



《大数据技术应用》

第七章 MapReduce 高级编程



提纲

7.1 自定义数据类型

7.2 自定义输入/输出

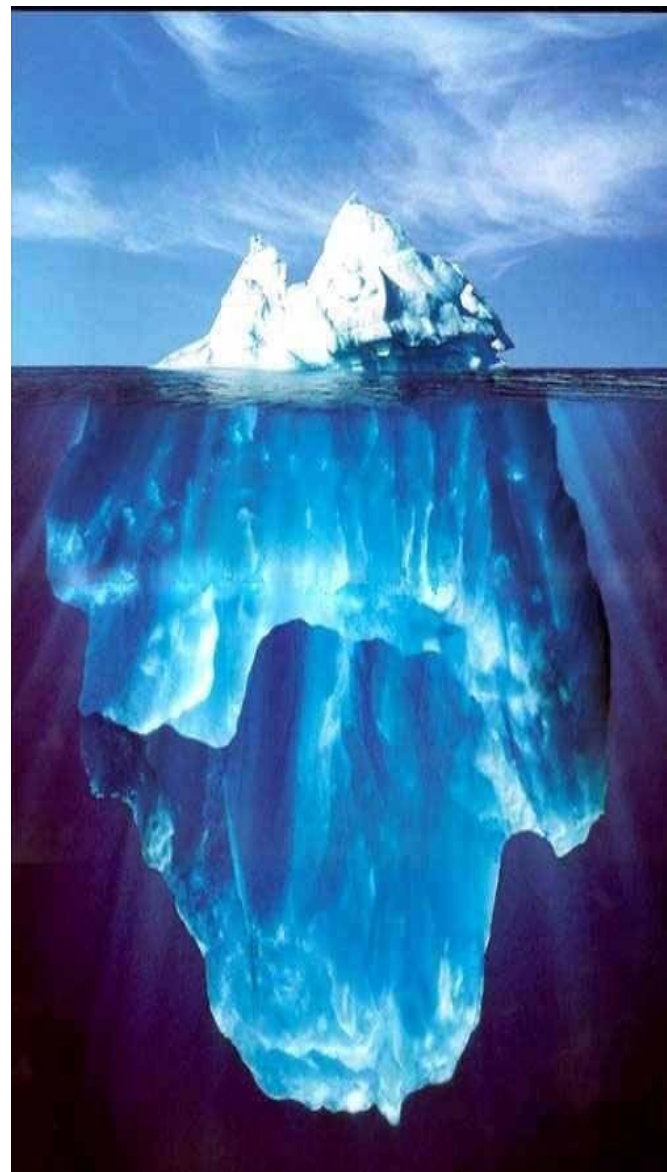
7.3 自定义**Combiner/Partitioner**

7.4 组合式计算作业

7.5 **MapReduce**的特性

7.6 **MapReduce**应用举例

——成绩分析系统的实现



7.1 自定义数据类型

- **Hadoop**提供了很多内置的数据类型，常用的是**Java**基本类型的**Writable**封装，例如：
- **FloatWritable**：浮点数
- **IntWritable**：整型数
- **LongWritable**：长整型数
- **Text**：使用**UTF-8**格式存储文本
- 这些数据类型都实现了**WritableComparable**接口，可以进行网络的传输和文件存储，以及大小的比较。

7.1 自定义数据类型

WritableComparable接口的定义如下：

```
public interface WritableComparable<T>
{
    public void readFields(DataInput in);
    public void write(DataOutput out);
    public int compareTo(T other);
}
```

相较于**Writable**接口，**WritableComparable**接口多了一个**compareTo()**方法，实现数据的比较。

7.1 自定义数据类型

- 自定义数据类型可以实现两个接口。一个是 **Writable** 接口，需要实现其 **write()** 和 **readFields()** 方法，以便数据能被序列化后完成网络传输或文件输入/输出；一个是 **WritableComparable** 接口，需要实现 **write()**、**readFields()**、**CompareTo()** 方法，以便数据能用作 **key** 或比较大小。

7.2 自定义输入/输出

7.2.1 RecordReader与RecordWriter

1. RecordReader

对于每一种数据输入格式，都需要有一个对应的**RecordReader**。**RecordReader**主要用于将一个文件中的数据记录拆分成具体的**key-value**键值对，并传送给**Map**节点作为输入数据。

7.2.1 RecordReader与RecordWriter

RecordReader的定义如下:

```
public abstract class RecordReader<KEYIN, VALUEIN> implements  
    Closeable  
{ //由一个InputSplit初始化  
public abstract void initialize(InputSplit split, TaskAttemptContext  
    context)  
//读取分片中的下一个key-value对  
public abstract boolean nextKeyValue();  
//获取当前key的值  
public abstract KEYIN getCurrentKey();  
//获取当前value的值  
public abstract VALUEIN getCurrentValue();  
//跟踪读取分片的进度  
public abstract float getProgress();  
public abstract void close();  
}
```

7.2.1 RecordReader与RecordWriter

2. RecordWriter

与输入格式中的**RecordReader**类似，每一种数据输出格式也需要有一个对应的**RecordWriter**。**RecordWriter**主要用于将作业输出结果按照格式写到文件中。**RecordWriter**的定义：

```
public abstract class RecordWriter<K, V>
{
    public abstract void write(K key, V value);
    public abstract void close(TaskAttemptContext context);
}
```


7.2.2 自定义输入

在一些特定情况下用户需要定制自己的数据输入格式和**RecordReader**。自定义输入格式步骤如下：

- (1) 定义一个继承自**InputFormat**的类，一般继承**FileInputFormat**类即可。
- (2) 实现其**createRecordReader()**方法，返回一个**RecordReader**。
- (3) 自定义一个继承**RecordReader**的类，实现其**initialize()**、**getCurrentKey()**、**getCurrentValue()**方法，选择性实现**nextKeyValue()**。

7.2.3 自定义输出

用户可以根据应用程序的需要自定义数据输出格式与**RecordWriter**。自定义输出格式步骤如下：

(1) 定义一个继承自**OutputFormat**的类，一般继承**FileOutputFormat**类即可。

(2) 实现其**getRecordWriter()**方法，返回一个**RecordWriter**。

(3) 自定义一个继承**RecordWriter**的类，实现其**writer()**方法，针对每个**key-value**键值对写入文件数据。

7.3 自定义Combiner/Partitioner

7.3.1 自定义Combiner

用户可以根据需要自定义**Combiner**，以减少**Map**阶段输出中间结果的数据量，降低数据的网络传输开销。自定义**Combiner**步骤如下：

- (1) 定义一个继承自**Reducer**的类。
- (2) 实现其**reduce()**方法。
- (3) 通过**job.setCombinerClass()**来设置自定义的**Combiner**。

7.3.2 自定义Partitioner

自定义**Partitioner**步骤如下：

- (1) 定义一个继承自**Partitioner**的类。
- (2) 实现其**getPartition()**方法。
- (3) 通过**job.setPartitionerClass()**来设置自定义的**Partitioner**。

【实例】从给定的文件中分别统计每种商品的周销售情况，并以不同的文件输出。

销售统计分析

分店1的销售清单

Branch1.txt

Desk 20

Sofa 2

Bed 5

Chair 10

分店2的销售清单

Branch2.txt

Desk 50

Sofa 21

Bed 54

Chair 60

- 所有分店相同商品数量求和。要以不同文件输出。
- **Partitioner**决定每个**key-value**键值对由那个**reducer**处理，每个**reducer**有一个输出文件。
- 将有**partitioner**将不同商品分发给不同**reducer**处理。
- 上述例子会产生4个**reducer**，进而产生4个输出文件。

GoodsStatistics.java

```
Configuration conf = new Configuration();
String[] ortherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
//获取程序的输入参数
if(ortherArgs.length!=2) //如果未获得两个输入参数，输出提示信息并结束程序
{
    System.err.println("Usage: GoodsStatisticst <in> <out>");
    System.exit(2);
}
Job job =Job.getInstance(conf," GoodsStatisticst");
job.setMapperClass(GoodsMapper.class);
job.setReducerClass(GoodsReducer.class);
job.setPartitionerClass(GoodsPartitoner.class);
job.setNumReduceTasks(4); //设置4个reducer
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job,new Path(ortherArgs[0]));
FileOutputFormat.setOutputPath(job,new Path(ortherArgs[1]));
System.exit(job.waitForCompletion(true)? 0:1);
}
```

GoodsMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class GoodsMapper extends Mapper <LongWritable, Text, Text, IntWritable>
{
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException
    {
        String[] data = value.toString().split("\\s+"); //将数据转换为字符数组
        System.out.println("map:"+data[0].toString());
        System.out.println("map:"+data[1].toString());
        context.write(new Text(data[0]), new IntWritable(Integer.parseInt(data[1])));
    }
}
```

GoodsPartition.java

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;
public class GoodsPartitoner extends Partitioner<Text, IntWritable>
{
    public int getPartition(Text key, IntWritable value, int numPartitons)
    {
        //将4种商品转发给4个不同的reducer
        if(key.toString().equals("desk"))
            return 0;
        if(key.toString().equals("sofa"))
            return 1;
        if(key.toString().equals("bed"))
            return 2;
        return 3;
    }
}
```


GoodsReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class GoodsReducer extends Reducer <Text, IntWritable, Text, IntWritable>
{
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException
    {
        int sum = 0;
        for(IntWritable val:values)
        {
            sum += val.get();
            System.out.println("reduce:"+key.toString());
            System.out.println(val.get());
        }
        context.write(key, new IntWritable(sum));
    }
}
```

7.4 组合式计算作业

7.4.1 迭代式计算

MapReduce迭代式计算的中心思想，类似**for**循环，前一个**MapReduce**的输出结果，作为下一个**MapReduce**的输入，任务完成后中间结果都可以删除。例如，现在有3个**MapReduce**子任务。其中，子任务1的输出目录**Outpath1**将作为子任务2的输入目录，而子任务2的输出目录**Outpath2**又作为子任务3的输入目录。

7.4.2 依赖关系组合式计算

- 对于依赖关系组合式计算，同样是需要多个**MapReduce**才能完成任务，却不是顺序执行。例如，**MapReduce**有3个子任务**job1**，**job2**，**job3**，其中**job1**和**job2**相互独立，**job3**要在**job1**和**job2**完成之后才执行，就称作**job3**依赖于**job1**和**job2**。
- **Hadoop**为这种依赖关系组合式计算提供了一种执行和控制的机制。**Hadoop**通过**Job**和**JobControl**类提供具体的编程方法。**Job**除了维护子任务的配置信息，还维护子任务的依赖关系。而**JobControl**控制整个作业流程，把所有的子任务作业加入到**JobControl**中，执行**JobControl**的**run()**方法即可运行程序。

7.4.3 链式计算

- 一个**MapReduce**作业可能会有一些前处理和后处理步骤，一个较好的办法就是在核心的**MapReduce**之外，增加一个辅助的**Map**过程，然后将这个辅助的**Map**过程和核心的**Mapreduce**过程合并为一个链式的**Mapreduce**，从而完成整个作业，这就是链式计算。简单来说，链式计算就是前面用多个**Mapper**处理任务，最后用一个**Reducer**输出结果。

7.5 MapReduce的特性

7.5.1 计数器

计数器是用来记录**Job**的执行进度和状态的，它的作用可以理解为日志。用户可以在程序的某个位置插入计数器，记录数据或者进度的变化情况。计数器还可辅助诊断系统故障。如果需要将日志信息传输到**Map**或**Reduce**任务，通常是尝试传输计数器值以监测某一特定事件是否发生。对于大型分布式作业而言，使用计数器更为方便。首先，获取计数器值比输出日志更方便；其次，根据计数器值统计特定事件的发生次数要比分析一堆日志文件容易得多。

7.5.1.1 内置计数器

1. **MapReduce**任务计数器
2. 文件系统计数器
3. **FileInputFormat**计数器
4. **FileOutputFormat**计数器
5. **Job**计数器

7.5.1.2 自定义计数器

1. 枚举类型定义计数器
2. 字符串类型定义计数器

7.5.2 连接

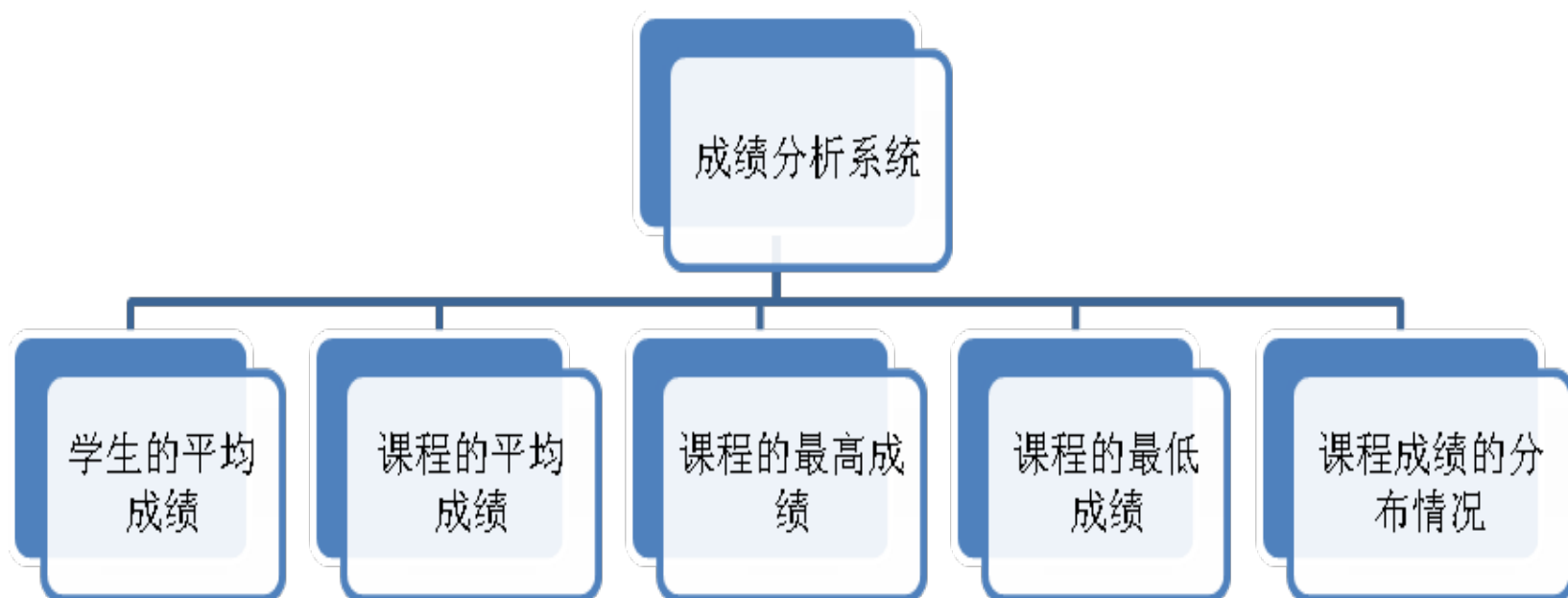
- 在关系型数据库中连接（**join**）是非常常见的操作。在海量数据的环境下，不可避免的也会遇到这种类型的需求。除了编写**MapReduce**程序，现在很多构建于**Hadoop**之上的应用，如**Hive**，**PIG**等在其内部都实现了**join**程序，用户可以通过很简单的**sql**语句或者数据操控脚本完成相应的**join**工作。
- 连接操作如果是由**Mapper**执行，则称为“**map**端连接”；如果由**Reducer**执行，则称为“**reduce**端连接”。

7.6 MapReduce应用举例——成绩分析系统的实现

7.6.1 成绩分析系统解析

- 成绩管理系统应用非常广泛，但基本上都是基于关系型数据库进行实现。若现在已有学生各科成绩汇总的文本文件，如何对这些数据进行分析？
- 创建数据库，将大量的数据手动添加到表中再通过**SQL**语句分析？
- 显然这是不明智的选择。此时可以选择文本分析工具或**MapReduce**程序实现分析功能。

7.6.2 成绩分析系统功能设计



7.6.3 成绩分析系统实现

1. 学生的平均成绩

- (1) 确定输入格式。采用默认的**TextInputFormat**。
- (2) 确定输出格式。这里为了防止有重名的学生，输出平均成绩时同时输出学号和姓名。对于要输出三个值，可以使用字符串拼接的方式和自定义数据类型两种方法。这里采用自定义数据类型。
- (3) 创建**StudentWritable**类，定义**StudentWritable**数据类型。
- (4) 创建**StudentMapper**类，实现**Mapper**类。
- (5) 创建**StudentAverager**类，实现主函数。
- (6) 准备需要处理的数据。
- (7) 运行。

```
1 import java.io.DataInput;
2 import java.io.DataOutput;
3 import java.io.IOException;
4 import org.apache.hadoop.io.WritableComparable;
5 public class StudentWritable implements WritableComparable<StudentWritable>
6 {
7     String id, name;
8     public StudentWritable() //无参数构造函数 {
9         super();
10    }
11    public StudentWritable(String id, String name)
12    {
13        super();
14        //将传递进来的String值转换为Long类型
15        this.id = id;
16        this.name = name;
17    }
18    public void write(DataOutput out) throws IOException
19    {
20        //序列化
21        out.writeUTF(id);
22        out.writeUTF(name);
23    }
24    public void readFields(DataInput in) throws IOException
25    {
26        //反序列化
27        this.id=in.readUTF();
28        this.name=in.readUTF();
29    }
30    public String toString()
31    {
32        return id + "\t" + name ;
33    }
34    public int compareTo(StudentWritable s) {
35        if(id.compareTo(s.id) > 0)
36            return 1;
37        else if(id.compareTo(s.id) < 0)
38            return -1;
39        else
40            return 0;
41    }
42 }
```

```
1 import java.io.IOException;
2
3 import org.apache.hadoop.io.FloatWritable;
4 import org.apache.hadoop.io.LongWritable;
5 import org.apache.hadoop.io.Text;
6 import org.apache.hadoop.mapreduce.Mapper;
7
8 public class StudentMapper extends Mapper<LongWritable, Text, StudentWritable,
FloatWritable>
9 {
10 public void map(LongWritable k1, Text v1, Context context)
11 throws IOException, InterruptedException
12 {
13     String line = v1.toString();
14     if (line.contains("id") == true)
15         return;
16     String[] splits = line.split("\t"); //按照制表符切割一行文本数据
17     if(splits.length != 4)
18         return;
19     //使用构造函数将id, name两个字段的值赋给StudentWritable类型对象s
20     StudentWritable s = new StudentWritable(splits[0], splits[1]);
21     int sum = Integer.parseInt(splits[2]) + Integer.parseInt(splits[3]);
22     float average = sum / 2;
23     context.write(s, new FloatWritable(average)); //将数据写入map上下文
24 }
25
```



代码



拆分



设计

标题:



检查页面

```
1 import java.io.IOException;
2
3 import org.apache.hadoop.conf.Configuration;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.FloatWritable;
6 import org.apache.hadoop.mapreduce.Job;
7 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
8 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
9 import org.apache.hadoop.util.GenericOptionsParser;
10
11 public class StudentAverage {
12
13     public static void main(String[] args) throws IOException, ClassNotFoundException,
14     InterruptedException {
15         Configuration conf = new Configuration();
16         String[] ortherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
17         //获取程序的输入参数
18         if(ortherArgs.length!=2) //如果未获得两个输入参数，输出提示信息并结束程序
19         {
20             System.err.println("Usage: StudentAverage <in> <out>");
21             System.exit(2);
22         }
23         Job job = Job.getInstance(conf, " StudentAverage");
24         job.setMapperClass(StudentMapper.class);
25         job.setOutputKeyClass(StudentWritable.class);
26         job.setOutputValueClass(FloatWritable.class);
27         FileInputFormat.addInputPath(job, new Path(ortherArgs[0]));
28         FileOutputFormat.setOutputPath(job, new Path(ortherArgs[1]));
29         System.exit(job.waitForCompletion(true)? 0:1);
30     }
31 }
```

7.6.3 成绩分析系统实现

2. 课程的最高成绩

- (1) 确定输入格式。采用默认的**TextInputFormat**。
- (2) 确定输出格式。输出课程名称与课程最高成绩，其中课程名称通过文件头部获取。
- (3) 创建**CourseMapper**类，实现**Mapper**类。
- (4) 创建**CourseReducer**类，实现**Reducer**类。
- (5) 创建**CourseMax**类，实现主函数。
- (6) 修改输出路径。
- (7) 运行。



代码



拆分



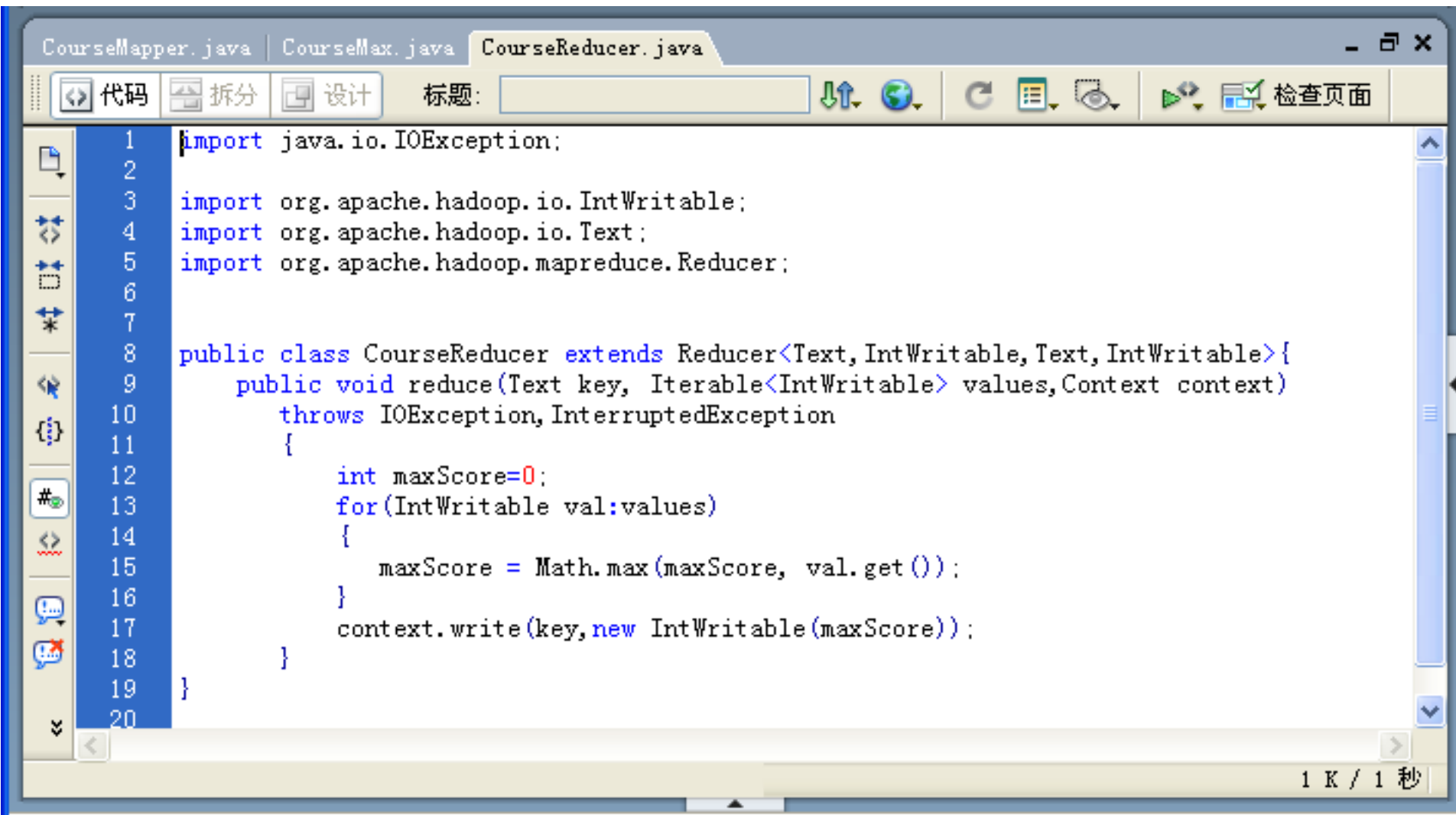
设计

标题:



检查页面

```
1 import java.io.IOException;
2 import org.apache.hadoop.io.IntWritable;
3 import org.apache.hadoop.io.LongWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Mapper;
6
7 public class CourseMapper extends Mapper<LongWritable, Text, Text, IntWritable>
8 {
9     static Text courseName1 = new Text();
10    static Text courseName2 = new Text();
11    public void map(LongWritable k1, Text v1, Context context)
12    throws IOException, InterruptedException
13    {
14        String[] splits = v1.toString().split("\\t"); //按照制表符切割一行文本数据
15        if(splits.length != 4)
16            return;
17        if(v1.toString().contains("id"))
18        {
19            courseName1.set(splits[2]);
20            courseName2.set(splits[3]);
21        }
22        else
23        {
24            int score1 = Integer.parseInt(splits[2]);
25            int score2 = Integer.parseInt(splits[3]);
26            context.write(courseName1, new IntWritable(score1));
27            context.write(courseName2, new IntWritable(score2));
28        }
29    }
30 }
```

The screenshot shows an IDE window with three tabs: CourseMapper.java, CourseMax.java, and CourseReducer.java. The CourseReducer.java tab is active, displaying the following Java code:

```
1 import java.io.IOException;
2
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Reducer;
6
7
8 public class CourseReducer extends Reducer<Text, IntWritable, Text, IntWritable>{
9     public void reduce(Text key, Iterable<IntWritable> values, Context context)
10         throws IOException, InterruptedException
11     {
12         int maxScore=0;
13         for(IntWritable val:values)
14         {
15             maxScore = Math.max(maxScore, val.get());
16         }
17         context.write(key, new IntWritable(maxScore));
18     }
19 }
20
```

The IDE interface includes a toolbar with icons for code, split, design, and other functions, and a status bar at the bottom indicating 1 K / 1 秒.



代码



拆分



设计

标题:



检查页面

```
1 import java.io.IOException;
2
3 import org.apache.hadoop.conf.Configuration;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Job;
8 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
10 import org.apache.hadoop.util.GenericOptionsParser;
11
12 public class CourseMax {
13
14     public static void main(String[] args) throws IOException, ClassNotFoundException,
15     InterruptedException {
16         Configuration conf = new Configuration();
17         String[] ortherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
18         if(ortherArgs.length!=2)
19         {
20             System.err.println("Usage:CourseMax <in> <out>");
21             System.exit(2);
22         }
23         Job job =Job.getInstance(conf, "CourseMax");
24         job.setJarByClass(CourseMax.class);
25         job.setMapperClass(CourseMapper.class);
26         job.setReducerClass(CourseReducer.class);
27         job.setOutputKeyClass(Text.class);
28         job.setOutputValueClass(IntWritable.class);
29         FileInputFormat.addInputPath(job, new Path(ortherArgs[0]));
30         FileOutputFormat.setOutputPath(job, new Path(ortherArgs[1]));
31         System.exit(job.waitForCompletion(true)?0:1);
32     }
33 }
```

7.6.3 成绩分析系统实现

3. 课程成绩分布情况

- (1) 确定输入格式。采用默认的**TextInputFormat**。
- (2) 确定输出格式。使用课程名称作为文件名，自定义输出。
- (3) 实现**Mapper**类。由于是通过**TextInputFormat**获得数据，获得课程名称与课程成绩的方法与前面的使用**CourseMapper**作为**Mapper**类即可。
- (4) 创建**DistributedPartitioner**类，实现**Partitioner**类。
- (5) 创建**DistributedReducer**类，实现**Reducer**类。
- (6) 创建自定义输出类**CourseNameOutputFormat**，实现指定名称的多文件输出。
- (7) 创建**ScoreDistributed**类，实现主函数。
- (8) 修改输出路径。
- (9) 运行。



代码



拆分



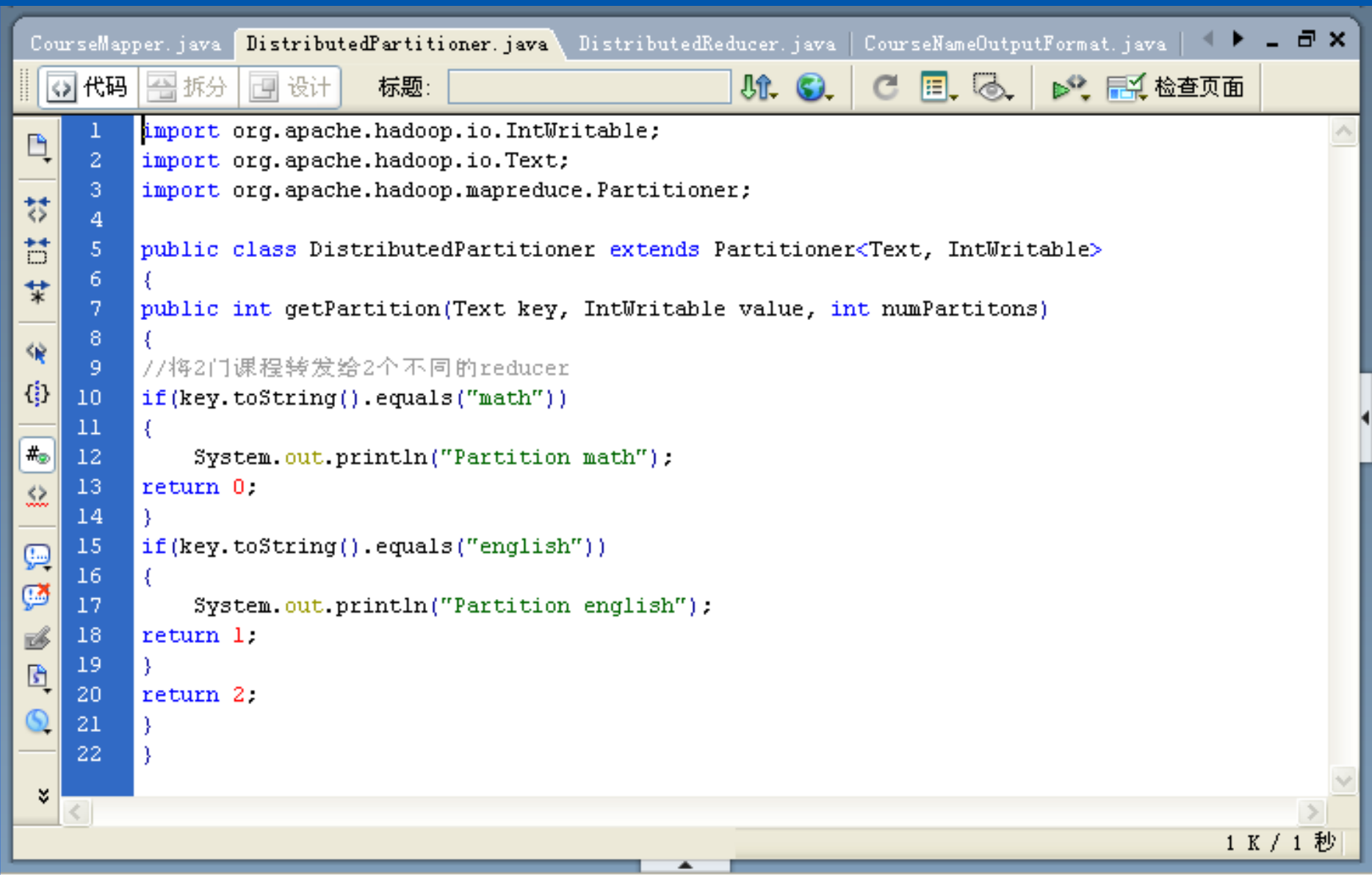
设计

标题:



检查页面

```
1 import java.io.IOException;
2 import org.apache.hadoop.io.IntWritable;
3 import org.apache.hadoop.io.LongWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Mapper;
6
7 public class CourseMapper extends Mapper<LongWritable, Text, Text, IntWritable>
8 {
9     static Text courseName1 = new Text();
10    static Text courseName2 = new Text();
11    public void map(LongWritable k1, Text v1, Context context)
12    throws IOException, InterruptedException
13    {
14        String[] splits = v1.toString().split("\\t"); //按照制表符切割一行文本数据
15        if(splits.length != 4)
16            return;
17        if(v1.toString().contains("id"))
18        {
19            courseName1.set(splits[2]);
20            courseName2.set(splits[3]);
21        }
22        else
23        {
24            int score1 = Integer.parseInt(splits[2]);
25            int score2 = Integer.parseInt(splits[3]);
26            context.write(courseName1, new IntWritable(score1));
27            context.write(courseName2, new IntWritable(score2));
28            System.out.println(courseName1.toString()+score1);
29            System.out.println(courseName2.toString()+score2);
30        }
31    }
32 }
33 }
34
```



```
CourseMapper.java DistributedPartitioner.java DistributedReducer.java CourseNameOutputFormat.java
<> 代码 拆分 设计 标题: 检查页面

1 import org.apache.hadoop.io.IntWritable;
2 import org.apache.hadoop.io.Text;
3 import org.apache.hadoop.mapreduce.Partitioner;
4
5 public class DistributedPartitioner extends Partitioner<Text, IntWritable>
6 {
7     public int getPartition(Text key, IntWritable value, int numPartitons)
8     {
9         //将2门课程转发给2个不同的reducer
10        if(key.toString().equals("math"))
11        {
12            System.out.println("Partition math");
13            return 0;
14        }
15        if(key.toString().equals("english"))
16        {
17            System.out.println("Partition english");
18            return 1;
19        }
20        return 2;
21    }
22 }
```

1 K / 1 秒



代码



拆分



设计

标题:



检查页面

```
1 import java.io.IOException;
2
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Reducer;
6
7 public class DistributedReducer extends Reducer<Text, IntWritable, Text, IntWritable>
8 {
9     public void reduce(Text key, Iterable<IntWritable> values, Context context)
10     throws IOException, InterruptedException
11     {
12         int a = 0, b = 0, c = 0, d = 0;
13         for(IntWritable val: values)
14         {
15             if(val.get() >= 90)
16                 a++;
17             else if(val.get() >= 80)
18                 b++;
19             else if(val.get() >= 60)
20                 c++;
21             else
22                 d++;
23         }
24         context.write(new Text("参加"+key+"课程考试总人数: "), new IntWritable(a+b+c+d));
25         context.write(new Text("成绩90分及以上的人数:"), new IntWritable(a));
26         context.write(new Text("成绩80-89分的人数:"), new IntWritable(b));
27         context.write(new Text("成绩60-79分的人数:"), new IntWritable(c));
28         context.write(new Text("成绩60分以下人数:"), new IntWritable(d));
29         System.out.println(key+"abc");
30     }
31 }
32
```



代码



拆分



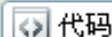
设计

标题:



检查页面

```
1 import java.io.IOException;
2 import org.apache.hadoop.fs.FSDataOutputStream;
3 import org.apache.hadoop.fs.FileSystem;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.RecordWriter;
8 import org.apache.hadoop.mapreduce.TaskAttemptContext;
9 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
10
11
12
13 public class CourseNameOutputFormat extends FileOutputFormat<Text, IntWritable>
14 {
15     public RecordWriter<Text, IntWritable> getRecordWriter(TaskAttemptContext job)
16         throws IOException, InterruptedException
17     {
18
19         Path outputDir = FileOutputFormat.getOutputPath(job);
20         // Path path1 = new Path(outputDir.toString()+"/math.txt");
21         // FileSystem fs = FileSystem.newInstance(job.getConfiguration());
22         String reduceId = job.getTaskAttemptID().getTaskID().toString();
23         FileSystem fs=outputDir.getFileSystem(job.getConfiguration());
24         //自定义输出路径及文件名, 把数学成绩和英语成绩分别输出到不同的文件中
25         if(reduceId.contains("r_000000"))
26         {
27             FSDataOutputStream course1= fs.create(new Path(outputDir.toString()+"/math.txt"));
28             return new CourseRecordWriter(course1);
29         }
30         else if(reduceId.contains("r_000001"))
31         {
32             FSDataOutputStream course2 =fs.create(new Path(outputDir.toString()+"/english.txt"));
33             return new CourseRecordWriter(course2);
34         }
35         return new CourseRecordWriter();
36     }
37 }
```



代码



拆分



设计

标题:



检查页面

```
1 import org.apache.hadoop.conf.Configuration;
2 import org.apache.hadoop.fs.Path;
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Job;
6 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
7 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
8 import org.apache.hadoop.util.GenericOptionsParser;
9
10 public class ScoreDistributed {
11     public static void main(String[] args) throws Throwable
12     {
13         Configuration conf = new Configuration();
14         String[] ortherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
15         //获取程序的输入参数
16         if(ortherArgs.length!=2) //如果未获得两个输入参数，输出提示信息并结束程序
17         {
18             System.err.println("Usage: ScoreDistributed <in> <out>");
19             System.exit(2);
20         }
21         Job job =Job.getInstance(conf," ScoreDistributed");
22         job.setMapperClass(CourseMapper.class);
23         job.setReducerClass(DistributedReducer.class);
24         job.setPartitionerClass(DistributedPartitioner.class);
25         job.setNumReduceTasks(2); //设置2个reducer
26         job.setOutputFormatClass(CourseNameOutputFormat.class);
27         job.setOutputKeyClass(Text.class);
28         job.setOutputValueClass(IntWritable.class);
29         FileInputFormat.addInputPath(job,new Path(ortherArgs[0]));
30         FileOutputFormat.setOutputPath(job,new Path(ortherArgs[1]));
31         System.exit(job.waitForCompletion(true)? 0:1);
32     }
33 }
```