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

```
public class SortBean implements WritableComparable<SortBean>{

    private String word;
    private int  num;

    public String getWord() {
        return word;
    }

    public void setWord(String word) {
        this.word = word;
    }

    public int getNum() {
        return num;
    }

    public void setNum(int num) {
        this.num = num;
    }

    @Override
    public String toString() {
        return   word + "\t"+ num ;
    }

    //实现比较器，指定排序的规则
    /*
      规则:
        第一列(word)按照字典顺序进行排列     //   aac    aad
        第一列相同的时候，第二列(num)按照升序进行排列
     */
    /*
        a  1
        a  5
        b  3
        b  8
     */
    @Override
    public int compareTo(SortBean sortBean) {
        //先对第一列排序: Word排序
        int result = this.word.compareTo(sortBean.word);

        //如果第一列相同，则按照第二列进行排序
```

```
44          if(result == 0){
45              return  this.num - sortBean.num;
46          }
47          return result;
48      }
49
50      //实现序列化
51      @Override
52      public void write(DataOutput out) throws IOException {
53          out.writeUTF(word);
54          out.writeInt(num);
55      }
56
57      //实现反序列
58      @Override
59      public void readFields(DataInput in) throws IOException {
60              this.word = in.readUTF();
61              this.num = in.readInt();
62      }
63  }
```

Step 2. Mapper

```
1
2   public class SortMapper extends
    Mapper<LongWritable,Text,SortBean,NullWritable> {
3       /*
4         map方法将K1和V1转为K2和V2：
5
6         K1            V1
7         0             a  3
8         5             b  7
9         ----------------------
10        K2                        V2
11        SortBean(a  3)         NullWritable
12        SortBean(b  7)         NullWritable
13       */
14      @Override
15      protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
16          //1:将行文本数据(V1)拆分，并将数据封装到SortBean对象,就可以得到K2
17          String[] split = value.toString().split("\t");
18
```

```
19          SortBean sortBean = new SortBean();
20          sortBean.setWord(split[0]);
21          sortBean.setNum(Integer.parseInt(split[1]));
22
23          //2:将K2和V2写入上下文中
24          context.write(sortBean, NullWritable.get());
25      }
26  }
```

Step 3. Reducer

```
1   public class SortReducer extends
    Reducer<SortBean,NullWritable,SortBean,NullWritable> {
2
3       //reduce方法将新的K2和V2转为K3和V3
4       @Override
5       protected void reduce(SortBean key, Iterable<NullWritable> values,
    Context context) throws IOException, InterruptedException {
6           context.write(key, NullWritable.get());
7       }
8   }
9
```

Step 4. Main 入口

```
1   public class JobMain extends Configured implements Tool {
2       @Override
3       public int run(String[] args) throws Exception {
4           //1:创建job对象
5           Job job = Job.getInstance(super.getConf(), "mapreduce_sort");
6
7           //2:配置job任务(八个步骤)
8               //第一步:设置输入类和输入的路径
9               job.setInputFormatClass(TextInputFormat.class);
10              ///TextInputFormat.addInputPath(job, new
    Path("hdfs://node01:8020/input/sort_input"));
11              TextInputFormat.addInputPath(job, new
    Path("file:///D:\\input\\sort_input"));
12
13              //第二步: 设置Mapper类和数据类型
14              job.setMapperClass(SortMapper.class);
15              job.setMapOutputKeyClass(SortBean.class);
16              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("file:///D:\\out\\sort_out"));
29
30
31          //3:等待任务结束
32          boolean bl = job.waitForCompletion(true);
33
34          return bl?0:1;
35      }
36
37      public static void main(String[] args) throws Exception {
38          Configuration configuration = new Configuration();
39
40          //启动job任务
41          int run = ToolRunner.run(configuration, new JobMain(), args);
42
43          System.exit(run);
44      }
45  }
```

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;  //上行数据包数
    private Integer downFlow;  //下行数据包数
    private Integer upCountFlow; //上行流量总和
    private Integer downCountFlow;//下行流量总和

    public Integer getUpFlow() {
        return upFlow;
    }

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

```

```
15      public Integer getDownFlow() {
16          return downFlow;
17      }
18
19      public void setDownFlow(Integer downFlow) {
20          this.downFlow = downFlow;
21      }
22
23      public Integer getUpCountFlow() {
24          return upCountFlow;
25      }
26
27      public void setUpCountFlow(Integer upCountFlow) {
28          this.upCountFlow = upCountFlow;
29      }
30
31      public Integer getDownCountFlow() {
32          return downCountFlow;
33      }
34
35      public void setDownCountFlow(Integer downCountFlow) {
36          this.downCountFlow = downCountFlow;
37      }
38
39      @Override
40      public String toString() {
41          return  upFlow +
42                  "\t" + downFlow +
43                  "\t" + upCountFlow +
44                  "\t" + downCountFlow;
45      }
46
47      //序列化方法
48      @Override
49      public void write(DataOutput out) throws IOException {
50          out.writeInt(upFlow);
51          out.writeInt(downFlow);
52          out.writeInt(upCountFlow);
53          out.writeInt(downCountFlow);
54      }
55
56      //反序列化
57      @Override
58      public void readFields(DataInput in) throws IOException {
```

```
59          this.upFlow = in.readInt();
60          this.downFlow = in.readInt();
61          this.upCountFlow = in.readInt();
62          this.downCountFlow = in.readInt();
63      }
64  }
```

Step 2: 定义FlowMapper类

```
1   public class FlowCountMapper extends
    Mapper<LongWritable,Text,Text,FlowBean> {
2       /*
3         将K1和V1转为K2和V2:
4         K1               V1
5         0            1360021750219    128 1177    16852   200
6        -------------------------------
7         K2               V2
8         13600217502      FlowBean(19   128 1177    16852)
9        */
10      @Override
11      protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
12          //1:拆分行文本数据,得到手机号--->K2
13          String[] split = value.toString().split("\t");
14          String phoneNum = split[1];
15
16          //2:创建FlowBean对象,并从行文本数据拆分出流量的四个四段,并将四个流量字段的值
    赋给FlowBean对象
17          FlowBean flowBean = new FlowBean();
18
19          flowBean.setUpFlow(Integer.parseInt(split[6]));
20          flowBean.setDownFlow(Integer.parseInt(split[7]));
21          flowBean.setUpCountFlow(Integer.parseInt(split[8]));
22          flowBean.setDownCountFlow(Integer.parseInt(split[9]));
23
24          //3:将K2和V2写入上下文中
25          context.write(new Text(phoneNum), flowBean);
26
27      }
28  }
29
```

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 {
        //1:遍历集合,并将集合中的对应的四个字段累计
         Integer upFlow = 0;  //上行数据包数
         Integer downFlow = 0;  //下行数据包数
         Integer upCountFlow = 0; //上行流量总和
         Integer downCountFlow = 0;//下行流量总和

        for (FlowBean value : values) {
            upFlow += value.getUpFlow();
            downFlow += value.getDownFlow();
            upCountFlow += value.getUpCountFlow();
            downCountFlow += value.getDownCountFlow();
        }

        //2:创建FlowBean对象,并给对象赋值  V3
        FlowBean flowBean = new FlowBean();
        flowBean.setUpFlow(upFlow);
        flowBean.setDownFlow(downFlow);
        flowBean.setUpCountFlow(upCountFlow);
        flowBean.setDownCountFlow(downCountFlow);

        //3:将K3和V3下入上下文中
        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)
```

```
11
12          //第一步:指定文件的读取方式和读取路径
13          job.setInputFormatClass(TextInputFormat.class);
14          //TextInputFormat.addInputPath(job, new
    Path("hdfs://node01:8020/wordcount"));
15          TextInputFormat.addInputPath(job, new
    Path("file:///D:\\input\\flowcount_input"));
16
17
18
19          //第二步:指定Map阶段的处理方式和数据类型
20           job.setMapperClass(FlowCountMapper.class);
21           //设置Map阶段K2的类型
22            job.setMapOutputKeyClass(Text.class);
23          //设置Map阶段V2的类型
24            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;  //上行数据包数
3       private Integer downFlow;  //下行数据包数
4       private Integer upCountFlow; //上行流量总和
5       private Integer downCountFlow;//下行流量总和
6
7       public Integer getUpFlow() {
8           return upFlow;
9       }
10
11      public void setUpFlow(Integer upFlow) {
12          this.upFlow = upFlow;
13      }
14
15      public Integer getDownFlow() {
16          return downFlow;
```

```
17      }
18
19      public void setDownFlow(Integer downFlow) {
20          this.downFlow = downFlow;
21      }
22
23      public Integer getUpCountFlow() {
24          return upCountFlow;
25      }
26
27      public void setUpCountFlow(Integer upCountFlow) {
28          this.upCountFlow = upCountFlow;
29      }
30
31      public Integer getDownCountFlow() {
32          return downCountFlow;
33      }
34
35      public void setDownCountFlow(Integer downCountFlow) {
36          this.downCountFlow = downCountFlow;
37      }
38
39      @Override
40      public String toString() {
41          return  upFlow +
42                  "\t" + downFlow +
43                  "\t" + upCountFlow +
44                  "\t" + downCountFlow;
45      }
46
47      //序列化方法
48      @Override
49      public void write(DataOutput out) throws IOException {
50          out.writeInt(upFlow);
51          out.writeInt(downFlow);
52          out.writeInt(upCountFlow);
53          out.writeInt(downCountFlow);
54      }
55
56      //反序列化
57      @Override
58      public void readFields(DataInput in) throws IOException {
59          this.upFlow = in.readInt();
60          this.downFlow = in.readInt();
```

```
61          this.upCountFlow = in.readInt();
62          this.downCountFlow = in.readInt();
63      }
64
65      //指定排序的规则
66      @Override
67      public int compareTo(FlowBean flowBean) {
68         // return this.upFlow.compareTo(flowBean.getUpFlow()) * -1;
69         return  flowBean.upFlow - this.upFlow ;
70      }
71  }
```

Step 2: 定义FlowMapper

```
1   public class FlowSortMapper extends
    Mapper<LongWritable,Text,FlowBean,Text> {
2       //map方法:将K1和V1转为K2和V2
3       @Override
4       protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
5           //1:拆分行文本数据(V1),得到四个流量字段,并封装FlowBean对象---->K2
6           String[] split = value.toString().split("\t");
7
8           FlowBean flowBean = new FlowBean();
9
10          flowBean.setUpFlow(Integer.parseInt(split[1]));
11          flowBean.setDownFlow(Integer.parseInt(split[2]));
12          flowBean.setUpCountFlow(Integer.parseInt(split[3]));
13          flowBean.setDownCountFlow(Integer.parseInt(split[4]));
14
15          //2:通过行文本数据,得到手机号--->V2
16          String phoneNum = split[0];
17
18          //3:将K2和V2下入上下文中
19          context.write(flowBean, new Text(phoneNum));
20
21      }
22  }
```

Step 3: 定义FlowReducer

```
1   /*
2     K2: FlowBean
```

```
  V2: Text  手机号

  K3: Text  手机号
  V3: FlowBean
 */

public class FlowSortReducer extends Reducer<FlowBean,Text,Text,FlowBean>
{
    @Override
    protected void reduce(FlowBean key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
        //1:遍历集合,取出 K3,并将K3和V3写入上下文中
        for (Text value : values) {
            context.write(value, key);
        }

    }
}

```

Step 4: 程序main函数入口

```
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_flowsort");


        //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:\\out\\flowcount_out"));


        //第二步:指定Map阶段的处理方式和数据类型
```

```
20            job.setMapperClass(FlowSortMapper.class);
21            //设置Map阶段K2的类型
22             job.setMapOutputKeyClass(FlowBean.class);
23           //设置Map阶段V2的类型
24             job.setMapOutputValueClass(Text.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(FlowSortReducer.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\\flowsort_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 FlowCountPartition extends Partitioner<Text,FlowBean> {

    /*
      该方法用来指定分区的规则:
        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

      参数:
        text : K2   手机号
        flowBean: V2
        i    : ReduceTask的个数
     */
    @Override
    public int getPartition(Text text, FlowBean flowBean, int i) {
        //1:获取手机号
        String phoneNum = text.toString();

        //2:判断手机号以什么开头,返回对应的分区编号(0-3)
        if(phoneNum.startsWith("135")){
            return  0;
        }else  if(phoneNum.startsWith("136")){
            return  1;
        }else  if(phoneNum.startsWith("137")){
            return  2;
        }else{
            return 3;
        }


    }
```

```
32    }
```

作业运行设置

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

修改输入输出路径，并运行

```
1   TextInputFormat.addInputPath(job, new
    Path("file:///D:\\input\\flowpartition_input"));
2   TextOutputFormat.setOutputPath(job, new
    Path("file:///D:\\out\\flowpartiton_out"));
```