

西安电子科技大学

# web 应用测试

---

15130140339 杨琦

2018/7/6

对构建的 web 应用（部分功能）进行测试

目录

概述.....3

测试目标.....3

测试对象.....3

应用测试.....3

    单元测试.....3

    功能测试.....5

    性能测试.....6

    安全性测试.....10

    SQL 注入测试.....12

    WebUI 测试.....12

总结与建议.....14

# 概述

随着 Web 应用开发技术和应用水平的飞速发展，用户对 Web 系统的功能、性能、安全性、稳定性等提出了更高的要求。Web 应用程序在发布之前必须进行深入全面的测试。

## 测试目标

发现错误，尤其是关键错误。完全的测试覆盖率是不可能的，因此测试的重点是减轻（最大）风险。

## 测试对象

主要针对对象为用户、软件开发人员和软件测试人员。

## 应用测试


### 单元测试

一个单元测试从整个系统中单独检验产品程序代码的『一个单元』并检查其得到的结果是否是预期的。要测试的一个单元，其大小是依据一组连贯的功能的大小及介于一个类别及一个 `package` 之间实际上的变化。其目的是在整合程序代码到系统的其余部分之前先测试以便找出程序代码中的 bugs。Junit, TestNG, Rspec 等支持在 Java 程序代码中撰写单元测试。

## 测试工具：Junit 4

### 测试过程：

1. 在项目中导入 Junit4 包

>  JUnit 4

2. 分别为每个 java 文件写 test 类，测试 Java 文件中所有方法

BookDao.java

BookDaoTest.java

Book.java

BookTest.java

BookService.java

BookServiceTest.java

BookServlet.java

BookServletTest.java

CategoryDao.java

CategoryDaoTest.java

Category.java

CategoryTest.java

CategoryService.java

CategoryServiceTest.java

CategoryServlet.java

CategoryServletTest.java

UserDao.java

UserDaoTest.java

User.java

UserTest.java

UserException.java

UserService.java

UserServiceTest.java

UserServlet.java

UserServletTest.java

结果: Errors: 0  
Failures: 0

## 功能测试

### 1.关于登录的测试

快捷键的使用是否正常:

- (1) Tab 键的使用是否正确
- (2) 上下左右键的使用是否正确
- (3) 若界面支持 ESC 键, 是否能正常工作
- (4) Enter 键的使用是否正确; 切换时是否正常

输入框的功能:

- (1) 输入合法的用户名和密码, 登录成功
- (2) 输入合法的用户名和不合法的密码, 登录失败, 并给出合理提示
- (3) 输入不合法的用户名和合法的密码, 登录失败, 并给出合理提示
- (4) 输入不合法的用户名和不合法的密码, 登录失败, 并给出合理提示

备注:

- (1) 不合法的用户名: 错误的用户名、使用字符大于用户名的限制的用户名、正常用户名不允许的特殊字符、空用户名
- (2) 不合法的密码: 错误的密码, 空密码、字符大于密码的限制、正常密码不允许的特殊字符

- (3) 验证用户名是否区分大小写
- (4) 验证必填项为空是否允许登录

## 2.关于搜索的测试

- (1) 默认条件的搜索
- (2) 按分类进行搜索
- (3) 错误，空记录的搜索
- (4) 搜索结果显示

测试结果：全部正确通过

## 性能测试

性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。负载测试和压力测试都属于性能测试，两者可以结合进行。通过负载测试，确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

测试工具：Httpunit

测试过程：

首先创建一个请求（WebRequest），然后让 WebConversation 返回响应（WebResponse）。如下：

```
    新创建一个“浏览器”对象 WebConversation wc = new  
WebConversation();  
  
// WebRequest 类，用于模仿客户的“请求”，通过它可以向服务器发送信息。  
  
// WebResponse 类，用于模仿浏览器获取服务器端的响应信息。  
  
WebResponse resp = wc.getResponse ( req );
```

1.获取指定页面的内容， 通过 getText 直接获取页面的所有内容

```
// 建立一个“浏览器”实例
```

```
WebConversation wc = new WebConversation();
```

```
// 将指定 URL 的请求传给 wc，然后获取相应的响应
```

```
// 用 wc 的 getText 方法获取相应的全部内容
```

2.处理页面的链接：

```
// 建立一个 WebConversation 实例
```

```
WebConversation wc = new WebConversation();
```

```
// 获取响应对象
```

```
// 获得页面链接对象
```

```
WebLink link = resp.getLinkWith( "图书列表 " );
```

```
// 模拟用户单击事件
```

```
link.click();
```

```
// 获得当前的响应对象
```

```
WebResponse nextLink = wc.getCurrentPage();
```

```
// 用 getText 方法获取相应的全部内容，并打印
```

### 3.处理页面的表格 (table)

在 HttpUnit 中使用数组来处理页面中的多个表格, 可以用 resp.getTables() 方法获取页面所有的表格对象。将它们依次出现在页面中的顺序保存在一个数组里。

```
// 创建一个 WebConversation 对象
```

```
WebConversation wc = new WebConversation();
```

```
// 设置 HTTP 代理服务器地址和端口
```

```
// 新建一个 URL 请求对象 req
```

```
WebRequest req = new GetMethodWebRequest("#");
```

```
// 发出一个请求 req，并取得它相对应的响应 resp
```



```
WebResponse resp = wc.getResponse(req);
```

```
// 获得响应的页面中的 Table
```

```
WebTable[] tables = resp.getTables();
```

```
// 取出第一个 table
```

```
WebTable table = tables[0];
```

```
// 从 2 * 2 的 table 取出 cell 里的值
```

#### 4. 处理页面的表单 (form)

在 HttpUnit 中使用数组来 处理页面中的多个表单，用 resp.getForms()方法获取页面所有的表单对象。他们依照出现在页面中的顺序保存在一个数组里面。

```
// 建立一个 WebConversation 实例
```

```
WebConversation wc = new WebConversation();
```

```
// 获取响应对象
```

```
WebResponse resp = wc.getResponse( "#" );
```

```
// 获得对应的表单对象
```

```
WebForm webForm = resp.getForms()[0];
```

```
// 获得表单中所有控件的名字
```

```
String[] pNames = webForm.getParameterNames();
```

```
int i = 0;
```

```
int m = pNames.length;
```

```
// 循环显示表单中所有控件的内容
```

```
while(i 里面的内容是"+webForm.getParameterValue(pNames[i]))
```

```
++i;
```

## 安全性测试

### 上传功能

- 绕过文件上传检查功能
- 上传文件大小和次数限制

### 注册功能

- 注册请求是否安全传输
- 注册时密码复杂度是否后台校验
- 激活链接测试
- 重复注册
- 批量注册问题

### 登录功能

- 登录请求是否安全传输
- 会话固定：**Session fixation attack**(会话固定攻击)是利用服务器的 **session** 不变机制，借他人之手获得认证和授权，然后冒充他人。

- 关键 cookie 是否 HTTPONLY: 如果 Cookie 设置了 HttpOnly 标志, 可以在发生 XSS 时避免 JavaScript 读取 Cookie。

但很多 Cookie 需要给前端 JS 使用。所以这里只需要关注关键 Cookie,即唯一标识用户及登录状态的会话标识需要添加这个属性。

- 登录请求错误次数限制
- “记住我”功能: 勾选“记住我”后, Cookie 中记录了用户名和密码信息。。。
- 本地存储敏感信息

### 验证码功能

- 验证码的一次性
- 验证码绕过
- 短信验证码轰炸: 如果这个接口没有限制策略, 就会被人恶意利用

### 忘记密码功能

- 通过手机号找回: 不过由于程序设计不合理, 导致可以绕过短信验证码, 从而修改别人的密码。(使用 burpsuite 抓包, 修改响应值 true)
- 通过邮箱找回

### 密码安全性要求

- 密码复杂度要求
- 密码保存要求

### 横向越权测试

不同用户之间 session 共享, 可以非法操作对方的数据。

### 纵向越权测试

很多应用简单的通过前端判断, 或者低权限角色看不到对应的菜单, 但并未在后台去做当前登录用户是否有权限。

- XSS 测试

跨站脚本攻击(Cross Site Scripting): 恶意攻击者往 Web 页面里插入恶意 Script 代码, 当用户浏览该页之时, 嵌入其中 Web 里面的 Script 代码会被执行, 从而达到恶意攻击用户的目的。

处理意见: 对特殊字符转义输出, 特别是"<>"。

## SQL 注入测试

SQL 注入攻击的基本原理是通过构建特殊的输入参数，迫使后台数据库执行额外的 SQL 语句，从而达到获取数据库数据的目的。

这些输入参数往往包含恶意的 SQL 注入语句，后台处理程序没有对这些参数进行过滤，且所使用的数据库查询手段为拼接方式，进而导致敏感数据外泄。

在动态构造 SQL 语句的过程中，除了特殊字符处理不当引起的 SQL 注入之外，错误处理不当也会为 Web 站点带来很多安全隐患。

最常见的问题就是将详细的内部错误信息显示给攻击者。这些细节会为攻击者提供与网站潜在缺陷相关的重要线索。

在 SQL 注入的过程中，如果 Web 服务器关闭了错误回显，那么是不是就安全了呢？答案显然是否定的，攻击者仍然可以通过 "盲注"技巧测试 SQL 命令是否注入成功。

所谓"盲注"就是在服务器没有错误回显时完成的注入方式，攻击者必须找到一个方法来验证注入的 SQL 语句是否执行。

"盲注"主要分为两种类型：基于时间的盲注和布尔盲注。

**测试方法（黑盒）：**sqlmap 是一个自动化的 SQL 注入工具，其主要功能是扫描，发现并利用给定的 URL 的 SQL 注入漏洞，

**测试方法（白盒）：**如果项目的数据库持久层框架是 mybatis，并且他的 sqlmap 中编写方式都是使用#{xxx}方式，而非使用\${xxx}方式，就不存在 SQL 注入问题。

## WebUI 测试

selenium 和 webdriver 是目前主流的 ui 自动化测试框架之一，selenium 又称为 selenium RC，基本原理为 js 注入，而 webdriver 是直接利用了浏览器的 native support（厂商支持）来操作浏览器，所以，对于不同浏览器，必须依赖一个特定的浏览器 native component 来实现把 webdriver API 转化为浏览器的 native invoke。在我们 new 出一个 webdriver 时，selenium 首先会确认浏览器的 native component 是否存在且版本匹配（所以在使用浏览器驱动时，

需要检查该驱动版本与 **selenium** 的版本是否匹配，不匹配则不可用)，接着在目标浏览器中启动一整套的 **Web service**，这套 **web service** 使用了 **selenium** 自己设计定义的协议，可以模拟用户操作浏览器做出一系列动作。

测试工具：python 和 selenium

.# ERP 自动化登录接口

```
def erp_login(cls, username, password):
```

```
    driver = cls.driver
```

```
    driver.find_element_by_id("btn-front").click()
```

# 登录名称

```
    driver.find_element_by_id("DloginName").send_keys(username)
```

# 登录密码

```
    driver.find_element_by_id("Dpassword").send_keys(password)
```

```
    driver.find_element_by_id("Flogin").click()
```

之后通过构建用例集来执行用例，并发送邮件

定义测试任务集

```
testreport = unittest.TestSuite()
```

# 给测试任务集添加测试用例任务的名称

```
testreport.addTest(TestErp("test_case_1"))
```

```
testreport.addTest(TestErp("test_case_2"))
```

```
testreport.addTest(TestErp("test_case_3"))
```

```
testreport.addTest(TestErp("test_case_4"))
```

```
testreport.addTest(TestErp("test_case_5"))
```

```
testreport.addTest(TestErp("test_case_6"))
```

```
testreport.addTest(TestErp("test_case_7"))
```

```
testreport.addTest(TestErp("test_case_8"))
```

```
testreport.addTest(TestErp("test_case_9"))
```

```
testreport.addTest(TestErp("test_case_10"))
```

```
testreport.addTest(TestErp("test_case_11"))
```

```
testreport.addTest(TestErp("test_case_12"))
```

```
testreport.addTest(TestErp("test_case_13"))
```

```
testreport.addTest(TestErp("test_case_14"))
```

```
testreport.addTest(TestErp("test_case_15"))
```

```
testreport.addTest(TestErp("test_case_16"))
```

# 定义测试报告的文件路径

```
file_path = "D:\\pythonproject\\webbookstore " + reportname
```

```
fp = file(file_path, "wb")
```

```
runner = HTMLTestRunner.HTMLTestRunner(stream=fp, title=u"测试报告",
```

```
description=u"用例执行情况：")
```

```
# 执行测试任务集
runner.run(testreport)
fp.close() # 关闭文件
# 根据文件路径、定义附件名称发送邮件
unittest_smtp(file_path, reportname)
```

测试结果：成功接收邮件。

## 总结与建议

从测试过程中发现 bug 的严重程度与分布状况来看，引起缺陷主要有以下几方面：

### 1.需求不明确

在项目开始的时候没有明确的需求文档，使得开发人员进行设计的时候，没有考虑相关的功能需求性，开发人员根据自己的理解进行设计，使得结果有所偏差。

### 2.功能性错误

在测试的过程中，部分功能没有实现，导致部分模块无法进行相关的测试。

建议：

项目开始的时候，应该制定相应的标准，开发人员和测试人员应严格按照标准进行，可以减少后期改动的花费。