|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 《大数据技术原理与应用》实验报告 | | | | |
| 题目： | 上机4-MapReduce初级编程实践 | 姓名： |  | 日期： |
| 实验环境：  操作系统：  PRETTY\_NAME="Ubuntu 22.04.1 LTS"  NAME="Ubuntu"  VERSION\_ID="22.04"  VERSION="22.04.1 LTS (Jammy Jellyfish)"  VERSION\_CODENAME=jammy  ID=ubuntu  ID\_LIKE=debian  HOME\_URL="https://www.ubuntu.com/"  SUPPORT\_URL="https://help.ubuntu.com/"  BUG\_REPORT\_URL="https://bugs.launchpad.net/ubuntu/"  PRIVACY\_POLICY\_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"  UBUNTU\_CODENAME=jammy  Hadoop 版本：3.3.6。  JDK 版本：1.8。  Java IDE：IntelliJ Idea Umltimate 远程开发） | | | | |
| 实验内容与完成情况：  **（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 |   通过hdfs dfs -put命令将A、B文件上传到HDFS，操作过程如下：    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 |   实现上述操作的Java代码如下：   |  | | --- | | package com.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 {  /\*\*  \* @param args  \* 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://localhost:9000");  String[] otherArgs = new String[]{"input","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);  }  } |   代码运行效果：      **（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 |   上传所需的文件1、2、3，过程如下：    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 |   实现上述操作的Java代码如下：   |  | | --- | | package com.MergeSort;  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 {  /\*\*  \* @param args  \* 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://localhost:9000");  String[] otherArgs = new String[]{"input","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 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);  }  } |   执行结果如下：    **（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 |   上传所需输入文件过程：    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 |   实现上述操作的Java代码如下：   |  | | --- | | package com.simple\_data\_mining;  import java.io.IOException;  import java.util.\*;  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 simple\_data\_mining {  public static int time = 0;  /\*\*  \* @param args  \* 输入一个child-parent的表格  \* 输出一个体现grandchild-grandparent关系的表格  \*/  //Map将输入文件按照空格分割成child和parent，然后正序输出一次作为右表，反序输出一次作为左表，需要注意的是在输出的value中必须加上左右表区别标志  public static class Map extends Mapper<Object, Text, Text, Text>{  public void map(Object key, Text value, Context context) throws IOException,InterruptedException{  String child\_name = new String();  String parent\_name = new String();  String relation\_type = new String();  String line = value.toString();  int i = 0;  while(line.charAt(i) != ' '){  i++;  }  String[] values = {line.substring(0,i),line.substring(i+1)};  if(values[0].compareTo("child") != 0){  child\_name = values[0];  parent\_name = values[1];  relation\_type = "1";//左右表区分标志  context.write(new Text(values[1]), new Text(relation\_type+"+"+child\_name+"+"+parent\_name));  //左表  relation\_type = "2";  context.write(new Text(values[0]), new Text(relation\_type+"+"+child\_name+"+"+parent\_name));  //右表  }  }  }    public static class Reduce extends Reducer<Text, Text, Text, Text>{  public void reduce(Text key, Iterable<Text> values,Context context) throws IOException,InterruptedException{  if(time == 0){ //输出表头  context.write(new Text("grand\_child"), new Text("grand\_parent"));  time++;  }  int grand\_child\_num = 0;  String grand\_child[] = new String[10];  int grand\_parent\_num = 0;  String grand\_parent[]= new String[10];  Iterator ite = values.iterator();  while(ite.hasNext()){  String record = ite.next().toString();  int len = record.length();  int i = 2;  if(len == 0) continue;  char relation\_type = record.charAt(0);  String child\_name = new String();  String parent\_name = new String();  //获取value-list中value的child    while(record.charAt(i) != '+'){  child\_name = child\_name + record.charAt(i);  i++;  }  i=i+1;  //获取value-list中value的parent  while(i<len){  parent\_name = parent\_name+record.charAt(i);  i++;  }  //左表，取出child放入grand\_child  if(relation\_type == '1'){  grand\_child[grand\_child\_num] = child\_name;  grand\_child\_num++;  }  else{//右表，取出parent放入grand\_parent  grand\_parent[grand\_parent\_num] = parent\_name;  grand\_parent\_num++;  }  }  if(grand\_parent\_num != 0 && grand\_child\_num != 0 ){  for(int m = 0;m<grand\_child\_num;m++){  for(int n=0;n<grand\_parent\_num;n++){  context.write(new Text(grand\_child[m]), new Text(grand\_parent[n]));  //输出结果  }  }  }  }  }  public static void main(String[] args) throws Exception{  // TODO Auto-generated method stub  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String[] otherArgs = new String[]{"input","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,"Single table join");  job.setJarByClass(simple\_data\_mining.class);  job.setMapperClass(Map.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);    }  } | | | | | |
| 出现的问题：  1.Exception in thread "main" org.apache.hadoop.mapred.FileAlreadyExistsException: Output directory hdfs://192.168.14.100:9000/user/galaxy-chaos/output3 already exists | | | | |
| 解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：  1.删除之前产生单独输出文件夹。 | | | | |