

(一) 编程实现文件合并和去重操作

1. 具体内容

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
```

2. 操作过程

1.启动 hadoop:

```
[xusheng@hadoop102 module]$ myhadoop.sh start
===== 启动 hadoop 集群=====
----- 启动 hdfs -----
Starting namenodes on [hadoop102]
Starting datanodes
Starting secondary namenodes [hadoop104]
----- 启动 yarn -----
Starting resourcemanager
Starting nodemanagers
----- 启动 historyserver -----
[xusheng@hadoop102 module]$ cd hadoop-3.1.3/
```

2. 需要首先删除HDFS中与当前Linux用户hadoop对应的input和output目录（即HDFS中的“/opt/module/hadoop-3.1.3/input”和“/opt/module/hadoop-3.1.3/output”目录），这样确保后面程序运行不会出现问题

```
cd /opt/module/hadoop-3.1.3/
./bin/hdfs dfs -rm -r input
./bin/hdfs dfs -rm -r output
```

```
[xusheng@hadoop102 module]$ cd
[xusheng@hadoop102 ~]$ cd /opt/module/hadoop-3.1.3/
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -rm -r input
rm: `input': No such file or directory
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -rm -r output
rm: `output': No such file or directory
```

3. 再在HDFS中新建与当前Linux用户hadoop对应的input目录，即“/opt/module/hadoop-3.1.3/input”目录

```
./bin/hdfs dfs -mkdir /input/test1/
```

创建A.txt B.txt,输入上述内容

```
vim A.txt
vim B.txt
```

```
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -mkdir /input/test1/
[xusheng@hadoop102 hadoop-3.1.3]$ vim A.txt
[xusheng@hadoop102 hadoop-3.1.3]$ vim B.txt
```

4. 将A,B上传到HDFS中

```
./bin/hdfs dfs -put ./A.txt /input/test1/
./bin/hdfs dfs -put ./B.txt /input/test1/
```

```
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -put ./A.txt /input/test1/
2022-05-30 11:16:50,607 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted
= false, remoteHostTrusted = false
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -put ./B.txt /input/test1/
2022-05-30 11:17:04,164 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted
```

```
package com.xusheng.mapreduce.shiyan;

import org.apache.hadoop.conf.Configuration;
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.io.IOException;

public class Merge {

    /**
     * @param xusheng
     * 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");
        conf.set("fs.defaultFS", "hdfs://hadoop102:8020");
    }
}
```

```

        String[] otherArgs = new String[] { "/input/test1", "/output/test1" }; /** 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);
    }
}

```

```

./bin/hdfs dfs -cat /output/test1/*

```

```

[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -cat /output/test1/*
2022-05-30 12:35:05,328 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
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

1. 具体内容

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
```

2. 操作过程

1. 创建1.txt , 2.txt , 3.txt,输入上述内容

再在HDFS中新建与当前Linux用户hadoop对应的input目录，即 “/opt/module/hadoop-3.1.3/input” 目录

```
[xusheng@hadoop102 hadoop-3.1.3]$ vim 1.txt
[xusheng@hadoop102 hadoop-3.1.3]$ vim 2.txt
[xusheng@hadoop102 hadoop-3.1.3]$ vim 3.txt
[xusheng@hadoop102 hadoop-3.1.3]$ vim 2.txt
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -mkdir /input/test2/
```

2. 将1.txt , 2.txt , 3.txt上传到HDFS中

```
./bin/hdfs dfs -put ./1.txt /input/test2/
./bin/hdfs dfs -put ./2.txt /input/test2/
./bin/hdfs dfs -put ./3.txt /input/test2/
```

```
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -put ./1.txt /input/test2/
2022-05-30 13:01:51,374 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted
= false, remoteHostTrusted = false
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -put ./2.txt /input/test2/
2022-05-30 13:02:00,566 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted
= false, remoteHostTrusted = false
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -put ./3.txt /input/test2/
2022-05-30 13:02:07,434 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted
= false, remoteHostTrusted = false
```

| <input type="checkbox"/> | Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|--------------------------|----------------------------|-------------------------|----------------------------|------|---------------|-------------------|------------|-----------------------|
| <input type="checkbox"/> | -rw-r--r-- | xusheng | supergroup | 12 B | May 30 13:01 | 3 | 128 MB | 1.txt |
| <input type="checkbox"/> | -rw-r--r-- | xusheng | supergroup | 10 B | May 30 13:02 | 3 | 128 MB | 2.txt |
| <input type="checkbox"/> | -rw-r--r-- | xusheng | supergroup | 8 B | May 30 | 3 | 128 MB | 3.txt |

```
package com.xusheng.mapreduce.shiyan;
```

```
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 xusheng
```

```
     * 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");
    conf.set("fs.defaultFS", "hdfs://hadoop102:8020");
    String[] otherArgs = new String[]{" /input/test2", "/output/test2"}; /* 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");//实例化Merge类
}

```

```

        job.setJarByClass(MergeSort.class); //设置主类名
        job.setMapperClass(Map.class); //指定使用上述代码自定义的Map类
        job.setReducerClass(Reduce.class); //指定使用上述代码自定义的Reduce类
        job.setPartitionerClass(Partition.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class); //设定Reduce类输出的<K,V>,V类型
        FileInputFormat.addInputPath(job, new Path(otherArgs[0])); //添加输入文件位置
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1])); //设置输出结果文件位置

        System.exit(job.waitForCompletion(true) ? 0 : 1); //提交任务并监控任务状态
    }
}

```

```
./bin/hdfs dfs -cat /output/test2/*
```

```

[xusheng@hadoop102 ~]$ ./bin/hdfs dfs -cat /output/test2/*
2022-05-30 13:09:38,877 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted
= false, remoteHostTrusted = false
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

1. 具体内容

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. 操作过程

1. 创建child.txt,输入上述内容

再在HDFS中新建与当前Linux用户hadoop对应的input目录，即“/opt/module/hadoop-3.1.3/input” 目录

```
./bin/hdfs dfs -mkdir /input/test3/  
vim child.txt
```

| 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 |

2. 将child.txt上传到HDFS中

```
./bin/hdfs dfs -put ./ child.txt /input/test3/
```

```
[xusheng@hadoop102 hadoop-3.1.3]$ ./bin/hdfs dfs -put ./child.txt /input/test3/
2022-05-30 13:20:13,016 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted
= false, remoteHostTrusted = false
```

```
package com.xusheng.mapreduce.shiyan;

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 xusheng
     * 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");
conf.set("fs.default.name", "hdfs://hadoop102:8020");
String[] otherArgs = new String[]{"input/test3", "output/test3"}; /* 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);

}
}

```