

# 实验四：合并两表实验报告

小组成员：MF20330007 陈明远 MF20330003 陈迪

MG20330095 朱志威

## ● 实验要求：

本次实验要求将两个文件通过 mapreduce 先进行合并，将结果输入到 HDFS 上，然后通过 Hive 进行建表管理输出结果，最后通过 Hive 语句能查询到所建的表。

## ● 设计思路：

### 1. 数据类型

本次实验进行合并的两张表为 order.txt 和 product.txt，所以本次实验选择定制自己的数据类型，定义一个自己的 Tablebean 类，然后实现 Writable 接口（如果 key 值需要进行自定义排序，则实现 WritableComparable 接口）

### 2. Map:

本次实验有两种实现方式，Map 端进行合并或者 reduce 端进行合并。

#### 1) . 在 Map 端进行合并

使用全局文件复制方法，利用 Hadoop 的 Distirbuted Cache 机制，在 Mapper 的 setup 中，将需要进行复制的文件装入 BufferedReader 中，然后建立一个 HashMap，将产品 id 作为 key，其余数据作为 value，然

后在 map 方法中直接进行合并，不需要通过 reduce 即可完成合并操作，此时 Map 输出的 key 为 Tablebean 类，value 为 Nullwritable 类型。这种方式适合一个表大一个表小的情况。

## 2). reduce 端进行合并

如果在 reduce 端进行合并，则只需要在 Map 中输出不同表的数据，然后送入 reduce 中，其中 map 输出的 k 为产品 id，value 为我们自定义的数据类型 Tablebean 类。这种合并方式会造成数据倾斜。

## 3. Reduce:

如果在 reduce 端进行合并，则需要将 map 输出的 (k, v) 进行整合，因为 reduce 中将同一 key 的不同 value 进行了 Merge，所以我们只需要将其中 order 中的数据存入一个 ArrayList 中，然后遍历这个数组，把所有的 order 数据与 product 中的数据进行合并，输出的 reduce 的 key 为 Tablebean 类，value 为 NullWriatable 类。

## ● 实现代码:

本次实验我们采用 reduce 端合并，代码如下:

### 1. 用户自制数据类型 Tablebean 类:

```
1. public static class Tablebean implements WritableComparable<Tablebean> {
2.     //自定义自己的用户类，用来当作 key 或者 value, 需要实现 writable 接口，如果需要自定义 map 的排序。则实现 writablecomparable
3.     private String order_id; //订单 id
4.     private String order_date; //订单日期
```

```
5.         private int product_id;//产品 id
6.         private String name;//产品名字
7.         private int amount;//购买数量
8.         private int price;//单价
9.         private int flag;//标记，用来记录是第几个文件的数据
10.        public Tablebean() {
11.            super();
12.        }
13.        public void set(String order_id, String order_date, int product_id,
    String name, int amount, int price, int flag) {
14.            this.order_id = order_id;
15.            this.order_date = order_date;
16.            this.product_id = product_id;
17.            this.name = name;
18.            this.amount = amount;
19.            this.price = price;
20.            this.flag = flag;
21.        }
22.
23.        public String getOrder_id() {
24.            return this.order_id;
25.        }
26.        public void setOrder_id(String order_id) {
27.            this.order_id = order_id;
28.        }
29.
30.        public String getOrder_date() {
31.            return this.order_date;
32.        }
33.        public void setOrder_date(String order_date) {
34.            this.order_date = order_date;
35.        }
36.
37.        public int getProduct_id() {
38.            return this.product_id;
39.        }
40.        public void setProduct_id(int product_id) {
41.            this.product_id = product_id;
42.        }
43.
44.        public String getName() {
45.            return this.name;
46.        }
47.        public void setName(String name) {
```

```
48.         this.name = name;
49.     }
50.     public int getAmount() {
51.         return this.amount;
52.     }
53.     public void setAmount(int amount) {
54.         this.amount = amount;
55.     }
56.
57.     public int getPrice() {
58.         return this.price;
59.     }
60.     public void setPrice(int price) {
61.         this.price = price;
62.     }
63.     public int getFlag() {
64.         return this.flag;
65.     }
66.     public void setFlag(int flag) {
67.         this.flag = flag;
68.     }
69.     @Override
70.     public void write(DataOutput output) throws IOException {
71.         //任何需要定制的数据类型都至少要实现 write 和 readFields 两个接口
72.         output.writeUTF(this.order_id);
73.         output.writeUTF(this.order_date);
74.         output.writeInt(this.product_id);
75.         output.writeUTF(this.name);
76.         output.writeInt(this.amount);
77.         output.writeInt(this.price);
78.         output.writeInt(this.flag);
79.     }
80.     @Override
81.     public void readFields(DataInput input) throws IOException {
82.         this.order_id = input.readUTF();
83.         this.order_date = input.readUTF();
84.         this.product_id = input.readInt();
85.         this.name = input.readUTF();
86.         this.amount = input.readInt();
87.         this.price = input.readInt();
88.         this.flag = input.readInt();
89.     }
90.     @Override //toString 方法用来指定输入到 context 的 value 格式
91.     public String toString() {
```

```

92.         return this.order_id + " " + this.order_date + " " + this.produc
           t_id + " " + this.name + " " + this.price + " " + this.amount;
93.     }
94.
95.     public int compareTo(Tablebean o) {
96.         return this.product_id - o.product_id;
97.     }
98. }
99. //定义自己的 map, 将产品 id 一样的(k,v)送到同一个 reduce
100. public static class DistributedMapper extends Mapper<LongWritable, Text
      , Text, Tablebean> {
101.     private Tablebean v = new Tablebean();
102.     private Text k = new Text();
103.     @Override
104.     protected void map(LongWritable key, Text value, Context context)
105.         throws IOException, InterruptedException {
106.         String line = value.toString();
107.         FileSplit fs = (FileSplit) context.getInputSplit();
108.         String filename = fs.getPath().getName();//获取文件名
109.         if(filename.startsWith("order")) {
110.             String[] fileds = line.split(" ");
111.             v.setOrder_id(fileds[0]);
112.             v.setOrder_date(fileds[1]);
113.             v.setProduct_id(Integer.valueOf(fileds[2]));
114.             v.setAmount(Integer.valueOf(fileds[3]));
115.             v.setName("");
116.             v.setPrice(0);
117.             v.setFlag(0);
118.
119.             k.set(fileds[2]);//将 key 设置成产品 id
120.         }
121.         else {
122.             String[] fileds = line.split(" ");
123.             v.setProduct_id(Integer.valueOf(fileds[0]));
124.             v.setName(fileds[1]);
125.             v.setPrice(Integer.valueOf(fileds[2]));
126.             v.setOrder_id("");
127.             v.setOrder_date("");
128.             v.setFlag(1);
129.             v.setAmount(0);
130.
131.             k.set(fileds[0]);
132.         }
133.         context.write(k, v);

```

```
134.         }
135.     }
```

## 2. Map 端:

```
1. public static class DistributedMapper extends Mapper<LongWritable, Text, Text, Tablebean> {
2.     private Tablebean v = new Tablebean();
3.     private Text k = new Text();
4.     @Override
5.     protected void map(LongWritable key, Text value, Context context)
6.         throws IOException, InterruptedException {
7.         String line = value.toString();
8.         FileSplit fs = (FileSplit) context.getInputSplit();
9.         String filename = fs.getPath().getName();//获取文件名
10.        if(filename.startsWith("order")) {
11.            String[] fileds = line.split(" ");
12.            v.setOrder_id(fileds[0]);
13.            v.setOrder_date(fileds[1]);
14.            v.setProduct_id(Integer.valueOf(fileds[2]));
15.            v.setAmount(Integer.valueOf(fileds[3]));
16.            v.setName("");
17.            v.setPrice(0);
18.            v.setFlag(0);
19.
20.            k.set(fileds[2]);//将 key 设置成产品 id
21.        }
22.        else {
23.            String[] fileds = line.split(" ");
24.            v.setProduct_id(Integer.valueOf(fileds[0]));
25.            v.setName(fileds[1]);
26.            v.setPrice(Integer.valueOf(fileds[2]));
27.            v.setOrder_id("");
28.            v.setOrder_date("");
29.            v.setFlag(1);
30.            v.setAmount(0);
31.
32.            k.set(fileds[0]);
33.        }
34.        context.write(k, v);
35.    }
36. }
```

### 3. Reduce 端:

```
4. public static class MyReducer extends Reducer<Text, Tablebean, Tablebean, NullWritable> {
5.     @Override
6.     public void reduce(Text key, Iterable<Tablebean> values, Context context)
7.         throws IOException, InterruptedException {
8.         Tablebean pt = new Tablebean();
9.         ArrayList<Tablebean> arrayList = new ArrayList<Tablebean>();
10.
11.         for(Tablebean value : values) {//一个 values 对应一个 key,所以
12.             values 中所有数据的产品 id 是一样的
13.             if(value.getFlag() == 1) {//如果是 product.txt 中的数据,则只进行一次复制,用来后面和 order 中的数据合并
14.                 try {
15.                     BeanUtils.copyProperties(pt, value);
16.                 } catch (Exception e){
17.                     e.printStackTrace();
18.                 }
19.             }else {
20.                 Tablebean other = new Tablebean();
21.                 try {
22.                     BeanUtils.copyProperties(other, value);
23.                     arrayList.add(other);//将 order 中的数据存储
24.                 }catch (Exception e) {
25.                     e.printStackTrace();
26.                 }
27.             }
28.         }
29.         for(Tablebean tablebean : arrayList) {//将 order 中的数据与
30.             product 中的数据合并
31.             tablebean.setName(pt.getName());
32.             tablebean.setPrice(pt.getPrice());
33.             context.write(tablebean, NullWritable.get());
34.         }
35.     }
```

### 4. Main 函数:

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

```

2. Configuration conf = new Configuration();
3. String[] otherargs = (new GenericOptionsParser(conf, args)).getRemaining
  Args();
4. if (otherargs.length != 2) {
5.     System.err.println("Usage:hadoop jar MyJoin2.jar MyJoin2  main input
    output");
6.     System.exit(2);
7. }
8.
9. Job job = Job.getInstance(conf, "MyJoin2");
10. job.setJarByClass(MyJoin2.class);
11. job.setMapperClass(DistributedMapper.class);
12. job.setMapOutputKeyClass(Text.class);
13. job.setMapOutputValueClass(Tablebean.class);
14. job.setReducerClass(MyReducer.class);
15. job.setOutputKeyClass(Tablebean.class);
16. job.setOutputValueClass(NullWritable.class);
17. job.setNumReduceTasks(1);
18. FileInputFormat.addInputPath(job, new Path(otherargs[0]));
19. FileOutputFormat.setOutputPath(job, new Path(otherargs[1]));
20. System.exit(job.waitForCompletion(true) ? 0 : 1);
21. }

```

## ● 实验结果

SQL On Hadoop

2021st06

表名

orders

Hive 2021st06 添加一个名字, 添加一段描述

运行 保存

select \* from orders

执行的历史记录 保存的语句 执行结果

| Orders.order Id | Orders.order Date | Orders.pid | Orders.name | Orders.price | Orders.amount |
|-----------------|-------------------|------------|-------------|--------------|---------------|
| 1162            | 20190801          | 1          | chulizi     | 3999         | 64            |
| 1165            | 20190801          | 1          | chulizi     | 3999         | 98            |
| 1213            | 20190802          | 1          | chulizi     | 3999         | 28            |
| 1168            | 20190801          | 1          | chulizi     | 3999         | 91            |
| 1175            | 20190801          | 1          | chulizi     | 3999         | 65            |
| 1210            | 20190802          | 1          | chulizi     | 3999         | 30            |
| 1016            | 20190731          | 1          | chulizi     | 3999         | 84            |
| 1177            | 20190801          | 1          | chulizi     | 3999         | 15            |
| 1207            | 20190802          | 1          | chulizi     | 3999         | 52            |
| 1206            | 20190802          | 1          | chulizi     | 3999         | 81            |
| 1281            | 20190802          | 1          | chulizi     | 3999         | 93            |

图 1.hive 部分输出结果截图(1)



SQL On Hadoop

2021st06

表名

orders

Hive

2021st06

添加一个名字...添加一段描述...

运行

保存

```
select * from orders
```

执行的历史记录

保存的语句

执行结果

|      |          |   |       |      |     |
|------|----------|---|-------|------|-----|
| 1023 | 20190731 | 4 | apple | 5999 | 78  |
| 1048 | 20190731 | 4 | apple | 5999 | 80  |
| 1173 | 20190801 | 4 | apple | 5999 | 49  |
| 1193 | 20190801 | 4 | apple | 5999 | 47  |
| 1083 | 20190731 | 4 | apple | 5999 | 50  |
| 1192 | 20190801 | 4 | apple | 5999 | 34  |
| 1172 | 20190801 | 4 | apple | 5999 | 30  |
| 1260 | 20190802 | 4 | apple | 5999 | 16  |
| 1266 | 20190802 | 4 | apple | 5999 | 39  |
| 1022 | 20190731 | 4 | apple | 5999 | 100 |
| 1231 | 20190802 | 4 | apple | 5999 | 15  |

图 2.hive 部分输出结果(2)

| ID                             | User     | Name    | Application Type | Queue           | StartTime                     | FinishTime                    | State    | FinalStatus | Progress | Tracking UI | Blacklisted Nodes |
|--------------------------------|----------|---------|------------------|-----------------|-------------------------------|-------------------------------|----------|-------------|----------|-------------|-------------------|
| application_1604923458283_0299 | 2021st06 | Myllon2 | MAPREDUCE        | root.2021team06 | Wed Dec 9 23:08:23 +0800 2020 | Wed Dec 9 23:08:39 +0800 2020 | FINISHED | SUCCEEDED   |          | History     | N/A               |

图 3.集群监控页面

hadoop

MapReduce Job job\_1604923458283\_0299

Job Overview

Job Name:

Myllon2

User Name:

2021st06

Queue:

root.2021team06

State:

SUCCEEDED

Uberized:

false

Submitted:

Wed Dec 09 23:08:23 CST 2020

Started:

Wed Dec 09 23:05:13 CST 2020

Finished:

Wed Dec 09 23:05:24 CST 2020

Elapsed:

10sec

Diagnostics:

Average Map Time

2sec

Average Shuffle Time

2mins, 37sec

Average Merge Time

0sec

Average Reduce Time

-2mins, -34sec

| Attempt Number | Start Time                   | Node          | Logs |
|----------------|------------------------------|---------------|------|
| 1              | Wed Dec 09 23:05:09 CST 2020 | slave019:8042 | logs |

| Task Type    | Total  | Complete |            |
|--------------|--------|----------|------------|
| Map          | 2      | 2        |            |
| Reduce       | 1      | 1        |            |
| Attempt Type | Failed | Killed   | Successful |
| Maps         | 0      | 0        | 2          |
| Reduces      | 0      | 0        | 1          |

图 4.Job 执行情况页面

## ● 实验总结

本次实验通过不同方式合并两张表，对 Hadoop 编程更加熟悉，同时掌握了 hive 查表操作，更加体会到了 hadoop 处理数据的强大之处。