览器发现output\_lab502文件夹已成功创建，生成了两个文件\_SUCCESS和part-r-00000。另外，可以发现文件part-r-00000的大小为54B，与Eclipse终端输出结果相一致。



·之后，在终端输入以下命令来查看生成的part-r-00000文件中的内容，可以发现与题目要求的结果相一致，达到了排序的目标。

**./bin/hadoop fs -text ./output\_lab502/part-r-00000**



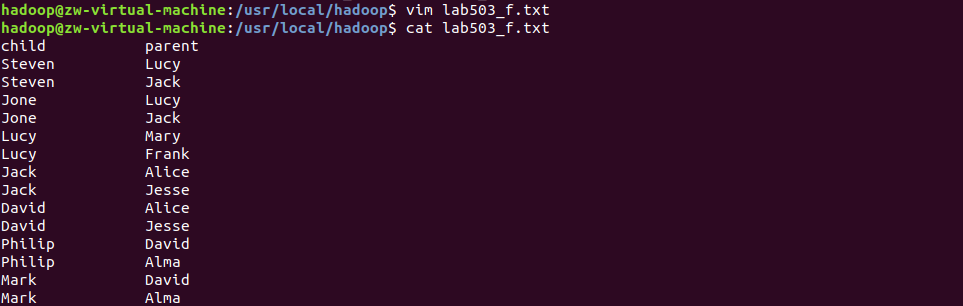
## 2.3 编程实现对给定的表格进行信息挖掘

### **2.3.1 创建输入文件**

·依次输入以下命令，按照题目给出的样例文件分别创建lab503\_f.txt。之后，查看其中的内容是否已完成写入或是否有误。lab503\_f.txt中的内容即为与题目要求一致的测试内容，接下来，可以开始实验。

**vim lab503\_f.txt**

**cat lab503\_f.txt**



### **2.3.2将输入文件上传至HDFS**

·输入以下命令，分别将输入文件lab503\_f.txt上传至HDFS中的lab5文件夹中。

**./bin/hadoop fs -put lab503\_f.txt ./lab5/**



·通过浏览器可查看是否已经上传完成。可以看到，本地文件lab503\_f.txt已上传至HDFS中的lab5路径下。



### **2.3.3 MapReduce Java编程**

·开始编写MapReduce程序，主要步骤如下：编写Map处理逻辑、编写Reduce处理逻辑、编写main方法、编译打包代码以及运行程序。

**·思路：**map阶段的<key,value>情况

每一行都以下的方式写两遍：**<parent,-child>、<child,+parent>**。例：

|  |  |
| --- | --- |
| **Key** | **Value** |
| **Lucy——(same)** | -Steven——**(grandchild)** |
| Steven | +Lucy |
| Frank | -Lucy |
| **Lucy——(same)** | +Frank——**(grandparent)** |

之后，相同的key会被分到同一个Reduce任务，若存在两个键值对的key相同且其中一个以“-”开头而另一个的value以“+”开头，则这两个value为grandchild-grandparent关系。

**·代码段：**

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**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** java.util.StringTokenizer;

**public** **class** Join {

//Map函数

**public** **static** **class** Map **extends** Mapper<Object, Text, Text, Text> {

**private** **static** **int** *linenum* = 0;

**public** **void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {

String line = value.toString();

**if** (*linenum* == 0)

{

++*linenum*;

}

**else**

{

StringTokenizer tokenizer = **new** StringTokenizer(line, "\n");

**while** (tokenizer.hasMoreElements())

{

StringTokenizer lineTokenizer = **new** StringTokenizer(tokenizer.nextToken());

String child = lineTokenizer.nextToken();

String parent = lineTokenizer.nextToken();

context.write(**new** Text(parent), **new** Text("-" + child));

context.write(**new** Text(child), **new** Text("+" + parent));

}

}

}

}

//Reduce函数

**public** **static** **class** Reduce **extends** Reducer<Text, Text, Text, Text> {

**private** **static** **int** *linenum* = 0;

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

**if** (*linenum* == 0)

{

context.write(**new** Text("grandchild"), **new** Text("grandparent"));

++*linenum*;

}

ArrayList<Text> grandChild = **new** ArrayList<Text>();

ArrayList<Text> grandParent = **new** ArrayList<Text>();

**for** (Text val : values)

{

String s = val.toString();

**if** (s.startsWith("-"))

{

grandChild.add(**new** Text(s.substring(1)));

}

**else**

{

grandParent.add(**new** Text(s.substring(1)));

}

}

**for** (Text text1 : grandChild)

{

**for** (Text text2 : grandParent)

{

context.write(text1, text2);

}

}

}

}

**public** **static** **void** main(String[] args) **throws** IOException, ClassNotFoundException, InterruptedException {

Configuration conf = **new** Configuration();

conf.set("fs.defaultFS", "hdfs://localhost:9000");

String[] otherArgs = **new** String[]{"lab5/lab503\_f.txt", "output\_lab503"};

//给定的输入输出路径少于2或多于2

**if** (otherArgs.length != 2)

{

System.***err***.println("Usage:Join <in> <out>");

System.*exit*(2);

}

Path path = **new** Path(otherArgs[1]);

FileSystem fileSystem = path.getFileSystem(conf);

//判断给定的输出路径是否存在

**if** (fileSystem.exists(path))

{

Scanner scan = **new** Scanner(System.***in***);

System.***err***.print("The output directory already exits!Delete it?(yes/no)");

String mark = scan.nextLine();

scan.close();

//删除

**if**(mark.equalsIgnoreCase("yes"))

{

fileSystem.delete(path, **true**);

}

//退出

**else**

{

System.***out***.println("exited.");

System.*exit*(0);

}

}

Job job = Job.getInstance(conf, "Join"); //设置环境参数

job.setJarByClass(Join.**class**); //设置整个程序的类名

job.setMapperClass(Map.**class**); //添加Mapper类

job.setReducerClass(Reduce.**class**); //添加Reducer类

job.setOutputKeyClass(Text.**class**); //设置输出类型

job.setOutputValueClass(Text.**class**); //设置输出类型

FileInputFormat.addInputPath(job, **new** Path(otherArgs[0])); //设置输入文件

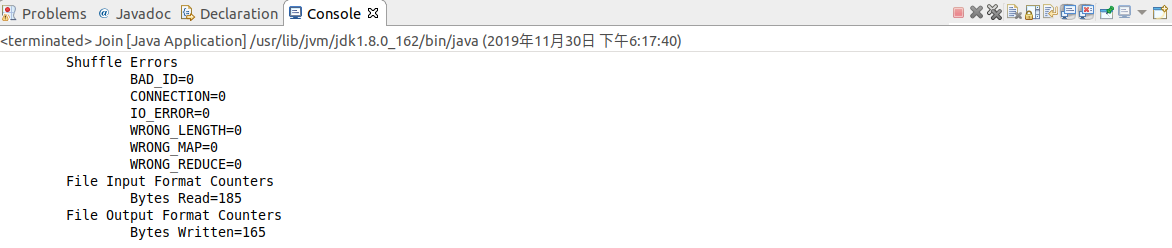
FileOutputFormat.setOutputPath(job, **new** Path(otherArgs[1])); //设置输出文件

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

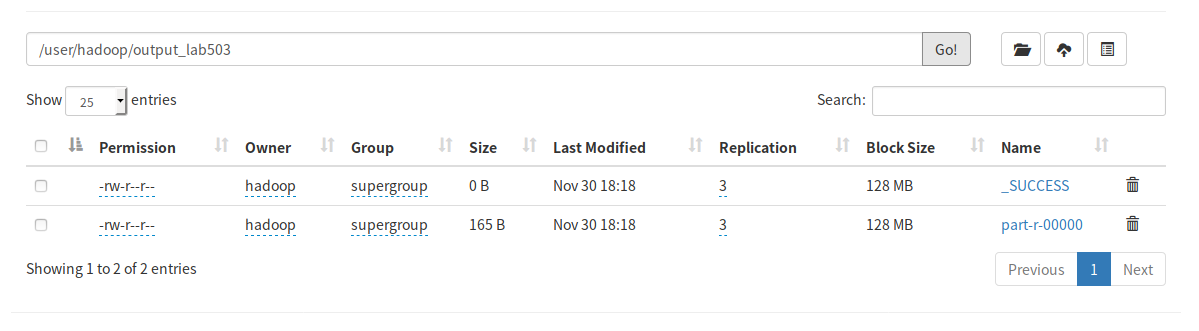
}

}

·**运行结果：**配置完log4j.properties后会显示执行的过程，最后显示输出文件的大小，这里即为165B。

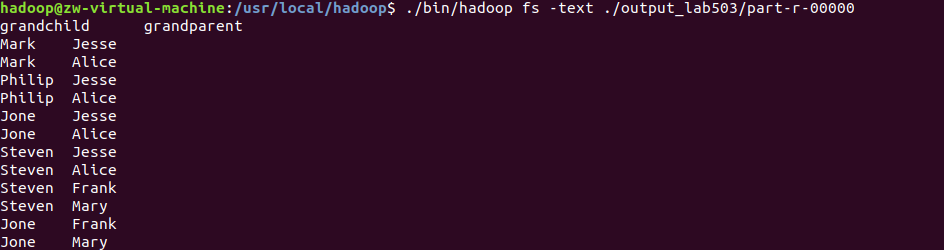


·运行成功后，可以通过浏览器发现output\_lab503文件夹已成功创建，生成了两个文件\_SUCCESS和part-r-00000。另外，可以发现文件part-r-00000的大小为165B，与Eclipse终端输出结果相一致。



·之后，在终端输入以下命令来查看生成的part-r-00000文件中的内容，可以发现与题目要求的结果相一致，达到了连接的目标。

**./bin/hadoop fs -text ./output\_lab503/part-r-00000**



# 3. 实验总结

大数据的最后一个实验，庆祝一下。还是让自己醒醒，继续写总结。

至于适合用MapReduce来处理的数据集需要满足这样一个前提条件：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。当然，在写这个实验的过程中基本无需考虑这一点，如果题目不适用于MapReduce解决的话，放在这里干嘛呢。

因此，让我深有体会的另有其“人”。

在写这个实验的过程中，联想到作为面向对象的三大特点之一的封装，即隐藏对象的属性和实现细节，仅对外公开接口。它弱化抽象细节，不仅称得上用户友好型，对于现在的我来说，更算是编程“拙劣”人员友好型。在MapReduce中，也算是有这么一个抽象的过程——shuffle。（ps：我在看书的过程中就被它惊叹到了）

在编程过程中，Map和Reduce操作需要我们自己定义相应Map类和Reduce类，以完成我们所需要的化简、合并操作，而shuffle则是系统自动帮我们实现的，了解shuffle的具体流程能帮助我们编写出更加高效的Mapreduce程序。shuffle简直是MapReduce中的小天使！可以说，有了shuffle过程，我们设计Map函数和Reduce函数的时候考虑的问题就能少一些。

官方文档中关于shuffle的定义如下，虽然很简短，但它的能力很强：对Map输出结果进行分区、排序、合并等处理，之后交给Reduce。

**Shuffle**

Input to the Reducer is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers, via HTTP.

因此，我们设计时候基本上都极大地依赖了shuffle过程的强大功能，至少在本实验的情况下尤为明显，第一、二题更为明显。

最后，再随便说说本学期的这些实验们，先吐槽一点，这门课绝对是我到现在需要装软件的数量最多的一门。但现在仔细想想，能广涉略也不错，并且没有什么问题是重装解决不了的。想想一开始安装Hadoop的困难重重，到现在已经可以做一些算是基本的东西了。当然，要学的东西还很多，大数据之大，深不可测，学无止境。

关于实验的本身，也算是遗憾的一点，一时兴起有感而发，中规中矩的答案在这里显然仅可能止步高于平均一两分。说实话，一些报告有附加额外做的东西，超出规则规定之外，最后满分。当然，相比中规中矩的，有它们各自的亮点。但我第一反应就是量大超额。写报告的过程中，我也在思考报告应该记录些什么。我该体现这个过程的困难重重、详细记录一点一滴，还是省去一些错误呈现出顺风顺水的假象呢？记录错误固然很重要的，包括测试文档，我们会碰到很多问题，更别说刚上手的新事物，但解决过程以及方法我还是不大体现在报告中，这就是缺少过程吧。

以上就是这次报告的全部内容。