# 航班数据的动态分析

姓 名 梁寒冰 学号170341317

小组编号 28 小组其他成员的学号170341301 170341306 170341308 170341331

实验日期 2019年10月20日 实验报告日期2019年10月20日

成 绩 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. 实验目的

1、 理解并掌握Java 中接口的基本概念和具体应用

2、 熟悉并熟练掌握Java 中线程的定义

3、 熟悉并掌握多线程间的消息传递、同步和调度等基本操作

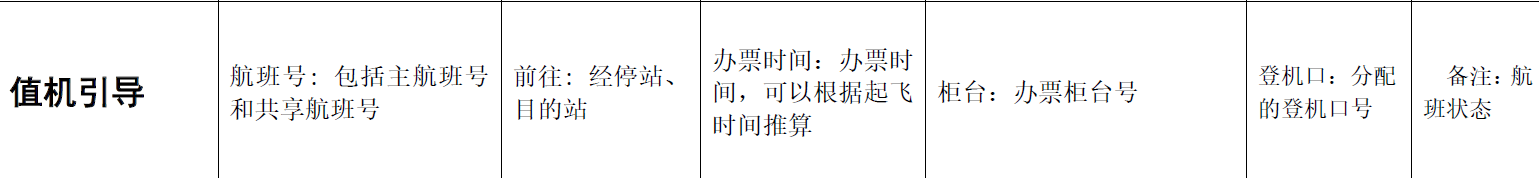
1. 实验环境

JDK，eclipse

文件里./src/ex2为本次实验内容。

1. 实验实际完成内容及结果分析(请先说明你在小组中所承担的任务)

负责值机引导：



1、 在实验一分析基础上，本次实验要求航班数据处理时，对于同一航班的不同类型或同一类型不同时间产生的消息数据要进行关联分析，即如果新读入的航班信息数据与原来处理的数据有相同项或新增项，则判断是否需要对航班的相关数据进行修改，并更具修改情况更

新航班信息

2、 同一个时刻同一个的航班在航显终端上仅显示一条数据

3、 航显终端显示的航班数据应该定时刷新，显示最新的航班数据信息

4、将航班信息的读取、处理和动态显示等功能进行分离，每一项独立功能可以分别设计不同的线程类进行处理

5.程序说明：程序首先封装所需数据成员为类，定义类数组airplane[]，将航班编号flid做唯一标识即数组下标，读入数据线程不变，将数据读入**LinkedBlockingQueue**<String> **analyzingDataVector**阻塞队列中，在分析数据线程中将哈希表的键值数据更改为类对象，使得每一个键值唯一标识一个对象，在分析数据时，将airplane[i]的数据成员进行赋值，并添加到哈希表中，最终在输出线程中根据哈希表键值取得相应的对象，进一步取得数据成员。

源码：

package ex2;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Hashtable;

import java.util.Set;

import java.util.concurrent.LinkedBlockingQueue;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.swing.text.StyledEditorKit.ForegroundAction;

import sun.print.BackgroundLookupListener;

public class EX2 {

public static void main(String[] args) {

// TODO 自动生成的方法存根

System.out.println("航班号 共享航班 经停站 目的站 办票时间 值机柜台 登机口 航班状态");

LinkedBlockingQueue<String> analyzingDataVector=new LinkedBlockingQueue<String>();

Hashtable<String,airPlane> displayDataVecotr=new Hashtable<String,airPlane>();//使哈希表的键值对应类对象

File file=new File("FDSdata.txt");

new ReadData(file,analyzingDataVector).start();

new AnalysisDataThread(analyzingDataVector,displayDataVecotr).start();

new OutputDataThread(displayDataVecotr).start();

}

}

class ReadData extends Thread{

File input;

LinkedBlockingQueue<String> inputData;

public ReadData(File input, LinkedBlockingQueue<String> analyzingDataVector) {

this.input=input;

this.inputData=analyzingDataVector;

}

public void run() {

try {

BufferedReader in=new BufferedReader(new FileReader(input));

String line=null;

while((line=in.readLine())!=null) {

inputData.put(line);

}

inputData.put("END");

if(in!=null) in.close();

}catch(FileNotFoundException e) {

e.printStackTrace();

}catch(IOException e) {

e.printStackTrace();

}catch(InterruptedException e) {

e.printStackTrace();

}

}

}

class AnalysisDataThread extends Thread{

LinkedBlockingQueue<String> analyzingData;

Hashtable<String,airPlane> displayDataVecotr;

public AnalysisDataThread(LinkedBlockingQueue<String> analyzingDataVector,Hashtable<String,airPlane> displayDataVecotr2) {

super();

this.analyzingData=analyzingDataVector;

this.displayDataVecotr=displayDataVecotr2;

}

airPlane[] Fight = new airPlane[10000000];//建立airplane类数组

@Override

public void run() {

for(int i = 1000000; i < 10000000; i++) {

Fight[i] = new airPlane();//给类数组对象开辟空间

}

while(true) {

String line = null;

try {

line =analyzingData.take();

}catch(InterruptedException e) {

e.printStackTrace();

}

if(("END").equals(line))

break;

if(line!=null) {

String string = null;//唯一标识

//航班编号

Pattern p=Pattern.compile("(?<=flid=).\*?(?=\\,)");

Matcher m=p.matcher(line);

if(m.find()) {

string = m.group();

Fight[Integer.parseInt(string)].planeflid = m.group();//将数据赋给对象

}

//航班标识

Pattern p1 = Pattern.compile("(?<=ffid=).\*?(?=\\,)");

Matcher m1 = p1.matcher(line);

if(m1.find())

{

String[] tt = m1.group().split("-");

Fight[Integer.parseInt(string)].planeffid = tt[0]+tt[1];//将数据赋给对象

}

//共享航班号

Pattern p2 = Pattern.compile("(?<=sfno=).\*?(?=\\,)");

Matcher m2 = p2.matcher(line);

if(m2.find())

{

Fight[Integer.parseInt(string)].planesfno = m2.group();//将数据赋给对象

}

//经停

Pattern p4 = Pattern.compile("(?<=arno=2, apcd=).\*?(?=\\,)");

Matcher m4 = p4.matcher(line);

if(m4.find())

{

Fight[Integer.parseInt(string)].planearno2 = m4.group();//将数据赋给对象

}

//目的

Pattern p5 = Pattern.compile("(?<=arno=3, apcd=).\*?(?=\\,)");

Matcher m5 = p5.matcher(line);

if(m5.find())

{

Fight[Integer.parseInt(string)].planearno3 = m5.group();//将数据赋给对象

}

//办票时间

Pattern p6 = Pattern.compile("(?<=fett=).\*?(?=\\,)");

Matcher m6 = p6.matcher(line);

if(m6.find())

{

Fight[Integer.parseInt(string)].planefett = m6.group();//将数据赋给对象

}

//柜台

Pattern p7\_ckls = Pattern.compile("(?<=ckls=).\*?(?=\\])");

Matcher m7\_ckls = p7\_ckls.matcher(line);

if(m7\_ckls.find())

{

Pattern p7\_code = Pattern.compile("(?<=code=).\*?(?=\\,)");

Matcher m7\_code = p7\_code.matcher(m7\_ckls.group());

if(m7\_code.find())

{

Fight[Integer.parseInt(string)].planeckls = m7\_code.group();//将数据赋给对象

}

}

//登机口

Pattern p8\_gtls = Pattern.compile("(?<=DFME\_GTLS\\[).\*?(?=\\])");

Matcher m8\_gtls = p8\_gtls.matcher(line);

if(m8\_gtls.find())

{

Pattern p8\_code = Pattern.compile("(?<=code=).\*?(?=\\,)");

Matcher m8\_code = p8\_code.matcher(m8\_gtls.group());

if(m8\_code.find())

{

Fight[Integer.parseInt(string)].planegtls = m8\_code.group();//将数据赋给对象

}

}

//航班状态

Pattern p9 = Pattern.compile("(?<=ista=).\*?(?=\\,)");

Matcher m9 = p9.matcher(line);

if(m9.find())

{

Fight[Integer.parseInt(string)].planeitsa = m9.group();//将数据赋给对象

}

synchronized(displayDataVecotr) {

displayDataVecotr.put(string,Fight[Integer.parseInt(string)]);//将类对象加入哈希表

displayDataVecotr.notify();

}

}

}

}

}

class OutputDataThread extends Thread{

Hashtable<String,airPlane> displayDataHashtable;

public OutputDataThread(Hashtable<String,airPlane> displayDataHashtable) {

this.displayDataHashtable=displayDataHashtable;

}

public void run() {

while(true) {

try {

synchronized(displayDataHashtable) {

Set<String> flidSet=displayDataHashtable.keySet();

int i=0;

for(String string:flidSet) {

if(i%10==0)

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

else {

/\*PrintWriter pout =new PrintWriter(new FileWriter("result2.txt"),true);

pout.print(displayDataHashtable.get(string).planeffid+" "+

displayDataHashtable.get(string).planesfno+" "+

displayDataHashtable.get(string).planearno2+" "+

displayDataHashtable.get(string).planearno3+" "+

displayDataHashtable.get(string).planeckls+" "+

displayDataHashtable.get(string).planegtls+" "+

displayDataHashtable.get(string).planeitsa);\*/

System.out.println(displayDataHashtable.get(string).planeffid+" "+

displayDataHashtable.get(string).planesfno+" "+

displayDataHashtable.get(string).planearno2+" "+

displayDataHashtable.get(string).planearno3+" "+

displayDataHashtable.get(string).planeckls+" "+

displayDataHashtable.get(string).planegtls+" "+

displayDataHashtable.get(string).planeitsa);

}

i++;

}

}

Thread.sleep(50);

}catch(InterruptedException e) {

e.printStackTrace();

}

/\*catch (IOException e) {

// TODO 自动生成的 catch 块

e.printStackTrace();

}\*/

}

}

}

//将数据成员封装为类

class airPlane {

String planeflid = null;//航班编号

String planeffid = null;//航班标识

String planesfno = null;//共享航班号

String planearno2 = null;//经停

String planearno3 = null;//目的

String planefett = null;//半票时间

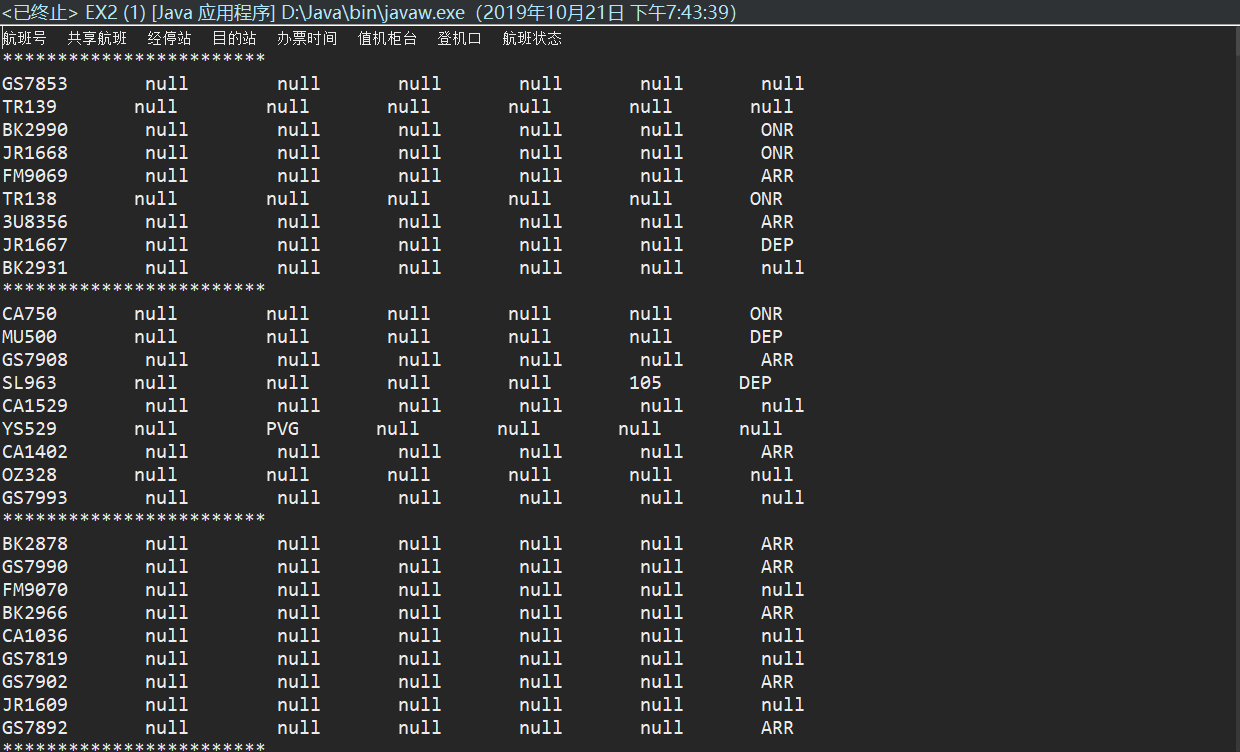
String planeckls = null;//柜台

String planegtls = null;//登机口

String planeitsa = null;//航班状态

boolean temp = false;

}



1. 思考题
2. 多线程的设计有什么特点？

（1）进程之间不能共享内存，但线程之间可以，且十分容易

（2）系统创建进程是需要为该进程重新分配系统资源，但创建线程则代价小得多，因此使用多线程来实现多任务并发比多进程的效率高。

2、本次实验用到了多线程处理的哪些机制？

多线程同步机制，休眠以及等待机制