# 读取文件报告

## 实验目的：

1. 按照飞机票的价格升序排序，查找起点到终点、再返回到起点的前5个路程规划，
2. 得出时间开销

## 运行环境：

1. 平台：Windows10+JAVAEE+JDK1.8
2. 运行内存：4G
3. 文件大小：2.53G+893kb

## 思考：

始发地、目的地、航班号、出发日期....，如果全写成数组或者集合的形式，就没有数据属性的关联性，故而吸取面向对象的思想，把需要的属性信息抽取出来封装成一个对象，进而数据易处理操作。

利用 java中的static 关键字将封装好的对象存放在RAM.

利用双向搜索思想，构建两个数据结构，集合一中的起点即是集合二中的终点，集合二中的起点即是集合一中的终点。

按照价格排序，Java中有对对象某一属性排序的接口，**implements** **Comparable<oriEng>，实现接口，必实现接口中的方法：**

//实现Comparable接口，实现接口类中的方法，对价格进行排序 按照 升序排序

@Override

**public** **int** compareTo(oriEng o) {

// **TODO** Auto-generated method stub

**return** **new** Float(**this**.getPrice()).compareTo(o.getPrice());

}

对象的数据结构

数据结构：

集合一：desEng类对象（起点，价格，终点，终点主题，航班号，起飞日期 ）

class desEng **implements** Comparable<desEng> {

//起点

**private** String oriEng;

//价格

**private** **float** price;

//终点

**private** String desEng;

//起点特征

**private** String oriEngthemeName;

//航班号

**private** String flightNo;

//起飞时间

**private** String flightDt;

Getter And Setter方法

//实现Comparable接口，实现接口类中的方法，对价格进行排序 按照 升序排序

@Override

**public** **int** compareTo(desEng o) {

**return** **new** Float(**this**.getPrice()).compareTo(o.getPrice());

}

}

List<desEng> *desEngList* = **new** ArrayList<desEng>();

集合二oriEng类对象（起点，价格，终点，终点主题，航班号，起飞日期 ）

class oriEng **implements** **Comparable<oriEng>** {

//起点

**private** String oriEng;

//价格

**private** **float** price;

//终点

**private** String desEng;

//起点特征

**private** String oriEngthemeName;

//航班号

**private** String flightNo;

//起飞时间

**private** String flightDt;

Getter And Setter方法

//实现Comparable接口，实现接口类中的方法，对价格进行排序 按照 升序排序

@Override

**public** **int** compareTo(oriEng o) {

**return** **new** Float(**this**.getPrice()).compareTo(o.getPrice());

}

}

List<oriEng> *desEngList* = **new** ArrayList<oriEng>();

## 实现

1. 通过城市获取城市的主题

城市主题特征在 cityProperty.json文件中

具体处理函数如：

**public** **static** String getCityThemes(String cityEnName) {

String cityThemes = "";

BufferedInputStream bis;

**try** {

bis = **new** BufferedInputStream(**new** FileInputStream(**new** File("F:/cityProperty.json")));

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(bis, "utf-8"), 10 \* 1024 \* 1024);

String s = **null**;

// 此此循环得到多少个机场

**while** ((s = in.readLine()) != **null**) {

// 创建一个包含原始json串的json对象

JSONObject dataJson = **new** JSONObject(s);

**if** (s.contains("airPortList")) {

String airport\_cd = "";

JSONArray airPortList = dataJson.getJSONArray("airPortList");

**int** size = airPortList.length();

**for** (**int** i = 0; i < size; i++) {

JSONObject airPort = airPortList.getJSONObject(i);

airport\_cd = airPort.getString("airport\_cd");

**if** (airport\_cd.equalsIgnoreCase(cityEnName)) {

JSONArray themesarray = dataJson.getJSONArray("themes");

**for** (**int** j = 0; j < themesarray.length(); j++) {

JSONObject Themes = themesarray.getJSONObject(j);

cityThemes = Themes.getString("themeName");

}

} **else**

**break**;

}

} **else**

**continue**;

}

} **catch** (IOException | JSONException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** cityThemes;

}

1. 得到集合一 desEngList：

将所有已封装好的对象存放在内存上。

**private** **static** List<desEng> *desEngList* = **new** ArrayList<desEng>();

具体函数处理如下：

**public** **static** List<desEng> getDesEngClass() **throws** FileNotFoundException {

List<desEng> desEngList = **new** ArrayList<desEng>();

BufferedInputStream bis;

**try** {

bis = **new** BufferedInputStream(**new** FileInputStream(**new** File("F:/farecachehc.json")));

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(bis, "utf-8"), 10 \* 1024 \* 1024);

String s = **null**;

// 此此循环得到多少个机场

**while** ((s = in.readLine()) != **null**) {

// 创建一个包含原始json串的json对象

JSONObject dataJson = **new** JSONObject(s);

desEng deseng = **new** desEng();

**if** (s.contains("fareInfo")) {

JSONArray fareinfoarray = dataJson.getJSONArray("fareInfo");

**int** size = fareinfoarray.length();

**for** (**int** i = 0; i < size; i++) {

JSONObject fareInfoList = fareinfoarray.getJSONObject(i);

deseng.setPrice(Integer.*parseInt*(fareInfoList.getString("price")));

}

} **else** {

**continue**;

}

**if** (s.contains("flightInfo")) {

JSONObject flightInfoList = dataJson.getJSONObject("flightInfo");

String tempori = flightInfoList.getString("oriEng");

deseng.setDesEng(flightInfoList.getString("desEng"));

deseng.setFlightNo(flightInfoList.getString("flightNo"));

deseng.setFlightDt(flightInfoList.getString("flightDt"));

deseng.setDesEng(tempori);

deseng.setDesEngthemeName(*getCityThemes*(tempori));

} **else** {

**continue**;

}

desEngList.add(deseng);

}

} **catch** (IOException | JSONException e) {

e.printStackTrace();

}

**return** desEngList;

}

1. 得到集合二 oriEngList：

将所有已封装好的对象存放在内存上。

**private** **static** List<oriEng> ori*EngList* = **new** ArrayList<oriEng>();

具体函数处理如下：

**public** **static** List<oriEng> getOriEngClass() **throws** FileNotFoundException {

List<oriEng> oriEngList = **new** ArrayList<oriEng>();

BufferedInputStream bis;

**try** {

bis = **new** BufferedInputStream(**new** FileInputStream(**new** File("F:/farecachehc.json")));

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(bis, "utf-8"), 10 \* 1024 \* 1024);

String s = **null**;

// 此此循环得到多少个机场

**while** ((s = in.readLine()) != **null**) {

// 创建一个包含原始json串的json对象

JSONObject dataJson = **new** JSONObject(s);

oriEng deseng = **new** oriEng();

**if** (s.contains("fareInfo")) {

JSONArray fareinfoarray = dataJson.getJSONArray("fareInfo");

**int** size = fareinfoarray.length();

**for** (**int** i = 0; i < size; i++) {

JSONObject fareInfoList = fareinfoarray.getJSONObject(i);

deseng.setPrice(Integer.*parseInt*(fareInfoList.getString("price")));

}

} **else** {

**continue**;

}

**if** (s.contains("flightInfo")) {

JSONObject flightInfoList = dataJson.getJSONObject("flightInfo");

String tempori = flightInfoList.getString("oriEng");

deseng.setDesEng(flightInfoList.getString("desEng"));

deseng.setFlightNo(flightInfoList.getString("flightNo"));

deseng.setFlightDt(flightInfoList.getString("flightDt"));

deseng.setOriEng(tempori);

deseng.setOriEngthemeName(*getCityThemes*(tempori));

deseng.setPrice(Integer.*parseInt*(flightInfoList.getString("price")));

} **else** {

**continue**;

}

oriEngList.add(deseng);

}

} **catch** (IOException | JSONException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** oriEngList;

}

1. 整合处理，得到结果：

**public** **static** **void** main(String[] args) {

**long** startTime = System.*currentTimeMillis*();

**float** excTime = 0.00f;

**try** {

*desEngList* = *getDesEngClass*();

*oriEngList* = *getOriEngClass*();

**int** i = 0;

**for** (desEng deseng : *desEngList*) {

**for** (oriEng orieng : *oriEngList*) {

**if** (orieng.getOriEng().equals(deseng.getDesEng())

&& orieng.getDesEng().equals(deseng.getOriEng())) {

**if** (i < 5) {

System.***out***.println("" + orieng.getOriEng() + "-->" + deseng.getDesEng() + "TOP(5)price : "+ orieng.getPrice());

i++;

} **else**

**break**;

} **else**

**break**;

}

}

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**long** endTime = System.*currentTimeMillis*();

excTime = (**float**) (endTime - startTime) / 1000;

System.***out***.println("执行时间：" + excTime + "s");

}

## 测试

两个多小时的时间还没有出现结果。就暂停运行了

## 6.总结

读取文件太频繁，并且读取文件太大，造成运行时间过长，如果应用将数据全部写到数组中，无法保证数据的关联性，若采取数组的方法 ，将会造成遍历数组（查找数据）时产生“爆炸 ”时间，时间会随着数组空间复杂度 指数级增长。