---------------------------------------------------

计划课程安排：

FUNDAMENTAL(7天)         该阶段已结束

OOP(11天)               该阶段已结束

JAVASE01(10天)    该阶段已结束

JAVASE02(8天) ←当前为该阶段第6天

WEBBASIC(7天)

JQUERY(2天)

DATABASE(6天)

JDBC(3天)

SERVLETJSP(12天)

SPRINGMYBATIS01(8天)

AJAX(2天)

SPRINGMYBATIS02(12天)

LINUX(2天)

NGINX(2天)

REDIS(2天)

T-STORE(5天)

预计合计：100天

预计剩余：63天

注：上述信息随时可能发生更改，仅供参考。

---------------------------------------------------

**2019年4月16日**

**Java就业班-第40天**

**8:32实训室2-晨讲**

今日晨讲内容——复习多线程。

我再次迟到，当前需上交数累计50。

**9:00实训室2-正课**

开始上课，继续昨天的课程项目内容。

WebServer\_v7

README.txt--------

本版本主要工作：重构代码

与请求相同，我们可以再设计一个类：HttpResponse,

并使用这类的每个实例表示服务端给客户端发送一个响应。

将响应的内容发送给客户端的工作也定义在HttpResponse中，

使得ClientHandler中发送响应代码得意重用。

实现：

1.在com.webserver.http包中定义类：HttpResponse

2.在HttpResponse中定义一个响应所对应的三部分的属性

3.为属性提供get、set方法

4.定义一个方法flush

这个方法用来将当前响应对象的内容以一个标准的HTTP

响应格式发送给客户端（原ClientHandler的工作）

5.ClientHandler在发送响应时改为调用HttpResponse的flush方法

------------------------

**11:50实训室2-正课**

代码重构完成：

**服务端**

package com.webserver.core;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

 \* 服务端

 \* **@author** Administrator

 \*

 \*/

//通过http://localhost:8088/可以实现本地访问

public class WebServer {

    private  ServerSocket server;

    /\*\*

     \* 构造方法，用于初始化

     \*/

    public WebServer() {

       try {

           System.***out***.println("正在启动服务器……");

           server = new ServerSocket(8088);

           System.***out***.println("服务器启动完毕！");

       } catch (Exception e) {

           e.printStackTrace();

       }

    }

    /\*\*

     \* 服务端开始工作的方法

     \*/

    public void start() {

       try {

           while(true) {

           System.***out***.println("等待客户端连接……");

           Socket socket = server.accept();

           System.***out***.println("一个客户端连接了！");

           //启动一个线程处理该客户端请求

           ClientHandler handler = new ClientHandler(socket);

           Thread t = new Thread(handler);

           t.start();

           }

       } catch (Exception e) {

           e.printStackTrace();

       }

    }

    public static void main(String[] args) {

       WebServer server = new WebServer();

       server.start();

    }

}

**请求处理端：**

package com.webserver.core;

import java.io.File;

import java.net.Socket;

import com.webserver.http.HttpRequest;

import com.webserver.http.HttpResponse;

/\*\*

 \* 用于处理客户端请求

 \*

 \* **@author** Administrator

 \*

 \*/

public class ClientHandler implements Runnable {

    private Socket socket;

    public ClientHandler(Socket socket) {

       this.socket = socket;

    }

    public void run() {

       try {

           System.***out***.println("ClientHandler:开始处理请求");

           //1.准备工作

           //1.1实例化请求对象，解析请求

           HttpRequest request = new HttpRequest(socket);

           //1.2实例化响应对象

           HttpResponse response = new HttpResponse(socket);

           // 2.处理请求

           /\*

            \* 2.1通过request获取url,用来得知用户请求的资源的路径 2.2从webapps目录下根据该资源路径找到对应资源 2.3判断该资源时是否真实存在

            \* 2.4存在则响应该资源 2.5不存在则响应404页面

            \*/

           // myweb/index.html

           String path = request.getUrl();

           // 通过路径找到webapps目录下对应资源。

           File file = new File("webapps" + path);

           // 判断用户请求的资源是否真实存在

           if (file.exists()) {

              System.***out***.println("ClientHandler:资源已找到！");

              // 将该资源以标准的HTTP响应格式发送给客户端

              // 将要相应的资源设置到response的entity属性上

              response.setEntity(file);

              response.flush();

           } else {

              System.***out***.println("ClientHandler:资源不存在！");

              File file404 = new File("webapps/root/404.html");

              // 设置状态代码和描述

              response.setStatusCode(404);

              response.setStatusReason("NOT FOUND");

              // 设置响应正文为404页面

              response.setEntity(file404);

           }

           // 3.发送响应

           response.flush();

           System.***out***.println("ClientHandler:处理完毕！");

       } catch (Exception e) {

           e.printStackTrace();

       } finally {

           // 处理完毕后与客户端断开连接

           try {

              socket.close();

           } catch (Exception e2) {

              e2.printStackTrace();

           }

       }

    }

}

**请求响应端：**

package com.webserver.http;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.io.UnsupportedEncodingException;

import java.net.Socket;

/\*\*

 \* 响应对象 该类的每个实例用于表示发送给客户端的响应内容

 \*

 \* 一个响应包含三部分：状态行，响应头，响应正文

 \* **@author** Administrator

 \*

 \*/

public class HttpResponse {

    /\*

     \* 状态行相关信息

     \*/

    /\*

     \* 状态代码，默认值200 默认值为200主要原因有两个： 1.如果不指定默认值，int默认值为0，若我们没有设置状态代码的话，

     \* HTTP协议是没有状态代码为0的这个情况的，所以不能用0作为默认值。 2.通常一个请求都能正确处理，回复客户端200是比较多的情况，

     \* 因此默认值用200可以在大部分响应时不用指定状态代码和对应描述了。

     \*/

    private int statusCode = 200;

    // 状态描述，默认值OK

    private String statusReason = "OK";

    /\*

     \* 响应头相关信息

     \*/

    /\*

     \* 响应正文相关信息

     \*/

    // 响应正文的实体文件

    private File entity;

    /\*

     \* 与连接相关的信息

     \*/

    private Socket socket;

    private OutputStream out;// 输出流

    public int getStatusCode() {

       return statusCode;

    }

    public void setStatusCode(int statusCode) {

       this.statusCode = statusCode;

    }

    public String getStatusReason() {

       return statusReason;

    }

    public void setStatusReason(String statusReason) {

       this.statusReason = statusReason;

    }

    public File getEntity() {

       return entity;

    }

    public void setEntity(File entity) {

       this.entity = entity;

    }

    /\*\*

     \* 创建HttpResponse的同时需要指定Socket 当前响应对象就是通过这个Socket获取输出流 给对应客户端发送响应内容的

     \*

     \* **@param** socket

     \*/

    // 构造方法

    public HttpResponse(Socket socket) {

       try {

           this.socket = socket;

           this.out = socket.getOutputStream();

       } catch (Exception e) {

           e.printStackTrace();

       }

    }

    /\*\*

     \* 将当前响应对象内容以一个标准HTTP响应格式发送给客户端

     \*/

    public void flush() {

       try {

           // 1发送状态行

           sendStatusLine();

           // 2发送响应头

           sendHeaders();

           // 3发送响应正文

           sendContent();

       } catch (Exception e) {

           e.printStackTrace();

       }

    }

    /\*

     \* 发送状态行

     \*/

    private void sendStatusLine() throws UnsupportedEncodingException, IOException {

       System.***out***.println("HttpResponse:开始发送状态行……");

       // 1发送状态行

       String line = "HTTP/1.1 200 OK";

       out.write(line.getBytes("ISO8859-1"));

       out.write(13);// written CR

       out.write(10);// written LF

       System.***out***.println("HttpResponse:发送状态行完毕！");

    }

    /\*

     \* 发送响应头

     \*/

    private void sendHeaders() throws IOException {

       System.***out***.println("HttpResponse:开始发送响应头……");

       // 2发送响应头

       String line = null;

       line = "Content-Type:text/html";

       out.write(line.getBytes("ISO8859-1"));

       out.write(13);// written CR

       out.write(10);// written LF

       // 告知浏览器响应正文的数据长度（字节）

       line = "Content-Length:" + entity.length();

       out.write(line.getBytes("ISO8859-1"));

       out.write(13);// written CR

       out.write(10);// written LF

       // 单独发送CRLF表示响应头发送

       out.write(13);// written CR

       out.write(10);// written LF

       System.***out***.println("HttpResponse:发送消息头完毕！");

    }

    /\*

     \* 发送响应正文

     \*/

    private void sendContent() throws IOException {

       System.***out***.println("HttpResponse:开始发送响应正文……");

       if(entity!=null) {//有时客户端访问没有正文

           //如果实体文件存在，则作为正文发送。

           try(FileInputStream fis = new FileInputStream(entity)){

              // 3.发送响应正文

              // 将用户请求的文件数据作为正文发送给客户端

              int len = -1;

              byte data[] = new byte[1024 \* 10];

              while ((len = fis.read(data)) != -1) {

                  out.write(data, 0, len);

              }

           }catch(IOException e) {

              throw e;

           }

       }

       System.***out***.println("HttpResponse:发送消息正文完毕！");

    }

}

**index.html页面：**

<**html**>

<**head**>

<!--

元数据（meta），其中charset使用来告知浏览器当前页面中出现的文本数据应当用什么字符集，

凡是将来浏览器看到中文是乱码的，都是这里出现的问题，注意拼写。

-->

<**meta** charset=*"utf-8"*>

<**title**>My page</**title**>

</**head**>

<**body**>

    <**center**>

    <!--

    在页面上我们也经常使用路径指定某个资源的位置，比如下面指定图片的位置。

    而这里指定位置的路径也常使用相对路径，在页面上相对路径中“./”对应的当前目录，

    浏览器会认为是当前页面所在的目录。

    例如：

    我们访问当前页面时在浏览器输入的路径为：

    http://localhost:8088/myweb/index.html

    那么浏览器认为“./”表示的是：

    http://localhost:8088/myweb/

    因此下面图片指定的路径src="./logo.png"的实际路径：

    http://localhost:8088/myweb/logo.png

    -->

    <**img** src=*"./logo.png"*><**br**>

    <**input** type=*"text"* size=*32*"">

    <**input** type=*"button"* value=*"AC一下"* onclick="alert('AC在爱一直在！')">

</**center**>

</**body**>

</**html**>

**14:02实训室2-正课**

导入机构的测试用前端页面——学子商城。

进行Debug。

WebServer\_v7

README.txt--------

本版本主要工作：解决空请求问题

HTTP协议允许客户端发送空请求，并且浏览器发送空请求后就会与服务端断开连接。

但是以我们现在的实现来看，若发送的是空请求，解析请求中的请求行时会出现下标越界的情况

（因为我们会将请求行按照空格拆分为三项，空请求在读取请求行操作得到的是一个空字符串，是无法拆分出三项的）。

并且处理最后还会给客户端发送响应，由于客户端已经断开了连接，因此会出现“断开的管道”这样的异常。

对此，我们应当在解析请求时处理空请求问题，一旦遇到直接停止本次处理请求的后续一切操作。

实现：

1.在com.webserver.core包中创建自定义异常

EmptyRequestException,即：空请求异常

2.在HttpRequest的解析请求行方法：parseRequestLine中读取请求行后判定，

若内容是一个空字符串，则认为是空请求，此时应当对外抛出空请求异常。

3.HttpRequest的构造方法要单独捕获空请求异常，捕获到后

再将其继续抛出给ClientHandler

4.ClientHandler添加一个catch单独捕获空请求，不用做任何处理，

目的在于通过捕获这个异常来跳过处理该请求的后续所有步骤，直接与客户端断开连接忽略本次请求即可。

------------------------

（因改动较少，代码就不贴了。详细参考后续WebServer\_v8的课程项目代码。）

**15:37实训室2-正课**

WebServer\_v8

README.txt--------

本版本主要工作：重构HttpResponse发送响应头的工作

之前版本中HttpResponse发送响应头的方法：sendHeaders

是发送两个固定得头：Content-Type和Content-Length.

甚至Content-Type的值是写死的，而不是根据客户端实际请求的资源类型而改变。

这导致我们显示学子商城这套页面时出现显示不正常的情况。

因此我们对发送响应头的工作作出如下两个改动：

1.发送响应头改为根据实际设置的头进行发送，而不是固定发送两个头。

2.对于Content-Type的值应当根据请求的资源而决定它对应的值，以便浏览器

可以正确理解其请求的资源进行显示。

实现：

1.在HttpResponse中定义一个属性：Map headers

其中key表示响应头名字，value表示响应头的值

2.修改发送响应头的方法：sendHeaders,将原有固定发送

改为遍历headers发送所有响应头。

3.对外提供putHeader方法，以便外界可以对HttpResponse设置要发送的响应头。

4.在com.webserver.http包中定义一个类：HttpContext

这个类用于将所有有关HTTP协议规定的内容定义在这里

本次先定义一个静态属性Map  mimeMapping,用来记录资源后缀

与Content-Type值得对应

然后在静态块中初始化这个Map.

最后定义一个静态方法：getMimeType,

用于让外界可以根据给定的资源后缀获取到对应的Content-Type的值

5.ClientHandler在处理请求中，若找到了资源，在向响应对象设置完该资源到正文中后，

再设置其对应的两个响应头，其中Content-Type对应的值可以根据该资源的后缀从HttpContext中找到

没有找到资源则设置响应头与原来一致。

当以上操作完成后，学子商城页面可以正常显示。

------------------------

**16:10实训室2-正课**

1.       在HttpResponse中新增代码：

/\*

 \* 响应头相关信息

 \*/

private Map<String,String> headers = new HashMap();

2.       在sendHeaders()方法中新增代码，替换原发送响应头代码：

// 2发送响应头

       Set<Entry<String,String>> entrySet = headers.entrySet();//遍历Map键值对

       for (Entry<String, String> header : entrySet) {

           String name =  header.getKey();//取得键

           String value = header.getValue();//取得值

           String line = name + ":" + value ;//组成响应头

           out.write(line.getBytes("ISO8859-1"));//指定编码规则

           out.write(13);//指定位置写入CR

           out.write(10);//指定位置写入LR

}

3.       新增设置headers键值的方法代码：

/\*\*

     \* 向当前相应对象中添加一个响应头

     \***@param** name 响应头名字

     \***@param** value 响应头对应的值

     \*/

    public  void putHeader(String name,String value) {

       this.headers.put(name, value);

}

4.       在ClientHandler的run()方法的if()判断中新增代码：

//获取资源名称

              response.setEntity(file);

              String fileName = file.getName();

              System.**out**.println("资源名：" + fileName + "-----------!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");

              //截取资源名的后缀。

              String ext = fileName.substring(fileName.lastIndexOf(".")+1);

              System.**out**.println("资源名后缀：" + ext + "--------------!!!!!!!!!!!!!!!!!!!!!!!!!!!!");

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

              map.put("html", "text/html");

              map.put("css", "text/css");

              map.put("js", "application/javascript");

              map.put("png", "image/png");

              map.put("gif", "image/gif");

              map.put("jpg", "image/jpeg");

              //（可继续添加，保证扩展性。）

//添加响应头

response.putHeader("Content-Type", map.get(ext));

response.putHeader("Content-Length", file.length()+"");

**17:08实训室2-正课**

但上述Map的使用方法不妥，而又由于其键值几乎不变，所以我们新建一个类专门用来保存HTTP协议规定的相关内容。

package com.webserver.http;

import java.util.HashMap;

import java.util.Map;

/\*\*

 \* HTTP协议规定的相关内容

 \* **@author** Administrator

 \*

 \*/

public class HttpContext {

    /\*

     \* 资源后缀名与Content-Type响应头对应值的映射关系

     \* key:资源后缀名

     \* value:Content-Type对应的值

     \*/

    private static Map<String,String> mimeMapping = new HashMap<>();

    //初始化所有静态资源

    static {

       initMimeMapping();

    }

    private static void initMimeMapping() {

       mimeMapping.put("html", "text/html");

       mimeMapping.put("css", "text/css");

       mimeMapping.put("js", "application/javascript");

       mimeMapping.put("png", "image/png");

       mimeMapping.put("gif", "image/gif");

       mimeMapping.put("jpg", "image/jpeg");

    }

    /\*\*

     \* 根据给定的资源后缀名获取对应的Content-Type值

     \* **@param** ext

     \* **@return**

     \*/

    public static String getMimeType(String ext) {

       return mimeMapping.get(ext);

    }

}

5.       随后将在ClientHandler的run()方法的if()判断中的此段代码修改为：

//添加响应头

response.putHeader("Content-Type", HttpContext.getMimeType(ext));

response.putHeader("Content-Length", file.length()+"");

**17:32实训室2-下课时间**

至此，此次课程项目全部结束，同时今天的课就上到这里了。

（各阶段完整的项目代码已上传UP主的Github，如果需要欢迎自取，地址就写在文章末尾。）

**18:15实训室2-下课时间**

今天的代码也成功上传了，github是真的好用！

晚饭就随便应付一下吧。今天的晚自习也要元气满满地写代码！

**18:53实训室2-晚自习**

一回到教室就看到班主任在传播负能量。

照例插上耳机，音乐走起，开始写代码~！

**21:08实训室2-晚自习**

**第三次尝试挑战第6日遗留练习题！**

package GameCollectionDemo01;

import java.util.HashMap;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Scanner;

import java.util.Set;

//面向对象理解练习

/\*需求：有3种游戏类型，游戏类型分为ACT、RTS、RPG。

 \*现有以下游戏：鬼泣（ACT），命令与征服（RTS），龙与地下城（RPG），勇者斗恶龙（RPG），红色警戒（RTS）

 \*，怪物猎人（RPG·ATC），噬神者（ACT·RPG），神界（RPG·RTS），如龙（RPG·ACT），帝国时代（RTS）。

 \*一个游戏可能同时属于2种游戏类型。

 \*这些游戏又分别由4个不同用户所拥有一部分 ，每个用户分别拥有自己的用户名及密码:

 \*Kamui:鬼泣，怪物猎人。      ID:Kamui  PW:001  Genders：male  Age:25

 \*Subaru:命令与征服，红色警戒，帝国时代，龙与地下城。  ID:Subaru  PW:002 Genders：male  Age:18

 \*Luna:神界，如龙，勇者斗恶龙，龙与地下城。       ID:Luna  PW:003      Genders：female  Age:17

 \*Nerd:鬼泣，命令与征服，龙与地下城，勇者斗恶龙，红色警戒，怪物猎人，噬神者，神界，如龙，帝国时代。

 \*ID:Nerd  PW:999 Genders：unknow  Age:0

 \*现要求在输入查找游戏类型、游戏名称、用户，这3个条件中的任意一项时可以输出另外两项的信息。

 \*当通过用户名进行查询时，必须输入密码。并在成功后输出用户所有信息。

 \*在用户名，游戏类型，游戏名输入有误时提供相关信息反馈用户。

 \*程序运行时需提供用户选择：1.通过游戏名查找，2.通过游戏类型查找，3.登录用户ID-输出用户所拥有的游戏。

\*/

/\*\* 程序分析：

 \* 属性关系：类型-游戏 相关，游戏-用户相关，类型-用户相关。

 \*/

class Database{//数据库

    private Map<String,String> gameNameAndType;

    private Map<String,String> userAndGameName;

    protected Database(){

       //游戏名与类型关联

       gameNameAndType = new HashMap<>();

       gameNameAndType.put("鬼泣", "ACT");

       gameNameAndType.put("命令与征服", "RTS");

       gameNameAndType.put("龙与地下城", "RPG");

       gameNameAndType.put("勇者斗恶龙", "RPG");

       gameNameAndType.put("红色警戒" , "RTS");

       gameNameAndType.put("帝国时代" , "RTS");

       gameNameAndType.put("怪物猎人" , "RTS" + "·" + "RPG");

       gameNameAndType.put("噬神者"  , "ATC" + "·" + "RPG");

       gameNameAndType.put("神界"    , "RTS" + "·" + "RPG");

       gameNameAndType.put("如龙"    , "ACT" + "·" + "RPG");

       //用户与游戏名关联

       userAndGameName = new HashMap<>();

       userAndGameName.put("Kamui", "鬼泣" + "、" + "怪物猎人");

       userAndGameName.put("Subaru", "命令与征服" + "、" + "红色警戒" + "、" + "帝国时代" + "、" + "龙与地下城");

       userAndGameName.put("Luna", "神界" + "、" + "如龙" + "、" + "勇者斗恶龙" + "、" + "龙与地下城");

       userAndGameName.put("Nerd", "鬼泣" + "、" + "命令与征服" + "、" + "龙与地下城" + "、" + "勇者斗恶龙"

                     + "、" + "红色警戒" + "、" + "噬神者" + "、" + "神界" + "、" + "如龙" + "、" + "帝国时代");

    }

    //setter&getter

    public Map<String, String> getGameNameAndType() {

       return gameNameAndType;

    }

    public void setGameNameAndType(Map<String, String> gameNameAndType) {

       this.gameNameAndType = gameNameAndType;

    }

    public Map<String, String> getUserAndGameName() {

       return userAndGameName;

    }

    public void setUserAndGameName(Map<String, String> userAndGameName) {

       this.userAndGameName = userAndGameName;

    }

}

class SearchPrint{//游戏类型或游戏名查找打印方法

    private String keyWord ;

    protected SearchPrint(String keyWord){

       this.keyWord = keyWord;

    }

    protected void sp(Database data,int select) {

       switch(select) {

       case 1:

           //keyWord为游戏类型时，遍历输出游戏类型

           Set<Entry<String,String>> allType = data.getGameNameAndType().entrySet();

           for (Entry<String, String> type : allType) {

              if(type.getValue().contains(keyWord)==true) {

                  System.**out**.println("属于" + keyWord + "类型的游戏有：" + type.getKey());

              }

           }

           break;

       case 2:

           //keyWord为游戏名时，遍历输出游戏名

           Set<String> allGameName = data.getGameNameAndType().keySet();

           for (String gameName : allGameName) {

              if(gameName.equals(keyWord)==true) {

                  System.**out**.println("名为" + keyWord + "的游戏属于：" + data.getGameNameAndType().get(keyWord) + "类型");

              }

           }

           break;

       }

    }

}

class Player{//玩家

    private String id ;

    private int password ;

    private String genders ;

    private int age ;

    public Player(String id,int password,String genders,int age) {//构造方法2

       this.id = id ;

       this.password = password;

       this.genders = genders;

       this.age = age;

    }

    //setter&getter

    public String getId() {

       return id;

    }

    public void setId(String id) {

       this.id = id;

    }

    public int getPassword() {

       return password;

    }

    public void setPassword(int password) {

       this.password = password;

    }

    public String getGenders() {

       return genders;

    }

    public void setGenders(String genders) {

       this.genders = genders;

    }

    public int getAge() {

       return age;

    }

    public void setAge(int age) {

       this.age = age;

    }

}

public class GameCollection01 {

    public static void main(String[] args) {

       start();

    }

    public static void start() {

       System.**out**.println("程序启动，正在初始化……");

       Database data = new Database();//实例化所有对象

       Scanner sc = new Scanner(System.**in**);

       System.**out**.println("初始化完成！");

//     System.out.println("------------------------------------------");

//     System.out.println("已进入游戏信息库查询程序！请选择查询方法：（请输入整数1/2/3）");

//     System.out.println("1.通过游戏名查询");

//     System.out.println("2.通过游戏类型查询");

//     System.out.println("3.通过登录玩家账号查询");

       System.**out**.println("请输入查询关键词：(- 游戏名 - 或  - 游戏类型 -)");

       String keyWord = sc.next();

       SearchPrint spt = new SearchPrint(keyWord);

       System.**out**.println("请选择查询方法：（请输入整数1-2）");

       System.**out**.println("1.游戏类型查询");

       System.**out**.println("2.游戏名查询");

       int select = sc.nextInt();

       System.**out**.println("开始搜寻数据库，请稍等……");

       spt.sp(data, select);

       System.**out**.println("搜寻完毕，程序结束！");

       sc.close();

       System.exit(1);//结束程序

    }

}

我成功了！！

我终于在这晚自习的2个小时里，成功实现了游戏名与游戏类型关联的输出的功能！！

虽然代码还没有写完，但是我已经攻克了当初最大的最困难，如今的我再看这个代码需求，真的是，除了“为什么不用数据库？？？”的问题之外，实在是很简单的练习了。不出意外，明天我晚自习就可以实现这个代码的剩下的用户登录输出信息功能。

从第6天开始遇到这个问题，直到第41天才解决掉这个练习题，一方面感觉到自己和当初比着实变强了不少，同时也觉得时间太漫长了。那时的我写出的代码都是通过各种莫名方法拼凑出来的程序，如今我也能写出一篇逻辑还算通顺的代码了！真是不容易……

那么时间也不早了，今天就到这里吧。

尽管这还只是刚刚开始，我仍然想说一句，真的非常感谢这一路陪伴、教导、支持我的各位前辈，没有你们的支持仅凭我的意志力永远不可能达到如此明显进步的效果。

今后也请多指教！

今日感想总结：谢谢各位，AC在，爱一直在！

今日评分：10/10分

今天就到这里，明天继续投稿

感谢支持！

===============================================

**Acer程序员群：790482850**

**up主的java学习日记word文件及全部代码：**

Adress：github.com/wt62635