第5章

MapReduce的介绍和简单使用

MapReduce是Hadoop上原生的分布式计算框架，本章主要对MapReduce计算框架的原理和开发环境的搭建进行介绍。本章内容安排如下。

5.1 MapReduce简介

对MapReduce进行简单的介绍。

5.2 Map过程

介绍MapReduce的Map过程。

5.3 Reduce过程

介绍MapReduce的Reduce过程。

5.4 开发环境的搭建

介绍如何搭建使用MapReduce需要的环境。

5.5 实验

描述一些简单的MapReduce实验。

通过本章的学习，读者将对MapReduce有初步的了解，对Map和Reduce过程的原理有更清晰的认识，同时通过对MapReduce环境的搭建和简单的程序示例，能够加深读者对MapReduce工作原理的认识。

5.1 MapReduce简介

MapReduce（MR）是现今一个非常流行的分布式计算框架，它被设计用于并行计算海量数据，通常是存储在HDFS上TB级和PB级别的数据。其前身是Google公司的MapReduce。MapReduce框架将复杂的大规模并行计算高度抽象为两个函数：Map函

数和 Reduce 函数。Map(映射)和 Reduce(归约)以及其主要思想都是从函数式编程语言中借鉴过来的。Map 负责把作业分解为多个任务,Reduce 负责把分解后的多个任务处理的结果汇总起来。

当向 MapReduce 框架提交一个计算作业(Job)时,它会首先把计算作业拆分成若干个 Map 任务(Task),然后以完全并行的方式处理,分配到不同的节点上去执行,每一个 Map 任务处理输入数据中的一部分,当 Map 任务完成后,它会生成一些中间文件,这些中间文件将会作为 Reduce 任务的输入数据。简单地说,MapReduce 就是"任务的分解与结果的汇总"这样的一个过程。

在 Hadoop 中,用于执行 MapReduce 任务的对象有两个:JobTracker 和 TaskTracker。JobTracker 是用于调度工作的,一个 Hadoop 集群中只有一个 JobTracker,位于 Master 上。TaskTracker 用于执行工作,位于各个 Slave 上。

需要特别注意的是,MapReduce 处理的数据集必须可以分解成许多小数据集,而且每个小的数据集都可以完全独立地并行处理。

5.2 Map 过程

在第 4 章中已经介绍过,HDFS 存储数据是按块存储,每个块的大小默认为 128MB,而一个块为一个分片,一个 Map 任务处理一个分片,当然,也可以根据需要自主设置块的大小。Map 输出的结果会暂时放在一个环形内存缓冲区中(缓冲区默认大小为 100MB),当该缓冲区接近溢出时(默认为缓冲区大小的 80%),会在本地文件系统中创建一个新文件,将该缓冲区中的数据写入这个文件;在写入磁盘之前,首先根据 Reduce 任务的数目将数据划分为相同数目的分区,一个分区的中间数据对应一个 Reduce 任务。这样做是为了避免数据分配不均匀的情况。

当 Map 任务输出最后一个记录时,可能会有很多的溢出文件,这时需要将这些文件进行合并。合并的过程中会不断地进行排序和合并操作(即 combia 操作),这样做的目的有以下两个。

(1) 尽量减少每次写入磁盘的数据量。

(2) 尽量减少下一复制阶段网络传输的数据量。

最后合并完成后,会形成一个已分区且已排序的文件。为了减少网络传输的数据量,这里可以将数据进行压缩。

5.3 Reduce 过程

Reduce 会接收到不同 Map 任务传来的数据,并且每个 Map 传来的数据都是有序的。如果 Reduce 端接收的数据量相当小,则直接存储在内存中,如果数据量超过了该缓冲区大小的一定比例,则对数据合并后溢写到磁盘中。

随着溢写文件的增多，后台线程会将它们合并成一个更大的有序的文件，这样做是为了给后面的合并节省时间。其实，不管在 Map 端还是 Reduce 端，MapReduce 都在反复地执行排序、合并操作；合并的过程中会产生许多的中间文件（写入磁盘），但 MapReduce 会让写入磁盘的数据尽可能的少，并且最后一次合并的结果并没有写入磁盘，而是直接输入到 Reduce 函数。

所以，一个完整的 MapReduce 过程如图 5.1 所示。

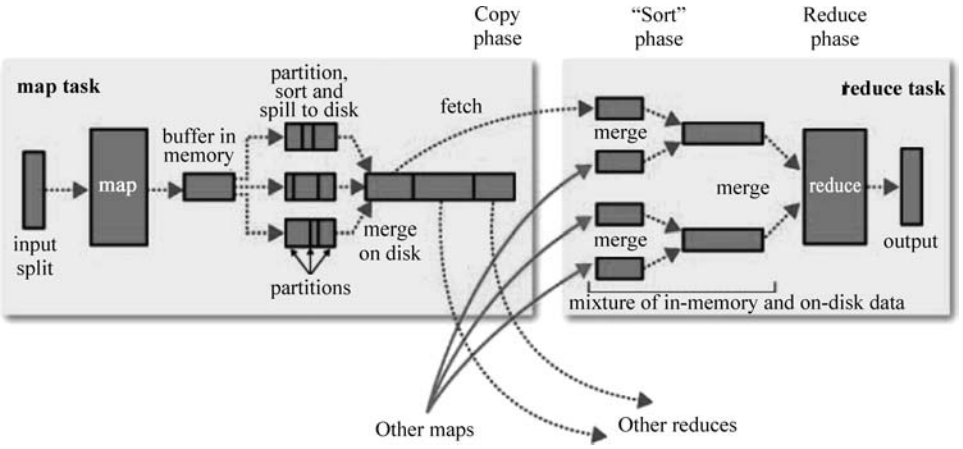

图 5.1　MapReduce 过程

该过程的流程说明如下。

1. Map 过程

（1）输入文件，InputFormat 产生键值对，并传送到 Mapper 类的 Map 函数中。

（2）Map 输出键值对到一个没有排序的缓冲内存中。

（3）当缓冲内存达到给定值或者 Map 完成时，就会对缓冲区内的键值对进行排序，然后溢写到磁盘上。

（4）如果有多个溢出文件，那么将这些文件整合到一个文件中，并且这些文件是经过排序的。

（5）在这些过程中，排序后的键值对等待 Reducer 获取。

2. Reduce 过程

（1）Reducer 获取 Mapper 的记录，作为输入。

（2）相同的 Key 被传入同一个 Reducer 中。

（3）当一个 Mapper 完成后，Reducer 就开始获取 Mapper 结果，所有溢出文件被排序后放到一个内存缓冲区。

（4）当内存缓冲区满后，就会产生溢出文件，存入本地磁盘。

（5）Reducer 中所有数据传输完成后，所有溢出文件被整合和排序。

（6）Reducer 将结果输出到 HDFS。

5.4 开发环境的搭建

在 Windows 环境下使用 MapReduce 进行实验前，需要搭建用于本地开发的 MapReduce 环境。本书使用的 IDE 为 Eclipse，因此在搭建 MapReduce 开发环境前需要安装 Eclipse 以及 Hadoop 插件。

本书在这里就不再对 Eclipse 的安装进行阐述了，读者可根据需要自行进行安装，接下来主要介绍 Hadoop 插件的安装。

(1) 下载 Hadoop 插件，将下载的插件存放到 Eclipse 的插件目录中，如图 5.2 所示。

图 5.2 保存插件到 Eclipse 的插件目录

(2) 删除 Eclipse 中 configuration 目录下的 update 文件夹，如图 5.3 所示，让 Eclipse 重新读取插件。

图 5.3 删除 update 文件夹

(3) 解压一份 Hadoop 插件文件到本地的磁盘，如图 5.4 所示。

图 5.4 解压 Hadoop 文件到本地

(4) 使用在 Windows 下编译 Hadoop 中的 bin 文件替换原本的 bin 文件，如图 5.5 所示。

图 5.5 替换 bin 文件

(5) 打开 Eclipse，在菜单栏中选择 Window→Preferences，如图 5.6 所示。

(6) 设置 Hadoop 文件的目录并添加环境变量，如图 5.7 所示。

(7) 设置 Hadoop 连接配置。在 Eclipse 菜单栏中选择 Window→Show View→Other→MapReduce Tools，如图 5.8 所示。

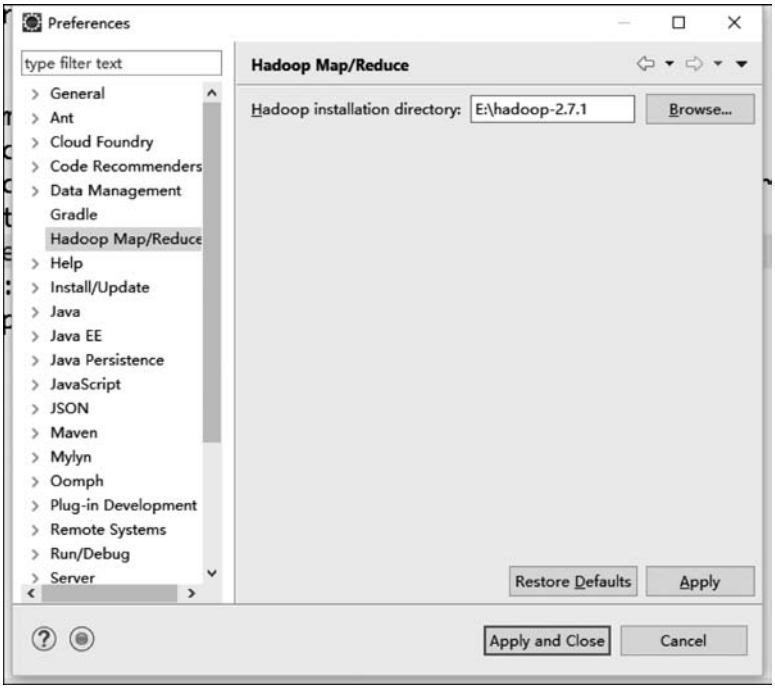

图 5.6　打开 Eclipse 的 Preferences 对话框

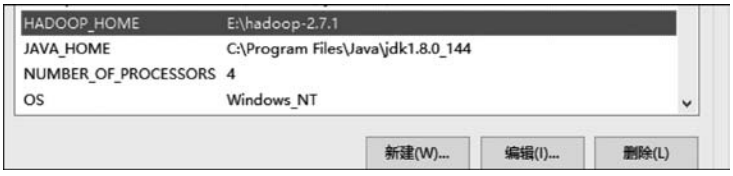

图 5.7　设置路径及环境变量

图 5.8　设置 Hadoop 连接配置

（8）在如图 5.9 所示的界面中单击右上角的“添加”按钮。然后按照如图 5.10 所示配置连接参数，单击“完成”按钮即可。

图 5.9　MapReduce Tool 窗口

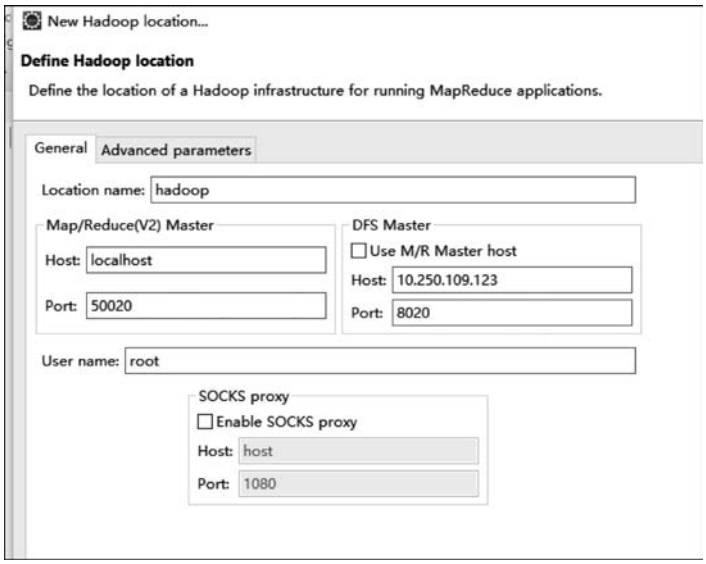

图 5.10　设置连接数据

（9）配置成功后，就可以在 Eclipse 的窗口中看到如图 5.11 所示的 Hadoop 连接窗口。

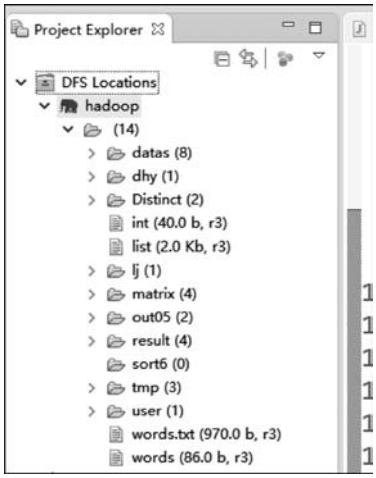

图 5.11　Hadoop 连接配置成功

5.5 实验

本节将使用几个简单的实验来加深读者对 MapReduce 工作原理的理解。

5.5.1 单词计数

本节将从项目的创建开始，向读者展示单词计数实验的整个操作过程，在后面的实验中，将只给出源码，其余操作请读者参考本节内容。

(1) 新建一个 Map/Reduce 项目，如图 5.12 所示。

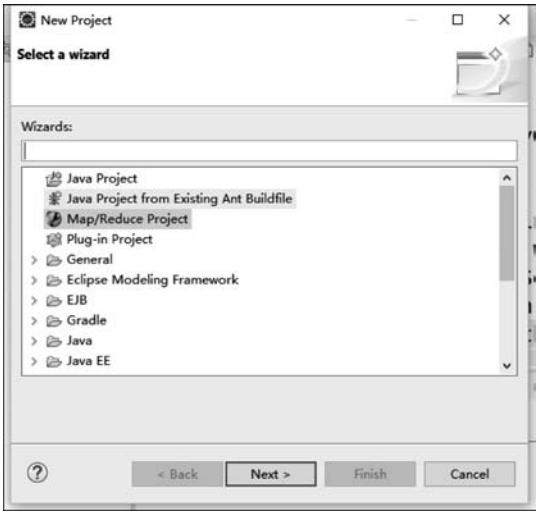

图 5.12 新建 Map/Reduce 项目

(2) 将项目命名为“WordCounter”，如图 5.13 所示。

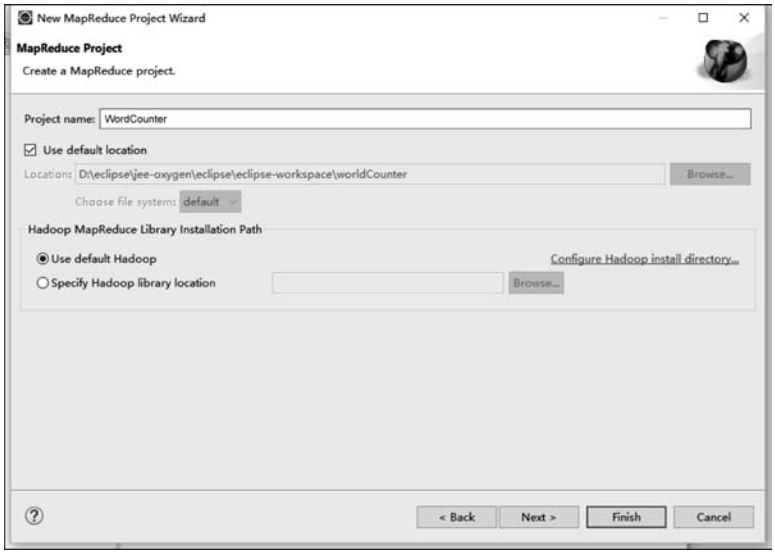

图 5.13 给项目命名

(3) 配置项目内容,如图 5.14 所示,单击 Finish 按钮,项目创建完成。

图 5.14 配置项目内容

(4) 在项目中新建一个类,输入以下代码。

```
package word;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCountApp {
    public static class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
        private Text word = new Text();
```

```
        private IntWritable one = new IntWritable(1);
        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
                throws IOException, InterruptedException {
            // 得到输入的每一行数据
            String line = value.toString();
            // 分割数据,通过空格来分割
            String[] words = line.split("_");
            // 循环遍历并输出
            for(String w :words) {
                word.set(w);
                context.write(word, one);
            }
        }
    }
    public static class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable sum = new IntWritable();

        @Override
        protected void reduce(Text key, Iterable<IntWritable> values,
                Reducer<Text, IntWritable, Text, IntWritable>.Context content)
                throws IOException, InterruptedException {
            Integer count = 0;
            for(IntWritable value :values) {
                count += value.get();
            }
            sum.set(count);
            content.write(key, sum);
        }
    }

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

        if(args.length < 2) {
            args = new String[]{
                    "hdfs://10.250.109.123:8020/words",
                    "hdfs://10.250.109.123:8020/out05"
            };
        }
        // 创建配置对象
        Configuration conf = new Configuration();
        // 创建 job 对象
        Job job = Job.getInstance(conf, "wordcount");
        // 设置运行 job 的主类
        job.setJarByClass(WordCountApp.class);
        // 设置 mapper 类
        job.setMapperClass(MyMapper.class);
        // 设置 reducer 类
```

```
        job.setReducerClass(MyReducer.class);
        // 设置 mapper 输出的 key value
        job.setMapOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        // 设置 reducer 输出的 key value 类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        // 设置输入的路径
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 提交 job
        boolean b = job.waitForCompletion(true);
        if(!b) {
            System.err.println("This task has failed!!!");
        }
    }
}
```

(5) 通过文件或者直接输入的方式将数据文件上传到 Hadoop，如图 5.15 所示，右键上传数据文件。本书使用的实验数据如图 5.16 所示。

图 5.15　上传数据

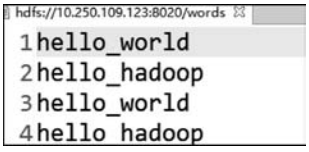

图 5.16　实验数据

(6) 运行程序，如图 5.17 所示，右击“类”，选择 Run As→1 Java Application 命令。

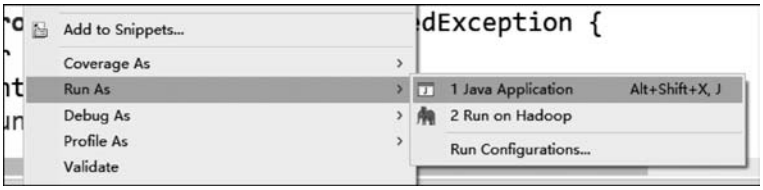

图 5.17　运行程序

（7）运行完成后，打开如图 5.18 所示的文件，查看运行结果。

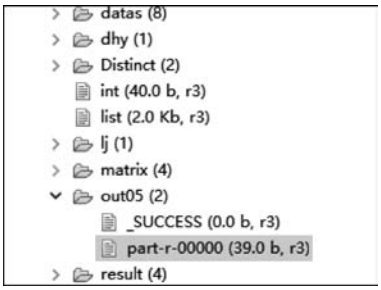

图 5.18　查看运行结果

5.5.2　二次排序实验

（1）新建一个项目。

（2）在项目中，新建一个类，这里命名为“IntPair”，类的实现代码如下。

```
package expBigData.MapReduce.SecondarySort;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.WritableComparable;

public class IntPair implements WritableComparable<IntPair> {
    private IntWritable first;
    private IntWritable second;

    public void set(IntWritable first, IntWritable second) {
        this.first = first;
        this.second = second;
    }

    //注意：需要添加无参的构造方法，否则反射时会报错
    public IntPair() {
        set(new IntWritable(), new IntWritable());
    }

    public IntPair(int first, int second) {
        set(new IntWritable(first), new IntWritable(second));
    }

    public IntPair(IntWritable first, IntWritable second) {
        set(first, second);
    }
```

```
    //其他成员函数
    public IntWritable getFirst() {
        return first;
    }

    public void setFirst(IntWritable first) {
        this.first = first;
    }

    public IntWritable getSecond() {
        return second;
    }

    public void setSecond(IntWritable second) {
        this.second = second;
    }

    public void write(DataOutput out) throws IOException {
        first.write(out);
        second.write(out);
    }

    public void readFields(DataInput in) throws IOException {
        first.readFields(in);
        second.readFields(in);
    }

    public int hashCode() {
        return first.hashCode() * 163 + second.hashCode();
    }

    public boolean equals(Object o) {
        if(o instanceof IntPair) {
            IntPair tp = (IntPair) o;
            return first.equals(tp.first) && second.equals(tp.second);
        }
        return false;
    }

    public String toString() {
        return first + "\\t" + second;
    }

    public int compareTo(IntPair tp) {
        int cmp = first.compareTo(tp.first);
        if(cmp != 0) {
            return cmp;
        }
        return second.compareTo(tp.second);
    }
}
```

（3）再新建一个 Java 文件，命名为“SecondarySort”，在该文件中编写主程序，主程序代码如下。

```
package expBigData.MapReduce.SecondarySort;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SecondarySort {
    static class TheMapper extends Mapper<LongWritable, Text, IntPair, NullWritable> {
        @Override
        protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String[] fields = value.toString().split("\\t");
            int field1 = Integer.parseInt(fields[0]);
            int field2 = Integer.parseInt(fields[1]);
            context.write(new IntPair(field1, field2), NullWritable.get());
        }
    }
    static class TheReduce extends Reducer<IntPair, NullWritable, IntPair, NullWritable> {
        // private static final Text SEPARATOR = new Text("------------------------");
        @Override
        protected void reduce(IntPair key, Iterable<NullWritable> values, Context
context)
                throws IOException, InterruptedException {
            context.write(key, NullWritable.get());
        }
    }

    public static class FirstPartitioner extends Partitioner<IntPair, NullWritable> {
        public int getPartition(IntPair key, NullWritable value, int numPartitions) {
            return Math.abs(key.getFirst().get()) % numPartitions;
```

```
        }
    }

    //如果不添加这个类,默认第一列和第二列是升序排序的
    //这个类的作用是使第一列升序排序,第二列降序排序
    public static class KeyComparator extends WritableComparator {
        // 必须加上无参构造器,否则报错
        protected KeyComparator() {
            super(IntPair.class, true);
        }

        public int compare(WritableComparable a, WritableComparable b) {
            IntPair ip1 = (IntPair) a;
            IntPair ip2 = (IntPair) b;
            // 第一列按升序排列
            int cmp = ip1.getFirst().compareTo(ip2.getFirst());
            if(cmp != 0) {
                return cmp;
            }
            // 在第一列相等的情况下,第二列按降序排序
            return -ip1.getSecond().compareTo(ip2.getSecond());
        }
    }

    //入口程序
    public static void main(String[] args) throws Exception {
        if(args.length < 2) {
            args = new String[] { "hdfs://10.250.109.123:8020/dhy/in/secondsort.txt",
                    "hdfs://10.250.109.123:8020/dhy/out/secondarysort_out00" };
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        job.setJarByClass(SecondarySort.class);
        // 设置 mapper 的相关属性
        job.setMapperClass(TheMapper.class);
        // 当 mapper 中的输出 key 和 value 类型和 reducer 中的相同时,以下两句省略
        job.setMapOutputKeyClass(IntPair.class);
        job.setMapOutputValueClass(NullWritable.class);
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        // 设置分区相关属性
        job.setPartitionerClass(FirstPartitioner.class);
        // 在 mapper 中对 key 进行排序
        job.setSortComparatorClass(KeyComparator.class);
        // job.setSortGroupComparatorClass(GroupComparator.class);
        // 设置 reducer 的相关属性
        job.setReducerClass(TheReduce.class);
```

```
        job.setOutputKeyClass(IntPair.class);
        job.setOutputValueClass(NullWritable.class);
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 设置 reducer 数量
        int reduceNum = 1;
        if(args.length >= 3 && args[2] != null) {
            reduceNum = Integer.parseInt(args[2]);
        }
        job.setNumReduceTasks(reduceNum);
        job.waitForCompletion(true);
    }
}
```

5.5.3 计数器实验

(1) 新建一个项目。

(2) 在项目中编写主程序,代码如下。

```
package cn.cqu.wzl;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Counters {
    public static class MyCounterMap extends Mapper<LongWritable,Text,Text,Text>{
        public static org.apache.hadoop.mapreduce.Counter ct = null;
        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
Text>.Context context)
                throws IOException, InterruptedException {
            String arr_value[] = value.toString().split("\t");
            if(arr_value.length > 3) {
                ct = context.getCounter("ERRorCounter", "toolong");
                System.out.println("toolong");
                ct.increment(1);
            }else if(arr_value.length < 3) {
                ct = context.getCounter("ERRorCounter", "tooshort");
                System.out.println("tooshort");
```

```
                ct.increment(1);

            }

        }

    }
    public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        if(args.length < 2) {
            args = new String[] {
                    "hdfs://10.250.109.123:8020/datas/counters",
                    "hdfs://10.250.109.123:8020/result/counters"
            };
        }

        Configuration conf = new Configuration();
        Job job = new Job(conf,"Counter");
        job.setMapperClass(MyCounterMap.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```