****

**软件测试技术**

**课程设计报告**

**课程设计题目：个人博客系统测试**

**日期：2021.07.12**

**班级：软件工程1801班**

**成员1：徐文 41809050108**

**成员2：王森 41809050107**

# 课程设计题目

关于对个人博客系统Web项目的测试工作

# 课程设计内容

本次课程设计将对个人博客系统这个web项目进行一系列操作，主要是进行单元测试，功能测试以及压力测试和接口自动化测试工作。通过idea进行单元测试，继续设计用例进行功能测试，通过idea自动化测试，建立检查点等操作进行测试，最终得到测试结果，整理文档。

# 课程设计环境

**硬件条件：**

一台Win10系统笔记本(8+1T左右内存)

**软件条件：**

操作系统：win10

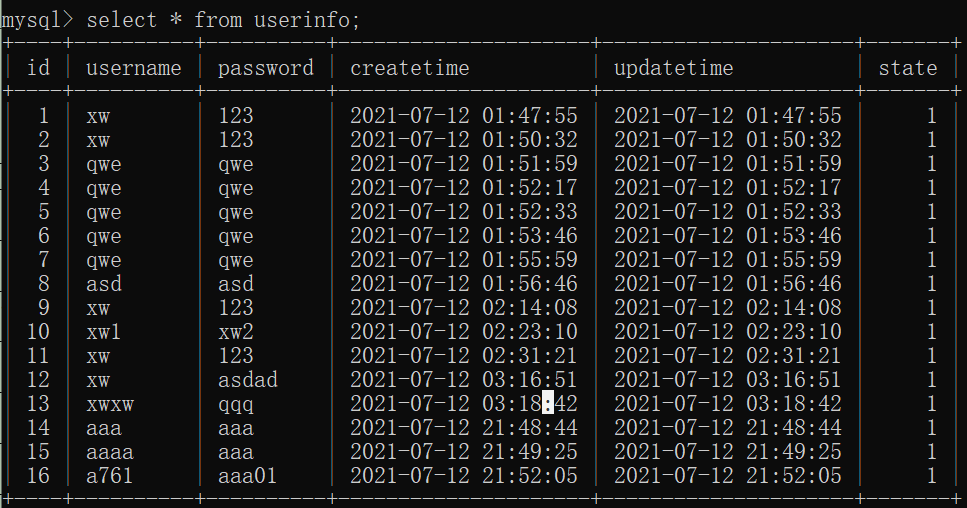
JAVA编程工具：idea

JAVA环境：JDK1.8

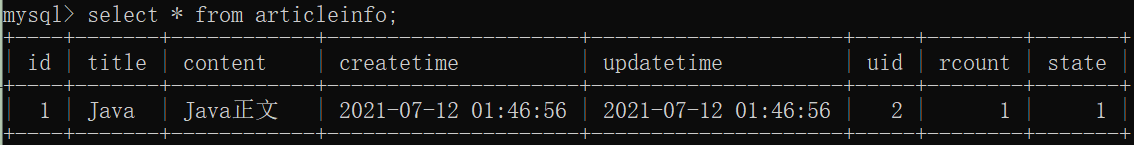
压力测试：软件Jmeter

# 课程设计测试计划

**用户表：**



**文章表：**



# 课程设计测试

**小组成员分工：**

|  |  |  |  |
| --- | --- | --- | --- |
| **负责人** | **学号** | **内 容** | **时间(天)** |
| 全体成员 |  | 测试项目部署、测试计划 | 1 |
| 徐文 | 41809050107 | 压力测试和接口自动化测试 | 1 |
| 王森 | 41809050107 | 黑盒测试与白盒测试 | 1 |

# **白盒测试**

### **1，白盒测试目的**

### 概念：单元测试(Unit Testing)是对软件基本组成单元进行的测试，如函数或是一个类的方法。这里的单元，就是软件设计的最小单位。 步骤：人工静态检查法与动态执行跟踪法。 人工静态检查是测试的第一步，这个阶段工作主要是保证代码算法的逻辑正确性（尽量通过人工检查发现代码的逻辑错误）、清晰性、规范性、一致性、算法高效性，并尽可能的发现程序中没有发现的错误。 第二步是通过设计测试用例，执行待测程序来跟踪比较实际结果与预期结果来发现错误。 2，**白盒测试操作**

**1.人工检查**  
 (1)、检查算法的逻辑正确性：确定所编写的代码算法、数据结构定义（如：队列、堆栈等）是否实现了模块或方法所要求的功能。  
 (2)、模块接口的正确性检查：确定形式参数个数、数据类型、顺序是否正确；确定返回值类型及返回值的正确性。  
 (3)、输入参数有没有作正确性检查：如果没有作正确性检查，确定该参数是否的确无需做参数正确性检查，否则请添加上参数的正确性检查。  
 (4)调用其他方法接口的正确性：检查实参类型正确与否、传入的参数值正确与否、个数正确与否，特别是具有多态的方法。返回值正确与否，有没有误解返回值所表示的意思。最好对每个被调用的方法的返回值用显示代码作正确性检查，如果被调用方法出现异常或错误程序应该给予反馈，并添加适当的出错处理代码。  
 (5)出错处理：模块代码要求能预见出错的条件，并设置适当的出错处理，以便一旦程序出错时，能对出错程序重做安排，保证其逻辑的正确性，这种出错处理应当是模块功能的一部分。若出现下列情况之一，则表明模块的错误处理功能包含有错误或缺陷：出错的描述难以理解；出错的描述不足以对错误定位，不足以确定出错的原因；显示的错误信息与实际的错误原因不符；对错误条件的处理不正确；在对错误进行处理之前，错误条件已经引起系统的干预等。  
 (6)保证表达式、SQL语句的正确性：检查所编写的SQL语句的语法、逻辑的正确性。对表达式应该保证不含二义性，对于容易产生歧义的表达式或运算符优先级 （如：<、=、 >、 &&、||、++、 --等）可以采用扩号“（）”运算符避免二义性，这样一方面能够保证代码的正确可靠，同时也能够提高代码的可读性。  
 (7)、检查常量或全局变量使用的正确性：确定所使用的常量或全局变量的取值和数值、数据类型；保证常量每次引用同它的取值、数值和类型的一致性。  
 (8)、表示符定义的规范一致性：保证变量命名能够见名知意，并且简洁但不宜过长或过短、规范、容易记忆、最好能够拼读。并尽量保证用相同的表示符代表相同功能，不要将不同的功能用相同的表示符表示；更不要用相同的表示符代表不同的功能意义。  
 (9)、程序风格的一致性、规范性：代码必须能保证符合规范，保证所有成员的代码风格一致、规范、工整。

(10)检查程序中使用到的神秘数字是否采用了表示符定义：神秘的数字包括各种常数、数组的大小、字符位置、变换因子以及程序中出现的其他以文字形式写出的数值。在程 序源代码里，一个具有原本形式的数对其本身的重要性或作用没提供任何指示性信息，它们也导致程序难以理解和修改。对于这类神秘数字必须采用相应的标量来表 示；如果该数字在整个系统中都可能使用到务必将它定义为全局常量；如果该神秘数字在一个类中使用可将其定义为类的属性（Attribute）,如果该神秘数字只在一个方法中出现务必将其定义为局部变量或常量。  
 (11)检查代码是否可以优化、算法效率是否最高：如：SQL语句是否可以优化，是否可以用1条SQL语句代替程序中的多条SQL语句的功能，循环是否必要，循环中的语句是否可以抽出到循环之外等。  
 (12)检查您的程序是否清晰简洁容易理解：注意：冗长的程序并不一定不是清晰的。  
 (13)检查方法内部注释是否完整：是否清晰简洁；是否正确的反映了代码的功能，错误的注释比没有注释更糟；是否做了多余的注释；对于简单的一看就懂的代码没有必要注释。  
 (14)检查注释文档是否完整：对包、类、属性、方法功能、参数、返回值的注释是否正确且容易理解；是否会落了或多了某个参数的注释，参数类型是否正确，参数的限 定值是否正确。特别是对于形式参数与返回值中关于神秘数值的注释，如：类型参数 应该指出 1.代表什么，2.代表什么，3.代表什么等。对于返回结果集(Result Set)的注释，应该注释结果集中包含那些字段及字段类型、字段顺序等。  
 2**. 动态执行跟踪：**

动态执行测试通常分为黑盒测试与白盒测试。对于单元测试来说主要应该采用白盒测试法对每个模块的内部作跟踪检查测试。对于单元白盒测试，应该对程序模块进行如下检查：(1)、对模块内所有独立的执行路径至少测试一次；(2)、对所有的逻辑判定，取“真”与“假”的两种情况都至少执行一次；(3)、在循环的边界和运行界限内执行循环体；(4)、测试内部数据的有效性等等。  
4.单元测试的目的：在于发现各模块内部可能存在的各种错误，主要是基于白盒测试。  
单元测试的目的主要有3方面：验证单元代码和详细设计文档的一致性；跟踪详细设计文档中设计的实现，发现详细设计文档中存在的错误；发现在编码过程中引入的错误。  
5. 单元的常见错误：(1)、单元接口；(2)、局部数据结构；(3)、独立路径；(4)、出错处理；(5)、边界条件。

### **3，白盒测试过程**

**一、**

**数据库添加用户和查询用户（白盒测试）**

package dao;

import models.UserInfo;

import org.junit.Assert;

import org.junit.Test;

import java.sql.SQLException;

public class UserInfoTest{

static UserInfoDao userInfoDao = new UserInfoDao();

/\*\*

\* 添加不存在的用户

\*/

@Test

public void add\_test() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("qqqqqqqqq");

userInfo.setPassword("qqqqqqqqq");

boolean flag = userInfoDao.add(userInfo);

Assert.assertTrue(flag);

}

/\*\*

\* 添加null用户

\*/

@Test

public void add\_test1() throws SQLException {

boolean flag = userInfoDao.add(null);

Assert.assertFalse(flag);

}

/\*\*

\* 添加已经存在的用户

\*/

@Test

public void add\_test2() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("xw");

userInfo.setPassword("123");

boolean flag = userInfoDao.add(userInfo);

Assert.assertFalse(flag);

}

/\*\*

\* 添加用户名为空的用户

\*/

@Test

public void add\_test3() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("");

userInfo.setPassword("123");

boolean flag = userInfoDao.add(userInfo);

Assert.assertFalse(flag);

}

/\*\*

\* 添加密码为空的用户

\*/

@Test

public void add\_test4() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("asd");

userInfo.setPassword("");

boolean flag = userInfoDao.add(userInfo);

Assert.assertFalse(flag);

}

/\*\*

\* 添加用户名长度大于5的用户

\*/

@Test

public void add\_test5() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("asdaaa");

userInfo.setPassword("");

boolean flag = userInfoDao.add(userInfo);

Assert.assertFalse(flag);

}

/\*\*

\* 添加密码长度大于5的用户

\*/

@Test

public void add\_test6() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("asd");

userInfo.setPassword("asdasd");

boolean flag = userInfoDao.add(userInfo);

Assert.assertFalse(flag);

}

/\*\*

\* 测试查找存在的用户

\*/

@Test

public void getUserInfo\_test() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("xw");

userInfo.setPassword("123");

UserInfo userInfo1 = userInfoDao.getUserInfo(userInfo);

Assert.assertNotEquals(0,userInfo1.getId());

}

/\*\*

\* 测试查找不存在的用户

\*/

@Test

public void getUserInfo\_test1() throws SQLException {

UserInfo userInfo = new UserInfo();

userInfo.setUsername("zzzzzzzzzzzzzzz");

userInfo.setPassword("zzzzzzzzzzzzzzz");

UserInfo userInfo1 = userInfoDao.getUserInfo(userInfo);

Assert.assertEquals(0,userInfo1.getId());

}

}

# 注册页面(黑盒测试)

### 黑盒测试目的

黑盒测试(black—box testing)又称功能测试、数据驱动测试或基于规范的测试。用这种方法进行测试时，被测程序被当作看不见内部的黑盒。在完全不考虑程序内部结构和内部特性的情况下，测试者仅依据程序 功能的需求规范考虑确定测试用例和推断测试结果的正确性。因此黑盒测试是从用户观点出发的测试，黑盒测试直观的想法就是既然程序被规定做某些事，那我们就看看它是不是在任何情况下都做的对。完整的“任何情况”是无法验证的，为此黑盒测试也有一套产生测试用例的方法，以产生有限的测试用例而覆盖足够多的“任何情况”。由于黑盒测试不需要了解程序内部结构，所以许多高层的测试如确认测试、系统测试、验收测试都采用黑盒测试。

黑盒测试首先是程序通常的功能性测试。要求：每个软件特性必须被一个测试用例或一个被认可的异常所覆盖；用数据类型和数据值的最小集测试；用一系列真实的数据类型和数据值运行，测试超负荷、饱和及其他“最坏情况”的结果；用假想的数据类型和数据值运行，测试排斥不规则输入的能力；对影响性能的关键模块，如基本算法、应测试单元性能(包括精度、时间、容量等)。不仅要考核“程序是否做了该做的?”还要考察“程序是否没做不该做的2”同时还要考察程序在其他一些情况下是否正常。这些情况包括数据类型和数据值的异常等等。 下述几种方法：

1. 等价类划分，
2. 因果图方法，
3. 边值分析法，
4. 猜错法，
5. 随机数法，

就是从更广泛的角度来进行黑盒测试。每一个方法都力图能涵盖更多的“任何情况”，但又各有长处，综合使用这些方法，会得到一个较好的测试用例集。

### 黑盒测试操作

等价类划分

等价类划分是一种典型的黑盒测试方法。等价类是指某个输入域的集合。它表示对揭露程序中的错误来说，集合中的每个输入条件是等效的。因此我们只要在一个集合中选取一个测试数据即可。等价类划分的办法是把程序的输入域划分成若干等价类，然后从每个部分中选取少数代表性数据当作测试用例。这样就可使用少数测试用例检验程序在一大类情况下的反映。

在考虑等价类时，应该注意区别以下两种不同的情况：

有效等价类：有效等价类指的是对程序的规范是有意义的、合理的输入数据所构成的集合。在具体问题中，有效等价类可以是一个，也可以是多个。

无效等价类：无效等价类指对程序的规范是不合理的或无意义的输入数据所构成的集合。对于具体的问题，无效等价类至少应有一个，也可能有多个。

确定等价类有以下几条原则：如果输入条件规定了取值范围或值的个数，则可确定一个有效等价类和两个无效等价类。 输入条件规定了输入值的集合，或是规定了“必须如何”的条件，则可确定一个有效等价类和一个无效等价类。如某程序涉及标识符，其输入条件规定“标识符应以字母开头……”则“以字母开头者”作为有效等价类，“以非字母开头”作为无效等价类。设计一个测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，最后使得所有有效等价类均被测试用例所覆盖；

设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步，使所有无效等价类均被覆盖。这里强调每次只覆盖一个无效等价类。这是因为一个测试用例中如果含有多个缺陷，有可能在测试中只发现其中的一个，另一些被忽视。等价类划分法能够全面、系统地考虑黑盒测试的测试用例设计问题，但是没有注意选用一些“高效的”、“有针对性的”测试用例。后面介绍的边值分析法可以弥补这一缺点。

因果图

等价类划分法并没有考虑到输入情况的各种组合。这样虽然各个输入条件单独可能出错的情况已经看到了，但多个输入情况组合起来可能出错的情况却被忽略。采用因果图方法能帮助我们按一定步骤选择一组高效的测试用例，同时，还能为我们指出程序规范的描述中存在什么问题。 利用因果图导出测试用例需要经过以下几个步骤： 分析程序规范的描述中哪些是原因，哪些是结果。原因常常是输入条件或是输入条件的等价类。结果是输出条件。 分析程序规范的描述中语义的内容，并将其表示成连接各个原因与各个结果的“因果图”。 由于语法或环境的限制，有些原因和结果的组合情况是不可能出现的。为表明这些特定的情况，在因果图上使用持殊的符号标明约束条件。把因果图转换成判定表。把判定表的每一列写成一个测试用例。

边值分析法

边值分析法是列出单元功能、输入、状态及控制的合法边界值和非法边界值，设计测试用例，包含全部边界值的方法。典型地包括IF语句中的判别值，定义域、值域边界，空或畸形输入，末受控状态等。边值分析法不是一类找一个例子的方法，而是以边界情况的处理作为主要目标专门设计测试用例的方法。另外，边值分析不仅考查输入的边值，也要考虑输出的边值。这是从人们的经验得出的一种有效方法。人们发现许多软件错误只是在下标、数据结构和标量值的边界值及其上、下出现，运行这个区域的测试用例发现错误的概率很高。

用边值分析法设计测试用例时，有以下几条原则：如果输入条件规定了取值范围，或是规定了值的个数，则应以该范围的边界内及刚刚超出范围的边界外的值，或是分别对最大、最小及稍小于最小、稍大于最大个数作为测试用例。如有规范“某文件可包含l至255”个记录……“，则测试用例可选1和255及0和256等。

### 3，黑盒测试过程

1. 用户名为空

预期：给出弹窗提示



1. 密码为空

预期：给出弹窗提示



1. 两次密码输入不一致

预期：给出弹窗提示



1. 创建用户名和密码已经存在的用户

预期：给出弹窗提示



1. 用户名长度超过限制

预期：给出弹窗提示



1. 密码长度超过限制

预期：给出弹窗提示



1. 正常操作

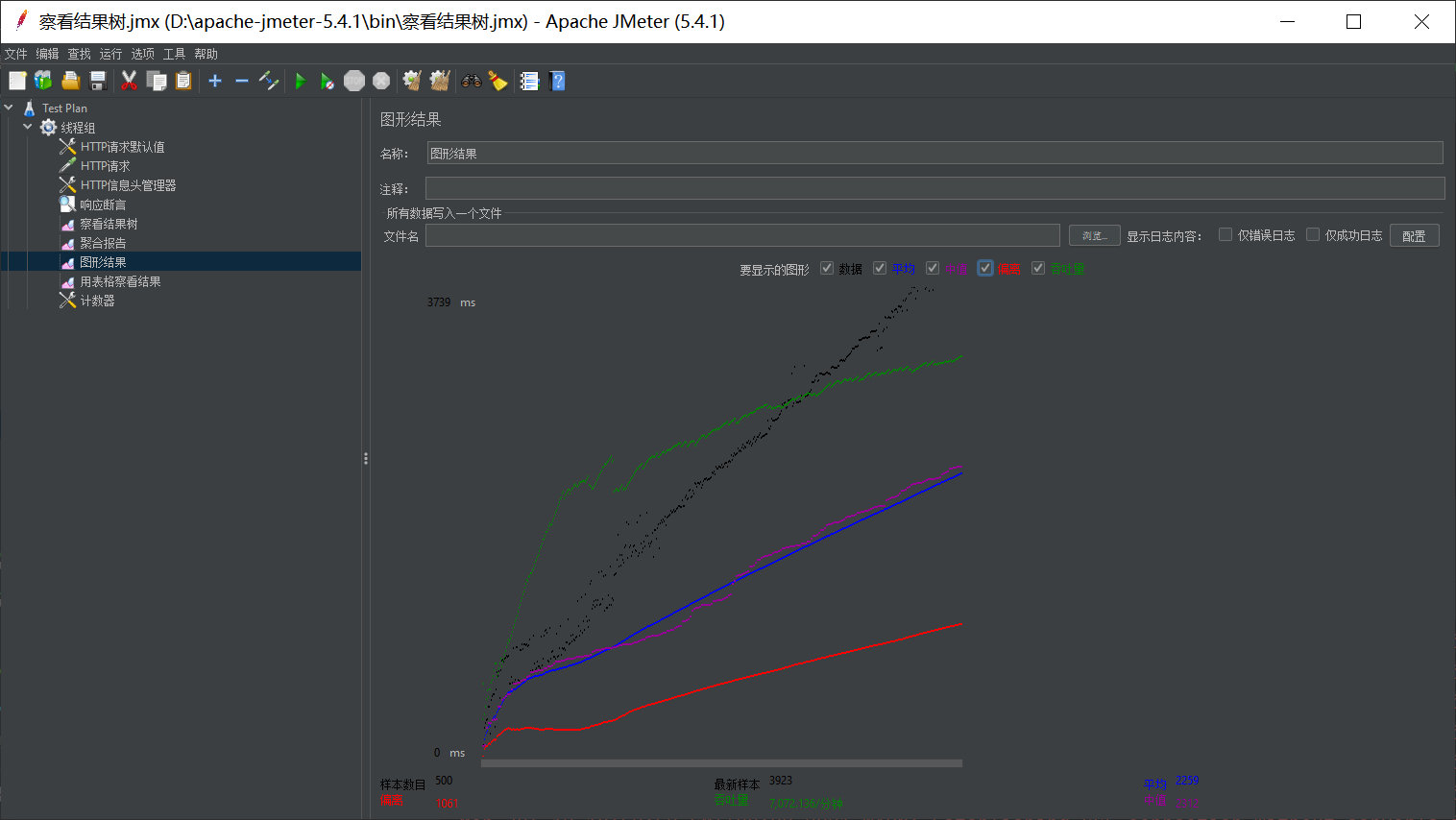
预期：给出弹窗提示：创建成功并跳转登录界面



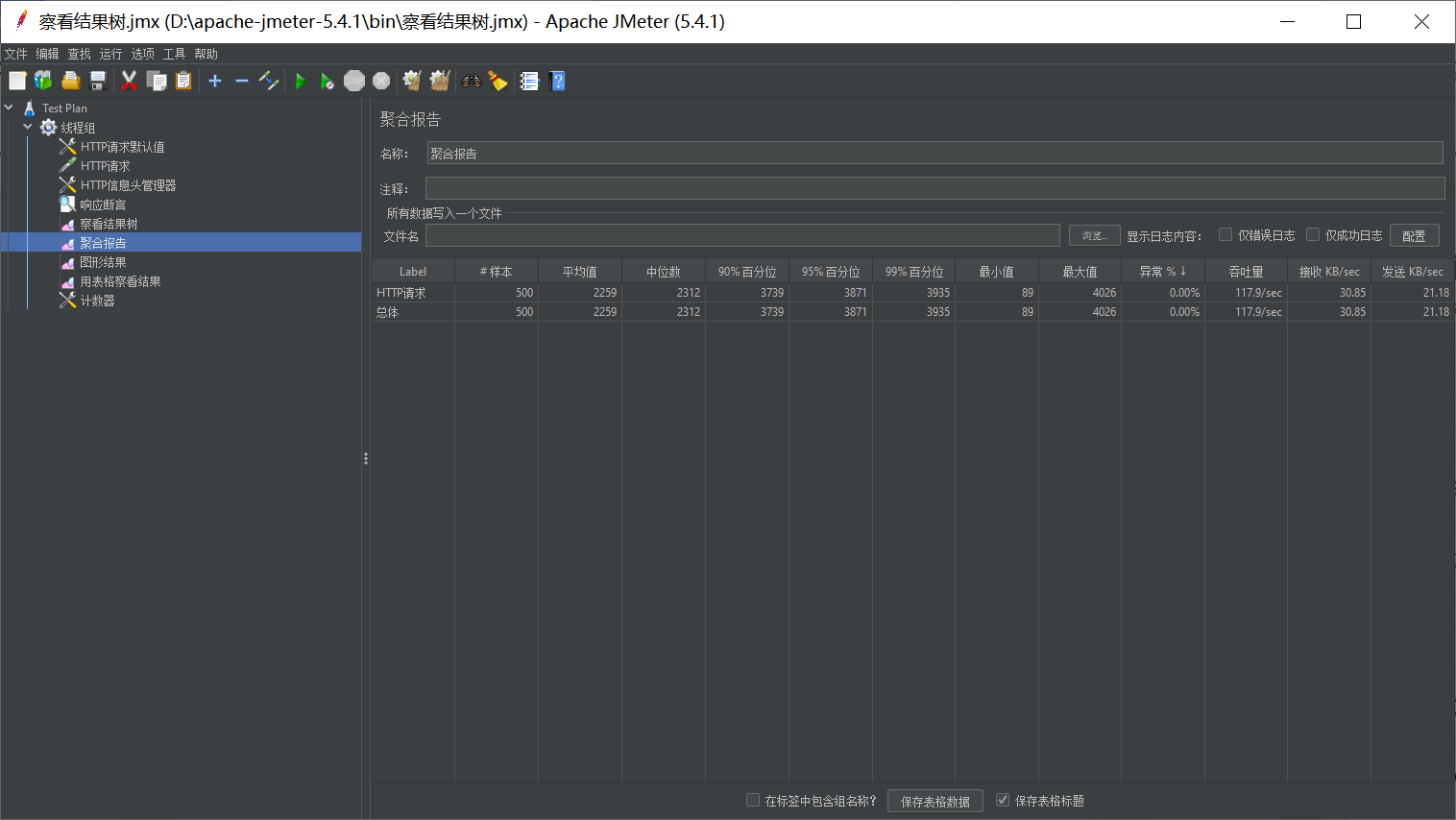


# 八、压力测试

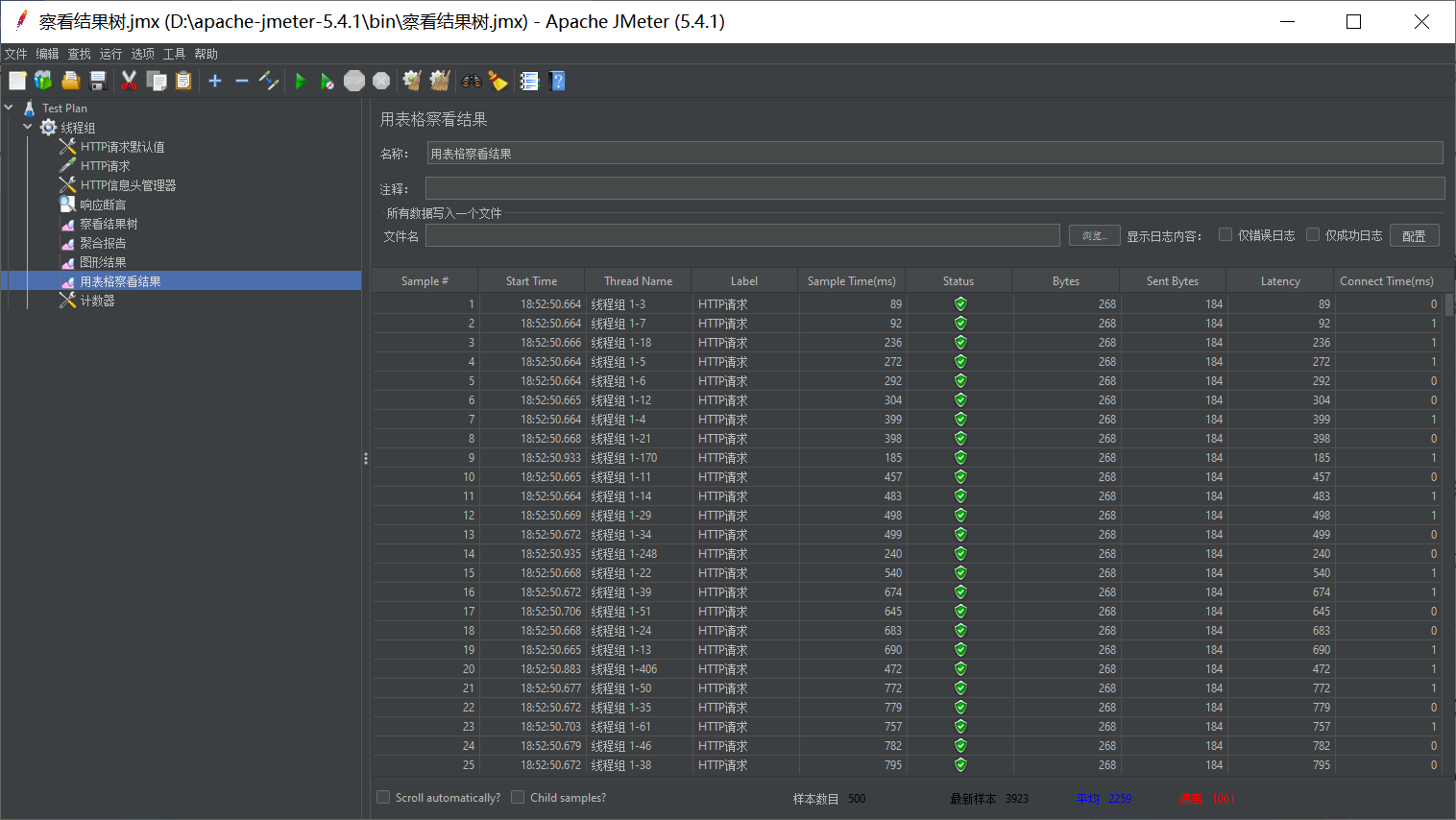
1. 图形结果(结果图形化展示)



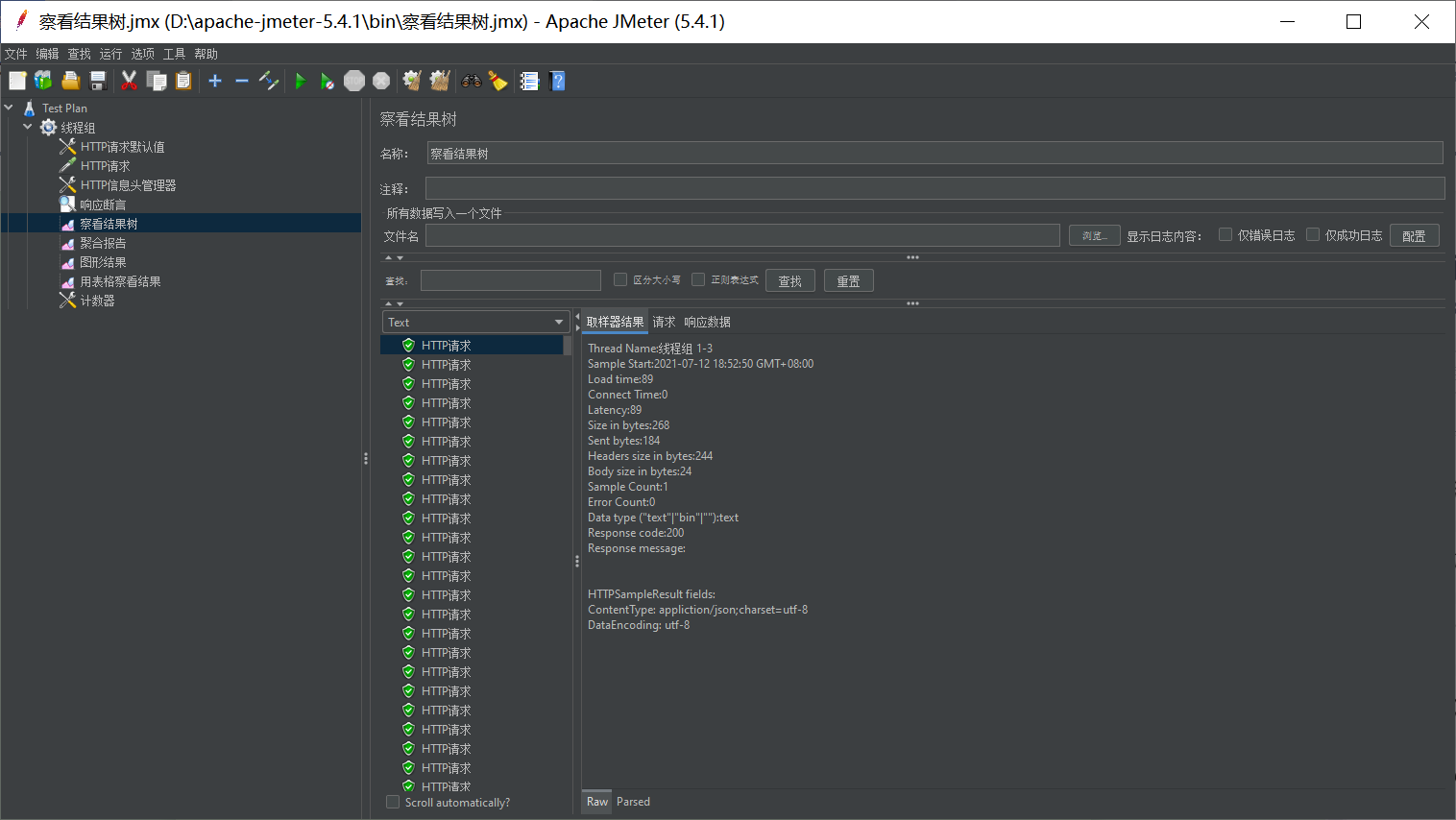
1. 聚合报告 异常0.00%



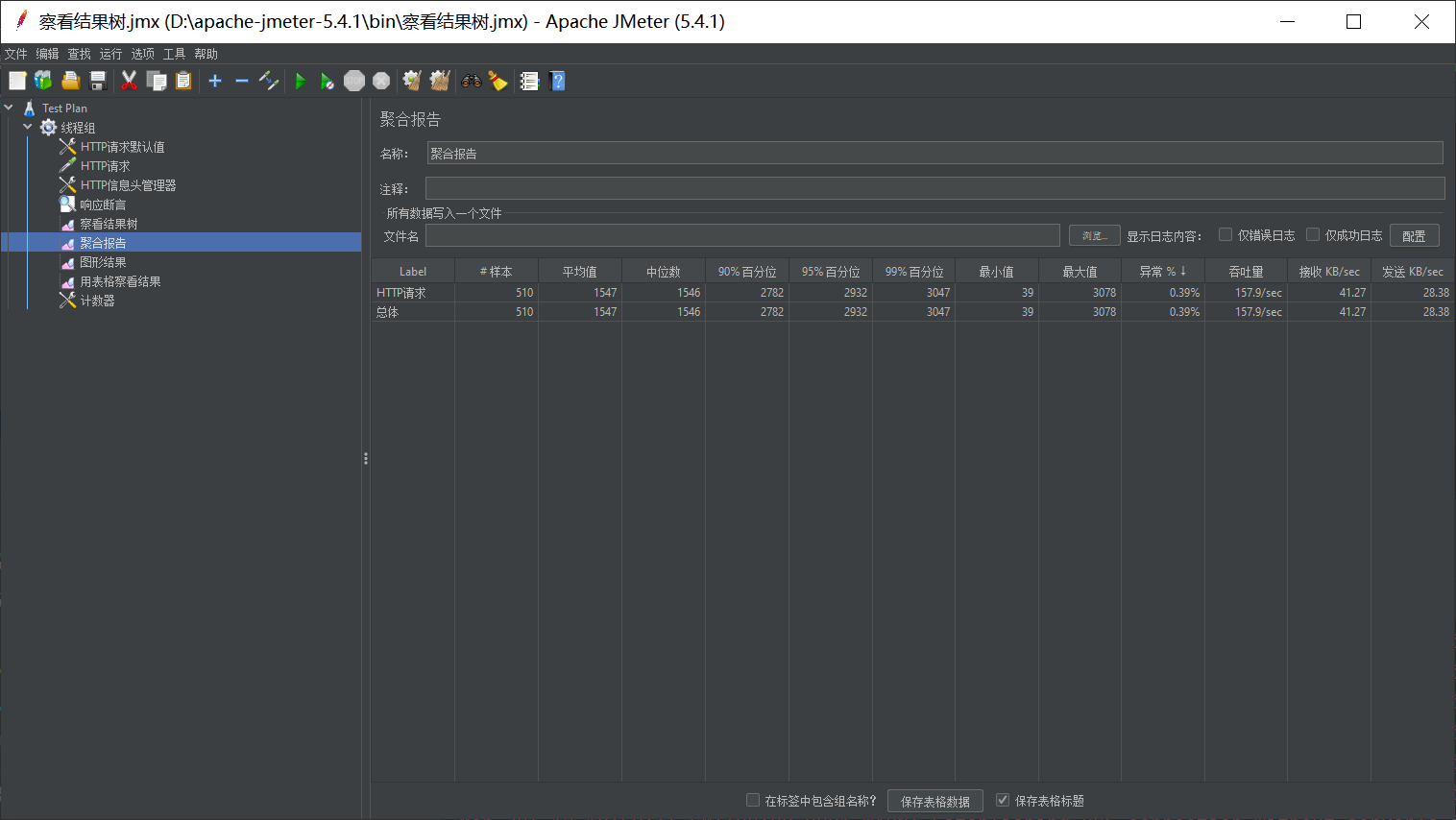
1. 用表格查看结果

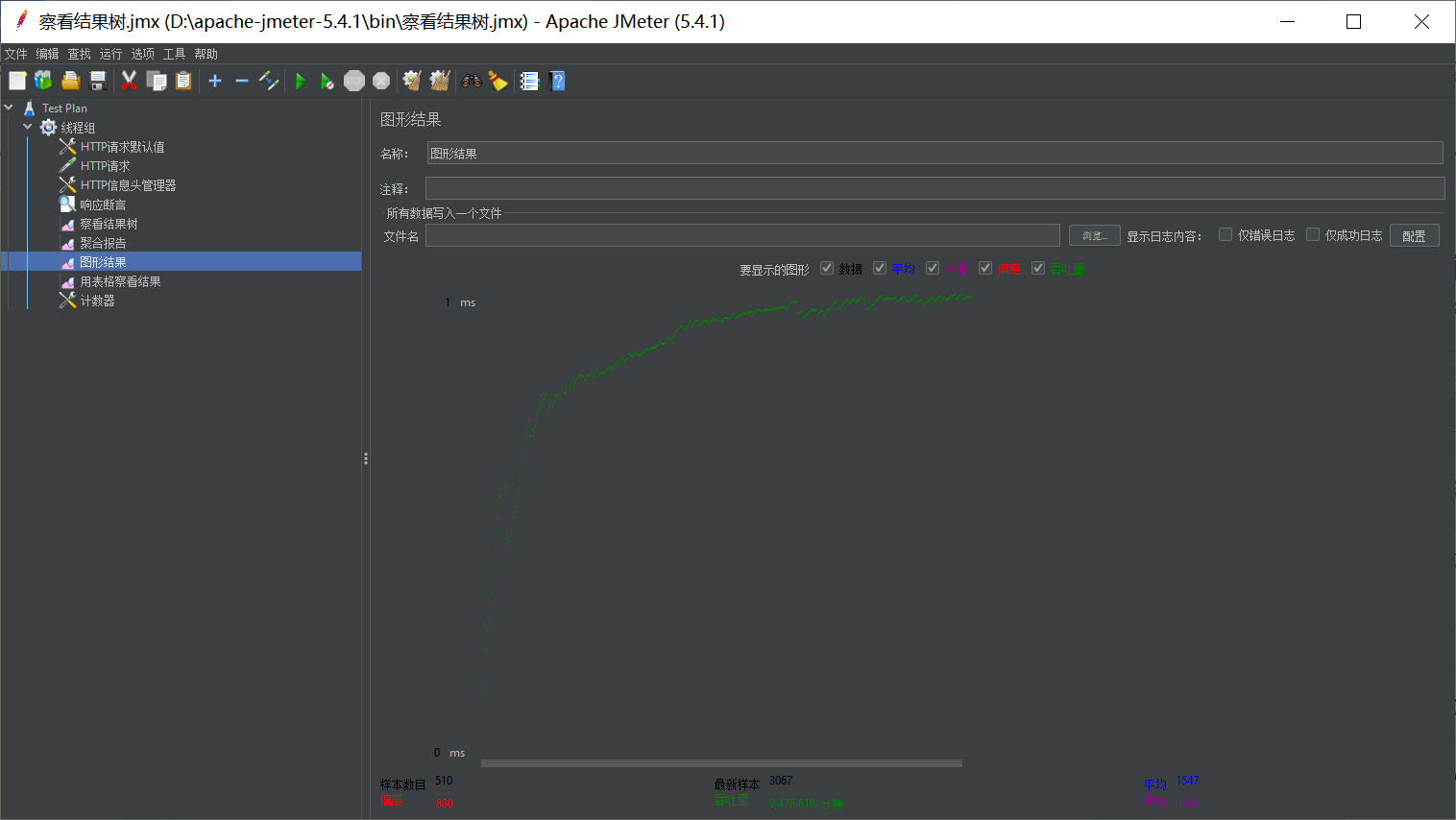


1. 查看结果树

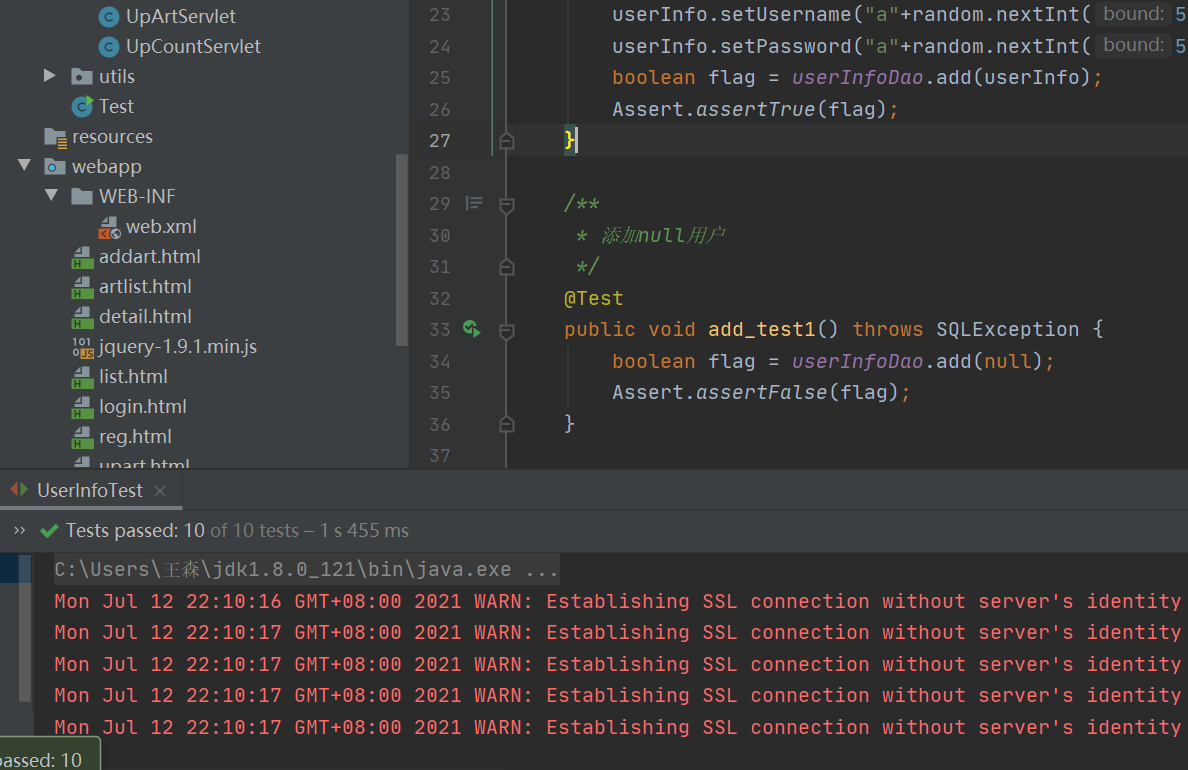


1. 扩大线程量，增加负载 从500增加到510





# 自动化测试登录接口



代码：

package dao;

import org.apache.http.HttpEntity;

import org.apache.http.HttpResponse;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.impl.client.DefaultHttpClient;

import org.junit.Assert;

import org.junit.Test;

import java.io.IOException;

import java.util.Arrays;

public class LoginTest {

/\*\*

\* 测试登录接口，账号为：正确的用户名和密码

\*/

@Test

public void LoginTest() throws IOException {

HttpGet httpGet = new HttpGet("http://127.0.0.1:8080/xw/login?username=xw&password=123");

HttpClient client = new DefaultHttpClient();

HttpResponse execute = client.execute(httpGet);

Assert.assertEquals(200,execute.getStatusLine().getStatusCode());

}

/\*\*

\* 测试登录接口，账号为：不传账号和密码

\*/

@Test

public void LoginTest1() throws IOException {

HttpGet httpGet = new HttpGet("http://127.0.0.1:8080/xw/login");

HttpClient client = new DefaultHttpClient();

HttpResponse execute = client.execute(httpGet);

HttpEntity entity = execute.getEntity();

byte[] bys = new byte[1024];

entity.getContent().read(bys);

String s = new String(bys);

String[] split = s.split(":");

String[] split1 = split[split.length - 1].split("}");

System.out.println(split1[0]);

Assert.assertEquals(split1[0],"-1");

}

}

# 十、课设结果报告

本次课程设计，我负责个人博客系统的需求分析、Java程序编写、数据库表的建立，对整体框架进行构建，设计系统的功能和用途，可实现性，可开发性。

在java程序开发中遇到很多问题，类方法的调用等，还有在Java程序和数据库之间的连接中遇到问题，也在查阅资料后解决了，测试也遇到很多问题，但都一一查阅资料解决啦，总体来说这次课程设计还是比较圆满的。设计确实有些辛苦，但苦中有乐，在如今单一的理论学习中，很少有机会能有实践的机会，期待下一次课程设计。