> 知之为知之，不知为不知

Hadoop序列化

序列化概念

> 序列化就是**把内存中的对象，转换成字节序列**（或其他数据传输协议）以便于存储（持久化）和网络传输反序列化就是将收到字节序列（或其他数据传输协议）或者是硬盘的持久化数据，转换成内存中的对象。

序列化作用

> “活的”对象只生存在内存里，关机断电就没有了。而且“活的”对象只能由本地的进程使用，**不能被发送到网络上的另外一台计算机。**然而序列化可以存储“活的”对象，可以将“活的”对象发送到远程计算机

为什么不用Java的序列化

Java的序列化是一个**重量级序列化框架 Serializable**，一个对象被序列化后，会附带很多额外的信息（各种校验信息， Header，继承体系等），不便于在网络中高效传输。所以， Hadoop自己开发了一套序列化机制（ Writable）。【有问题都可以私聊我WX：focusbigdata，或者关注我的公众号：FocusBigData，注意大小写】
Hadoop序列化特点

```
（1）紧凑：高效使用存储空间。
（2）快速：读写数据的额外开销小。
（3）可扩展：随着通信协议的升级而可升级
（4）互操作：支持多语言的交互
```

常用数据的序列化类型

Java类型	Hadoop Writable类型
boolean	BooleanWritable
byte	ByteWritable
int	IntWritable
float	FloatWritable
long	LongWritable
double	DoubleWritable
String	Text
map	MapWritable
array	ArrayWritable

自定义bean对象实现序列化接口步骤

(1) 必须实现Writable接口

(2) 空参构造函数

```
public FlowBean() { super();}
```

(3) 重写序列化方法

```
@Overridepublic void write(DataOutput out) throws IOException {
out.writeLong(upFlow);  out.writeLong(downFlow);    out.writeLong(sumFlow);}
```

(4) 重写反序列化方法

```
@Overridepublic void readFields(DataInput in) throws IOException {  upFlow =
in.readLong(); downFlow = in.readLong();   sumFlow = in.readLong();}
```

(5) 方法顺序一致

> 注意反序列化的顺序和序列化的顺序完全一致

(6) 重写toString

> 要想把结果显示在文件中，需要重写toString()，可用”\t”分开，方便后续用。

(7) 实现Comparable接口

> 如果需要将自定义的bean放在key中传输，则还需要实现Comparable接口，因为MapReduce框中的Shuffle过程要求对key必须能排序。

```
@Overridepublic int compareTo(FlowBean o) { // 倒序排列，从大到小    return
this.sumFlow > o.getSumFlow() ? -1 : 1;}
```

序列化案例实操

需求

> 统计每一个手机号耗费的总上行流量、下行流量、总流量

1	13736230513	192.196.100.1	www.atguigu.com	2481	24681	200
2	13846544121	192.196.100.2		264	0	200
3	13956435636	192.196.100.3		132	1512	200
4	13966251146	192.168.100.1		240	0	404
5	18271575951	192.168.100.2	www.atguigu.com	1527	2106	200
6	84188413	192.168.100.3	www.atguigu.com	4116	1432	200
7	13590439668	192.168.100.4		1116	954	200
8	15910133277	192.168.100.5	www.hao123.com	3156	2936	200
9	13729199489	192.168.100.6		240	0	200

输出数据

```
13560436666          1116              954            2070
手机号码            上行流量        下行流量        总流量
```

需求分析

1、需求：统计每一个手机号耗费的总上行流量、下行流量、总流量

2、输入数据格式

```
7    13560436666   120.196.100.99  1116     954        200
Id   手机号码      网络ip          上行流量 下行流量   网络状态码
```

3、期望输出数据格式

```
13560436666  1116          954          2070
手机号码     上行流量      下行流量     总流量
```

4、Map阶段

（1）读取一行数据，切分字段

```
7    13560436666   120.196.100.99  1116     954        200
```

（2）抽取手机号、上行流量、下行流量

```
13560436666   1116          954
手机号码      上行流量      下行流量
```

（3）以手机号为key，bean对象为value输出，即context.write(手机号,bean);

5、Reduce阶段

（1）累加上行流量和下行流量得到总流量。

```
13560436666  1116     +    954    =    2070
手机号码     上行流量      下行流量     总流量
```

（4）bean对象要想能够传输，必须实现序列化接口

Bean代码

```
// 1 实现writable接口
public class FlowBean implements Writable{

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

    //2  反序列化时，需要反射调用空参构造函数，所以必须有
    public FlowBean() {
        super();
    }

    public FlowBean(long upFlow, long downFlow) {
        super();
        this.upFlow = upFlow;
        this.downFlow = downFlow;
```

```
        this.sumFlow = upFlow + downFlow;
    }

    //3  写序列化方法
    @Override
    public void write(DataOutput out) throws IOException {
        out.writeLong(upFlow);
        out.writeLong(downFlow);
        out.writeLong(sumFlow);
    }

    //4 反序列化方法
    //5 反序列化方法读顺序必须和写序列化方法的写顺序必须一致
    @Override
    public void readFields(DataInput in) throws IOException {
        this.upFlow  = in.readLong();
        this.downFlow = in.readLong();
        this.sumFlow = in.readLong();
    }

    // 6 编写toString方法，方便后续打印到文本
    @Override
    public String toString() {
        return upFlow + "\t" + downFlow + "\t" + sumFlow;
    }

    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;
    }
}
```

Mapper代码

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

    FlowBean v = new FlowBean();
    Text k = new Text();
```

```
    @Override
    protected void map(LongWritable key, Text value, Context context)   throws
IOException, InterruptedException {

        // 1 获取一行
        String line = value.toString();

        // 2 切割字段
        String[] fields = line.split("\t");

        // 3 封装对象
        // 取出手机号码
        String phoneNum = fields[1];
        // 取出上行流量和下行流量
        long upFlow = Long.parseLong(fields[fields.length - 3]);
        long downFlow = Long.parseLong(fields[fields.length - 2]);

        k.set(phoneNum);
        v.set(downFlow, upFlow);

        // 4 写出
        context.write(k, v);
    }
}
```

Reducer代码

```
public class FlowCountReducer extends Reducer<Text, FlowBean, Text, FlowBean> {

    @Override
    protected void reduce(Text key, Iterable<FlowBean> values, Context
context)throws IOException, InterruptedException {

        long sum_upFlow = 0;
        long sum_downFlow = 0;

        // 1 遍历所用bean，将其中的上行流量，下行流量分别累加
        for (FlowBean flowBean : values) {
            sum_upFlow += flowBean.getUpFlow();
            sum_downFlow += flowBean.getDownFlow();
        }

        // 2 封装对象
        FlowBean resultBean = new FlowBean(sum_upFlow, sum_downFlow);

        // 3 写出
        context.write(key, resultBean);
    }
}
```

Driver代码

```
public class FlowsumDriver {
```

```
    public static void main(String[] args) throws IllegalArgumentException,
IOException, ClassNotFoundException, InterruptedException {

// 输入输出路径需要根据自己电脑上实际的输入输出路径设置
args = new String[] { "e:/input/inputflow", "e:/output1" };

        // 1 获取配置信息，或者job对象实例
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);

        // 6 指定本程序的jar包所在的本地路径
        job.setJarByClass(FlowsumDriver.class);

        // 2 指定本业务job要使用的mapper/Reducer业务类
        job.setMapperClass(FlowCountMapper.class);
        job.setReducerClass(FlowCountReducer.class);

        // 3 指定mapper输出数据的kv类型
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(FlowBean.class);

        // 4 指定最终输出的数据的kv类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FlowBean.class);

        // 5 指定job的输入原始文件所在目录
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // 7 将job中配置的相关参数，以及job所用的java类所在的jar包， 提交给yarn去运行
        boolean result = job.waitForCompletion(true);
        System.exit(result ? 0 : 1);
    }
}
```