

**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** | 实验五：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** |  |

### 一、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。

### 二、实验平台

* 操作系统：LinuxUbuntu16.04
* Hadoop版本：2.7.1

### 三、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 |

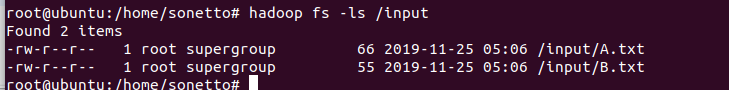
输入文件B的样例如下：

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

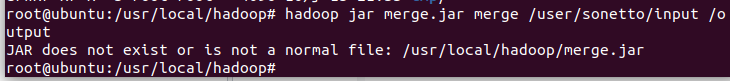
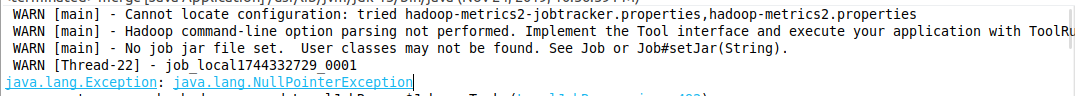
根据输入文件A和B合并得到的输出文件C的样例如下：

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

* 前期准备文件：



* 出错：

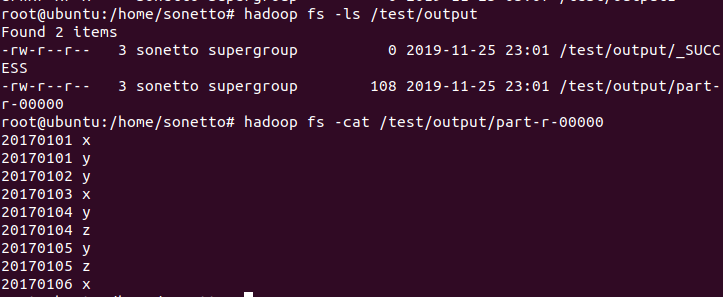


尝试 打包进去 结果出现一样的错

* 最后发现出错原因：

context.write的value被我设置成了null，但是这样不行。应该要有text类型（前面设置的输出类型）的内容。

* 成功结果：



* 代码：

package mapreduce;

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 {

public static class MergeMapper extends Mapper<Object, Text, Text, Text>

{

/\*写在public class里面，前两个参数 设置Mapper的输入类型 key为Obj，value为Text；

输出类型：key为TEXT，Value是text类型

根据业务类型自己定义

这些类型都是在hadoop里面定义的,更加高效\*/

private static Text text = new Text();

//value为text指每次读文本的一行

//Context类型是Map函数的一种输出方式，存储中间结果

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

{

text = value;

context.write(text, new Text("")); //对应前面继承的mapper输出类型：Text,text

//我的理解是是把输入的value设置为输出的key，做到筛选

查： MapReduce在shuffle阶段，默认帮我们将数据做了排序，并且是做了合并，相同的数据归为一组了。

}

}

public static class MergeReducer extends Reducer<Text, Text, Text, Text>

{

//Iterable是迭代容器接口，泛型规定是Text类型，可以遍历里面的值

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

{

context.write(key, new Text(""));//此处key为map函数的text

}

}

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

Configuration conf = new Configuration();

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

// 我本来以为是本地的路径

// String A = "/home/sonetto/A.txt";

// String B = "/home/sonetto/B.txt";

// String C = "/home/sonetto/C.txt";

String[] str = new String[] {"/input","/test/output"};

1.如果只写“input”,”output”，那么程序默认的是hadoop用户的目录下的input,output

如：/user/sonetto/input

2.设置成字符串数组类型可以用shell跑，但是eclipse不设置字符数组也可以。

//Job类型设置程序运行的环境变量

Job job = Job.getInstance(conf);

//设置程序的类名

job.setJarByClass(merge.class);

//继承的类 :前面自己定义的

job.setMapperClass(MergeMapper.class);

job.setCombinerClass(MergeReducer.class);

job.setReducerClass(MergeReducer.class);

//设置输入输出类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

//设置输入、输出文件

FileInputFormat.addInputPath(job, new Path(str[0]));

FileOutputFormat.setOutputPath(job, new Path(str[1]));

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

}

}

**（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 |

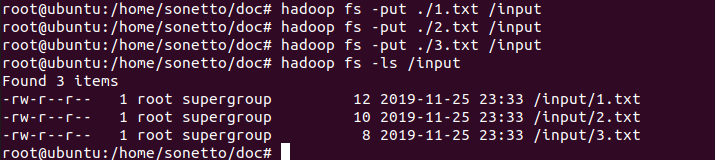
根据输入文件1、2和3得到的输出文件如下：

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

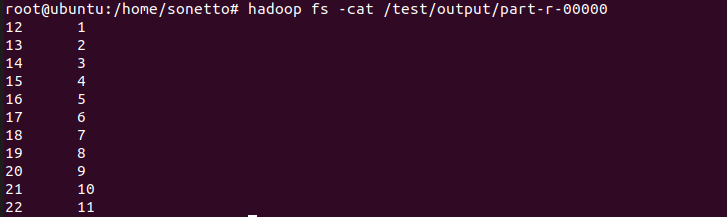
* 一些思考：

查了资料，shuffle阶段mapreduce程序会自动排序，为key相同的键值对合并（所以上一题需要combiner，这一题不用），所以只需要输出，并在key的位置上+1就好了。

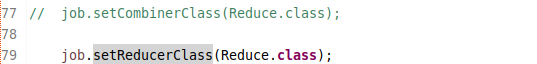
* 前期准备文件：



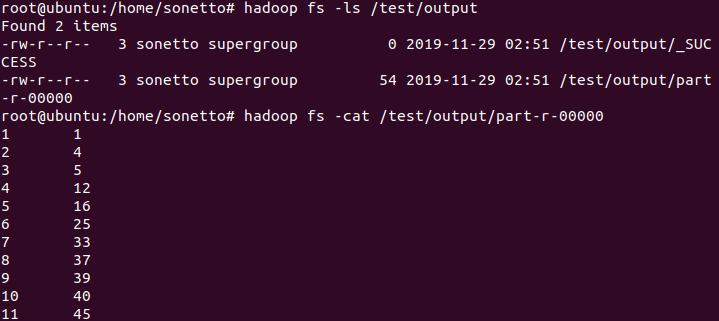
* 错误结果：



* 原因：设置了combinerclass，导致reduce进行了2次！把正确的1~11位排序再次传入进去，1~11变成了value！



* 正确结果：



* 代码：

**package** mapreduce;

**import** java.io.IOException;

**import** java.util.StringTokenizer;

**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;

**publicclass** sort {

//输出类型均为整型

**publicstaticclass** Map **extends** Mapper<Object, Text, IntWritable, IntWritable>

{

**protectedvoid** map(Object key,Text value,Context context)**throws** IOException,InterruptedException

{

**final** IntWritable one = **new** IntWritable(1);

//遇到换行的时候就切分

StringTokenizer token = **new** StringTokenizer(value.toString());

**while**(token.hasMoreElements())

{

//Integer.parseInt将数据转换为Integer整型数据

IntWritable text = **new** IntWritable(Integer.*parseInt*(token.nextToken()));

context.write(text, one);//将数字作为key,写入键值对

}

}

}

**publicstaticclass** Reduce **extends** Reducer<IntWritable,IntWritable,IntWritable,IntWritable>

{

//MapReduce在shuffle阶段默认排序、合并，所以只要计算位置+1就好了

@SuppressWarnings("unused")

**privatestatic** IntWritable *count* = **new** IntWritable(1);

//Iterable<IntWritable> values --> value-list

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

{

**for**(IntWritable i:values) {

context.write(*count*, key);//shuffle已经排序，只要按顺序输出，输出一个，序位+1

*count*.set(*count*.get()+1);

}

}

}

**publicstaticvoid** main(String[] args)**throws** Exception

{

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

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

Job job = Job.*getInstance*(conf);

String[] str = **new** String[] {"/input","/test/output"};

//如果输出路径已经有了！！就删掉重新创建！！省了好多麻烦

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

//上面这句确定是HDFS

**boolean**exists = fs.exists(**new** Path(str[1]));

**if**(exists) {

fs.delete(**new** Path(str[1]), **true**);

}

//设置程序的类名

job.setJarByClass(sort.**class**);

//继承的类 :前面自己定义的

job.setMapperClass(Map.**class**);

// job.setCombinerClass(Reduce.class);

job.setReducerClass(Reduce.**class**);

//设置输入输出类型

job.setOutputKeyClass(IntWritable.**class**);

job.setOutputValueClass(IntWritable.**class**);

//

// //设置输入、输出文件

FileInputFormat.*addInputPath*(job, **new** Path(str[0]));

FileOutputFormat.*setOutputPath*(job, **new** Path(str[1]));

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

}

}

**（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 |

* 一些思考:

根据之前的wordcount的例子和网上的资料，可以得知map阶段的shuffle会把同一个key的value组合在一起，再送到reduce。

所以，在map阶段通过在value的前面补上前缀，来识别父子、子父关系（父子：-，子父+），送入context中。

前提是需要一张正序表和一张逆序表，用来进行关联运算。

如：Jone Lucy

Lucy Mary

* Map后

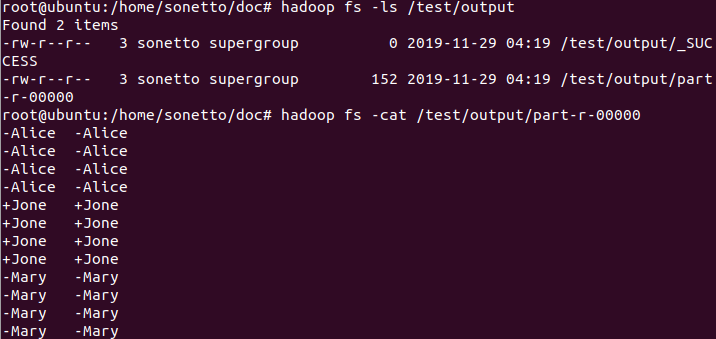
得到正序表键值对<Jone, -Lucy>.<Lucy,-Mary>

对应逆序表键值对<Lucy,+Jone>,<Mary,+Lucy>

* Shuffle后

<Lucy,<-Mary,+Jone>>,<Mary,+Lucy>,<Jone, -Lucy>

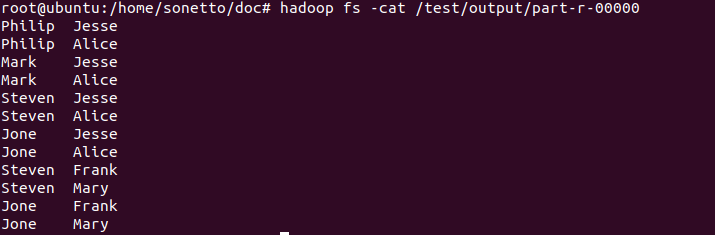
* 由此，可以得知Mary的grandparent是Jone!
* 错误结果：（我已经习惯出错了）



* 原因：注释掉的那句，是原来标记过的对象，没有去掉第一个符号，reduce就会当成不同的key。



* 正确结果：



* 代码：

package mapreduce;

import java.io.IOException;

import java.util.ArrayList;

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;

import org.apache.hadoop.util.GenericOptionsParser;

import mapreduce.sort.Map;

import mapreduce.sort.Reduce;

public class dig {

public static class DigMapper extends Mapper<Object, Text, Text, Text>

{ //key还是没有用到，此时的VALUE包括了child和parent两个值，通过split分开

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

{

//不用tokenizer是因为键值两个数据都需要用

String child = value.toString().split("")[0];

String parent = value.toString().split("")[1];

//在map阶段把正序、逆序都存入context

//注意是text类型键值对

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

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

}

}

public static class DigReducer extends Reducer<Text, Text, Text, Text>

{

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

{

//Text类型数组,因为key、VALUE根据实际情况来看，是可以不唯一的

ArrayList<Text> grandparent = new ArrayList<Text>();

ArrayList<Text> grandchild = new ArrayList<Text>();

//遍历程序当前key的所有value

for(Text t:value)

{

String s = t.toString();

if(s.startsWith("-"))

{

//不可以是grandparent.add(t);

grandparent.add(new Text(s.substring(1)));

//从第一个字符开始取，把标记去掉

}

else

{

grandchild.add(new Text(s.substring(1)));

}

}

//把当前key的两个数组数据写入grand表

for(int i=0;i<grandchild.size();i++)

for(int j=0;j<grandparent.size();j++)

{

context.write(grandchild.get(i), grandparent.get(j));

}

}

}

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

{

Configuration conf = new Configuration();

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

Job job = Job.getInstance(conf);

String[] str = new String[] {"/input","/test/output"};

FileSystem fs = FileSystem.get(conf);

boolean exists = fs.exists(new Path(str[1]));

if(exists) {

fs.delete(new Path(str[1]), true);

}

job.setJarByClass(dig.class);

job.setMapperClass(DigMapper.class);

job.setReducerClass(DigReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(str[0]));

FileOutputFormat.setOutputPath(job, new Path(str[1]));

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

}

}

1. **实验总结**
2. Mapreduce的工作都是在hdfs上进行的，所以要注意文件路径不是本机路径。
3. 、输入类型 、输出类型是根据业务类型自己定义的。

3.value为text指每次读文本的一行

Context类型是Map函数的一种输出方式，存储中间结果

4. MapReduce在shuffle阶段，默认帮我们将数据做了排序，并且是做了合并，相同的数据归为一组。

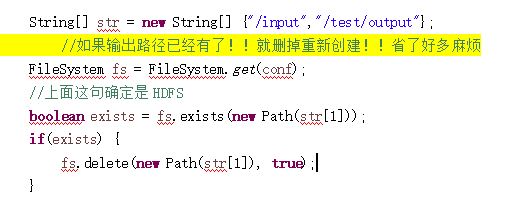
5.如果路径只写像“input”,”output”这样的，那么程序默认的是当前hadoop用户的目录下的input,output

（如：/user/sonetto/input）

1. 将路径设置成字符串数组类型可以用shell跑，但是eclipse不设置字符数组也可以。

（如：林子雨老师的args[0]、args[1]）

1. Integer.*parseInt()可以把其他类型的数据转换承interger类型*
2. 避免每次运行都要重新设置输出路径的方法之一：



9.Iterable<Text> value可以遍历value