课后作业：案例——找爷孙关系

2018年8月1日 12:59

案例文件：

褚英 努尔哈赤
皇太极 努尔哈赤
多尔衮 努尔哈赤
多铎 努尔哈赤
豪格 皇太极
福临(顺治) 皇太极
福全 福临(顺治)
玄烨(康熙) 福临(顺治)

第一列是孩子辈，第二列是父母辈。现在要得到爷孙辈的关系

比如最后的输出结果：

爷爷辈:[努尔哈赤]-->孙子辈:[福临(顺治), 豪格]
爷爷辈:[皇太极]-->孙子辈:[玄烨(康熙), 福全]

YesunMapper代码：

```
public class YesunMapper extends Mapper<LongWritable,Text,Text,Text>{

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
    Text>.Context context)
            throws IOException, InterruptedException {

        String line=value.toString();
        String s1=line.split(" ")[0];
        String s2=line.split(" ")[1];
        context.write(new Text(s1), new Text("+" + s2));
        context.write(new Text(s2), new Text("-" + s1));

    }
}
```

YesunReducer代码：

```
public class YesunReducer extends Reducer<Text,Text,Text,NullWritable>{

	@Override
	protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text,
	NullWritable>.Context context)
			throws IOException, InterruptedException {
		ArrayList<String> grandpaList=new ArrayList<>();
		ArrayList<String> grandsonList=new ArrayList<>();

		for(Text value:values){
			if(value.toString().startsWith("+")){
				grandpaList.add(value.toString().substring(1));
			}else{
				grandsonList.add(value.toString().substring(1));
			}
		}
		if(grandpaList.size()>0&&grandsonList.size()>0){
			String grandpa=grandpaList.toString();
			String grandson=grandsonList.toString();

			String relation="爷爷辈:"+grandpa+"-->孙子辈:"+grandson;

			context.write(new Text(relation),NullWritable.get());
		}


	}
}
```

Driver代码：

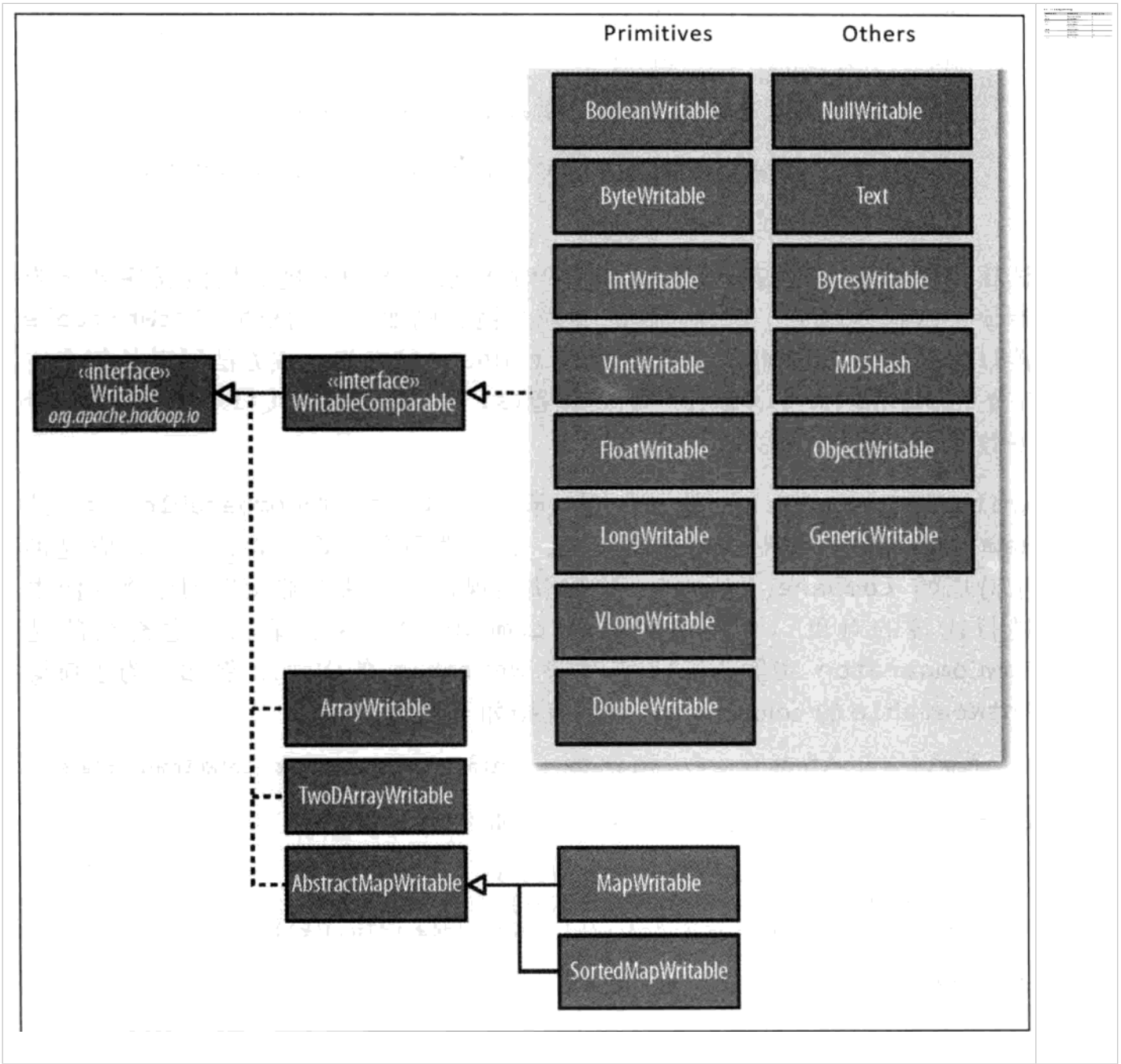
```
public class Driver {

	public static void main(String[] args) throws Exception {
		Configuration conf=new Configuration();
		Job job=Job.getInstance(conf);

		job.setJarByClass(Driver.class);
```

```
        job.setMapperClass(YesunMapper.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setReducerClass(YesunReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);

        FileInputFormat.addInputPath(job, new Path("hdfs://192.168.150.137:9000/yesun"));

        FileOutputFormat.setOutputPath(job,new
        Path("hdfs://192.168.150.137:9000/yesun/result"));

        job.waitForCompletion(true);

    }


}
```

序列化机制+FlowCount案例

由于集群工作过程中，需要用到RPC操作，所以MR处理的对象必须可以进行序列化/反序列操作。Hadoop利用的是avro实现的序列化和反序列，并且在其基础上提供了便捷的API
要序列化的对象必要实现相关的接口：
Writable接口--WritableComparable

Hadoop提供的Writable类



常用的类型：

LongWritable

IntWritable

Text

NullWritbale

ByteWritable

DoubleWritable

以上这些类型，之所以能够被序列化，就是因为都实现了Writable接口

zs [phone:12312 addr:bj name:zs flow:2321]

LongWritable源码举例：

```
public class LongWritable implements WritableComparable<LongWritable>
@Override //反序列化
  public void readFields(DataInput in) throws IOException {
    value = in.readLong();
  }

  @Override //序列化
  public void write(DataOutput out) throws IOException {
    out.writeLong(value);
  }
```

案例：通过统计流量

文件：

```
12321445 zs bj 343
12321312 ww sh 234
……
```

Pojo代码：

```
public class FlowBean implements Writable{

    private String phone;
    private String name;
    private long flow;
    private String addr;

    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```
        this.name = name;
    }
    public long getFlow() {
        return flow;
    }
    public void setFlow(long flow) {
        this.flow = flow;
    }
    public String getAddr() {
        return addr;
    }
    public void setAddr(String addr) {
        this.addr = addr;
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(phone);
        out.writeUTF(name);
        out.writeUTF(addr);
        out.writeLong(flow);

    }
    /*要和write的顺序保持一致
     */
    @Override
    public void readFields(DataInput in) throws IOException {
        this.phone=in.readUTF();
        this.name=in.readUTF();
        this.addr=in.readUTF();
        this.flow=in.readLong();


    }


}
```

Mapper代码：

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

    public void map(LongWritable ikey, Text ivalue, Context context) throws
    IOException, InterruptedException {
        String line=ivalue.toString();
        String[] info=line.split(" ");
        FlowBean flowBean=new FlowBean();
        String phone=info[0];
        flowBean.setPhone(info[0]);
        flowBean.setAddr(info[1]);
        flowBean.setName(info[2]);
        flowBean.setFlow(Long.parseLong(info[3]));
        context.write(new Text(phone), flowBean);
    }

}
```

Reducer代码：

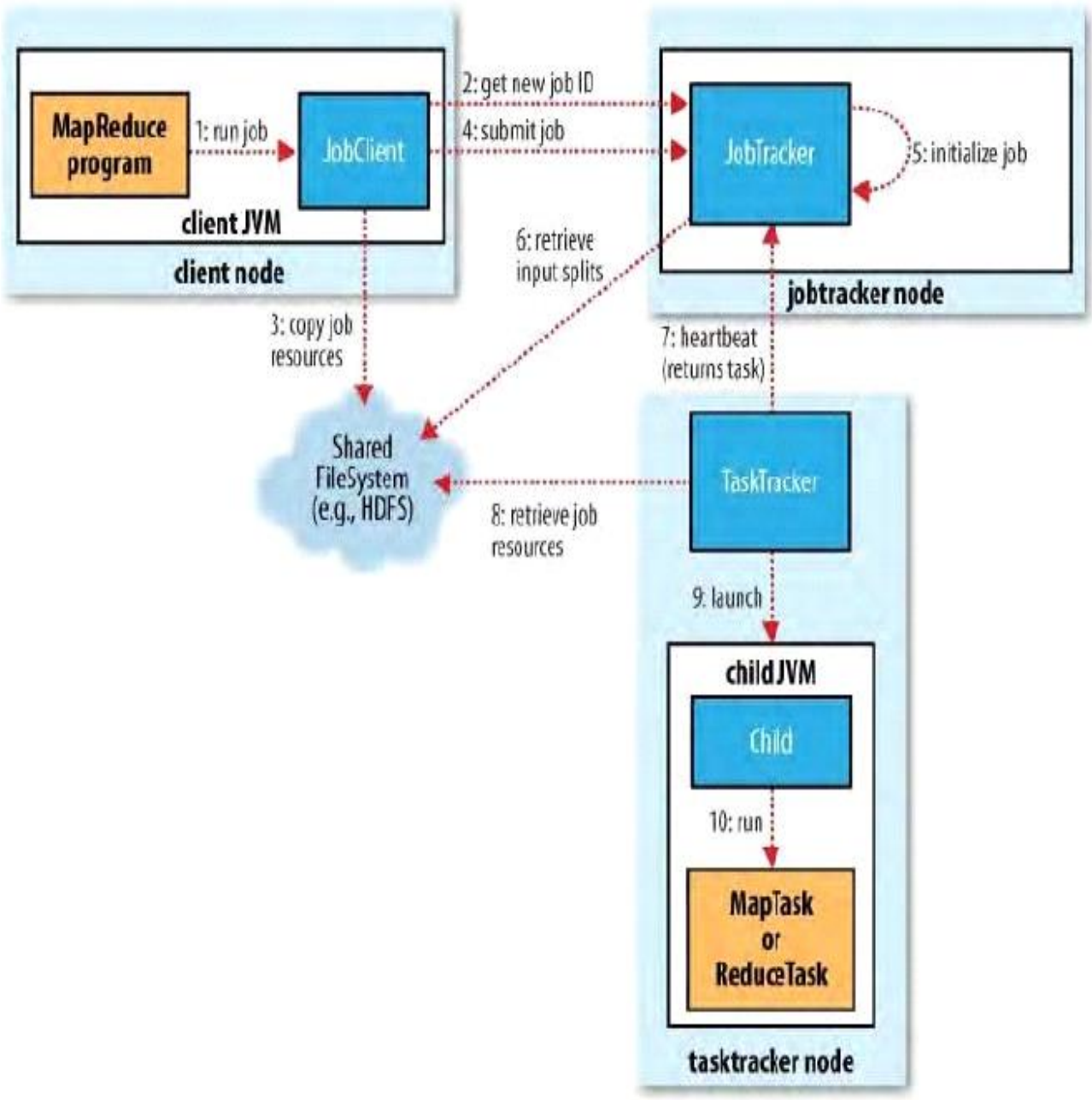
```
public class FlowCountReducer extends Reducer<Text, FlowBean, Text, FlowBean> {
```

```
    public void reduce(Text _key, Iterable<FlowBean> values, Context context)
    throws IOException, InterruptedException {
        // process values
        FlowBean flowBean=new FlowBean();
        flowBean.setFlow(0);
        for (FlowBean val : values) {
            flowBean.setFlow(flowBean.getFlow()+val.getFlow());
        }
        context.write(_key,flowBean );
    }

}
```

Driver代码：

```
public class FlowCountDriver {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "JobName");
        job.setJarByClass(cn.tedu.hadoop.FlowCountDriver.class);
        // TODO: specify a mapper
        job.setMapperClass(FlowCountMapper.class);

        // TODO: specify a reducer
        job.setReducerClass(FlowCountReducer.class);

        // TODO: specify output types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FlowBean.class);

        // TODO: specify input and output DIRECTORIES (not files)
        FileInputFormat.setInputPaths(job, new
        Path("hdfs://192.168.234.21:9000/wc/flow.txt"));
        FileOutputFormat.setOutputPath(job, new
        Path("hdfs://192.168.234.21:9000/result"));

        if (!job.waitForCompletion(true))
            return;
    }

}
```

job任务执行流程



1.run job阶段。此阶段发生在客户端进程中，底层有一个JobClient类在工作，JobClient在这个阶段的职责是：

①收集整个job的环境信息（比如通过conf设定的参数，还有mapperClass,reducerClass,以及输出kv类型）

②会计算当前job的切片数量（切片不同等切块，用FileSplit：path start length）

③检测环境信息的合法性，以及输入和输出的路劲合法性。

比如输出的kv类型不匹配，输入路径不存在，输出的结果目录已存在，以上都是不合法的，所以就会直接报

错返回，不会再继续下一步了。

2.如果第一步的检测通过之后，会去找JobTracker，为当前的job申请jobid，用于标识job。jobid是全局唯一的，目的是管理job，因为整个集群同一时间内可能跑多个job。

3.JobClient收到jobid，就将此job的运算资源（①conf.xml ②summary ③jar 包）

提交到HDFS上，目录路径：/tmp/hadoop-yarn/history/done_intermediate/root

conf.xml：存储的是job的环境配置信息

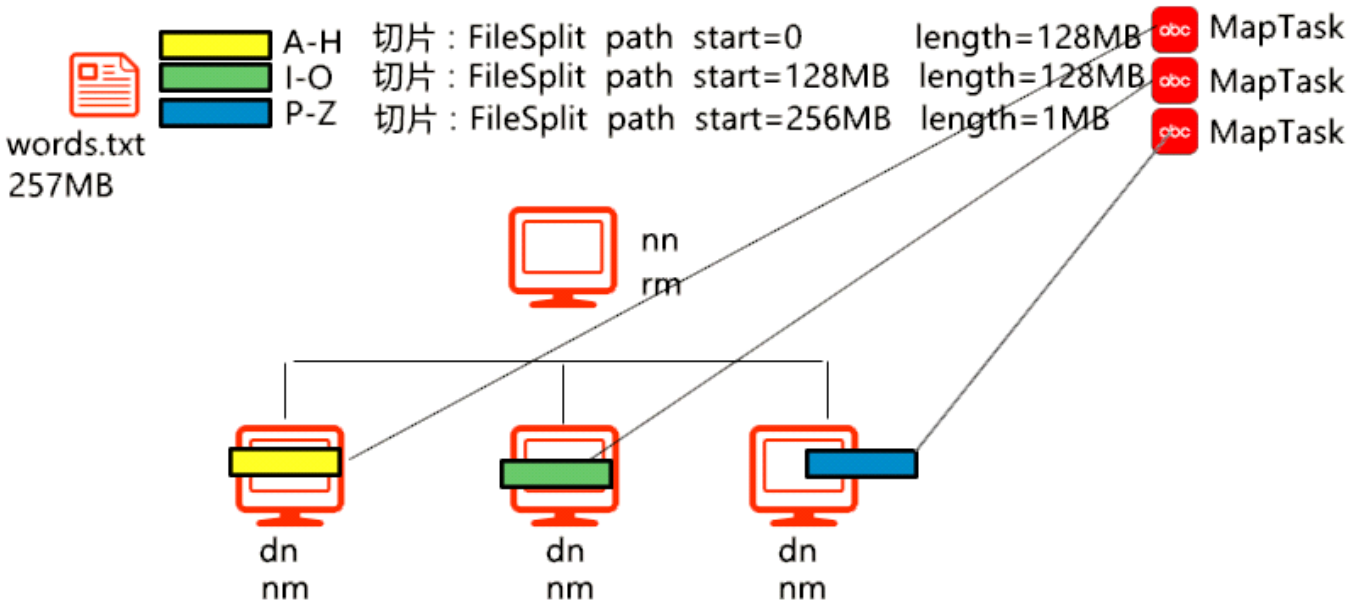
summary:jobid，mapTask数量和reduceTask数

jar包:程序员写的代码

4.JobClient 做submit job动作，底层是把第三步的job的资源路径信息告知给JobTracker。

5.6 去HDFS上拿取job的运算资源，然后做job的初始化，查看配置信息，以及拿到job的切片数量（本地目的是获取有几个mapTask）

7.任务的领取，底层要满足数据本地化策略，节省集群的带宽



MR任务分配的机制，是干活的节点（nodemanager)去领任务，这里底层要满足的是数据本地化策略，目的是为了避免网络大量传输数据，节省集群带宽。

补充：因为MapTask读取文件是按行读取，所以必须要保证读取的是完整一行。底层会发生一个**位置追溯**的动作，此过程不可避免的会发生网络数据的传输，但数据量很小。

8.去HDFS获取job 的运算资源（主要是jar包），然后结合代码来处理数据了。

这里体现了Hadoop的思想：移动的是运算，而不是数据。目的也是节省集群带宽。

9.10启动JVM进程,执行MapTask或ReduceTask。

补充：MapTask任务的数量=job的切片数量

ReduceTask任务数量，默认就一个。

ReduceTask数+分区机制

2018年8月28日 10:59

概述

1.一个Job的ReduceTask数量，默认就1个。可以通过代码来设置

代码示例：

```
public static void main(String[] args) throws Exception {

		Configuration conf=new Configuration();

		Job job=Job.getInstance(conf);

		job.setJarByClass(WordCountDriver.class);

		job.setMapperClass(WordCountMapper.class);

		job.setMapOutputKeyClass(Text.class);

		job.setMapOutputValueClass(IntWritable.class);

		job.setReducerClass(WordCountReducer.class);

		job.setOutputKeyClass(Text.class);

		job.setOutputValueClass(IntWritable.class);

		//--设置reduceTask任务数量，默认1个
		job.setNumReduceTasks(3);


		FileInputFormat.setInputPaths(
				job,new Path("hdfs://192.168.150.137:9000/word"));


		FileOutputFormat.setOutputPath(
				job,new Path("hdfs://192.168.150.137:9000/word/result"));

		job.waitForCompletion(true);
	}
```

2.习惯上，把reduceTask叫做分区，即有几个reduceTask，就有几个分区。

3.Hadoop底层有一个默认的分区器（HashPartitioner），此分区器的作用可以确保相同的Mapper输出key落到同一个分区（reduceTask)里。
算法： return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;

4.最后的结果文件数量=分区（reduceTask）数量，即每个结果文件存储的是对应分区的结果数据。

5.因为底层用的是简单hash算法，所以会产生数据倾斜，有时会产生某个结果文件数据很少或没有的情况。

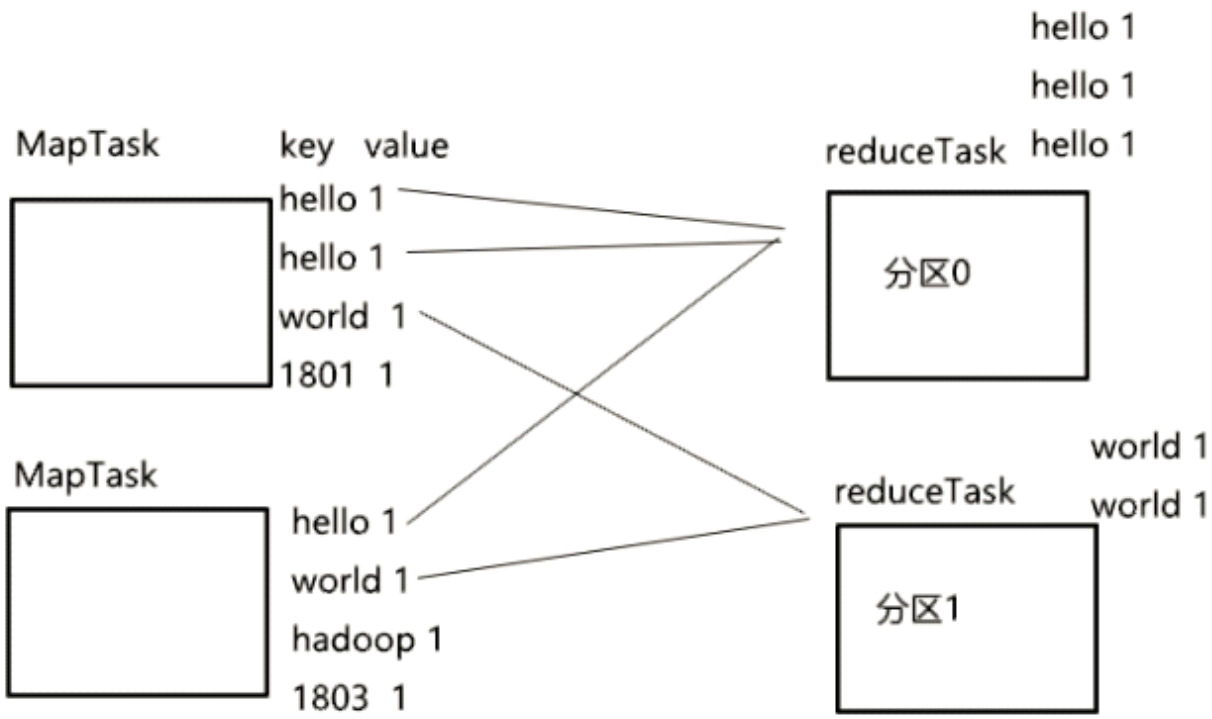
6.如果最后想多个结果文件的结果合并在一起，

可以通过指令：hadoop fs getmerge /word/result ./result.txt。

多个reduceTask做合并，需要满足的条件：相同的key，落到同一reduceTask里。这种机制称为分区（partition）机制

key.hashcode%2

确保相同key落到同一分区（reduceTask)里



分区案例+自定义分区案例

2016年9月8日 10:29

分区案例：

计算每月的利润

1 280

1 560

2 234

2 264

3 873

3 2323

要求：用三个分区来存放三个月的总利润

实现思路

输出map的key值为月份数。然后在Driver里设置分区数量（reduceTask数量）为3。Hadoop会根据key的hashcode值来进行分区。

自定义分区案例：

FlowCountPartition代码：

```
/**
 * 想要实现自定义分区，需要继承Partitioner类，并重写getPartition（ ）方法。
 *
 *
 * 根据用户自定义的分区，把输出map的value值（本例中是一个一个的FlowBean对象），根
```

```
据用户定义的条件放到对应的分区里。
 * 比如如下的数据：
 *     13877779999 bj zs 2145
     13766668888 sh ls 1028
     13766668888 sh ls 9987
     13877779999 bj zs 5678
     13544445555 sz ww 10577
     13877779999 sh zs 2145
     13766668888 sh ls 9987
 * 我们如果想按地区分区，bj,sh,sz 三个区，
 * bj区：13877779999 bj zs 2145 ，13877779999 bj zs 5678
 * sh区：13766668888 sh ls 1028 ， 13766668888 sh ls 9987 ，13877779999 sh zs
2145  ，13766668888 sh ls 9987
 * sz区：13544445555 sz ww 10577
 * 分完区之后，在每个区里，会按照输出map的key值整理成如下形式，然后发送给reduce
端：
 * bj区：[zs,{13877779999 bj zs 2145},{13877779999 bj zs 2145}]
 * sh区：[ls,{},{},{}],[zs,{}]
 * sz去：[ww,{}]
 *
 * 然后reduce端拿到数据后，根据用户的自定义代码进行每个区里的数据合并
 * @author ysq
 *
 */
public class FlowCountPartition extends Partitioner<Text,FlowBean>{
```

```
    @Override
    public int getPartition(Text key, FlowBean value, int numPartitions) {
        if(value.getAddress().equals("bj")){
            return 0;
        }else if(value.getAddress().equals("sh")){
            return 1;
        }else if(value.getAddress().equals("sz")){
            return 2;
        }
        else{
            return 3;
        }
    }

}
```

FlowCountDriver代码：

```
public class FlowCountDriver {

    public static void main(String[] args) throws Exception {
        Configuration conf=new Configuration();
        Job job=Job.getInstance(conf);

        job.setJarByClass(FlowCountDriver.class);
        job.setMapperClass(FlowCountMapper.class);
```

```
        job.setReducerClass(FlowCountReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(FlowBean.class);

        job.setPartitionerClass(FlowCountPartition.class);
        job.setNumReduceTasks(4);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FlowBean.class);

        FileInputFormat.setInputPaths(job, new Path("/park/flow.txt"));
        FileOutputFormat.setOutputPath(job, new Path("/park/result01"));

        job.waitForCompletion(true);

    }
}
```

FlowCountMapper代码（姓名为输出key值）：

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

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
    Text, FlowBean>.Context context)
            throws IOException, InterruptedException {
```

```
        String line=value.toString();
        String[] data=line.split(" ");
        FlowBean flow=new FlowBean();
        flow.setPhone(data[0]);
        flow.setAddress(data[1]);
        flow.setName(data[2]);
        flow.setFlow(Long.parseLong(data[3]));

        context.write(new Text(flow.getName()), flow);
    }
}
```

FlowCountReducer代码（姓名为输出key值）：

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

    @Override
    protected void reduce(Text key, Iterable<FlowBean> values, Reducer<Text,
    FlowBean, Text, FlowBean>.Context context)
            throws IOException, InterruptedException {
        FlowBean reduceBean=new FlowBean();
        for(FlowBean flow:values){
            reduceBean.setFlow(reduceBean.getFlow()+flow.getFlow());
            reduceBean.setPhone(flow.getPhone());
            reduceBean.setName(flow.getName());
            reduceBean.setAddress(flow.getAddress());
```

```
            }

            context.write(new Text(key), reduceBean);


        }

    }
```

案例——排序

2015年12月7日 19:50

源文件：

惊天破 72

机械师2 83

奇异博士 67

但丁密码 79

比利林恩的中场战事 84

侠探杰克:永不回头 68

龙珠Z:复活的弗利萨 79

长城 56

夺路而逃 69

神奇动物在哪里 57

驴得水 68

我不是潘金莲 75

你的名字 77

大闹天竺 42

捉迷藏 78

凶手还未睡 23

魔发精灵 68

勇士之门 35

罗曼蒂克消亡史 67

小明和他的小伙伴们 36

Hadoop插件中文乱码解决办法：

Eclipse=>Preferences=》workspace

MR会对Mapper输出key做排序。

结果文件：

电影名 热度值

Movie代码：

```
public class Movie implements WritableComparable<Movie>{
	private String name;
	private int hot;

	@Override
	public void write(DataOutput out) throws IOException {
		out.writeUTF(name);
		out.writeInt(hot);

	}

	@Override
	public void readFields(DataInput in) throws IOException {
		this.name=in.readUTF();
		this.hot=in.readInt();

	}
```

```
@Override
public int compareTo(Movie o) {

     return this.hot-o.hot;
}

public String getName() {
     return name;
}

public void setName(String name) {
     this.name = name;
}

public int getHot() {
     return hot;
}

public void setHot(int hot) {
     this.hot = hot;
}

@Override
public String toString() {
```

```
        return "Movie [name=" + name + ", hot=" + hot + "]";
    }



}
```

SortMapper代码：

```
public class SortMapper extends Mapper<LongWritable,Text,Movie,NullWritable>{
    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
    Movie,NullWritable>.Context context)
            throws IOException, InterruptedException {

        String line=value.toString();
        String name=line.split(" ")[0];
        int hot=Integer.parseInt(line.split(" ")[1]);
        System.out.println(name+":"+hot);
        Movie movie=new Movie();
        movie.setName(name);
        movie.setHot(hot);
        context.write(movie,NullWritable.get());

    }

}
```

SortDriver代码：

```
public static void main(String[] args) throws Exception {
        Configuration conf=new Configuration();
        Job job=Job.getInstance(conf);


        job.setJarByClass(SortDriver.class);

        job.setMapperClass(SortMapper.class);


        job.setMapOutputKeyClass(Movie.class);

        job.setMapOutputValueClass(NullWritable.class);



        FileInputFormat.setInputPaths(job,new
        Path("hdfs://192.168.234.191:9000/sort"));

        FileOutputFormat.setOutputPath(job,new
        Path("hdfs://192.168.234.191:9000/sort/result"));

        job.waitForCompletion(true);
    }
```

案例——全排序

需求说明

有这样一组数字，要求利用3个reduce来处理，并且生成的三个结果文件，是整体有序的。

源数据：

```
82 239 231
23 22 213
123 232 124
213 3434 232
4546 565 123
231 231
2334 231
1123 5656 657
12313 4324 213
123 2 232 32
343 123 4535
12321 3442 453
1233 342 453
1231 322 452
232 343 455
3123 3434 3242
```

生成的三个结果文件：

文件1		文件2		文件3	
2	1	123	4	1123	1
22	1	124	1	1231	1
23	1	213	3	1233	1
32	1	231	4	2334	1
82	1	232	4	3123	1
		239	1	3242	1
		322	1	3434	2
		342	1	3442	1
		343	2	4324	1
		452	1	4535	1
		453	2	4546	1
		455	1	5656	1
		565	1	12313	1
		657	1	12321	1

实现思路

如果是一个reduce，那肯定就是全排序。但如果只单纯指定3个reduce数量，Hadoop会默认根据Key的hash进行分区，这样不会出现三个结果文件是全排序的情况。所以，需要自定义分区，在自定义分区里，根据数据采样边界来对数据进行分区。

自定义分区代码：

```
public class AuthPartitioner extends Partitioner<IntWritable, IntWritable>{

	@Override
	public int getPartition(IntWritable key, IntWritable value, int
	numPartitions) {
		String num=String.valueOf(key.get());
		if(num.matches("[0-9][0-9]")||num.matches("[0-9]")){
			return 0;
		}
		if(num.matches("[0-9][0-9][0-9]")){
			return 1;
		}else{
			return 2;
		}

	}

}
```

Mapper代码：

```
public class SimpleMapper extends
Mapper<LongWritable,Text,IntWritable,IntWritable>{
	@Override
	protected void map(LongWritable key, Text value,
			Mapper<LongWritable, Text, IntWritable, IntWritable>.Context
			context)
			throws IOException, InterruptedException {
		String line=value.toString();
		String[] data=line.split(" ");
		for(String num:data){
			context.write(new IntWritable(Integer.parseInt(num)), new
			IntWritable(1));
		}

	}

}
```

Reducer代码：

```
public class SimpleReducer extends Reducer<IntWritable, IntWritable, IntWritable,
IntWritable>{
	@Override
	protected void reduce(IntWritable num, Iterable<IntWritable> counts,
			Reducer<IntWritable, IntWritable, IntWritable, IntWritable>.Context
			context)
			throws IOException, InterruptedException {
		int result=0;
		for(IntWritable count:counts){
			result=result+count.get();
		}
		context.write(num, new IntWritable(result));
	}

}
```

Driver代码：

```
public class SimpleDriver {

	public static void main(String[] args) throws Exception {
		Configuration conf=new Configuration();
		Job job=Job.getInstance(conf);

		job.setJarByClass(SimpleDriver.class);
		job.setMapperClass(SimpleMapper.class);
		job.setReducerClass(SimpleReducer.class);

		job.setMapOutputKeyClass(IntWritable.class);
		job.setMapOutputValueClass(IntWritable.class);
		job.setOutputKeyClass(IntWritable.class);
		job.setOutputValueClass(IntWritable.class);
		job.setPartitionerClass(AuthPartitioner.class);

		job.setNumReduceTasks(3);

		FileInputFormat.setInputPaths(job, new
		Path("hdfs://192.168.234.21:9000/totalsort"));
		FileOutputFormat.setOutputPath(job, new
		Path("hdfs://192.168.234.21:9000/totalsort/result"));

		job.waitForCompletion(true);

	}
}
```

案例——Job嵌套（链）

2018年8月1日 11:17

文件：

1|zhang 100

2|wang 200

3|zhang 150

4|lisi 190

5|wang 50

6|zhang 80

7|lisi 50

要求：统计每个人的总利润，并按利润做降序排序

最后的结果如下：

Profit [name=zhang, profit=330]

Profit [name=wang, profit=250]

Profit [name=lisi, profit=240]

完成这个要求需要两个MR job，第一个Job用于利润的统计，第二个Job用于排序

ProfitMapper代码：

```
public class ProfitMapper extends Mapper<LongWritable,Text,Text,IntWritable>{
	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
	IntWritable>.Context context)
			throws IOException, InterruptedException {

		String line=value.toString();
		String name=line.split("\\|")[1].split(" ")[0];
		int profit=Integer.parseInt(line.split("\\|")[1].split(" ")[1]);

		context.write(new Text(name), new IntWritable(profit));
	}

}
```

ProfitReducer代码：

```
public class ProfitReducer extends Reducer<Text, IntWritable,Text,IntWritable>{

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

		int sum=0;
		for(IntWritable value:values){
			sum=sum+value.get();
		}
		context.write(key, new IntWritable(sum));
	}
}
```

Profit代码：

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

	private String name;
	private int profit;
	@Override
	public void write(DataOutput out) throws IOException {
		out.writeUTF(name);
		out.writeInt(profit);

	}
	@Override
	public void readFields(DataInput in) throws IOException {
		this.name=in.readUTF();
		this.profit=in.readInt();

	}
	@Override
	public int compareTo(Profit o) {

		return o.profit-this.profit;
```

```
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getProfit() {
		return profit;
	}
	public void setProfit(int profit) {
		this.profit = profit;
	}
	@Override
	public String toString() {
		return "Profit [name=" + name + ", profit=" + profit + "]";
	}

}
```

SortMapper代码：

```
public class SortMapper extends Mapper<LongWritable, Text,Profit,NullWritable>{

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Profit,
	NullWritable>.Context context)
			throws IOException, InterruptedException {

			String line=value.toString();
			String name=line.split("\t")[0];
			int profit=Integer.parseInt(line.split("\t")[1]);

			Profit p=new Profit();
			p.setName(name);
			p.setProfit(profit);

			context.write(p, NullWritable.get());
```

```
    }
}
```

Driver代码：

```
public class Driver {

    public static void main(String[] args) throws Exception {
        Configuration conf=new Configuration();
        Job job=Job.getInstance(conf);

        job.setJarByClass(Driver.class);
        job.setMapperClass(ProfitMapper.class);
        job.setReducerClass(ProfitReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path("hdfs://192.168.150.137:9000/profit"));
        FileOutputFormat.setOutputPath(job,new
        Path("hdfs://192.168.150.137:9000/profit/result"));

        if(job.waitForCompletion(true)){
            Job job2=Job.getInstance(conf);
            job2.setMapperClass(SortMapper.class);
            job2.setMapOutputKeyClass(Profit.class);
            job2.setOutputValueClass(NullWritable.class);

            FileInputFormat.addInputPath(job2, new
            Path("hdfs://192.168.150.137:9000/profit/result"));
            FileOutputFormat.setOutputPath(job2,new
            Path("hdfs://192.168.150.137:9000/profit/sortresult"));

            job2.waitForCompletion(true);
        }
```

```
        }
    }
```

案例——计算成绩

2016年9月8日　　13:23

案列说明：

有三个成绩文件，chinese.txt,english.txt,math.txt

文件数据内容：

chinese.txt：	english.txt：	math.txt：
1 zhang 89	1 zhang 55	1 zhang 85
2 zhang 73	2 zhang 69	2 zhang 59
3 zhang 67	3 zhang 75	3 zhang 95
1 wang 49	1 wang 44	1 wang 74
2 wang 83	2 wang 64	2 wang 67
3 wang 27	3 wang 86	3 wang 96
1 li 77	1 li 76	1 li 45
2 li 66	2 li 84	2 li 76
3 li 89	3 li 93	3 li 67

作业要求：

计算每个人 三个月，每科的总成绩

结果格式：

```
li     Student [ name=li, chinese=232, english=253, math=188]
wang   Student [ name=wang, chinese=159, english=194, math=237]
zhang  Student [ name=zhang, chinese=229, english=199, math=239]
```

技术点

FileSpilt 对象获取文件名

Student代码：

```
public class Student implements Writable{


    private  int month;

    private String name;

    private int chinese;

    private int english;

    private int math;



    @Override

    public void write(DataOutput out) throws IOException {

        out.writeInt(month);

        out.writeUTF(name);

        out.writeInt(chinese);

        out.writeInt(english);

        out.writeInt(math);


    }


    @Override
```

```
    public void readFields(DataInput in) throws IOException {
        this.month=in.readInt();
        this.name=in.readUTF();
        this.chinese=in.readInt();
        this.english=in.readInt();
        this.math=in.readInt();

    }

.....

    @Override
    public String toString() {
        return "Student [ name=" + name + ", chinese=" + chinese + ", english="
        + english
                + ", math=" + math + "]";
    }

}
```

ScoreCountMapper代码：

```
public class ScoreCountMapper extends Mapper<LongWritable, Text, Text, Student> {

    @Override
    protected void map(LongWritable key, Text value,
            Mapper<LongWritable, Text, Text, Student>.Context context)
            throws IOException, InterruptedException {
```

```
        String line=value.toString();
        String[] data=line.split(" ");

        Student student=new Student();
        student.setName(data[1]);


        //注意：包别导错了，用这个包：
        org.apache.hadoop.mapreduce.lib.input.FileSplit
        FileSplit split= (FileSplit) context.getInputSplit();
        if(split.getPath().getName().equals("chinese.txt")){
            student.setChinese(Integer.parseInt(data[2]));

        }
        if(split.getPath().getName().equals("english.txt")){
            student.setEnglish(Integer.parseInt(data[2]));

        }
        if(split.getPath().getName().equals("math.txt")){
            student.setMath(Integer.parseInt(data[2]));

        }

        context.write(new Text(student.getName()), student);
    }
```

```
}
```

ScoreCountReducer代码：

```
public class ScoreCountReducer extends Reducer<Text, Student,Text,Student>{


    @Override
    protected void reduce(Text key, Iterable<Student> values, Reducer<Text,
    Student, Text, Student>.Context context)
            throws IOException, InterruptedException {
        Student reduceStudent=new Student();
        reduceStudent.setName(key.toString());
        for(Student student:values){
            reduceStudent.setChinese(reduceStudent.getChinese()+student.getChin
            ese());
            reduceStudent.setEnglish(reduceStudent.getEnglish()+student.getEngl
            ish());
            reduceStudent.setMath(reduceStudent.getMath()+student.getMath());
        }
        context.write(key, reduceStudent);
    }
}
```

ScoreDriver代码：

```
public class ScoreCountDriver {


    public static void main(String[] args) throws Exception {
```

```
        Configuration conf=new Configuration();
        Job job=Job.getInstance(conf);

        job.setJarByClass(ScoreCountDriver.class);
        job.setMapperClass(ScoreCountMapper.class);
        job.setReducerClass(ScoreCountReducer.class);



        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Student.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Student.class);

        FileInputFormat.setInputPaths(job,new
        Path("hdfs://192.168.234.22:9000/score"));
        FileOutputFormat.setOutputPath(job, new
        Path("hdfs://192.168.234.22:9000/score/result"));

        job.waitForCompletion(true);



    }
}
```

课后作业：案例——找出潜在好友

2018年8月1日 14:37

案例文件：

tom rose

tom jim

tom smith

tom lucy

rose tom

rose lucy

rose smith

jim tom

jim lucy

smith jim

smith tom

smith rose

比如tom认识rose，但是rose不认识jim，则那么rose-jim就是一对潜在好友，但tom-rose早就认识了，因此不算为潜在好友。

最后的结果为：

jim-rose

lucy-smith