第九节、MapReduce练习

一、使用实体类型进行操作

1. 原始数据

手机号	上行流量	下行流量
13726230501	200	1100
13396230502	300	1200
13897230503	400	1300
13897230503	100	300
13597230534	500	1400
13597230534	300	1200

2. 目标结果

将同一个用户的上行流量、下行流量进行累加，并总计

3. 实现过程

- Master

```java
public class FlowCountMaster {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        job.setJarByClass(FlowCountMaster.class);
        job.setMapperClass(FlowCountMapper.class);
        job.setReducerClass(FlowCountReducer.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Flow.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Flow.class);
        FileInputFormat.setInputPaths(job, new Path("hdfs://192.168.116.111:8020/input/flow.log"));
        FileOutputFormat.setOutputPath(job, new Path("hdfs://192.168.116.111:8020/output/flowCount"));
        boolean result = job.waitForCompletion(true);
        if (result) {
            System.out.println("Congratulations!");
        }
    }

}
```

- Mapper

```java
public class FlowCountMapper extends Mapper<LongWritable, Text, Text, Flow> {

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Flow>.Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] fields = line.split("\t");
        String phoneNumber = fields[0];
        long upFlow = Long.parseLong(fields[1]);
        long downFlow = Long.parseLong(fields[2]);
        context.write(new Text(phoneNumber), new Flow(upFlow, downFlow));
    }

}
```

- Reducer

```java
public class FlowCountReducer extends Reducer<Text, Flow, Text, Flow> {

    @Override
    protected void reduce(Text key, Iterable<Flow> values, Reducer<Text, Flow, Text, Flow>.Context context)
            throws IOException, InterruptedException {
        long upFlow = 0;
        long downFlow = 0;
        for (Flow bean : values) {
            upFlow += bean.getUpFlow();
            downFlow += bean.getDownFlow();
        }
        Flow result = new Flow(upFlow, downFlow);
        context.write(key, result);
    }

}
```

- Flow

```java
public class Flow implements Writable {

    private long upFlow;
    private long downFlow;
    private long sumFlow;

    public Flow() {

    }

    public Flow(long upFlow, long downFlow) {
        super();
        this.upFlow = upFlow;
        this.downFlow = downFlow;
        this.sumFlow = upFlow + downFlow;
    }

    public long getUpFlow() {
        return upFlow;
    }

    public void setUpFlow(long upFlow) {
        this.upFlow = upFlow;
    }

    public long getDownFlow() {
        return downFlow;
    }

    public void setDownFlow(long downFlow) {
        this.downFlow = downFlow;
    }

    public long getSumFlow() {
        return sumFlow;
    }

    public void setSumFlow(long sumFlow) {
        this.sumFlow = sumFlow;
    }

    public void write(DataOutput output) throws IOException {
        output.writeLong(upFlow);
        output.writeLong(downFlow);
        output.writeLong(sumFlow);
    }

    public void readFields(DataInput input) throws IOException {
        upFlow = input.readLong();
        downFlow = input.readLong();
        sumFlow = input.readLong();
```

```
52.     }
53. 
54.     @Override
55.     public String toString() {
56.         return upFlow + "\t" + downFlow + "\t" + sumFlow;
57.     }
58. 
59. }
```

4. 使用实体作为数据类型

> 实现Writable序列化接口，注意数据写入及输出的顺序，某些运算可写在实体类中