MapReduce 排序和序列化

- 序列化 (Serialization) 是指把结构化对象转化为字节流
- 反序列化 (Deserialization) 是序列化的逆过程. 把字节流转为结构化对象. 当要在进程间传递对象或持久化对象的时候, 就需要序列化对象成字节流, 反之当要将接收到或从磁盘读取的字节流转换为对象, 就要进行反序列化
- Java 的序列化 (Serializable) 是一个重量级序列化框架, 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), 精简高效. 不用像 Java 对象类一样传输多层的父子关系, 需要哪个属性就传输哪个属性值, 大大的减少网络传输的开销
- Writable 是 Hadoop 的序列化格式, Hadoop 定义了这样一个 Writable 接口. 一个类要支持可序列化只需实现这个接口即可
- 另外 Writable 有一个子接口是 WritableComparable, WritableComparable 是既可实现序列化, 也可以对key进行比较, 我们这里可以通过自定义 Key 实现 WritableComparable 来实现我们的排序功能

数据格式如下

```
1   a   1
2   a   9
3   b   3
4   a   7
5   b   8
6   b   10
7   a   5
```

要求:

- 第一列按照字典顺序进行排列
- 第一列相同的时候, 第二列按照升序进行排列

解决思路:

- 将 Map 端输出的 `<key,value>` 中的 key 和 value 组合成一个新的 key (newKey), value值不变
- 这里就变成 `<(key,value),value>`, 在针对 newKey 排序的时候, 如果 key 相同, 就再对value进行排序

Step 1. 自定义类型和比较器

```java
public class PairWritable implements WritableComparable<PairWritable> {
    // 组合key,第一部分是我们第一列，第二部分是我们第二列
    private String first;
    private int second;
    public PairWritable() {
    }
    public PairWritable(String first, int second) {
        this.set(first, second);
    }
    /**
     * 方便设置字段
     */
    public void set(String first, int second) {
        this.first = first;
        this.second = second;
    }
    /**
     * 反序列化
     */
    @Override
    public void readFields(DataInput input) throws IOException {
        this.first = input.readUTF();
        this.second = input.readInt();
    }
    /**
     * 序列化
     */
    @Override
    public void write(DataOutput output) throws IOException {
        output.writeUTF(first);
        output.writeInt(second);
    }
    /*
     * 重写比较器
     */
    public int compareTo(PairWritable o) {
        //每次比较都是调用该方法的对象与传递的参数进行比较，说白了就是第一行与第二行比较完了之后的结果与第三行比较，
        //得出来的结果再去与第四行比较，依次类推
        System.out.println(o.toString());
        System.out.println(this.toString());
        int comp = this.first.compareTo(o.first);

        if (comp != 0) {
```

```
            return comp;
        } else { // 若第一个字段相等，则比较第二个字段
            return Integer.valueOf(this.second).compareTo(
                    Integer.valueOf(o.getSecond()));
        }
    }

    public int getSecond() {
        return second;
    }

    public void setSecond(int second) {
        this.second = second;
    }
    public String getFirst() {
        return first;
    }
    public void setFirst(String first) {
        this.first = first;
    }
    @Override
    public String toString() {
        return "PairWritable{" +
                "first='" + first + '\'' +
                ", second=" + second +
                '}';
    }
}
```

Step 2. Mapper

```
public class SortMapper extends
Mapper<LongWritable,Text,PairWritable,IntWritable> {

    private PairWritable mapOutKey = new PairWritable();
    private IntWritable mapOutValue = new IntWritable();

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

        //设置组合key和value ==> <(key,value),value>
```

```
            mapOutKey.set(strs[0], Integer.valueOf(strs[1]));
            mapOutValue.set(Integer.valueOf(strs[1]));
            context.write(mapOutKey, mapOutValue);
        }
    }
```

Step 3. Reducer

```
public class SortReducer extends
Reducer<PairWritable,IntWritable,Text,IntWritable> {

    private Text outPutKey = new Text();
    @Override
    public void reduce(PairWritable key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
//迭代输出
        for(IntWritable value : values) {
            outPutKey.set(key.getFirst());
            context.write(outPutKey, value);
        }
    }
}
```

Step 4. Main 入口

```
public class JobMain extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception {
        //1:创建job对象
        Job job = Job.getInstance(super.getConf(), "mapreduce_sort");

        //2:配置job任务(八个步骤)
            //第一步:设置输入类和输入的路径
            job.setInputFormatClass(TextInputFormat.class);
            ///TextInputFormat.addInputPath(job, new
Path("hdfs://node01:8020/input/sort_input"));
            TextInputFormat.addInputPath(job, new
Path("file:///D:\\input\\sort_input"));

            //第二步: 设置Mapper类和数据类型
            job.setMapperClass(SortMapper.class);
            job.setMapOutputKeyClass(SortBean.class);
            job.setMapOutputValueClass(NullWritable.class);
```

```
17
18              //第三，四，五，六
19
20              //第七步：设置Reducer类和类型
21              job.setReducerClass(SortReducer.class);
22              job.setOutputKeyClass(SortBean.class);
23              job.setOutputValueClass(NullWritable.class);
24
25
26              //第八步：设置输出类和输出的路径
27              job.setOutputFormatClass(TextOutputFormat.class);
28              //TextOutputFormat.setOutputPath(job, new
    Path("hdfs://node01:8020/out/sort_out"));
29              TextOutputFormat.setOutputPath(job, new
    Path("file:///D:\\out\\sort_out"));
30
31
32          //3:等待任务结束
33          boolean bl = job.waitForCompletion(true);
34
35          return bl?0:1;
36      }
37
38      public static void main(String[] args) throws Exception {
39          Configuration configuration = new Configuration();
40
41          //启动job任务
42          int run = ToolRunner.run(configuration, new JobMain(), args);
43
44          System.exit(run);
45      }
46  }
```

规约Combiner

概念

每一个 map 都可能会产生大量的本地输出，Combiner 的作用就是对 map 端的输出先做一次合并，以减少在 map 和 reduce 节点之间的数据传输量，以提高网络IO 性能，是 MapReduce 的一种优化手段之一

- combiner 是 MR 程序中 Mapper 和 Reducer 之外的一种组件

- combiner 组件的父类就是 Reducer
- combiner 和 reducer 的区别在于运行的位置
  - Combiner 是在每一个 maptask 所在的节点运行
  - Reducer 是接收全局所有 Mapper 的输出结果
- combiner 的意义就是对每一个 maptask 的输出进行局部汇总，以减小网络传输量

实现步骤

1. 自定义一个 combiner 继承 Reducer，重写 reduce 方法
2. 在 job 中设置 `job.setCombinerClass(CustomCombiner.class)`

combiner 能够应用的前提是不能影响最终的业务逻辑，而且，combiner 的输出 kv 应该跟 reducer 的输入 kv 类型要对应起来

MapReduce案例-流量统计

需求一: 统计求和

统计每个手机号的上行数据包总和，下行数据包总和，上行总流量之和，下行总流量之和 分析：以手机号码作为key值，上行流量，下行流量，上行总流量，下行总流量四个字段作为value值，然后以这个key，和value作为map阶段的输出，reduce阶段的输入

Step 1: 自定义map的输出value对象FlowBean

```
public class FlowBean implements Writable {
    private Integer upFlow;
    private Integer  downFlow;
    private Integer upCountFlow;
    private Integer downCountFlow;
    @Override
    public void write(DataOutput out) throws IOException {
        out.writeInt(upFlow);
        out.writeInt(downFlow);
        out.writeInt(upCountFlow);
        out.writeInt(downCountFlow);
    }
    @Override
```

```
14      public void readFields(DataInput in) throws IOException {
15          this.upFlow = in.readInt();
16          this.downFlow = in.readInt();
17          this.upCountFlow = in.readInt();
18          this.downCountFlow = in.readInt();
19      }
20      public FlowBean() {
21      }
22      public FlowBean(Integer upFlow, Integer downFlow, Integer
    upCountFlow, Integer downCountFlow) {
23          this.upFlow = upFlow;
24          this.downFlow = downFlow;
25          this.upCountFlow = upCountFlow;
26          this.downCountFlow = downCountFlow;
27      }
28      public Integer getUpFlow() {
29          return upFlow;
30      }
31      public void setUpFlow(Integer upFlow) {
32          this.upFlow = upFlow;
33      }
34      public Integer getDownFlow() {
35          return downFlow;
36      }
37      public void setDownFlow(Integer downFlow) {
38          this.downFlow = downFlow;
39      }
40      public Integer getUpCountFlow() {
41          return upCountFlow;
42      }
43      public void setUpCountFlow(Integer upCountFlow) {
44          this.upCountFlow = upCountFlow;
45      }
46      public Integer getDownCountFlow() {
47          return downCountFlow;
48      }
49      public void setDownCountFlow(Integer downCountFlow) {
50          this.downCountFlow = downCountFlow;
51      }
52      @Override
53      public String toString() {
54          return "FlowBean{" +
55                  "upFlow=" + upFlow +
56                  ", downFlow=" + downFlow +
```

```
                ", upCountFlow=" + upCountFlow +
                ", downCountFlow=" + downCountFlow +
                '}';
    }
}
```

Step 2: 定义FlowMapper类

```
public class FlowCountMapper extends
Mapper<LongWritable,Text,Text,FlowBean> {
    @Override
    protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        //1:拆分手机号
         String[] split = value.toString().split("\t");
         String phoneNum = split[1];
         //2:获取四个流量字段
         FlowBean flowBean = new FlowBean();
         flowBean.setUpFlow(Integer.parseInt(split[6]));
         flowBean.setDownFlow(Integer.parseInt(split[7]));
         flowBean.setUpCountFlow(Integer.parseInt(split[8]));
         flowBean.setDownCountFlow(Integer.parseInt(split[9]));

         //3:将k2和v2写入上下文中
         context.write(new Text(phoneNum), flowBean);
    }
}
```

Step 3: 定义FlowReducer类

```
public class FlowCountReducer extends
Reducer<Text,FlowBean,Text,FlowBean> {
    @Override
    protected void reduce(Text key, Iterable<FlowBean> values, Context
context) throws IOException, InterruptedException {
        //封装新的FlowBean
         FlowBean flowBean = new FlowBean();
         Integer upFlow = 0;
         Integer  downFlow = 0;
         Integer upCountFlow = 0;
         Integer downCountFlow = 0;
         for (FlowBean value : values) {
             upFlow  += value.getUpFlow();
```

```
            downFlow += value.getDownFlow();
            upCountFlow += value.getUpCountFlow();
            downCountFlow += value.getDownCountFlow();
        }
        flowBean.setUpFlow(upFlow);
        flowBean.setDownFlow(downFlow);
        flowBean.setUpCountFlow(upCountFlow);
        flowBean.setDownCountFlow(downCountFlow);
        //将K3和V3写入上下文中
        context.write(key, flowBean);
    }
}

```

Step 4: 程序main函数入口FlowMain

```
public class JobMain extends Configured implements Tool {

    //该方法用于指定一个job任务
    @Override
        public int run(String[] args) throws Exception {
        //1:创建一个job任务对象
        Job job = Job.getInstance(super.getConf(),
"mapreduce_flowcount");
        //如果打包运行出错，则需要加该配置
        job.setJarByClass(JobMain.class);
        //2:配置job任务对象(八个步骤)

        //第一步:指定文件的读取方式和读取路径
        job.setInputFormatClass(TextInputFormat.class);
        //TextInputFormat.addInputPath(job, new
Path("hdfs://node01:8020/wordcount"));
        TextInputFormat.addInputPath(job, new
Path("file:///D:\\input\\flowcount_input"));


        //第二步:指定Map阶段的处理方式和数据类型
         job.setMapperClass(FlowCountMapper.class);
         //设置Map阶段K2的类型
          job.setMapOutputKeyClass(Text.class);
        //设置Map阶段V2的类型

          job.setMapOutputValueClass(FlowBean.class);
```

```
25
26
27              //第三（分区），四 （排序）
28              //第五步：规约(Combiner)
29              //第六步 分组
30
31
32              //第七步：指定Reduce阶段的处理方式和数据类型
33              job.setReducerClass(FlowCountReducer.class);
34              //设置K3的类型
35               job.setOutputKeyClass(Text.class);
36              //设置V3的类型
37               job.setOutputValueClass(FlowBean.class);
38
39               //第八步：设置输出类型
40               job.setOutputFormatClass(TextOutputFormat.class);
41               //设置输出的路径
42               TextOutputFormat.setOutputPath(job, new
    Path("file:///D:\\out\\flowcount_out"));
43
44
45
46          //等待任务结束
47              boolean bl = job.waitForCompletion(true);
48
49              return bl ? 0:1;
50      }
51
52      public static void main(String[] args) throws Exception {
53          Configuration configuration = new Configuration();
54
55          //启动job任务
56          int run = ToolRunner.run(configuration, new JobMain(), args);
57          System.exit(run);
58
59      }
60  }
```

需求二: 上行流量倒序排序（递减排序）

分析，以需求一的输出数据作为排序的输入数据，自定义FlowBean,以FlowBean为map输出的key，以手机号作为Map输出的value，因为MapReduce程序会对Map阶段输出的key进行排序

Step 1: 定义FlowBean实现WritableComparable实现比较排序

Java 的 compareTo 方法说明:

- compareTo 方法用于将当前对象与方法的参数进行比较。
- 如果指定的数与参数相等返回 0。
- 如果指定的数小于参数返回 -1。
- 如果指定的数大于参数返回 1。

例如：`o1.compareTo(o2);` 返回正数的话，当前对象（调用 compareTo 方法的对象 o1）要排在比较对象（compareTo 传参对象 o2）后面，返回负数的话，放在前面

```
1  public class FlowBean implements WritableComparable<FlowBean> {
2      private Integer upFlow;
3      private Integer  downFlow;
4      private Integer upCountFlow;
5      private Integer downCountFlow;
6      public FlowBean() {
7      }
8
9      public FlowBean(Integer upFlow, Integer downFlow, Integer
   upCountFlow, Integer downCountFlow) {
10         this.upFlow = upFlow;
11         this.downFlow = downFlow;
12         this.upCountFlow = upCountFlow;
13         this.downCountFlow = downCountFlow;
14     }
15
16     @Override
17     public void write(DataOutput out) throws IOException {
18         out.writeInt(upFlow);
19         out.writeInt(downFlow);
20         out.writeInt(upCountFlow);
21         out.writeInt(downCountFlow);
22     }
23
24     @Override
25     public void readFields(DataInput in) throws IOException {
26         upFlow = in.readInt();
27         downFlow = in.readInt();
28         upCountFlow = in.readInt();
29         downCountFlow = in.readInt();
30     }
31
```

```
32      public Integer getUpFlow() {
33          return upFlow;
34      }
35
36      public void setUpFlow(Integer upFlow) {
37          this.upFlow = upFlow;
38      }
39
40      public Integer getDownFlow() {
41          return downFlow;
42      }
43
44      public void setDownFlow(Integer downFlow) {
45          this.downFlow = downFlow;
46      }
47
48      public Integer getUpCountFlow() {
49          return upCountFlow;
50      }
51      public void setUpCountFlow(Integer upCountFlow) {
52          this.upCountFlow = upCountFlow;
53      }
54      public Integer getDownCountFlow() {
55          return downCountFlow;
56      }
57      public void setDownCountFlow(Integer downCountFlow) {
58          this.downCountFlow = downCountFlow;
59      }
60      @Override
61      public String toString() {
62          return upFlow+"\t"+downFlow+"\t"+upCountFlow+"\t"+downCountFlow;
63      }
64      @Override
65      public int compareTo(FlowBean o) {
66          return this.upCountFlow > o.upCountFlow ?-1:1;
67      }
68  }
```

Step 2: 定义FlowMapper

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

2       @Override
```

```
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        FlowBean flowBean = new FlowBean();
        String[] split = value.toString().split("\t");

        //获取手机号，作为V2
        String phoneNum = split[0];
        //获取其他流量字段,封装flowBean，作为K2
        flowBean.setUpFlow(Integer.parseInt(split[1]));
        flowBean.setDownFlow(Integer.parseInt(split[2]));
        flowBean.setUpCountFlow(Integer.parseInt(split[3]));
        flowBean.setDownCountFlow(Integer.parseInt(split[4]));

        //将K2和V2写入上下文中
        context.write(flowBean, new Text(phoneNum));

    }
}

```

Step 3: 定义FlowReducer

```
public class FlowCountSortReducer extends Reducer<FlowBean,Text,Text,FlowBean> {
    @Override
    protected void reduce(FlowBean key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        for (Text value : values) {
            context.write(value, key);
        }
    }
}
```

Step 4: 程序main函数入口

```
public class JobMain extends Configured  implements Tool {
    @Override
    public int run(String[] strings) throws Exception {
        //创建一个任务对象
        Job job = Job.getInstance(super.getConf(), "mapreduce_flowcountsort");

        //打包放在集群运行时，需要做一个配置
```

```
        job.setJarByClass(JobMain.class);
        //第一步:设置读取文件的类: K1 和V1
        job.setInputFormatClass(TextInputFormat.class);
        TextInputFormat.addInputPath(job, new
Path("hdfs://node01:8020/out/flowcount_out"));

        //第二步: 设置Mapper类
        job.setMapperClass(FlowCountSortMapper.class);
        //设置Map阶段的输出类型: k2 和V2的类型
        job.setMapOutputKeyClass(FlowBean.class);
        job.setMapOutputValueClass(Text.class);

        //第三,四, 五, 六步采用默认方式(分区, 排序, 规约, 分组)


        //第七步 : 设置文的Reducer类
        job.setReducerClass(FlowCountSortReducer.class);
        //设置Reduce阶段的输出类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FlowBean.class);

        //设置Reduce的个数

        //第八步:设置输出类
        job.setOutputFormatClass(TextOutputFormat.class);
        //设置输出的路径
        TextOutputFormat.setOutputPath(job, new
Path("hdfs://node01:8020/out/flowcountsort_out"));


        boolean b = job.waitForCompletion(true);
        return b?0:1;

    }
    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();

        //启动一个任务
        int run = ToolRunner.run(configuration, new JobMain(), args);
        System.exit(run);
    }

}
```

需求三: 手机号码分区

在需求一的基础上，继续完善，将不同的手机号分到不同的数据文件的当中去，需要自定义分区来实现，这里我们自定义来模拟分区，将以下数字开头的手机号进行分开

```
135 开头数据到一个分区文件
136 开头数据到一个分区文件
137 开头数据到一个分区文件
其他分区
```

自定义分区

```
public class FlowPartition extends Partitioner<Text,FlowBean> {
    @Override
    public int getPartition(Text text, FlowBean flowBean, int i) {
        String line = text.toString();
        if (line.startsWith("135")){
            return 0;
        }else if(line.startsWith("136")){
            return 1;
        }else if(line.startsWith("137")){
            return 2;
        }else{
            return 3;
        }
    }
}
```

作业运行设置

```
job.setPartitionerClass(FlowPartition.class);
 job.setNumReduceTasks(4);
```

修改输入输出路径, 并放入集群运行

```
TextInputFormat.addInputPath(job,new
Path("hdfs://node01:8020/partition_flow/"));
TextOutputFormat.setOutputPath(job,new
Path("hdfs://node01:8020/partition_out"));
```