****

**大数据技术原理及应用**

实验报告



学 院： 计算机科学与技术学院

专 业： 电力信息技术

报告名称： Hadoop MapReduce编程实践

学生姓名： 崔荣成 学号： 19108002

指导老师： 周 平

时 间： 2020年5月19日

目 录

[Hadoop MapReduce编程实践 2](#_Toc43970156)

[MapReduce原理 2](#_Toc43970157)

[1、分析MapReduce执行过程 2](#_Toc43970158)

[2、Mapper任务的执行过程详解 2](#_Toc43970159)

[3、Reducer任务的执行过程详解 3](#_Toc43970160)

[题1、WordCount实例 3](#_Toc43970161)

[1、整体流程 3](#_Toc43970162)

[2、实验代码 5](#_Toc43970163)

[3、流程说明 8](#_Toc43970164)

[题2、统计访问次数 9](#_Toc43970165)

[1、分析思路和逻辑 9](#_Toc43970166)

[2、实验代码 9](#_Toc43970167)

[3、流程说明 12](#_Toc43970168)

[题3、统计每年最高气温 14](#_Toc43970169)

[1、功能要求 14](#_Toc43970170)

[2、实验代码 15](#_Toc43970171)

[3、流程说明 18](#_Toc43970172)

[题4、数据去重 20](#_Toc43970173)

[1、实验原理 20](#_Toc43970174)

[2、实验数据 20](#_Toc43970175)

[3、实验代码 21](#_Toc43970176)

[4、流程说明 24](#_Toc43970177)

[题5、日志分析：分析非结构化文件 27](#_Toc43970178)

[1、需求 27](#_Toc43970179)

[2、测试数据集 27](#_Toc43970180)

[3、实验代码 28](#_Toc43970181)

[LogMapper 28](#_Toc43970182)

[LogReduce 29](#_Toc43970183)

[JobMain 30](#_Toc43970184)

[4、流程说明 31](#_Toc43970185)

[题6、MapReduce自定义排序 33](#_Toc43970186)

[1、功能要求 33](#_Toc43970187)

[2、解决思路 33](#_Toc43970188)

[3、实验代码 33](#_Toc43970189)

[自定义数据类StudentScore 33](#_Toc43970190)

[主类MyScoreMapReduce 35](#_Toc43970191)

[4、流程说明 37](#_Toc43970192)

[题7、MapReduce自定义分区 38](#_Toc43970193)

[1、功能要求 38](#_Toc43970194)

[2、解决思路 38](#_Toc43970195)

[3、实验代码 38](#_Toc43970196)

[自定义分区MyPartition 38](#_Toc43970197)

[自定义排序MySort 39](#_Toc43970198)

[自定义分组MyGrouping 41](#_Toc43970199)

[主类MyClassMapReduce 41](#_Toc43970200)

[4、流程说明 45](#_Toc43970201)

[题8、MapReduce自定义分组 47](#_Toc43970202)

[1、功能要求 47](#_Toc43970203)

[2、解决思路 47](#_Toc43970204)

[3、实验代码 48](#_Toc43970205)

[自定义排序MyClassAndScore 48](#_Toc43970206)

[自定义分区MyGroup2 49](#_Toc43970207)

[主类MyAllSource 50](#_Toc43970208)

[4、流程说明 52](#_Toc43970209)

Hadoop MapReduce编程实践

# MapReduce原理

## 1、分析MapReduce执行过程

MapReduce运行的时候，会通过Mapper运行的任务读取HDFS中的数据文件，然后调用自己的方法，处理数据，最后输出。Reducer任务会接收Mapper任务输出的数据，作为自己的输入数据，调用自己的方法，最后输出到HDFS的文件中。

## 2、Mapper任务的执行过程详解

每个Mapper任务是一个java进程，它会读取HDFS中的文件，解析成很多的键值对，经过我们覆盖的map方法处理后，转换为很多的键值对再输出。整个Mapper任务的处理过程又可以分为以下六个阶段。

第一阶段是把输入文件按照一定的标准分片(InputSplit)，每个输入片的大小是固定的。默认情况下，输入片(InputSplit)的大小与数据块(Block)的大小是相同的。如果数据块(Block)的大小是默认值64MB，输入文件有两个，一个是32MB，一个是72MB。那么小的文件是一个输入片，大文件会分为两个数据块，那么是两个输入片。一共产生三个输入片。每一个输入片由一个Mapper进程处理。这里的三个输入片，会有三个Mapper进程处理。

第二阶段是对输入片中的记录按照一定的规则解析成键值对。有个默认规则是把每一行文本内容解析成键值对。“键”是每一行的起始位置(单位是字节)，“值”是本行的文本内容。

第三阶段是调用Mapper类中的map方法。第二阶段中解析出来的每一个键值对，调用一次map方法。如果有1000个键值对，就会调用1000次map方法。每一次调用map方法会输出零个或者多个键值对。

第四阶段是按照一定的规则对第三阶段输出的键值对进行分区。比较是基于键进行的。比如我们的键表示省份(如北京、上海、山东等)，那么就可以按照不同省份进行分区，同一个省份的键值对划分到一个区中。默认是只有一个区。分区的数量就是Reducer任务运行的数量。默认只有一个Reducer任务。

第五阶段是对每个分区中的键值对进行排序。首先，按照键进行排序，对于键相同的键值对，按照值进行排序。比如三个键值对<2,2>、<1,3>、<2,1>，键和值分别是整数。那么排序后的结果是<1,3>、<2,1>、<2,2>。如果有第六阶段，那么进入第六阶段；如果没有，直接输出到本地的linux文件中。

第六阶段是对数据进行归约处理，也就是reduce处理。键相等的键值对会调用一次reduce方法。经过这一阶段，数据量会减少。归约后的数据输出到本地的linxu文件中。本阶段默认是没有的，需要用户自己增加这一阶段的代码。

## 3、Reducer任务的执行过程详解

每个Reducer任务是一个java进程。Reducer任务接收Mapper任务的输出，归约处理后写入到HDFS中，可以分为几个阶段。

第一阶段是Reducer任务会主动从Mapper任务复制其输出的键值对。Mapper任务可能会有很多，因此Reducer会复制多个Mapper的输出。

第二阶段是把复制到Reducer本地数据，全部进行合并，即把分散的数据合并成一个大的数据。再对合并后的数据排序。

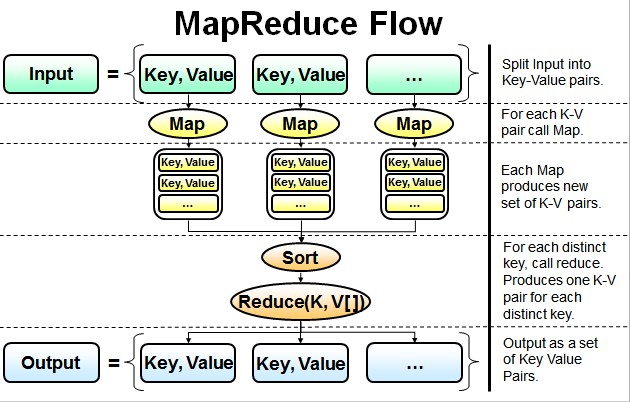
第三阶段是对排序后的键值对调用reduce方法。键相等的键值对调用一次reduce方法，每次调用会产生零个或者多个键值对。最后把这些输出的键值对写入到HDFS文件中。在整个MapReduce程序的开发过程中，我们最大的工作量是覆盖map函数和覆盖reduce函数。

# 题1、WordCount实例

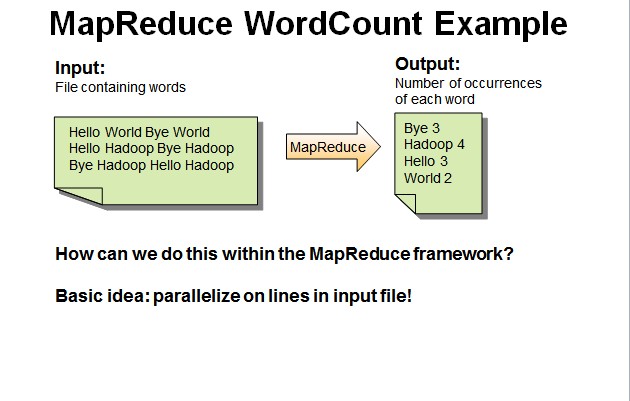
功能：统计每一个单词在整个数据集中出现的总次数

## 1、整体流程

　　最简单的MapReduce应用程序至少包含 3 个部分：一个 Map 函数、一个 Reduce 函数和一个 main 函数。在运行一个MapReduce计算任务时候，任务过程被分为两个阶段：map阶段和reduce阶段，每个阶段都是用键值对（key/value）作为输入（input）和输出（output）。main 函数将作业控制和文件输入/输出结合起来。



并行读取文本中的内容，然后进行MapReduce操作。



Map过程：并行读取文本，对读取的单词进行map操作，每个词都以<key,value>形式生成。

　　一个有三行文本的文件进行MapReduce操作。

　　读取第一行Hello World Bye World ，分割单词形成Map。

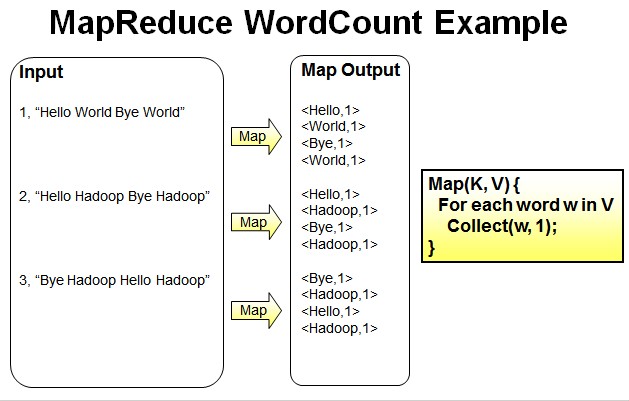
　　<Hello,1> <World,1> <Bye,1> <World,1>

　　读取第二行Hello Hadoop Bye Hadoop ，分割单词形成Map。

　　<Hello,1> <Hadoop,1> <Bye,1> <Hadoop,1>

　　读取第三行Bye Hadoop Hello Hadoop，分割单词形成Map。

　　<Bye,1> <Hadoop,1> <Hello,1> <Hadoop,1>



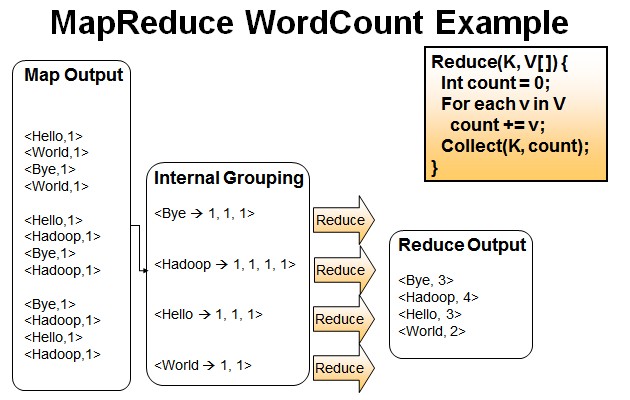
Reduce操作是对map的结果进行排序，合并，最后得出词频。

　　经过进一步处理(combiner),将形成的Map根据相同的key组合成value数组。

　　<Bye,1,1,1> <Hadoop,1,1,1,1> <Hello,1,1,1> <World,1,1>

　　循环执行Reduce(K,V[])，分别统计每个单词出现的次数。

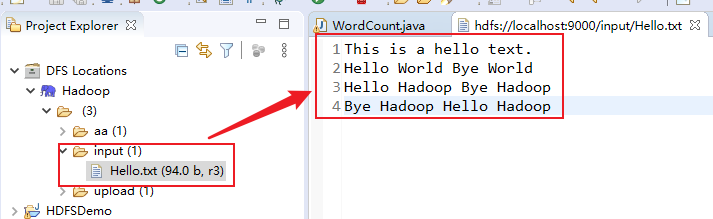
　　<Bye,3> <Hadoop,4> <Hello,3> <World,2>



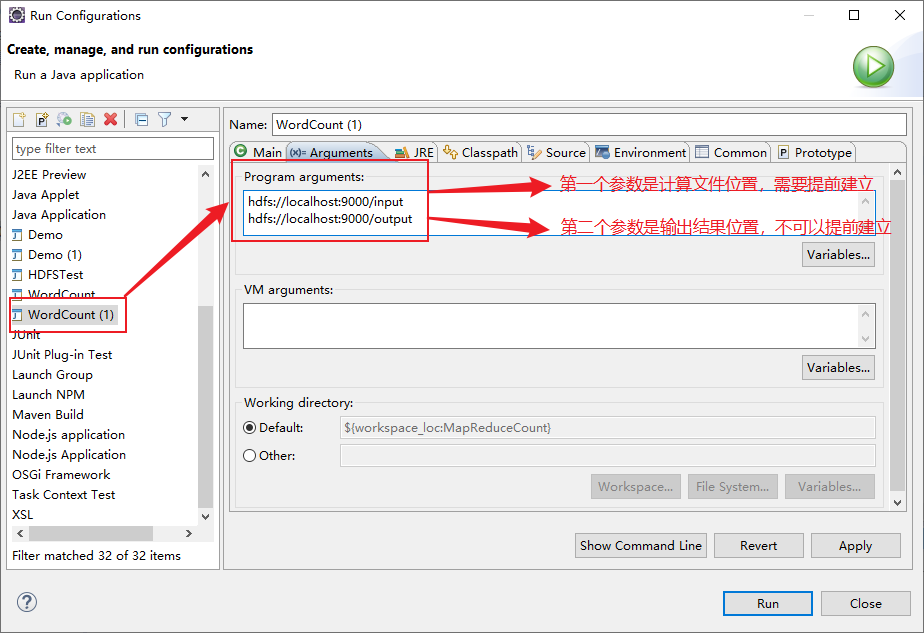
## 2、实验代码

|  |
| --- |
| **package** cn.crc;  **import** java.io.IOException;  **import** java.util.StringTokenizer;  **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** WordCount {  /\*\*  \* map函数的输入键、输入值、输出键和输出值  \* Text（相当于java的string类型） IntWritable（相当于java中的Integer）  \*/  **public** **static** **class** TokenizerMapper **extends** Mapper<Object, Text, Text, IntWritable> {  **private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);  **private** Text word = **new** Text();    **public** **void** map(Object key, Text value, Context context) //content用于输出内容的写入  //map（）方法的输入是一个键和一个值。我们首先将包含一行输入的Text值转换成java中的string类型  **throws** IOException, InterruptedException {  StringTokenizer itr = **new** StringTokenizer(value.toString());  //这是一个分割字符串的类,java中默认的分隔符是:"空格","\t"制表符,"\n"换行符,"\r"回车符  **while** (itr.hasMoreTokens()) { //判断是否还有分隔符  word.set(itr.nextToken()); //下一个字符串转换为Text类型  //String nextToken（）：返回从当前位置到下一个分隔符的字符串。  context.write(word, ***one***);  }  }  }    /\*\*  \* reduce函数也有四个形式参数类型用于指定输入和输出类型  \* reduce函数的输入类型必须匹配map函数的输出类型:即Text类型和Intwritable在这种情况下,reduce的输出也是Text和Intwritable  \*/  **public** **static** **class** IntSumReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {    **private** IntWritable result = **new** IntWritable();    **public** **void** reduce(Text key, Iterable<IntWritable> values, Context context)  **throws** IOException, InterruptedException {  **int** sum = 0; //定义累加器，初始值为0  **for** (IntWritable val : values) {  sum += val.get(); //将相同键的所有值进行累加  }  result.set(sum);  context.write(key, result);//输出单词出现次数  }  }    **public** **static** **void** main(String[] args) **throws** Exception {  Configuration conf = **new** Configuration(); //初始化相关Hadoop配置  String[] otherArgs = **new** GenericOptionsParser(conf, args).getRemainingArgs();  //输入是一个或多个文件，需要创建一个输入参数的子数组，包括数组的最后一个项目，即MR作业的输出目录  **if** (otherArgs.length != 2) {  System.***out***.println(otherArgs.length);  System.***err***.println("Usage: wordcount <in> <out>");  System.*exit*(2);  }  //Job对象指定作业执行规范.用它控制整个作业的运行.在集群上运行这个作业时,要把代码打包成一个JAR文件  //(Hadoop在集群上发布这个文件).不必明确指定JAR文件的名称.在Job对象的setJarByClass()方法中传递一个类即可  Job job = **new** ~~Job~~(conf, "word count"); //新建Job并设置主类  job.setJarByClass(WordCount.**class**);  job.setMapperClass(TokenizerMapper.**class**); //指定要用的map类型，必须  job.setCombinerClass(IntSumReducer.**class**);  job.setReducerClass(IntSumReducer.**class**); //指定要用的reduce类型，必须  job.setOutputKeyClass(Text.**class**); //控制reduce函数的输出键的格式类型  job.setOutputValueClass(IntWritable.**class**); //控制reduce函数的输出值的格式类型  FileInputFormat.*addInputPath*(job, **new** Path(otherArgs[0]));  //定义输入数据的路径,可以是单个文件,也可以是一个目录(此时,将目录下所有文件当做输入)  FileOutputFormat.*setOutputPath*(job, **new** Path(otherArgs[1]));  //定义输出路径,指定reduce函数输出文件的写入目录.在运行作业前该目录是不应该存在的,否则Hadoop会报错并拒绝运行作业.  System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);  //waitForCompletion()方法提交作业并等待执行完成.该方法唯一的参数是一个标识,指示是否已生成详细输出.当标识为true(成功)时,作业会把其进度信息写到控制台  }  } |

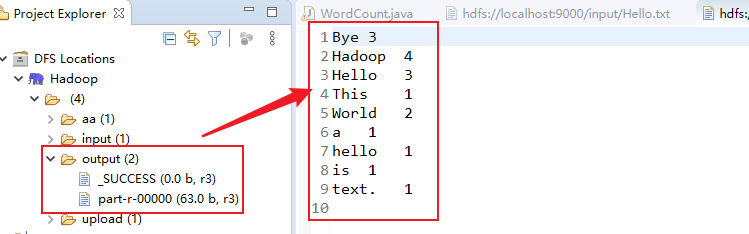
## 3、流程说明



添加待计算文件内容



配置计算文件与输出结果参数



运行结果截图

# 题2、统计访问次数

**任务描述：**

    统计用户在2016年度每个自然日的总访问次数。原始数据文件中提供了用户名称与访问日期。这个任务就是要获取以每个自然日为单位的所有用户访问次数的累加值。如果通过MapReduce编程实现这个任务，首先要考虑的是，Mapper与Reducer各自的处理逻辑是怎样的；然后根据处理逻辑编写出核心代码；最后在Eclipse中编写完整代码，编译打包后提交给集群运行。

## 1、分析思路和逻辑

（1） 输入/输出格式。

这里社交网站用户的访问日期在格式上都属于文本格式，访问次数为整型数据格式。其组成的键值对为<访问日期，访问次数>,因此Mapper的输出与Reducer的输出都选用Text类与IntWritble类。

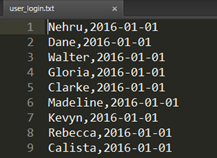
（2） Mapper要实现的计算逻辑

Map函数的主要任务是读取用户访问文件中的数据，输出所有访问日期与初始次数的键值对。<访问日期，1 >

（3）  Reducer要实现的计算逻辑

读取Mapper输出的键值对<访问日期，1>,进行累加。

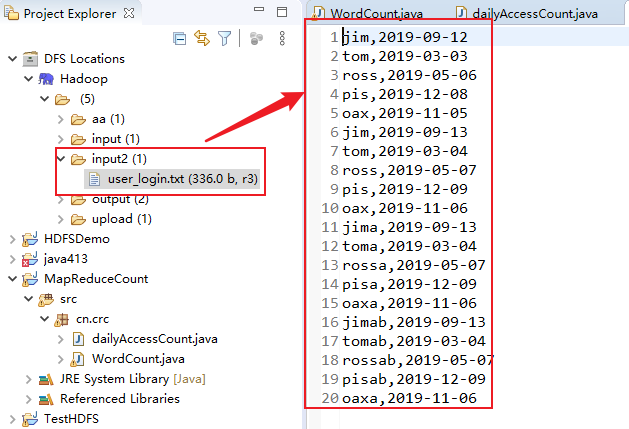
例：user\_login.txt访问日期格式如下：



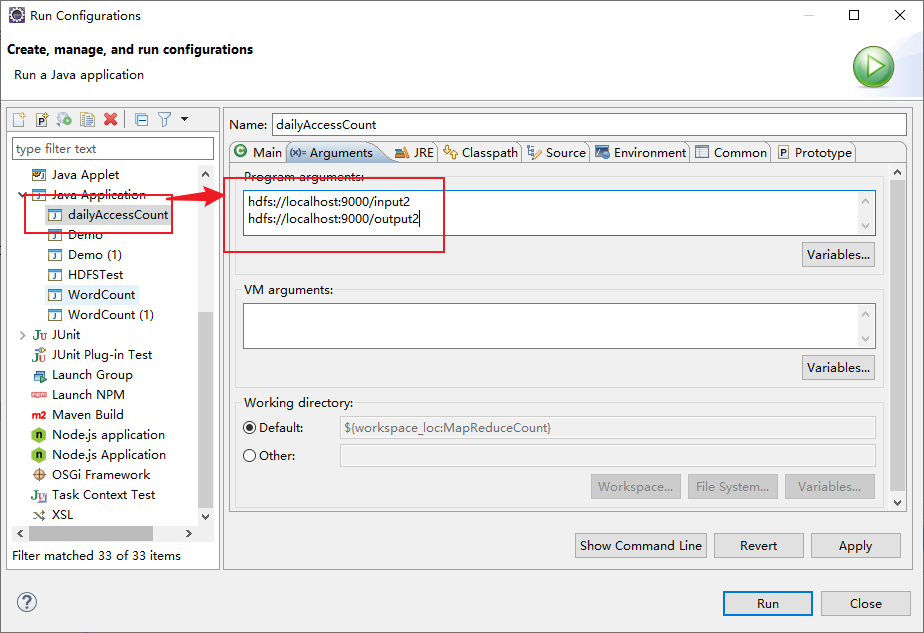
## 2、实验代码

|  |
| --- |
| **package** cn.crc;  **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** dailyAccessCount {    **public** **static** **class** MyMapper  **extends** Mapper<Object,Text,Text,IntWritable>{  **private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);    **public** **void** map(Object key,Text value,Context context)  **throws** IOException,InterruptedException{  String line = value.toString();  //指定逗号为分隔符，组成数组  String array[] = line.split(",");  //提取数组中的访问日期作为Key  String keyOutput = array[1];  //组成键值对  context.write(**new** Text(keyOutput),***one***);  }  }    **public** **static** **class** MyReducer  **extends** Reducer<Text,IntWritable,Text,IntWritable>{  **private** IntWritable result = **new** IntWritable();  **public** **void** reduce(Text key,Iterable<IntWritable> values,Context context)  **throws** IOException,InterruptedException {  //定义累加器，初始值为0  **int** sum = 0;  **for** (IntWritable val : values) {  //将相同键的所有值进行累加  sum += val.get();  }  result.set(sum);  context.write(key,result);//输出访问日期，总访问次数  }  }    **public** **static** **void** main(String[] args) **throws** Exception {  Configuration conf = **new** Configuration(); //初始化相关Hadoop配置  String[] otherArgs = **new** GenericOptionsParser(conf, args).getRemainingArgs();  //输入是一个或多个文件，需要创建一个输入参数的子数组，包括数组的最后一个项目，即MR作业的输出目录  **if** (otherArgs.length != 2) {  System.***out***.println(otherArgs.length);  System.***err***.println("Usage: wordcount <in> <out>");  System.*exit*(2);  }  //Job对象指定作业执行规范.用它控制整个作业的运行.在集群上运行这个作业时,要把代码打包成一个JAR文件  //(Hadoop在集群上发布这个文件).不必明确指定JAR文件的名称.在Job对象的setJarByClass()方法中传递一个类即可  Job job = Job.*getInstance*(conf,"Daily Access Count");//新建Job并设置主类  job.setJarByClass(dailyAccessCount.**class**);  job.setMapperClass(MyMapper.**class**); //为作业设置map类，必须  job.setReducerClass(MyReducer.**class**); //为作业设置reduce类，必须  job.setMapOutputKeyClass(Text.**class**); //设置map的输出键的格式类型  job.setMapOutputValueClass(IntWritable.**class**); //设置map的输出值的格式类型  job.setOutputKeyClass(Text.**class**); //设置reduce的输出键的格式类型  job.setOutputValueClass(IntWritable.**class**); //设置reduce的输出值的格式类型    **for** (**int** i = 0; i < args.length - 1; ++ i) {  FileInputFormat.*addInputPath*(job, **new** Path(args[i]));  }//设置作业HDFS的输入文件，Hadoop可设置多个输入文件，文件名使用逗号隔开    FileOutputFormat.*setOutputPath*(job, **new** Path(args[args.length - 1]));//设置作业HDFS的输出目录  System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);//通知集群运行这个作业，并阻塞到作业完成  }  } |

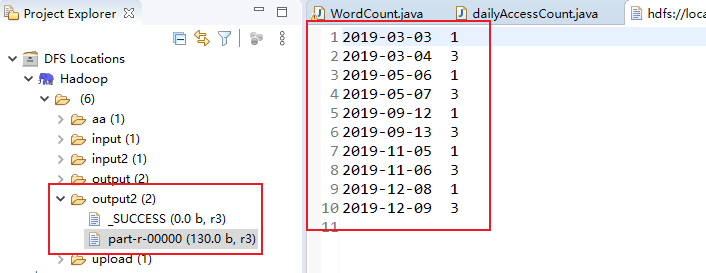
## 3、流程说明



添加待统计文件内容



配置输入文件位置和输出结果位置



统计结果截图

# 题3、统计每年最高气温

## 1、功能要求

在HDFS中的根目录下有以下文件格式： /input.txt

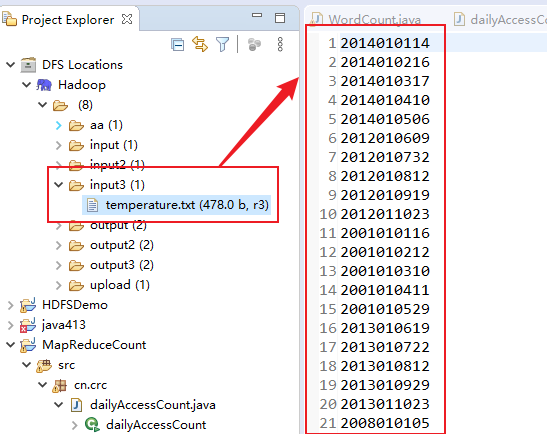
|  |
| --- |
| 2014010114  2014010216  2014010317  2014010410  2014010506  2012010609  2012010732  2012010812  2012010919  2012011023  2001010116  2001010212  2001010310  2001010411  2001010529  2013010619  2013010722  2013010812  2013010929  2013011023  2008010105  2008010216  2008010337  2008010414  2008010516  2007010619  2007010712  2007010812  2007010999  2007011023  2010010114  2010010216  2010010317  2010010410  2010010506  2015010649  2015010722  2015010812  2015010999  2015011023 |

比如：2010012325表示在2010年01月23日的气温为25度。现在要求使用MapReduce，计算每一年出现过的最大气温。

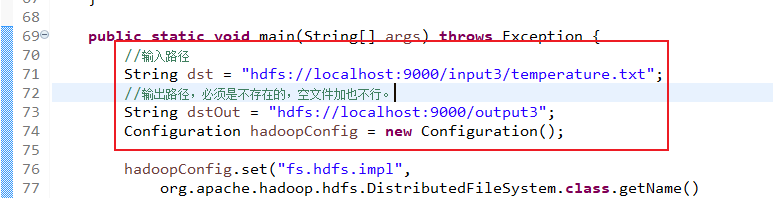
## 2、实验代码

|  |
| --- |
| **package** cn.crc;  **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.LongWritable;  **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** Temperature {  /\*\*  \* 四个泛型类型分别代表：  \* KeyIn Mapper的输入数据的Key，这里是每行文字的起始位置（0,11,...）  \* ValueIn Mapper的输入数据的Value，这里是每行文字  \* KeyOut Mapper的输出数据的Key，这里是每行文字中的“年份”  \* ValueOut Mapper的输出数据的Value，这里是每行文字中的“气温”  \*/  **static** **class** TempMapper **extends**  Mapper<LongWritable, Text, Text, IntWritable> {  @Override  **public** **void** map(LongWritable key, Text value, Context context)  **throws** IOException, InterruptedException {  // 打印样本: Before Mapper: 0, 2000010115  System.***out***.print("Before Mapper: " + key + ", " + value);  String line = value.toString();  String year = line.substring(0, 4);  **int** temperature = Integer.*parseInt*(line.substring(8));  context.write(**new** Text(year), **new** IntWritable(temperature));  // 打印样本: After Mapper:2000, 15  System.***out***.println(  "======" +  "After Mapper:" + **new** Text(year) + ", " + **new** IntWritable(temperature));  }  }  /\*\*  \* 四个泛型类型分别代表：  \* KeyIn Reducer的输入数据的Key，这里是每行文字中的“年份”  \* ValueIn Reducer的输入数据的Value，这里是每行文字中的“气温”  \* KeyOut Reducer的输出数据的Key，这里是不重复的“年份”  \* ValueOut Reducer的输出数据的Value，这里是这一年中的“最高气温”  \*/  **static** **class** TempReducer **extends**  Reducer<Text, IntWritable, Text, IntWritable> {  @Override  **public** **void** reduce(Text key, Iterable<IntWritable> values,  Context context) **throws** IOException, InterruptedException {  **int** maxValue = Integer.***MIN\_VALUE***;  StringBuffer sb = **new** StringBuffer();  //取values的最大值  **for** (IntWritable value : values) {  maxValue = Math.*max*(maxValue, value.get());  sb.append(value).append(", ");  }  // 打印样本： Before Reduce: 2000, 15, 23, 99, 12, 22,  System.***out***.print("Before Reduce: " + key + ", " + sb.toString());  context.write(key, **new** IntWritable(maxValue));  // 打印样本： After Reduce: 2000, 99  System.***out***.println(  "======" +  "After Reduce: " + key + ", " + maxValue);  }  }  **public** **static** **void** main(String[] args) **throws** Exception {  //输入路径  String dst = "hdfs://localhost:9000/input3/temperature.txt";  //输出路径，必须是不存在的，空文件加也不行。  String dstOut = "hdfs://localhost:9000/output3";  Configuration hadoopConfig = **new** Configuration();    hadoopConfig.set("fs.hdfs.impl",  org.apache.hadoop.hdfs.DistributedFileSystem.**class**.getName()  );  hadoopConfig.set("fs.file.impl",  org.apache.hadoop.fs.LocalFileSystem.**class**.getName()  );  Job job = **new** ~~Job~~(hadoopConfig);    //如果需要打成jar运行，需要下面这句  //job.setJarByClass(NewMaxTemperature.class);  //job执行作业时输入和输出文件的路径  FileInputFormat.*addInputPath*(job, **new** Path(dst));  FileOutputFormat.*setOutputPath*(job, **new** Path(dstOut));  //指定自定义的Mapper和Reducer作为两个阶段的任务处理类  job.setMapperClass(TempMapper.**class**);  job.setReducerClass(TempReducer.**class**);    //设置最后输出结果的Key和Value的类型  job.setOutputKeyClass(Text.**class**);  job.setOutputValueClass(IntWritable.**class**);    //执行job，直到完成  job.waitForCompletion(**true**);  System.***out***.println("Finished");  }  } |

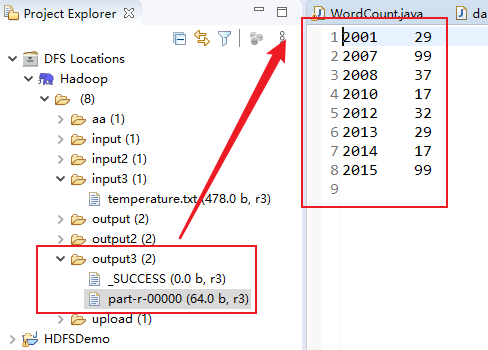
## 3、流程说明



添加待统计的温度文件内容



配置输入路径和输出路径



运行结果截图

# 题4、数据去重

## 1、实验原理

“数据去重”主要是为了掌握和利用并行化思想来对数据进行有意义的筛选。统计大数据集上的数据种类个数、从网站日志中计算访问地等这些看似庞杂的任务都会涉及数据去重。

数据去重的最终目标是让原始数据中出现次数超过一次的数据在输出文件中只出现一次。在MapReduce流程中，map的输出<key,value>经过shuffle过程聚集成<key,value-list>后交给reduce。我们自然而然会想到将同一个数据的所有记录都交给一台reduce机器，无论这个数据出现多少次，只要在最终结果中输出一次就可以了。具体就是reduce的输入应该以数据作为key，而对value-list则没有要求（可以设置为空）。当reduce接收到一个<key,value-list>时就直接将输入的key复制到输出的key中，并将value设置成空值，然后输出<key,value>。

## 2、实验数据

现有一个某电商网站的数据文件，名为buyer\_favorite1，记录了用户收藏的商品以及收藏的日期，文件buyer\_favorite1中包含（用户id，商品id，收藏日期）三个字段，数据内容以“\t”分割，由于数据很大，所以为了方便统计我们只截取它的一部分数据，内容如下：

|  |
| --- |
| 用户id   商品id    收藏日期  10181   1000481   2010-04-04 16:54:31  20001   1001597   2010-04-07 15:07:52  20001   1001560   2010-04-07 15:08:27  20042   1001368   2010-04-08 08:20:30  20067   1002061   2010-04-08 16:45:33  20056   1003289   2010-04-12 10:50:55  20056   1003290   2010-04-12 11:57:35  20056   1003292   2010-04-12 12:05:29  20054   1002420   2010-04-14 15:24:12  20055   1001679   2010-04-14 19:46:04  20054   1010675   2010-04-14 15:23:53  20054   1002429   2010-04-14 17:52:45  20076   1002427   2010-04-14 19:35:39  20054   1003326   2010-04-20 12:54:44  20056   1002420   2010-04-15 11:24:49  20064   1002422   2010-04-15 11:35:54  20056   1003066   2010-04-15 11:43:01  20056   1003055   2010-04-15 11:43:06  20056   1010183   2010-04-15 11:45:24  20056   1002422   2010-04-15 11:45:49  20056   1003100   2010-04-15 11:45:54  20056   1003094   2010-04-15 11:45:57  20056   1003064   2010-04-15 11:46:04  20056   1010178   2010-04-15 16:15:20  20076   1003101   2010-04-15 16:37:27  20076   1003103   2010-04-15 16:37:05  20076   1003100   2010-04-15 16:37:18  20076   1003066   2010-04-15 16:37:31  20054   1003103   2010-04-15 16:40:14  20054   1003100   2010-04-15 16:40:16 |

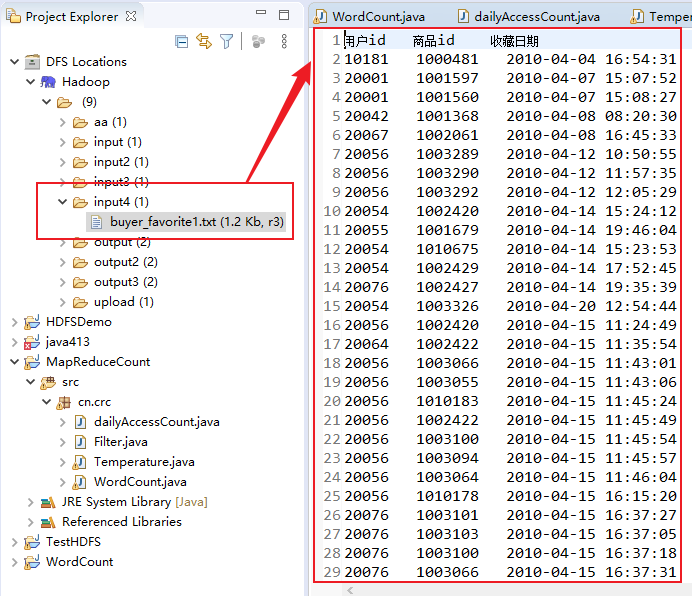
要求用Java编写MapReduce程序，根据商品id进行去重，统计用户收藏商品中都有哪些商品被收藏。结果数据如下：

|  |
| --- |
| 商品id  1000481  1001368  1001560  1001597  1001679  1002061  1002420  1002422  1002427  1002429  1003055  1003064  1003066  1003094  1003100  1003101  1003103  1003289  1003290  1003292  1003326  1010178  1010183  1010675 |

## 3、实验代码

|  |
| --- |
| **package** cn.crc;  **import** java.io.IOException;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.FileSystem;  **import** org.apache.hadoop.fs.Path;  **import** org.apache.hadoop.io.NullWritable;  **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.input.TextInputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  //import Utils.FindHDFSText;  **public** **class** Filter{  **public** **static** **class** Map **extends** Mapper<Object , Text , Text , NullWritable>{    **private** **static** Text *newKey*=**new** Text();  **public** **void** map(Object key,Text value,Context context)  **throws** IOException, InterruptedException{  String line=value.toString();  System.***out***.println("line是："+line);  **if**(line!=**null**){  String arr[]=line.split(" ");//指定\t为分隔符，组成数组  System.***out***.println("a[1]是"+arr[1]);  *newKey*.set(arr[1]); //提取数组中的商品id作为Key  context.write(*newKey*, NullWritable.*get*()); //组成键值对  System.***out***.println("新的截取的值是"+*newKey*); }  }  }    **public** **static** **class** Reduce **extends** Reducer<Text, NullWritable, Text, NullWritable>{  **public** **void** reduce(Text key,Iterable<NullWritable> values,Context context)  **throws** IOException, InterruptedException{  context.write(key,NullWritable.*get*());  }  }    **public** **static** **void** main(String[] args)  **throws** IOException, ClassNotFoundException, InterruptedException{  Configuration conf=**new** Configuration(); //初始化相关Hadoop配置  // FindHDFSText find = new FindHDFSText();  conf.set("dfs.client.use.datanode.hostname", "true");  System.***out***.println("start");    @SuppressWarnings("deprecation")  Job job =**new** ~~Job~~(conf,"filter"); //新建Job并设置主类  job.setJarByClass(Filter.**class**);  System.***out***.println("fiter执行完毕");  job.setMapperClass(Map.**class**); //指定要用的map类型  System.***out***.println("map执行完毕");  job.setReducerClass(Reduce.**class**);//指定要用的reduce类型  System.***out***.println("reduce执行完毕");    job.setOutputKeyClass(Text.**class**); //控制reduce函数的输出键的格式类型  job.setOutputValueClass(NullWritable.**class**); //控制reduce函数的输出值的格式类型    job.setInputFormatClass(TextInputFormat.**class**);  System.***out***.println("TextInputFormat执行完毕");  job.setOutputFormatClass(TextOutputFormat.**class**);  Path in=**new** Path("hdfs://localhost:9000/input4/buyer\_favorite1.txt");  System.***out***.println("in执行完毕");  Path out=**new** Path("hdfs://localhost:9000/output4");  System.***out***.println("out执行完毕");  Path path = **new** Path("hdfs://localhost:9000/output4");    FileSystem fileSystem = path.getFileSystem(conf);// 根据path找到这个文件  **if** (fileSystem.exists(path)) {  fileSystem.delete(path, **true**);// true的意思是，就算output有东西，也一带删除  }  FileInputFormat.*addInputPath*(job,in);  System.***out***.println("读入执行完毕");  FileOutputFormat.*setOutputPath*(job,out);  System.***out***.println("输出执行完毕");    System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);  }  } |

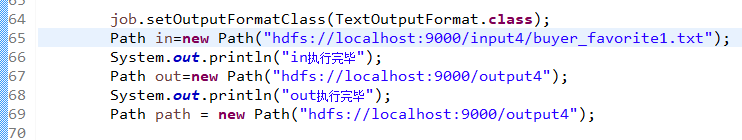
## 4、流程说明



上传数据文件到HDFS文件系统，存放路径:

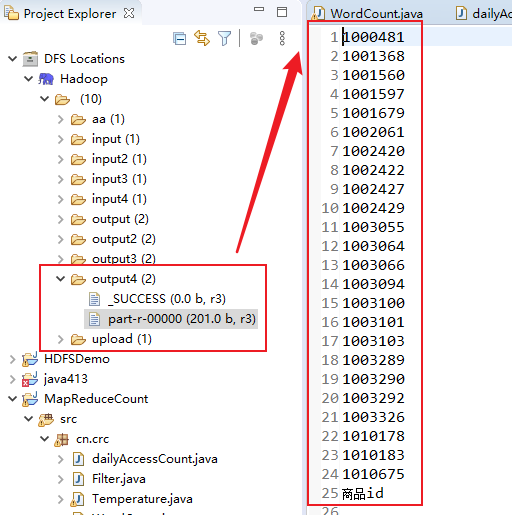
hdfs://localhost:9000/input4/buyer\_favorite1.txt

配置输入路径与输出路径



远程程序代码，运行过程输出截图如下





运行结果截图如上。

# 题5、日志分析：分析非结构化文件

## 1、需求

根据tomcat日志计算url访问了情况，具体的url如下，

要求：区别统计GET和POST URL访问量

结果为：访问方式、URL、访问量

## 2、测试数据集

|  |
| --- |
| 196.168.2.1 - - [03/Jul/2014:23:36:38 +0800] "GET /course/detail/3.htm HTTP/1.0" 200 38435 0.038  182.131.89.195 - - [03/Jul/2014:23:37:43 +0800] "GET /html/notes/20140617/888.html HTTP/1.0" 301 - 0.000  196.168.2.1 - - [03/Jul/2014:23:38:27 +0800] "POST /service/notes/addViewTimes\_23.htm HTTP/1.0" 200 2 0.003  196.168.2.1 - - [03/Jul/2014:23:39:03 +0800] "GET /html/notes/20140617/779.html HTTP/1.0" 200 69539 0.046  196.168.2.1 - - [03/Jul/2014:23:43:00 +0800] "GET /html/notes/20140318/24.html HTTP/1.0" 200 67171 0.049  196.168.2.1 - - [03/Jul/2014:23:43:59 +0800] "POST /service/notes/addViewTimes\_779.htm HTTP/1.0" 200 1 0.003  196.168.2.1 - - [03/Jul/2014:23:45:51 +0800] "GET /html/notes/20140617/888.html HTTP/1.0" 200 70044 0.060  196.168.2.1 - - [03/Jul/2014:23:46:17 +0800] "GET /course/list/73.htm HTTP/1.0" 200 12125 0.010  196.168.2.1 - - [03/Jul/2014:23:46:58 +0800] "GET /html/notes/20140609/542.html HTTP/1.0" 200 94971 0.077  196.168.2.1 - - [03/Jul/2014:23:48:31 +0800] "POST /service/notes/addViewTimes\_24.htm HTTP/1.0" 200 2 0.003  196.168.2.1 - - [03/Jul/2014:23:48:34 +0800] "POST /service/notes/addViewTimes\_542.htm HTTP/1.0" 200 2 0.003  196.168.2.1 - - [03/Jul/2014:23:49:31 +0800] "GET /notes/index-top-3.htm HTTP/1.0" 200 53494 0.041  196.168.2.1 - - [03/Jul/2014:23:50:55 +0800] "GET /html/notes/20140609/544.html HTTP/1.0" 200 183694 0.076  196.168.2.1 - - [03/Jul/2014:23:53:32 +0800] "POST /service/notes/addViewTimes\_544.htm HTTP/1.0" 200 2 0.004  196.168.2.1 - - [03/Jul/2014:23:54:53 +0800] "GET /service/notes/addViewTimes\_900.htm HTTP/1.0" 200 151770 0.054  196.168.2.1 - - [03/Jul/2014:23:57:42 +0800] "GET /html/notes/20140620/872.html HTTP/1.0" 200 52373 0.034  196.168.2.1 - - [03/Jul/2014:23:58:17 +0800] "POST /service/notes/addViewTimes\_900.htm HTTP/1.0" 200 2 0.003  196.168.2.1 - - [03/Jul/2014:23:58:51 +0800] "GET /html/notes/20140617/888.html HTTP/1.0" 200 70044 0.057  186.76.76.76 - - [03/Jul/2014:23:48:34 +0800] "POST /service/notes/addViewTimes\_542.htm HTTP/1.0" 200 2 0.003  186.76.76.76 - - [03/Jul/2014:23:46:17 +0800] "GET /course/list/73.htm HTTP/1.0" 200 12125 0.010  8.8.8.8 - - [03/Jul/2014:23:46:58 +0800] "GET /html/notes/20140609/542.html HTTP/1.0" 200 94971 0.077 |

由于Tomcat日志是不规则的，需要先过滤清洗数据。

## 3、实验代码

### LogMapper

|  |
| --- |
| **package** cn.crc;  **import** java.io.IOException;  **import** javax.naming.spi.DirStateFactory.Result;  **import** org.apache.hadoop.io.IntWritable;  **import** org.apache.hadoop.io.LongWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Mapper;    **public** **class** LogMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {    **private** IntWritable val = **new** IntWritable(1);  @Override  **protected** **void** map(LongWritable key, Text value,Context context)  **throws** IOException, InterruptedException {    String line = value.toString().trim();  String tmp = handlerLog(line);//调用清洗程序实现数据清洗  **if**(tmp.length()>0){  context.write(**new** Text(tmp), val);//提取清洗后的数据作为Key，组成键值对  }  }    //127.0.0.1 - - [03/Jul/2014:23:36:38 +0800] "GET /course/detail/3.htm HTTP/1.0" 200 38435 0.038  **private** String handlerLog(String line){  String result = ""; //存储清洗后的数据  **try**{  **if**(line.length()>20){  **if**(line.indexOf("GET")>0){ //如果是GET日志，获取GET-HTTP/1.0的数据段  result = line.substring(line.indexOf("GET"), line.indexOf("HTTP/1.0")).trim();  }**else** **if**(line.indexOf("POST")>0){ //如果是POST日志，获取POST-HTTP/1.0的数据段  result = line.substring(line.indexOf("POST"), line.indexOf("HTTP/1.0")).trim();  }  }  }**catch** (Exception e) {  System.***out***.println(line);  }    **return** result; //返回清洗后的数据  }    **public** **static** **void** main(String[] args){  String line = "127.0.0.1 - - [03/Jul/2014:23:36:38 +0800] \"GET /course/detail/3.htm HTTP/1.0\" 200 38435 0.038";  System.***out***.println(**new** LogMapper().handlerLog(line)); //先调用清洗程序实现数据清洗，在对清洗后的数据进行map  }  } |

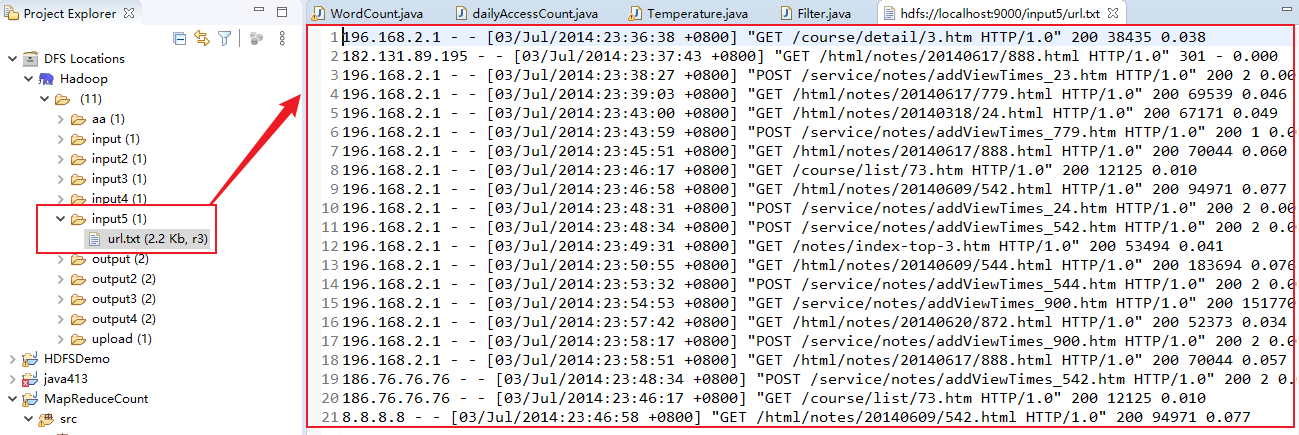
### LogReduce

|  |
| --- |
| **package** cn.crc;  **import** java.io.IOException;  **import** org.apache.hadoop.io.IntWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Reducer;    **public** **class** LogReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {    @Override  **protected** **void** reduce(Text key, Iterable<IntWritable> values,Context context)  **throws** IOException, InterruptedException {  **int** sum = 0;//定义累加器，初始值为0  **for**(IntWritable val : values){  sum += val.get();//将相同键的所有值进行累加  }  context.write(key, **new** IntWritable(sum));//输出日志数据出现次数    }    } |

### JobMain

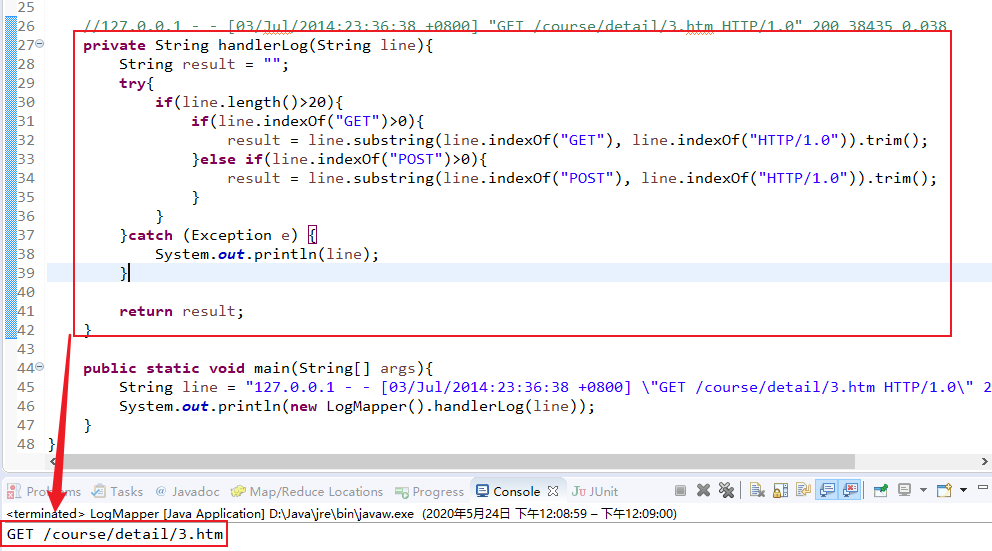
|  |
| --- |
| **package** cn.crc;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.FileSystem;  **import** org.apache.hadoop.fs.Path;  **import** org.apache.hadoop.io.IntWritable;  **import** org.apache.hadoop.io.Text;  **import** org.apache.hadoop.mapreduce.Job;  **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;    **public** **class** JobMain {    /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args)**throws** Exception {    Configuration configuration = **new** Configuration();//初始化相关Hadoop配置    //新建Job并设置主类  Job job = **new** ~~Job~~(configuration,"log\_job");  job.setJarByClass(JobMain.**class**);    //指定自定义的Reduce，设置reduce的输出键和输出值的格式类型  job.setMapperClass(LogMapper.**class**);  job.setMapOutputKeyClass(Text.**class**);  job.setMapOutputValueClass(IntWritable.**class**);    //指定自定义的Mapper，设置map的输出键和输出值的格式类型  job.setReducerClass(LogReducer.**class**);  job.setOutputKeyClass(Text.**class**);  job.setOutputValueClass(IntWritable.**class**);    //配置输入路径与输出路径  FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  Path path = **new** Path(args[1]);  FileSystem fs = path.getFileSystem(configuration);  **if**(fs.exists(path)){  fs.delete(path, **true**);  }  FileOutputFormat.*setOutputPath*(job, path);  System.***out***.println("输出执行完毕");    //通知集群运行这个作业，并阻塞到作业完成  System.*exit*(job.waitForCompletion(**true**)?0:1);    }    } |

## 4、流程说明

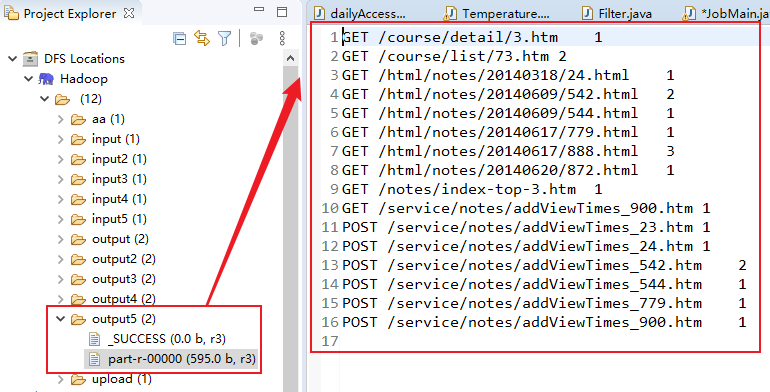


上传数据文件到HDFS文件系统，存放路径:

hdfs://localhost:9000/input5/url.txt



运行LogMapper程序测试数据清洗



运行JobMain程序调用LogMapper和LogReduce程序进行数据清洗与日志分析，最后运行结果截图如上。

# 题6、MapReduce自定义排序

字段说明

班级 学号 姓名 语文 数学 英语

1307 7026 邝卓男 95 88 98

## 1、功能要求

求每个学生的总分和平均分，并按总分降序排序

## 2、解决思路

当看到“每个”时，就把后面的字段当成分组字段，需要对总分进行排序，所以需要自定义排序。

在开始代码的编写前，首先要确定map、reduce中的key和value各是什么？

Map把每行字符按照”\t”分割，分别记录语文，数学，英语的成绩，计算总分和平均分。然后定义新的数据类型记录学生姓名、总分和平均分。把新定义的数据类型做map的key，学生的班级和学号信息组合成map的value。

Reduce任务接收Map任务的输出，归约处理后写入到HDFS中。

## 3、实验代码

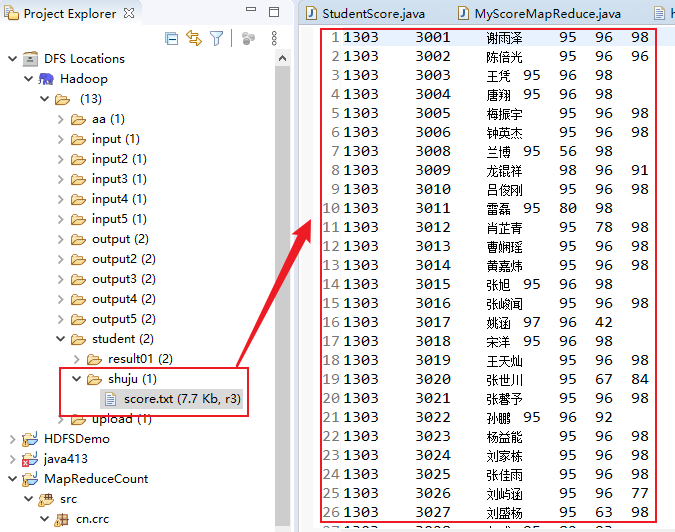
### 自定义数据类StudentScore

|  |
| --- |
| **package** cn.crc;  **import** java.io.DataInput;  **import** java.io.DataOutput;  **import** java.io.IOException;  **import** org.apache.hadoop.io.WritableComparable;  **public** **class** StudentScore **implements** WritableComparable<StudentScore> {  **private** String name;  **private** **int** sum;  **private** **double** avg;  **public** StudentScore(){  **super**();  }  **public** StudentScore(String name,**int** sum,Double avg){  **this**.name=name;  **this**.sum=sum;  **this**.avg=avg;  }  **public** **int** getSum() {  **return** sum;  }  **public** **double** getAvg() {  **return** avg;  }  **public** **void** setSum(**int** sum) {  **this**.sum = sum;  }  **public** **void** setAvg(**double** avg) {  **this**.avg = avg;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  @Override  **public** String toString() {  **return** **this**.name+"\t"+**this**.sum+"\t"+**this**.avg;  }    @Override  **public** **void** write(DataOutput out) **throws** IOException {  out.writeUTF(name);  out.writeInt(sum);  out.writeDouble(avg);  }    @Override  **public** **void** readFields(DataInput in) **throws** IOException {  **this**.name=in.readUTF();  **this**.sum=in.readInt();  **this**.avg=in.readDouble();    }  @Override  **public** **int** compareTo(StudentScore o) {  **return** o.getSum()-**this**.getSum(); //根据总分进行倒序排序  }  } |

### 主类MyScoreMapReduce

|  |
| --- |
| **package** cn.crc;  **import** java.io.IOException;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.Path;  **import** org.apache.hadoop.io.LongWritable;  **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** MyScoreMapReduce {  **public** **static** **class** MyMapper **extends** Mapper<LongWritable,Text,StudentScore,Text> {  Text ovalue=**new** Text();  @Override  **protected** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  //读取一行数据进行切分  String[] fields = value.toString().split("\t");//按照"\t"分割字符串  **int** chinese=Integer.*parseInt*(fields[3]);//取出语文成绩  **int** math=Integer.*parseInt*(fields[4]);//取出数学成绩  **int** english=Integer.*parseInt*(fields[5]);//取出英语成绩  **int** sum=chinese+math+english;//计算总分  Double avg=(1.0)\*sum/3;//计算平均分  StudentScore ss=**new** StudentScore(fields[2],sum,avg); //创建自定义类对象，包含姓名，总分，平均分三个值  ovalue.set(fields[0]+"\t"+fields[1]);//把班级和学号设置为map的值  context.write(ss,ovalue);//组成键值对  }  }  **public** **static** **class** MyReduce **extends** Reducer<StudentScore,Text,Text,StudentScore>{  @Override  **protected** **void** reduce(StudentScore key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  **for**(Text v: values){  context.write(v,key);  }  }  }  **public** **static** **void** main(String[] args) {  Configuration conf=**new** Configuration();  System.*setProperty*("HADOOP\_HOME\_USER","qyl");  conf .set( "fs.defaultFS" , "hdfs://localhost:9000" );  **try** {  Job job=Job.*getInstance*(conf);//新建Job  job.setJarByClass(MyScoreMapReduce.**class**);//设置主类  job.setMapperClass(MyMapper.**class**);//设置Mapper类  job.setReducerClass(MyReduce.**class**);//设置Reduce类  job.setMapOutputKeyClass(StudentScore.**class**);//设置map的输出键的格式类型  job.setMapOutputValueClass(Text.**class**);//设置map的输出值的格式类型  job.setOutputKeyClass(Text.**class**);//设置reduce的输出键的格式类型  job.setOutputValueClass(StudentScore.**class**);//设置reduce的输出值的格式类型  Path inpath=**new** Path("/student/shuju");//输入数据路径  FileInputFormat.*addInputPath*(job,inpath);  Path outpath=**new** Path("/student/result01");//输出结果路径  **if**(outpath.getFileSystem(conf).exists(outpath)){  outpath.getFileSystem(conf).delete(outpath,**true**);  }  FileOutputFormat.*setOutputPath*(job,outpath);  job.waitForCompletion(**true**);  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

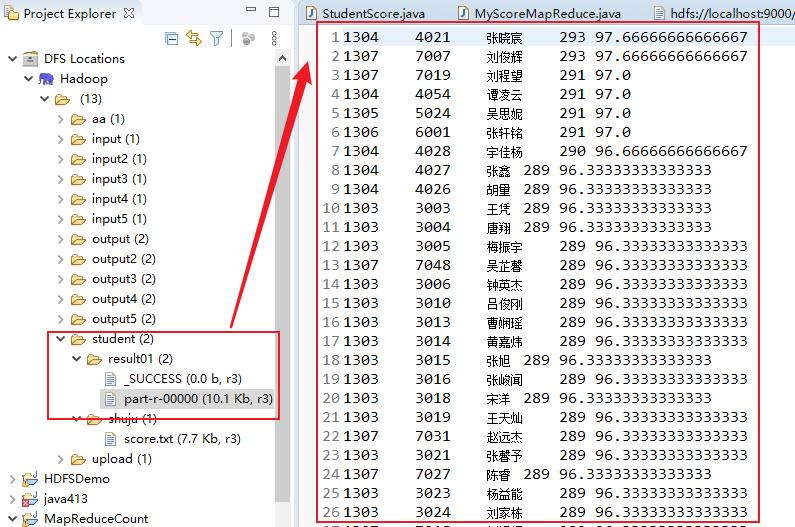
## 4、流程说明



添加数据文件到HDFS文件系统，文件路径

hdfs://localhost:9000/student/shuju

运行MyScoreMapReduce程序



得到最后排序结果。

# 题7、MapReduce自定义分区

## 1、功能要求

求每个班级每一门课程的平均分，不同班级的结果输出到不同的结果文件

## 2、解决思路

分区字段为班级

排序字段为班级和课程

分组字段为班级和课程

## 3、实验代码

### 自定义分区MyPartition

|  |
| --- |
| **package** cn.crc;  **import** org.apache.hadoop.io.IntWritable;  **import** org.apache.hadoop.mapreduce.Partitioner;  **public** **class** MyPartition **extends** Partitioner<MySort, IntWritable> {  @Override  //按照班级划分区域  **public** **int** getPartition(MySort key, IntWritable arg1, **int** arg2) {  **if**(key.getClassname().equals("1303")){  **return** 0;  }  **if**(key.getClassname().equals("1304")){  **return** 1;  }  **if**(key.getClassname().equals("1305")){  **return** 2;  }  **if**(key.getClassname().equals("1306")){  **return** 3;  }**else**{  **return** 4;  }  }  } |

### 自定义排序MySort

|  |
| --- |
| **package** cn.crc;  **import** java.io.DataInput;  **import** java.io.DataOutput;  **import** java.io.IOException;  **import** org.apache.hadoop.io.WritableComparable;  **public** **class** MySort **implements** WritableComparable<MySort> {  **private** String classname;  **private** String course;  **public** MySort() {  **super**();  }  **public** MySort(String classname, String course) {  **this**.classname = classname;  **this**.course = course;  }  **public** String getClassname() {  **return** classname;  }  **public** String getCourse() {  **return** course;  }  @Override  **public** String toString() {  **return** classname+"\t"+course;  }  **public** **void** setClassname(String classname) {  **this**.classname = classname;  }  **public** **void** setCourse(String course) {  **this**.course = course;  }  @Override  **public** **void** write(DataOutput out) **throws** IOException {  out.writeUTF(classname);  out.writeUTF(course);  }  @Override  **public** **void** readFields(DataInput in) **throws** IOException {  **this**.classname=in.readUTF();  **this**.course=in.readUTF();  }  @Override  **public** **int** compareTo(MySort o) {//班级相同的情况下，比较课程，按照课程排序  **int** temp=**this**.getClassname().compareTo(o.getClassname());  **if**(temp==0){  temp=**this**.getCourse().compareTo(o.getCourse());  }  **return** temp;  }  } |

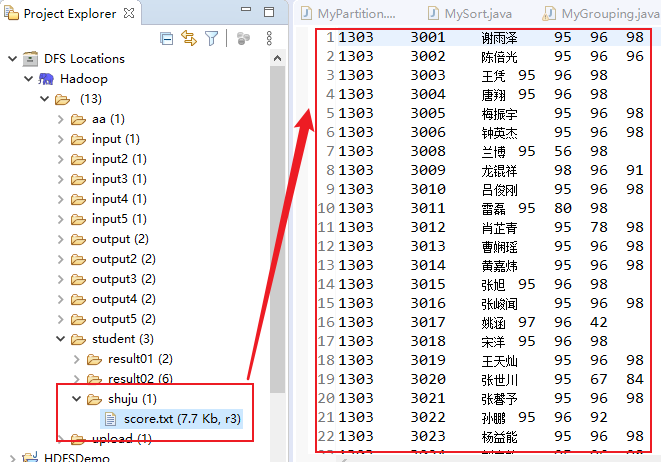
### 自定义分组MyGrouping

|  |
| --- |
| **package** cn.crc;  **import** org.apache.hadoop.io.WritableComparable;  **import** org.apache.hadoop.io.WritableComparator;  **public** **class** MyGrouping **extends** WritableComparator {  **public** MyGrouping(){  **super**(MySort.**class**,**true**);  }  @Override  **public** **int** compare(WritableComparable a, WritableComparable b) {  MySort aa=(MySort)a;  MySort bb=(MySort)b;//班级相同的情况下，比较课程，按照课程排序  **int** i= aa.getClassname().compareTo(bb.getClassname());  **if**(i==0){  **return** aa.getCourse().compareTo(bb.getCourse());  }  **return** i;  }  } |

### 主类MyClassMapReduce

|  |
| --- |
| **package** cn.crc;  **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.LongWritable;  **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** MyClassMapReduce {  **public** **static** **class** MyMapper **extends** Mapper<LongWritable,Text,MySort,IntWritable>{  MySort ms=**new** MySort();//定义自定义排序对象  @Override  **protected** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  String[] fields = value.toString().split("\t");//按照"\t"分割字符串  **int** score=0;//定义分数器  ms.setClassname(fields[0]);//设置自定义对象的班级字段  **for**(**int** i=3;i<fields.length;i++){  **if**(i==3){  ms.setCourse("语文");//设置自定义对象的语文字段  score=Integer.*parseInt*(fields[3]);//设置语文成绩  context.write(ms,**new** IntWritable(score));//设置键值对象-语文成绩对  }  **if**(i==4){  ms.setCourse("数学");//设置自定义对象的数学字段  score=Integer.*parseInt*(fields[4]);//设置数学成绩  context.write(ms,**new** IntWritable(score));//设置键值对象-数学成绩对  }**else**{  ms.setCourse("英语");//设置自定义对象的英语字段  score=Integer.*parseInt*(fields[5]);//设置英语成绩  context.write(ms,**new** IntWritable(score));//设置键值对象-英语成绩对  }  }  }  }  **public** **static** **class** MyReducer **extends** Reducer<MySort,IntWritable,MySort,Text>{  @Override  **protected** **void** reduce(MySort key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  **int** sum=0;**int** count=0;//定义累加器和计数器  **for**(IntWritable v: values){  sum+=v.get();//累加器累加成绩  count++;//计数器计数  }  context.write(key,**new** Text(""+1.0\*sum/count));//设置键值对对象-平均分  System.***out***.println(key.toString()+"---------"+1.0\*sum/count);    }  }  **public** **static** **void** main(String[] args) {  Configuration conf =**new** Configuration();  System.*setProperty*("HADOOP\_USER\_NAME", "crc");  conf .set( "fs.defaultFS" , "hdfs://localhost:9000" );  System.***out***.println("连接成功");  **try** {  Job job=Job.*getInstance*(conf);//新建Job  job.setJarByClass(MyClassMapReduce.**class**);//设置Job主类  job.setMapperClass(MyMapper.**class**);//设置map类  job.setReducerClass(MyReducer.**class**);//设置reduce类    job.setMapOutputKeyClass(MySort.**class**);//设置map输出键类型  job.setMapOutputValueClass(IntWritable.**class**);//设置map输出值类型    job.setOutputKeyClass(MySort.**class**);//设置reduce输出键类型  job.setOutputValueClass(Text.**class**);//设置reduce输出值类型    job.setPartitionerClass(MyPartition.**class**);//设置分区类  job.setGroupingComparatorClass(MyGrouping.**class**);//设置分组类  job.setNumReduceTasks(5);//设置reduce任务数    //指定需要统计的文件输入路径  Path inpath=**new** Path("/student/shuju");  FileInputFormat.*addInputPath*(job, inpath);  System.***out***.println("输入成功");    //指定输出目录 输出路径不能存在，否则就会报错 默认是覆盖式的输出  Path outpath=**new** Path("/student/result02");  **if**(outpath.getFileSystem(conf).exists(outpath)){  outpath.getFileSystem(conf).delete(outpath,**true**);  }  FileOutputFormat.*setOutputPath*(job, outpath);  System.***out***.println("输出成功");    job.waitForCompletion(**true**);  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

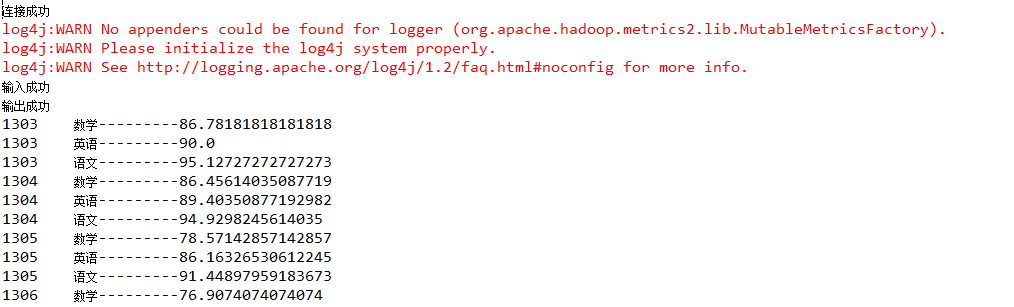
## 4、流程说明



输入数据和题6的数据一样，文件路径

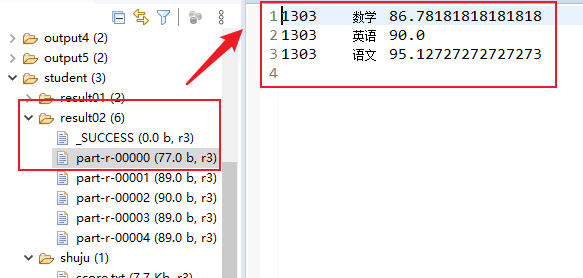
hdfs://localhost:9000/student/shuju

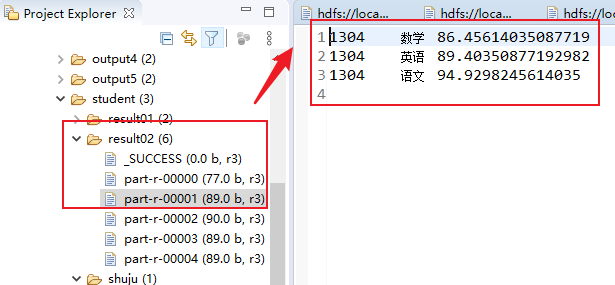
运行MyClassMapReduce程序

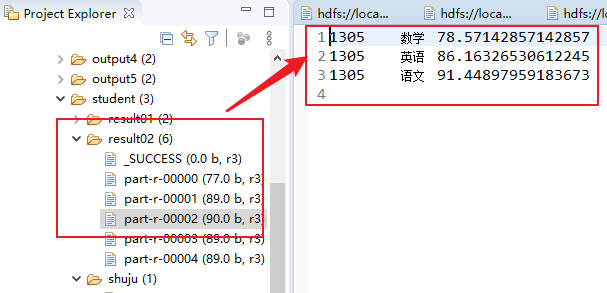


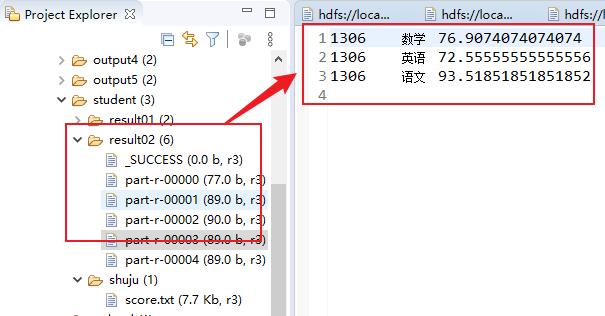
控制台显示运行结果

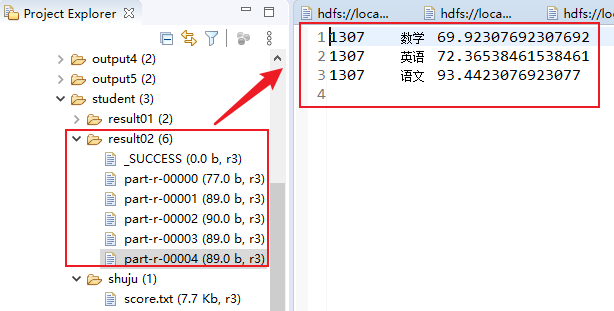
运行结果截图











# 题8、MapReduce自定义分组

## 1、功能要求

求每个班级的总分最高的前5个学生

## 2、解决思路

自定义排序 字段为 班级和总成绩

自定义分组 字段为 班级

## 3、实验代码

### 自定义排序MyClassAndScore

|  |
| --- |
| **package** cn.crc;  **import** java.io.DataInput;  **import** java.io.DataOutput;  **import** java.io.IOException;  **import** org.apache.hadoop.io.WritableComparable;  **public** **class** MyClassAndScore **implements** WritableComparable<MyClassAndScore> {  **private** String classname;  **private** **int** sum;  **public** String getClassname() {  **return** classname;  }  **public** **int** getSum() {  **return** sum;  }  **public** **void** setClassname(String classname) {  **this**.classname = classname;  }  **public** **void** setSum(**int** sum) {  **this**.sum = sum;  }  **public** MyClassAndScore() {  **super**();  }  @Override  **public** String toString() {  **return** classname+"\t"+sum;  }    **public** MyClassAndScore(String classname, **int** sum) {  **this**.classname = classname;  **this**.sum = sum;  }  @Override  **public** **void** write(DataOutput out) **throws** IOException {  out.writeUTF(classname);  out.writeInt(sum);  }    @Override  **public** **void** readFields(DataInput in) **throws** IOException {  **this**.classname=in.readUTF();  **this**.sum=in.readInt();  }  @Override  **public** **int** compareTo(MyClassAndScore o) {//班级相同的情况下，比较总分，按照总分排序  **int** temp=**this**.getClassname().compareTo(o.getClassname());  **if**(temp==0){  temp=o.getSum()-**this**.getSum();  }  **return** temp;  }  } |

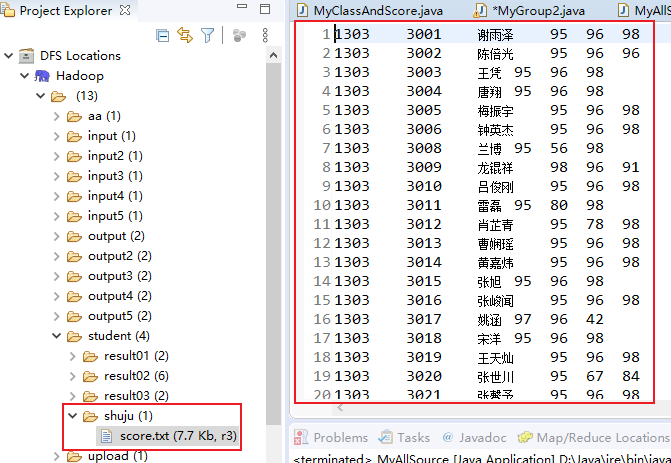
### 自定义分区MyGroup2

|  |
| --- |
| **package** cn.crc;  **import** org.apache.hadoop.io.WritableComparable;  **import** org.apache.hadoop.io.WritableComparator;  **public** **class** MyGroup2 **extends** WritableComparator {  **public** MyGroup2(){  **super**(MyClassAndScore.**class**,**true**);  }    @Override//按照班级进行分组  **public** **int** compare(WritableComparable a, WritableComparable b) {  MyClassAndScore aa=(MyClassAndScore)a;  MyClassAndScore bb=(MyClassAndScore)b;  **return** aa.getClassname().compareTo(bb.getClassname());  }  } |

### 主类MyAllSource

|  |
| --- |
| **package** cn.crc;  **import** java.io.IOException;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.Path;  **import** org.apache.hadoop.io.LongWritable;  **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** MyAllSource {  **public** **static** **class** MyMapper **extends** Mapper<LongWritable,Text,MyClassAndScore,Text>{    @Override  **protected** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {  String[] fields = value.toString().split("\t");//按照"\t"分割字符串  **int** chinese=Integer.*parseInt*(fields[3]);//取出语文成绩  **int** math=Integer.*parseInt*(fields[4]);//取出数学成绩  **int** english=Integer.*parseInt*(fields[5]);//取出英语成绩  **int** sum=chinese+math+english;//计算总分  MyClassAndScore ms=**new** MyClassAndScore(fields[0],sum);//新数据类型保存班级和总分字段  context.write(ms,**new** Text(fields[1]+"\t"+fields[2]));//新对象-（学号+姓名）组成键值对  }  }  **public** **static** **class** MyReducer **extends** Reducer<MyClassAndScore,Text,Text,Text>{    @Override  **protected** **void** reduce(MyClassAndScore key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  **int** count=0;//定义计数器  **for**(Text v:values){  count++;  **if**(count<=5) {//计算排名前五的map键值对  context.write(**new** Text(key.getClassname()), **new** Text(v.toString() + "\t" + key.getSum()));//班级-（学号+姓名+总分）组成键值对  }  }  }  }  **public** **static** **void** main(String[] args) {  Configuration conf =**new** Configuration();//初始化hadoop  System.*setProperty*("HADOOP\_USER\_NAME", "crc");//hadoop 名称  conf .set( "fs.defaultFS" , "hdfs://localhost:9000" );//hadoop接口地址  **try** {  Job job=Job.*getInstance*(conf);//新建job类  job.setJarByClass(MyAllSource .**class**);//设置主类  job.setMapperClass(MyMapper.**class**);//设置map类  job.setReducerClass(MyReducer.**class**);//设置reduce类    job.setMapOutputKeyClass(MyClassAndScore.**class**);//设置map输出键格式  job.setMapOutputValueClass(Text.**class**);//设置map输出值格式    job.setOutputKeyClass(Text.**class**);//设置reduce输出键格式  job.setOutputValueClass(Text.**class**);//设置reduce输出值格式    job.setGroupingComparatorClass(MyGroup2.**class**);//设置分组类    //指定需要统计的文件输入路径  Path inpath=**new** Path("/student/shuju");  FileInputFormat.*addInputPath*(job, inpath);    //指定输出目录 输出路径不能存在，否则就会报错 默认是覆盖式的输出  Path outpath=**new** Path("/student/result03");  **if**(outpath.getFileSystem(conf).exists(outpath)){  outpath.getFileSystem(conf).delete(outpath,**true**);  }  FileOutputFormat.*setOutputPath*(job, outpath);    job.waitForCompletion(**true**);  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

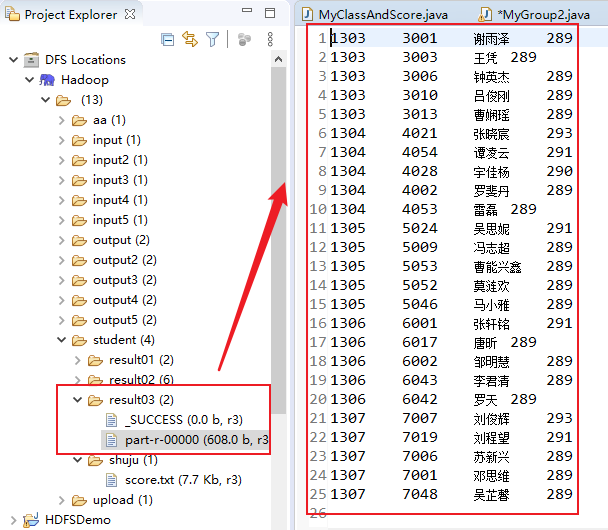
## 4、流程说明



输入数据和题6的数据一样，文件路径

hdfs://localhost:9000/student/shuju

运行MyAllSource程序



运行结果如上。