1. 自定义InputFormat合并小文件

1.1 需求

无论hdfs还是mapreduce，对于小文件都有损效率，实践中，又难免面临处理大量小文件的场景，此时，就需要有相应解决方案

1.2 分析

小文件的优化无非以下几种方式：

1、 在数据采集的时候，就将小文件或小批数据合成大文件再上传HDFS

2、 在业务处理之前，在HDFS上使用mapreduce程序对小文件进行合并

3、 在mapreduce处理时，可采用combineInputFormat提高效率

1.3 实现

本节实现的是上述第二种方式

程序的核心机制：

自定义一个InputFormat

改写RecordReader，实现一次读取一个完整文件封装为KV

在输出时使用SequenceFileOutPutFormat输出合并文件

代码如下：

自定义InputFromat

```
1  public class MyInputFormat extends
   FileInputFormat<NullWritable,BytesWritable> {
2      @Override
3      public RecordReader<NullWritable, BytesWritable>
   createRecordReader(InputSplit inputSplit, TaskAttemptContext
   taskAttemptContext) throws IOException, InterruptedException {
4          //1:创建自定义RecordReader对象
5          MyRecordReader myRecordReader = new MyRecordReader();
6          //2:将inputSplit和context对象传给MyRecordReader
```

```
        myRecordReader.initialize(inputSplit, taskAttemptContext);


        return myRecordReader;
    }

    /*
     设置文件是否可以被切割
     */
    @Override
    protected boolean isSplitable(JobContext context, Path filename) {
        return false;
    }
}

```

自定义RecordReader

```
public class MyRecordReader extends
RecordReader<NullWritable,BytesWritable>{

    private Configuration configuration = null;
    private  FileSplit fileSplit = null;
    private boolean processed = false;
    private BytesWritable bytesWritable = new BytesWritable();
    private  FileSystem fileSystem = null;
    private  FSDataInputStream inputStream = null;
    //进行初始化工作
    @Override
    public void initialize(InputSplit inputSplit, TaskAttemptContext
taskAttemptContext) throws IOException, InterruptedException {
        //获取文件的切片
          fileSplit= (FileSplit)inputSplit;

        //获取Configuration对象
         configuration = taskAttemptContext.getConfiguration();
    }

    //该方法用于获取K1和V1
    /*
     K1: NullWritable
     V1: BytesWritable

     */
```

```
24      @Override
25      public boolean nextKeyValue() throws IOException,
    InterruptedException {
26          if(!processed){
27              //1:获取源文件的字节输入流
28              //1.1 获取源文件的文件系统 (FileSystem)
29               fileSystem = FileSystem.get(configuration);
30              //1.2 通过FileSystem获取文件字节输入流
31               inputStream = fileSystem.open(fileSplit.getPath());
32
33              //2:读取源文件数据到普通的字节数组(byte[])
34              byte[] bytes = new byte[(int) fileSplit.getLength()];
35              IOUtils.readFully(inputStream, bytes, 0,
    (int)fileSplit.getLength());
36
37              //3:将字节数组中数据封装到BytesWritable ,得到v1
38
39              bytesWritable.set(bytes, 0, (int)fileSplit.getLength());
40
41              processed = true;
42
43              return true;
44          }
45
46          return false;
47      }
48
49      //返回K1
50      @Override
51      public NullWritable getCurrentKey() throws IOException,
    InterruptedException {
52          return NullWritable.get();
53      }
54
55      //返回V1
56      @Override
57      public BytesWritable getCurrentValue() throws IOException,
    InterruptedException {
58          return bytesWritable;
59      }
60
61      //获取文件读取的进度
62      @Override
63      public float getProgress() throws IOException, InterruptedException {
```

```
64          return 0;
65      }
66
67      //进行资源释放
68      @Override
69      public void close() throws IOException {
70          inputStream.close();
71          fileSystem.close();
72      }
73  }
74
```

Mapper类:

```
1   public class SequenceFileMapper extends
    Mapper<NullWritable,BytesWritable,Text,BytesWritable> {
2       @Override
3       protected void map(NullWritable key, BytesWritable value, Context
    context) throws IOException, InterruptedException {
4           //1:获取文件的名字,作为K2
5           FileSplit fileSplit = (FileSplit) context.getInputSplit();
6           String fileName = fileSplit.getPath().getName();
7
8           //2:将K2和V2写入上下文中
9           context.write(new Text(fileName), value);
10      }
11  }
```

主类:

```
1   public class JobMain extends Configured implements Tool {
2       @Override
3       public int run(String[] args) throws Exception {
4           //1:获取job对象
5           Job job = Job.getInstance(super.getConf(), "sequence_file_job");
6
7           //2:设置job任务
8               //第一步:设置输入类和输入的路径
9               job.setInputFormatClass(MyInputFormat.class);


10              MyInputFormat.addInputPath(job, new
```

```
    Path("file:///D:\\input\\myInputformat_input"));

            //第二步:设置Mapper类和数据类型
            job.setMapperClass(SequenceFileMapper.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(BytesWritable.class);

            //第七步: 不需要设置Reducer类,但是必须设置数据类型
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(BytesWritable.class);

            //第八步:设置输出类和输出的路径
            job.setOutputFormatClass(SequenceFileOutputFormat.class);
            SequenceFileOutputFormat.setOutputPath(job, new
    Path("file:///D:\\out\\myinputformat_out"));


        //3:等待job任务执行结束
        boolean bl = job.waitForCompletion(true);
        return bl ? 0 : 1;
    }

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

        int run = ToolRunner.run(configuration, new JobMain(), args);

        System.exit(run);
    }
}
```

2. 自定义outputFormat

2.1 需求

现在有一些订单的评论数据，需求，将订单的好评与差评进行区分开来，将最终的数据分开到不同的文件夹下面去，数据内容参见资料文件夹，其中数据第九个字段表示好评，中评，差评。0：好评，1：中评，2：差评

2.2 分析

程序的关键点是要在一个mapreduce程序中根据数据的不同输出两类结果到不同目录，这类灵活的输出需求可以通过自定义outputformat来实现

2.3 实现

实现要点:

1、 在mapreduce中访问外部资源

2、 自定义outputformat，改写其中的recordwriter，改写具体输出数据的方法write()

第一步：自定义MyOutputFormat

MyOutputFormat类:

```
public class MyOutputFormat extends FileOutputFormat<Text,NullWritable> {
    @Override
    public RecordWriter<Text, NullWritable>
getRecordWriter(TaskAttemptContext taskAttemptContext) throws
IOException, InterruptedException {
        //1:获取目标文件的输出流(两个)
        FileSystem fileSystem =
FileSystem.get(taskAttemptContext.getConfiguration());
        FSDataOutputStream goodCommentsOutputStream =
fileSystem.create(new
Path("file:///D:\\out\\good_comments\\good_comments.txt"));
        FSDataOutputStream badCommentsOutputStream =
fileSystem.create(new
Path("file:///D:\\out\\bad_comments\\bad_comments.txt"));

        //2:将输出流传给MyRecordWriter
        MyRecordWriter myRecordWriter = new
MyRecordWriter(goodCommentsOutputStream,badCommentsOutputStream);

        return myRecordWriter;
    }
}

```

MyRecordReader类:

```
public class MyRecordWriter extends RecordWriter<Text,NullWritable> {
```

```
    private FSDataOutputStream goodCommentsOutputStream;
    private FSDataOutputStream badCommentsOutputStream;

    public MyRecordWriter() {
    }

    public MyRecordWriter(FSDataOutputStream goodCommentsOutputStream, FSDataOutputStream badCommentsOutputStream) {
        this.goodCommentsOutputStream = goodCommentsOutputStream;
        this.badCommentsOutputStream = badCommentsOutputStream;
    }

    /**
     *
     * @param text  行文本内容
     * @param nullWritable
     * @throws IOException
     * @throws InterruptedException
     */
    @Override
    public void write(Text text, NullWritable nullWritable) throws IOException, InterruptedException {
        //1:从行文本数据中获取第9个字段
        String[] split = text.toString().split("\t");
        String numStr = split[9];

        //2:根据字段的值,判断评论的类型,然后将对应的数据写入不同的文件夹文件中
        if(Integer.parseInt(numStr) <= 1){
            //好评或者中评
            goodCommentsOutputStream.write(text.toString().getBytes());
            goodCommentsOutputStream.write("\r\n".getBytes());
        }else{
            //差评
            badCommentsOutputStream.write(text.toString().getBytes());
            badCommentsOutputStream.write("\r\n".getBytes());
        }

    }

    @Override
    public void close(TaskAttemptContext taskAttemptContext) throws IOException, InterruptedException {
        IOUtils.closeStream(goodCommentsOutputStream);
        IOUtils.closeStream(badCommentsOutputStream);
```

```
43      }
44  }
```

第二步：自定义Mapper类

```
1   public class MyOutputFormatMapper extends
    Mapper<LongWritable,Text,Text,NullWritable> {
2       @Override
3       protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
4           context.write(value, NullWritable.get());
5       }
6   }
```

第三步:主类JobMain

```
1   public class JobMain extends Configured implements Tool {
2       @Override
3       public int run(String[] args) throws Exception {
4           //1:获取job对象
5           Job job = Job.getInstance(super.getConf(), "myoutputformat_job");
6
7           //2:设置job任务
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("file:///D:\\input\\myoutputformat_input"));
11
12              //第二步:设置Mapper类和数据类型
13              job.setMapperClass(MyOutputFormatMapper.class);
14              job.setMapOutputKeyClass(Text.class);
15              job.setMapOutputValueClass(NullWritable.class);
16
17              //第八步:设置输出类和输出的路径
18              job.setOutputFormatClass(MyOutputFormat.class);
19              MyOutputFormat.setOutputPath(job, new
    Path("file:///D:\\out\\myoutputformat_out"));
20
21
22          //3:等待任务结束
23          boolean bl = job.waitForCompletion(true);
24          return bl ? 0 : 1;
25      }
```

```
26
27      public static void main(String[] args) throws Exception {
28          Configuration configuration = new Configuration();
29          int run = ToolRunner.run(configuration, new JobMain(), args);
30          System.exit(run);
31      }
32  }
```

3. 自定义分组求取topN

分组是mapreduce当中reduce端的一个功能组件，主要的作用是决定哪些数据作为一组，调用一次reduce的逻辑，默认是每个不同的key，作为多个不同的组，每个组调用一次reduce逻辑，我们可以自定义分组实现不同的key作为同一个组，调用一次reduce逻辑

3.1 需求

有如下订单数据

订单id	商品id	成交金额
Order_0000001	Pdt_01	222.8
Order_0000001	Pdt_05	25.8
Order_0000002	Pdt_03	522.8
Order_0000002	Pdt_04	122.4
Order_0000002	Pdt_05	722.4
Order_0000003	Pdt_01	222.8

现在需要求出每一个订单中成交金额最大的一笔交易

3.2 分析

1、利用“订单id和成交金额”作为key，可以将map阶段读取到的所有订单数据按照id分区，按照金额排序，发送到reduce

2、在reduce端利用分组将订单id相同的kv聚合成组，然后取第一个即是最大值

3.3 实现

第一步:定义OrderBean

定义一个OrderBean，里面定义两个字段，第一个字段是我们的orderId，第二个字段是我们的金额（注意金额一定要使用Double或者DoubleWritable类型，否则没法按照金额顺序排序）

```
public class OrderBean  implements WritableComparable<OrderBean>{
    private  String orderId;
    private  Double price;

    public String getOrderId() {
        return orderId;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return  orderId + "\t" + price;
    }

    //指定排序规则
    @Override
    public int compareTo(OrderBean orderBean) {
        //先比较订单ID,如果订单ID一致,则排序订单金额(降序)
        int i = this.orderId.compareTo(orderBean.orderId);
        if(i == 0){
            i = this.price.compareTo(orderBean.price) * -1;
        }

}
```

```
35          return i;
36      }
37
38      //实现对象的序列化
39      @Override
40      public void write(DataOutput out) throws IOException {
41           out.writeUTF(orderId);
42           out.writeDouble(price);
43      }
44
45      //实现对象反序列化
46      @Override
47      public void readFields(DataInput in) throws IOException {
48          this.orderId = in.readUTF();
49          this.price  = in.readDouble();
50      }
51  }
```

第二步: 定义Mapper类

```
1
2   public class GroupMapper extends Mapper<LongWritable,Text,OrderBean,Text>
    {
3       @Override
4       protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
5           //1:拆分行文本数据,得到订单的ID,订单的金额
6           String[] split = value.toString().split("\t");
7
8           //2:封装OrderBean,得到K2
9           OrderBean orderBean = new OrderBean();
10          orderBean.setOrderId(split[0]);
11          orderBean.setPrice(Double.valueOf(split[2]));
12
13          //3:将K2和V2写入上下文中
14          context.write(orderBean, value);
15      }
16  }
```

第三步:自定义分区

自定义分区，按照订单id进行分区，把所有订单id相同的数据，都发送到同一个reduce中去

```

public class OrderPartition extends Partitioner<OrderBean,Text> {
    //分区规则: 根据订单的ID实现分区

    /**
     *
     * @param orderBean K2
     * @param text  V2
     * @param i  ReduceTask个数
     * @return 返回分区的编号
     */
    @Override
    public int getPartition(OrderBean orderBean, Text text, int i) {
        return (orderBean.getOrderId().hashCode() & 2147483647) % i;
    }
}

```

第四步:自定义分组

按照我们自己的逻辑进行分组，通过比较相同的订单id，将相同的订单id放到一个组里面去，进过分组之后当中的数据，已经全部是排好序的数据，我们只需要取前topN即可

```
// 1: 继承WriteableComparator
public class OrderGroupComparator extends WritableComparator {
    // 2: 调用父类的有参构造
    public OrderGroupComparator() {
        super(OrderBean.class,true);
    }

    //3: 指定分组的规则(重写方法)
    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        //3.1 对形参做强制类型转换
        OrderBean first = (OrderBean)a;
        OrderBean second = (OrderBean)b;

        //3.2 指定分组规则
        return first.getOrderId().compareTo(second.getOrderId());
    }

}
```

第五步:定义Reducer类

```
public class GroupReducer extends
Reducer<OrderBean,Text,Text,NullWritable> {
    @Override
    protected void reduce(OrderBean key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
        int i = 0;
        //获取集合中的前N条数据
        for (Text value : values) {
            context.write(value, NullWritable.get());
            i++;
            if(i >= 1){
                break;
            }
        }
    }
}
```

第六步:程序main函数入口

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

        //2:设置job任务
            //第一步:设置输入类和输入路径
            job.setInputFormatClass(TextInputFormat.class);
            TextInputFormat.addInputPath(job, new
Path("file:///D:\\input\\mygroup_input"));

            //第二步:设置Mapper类和数据类型
            job.setMapperClass(GroupMapper.class);
            job.setMapOutputKeyClass(OrderBean.class);
            job.setMapOutputValueClass(Text.class);

            //第三,四,五,六
            //设置分区
```

```
19              job.setPartitionerClass(OrderPartition.class);
20              //设置分组
21              job.setGroupingComparatorClass(OrderGroupComparator.class);
22
23              //第七步:设置Reducer类和数据类型
24              job.setReducerClass(GroupReducer.class);
25              job.setOutputKeyClass(Text.class);
26              job.setOutputValueClass(NullWritable.class);
27
28              //第八步:设置输出类和输出的路径
29              job.setOutputFormatClass(TextOutputFormat.class);
30              TextOutputFormat.setOutputPath(job, new
    Path("file:///D:\\out\\mygroup_out"));
31
32          //3:等待job任务结束
33          boolean bl = job.waitForCompletion(true);
34
35
36
37          return bl ? 0: 1;
38      }
39
40      public static void main(String[] args) throws Exception {
41          Configuration configuration = new Configuration();
42
43          //启动job任务
44          int run = ToolRunner.run(configuration, new JobMain(), args);
45
46          System.exit(run);
47      }
48  }
49
```