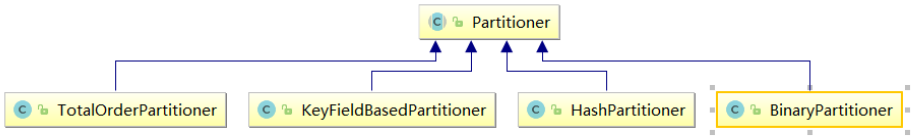
1、MapReduce 的分区与 reduceTask 的数量

在 MapReduce 中，通过我们指定分区，会将同一个分区的数据发送到同一个 reduce 当中进行处理，例如我们为了数据的统计，我们可以把一批类似的数据发送到同一个 reduce 当中去，在同一个 reduce 当中统计相同类型的数据，就可以实现类似数据的分区，统计等

说白了就是相同类型的数据，送到一起去处理，在 reduce 当中默认分区只有 1 个。

MapReduce 当中的分区类图



需求：将以下数据进行分开处理

详细数据参见 partition.csv　这个文本文件，其中第五个字段表示开奖结果数值，现在需求将 15 以上的结果以及 15 以下的结果进行分开成两个文件进行保存

注意：分区的案例，只能打成 jar 包发布到集群上面去运行，本地模式已经不能正常运行了

第一步：定义我们的 mapper

我们这里的 mapper 程序不做任何逻辑，也不对 key，与 value 做任何改变，只是接收我们的数据，然后往下发送

```
public class MyMapper extends
```

```
Mapper<LongWritable,Text,Text,NullWritable>{
    @Override
    protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        context.write(value,NullWritable.get());
    }
}
```

第二步：定义我们的 reducer 逻辑

我们的 reducer 也不做任何处理，将我们的数据原封不动的输出即可

```
public class MyReducer extends
Reducer<Text,NullWritable,Text,NullWritable> {
    @Override
    protected void reduce(Text key, Iterable<NullWritable> values,
Context context) throws IOException, InterruptedException {
        context.write(key,NullWritable.get());
    }
}
```

第三步：自定义 partitioner

```
/**
 * 这里的输入类型与我们 map 阶段的输出类型相同
 */
```

```
public class MyPartitioner extends Partitioner<Text,NullWritable>{
    /**
     * 返回值表示我们的数据要去到哪个分区
     * 返回值只是一个分区的标记，标记所有相同的数据去到指定的分区
     */
    @Override
    public int getPartition(Text text, NullWritable nullWritable, int
i) {
        String result = text.toString().split("\t")[5];
        System.out.println(result);
        if (Integer.parseInt(result) > 15){
            return 1;
        }else{
            return 0;
        }
    }
}
```

第四步：程序 main 函数入口

```
public class PartitionMain  extends Configured implements Tool {
    public static void main(String[] args) throws  Exception{
        int    run    =    ToolRunner.run(new    Configuration(),    new
PartitionMain(), args);
        System.exit(run);
    }
    @Override
    public int run(String[] args) throws Exception {
        Job        job        =        Job.getInstance(super.getConf(),
```

```
PartitionMain.class.getSimpleName());
        job.setJarByClass(PartitionMain.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        TextInputFormat.addInputPath(job,new
Path("hdfs://192.168.52.100:8020/partitioner"));
        TextOutputFormat.setOutputPath(job,new
Path("hdfs://192.168.52.100:8020/outpartition"));
        job.setMapperClass(MyMapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(NullWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setMapOutputValueClass(NullWritable.class);
        job.setReducerClass(MyReducer.class);
        /**
         * 设置我们的分区类，以及我们的 reducetask 的个数，注意
reduceTask 的个数一定要与我们的
         * 分区数保持一致
         */
        job.setPartitionerClass(MyPartitioner.class);
        job.setNumReduceTasks(2);
        boolean b = job.waitForCompletion(true);
        return b?0:1;
    }
}
```