

**并 行 计 算**

课程实验报告

实验名称：Java并行程序设计

任课教师：徐悦甡

课程班级：15级 云计算方向

学号姓名：15130130273 石明皓

提交日期：2018年5月23日

**软件学院本科生《并行计算》**

**课程实验报告**

1. **实验名称**

第2次实验：Java并行程序设计

1. **实验日期**

2018年5月23日 软件学院实验室G346

1. **实验学生**

15130130273 石明皓

1. **实验目的**

本次实验通过给出2个适用于Fork/Join框架并行执行的情景，要求使用Fork/Join框架正确编写实现其逻辑的Java多线程并行程序。学习Fork/Join框架的基本使用方法，理解处理并行任务的分解、合并、创建线程池等步骤，掌握Java基础并行程序设计的方法。

1. **实验内容**

题目一：

分别从两个列表中提取出数字，各自相加后，比较大小。两个列表的内容如下：

list\_1={“\*”，“%”，“3”，“#”，“6”，“~”，“!”，“2”}

list\_2={“&”，“¥”，“@”，“1”，“4”，“:”，“2”，“1”}

要求正确地使用Fork/Join框架，计算出list\_1中数字的和（3+6+2）与list\_2中数字的和（1+4+2+1），然后比较两个和的大小。

题目二：

统计两个文件中单词‘book’出现的总次数。

有两个文件，file\_1.dat与file\_2.dat，文件内容如下：

file\_1.dat内容：and, with, we, me, university, with, book, computer, country, book

file\_2.dat内容：bag, boy, book, school, teacher, student, book, book

要求正确地使用Fork/Join框架，首先统计出file\_1.dat中出现2次，file\_2.dat中出现3次，然后统计出“book”出现的总次数（5次）。

1. **程序思路、结构**

题目一：

题目要求计算并返回数字之和，故应当继承有返回值的RecursiveTask类；

创建一个静态常量来指定并行任务分解的阈值，该阈值设为1；

给extract\_sum\_compare类的构造函数extract\_sum\_compare()传入的参数为：list列表、列表起始位置和列表结束位置；

重写compute方法，按照if(小于阈值){计算子任务;}else{分解任务为子任务;递归执行子任务;最后合并并行计算结果;}的编程模式完成并行任务；

通过Integer.parseInt()来将表示数字的字符串转换为整型数字，而非数字的字符串将抛出异常，跳出循环，结束本线程的计算；

分解子任务时采用二分法，将列表分成左右两段，直到每段长度为1，达到阈值；

对左右子任务对象使用fork()方法开始并行计算，并分别将使用join()方法返回的计算结果相加合并，得到最终计算结果，即列表中的数字之和。

要执行Fork/Join框架的并行程序，需创建一个线程池，通过invoke()方法把并行任务（即RecursiveTask类的对象）导入线程池，才能得到结果。

题目二：

先分别读入file\_1.dat和file\_2.dat，将其中的内容转换为字符串数组list1和list2，这两个字符串数组的元素是字符串文件内容去除空格、以逗号分隔的结果；

题目要求计算并返回book的出现次数，故应当继承有返回值的RecursiveTask类；

创建一个静态常量来指定并行任务分解的阈值，该阈值设为1；

给java\_book\_sum类的构造函数java\_book\_sum()传入的参数为：list列表、列表起始位置和列表结束位置；

重写compute方法，按照if(小于阈值){计算子任务;}else{分解任务为子任务;递归执行子任务;最后合并并行计算结果;}的编程模式完成并行任务；

通过list[start].matches(“book”)函数正则匹配book，元素从头到尾必须完全匹配才为true；

分解子任务时采用二分法，将字符串数组分成左右两段，直到每段长度为1，达到阈值；

对左右子任务对象使用fork()方法开始并行计算，并分别将使用join()方法返回的计算结果相加合并，得到最终计算结果，即book的出现次数；

要执行Fork/Join框架的并行程序，需创建一个线程池，通过invoke()方法把并行任务（即RecursiveTask类的对象）导入线程池，才能得到结果。

1. **程序代码**

题目一：

import java.util.concurrent.ForkJoinPool;  
import java.util.concurrent.RecursiveTask;  
public class extract\_sum\_compare extends RecursiveTask<Integer> {  
 private static final int *threshold* = 1; //阈值  
 private String[] list;  
 private int start;  
 private int end;  
 public extract\_sum\_compare(String[] list, int start, int end) {  
 this.list = list;  
 this.start = start;  
 this.end = end;  
 }  
 @Override  
 protected Integer compute() {  
 int sum = 0;  
 boolean can\_compute = (end - start) < *threshold*;

//end-start等于0时才为true  
 if (can\_compute) {  
 for (int i = start; i <= end; i++) {

//只循环start到start+1  
 try {  
 Integer.*parseInt*(list[i]); //字符串转换为整型数字  
 } catch (Exception e) {

//非数字不能转换，抛出异常，跳出循环，结束本线程的计算  
 break;  
 }  
 sum = Integer.*parseInt*(list[i]);

//数字计入结果中，下一步就直接返回结果了  
 }  
 } else {  
 int middle = (start + end) / 2;

//二分任务，直到每个小任务list长度仅为1  
  
 //创建左右子任务  
 extract\_sum\_compare left = new extract\_sum\_compare(this.list, start, middle);  
 extract\_sum\_compare right = new extract\_sum\_compare(this.list, middle + 1, end);  
  
 //开始并行计算，也可写成invokeAll(left, right);  
 left.fork();  
 right.fork();  
  
 //合并计算结果  
 sum = left.join() + right.join();  
 }  
 return sum;  
 }  
 public static void main(String[] args) { //并行实现  
 String[] list\_1 = {"\*", "%", "3", "#", "6", "~", "!", "2"};  
 String[] list\_2 = {"&", "¥", "@", "1", "4", ":", "2", "1"};  
 ForkJoinPool fork\_join\_pool\_1=new ForkJoinPool();//创建线程池  
 extract\_sum\_compare test1 = new extract\_sum\_compare(list\_1, 0, list\_1.length - 1); //创建任务  
 int result1 = fork\_join\_pool\_1.invoke(test1);

//invoke()返回计算结果  
 System.*out*.println(result1);  
 ForkJoinPool fork\_join\_pool\_2 = new ForkJoinPool();  
 extract\_sum\_compare test2 = new extract\_sum\_compare(list\_2, 0, list\_2.length - 1);  
 int result2 = fork\_join\_pool\_2.invoke(test2);  
 System.*out*.println(result2);  
 if (result1 > result2) {  
 System.*out*.println("list\_1数字之和 > list\_2数字之和");  
 } else if (result1 == result2) {  
 System.*out*.println("list\_1数字之和 == list\_2数字之和");  
 } else {  
 System.*out*.println("list\_1数字之和 < list\_2数字之和");  
 }  
 }  
}

题目二：

import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.InputStreamReader;  
import java.util.concurrent.ForkJoinPool;  
import java.util.concurrent.RecursiveTask;  
public class java\_book\_sum extends RecursiveTask<Integer> {  
 private static final int *threshold* = 1; //阈值  
 private String[] list;  
 private int start;  
 private int end;  
 public java\_book\_sum(String[] list, int start, int end) {  
 this.list = list;  
 this.start = start;  
 this.end = end;  
 }  
 @Override  
 protected Integer compute() {  
 int count = 0;  
 boolean can\_compute = (end - start) < *threshold*;

//end-start等于0时才为true  
 if (can\_compute) {  
 if (list[start].matches("book")) {

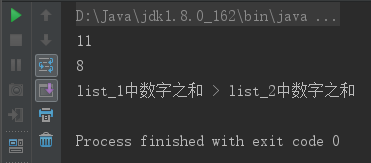
//matches()函数匹配book，从头到尾完全匹配才为true  
 count++;  
 }  
 } else {  
 int middle = (start + end) / 2;

//二分任务，直到每个小任务list长度仅为1  
  
 //创建左右子任务  
 java\_book\_sum left = new java\_book\_sum(this.list, start, middle);  
 java\_book\_sum right = new java\_book\_sum(this.list, middle + 1, end);  
  
 //开始并行计算，也可写成invokeAll(left, right);  
 left.fork();  
 right.fork();  
  
 //合并计算结果  
 count = left.join() + right.join();  
 }  
 return count;  
 }  
 public static void main(String[] args) {  
 String[] list1 = null;  
 String[] list2 = null;  
 try {  
 //读入file\_1.dat，将其中的内容转换为字符串数组list1（去除空格，以逗号分隔）  
 String file\_path1 = "D:\\Kent's Workspace\\java\\Parallel\_Computing\\2nd\\src\\file\_1.dat";  
 File file1 = new File(file\_path1);  
 FileInputStream stream1 = new FileInputStream(file1);  
 InputStreamReader reader1 = new InputStreamReader(stream1);  
 BufferedReader buffer1 = new BufferedReader(reader1);  
 String line1;  
 while ((line1 = buffer1.readLine()) != null) {  
 list1 = line1.trim().replace(" ", "").split(",");  
 }  
 //读入file\_2.dat，将其中的内容转换为字符串数组list2（去除空格，以逗号分隔）  
 String file\_path2 = "D:\\Kent's Workspace\\java\\Parallel\_Computing\\2nd\\src\\file\_2.dat";  
 File file2 = new File(file\_path2);  
 FileInputStream stream2 = new FileInputStream(file2);  
 InputStreamReader reader2 = new InputStreamReader(stream2);  
 BufferedReader buffer2 = new BufferedReader(reader2);  
 String line2;  
 while ((line2 = buffer2.readLine()) != null) {  
 list2 = line2.trim().replace(" ", "").split(",");  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 ForkJoinPool fork\_join\_pool\_1=new ForkJoinPool();//创建线程池  
 java\_book\_sum test1 = new java\_book\_sum(list1, 0, list1.length - 1); //创建任务  
 int result1 = fork\_join\_pool\_1.invoke(test1);

//invoke()返回计算结果  
 System.*out*.println("file\_1.dat中的出现次数：" + result1);  
 ForkJoinPool fork\_join\_pool\_2 = new ForkJoinPool();  
 java\_book\_sum test2 = new java\_book\_sum(list2, 0, list2.length - 1);  
 int result2 = fork\_join\_pool\_2.invoke(test2);  
 System.*out*.println("file\_2.dat中的出现次数：" + result2);  
 int result = result1 + result2;  
 System.*out*.println("book出现的总次数：" + result);  
 }  
}

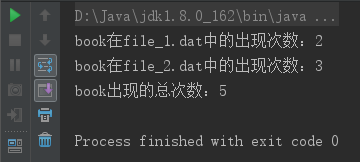
1. **实验结果**

题目一：



从程序运行结果可知，list\_1中的数字之和为11，list\_2中的数字之和为8，list\_1中的数字之和大于list\_2中的数字之和，结果无误，说明Fork/Join框架的并行程序编写成功。

题目二：



从程序运行结果可知，book在file\_1.dat中出现次数为2，在file\_2.dat中出现次数为3，出现的总次数为5，结果无误，说明Fork/Join框架的并行程序编写成功。

1. **总结建议**

经过本次实验，我对并行程序设计的原理有了更深的认识，学会运用Fork/Join框架解决简单的并行计算问题；对于Fork/Join框架，最重要的是找到合理且有效的任务分解方式，给出符合系统算力的阈值；同时我通过进一步学习，对线程池的概念有了一定了解；另外实验的代码编写环节加深了我在Java多线程并行方面编程的熟练度，进而提高了个人的Java编程能力。