

- [3\) 测试：](#)
- [6.双表关联：](#)
- [1\) 数据：](#)
- [2\) 代码：](#)
- [\(1\) map和reduce：](#)
- [\(2\) 配置输出：](#)
- [3\) 测试：](#)

算法是程序的精髓所在，算法也是一个人是否适合做软件开发的衡量标准。当然算法不是衡量一个人是否聪明的标准，熟练掌握以下几种，做到触类旁通即可。

以下几个例子测试环境：伪分布式，IP 为 localhost，集群和 eclipse 在同一个系统内。

1.排序：

1) 数据：

```
hadoop fs -mkdir /import
创建一个或者多个文本，上传
hadoop fs -put test.txt /import/
```

test.txt ✕

34
3
2
54
23
872
58385
638623496
76630
2

test2.txt ✕

7039
27
4
659
299
2736
638836
425849
5

<http://my.oschina.net/vigiles>

hm@hm-ubuntu:~\$ hadoop fs -lsr /import/
-rw-r--r-- 1 hm supergroup 41 2013-08-04 12:22 /import/test.txt
-rw-r--r-- 1 hm supergroup 39 2013-08-04 12:22 /import/test2.txt
hm@hm-ubuntu:~\$

2) 代码：

```
1 package com.cuiweiyou.sort;
2
3 import java.io.IOException;
4
5 import org.apache.hadoop.conf.Configuration;
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.io.LongWritable;
8 import org.apache.hadoop.io.NullWritable;
9 import org.apache.hadoop.io.Text;
10 import org.apache.hadoop.mapreduce.Job;
11 import org.apache.hadoop.mapreduce.Mapper;
12 import org.apache.hadoop.mapreduce.Reducer;
13 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
14 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
15
16 //hadoop默认排序:
```

```
17 //如果k2、v2类型是Text-文本，结果是按照字典顺序
18 //如果k2、v2类型是LongWritable-数字，结果是按照数字大小顺序
19
20 public class SortTest {
21     /**
22      * 内部类：映射器 Mapper<KEY_IN, VALUE_IN, KEY_OUT, VALUE_OUT>
23      */
24     public static class MyMapper extends Mapper<LongWritable, Text, LongWritable, NullWritable> {
25         /**
26          * 重写map方法
27          */
28         public void map(LongWritable k1, Text v1, Context context) throws IOException, InterruptedException {
29             //这里v1转为k2-数字类型，舍弃k1。null为v2
30             context.write(new LongWritable(Long.parseLong(v1.toString())), NullWritable.get());
31             //因为v1可能重复，这时，k2也是可能有重复的
32         }
33     }
34
35     /**
36      * 内部类：拆分器 Reducer<KEY_IN, VALUE_IN, KEY_OUT, VALUE_OUT>
37      */
38     public static class MyReducer extends Reducer<LongWritable, NullWritable, LongWritable, NullWritable> {
39         /**
40          * 重写reduce方法
41          * 在此方法执行前，有个shuffle过程，会根据k2将对应的v2归并为v2[...]
42          */
43         protected void reduce(LongWritable k2, Iterable<NullWritable> v2, Reducer<LongWritable, Context context) throws IOException, InterruptedException {
44             //k2=>k3, v2[...]舍弃。null => v3
45             context.write(k2, NullWritable.get());
46             //此时，k3如果发生重复，根据默认算法会发生覆盖，即最终仅保存一个k3
47         }
48     }
49
50     public static void main(String[] args) throws Exception {
51         // 声明配置信息
52         Configuration conf = new Configuration();
53         conf.set("fs.default.name", "hdfs://localhost:9000");
54
55         // 创建作业
56         Job job = new Job(conf, "SortTest");
57         job.setJarByClass(SortTest.class);
58
59         // 设置mr
60         job.setMapperClass(MyMapper.class);
61         job.setReducerClass(MyReducer.class);
62
63         // 设置输出类型，和Context上下文对象write的参数类型一致
64         job.setOutputKeyClass(LongWritable.class);
65         job.setOutputValueClass(NullWritable.class);
66
67         // 设置输入输出路径
68         FileInputFormat.setInputPaths(job, new Path("/import/"));
69         FileOutputFormat.setOutputPath(job, new Path("/out"));
70
71         // 执行
72         System.exit(job.waitForCompletion(true) ? 0 : 1);
73     }
74 }
```

3) 测试：

可以看到，不仅排序而且去重了。

```
drwxr-xr-x  - hm supergroup      0 2013-08-04 12:25 /out
-rw-r--r--  3 hm supergroup      0 2013-08-04 12:25 /out/_SUCCESS
-rw-r--r--  3 hm supergroup    78 2013-08-04 12:25 /out/part-r-00000
hm@hm-ubuntu:~$ hadoop fs -cat /out/*
2
3
4
5
23
27
34
54
299
659
872
2736
7039
58385
76630
425849
638836
638623496
hm@hm-ubuntu:~$
```

2.去重：

需求：查取手机号有哪些。这里的思路 and 上面排序算法的思路是一致的，仅仅多了分割出手机号这一步骤。

1) 数据：

创建两个文本，手动输入一些测试内容。每个字段用制表符隔开。日期，电话，地址，方式，数据量。

```

hm@ohm-ubuntu:~$ hadoop fs -cat /import/test.txt
201306030101 13141407390 beijing wap 45039624
201306040124 13141407390 beijing wap 458966788
201306121138 13141407390 beijing wap 59896403
201307150710 13121399732 beijing wap 331039621
201307172339 13141407390 beijing wap 628396888
201307271020 13121399732 beijing wap 1223962240
201307291051 13121399732 shandong wap 1443963540
201308030101 13141407390 beijing wap 5240396
201308040101 13141407390 henan wap 456788896
201308050338 13141407390 beijing wap 23059840396
201308050930 13121399732 beijing wap 94331210396
201308071339 13141407390 shandong wap 28888396
201308071349 13121399732 anhui wap 1122222240396
201308091359 13121399732 shandong wap 443540396001
hm@ohm-ubuntu:~$ hadoop fs -cat /import/test2.txt
201306030101 18253395109 beijing wap 39624
201306040124 18253395109 beijing wap 966788
201306121138 18253395109 beijing wap 96403
201307150710 13121399732 beijing wap 31039621
201307172339 18253395109 beijing wap 8396888
201307271020 13121399732 beijing wap 23962240
201307291051 18253395109 shandong wap 43963540
201308030101 18253395109 beijing wap 40396
201308040101 18253395109 henan wap 56788896
201308050338 13141407390 beijing wap 3059840396
201308050930 13121399732 beijing wap 4331210396
201308071339 18253395109 shandong wap 8888396
201308071349 18253395109 anhui wap 1222222240396
201308091359 13121399732 shandong wap 3540396001

```

2) 代码：

(1) map和reduce：

```

1  /**
2      * 映射器 Mapper<KEY_IN, VALUE_IN, KEY_OUT, VALUE_OUT>
3      */
4  public static class MyMapper extends Mapper<LongWritable, Text, Text, NullWritable> {
5      /**
6       * 重写map方法
7       */
8      protected void map(LongWritable k1, Text v1, Context context) throws IOException ,InterruptedException {
9          //按照制表符进行分割
10         String[] tels = v1.toString().split("\t");
11         //k1 => k2-第2列手机号, null => v2
12         context.write(new Text(tels[1]), NullWritable.get());
13     }
14 }
15
16
17 /*****
18  * 在map后, reduce前, 有个shuffle过程, 会根据k2将对应的v2归并为v2[...]
19  *****/
20
21
22 /**
23  * 拆分器 Reducer<KEY_IN, VALUE_IN, KEY_OUT, VALUE_OUT>
24  */

```



```
25     public static class MyReducer extends Reducer<Text, NullWritable, Text, NullWritable> {
26         /**
27          * 重写reduce方法
28          */
29         protected void reduce(Text k2, Iterable<NullWritable> v2, Context context) throws IOException ,InterruptedException {
30             //此时，k3如果发生重复，根据默认算法会发生覆盖，即最终仅保存一个k3，达到去重到效果
31             context.write(k2, NullWritable.get());
32         }
33     }
```

(2) 配置输出：

```
1 // 设置输出类型，和Context上下文对象write的参数类型一致
2 job.setOutputKeyClass(Text.class);
3 job.setOutputValueClass(NullWritable.class);
```

3) 测试：

```
hm@hm-ubuntu:~$ hadoop fs -cat /out/*
13121399732
13141407390
18253395109
hm@hm-ubuntu:~$
```

3.过滤：

需求：查询在北京地区发生的上网记录。思路同上，当写出 k2 、 v2 时加一个判断即可。

1) 数据：

同上。

2) 代码：

(1) map和reduce：

```
1 /**
2  * 内部类：映射器 Mapper<KEY_IN, VALUE_IN, KEY_OUT, VALUE_OUT>
3  */
4     public static class MyMapper extends Mapper<LongWritable, Text, Text, NullWritable> {
5         /**
6          * 重写map方法
7          */
8         protected void map(LongWritable k1, Text v1, Context context) throws IOException ,InterruptedException {
9             //按照制表符进行分割
10             final String[] adds = v1.toString().split("\t");
11             //地址在第3列
12             //k1 => k2-地址, null => v2
13             if(adds[2].equals("beijing")){
14                 context.write(new Text(v1.toString()), NullWritable.get());
15             }
```

```
15     }
16 }
17 }
18
19 /**
20  * 内部类：拆分器 Reducer<KEY_IN, VALUE_IN, KEY_OUT, VALUE_OUT>
21  */
22 public static class MyReducer extends Reducer<Text, NullWritable, Text, NullWritable> {
23     /**
24      * 重写reduce方法
25      */
26     protected void reduce(Text k2, Iterable<NullWritable> v2, Context context) throws IOException ,InterruptedException {
27         context.write(k2, NullWritable.get());
28     }
29 }
```

(2) 配置输出：

```
1 // 设置输出类型，和Context上下文对象write的参数类型一致
2 job.setOutputKeyClass(Text.class);
3 job.setOutputValueClass(NullWritable.class);
```

3) 测试：

```
hm@hm-ubuntu:~$ hadoop fs -cat /out/*
201306030101 13141407390 beijing wap 45039624
201306030101 18253395109 beijing wap 39624
201306040124 13141407390 beijing wap 458966788
201306040124 18253395109 beijing wap 966788
201306121138 13141407390 beijing wap 59896403
201306121138 18253395109 beijing wap 96403
201307150710 13121399732 beijing wap 31039621
201307150710 13121399732 beijing wap 331039621
201307172339 13141407390 beijing wap 628396888
201307172339 18253395109 beijing wap 8396888
201307271020 13121399732 beijing wap 1223962240
201307271020 13121399732 beijing wap 23962240
201308030101 13141407390 beijing wap 5240396
201308030101 18253395109 beijing wap 40396
201308050338 13141407390 beijing wap 23059840396
201308050338 13141407390 beijing wap 3059840396
201308050930 13121399732 beijing wap 4331210396
201308050930 13121399732 beijing wap 94331210396
hm@hm-ubuntu:~$
```

4.TopN：

这个算法非常经典，面试必问。实现这个效果的算法也很多。下面是个简单的示例。
需求：找到流量最大值；找出前5个最大值。

1) 数据：

同上。

2) 代码1-最大值 :

(1) map和reduce :

```
1  //map
2  public static class MyMapper extends Mapper<LongWritable, Text, LongWritable, NullWritable> {
3
4      //首先创建一个临时变量, 保存一个可存储的最小值: Long.MIN_VALUE=-9223372036854775808
5      long temp = Long.MIN_VALUE;
6
7      //找出最大值
8      protected void map(LongWritable k1, Text v1, Context context) throws IOException ,InterruptedException {
9          //按照制表符进行分割
10         final String[] flows = v1.toString().split("\t");
11         //将文本转数值
12         final long val = Long.parseLong(flows[4]);
13         //如果v1比临时变量大, 则保存v1的值
14         if(temp<val){
15             temp = val;
16         }
17     }
18
19     /** ---此方法在全部的map任务结束后执行一次。这时仅输出临时变量到最大值--- */
20     protected void cleanup(Context context) throws IOException ,InterruptedException {
21         context.write(new LongWritable(temp), NullWritable.get());
22         System.out.println("文件读取完毕");
23     }
24 }
25
26 //reduce
27 public static class MyReducer extends Reducer<LongWritable, NullWritable, LongWritable, NullWritable> {
28     //临时变量
29     Long temp = Long.MIN_VALUE;
30
31     //因为一个文件得到一个最大值, 再次将这些值比对, 得到最大的
32     protected void reduce(LongWritable k2, Iterable<NullWritable> v2, Context context) throws IOException ,InterruptedException {
33
34         long long1 = Long.parseLong(k2.toString());
35         //如果k2比临时变量大, 则保存k2的值
36         if(temp<long1){
37             temp = long1;
38         }
39     }
40
41     /** !!! 此方法在全部的reduce任务结束后执行一次。这时仅输出临时变量到最大值!!! */
42     protected void cleanup(Context context) throws IOException, InterruptedException {
43         context.write(new LongWritable(temp), NullWritable.get());
44     }
45 }
```

(2) 配置输出 :

```
1  // 设置输出类型
2  job.setOutputKeyClass(LongWritable.class);
3  job.setOutputValueClass(NullWritable.class);
```

3) 测试1 :

```
hm@hm-ubuntu:~$ hadoop fs -cat /out/*
11222222240396
hm@hm-ubuntu:~$
```

4) 代码2-TopN :

(1) map和reduce :

```
1  //map
2  public static class MyMapper extends Mapper<LongWritable, Text, LongWritable, NullWritable> {
3
4      //首先创建一个临时变量, 保存一个可存储的最小值: Long.MIN_VALUE=-9223372036854775808
5      long temp = Long.MIN_VALUE;
6      //Top5存储空间
7      long[] tops;
8
9      /** 次方法在run中调用, 在全部map之前执行一次 */
10     protected void setup(Context context) {
11         //初始化数组长度为5
12         tops = new long[5];
13     }
14
15     //找出最大值
16     protected void map(LongWritable k1, Text v1, Context context) throws IOException ,InterruptedException {
17         //按照制表符进行分割
18         final String[] flows = v1.toString().split("\t");
19         //将文本转数值
20         final long val = Long.parseLong(flows[4]);
21         //保存在0索引
22         tops[0] = val;
23         //排序后最大值在最后一个索引, 这样从后到前依次减小
24         Arrays.sort(tops);
25     }
26
27     /** ---此方法在全部到map任务结束后执行一次。这时仅输出临时变量到最大值--- */
28     protected void cleanup(Context context) throws IOException ,InterruptedException {
29         //保存前5条数据
30         for( int i = 0; i < tops.length; i++) {
31             context.write(new LongWritable(tops[i]), NullWritable.get());
32         }
33     }
34 }
35
36 //reduce
37 public static class MyReducer extends Reducer<LongWritable, NullWritable, LongWritable, NullWritable> {
38     //临时变量
39     Long temp = Long.MIN_VALUE;
40     //Top5存储空间
41     long[] tops;
42
43     /** 次方法在run中调用, 在全部map之前执行一次 */
44     protected void setup(Context context) {
45         //初始化长度为5
46         tops = new long[5];
47     }
```



```
48 //因为每个文件都得到5个值，再次将这些值比对，得到最大的
49 protected void reduce(LongWritable k2, Iterable<NullWritable> v2, Context context) throws IOException ,InterruptedException {
50
51     long top = Long.parseLong(k2.toString());
52     //
53     tops[0] = top;
54     //
55     Arrays.sort(tops);
56 }
57
58 /** ---此方法在全部到reduce任务结束后执行一次。输出前5个最大值--- */
59 protected void cleanup(Context context) throws IOException, InterruptedException {
60     //保存前5条数据
61     for( int i = 0; i < tops.length; i++) {
62         context.write(new LongWritable(tops[i]), NullWritable.get());
63     }
64 }
65 }
66 }
```

(2) 配置输出 :

```
1 // 设置输出类型
2 job.setOutputKeyClass(LongWritable.class);
3 job.setOutputValueClass(NullWritable.class);
```

5) 测试2 :

```
hm@hm-ubuntu:~$ hadoop fs -cat /out/*
23059840396
94331210396
443540396001
1222222240396
11222222240396
hm@hm-ubuntu:~$
```

5.单表关联 :

本例中的单表实际就是一个文本文件。

1) 数据 :

```
hm@hm-ubuntu:~$ hadoop fs -cat /import/test.txt
张三    张三爸爸
张三爸爸    张三爷爷
李四    李四爸爸
李四爸爸    李四爷爷
王五    王五爸爸
王五爸爸    王五爷爷
hm@hm-ubuntu:~$
```

这是提供的数据

<http://my.oschina.net/vigiles>

这是我们想得到的数

new	1
1	张三 张三爷爷
2	李四 李四爷爷
3	王五 王五爷爷

2) 代码：

(1) map和reduce：

```
1  //map
2  public static class MyMapper extends Mapper<LongWritable, Text, Text, Text> {
3      //拆分原始数据
4      protected void map(LongWritable k1, Text v1, Context context) throws IOException ,InterruptedException {
5          //按制表符拆分记录
6          String[] splits = v1.toString().split("\t");
7          //一条k2v2记录: 把孙辈作为k2; 祖辈加下划线区分, 作为v2
8          context.write(new Text(splits[0]), new Text("_"+splits[1]));
9          //一条k2v2记录: 把祖辈作为k2; 孙辈作为v2。就是把原两个单词调换位置保存
10         context.write(new Text(splits[1]), new Text(splits[0]));
11     }
12
13     /**
14      * 张三      _张三爸爸
15      * 张三爸爸   _张三
16
17      * 张三爸爸   _张三爷爷
18      * 张三爷爷   _张三爸爸
19      */
20 }
21
22 //reduce
23 public static class MyReducer extends Reducer<Text, Text, Text, Text> {
24     //拆分k2v2[...]数据
25     protected void reduce(Text k2, Iterable<Text> v2, Context context) throws IOException ,InterruptedException {
26         String grandchild = ""; //孙辈
27         String grandfather = ""; //祖辈
28
29         /**
30          * 张三爸爸      [_张三爷爷, 张三]
31          */
32
33         //从迭代中遍历v2[...]
34         for (Text man : v2) {
35             String p = man.toString();
36             //如果单词是以下划线开始的
37             if(p.startsWith("_")){
38                 //从索引1开始截取字符串, 保存到祖辈变量
39                 grandfather = p.substring(1);
40             }
41             //如果单词没有下划线起始
42             else{
43                 //直接赋值给孙辈变量
44                 grandchild = p;
45             }
46         }
47
48         //在得到有效数据的情况下
49         if( grandchild!="" && grandfather!=""){
50             //写出得到的结果。
51             context.write(new Text(grandchild), new Text(grandfather));
52         }
53
54         /**
55          * k3=张三, v3=张三爷爷
56          */
57     }
```

```
57     }
58 }
```

(2) 配置输出 :

```
1 // 设置输出类型
2 job.setOutputKeyClass(Text.class);
3 job.setOutputValueClass(Text.class);
```

3) 测试 :

```
hm@hm-ubuntu:~$ hadoop fs -cat /out/*
张三    张三爷爷
李四     李四爷爷
王五     王五爷爷
hm@hm-ubuntu:~$
```

6.双表关联 :

本例中仍简单采用两个文本文件。

1) 数据 :

```
hm@hm-ubuntu:~$ hadoop fs -cat /import/test.txt
张三    1
张三爸爸    2
李四     3
李四爸爸    4
王五     5
王五爸爸    6
hm@hm-ubuntu:~$ hadoop fs -cat /import/test2.txt
1    山东
2    上海
3    广州
4    北京
5    广西
6    云南
hm@hm-ubuntu:~$
```

new	1	
1	张三	山东
2	张三爸爸	上海
3	李四	广州
4	李四爸爸	北京
5	王五	广西
6	王五爸爸	云南

提供的两个表单数据

我们想要得到的数据

<http://my.oschina.net/vigiles>

2) 代码 :

(1) map和reduce :

```
1 //map
2 public static class MyMapper extends Mapper<LongWritable, Text, Text, Text> {
3     //拆分原始数据
4     protected void map(LongWritable k1, Text v1, Context context) throws IOException ,InterruptedException {
5         //拆分记录
6         String[] splited = v1.toString().split("\\t");
```

```

7      //如果第一列是数字（使用正则判断），就是地址表
8      if(splited[0].matches("^[-+]?([0-9]+)([.]( [0-9]+))?|([.]( [0-9]+))?$")){
9          String addreId = splited[0];
10         String address = splited[1];
11         //k2, v2-加两条下划线作为前缀标识为地址
12         context.write(new Text(addreId), new Text("__"+address));
13     }
14     //否则就是人员表
15     else{
16         String personId = splited[1];
17         String persName = splited[0];
18         //k2, v2-加两条横线作为前缀标识为人员
19         context.write(new Text(personId), new Text("--"+persName));
20     }
21     /**
22     1   __北京
23     1   --张三
24     */
25 }
26 }
27
28 //reduce
29 public static class MyReducer extends Reducer<Text, Text, Text, Text> {
30     //拆分k2v2[...]数据
31     protected void reduce(Text k2, Iterable<Text> v2, Context context) throws IOException ,InterruptedException {
32         String address = ""; //地址
33         String person = ""; //人员
34         /**
35         1, [__北京, --张三]
36         */
37         //迭代的是address或者person
38         for (Text text : v2) {
39             String tmp = text.toString();
40
41             if(tmp.startsWith("__")){
42                 //如果是__开头的是address
43                 address = tmp.substring(2); //从索引2开始截取字符串
44             }
45             if(tmp.startsWith("--")){
46                 //如果是--开头的是person
47                 person = tmp.substring(2);
48             }
49         }
50         context.write(new Text(person), new Text(address));
51     }
52     /**
53     k3=张三, v3=北京
54     */
55 }

```

(2) 配置输出：

```

1 // 设置输出类型
2 job.setOutputKeyClass(Text.class);
3 job.setOutputValueClass(Text.class);

```


3) 测试：

```
hm@hm-ubuntu:~$ hadoop fs -cat /out/*
张三      山东
张三爸爸   上海
李四       广州
李四爸爸   北京
王五       广西
王五爸爸   云南
hm@hm-ubuntu:~$
```


- end

分享到： 新浪微博  腾讯微博 

声明：OSCHINA 博客文章版权属于作者，受法律保护。未经作者同意不得转载。

- [« 上一篇](#)
- [下一篇 »](#)

评论0



插入：[表情](#) [开源软件](#)

发表评论

[关闭](#)插入表情

关闭相关文章阅读

- 2013/05/20 [Hadoop上路_03-伪分布式集群配置...](#)
- 2013/12/05 [Hadoop完全分布式搭建](#)
- 2014/12/24 [完全分布式 hadoop 1.X集群部署...](#)
- 2011/08/19 [分布式计算核武器hadoop--性能篇...](#)
- 2013/05/22 [Hadoop上路_10-分布式Hadoop集群搭建...](#)