金融大数据实验二

221275038 杨念

一、实验任务

1、每日资金流入流出统计；

2、星期交易量统计；

3、用户活跃度分析；

4、交易行为影响因素分析。

二、具体实验任务内容

（一）任务一：每日资金流入流出统计

1、实验要求

根据 user_balance_table 表中的数据，编写MapReduce程序，统计所有用户每日的资金流入与 流出情况。

资金流入意味着申购行为，资金流出为赎回行为 。

注：每笔交易的资金流入和流出量分别由字段 total_purchase_amt 和 total_redeem_amt 表 示。请注意处理数据中的缺失值，将其视为零交易。

输出格式：<日期> TAB <资金流入量>,<资金流出量> , 例如： 20130701 32488348,5525022

2、实验思路

（1）mapper：写在`InflowOutflowMapper.java`文件里面，用于读取每一行数据，提取日期、流入量和流出量。

生成 <日期>*purchase 和 <日期>*redeem 作为键，分别输出流入和流出数据，以便 Reducer 进行区分。

（2）reducer：写在`InflowOutflowReducer.java`文件里面，根据键的后缀 _purchase 或 _redeem 来区分数据。

累加相同日期的流入和流出量。

在 cleanup 方法中，将每个日期的总流入和总流出结果输出为 <日期> TAB <流入量>,<流出量>。

（3）driver：写在`InflowOutflowDriver.java`文件里面，配置并启动任务，指定 Mapper 和 Reducer 类，设置输入输出路径。

这种设计让可以实现让Mapper 标记数据类型，Reducer 进行聚合，Driver 控制任务流。

3、实验步骤

（1）检查hdfs是否正常工作：`hdfs dfs -ls /`

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ ./sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [yawn-virtual-machine]
hadoop@yawn-virtual-machine:/usr/local/hadoop$ jps
3857 SecondaryNameNode
4001 Jps
3656 DataNode
3529 NameNode
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2024-10-23 00:35 /user
hadoop@yawn-virtual-machine:/usr/local/hadoop$
```

能够正常输出hdfs里面已有的东西就说明在正常工作。这里展示的是上次作业五的修改。

（2）上传四个csv文件到hdfs里面的/user/hadoop/input去

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -mkdir -p
/user/hadoop/input
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -put
/home/hadoop/Downloads/user_balance_table.csv
/home/hadoop/Downloads/user_profile_table.csv
/home/hadoop/Downloads/mfd_day_share_interest.csv
/home/hadoop/Downloads/mfd_bank_shibor.csv /user/hadoop/input/
```

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -mkdir -p /user/hadoop/i
nput
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -put /home/hadoop/Downlo
ads/user_balance_table.csv /home/hadoop/Downloads/user_profile_table.csv /home/h
adoop/Downloads/mfd_day_share_interest.csv /home/hadoop/Downloads/mfd_bank_shibo
r.csv /user/hadoop/input/
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/input
Found 6 items
-rw-r--r--   1 hadoop supergroup   52462980 2024-10-23 00:37 /user/hadoop/input/
analyst_ratings.csv
-rw-r--r--   1 hadoop supergroup      19516 2024-10-31 23:00 /user/hadoop/input/
mfd_bank_shibor.csv
-rw-r--r--   1 hadoop supergroup       9760 2024-10-31 23:00 /user/hadoop/input/
mfd_day_share_interest.csv
-rw-r--r--   1 hadoop supergroup       2231 2024-10-23 00:38 /user/hadoop/input/
stop-word-list.txt
-rw-r--r--   1 hadoop supergroup  157761201 2024-10-31 23:00 /user/hadoop/input/
user_balance_table.csv
-rw-r--r--   1 hadoop supergroup     746033 2024-10-31 23:00 /user/hadoop/input/
user_profile_table.csv
hadoop@yawn-virtual-machine:/usr/local/hadoop$
```

上传好了可以用`hdfs dfs -ls /user/hadoop/input`查看当前input路径下有哪些文件。

然后将上一次的作业五传进来的文件删除，整理一下input的内容，再重新查看input里的文件：

可以使用`hdfs dfs -rm 路径/文件`来删除。

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -rm /user/hadoop/input/a
nalyst_ratings.csv
Deleted /user/hadoop/input/analyst_ratings.csv
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -rm /user/hadoop/input/s
top-word-list.txt
Deleted /user/hadoop/input/stop-word-list.txt
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/input
Found 4 items
-rw-r--r--   1 hadoop supergroup      19516 2024-10-31 23:00 /user/hadoop/input/
mfd_bank_shibor.csv
-rw-r--r--   1 hadoop supergroup       9760 2024-10-31 23:00 /user/hadoop/input/
mfd_day_share_interest.csv
-rw-r--r--   1 hadoop supergroup  157761201 2024-10-31 23:00 /user/hadoop/input/
user_balance_table.csv
-rw-r--r--   1 hadoop supergroup     746033 2024-10-31 23:00 /user/hadoop/input/
user_profile_table.csv
```

（3）编写具体代码

在workspace目录下新建一个lab_12的文件夹，新建src文件夹存放java文件。

a）mapper

写在`InflowOutflowMapper.java`文件里面

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class InflowOutflowMapper extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String[] fields = value.toString().split(",");


•    if (fields.length >= 9) {
•        String date = fields[1]; // report_date
•        String totalPurchaseAmt = fields[3].isEmpty() ? "0" : fields[3]; //
total_purchase_amt
•        String totalRedeemAmt = fields[8].isEmpty() ? "0" : fields[8]; //
total_redeem_amt
•
•        context.write(new Text(date + "_purchase"), new Text(totalPurchaseAmt));
•        context.write(new Text(date + "_redeem"), new Text(totalRedeemAmt));
•    }
}

}
```

b）reducer

写在`InflowOutflowReducer.java`文件里面

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

```
import java.io.IOException;

public class InflowOutflowMapper extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String[] fields = value.toString().split(",");


•     if (fields.length >= 9) {
•         String date = fields[1]; // report_date
•         String totalPurchaseAmt = fields[3].isEmpty() ? "0" : fields[3]; //
total_purchase_amt
•         String totalRedeemAmt = fields[8].isEmpty() ? "0" : fields[8]; //
total_redeem_amt
•
•         context.write(new Text(date + "_purchase"), new Text(totalPurchaseAmt));
•         context.write(new Text(date + "_redeem"), new Text(totalRedeemAmt));
•     }
}

}
```

c) driver

写在`InflowOutflowDriver.java`文件里面

```
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class InflowOutflowDriver {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: InflowOutflowDriver <input path> <output
path>");
            System.exit(-1);
        }

•     Configuration conf = new Configuration();
•     Job job = Job.getInstance(conf, "Daily Capital Inflow and Outflow");

•     job.setJarByClass(InflowOutflowDriver.class);
•     job.setMapperClass(InflowOutflowMapper.class);
•     job.setReducerClass(InflowOutflowReducer.class);

•     job.setOutputKeyClass(Text.class);
•     job.setOutputValueClass(Text.class);

•     FileInputFormat.addInputPath(job, new Path(args[0]));
•     FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}

}
```

（4）打包

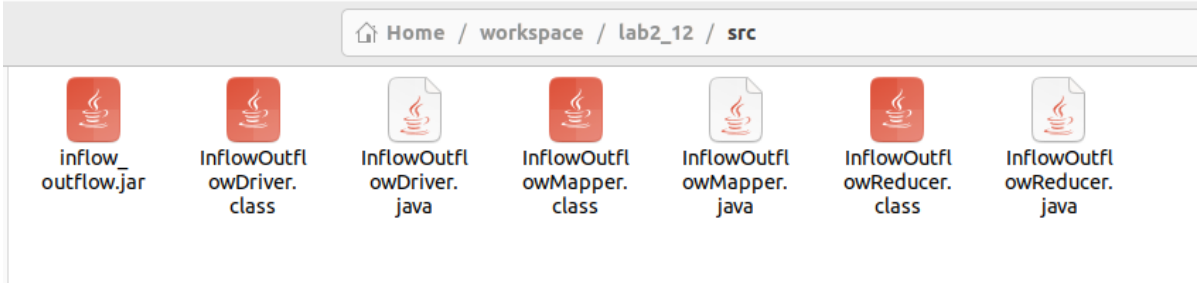
在workspace里的src目录下打开终端，先对该目录下的三个java文件进行编译，生成三个.class文件；然后再将这三个.class文件打包成Hadoop可以运行的jar文件，便于之后运行：

`javac -classpath hadoop classpath -d . InflowOutflowMapper.java InflowOutflowReducer.java InflowOutflowDriver.java`

`jar -cvf inflow_outflow.jar *.class`

```
hadoop@yawn-virtual-machine:~/workspace/lab2_12/src$ javac -classpath `hadoop classpath` -d . InflowOutflowMapper.java InflowOutflowReducer.java InflowOutflowDriver.java
hadoop@yawn-virtual-machine:~/workspace/lab2_12/src$ jar -cvf inflow_outflow.jar *.class
added manifest
adding: InflowOutflowDriver.class(in = 1549) (out= 849)(deflated 45%)
adding: InflowOutflowMapper.class(in = 1856) (out= 780)(deflated 57%)
adding: InflowOutflowReducer.class(in = 3343) (out= 1434)(deflated 57%)
```

打包的结果如下：



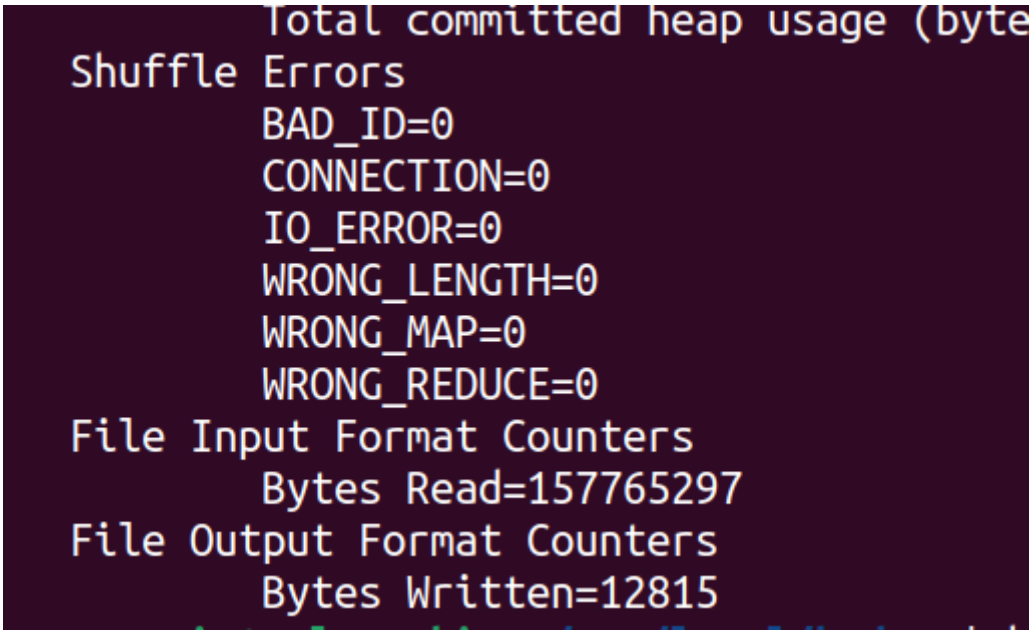
（5）运行

之前已经启动了Hadoop，现在进入到/usr/local/hadoop路径下，**先清除上次的output结果部分**，否则会报错。

```
hdfs dfs -rm -r /user/hadoop/output
```

然后就可以运行指定的jar包了：

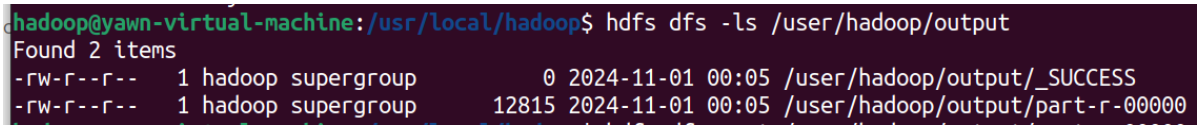
```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hadoop jar /home/hadoop/workspace/lab2_12/src/inflow_outflow.jar InflowOutflowDriver /user/hadoop/input/user_balance_table.csv /user/hadoop/output
```



运行结束，查看output里的内容：

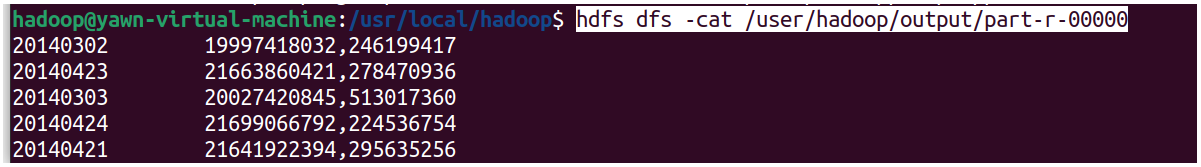
```
hdfs dfs -ls /user/hadoop/output
```

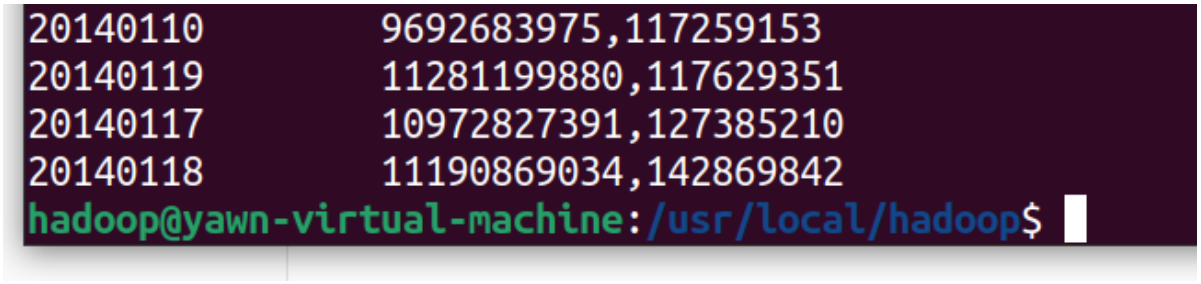
可以看到有一个空的文件success，表示成功运行；另一个part-r-00000就是结果文件：



现在打印结果文件查看内容：

```
hdfs dfs -cat /user/hadoop/output/part-r-00000
```





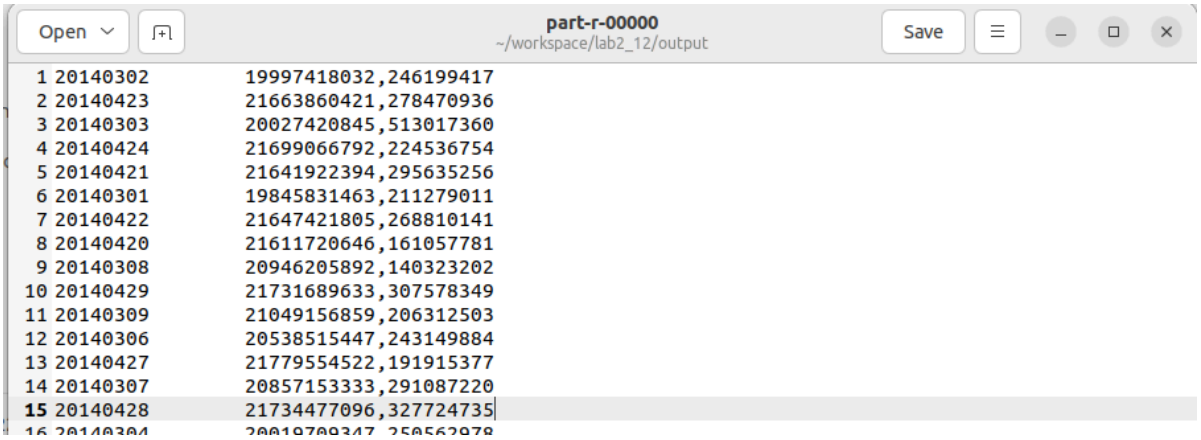
再将结果文件导出：

首先先在lab_12下新建一个output的文件夹，里面存放实验的结果文件。

然后将指定的结果文件抓取到本机：

```
hdfs dfs -get /user/hadoop/output/part-r-00000
/home/hadoop/workspace/lab_12/output
```

part-t-00000的部分内容：



然后删除output里的part-r-00000文件，继续进行下一个部分的实现。

```
hdfs dfs -rm -r /user/hadoop/output
```

（二）任务二：星期交易量统计

1、实验要求

基于任务一的结果，编写MapReduce程序，统计一周七天中每天的平均资金流入与流出情况，并 按照资金流入量从大到小排序。

输出格式： TAB <资金流入量>,<资金流出量>

例如： Sunday 155914552,132427205

2、实验思路

（1）**Mapper**：基于任务一的输出，将每条记录的日期映射为具体的星期几，并将该星期的流入量和流出量作为值输出。

（2）**Reducer**：对每个星期几的记录累加流入量和流出量，同时统计条目数量。计算每周每天的平均流入和流出。

（3）**Driver**：配置排序功能，按照流入量从大到小的顺序输出。

3、实验步骤

（1）将第一个任务生成的part-r-00000上传到hdfs：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -put /home/hadoop/workspace/lab2_12/output/part-r-0
0000 /user/hadoop/input/
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/input
Found 5 items
-rw-r--r--   1 hadoop supergroup      19516 2024-10-31 23:00 /user/hadoop/input/mfd_bank_shibor.csv
-rw-r--r--   1 hadoop supergroup       9760 2024-10-31 23:00 /user/hadoop/input/mfd_day_share_interest.csv
-rw-r--r--   1 hadoop supergroup      12815 2024-11-01 01:15 /user/hadoop/input/part-r-00000
-rw-r--r--   1 hadoop supergroup  157761201 2024-10-31 23:00 /user/hadoop/input/user_balance_table.csv
-rw-r--r--   1 hadoop supergroup     746033 2024-10-31 23:00 /user/hadoop/input/user_profile_table.csv
hadoop@yawn-virtual-machine:/usr/local/hadoop$
```

这是我们任务二的起点文件，所以也要用之前的方法上传到hdfs里面。

（2）编写具体代码

mapper:

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
```

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class WeeklyAvgMapper extends Mapper<LongWritable, Text, Text, Text> {
    private Text dayOfWeek = new Text();
    private SimpleDateFormat sdfInput = new SimpleDateFormat("yyyyMMdd");
    private SimpleDateFormat sdfOutput = new SimpleDateFormat("EEEE"); // 输出星期
几

@Override
protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
    String[] fields = value.toString().split("\t");

•    if (fields.length == 2) {
•        String dateStr = fields[0];
•        String[] amounts = fields[1].split(",");
•
•        if (amounts.length == 2) {
•            try {
•                Date date = sdfInput.parse(dateStr);
•                dayOfWeek.set(sdfOutput.format(date)); // 获取星期几
•                context.write(dayOfWeek, new Text(amounts[0] + "," +
amounts[1]));
•            } catch (Exception e) {
•                System.err.println("Invalid date format: " + dateStr);
•            }
•        }
•    }
}

}
```

reducer:

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class WeeklyAvgReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
        long totalInflow = 0;
        long totalOutflow = 0;
        int count = 0;

•    for (Text value : values) {
•        String[] amounts = value.toString().split(",");
•        if (amounts.length == 2) {
•            totalInflow += Long.parseLong(amounts[0]);
•            totalOutflow += Long.parseLong(amounts[1]);
•            count++;
•        }
•    }
```

```
    long avgInflow = count == 0 ? 0 : totalInflow / count;
    long avgOutflow = count == 0 ? 0 : totalOutflow / count;
    context.write(key, new Text(avgInflow + "," + avgOutflow));
}

}
```

driver:

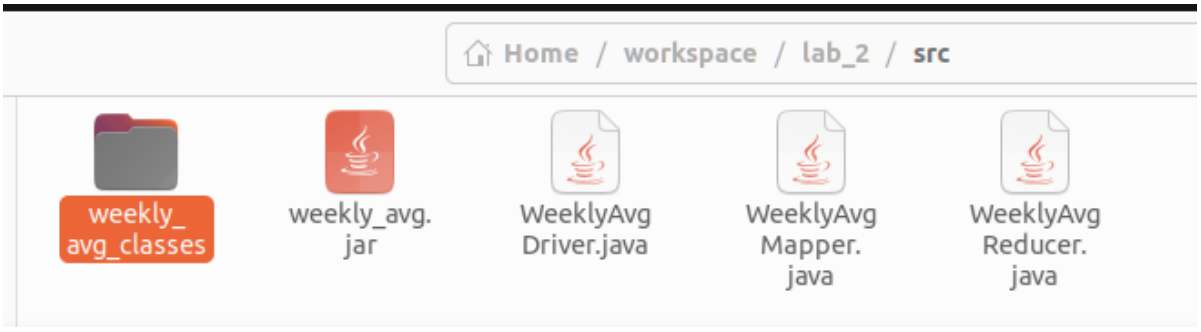
```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WeeklyAvgDriver {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: WeeklyAvgDriver <input path> <output
path>");
            System.exit(-1);
        }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Weekly Average Inflow and Outflow");

    job.setJarByClass(WeeklyAvgDriver.class);
    job.setMapperClass(WeeklyAvgMapper.class);
    job.setReducerClass(WeeklyAvgReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
}

}
```

(3) 打包

进入到src里面，打开终端：

```
#创建编译目录

mkdir -p weekly_avg_classes

#编译源文件

javac -classpath $(hadoop classpath) -d weekly_avg_classes WeeklyAvgMapper.java
WeeklyAvgReducer.java WeeklyAvgDriver.java

#打包为 JAR 文件

jar -cvf weekly_avg.jar -C weekly_avg_classes/ .
```

打包之后的结构：



（4）运行

```
hadoop jar /home/hadoop/workspace/lab_2/src/weekly_avg.jar WeeklyAvgDriver
/user/hadoop/input/part-r-00000 /user/hadoop/output/
```

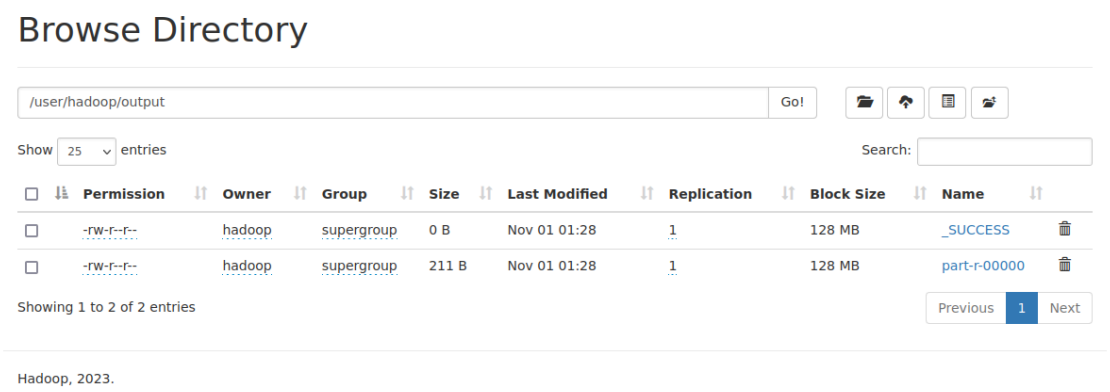
```
        Total committed heap usage (bytes)=763363328
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
        Bytes Read=12815
File Output Format Counters
        Bytes Written=211
```

查看output现在的内容：

```
hdfs dfs -ls /user/hadoop/output
```

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/output
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2024-11-01 01:28 /user/hadoop/output/_SUCCESS
-rw-r--r--   1 hadoop supergroup        211 2024-11-01 01:28 /user/hadoop/output/part-r-00000
```

打开网页可见运行结果：



最后将结果拉取到本地，也是在lab_2里面增加一个output文件夹：

```
hdfs dfs -get /user/hadoop/output/part-r-00000
/home/hadoop/workspace/lab_2/output
```

（三）任务三：用户活跃度分析

1、实验要求

根据 user_balance_table 表中的数据，编写MapReduce程序，统计每个用户的活跃天数，并按 照活跃天数降序排列。 当用户当日有直接购买（ direct_purchase_amt 字段大于0）或赎回行为（ total_redeem_amt 字段大于0）时，则该用户当天活跃。

输出格式： <用户ID> TAB <活跃天数>

例如： 125 24

2、实验思路

通过两个 MapReduce 任务（Job）实现按用户活跃天数降序排序的功能：

（1）**第一个 Job：计算每个用户的活跃天数**

#Mapper：读取用户的交易记录，检查用户的直接购买金额（directPurchaseAmt）或赎回金额（totalRedeemAmt）是否大于 0。如果满足条件，将用户 ID 作为键、1 作为值输出，表示该用户在这一天有活跃记录。
#Reducer：汇总每个用户的活跃天数，即将每个用户的 1 值相加，得到该用户的总活跃天数。输出结果是每个用户的 ID 和其活跃天数。

（2）**第二个 Job：按活跃天数降序排序用户**
#SortMapper：读取第一个任务的输出（用户 ID 和活跃天数），将活跃天数作为键、用户 ID 作为值进行输出，这样可以在后续排序中按活跃天数排序。
•#SortReducer：简单地输出 SortMapper 的结果，因为 SortMapper 的输出已经按照降序排序的要求排列。
#自定义 Comparator：DescendingIntComparator 实现了降序排序（通过返回负数的比较结果），确保活跃天数从高到低排列。

3、实验步骤

（1）上传需要的文件到hdfs里面

因为已经上传过了所以可以直接开始写代码。

（2）编写具体代码

mapper:

```
public static class ActiveDaysMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private Text userId = new Text();
    private final static IntWritable activeDay = new IntWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] fields = line.split(",");

        // 跳过标题行
        if (fields[0].equalsIgnoreCase("user_id")) {
            return;
        }

        if (fields.length > 8) {
            String id = fields[0];
            double directPurchaseAmt;
            double totalRedeemAmt;

            try {
                directPurchaseAmt = Double.parseDouble(fields[7]);
                totalRedeemAmt = Double.parseDouble(fields[9]);
            } catch (NumberFormatException e) {
                return;
            }

            if (directPurchaseAmt > 0 || totalRedeemAmt > 0) {
                userId.set(id);
                context.write(userId, activeDay);
            }
        }
    }
}
```

reducer

```
public static class ActiveDaysReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
        private IntWritable result = new IntWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

自定义mapper输出，实现降序：

```
 // 自定义的 Mapper 输出，以便在排序时使用活跃天数进行排序
    public static class SortMapper extends Mapper<LongWritable, Text,
IntWritable, Text> {
        private IntWritable activeDays = new IntWritable();
        private Text userId = new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String[] fields = value.toString().split("\t");
        if (fields.length == 2) {
            userId.set(fields[0]);
            activeDays.set(Integer.parseInt(fields[1]));
            context.write(activeDays, userId);
        }
    }
}
```

driver

```
public static void main(String[] args) throws Exception {
        // 第一个 Job: 计算每个用户的活跃天数
        Configuration conf1 = new Configuration();
        Job job1 = Job.getInstance(conf1, "user active days count");
        job1.setJarByClass(UserActiveDays.class);
        job1.setMapperClass(ActiveDaysMapper.class);
        job1.setCombinerClass(ActiveDaysReducer.class);
        job1.setReducerClass(ActiveDaysReducer.class);
        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job1, new Path(args[0]));
    Path tempOutput = new Path("temp_output");
    FileOutputFormat.setOutputPath(job1, tempOutput);
```

```
    job1.waitForCompletion(true);

    // 第二个 Job: 根据活跃天数降序排序
    Configuration conf2 = new Configuration();
    Job job2 = Job.getInstance(conf2, "sort by active days");
    job2.setJarByClass(UserActiveDays.class);
    job2.setMapperClass(SortMapper.class);
    job2.setReducerClass(SortReducer.class);
    job2.setSortComparatorClass(DescendingIntComparator.class);
    job2.setOutputKeyClass(IntWritable.class);
    job2.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(job2, tempOutput);
    FileOutputFormat.setOutputPath(job2, new Path(args[1]));

    System.exit(job2.waitForCompletion(true) ? 0 : 1);
}
```

（3）打包

编译生成.class文件：

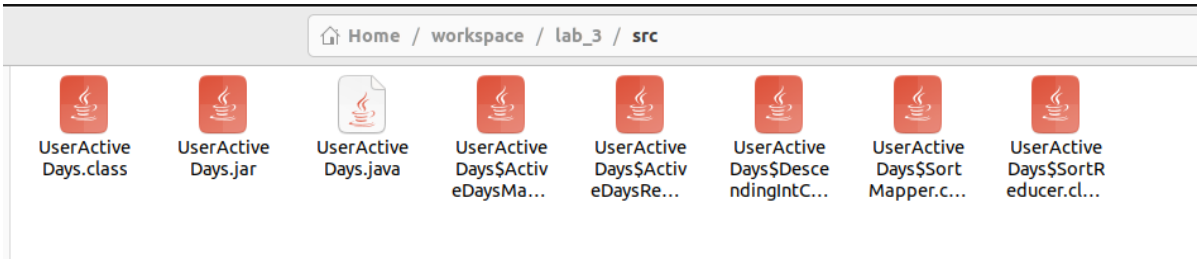
进入到java文件所在的src文件夹下，打开终端：输入

```
javac -classpath `hadoop classpath` -d . UserActiveDays.java
```

打包为jar文件

```
jar -cvf UserActiveDays.jar -C . .
```

打包好之后会出现如下的文件：



（4）运行

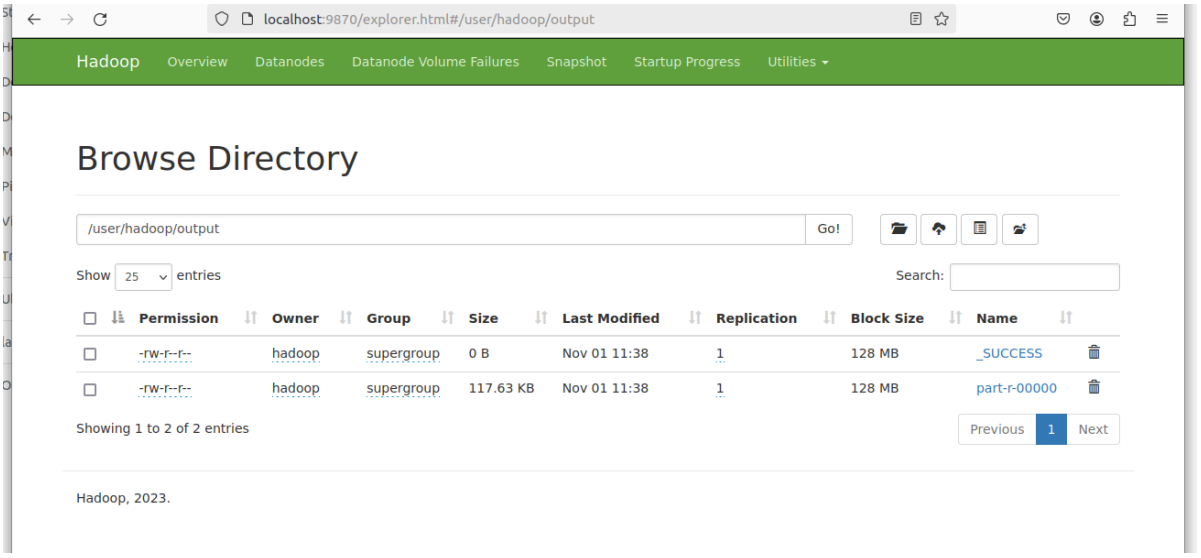
```
hadoop jar /home/hadoop/workspace/lab_3/src/UserActiveDays.jar UserActiveDays
/user/hadoop/input/user_balance_table.csv /user/hadoop/output
```

查看hdfs里的结果：

```
 hdfs dfs -ls /user/hadoop/output
```

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/output
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2024-11-01 11:38 /user/hadoop/output/_SUCCESS
-rw-r--r--   1 hadoop supergroup     120458 2024-11-01 11:38 /user/hadoop/output/part-r-00000
hadoop@yawn-virtual-machine:/usr/local/hadoop$
```

在可视的网页也可以看到本次运行的结果：



再将part-r-00000文件拉取到本地：

首先先在lab_3新建一个output文件夹，用于存放part-r-00000文件。

然后拉取文件到本地：

```
hdfs dfs -get /user/hadoop/output/part-r-00000
/home/hadoop/workspace/lab_3/output
```

可看到文本内容，的确是降序排列输出的：

（四）任务四：交易行为影响因素分析

1、实验要求

用户的交易行为（如：余额宝或银行卡的购买或赎回，用户的消费情况等）受到很多因素的影 响。例如：用户特性（参考用户信息表 user_profile_table ），当前利率（参考支付宝收益率 表 mfd_day_share_interest 以及银行利率表 mfd_bank_shibor ）。 在上面的三个任务中，我们重点研究了 user_balance_table 表中的数据。现在，请你从其他的 表中自行选取研究对象，通过MapReduce（或其他工具），根据统计结果（也即类似于上面三个 任务的结果）阐述某一因素对用户交易行为的影响。

2、实验思路

通过分析 mfd_bank_shibor 表中的 Interest_1_W (一周利率)，将其划分为不同的利率区间，并统计每个区间内的日均资金流入和流出总量，以观察利率与交易资金量之间的关系。

3、实现步骤

（1）布局

我们可以用Hadoop的MapReduce框架实现这个过程。以下是MapReduce的基本流程：

1. Mapper：读取数据，将数据按 Interest_1_W 划分为不同的区间。
2. Reducer：对每个区间内的数据进行汇总，计算每个区间的日均资金流入和流出

（2）编写代码

mapper

```
    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
        String[] fields = value.toString().split(",");
        if (fields.length > 3) { // 假设Interest_1_W在第3列
            try {
                double interest = Double.parseDouble(fields[3]); // 获取一周利率
                double transactionAmount = Double.parseDouble(fields[4]); // 假设
交易金额在第4列

                // 按照利率将数据分区
                if (interest < 1.0) {
                    interestRange.set("0-1%");
                } else if (interest < 2.0) {
                    interestRange.set("1-2%");
                } else if (interest < 3.0) {
                    interestRange.set("2-3%");
                } else {
                    interestRange.set("3%以上");
                }
                context.write(interestRange, new
DoubleWritable(transactionAmount));
            } catch (NumberFormatException e) {
                // 忽略格式错误的行
            }
        }
    }
```

reducer

```
public static class InterestReducer extends Reducer<Text, DoubleWritable, Text,
DoubleWritable> {
        public void reduce(Text key, Iterable<DoubleWritable> values, Context
context) throws IOException, InterruptedException {
            double sum = 0;
            int count = 0;
            for (DoubleWritable val : values) {
                sum += val.get();
                count++;
            }
            double average = sum / count;
            context.write(key, new DoubleWritable(average));
        }
    }
```

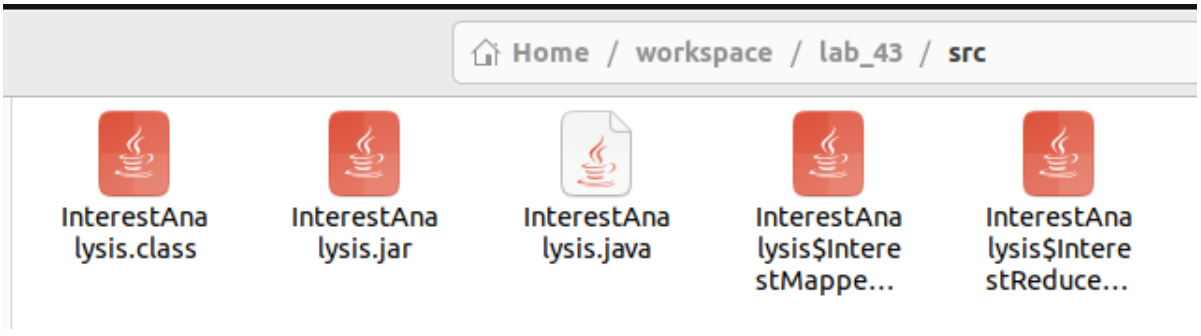
driver

```
 public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Interest Analysis");

    job.setJarByClass(InterestAnalysis.class);
    job.setMapperClass(InterestMapper.class);
    job.setCombinerClass(InterestReducer.class);
    job.setReducerClass(InterestReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(DoubleWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

(3) 打包运行

```
hadoop@yawn-virtual-machine:~/workspace/lab_43/src$ javac -classpath `hadoop classpath` -d . InterestAnalysis.java
hadoop@yawn-virtual-machine:~/workspace/lab_43/src$ jar -cvf InterestAnalysis.jar *.class
added manifest
adding: InterestAnalysis$InterestMapper.class(in = 2029) (out= 910)(deflated 55%)
adding: InterestAnalysis$InterestReducer.class(in = 1711) (out= 720)(deflated 57%)
adding: InterestAnalysis.class(in = 1531) (out= 815)(deflated 46%)
hadoop@yawn-virtual-machine:~/workspace/lab_43/src$
```

打包结束形成jar文件，用于之后的运行：



运行：进入/usr/local/hadoop里面，输入：

```
hadoop jar /home/hadoop/workspace/lab_43/src/InterestAnalysis.jar
InterestAnalysis /user/hadoop/input4/mfd_bank_shibor.csv /user/hadoop/output
```

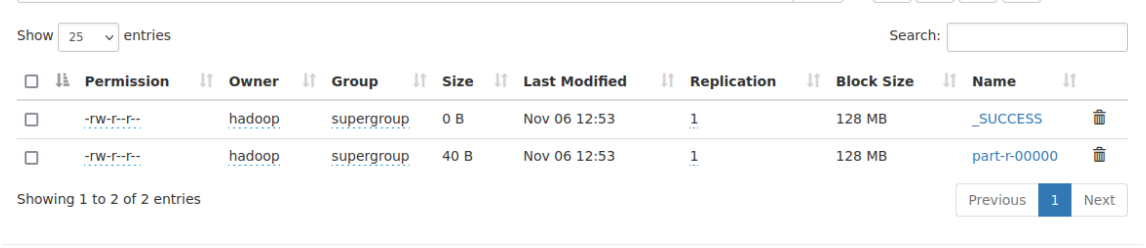
检查output内容：

```
hdfs dfs -ls /user/hadoop/output
```

可看见有一个成功文件success，也有一个输出文件part-r-00000：

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -ls /user/hadoop/output
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2024-11-06 12:53 /user/hadoop/output
/_SUCCESS
-rw-r--r--   1 hadoop supergroup         40 2024-11-06 12:53 /user/hadoop/output
/part-r-00000
```

网页里也可以看到：



将结果拉取至本地：

```
hdfs dfs -get /user/hadoop/output/part-r-00000
/home/hadoop/workspace/lab_43/output
```

（4）结果分析

```
hadoop@yawn-virtual-machine:/usr/local/hadoop$ hdfs dfs -cat /user/hadoop/output
/part-r-00000
2-3%    4.0154
3%以上  5.0426475352112705
```

说明在利率>3%的时候，用户的平均交易金额更高，高于2-3%这个区间，说明用户在利率>3%的时候更倾向于增加交易金额。

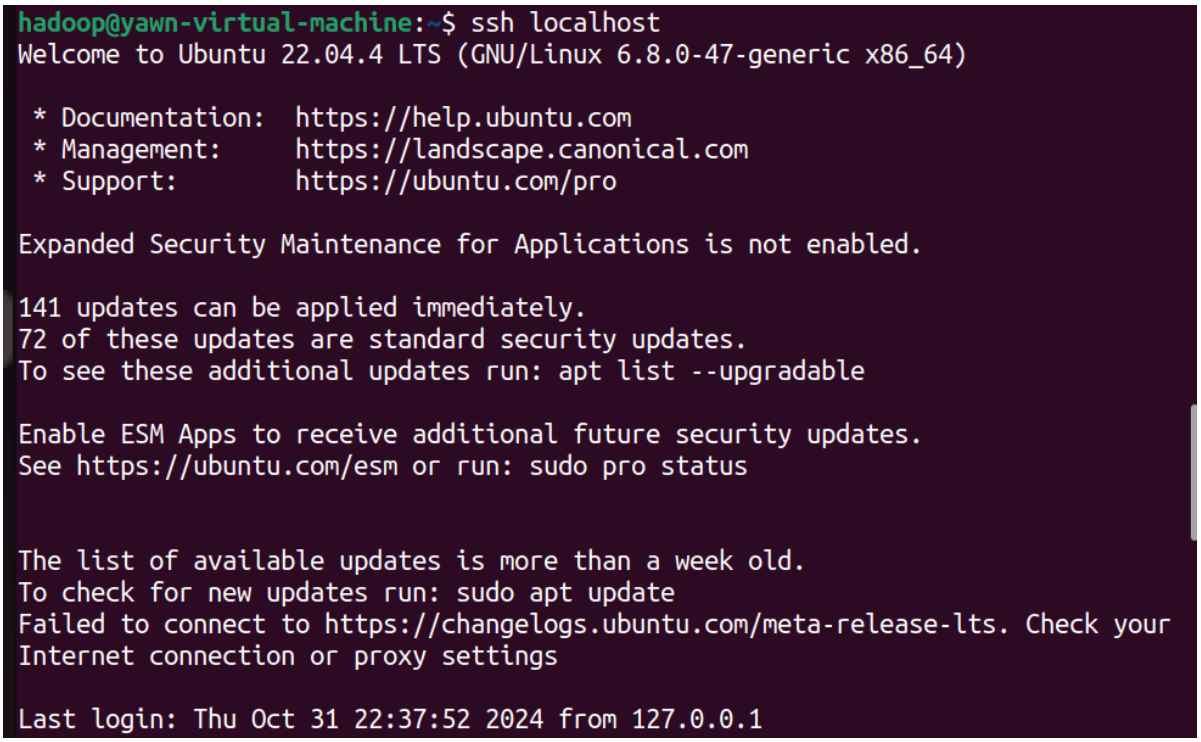
三、实验中遇到的问题

1、在最开始启动伪分布式的Hadoop的时候，会说我的ssh有问题，确实我在输入`ssh localhost`的时候需要让我输入密码，所以就应该是ssh的配置失效了：

```
hadoop@yawn-virtual-machine:~$ cd /usr/local/hadoop
hadoop@yawn-virtual-machine:/usr/local/hadoop$ ./sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: hadoop@localhost: Permission denied (publickey,password).
Starting datanodes
localhost: hadoop@localhost: Permission denied (publickey,password).
Starting secondary namenodes [yawn-virtual-machine]
yawn-virtual-machine: hadoop@yawn-virtual-machine: Permission denied (publickey,
password).
hadoop@yawn-virtual-machine:/usr/local/hadoop$ jps
2793 Jps
```

解决办法：重新配置了一遍ssh，在“~”根目录下输入：

```
ssh-keygen -t rsa -P ""

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

chmod 600 ~/.ssh/authorized_keys

chmod 700 ~/.ssh

ssh localhost
```



四、可以改进的部分

1、任务四可以再细分利率区间，比如以0.5%来划分，可是看是否在更小的利率变化范围内也有类似的趋势。