**基于分布式计算的招聘网站职位分析**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 成文信息 | | | | | | | |
| 主题词： | | 详细设计 | | | | | |
| 作 者： | |  | | 文档类别： | | | 技术方案、设计文档 |
| 审 核： | |  | |
| 批 准： | |  | | 文档性质： | | | 正式稿 |
| 主 送： | |  | | 存档日期： | | |  |
| 抄 送： | |  | | 发布日期： | | |  |
| 签收信息 | | | | | | | |
| 发送方 | | | 接收方： | 接收方： | | 接收方： | |
| 代表人： | | | 代表人： | 代表人： | | 代表人： | |
| 日期： | | | 日期： | 日期： | | 日期： | |
| 变更信息 | | | | | | | |
| 版本 | 原因 | | | | 作者 | 日期 | |

目录

[第一章 引言 1](#_Toc433639044)

[1.1 编写目的 1](#_Toc433639045)

[1.2 背景 1](#_Toc433639046)

[1.3 定义 1](#_Toc433639047)

[1.4 列出有关的参考资料 1](#_Toc433639048)

[第二章 程序系统结构 2](#_Toc433639049)

[2.1 体系架构 2](#_Toc433639050)

[2.2 程序结构设计 3](#_Toc433639051)

[2.2.1 名称设计 3](#_Toc433639052)

[2.2.2 目录规范 3](#_Toc433639053)

[2.2.3 配置文件规划 5](#_Toc433639054)

[第三章 饼状图模块设计说明 6](#_Toc433639055)

[3.1.1 模块描述 6](#_Toc433639056)

[3.1.2 功能 6](#_Toc433639057)

[3.1.3 性能 7](#_Toc433639058)

[3.1.4 类设计 7](#_Toc433639059)

[3.1.5 算法 10](#_Toc433639060)

[第四章 柱状图模块设计说明 10](#_Toc433639061)

[4.1.1 模块描述 10](#_Toc433639062)

[4.1.2 功能 11](#_Toc433639063)

[4.1.3 人机交互 11](#_Toc433639064)

[4.1.4 类设计 14](#_Toc433639065)

[4.1.5 算法 15](#_Toc433639066)

[第五章 折线图模块设计说明 15](#_Toc433639067)

[5.1.1 模块描述 15](#_Toc433639068)

[5.1.2 功能 16](#_Toc433639069)

[5.1.3 人机交互 16](#_Toc433639070)

[5.1.4 类设计 19](#_Toc433639071)

[5.1.5 算法 20](#_Toc433639072)

# 引言

## 编写目的

在基于分布式计算程序的基础上，完成代码的结构设计，划分系统模块并明确模块之间的关系，同时给出相关的用户界面设计招聘网站职位分析详细设计。本文的预期读者为： 分布式程序的设计、开发人员。 工程管理软件的验收、评审人员。

## 背景

基于分布式的程序计算系统。

## 定义

1、平台：51job招聘平台。

2、SQL：Structured Query Language（结构化查询语言）。

# 程序系统结构

## 体系架构

从功能构成的角度出发划分系统的职能层次，按照从底层到顶层，服务到应用的划分方法，基于分布式计算的招聘网站职位分析可以划分为三个部分：上传文件到Hadoop集群,利用MapReduce计算结果，Echarts显示结果到Web页面

（一）上传文件到Hadoop集群

将本地文件利用java操作HDFS上传到Hadoop集群

1. 利用MapReduce计算结果

先利用MapReduce对文件所有内容进行计数，然后将计数的文件进行排序，取前几个数据

1. ECharts显示结果到Web页面

将计算不出来的结果利用ECharts打印在Web界面

## 程序结构设计

### 名称设计

基于分布式的招聘职位分析

### 目录规范

为了管理数据，避免因系统各组件目录重复导致的错误，故需要统一规划应用软件相关目录结构。

### 配置文件规划

都统一标准的放在固定的文件下。

# 饼状图模块设计说明

### 模块描述

本模块是将发布公司最多的Top 10 公司名称及发布职位数量制作成饼状图展示给用户观看。

### 功能

展示公司最多的Top 10 公司名称及发布职位数量。

#### 精度

每个文本字符串必须严格按照：招聘职位,公司,地址,薪资,日期。格式写入文件中，否则将会被当成脏数据处理掉。

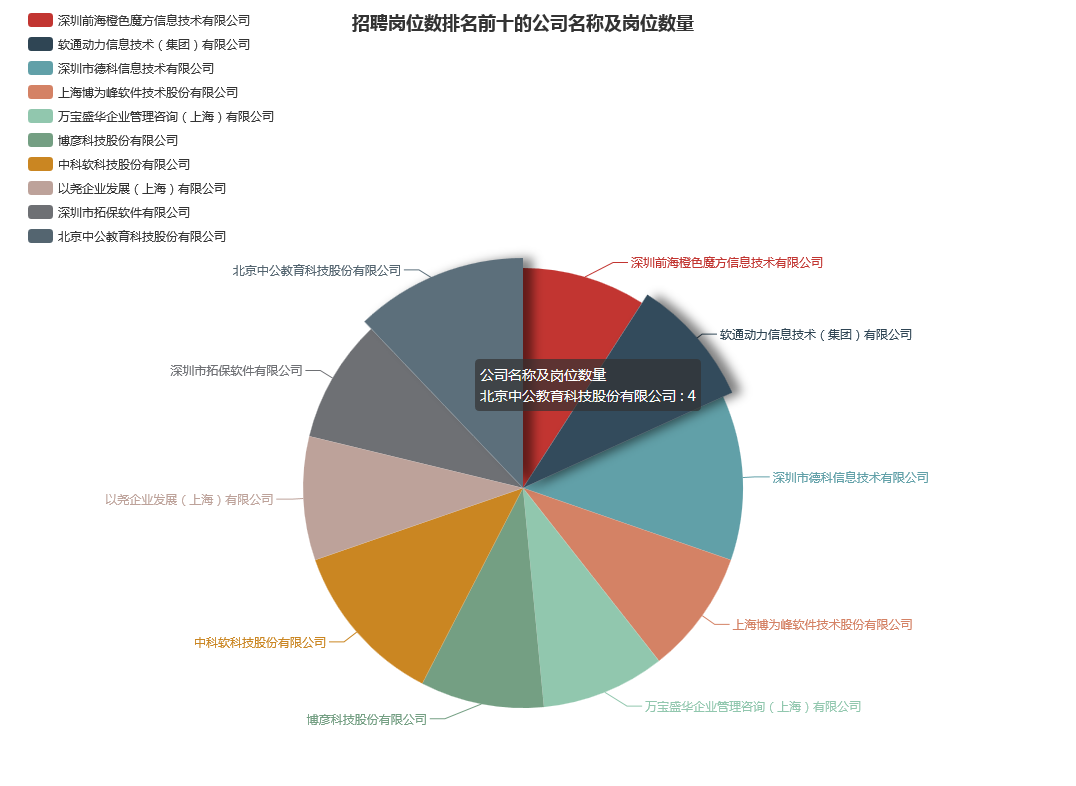
#### 输入项

将文件上传至hadoop集群。

#### 输出项

将分析发布公司最多的Top 10 公司名称及发布职位数量写入到文件中。

#### 



### 类设计

给出本模块的类设计，包括**类说明**

一个完整的 mapreduce 程序在分布式运行时有两类实例进程：

(1) MRAppMaster：负责整个程序的过程调度及状态协调 (该进程在yarn节点上)

(2) Yarnchild：负责 map 阶段的整个数据处理流程

(3) Yarnchild：负责 reduce 阶段的整个数据处理流程

以上两个阶段 maptask 和 reducetask 的进程都是 yarnchild，并不是说这 maptask 和 reducetask 就跑在同一个 yarnchild 进行里

（Yarnchild进程在运行该命令的节点上）

#### 类说明

HDFS：将文件上传到hadoop集群。

（1）、Reduce类：reduce 函数也有四个形式参数类型用于指定输入和输出类型。

reduce函数的输入类型必须匹配map函数的输出类型。即Text类型和IntWritable 类型。

1. 、Master类：是程序代码的核心，用于设置MapReduce和其他组件。
2. 、Map类：将读取到的信息进行切片，并且标记出现过的所有信息。

### 算法

将文件上传到hadoop集群：

public class HDFSTest {

private static Configuration conf=new Configuration();

private static final String HADOOP\_URL="hdfs://etc01:8020";

private static FileSystem fs;

static {

FileSystem.setDefaultUri(conf,HADOOP\_URL);

try {

fs=FileSystem.get(conf);

} catch (IOException e) {

e.printStackTrace();

}

}

public static Map<String,String> read(String s)

{

Map<String,String> map= new HashMap<>();

Path path=new Path(s);

try {

FSDataInputStream dataInputStream=fs.open(path);

InputStreamReader inputStreamReader=new InputStreamReader(dataInputStream,"UTF-8");

BufferedReader bufferedReader=new BufferedReader(inputStreamReader);

String sum=" ";

while ((sum=bufferedReader.readLine())!=null)

{

String []k=sum.split(" ");

map.put(k[0], k[1]);

}

} catch (IOException e) {

e.printStackTrace();

}

return map;

}

}

将文件中的信息排序：

Master：

public class Recruit {

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

Path inputPath=new Path("/document/600.csv");

Path outPath=new Path("/document/700.csv");

Configuration conf=new Configuration();

FileSystem fs=FileSystem.get(conf);

Job job=Job.getInstance(conf,"recruit");

job.setJarByClass(Recruit.class);

job.setMapperClass(Recruitmapper.class);

job.setReducerClass(Recruitreducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.setInputPaths(job,inputPath);

FileOutputFormat.setOutputPath(job,outPath);

if(fs.exists(outPath)){

fs.delete(outPath,true);

}

boolean result=job.waitForCompletion(true);

if (result){

System.out.println("成功！！！");

}

}

}

Map：

publicclassRecruitmapper extends Mapper<LongWritable,Text,Text,IntWritable> {

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String data=value.toString();

String[] temp=data.split(",");

for(int i=0;i<temp.length-4;i++){

if(temp.length>5){

}

else {

context.write(new Text(temp[1]), new IntWritable(1));

}

}

}

}

Reduce：

public class Recruitreducer extends Reducer<Text,IntWritable,Text,IntWritable> {

@Override

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

int count=0;

for (IntWritable value:values){

count+=value.get();

}

context.write(key,new IntWritable(count));

}

}

将文件中所需要的信息筛选出来：

Master：

public class WordCountMaster {

public void test() throws IOException, ClassNotFoundException, InterruptedException {

//输入路径文件

Path inputPath = new Path("/input/51Job\_python\_5000.txt");

//输出路径文件

Path outputPath = new Path("/output/test");

//初始化配置

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(conf);

//初始化Job参数，指定Job名称

Job job = Job.getInstance(conf,"wordcount");

//设置执行的Job类

job.setJarByClass(WordCountMaster.class);

//设置Mapper类

job.setMapperClass(WordCountMapper.class);

//设置Reducer类

job.setReducerClass(WordCountReducer.class);

//设置Map的输出类型

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

//设置Reducer输出类型

job.setOutputKeyClass(IntWritable.class);

job.setOutputValueClass(Text.class);

//设置输入路径

FileInputFormat.setInputPaths(job,inputPath);

//设置输出路径

FileOutputFormat.setOutputPath(job,outputPath);

//提交Job

if (fs.exists(outputPath)) {

fs.delete(outputPath,true);

}

job.waitForCompletion(true);

}

public void test1() throws IOException, ClassNotFoundException, InterruptedException{

//输入路径文件

Path inputPath = new Path("/output/test/part-r-00000");

//输出路径文件

Path outputPath = new Path("/output/test1");

//初始化配置

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(conf);

//初始化Job参数，指定Job名称

Job job = Job.getInstance(conf,"wordcount");

//设置执行的Job类

job.setJarByClass(WordCountMaster.class);

//设置Mapper类

job.setMapperClass(DataMapper.class);

//设置Reducer类

job.setReducerClass(DataReducer.class);

//设置Map的输出类型

job.setMapOutputKeyClass(ReLongWritable.class);

job.setMapOutputValueClass(Text.class);

//设置Reducer输出类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(ReLongWritable.class);

//设置输入路径

FileInputFormat.setInputPaths(job,inputPath);

//设置输出路径

FileOutputFormat.setOutputPath(job,outputPath);

//提交Job

if (fs.exists(outputPath)) {

fs.delete(outputPath,true);

}

job.waitForCompletion(true);

}

}

Map：

public class WordCountMapper extends Mapper<LongWritable,Text,Text,IntWritable> {

//LongWritable 对应是长整形（行号）

//Text 对应字符串

//IntWritable 对应正常的整形（每个单词个数）

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

//value 代表读取到的当前行的数据

String data = value.toString();

//通过逗号分隔

String[] temp = data.split(",");

//将每个公司标记为已经出现一次

if(temp.length>1)

context.write(new Text(temp[1]),new IntWritable(1));

}

}

Reduce：

public class WordCountReducer extends Reducer<Text,IntWritable,IntWritable,Text> {

@Override

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

int count = 0;

//迭代计算，将元组中的值累加，得到该公司总共出现的数量

for(IntWritable value:values){

count += value.get();

}

//结果输出，会生成一个数据文件

key.set("&&"+key.toString());

context.write(new IntWritable(count),key);

}

}

# **柱状图模块设计说明**

### 模块描述

### 分析招聘职位最多的Top5 职位名称及数量功能，利用柱状图展示

#### 输入项

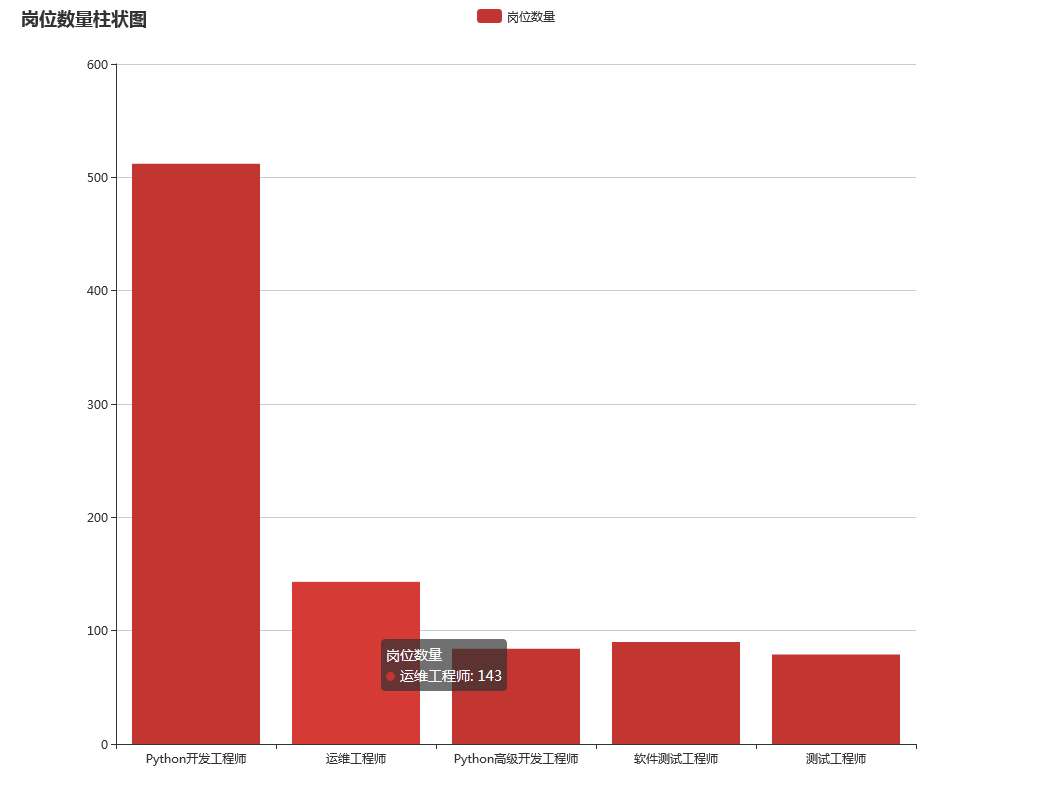
　　上传文件到Hadoop集群

#### 输出项

　　输出排序后的前5名结果

#### 界面设计

柱形图



### 类设计

1. LearnHdfs类:将文件上传到Hadoop集群
2. Comparator类:自定义排序算法
3. Reduce类：reduce 函数也有四个形式参数类型用于指定输入和输出类型。reduce函数的输入类型必须匹配map函数的输出类型。即Text类型和IntWritable 类型。
4. Master类：是程序代码的核心，用于设置MapReduce和其他组件。
5. Map类：将读取到的信息进行切片，并且标记出现过的所有信息。

### 算法

**上传文件：**

fs.copyFromLocalFile(new Path("E://PX.txt"),new Path("/a1/PX.txt"));

**JobMapper:**

public class DataMapper extends Mapper<LongWritable,Text,ReLongWritable,Text>{

@Override

protected void map(LongWritable key, Text value,Context context)

throws IOException, InterruptedException {

// TODO Auto-generated method stub

String data = value.toString();

//通过逗号分隔

String[] temp = data.split(" &&");

//将每个公司标记为已经出现一次

if(temp.length>1)

context.write(new ReLongWritable(Long.parseLong(temp[0])),new Text(temp[1]));

}

}**JobMaster:**

public class WordCountMaster1 {

public void test() throws IOException, ClassNotFoundException, InterruptedException {

//输入路径文件

Path inputPath = new Path("/input/51Job\_python\_5000.txt");

//输出路径文件

Path outputPath = new Path("/output/outdata1");

//初始化配置

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(conf);

//初始化Job参数，指定Job名称

Job job = Job.getInstance(conf,"wordcount");

//设置执行的Job类

job.setJarByClass(WordCountMaster1.class);

//设置Mapper类

job.setMapperClass(WordCountMapper1.class);

//设置Reducer类

job.setReducerClass(WordCountReducer1.class);

//设置Map的输出类型

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

//设置Reducer输出类型

job.setOutputKeyClass(IntWritable.class);

job.setOutputValueClass(Text.class);

//设置输入路径

FileInputFormat.setInputPaths(job,inputPath);

//设置输出路径

FileOutputFormat.setOutputPath(job,outputPath);

//提交Job

if (fs.exists(outputPath)) {

fs.delete(outputPath,true);

}

boolean result = job.waitForCompletion(true);

if(result)

System.out.println("程序成功执行");

}

public void test1() throws IOException, ClassNotFoundException, InterruptedException{

//输入路径文件

Path inputPath = new Path("/output/outdata1/part-r-00000");

//输出路径文件

Path outputPath = new Path("/output/outdata2");

//初始化配置

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(conf);

//初始化Job参数，指定Job名称

Job job = Job.getInstance(conf,"wordcount");

//设置执行的Job类

job.setJarByClass(WordCountMaster1.class);

//设置Mapper类

job.setMapperClass(DataMapper.class);

//设置Reducer类

job.setReducerClass(DataReducer1.class);

//设置Map的输出类型

job.setMapOutputKeyClass(ReLongWritable.class);

job.setMapOutputValueClass(Text.class);

//设置Reducer输出类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(ReLongWritable.class);

//设置输入路径

FileInputFormat.setInputPaths(job,inputPath);

//设置输出路径

FileOutputFormat.setOutputPath(job,outputPath);

//提交Job

if (fs.exists(outputPath)) {

fs.delete(outputPath,true);

}

boolean result = job.waitForCompletion(true);

if(result)

System.out.println("程序成功执行");

}

}

**JobReduce:**

public class WordCountReducer1 extends Reducer<Text,IntWritable,IntWritable,Text> {

@Override

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

int count = 0;

//迭代计算，将元组中的值累加，得到该岗位总共出现的数量

for(IntWritable value:values){

count += value.get();

}

//结果输出，会生成一个数据文件

key.set("&&"+key.toString());

context.write(new IntWritable(count),key);

}

}

**JobPxMapper:**

public class WordCountMapper1 extends Mapper<LongWritable,Text,Text,IntWritable> {

//LongWritable 对应是长整形（行号）

//Text 对应字符串

//IntWritable 对应正常的整形（每个单词个数）

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

//value 代表读取到的当前行的数据

String data = value.toString();

//通过逗号分隔

String[] temp = data.split(",");

//将每个岗位标记为已经出现一次

if(temp.length>1)

context.write(new Text(temp[0]),new IntWritable(1));

}

}

# 折线图模块设计说明

### 模块描述

* 模块是基于MapReduce分布式计算，分析工资最高的职位Top3 职位名称及工资【折线图展示】

#### 输入项

将51Job\_python\_5000.xls文件上传到Hadoop集群

#### 输出项

将工资最高的职位Top3 职位名称及工资，以折线图的形式展现到画面上

#### 界面设计



#### 代码

package mapper;

public class WordCountMapper2 extends Mapper<LongWritable,Text,ReDoubleWritable,Text>{

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

//value 代表读取到的当前行的数据

String data = value.toString();

//通过逗号分隔

String[] temp = data.split(",");

//将每个岗位标记为已经出现一次

if(temp.length>1)

context.write(new ReDoubleWritable(Salary.converson(temp[3])),new Text(temp[0]));

}

}

package mapper;

public class DataMapper extends Mapper<LongWritable,Text,ReLongWritable,Text>{

@Override

protected void map(LongWritable key, Text value,Context context)

throws IOException, InterruptedException {

// TODO Auto-generated method stub

String data = value.toString();

//通过逗号分隔

String[] temp = data.split(" &&");

//将每个公司标记为已经出现一次

if(temp.length>1)

context.write(new ReLongWritable(Long.parseLong(temp[0])),new Text(temp[1]));

}

}

package master;

public class WordCountMaster2 {

public void test() throws IOException, ClassNotFoundException, InterruptedException {

//输入路径文件

Path inputPath = new Path("/input/51Job\_python\_5000.txt");

//输出路径文件

Path outputPath = new Path("/output/sal");

//初始化配置

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(conf);

//初始化Job参数，指定Job名称

Job job = Job.getInstance(conf,"wordcount");

//设置执行的Job类

job.setJarByClass(WordCountMaster2.class);

//设置Mapper类

job.setMapperClass(WordCountMapper2.class);

//设置Reducer类

job.setReducerClass(WordCountReducer2.class);

//设置Map的输出类型ReDoubleWritable

job.setMapOutputKeyClass(ReDoubleWritable.class);

job.setMapOutputValueClass(Text.class);

//设置Reducer输出类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(ReDoubleWritable.class);

//设置输入路径

FileInputFormat.setInputPaths(job,inputPath);

//设置输出路径

FileOutputFormat.setOutputPath(job,outputPath);

//提交Job

if (fs.exists(outputPath)) {

fs.delete(outputPath,true);

}

job.waitForCompletion(true);

}

}

package reducer;

public class WordCountReducer2 extends Reducer<ReDoubleWritable,Text,Text,ReDoubleWritable> {

private static int a = 0;

@Override

protected void reduce(ReDoubleWritable key, Iterable<Text> values,

Context context)

throws IOException, InterruptedException {

for(Text val:values){

if(a < 3){

a += 1;

context.write(val, key);

}

}

}

}

package reducer;

public class SalaryReducer extends Reducer<Text,DoubleWritable,Text,DoubleWritable> {

@Override

protected void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException {

double count=0;

double sum=0;

//迭代计算，将元组中的值累加，得到该公司总共出现的次数

for(DoubleWritable value:values){

sum+=value.get();//累加计算

count++;

}

//结果输出，会生成一个数据文件

context.write(key,new DoubleWritable(sum/count));

}

}