

# 大数据处理综合实验

## 实验三 Hive-MyJoin 实验报告

组别：2020st39

组长：171860662 山越 [1025331716@qq.com](mailto:1025331716@qq.com)

### 目录

大数据处理综合实验.....	1
实验三 Hive-MyJoin 实验报告 .....	1
组别：2020st39 .....	1
组长：171860662 山越 1025331716@qq.com.....	1
一 .      实验设计.....	1
(一)    设计思路.....	1
(二)    源代码说明.....	4
二 .      程序运行和实验结果说明.....	7
(一)    输出结果.....	7
(二)    WebUI 执行报告.....	8
三 .      遇到的问题和不足之处 .....	9
四 .      小组成员分工.....	9

### 一 . 实验设计

#### (一)  设计思路

##### 1. 安装 Hive

从官网下载 hive-2.3.7-bin 压缩包，在本地解压后，修改.bashrc，添加 HIVE\_HOME 和对应 PATH，使用 source 命令使其生效即可。

## 2. 运行 hive

进入 hive 文件夹，找到 bin 文件夹并进入，在终端执行指令 `./schematool -initSchema -dbType derby` 进行初始化，启动 hive 即可。

## 3. 设计 MapReduce 进行 join

为了提高效率与美观度，使得数据能够在到达 reducer 时已经按照某种顺序排好，我们构造了一个 OrderBean 类，以此实现 hadoop 的序列化机制，OrderBean 中包含两个表的订单 ID、订单日期、商品 ID 等所有信息，设置各变量的 set 和 get 函数以设置和查看信息，在其 compareTo()方法中定义比较两者的商品 ID pid 与商品名称 pname，根据两者的升序来排列数据，这样 mapper 输出的数据就是有序排列。

在 mapper 中，首先获得读取的文件名，定义 OrderBean 类变量 ob，如果文件名为 product.txt，则设置 ob 中的成员变量 pid、pname 和 price；如果文件名为 order.txt，则设置 ob 中的成员变量 oid、odata、pid 和 oamount。设置完后输出发送数据。

在数据传输到 reducer 前，要先将数据进行聚合，实现两张表数据的合并，自定义类 JoinComparator，对于两个不同的 OrderBean a,b，调用 compareTo()方法比较两者的 pid 并将值返回，如果一致则会将两者合并。

当数据传输到 reducer 里后，已经是合并好的有序排列，因此只需要将这些数据逐条输出即可。

在主函数中，设置 Mapper、Reducer、GroupingComparator 为自定义的三个类，设置 Map 的输出 key 格式为 Orderbean，value 格式为 Nullwriteable，最终输出的 key 格式为 Text，value 格式也为 Nullwriteable，其他设置均与之前实验类似。

## 4. 运行结果与 Hive 查看结果

运行 jar 包后，通过 `hadoop fs -cat` 查看结果文件夹中的 part-r-00000 文件，发现最终输出的表数据正确，进入 hive 后，运行 create table 时报错 FAILED: SemanticException ...，经查询后发现使用 hive 自带的内存数据库 derby 时应该先初始化，解决方法为，将 hive 目录中自动生成的 metastore\_db 文件夹重命名为 metastore\_db.tmp 后重新初始化即可。使用所给的 create table 指令建立 orders 表，建完后调用 `select * from orders` 命令查看表数据，结果正确。

```
[msc@RHELS7host ~]$ hive
which: no hbase in (/usr/java/jdk1.7.0_80/bin:/home/msc/hadoop/hadoop-2.7.1/bin:/home/msc/hadoop/hive-2.3.7/bin:/usr/java/jdk1.7.0_80/bin:/usr/java/jdk1.7.0_80/jre/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/home/msc/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/msc/hadoop/hive-2.3.7/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/msc/hadoop/hadoop-2.7.1/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/home/msc/hadoop/hive-2.3.7/lib/hive-common-2.3.7.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. tez, spark) or using Hive 1.X releases.
hive> create table orders(order_id int,order_date string,product_id string,product_name string,product_price int,amount int) row format delimited fields terminated by '\t' location '/lab3-out/part-r-00000';

FAILED: SemanticException org.apache.hadoop.hive.ql.metadata.HiveException: java.lang.RuntimeException: Unable to instantiate org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient
hive>
```

```
hive> show tables;
OK
orders
Time taken: 0.103 seconds, Fetched: 1 row(s)
hive> select * from orders;
OK
1010      20190802      2      huawei  3999      2
1009      20190802      2      huawei  3999      10
1007      20190801      2      huawei  3999      3
1004      20190731      2      huawei  3999      23
1003      20190731      2      huawei  3999      40
1012      20190802      3      xiaomi  2999      18
1011      20190802      3      xiaomi  2999      14
1006      20190801      3      xiaomi  2999      20
1002      20190731      3      xiaomi  2999      100
1008      20190801      4      apple  5999      23
1005      20190801      4      apple  5999      55
1001      20190731      4      apple  5999      2
Time taken: 10.27 seconds, Fetched: 12 row(s)
hive>
```

## (二) 源代码说明

### MyJoin.java

```
//MyJoin.java
import org.apache.hadoop.conf.Configuration;

public class MyJoin {
    public static void main(String[] args) throws Exception
    {
        Configuration conf=new Configuration();
        Job job=Job.getInstance(conf,"JoinApp");
        job.setJobName("JoinApp");

        job.setJarByClass(MyJoin.class);
        job.setInputFormatClass(TextInputFormat.class); //按行读入
        job.setMapperClass(JoinMapper.class);
        job.setReducerClass(JoinReducer.class);
        job.setGroupingComparatorClass(JoinComparator.class);
        //自定义分组比较器，以实现同一个pid的记录在一遍reduce函数中处理

        job.setMapOutputKeyClass(OrderBean.class); //map输出key为自定义OrderBean类型
        job.setMapOutputValueClass(NullWritable.class); //map输出value为空
        job.setOutputKeyClass(Text.class); //reduce输出key为Text类型
        job.setOutputValueClass(NullWritable.class); //reduce输出value为空

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

## OrderBean.java

```
//OrderBean.java
import org.apache.hadoop.io.WritableComparable;

public class OrderBean implements WritableComparable<OrderBean> {
    private String oid;
    private String odate;
    private String pid;
    private String pname;
    private String price;
    private String oamount;

    public OrderBean() { //初始化
        super();
        this.oid = "";
        this.odate = "";
        this.pid = "";
        this.pname = "";
        this.price = "";
        this.oamount = "";
    }

    public void setOid(String s){}
    public String getOid(){}

    public void setOdate(String s){}
    public String getOdate(){}

    public void setPid(String s){}
    public String getPid(){}

    public void setPname(String s){}
    public String getPname(){}

    public void setPrice(String s){}
    public String getPrice(){}

    public void setOamount(String s){}
    public String getOamount(){}

    //由于OrderBean要作为map传出的key，所以需要实现Comparable接口中的compareTo方法
    @Override
    public int compareTo(OrderBean o) {
        //先比较pid，若相等再比较pname，这样保证在Reducer端中同一组内，product记录会排在第一个
        int compare=this.pid.compareTo(o.pid);
        if(compare==0)
            return o.pname.compareTo(this.pname);
        else
            return compare;
    }

    //由于OrderBean要作为map传出的key，所以需要实现Writable接口中的write和readFields方法
    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeUTF(oid);
        dataOutput.writeUTF(odate);
        dataOutput.writeUTF(pid);
        dataOutput.writeUTF(pname);
        dataOutput.writeUTF(price);
        dataOutput.writeUTF(oamount);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        this.oid = dataInput.readUTF();
        this.odate = dataInput.readUTF();
        this.pid = dataInput.readUTF();
        this.pname = dataInput.readUTF();
        this.price = dataInput.readUTF();
        this.oamount = dataInput.readUTF();
    }
}
```

## JoinMapper.java

```
//JoinMapper.java
import org.apache.hadoop.io.NullWritable;

public class JoinMapper extends Mapper<Object,Text,OrderBean, NullWritable>
{
    private String filename;

    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        FileSplit fs = (FileSplit) context.getInputSplit();
        filename = fs.getPath().getName(); //获取输入的文件名
    }

    @Override
    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException
    {
        String[] fields=value.toString().split(" "); //对每行内容进行分割
        OrderBean ob = new OrderBean();
        if(filename.equals("product.txt")) //根据两种文件的内容填充OrderBean类型的变量
        {
            ob.setPid(fields[0]);
            ob.setName(fields[1]);
            ob.setPrice(fields[2]);
        }
        else
        {
            ob.setOid(fields[0]);
            ob.setOdate(fields[1]);
            ob.setPid(fields[2]);
            ob.setOamount(fields[3]);
        }
        context.write(ob,NullWritable.get()); //发射键值对，value为空
    }
}
```

## JoinReducer.java

```
//JoinReducer.java
import org.apache.hadoop.io.NullWritable;

public class JoinReducer extends Reducer<OrderBean,NullWritable,Text,NullWritable>{
    @Override
    protected void reduce(OrderBean key, Iterable<NullWritable> values, Context context) throws IOException, InterruptedException
    {
        Iterator<NullWritable> it=values.iterator();
        it.next(); //跳到该组记录中的第一条，即为product中的记录
        String name=key.getPname(); //获取product记录中的pname
        String price=key.getPrice(); //获取product记录中的price
        while(it.hasNext())
        {
            it.next(); //获取该组中的下一条记录，理应为一条order记录（此处对value迭代的同时会更新key的值）
            key.setName(name); //填充order记录中的pname
            key.setPrice(price); //填充order记录中的price
            Text outputKey = new Text(); //填充输出key
            outputKey.set(
                key.getOid() + "\t"
                + key.getOdate() + "\t"
                + key.getPid() + "\t"
                + key.getPname() + "\t"
                + key.getPrice() + "\t"
                + key.getOamount() + "\t");
            context.write(outputKey, NullWritable.get()); //发射键值对，value仍为空
        }
    }
}
```

## JoinComparator.java

```
//JoinComparator.java
import org.apache.hadoop.io.WritableComparable;

//利用reduce端的GroupingComparator来实现将一组bean看成相同的key
public class JoinComparator extends WritableComparator {
    protected JoinComparator() {
        super(OrderBean.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        OrderBean oa = (OrderBean) a;
        OrderBean ob = (OrderBean) b;
        return oa.getPid().compareTo(ob.getPid()); //比较两个bean时，只比较pid
        //若pid相等，则会分到一组中，交由一个Reducer的一遍reduce函数处理
    }
}
```

## 二．程序运行和实验结果说明

### (一) 输出结果

#### 1. join 的输出文件

输出文件在 HDFS 上的路径：/user/2020st39/Lab3\_out

File - /user/2020st39/Lab3_out/part-r-00...						Page 1 of 3
1162	20190801	1	chuizi	3999	64	
1165	20190801	1	chuizi	3999	98	
1213	20190802	1	chuizi	3999	28	
1168	20190801	1	chuizi	3999	91	
1175	20190801	1	chuizi	3999	65	
1210	20190802	1	chuizi	3999	30	
1016	20190731	1	chuizi	3999	84	
1177	20190801	1	chuizi	3999	15	
1207	20190802	1	chuizi	3999	52	
1206	20190802	1	chuizi	3999	81	
1281	20190802	1	chuizi	3999	93	
1178	20190801	1	chuizi	3999	50	
1047	20190731	1	chuizi	3999	52	
1202	20190802	1	chuizi	3999	47	
1201	20190802	1	chuizi	3999	78	

Cancel Download

## 2. Hive 建表结果

在 SQL On Hadoop 页面，执行下面的建表语句：

```
create table 2020st39_orders(order_id int,order_date string,product_id
string,product_name string,product_price int,amount int) row format delimited fields
terminated by '\t' location '/user/2020st39/Lab3_out';
```

建表成功，然后执行查询语句：

```
select * from 2020st39_orders limit 10;
```

结果如下：

SQL

default

Tables

- 2020st13\_wy\_orders
- 2020st17\_orders
- 2020st27\_basic\_orders
- 2020st27\_mapsidejoin\_...
- 2020st30\_orders
- 2020st47\_orders
- 2020st51orders
- 29\_orders
- 29\_orders1
- 29\_orders2
- 33\_orders
- neworders
- orders
- orders\_1
- orders\_2
- orders\_3
- orderss

Hive 2020st39 添加一个名字... 添加一段描述...

运行 保存

```
1 select * from 2020st39_orders limit 10;
```

Executed History Saved Queries Execute Result

2020st39 Orders.order Id	2020st39 Orders.order Date	2020st39 Orders.product Id	2020st39 Orders.product Name	2020st39 Orders.product Price	2020st39 Orders.amount
1162	20190801	1	chuizi	3999	64
1165	20190801	1	chuizi	3999	98
1213	20190802	1	chuizi	3999	28
1168	20190801	1	chuizi	3999	91
1175	20190801	1	chuizi	3999	65

## (二) WebUI 执行报告

doop

MapReduce Job job\_1572597966684\_3695

Logged in as: dr.who

Job Overview

Job Name: JoinApp

User Name: 2020st39

Queue: root.team39

State: SUCCEEDED

Uberized: false

Submitted: Thu Apr 30 16:42:28 CST 2020

Started: Thu Apr 30 16:42:15 CST 2020

Finished: Thu Apr 30 16:42:25 CST 2020

Elapsed: 10sec

Diagnostics:

Average Map Time: 2sec

Average Shuffle Time: -3sec

Average Merge Time: 0sec

Average Reduce Time: 6sec

ApplicationMaster	Attempt Number	Start Time	Node	Logs
	1	Thu Apr 30 16:42:11 CST 2020	slave012:8042	logs

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



### 三．遇到的问题 and 不足之处

1. 最开始仅仅是按照 PPT 上的代码结构进行填充, 并未深究 GroupingComparator 的用法、WritableComparable 接口实现, 所以实现过程中遇到了很多问题。于是我们查找了一些资料, 分析了 Hadoop 关于这部分的源码, 从而推进了实现过程。

### 四．小组成员分工

学号	姓名	分工
171860662	山越	提交集群和实验报告撰写
171860663	马少聪	代码实现
171860664	谢鹏飞	实验报告撰写
171860681	冯旭晨	代码实现