

软件测试 实验报告—EasyMock

1、实验目的

本实验主要是完成不借助 J2EE 容器下的 LoginServlet 的测试。

- 1、使用 Junit 进行测试。
- 2、利用 EasyMock 技术来辅助 Junit 完成测试。

2、测试范围

2.1 LoginServlet 的 doPost 方法，包括验证正确的登录方法和错误的登录方法。

3、测试过程

3.1 确定需要 EasyMock 的对象和其需要的能力（属性或者方法）。

- 1、创建 LoginServlet 对象，并使用其 init 方法注入模拟的 ServletConfig，然后链式模拟 ServletContext 和 RequestDispatcher
- 2、模拟 HttpServletRequest 和 HttpServletResponse
- 3、使用 ResultHook 单例对象捕获运行结果。

3.2 编写 Junit 的测试用例

我们使用 HashMap 来模拟账号和密码的传参。然后使用 ResultHook 来定义结果，使用”ok”表示成功登录，”fail”表示登录失败。

具体的测试用例导入方法沿用实验一，因此不再此处介绍。

根据白盒测试和黑盒测试的原则，设计以下几组测试用例。

账号正确	密码正确	(admin,123456,ok)
账号正确	密码错误（前缀相等）	(admin,1234567,fail)
	密码错误（后缀相等）	(admin,0123456,fail)
	密码错误（长度相等）	(admin,124456,fail)
	密码错误（缺少几位）	(admin,12345,fail)
	密码错误（null）	(admin,null,fail)
	密码错误（空）	(admin,/,fail)
	密码错误（无该参数）	(admin,~,fail)
账号错误	密码正确	在这里和密码错误类似，故省略

账号错误	密码错误	(admi,1234567,fail)
无参数输入		(~,~,fail)
比预期更多的参数输入		(admin,123456,_gmmode=1,ok)

3.3 编写代码

MockFactory 的代码，用于生成模拟对象：

```
package org.tty.test2.foundation;

import org.easymock.EasyMock;
import org.tty.test2.LoginServlet;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.HashMap;

public class MockFactory {
    public static RequestDispatcher mockRequestDispatcher() {
        ResultHook.getInstance().setValue("no_init");
        return new RequestDispatcher() {
            @Override
            public void forward(ServletRequest request, ServletResponse response) throws
ServletException, IOException {

                ResultHook.getInstance().setValue("ok");
            }

            @Override
            public void include(ServletRequest request, ServletResponse response) throws
ServletException, IOException {

            }
        };
    }

    public static ServletContext mockServletContext() {
        ServletContext servletContext = EasyMock.createMock(ServletContext.class);

        EasyMock.expect(servletContext.getNamedDispatcher("dispatcher")).andReturn(mockRequ
estDispatcher());
        EasyMock.replay(servletContext);
        return servletContext;
    }
}
```

```

    }

    public static ServletConfig mockServletConfig() {
        ServletConfig servletConfig = EasyMock.createMock(ServletConfig.class);

        EasyMock.expect(servletConfig.getServletContext()).andReturn(mockServletContext());
        EasyMock.replay(servletConfig);
        return servletConfig;
    }

    public static LoginServlet mockLoginServlet() {
        LoginServlet loginServlet = EasyMock.createMock(LoginServlet.class);

        EasyMock.expect(loginServlet.getServletContext()).andReturn(mockServletContext());

        EasyMock.replay(loginServlet);
        return loginServlet;
    }

    public static HttpServletRequest mockHttpServletRequest(HashMap<String, String>
dic) {
        HttpServletRequest httpServletRequest =
        EasyMock.createMock(HttpServletRequest.class);
        dic.forEach((key, value) -> {
            EasyMock.expect(httpServletRequest.getParameter(key)).andReturn(value);
        });
        EasyMock.expect(httpServletRequest.getMethod()).andReturn("POST");
        EasyMock.replay(httpServletRequest);
        return httpServletRequest;
    }

    public static HttpServletResponse mockHttpServletResponse() {
        HttpServletResponse httpServletResponse =
        EasyMock.createMock(HttpServletResponse.class);
        EasyMock.replay(httpServletResponse);
        return httpServletResponse;
    }
}

```

ResultHook 的代码，用于记录中间结果

```

package org.tty.test2.foundation;

public class ResultHook {
    private static ResultHook gResultHook = null;

```

```

private ResultHook() {}

public Object getValue() {
    return value;
}

public void setValue(Object value) {
    this.value = value;
}

private Object value;

public static ResultHook getInstance() {
    if (gResultHook == null){
        gResultHook = new ResultHook();
    }
    return gResultHook;
}
}

```

LoginTestCase, 定义每次测试用例的尸体

```

package org.tty.test2.testCase;

import java.util.HashMap;

public class LoginTestCase {
    private HashMap<String, String> params;
    private String result;

    public HashMap<String, String> getParams() {
        return params;
    }

    public void setParams(HashMap<String, String> params) {
        this.params = params;
    }

    public String getResult() {
        return result;
    }

    public void setResult(String result) {
        this.result = result;
    }
}

```

```
@Override
public String toString() {
    return params + " " + result;
}
}
```

执行测试的代码

```
package org.tty.test2.test;

import org.junit.Test;
import org.junit.jupiter.api.Assertions;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.tty.test2.LoginServlet;
import org.tty.test2.foundation.MockFactory;
import org.tty.test2.foundation.ResultHook;
import org.tty.test2.foundation.TestCaseManager;
import org.tty.test2.testCase.LoginTestCase;

import javax.servlet.ServletException;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;

@RunWith(Parameterized.class)
public class LoginTest {

    private final LoginTestCase loginTestCase;

    public LoginTest(LoginTestCase loginTestCase) {
        this.loginTestCase = loginTestCase;
    }

    /**
     * 读取测试用例数据
     */
    @Parameterized.Parameters(name = "{index}:{0}")
    public static List<LoginTestCase> data() throws FileNotFoundException {
        return TestCaseManager.getInstance().loadTestCase("LoginTest",
LoginTestCase.class);
    }

    /**
```

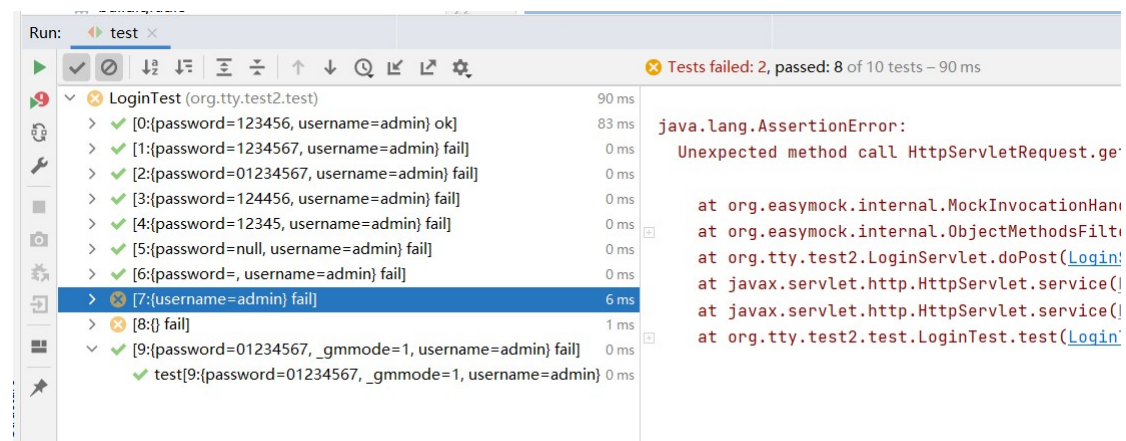
```

    * 执行一次测试
    */
@Test
public void test() throws ServletException, IOException {
    // 读取测试用例的参数
    HashMap<String, String> params = loginTestCase.getParams();
    LoginServlet loginServlet = new LoginServlet();
    loginServlet.init(MockFactory.mockServletConfig());
    try {
        loginServlet.service(MockFactory.mockHttpServletRequest(params),
MockFactory.mockHttpServletResponse());
    } catch (RuntimeException e) {
        // 在执行 service 时出现了错误
        ResultHook.getInstance().setValue("fail");
    } catch (Exception e) {
        ResultHook.getInstance().setValue("bug");
    }

    // 判断实际执行结果是否和预期输入一致。
    Assertions.assertEquals(loginTestCase.getResult(),
ResultHook.getInstance().getValue());
}
}

```

测试结果



Run: test x

Tests failed: 2, passed: 8 of 10 tests – 90 ms

LoginTest (org.tty.test2.test) 90 ms

- > [0: {password=123456, username=admin} ok] 83 ms
- > [1: {password=1234567, username=admin} fail] 0 ms
- > [2: {password=01234567, username=admin} fail] 0 ms
- > [3: {password=124456, username=admin} fail] 0 ms
- > [4: {password=12345, username=admin} fail] 0 ms
- > [5: {password=null, username=admin} fail] 0 ms
- > [6: {password=, username=admin} fail] 0 ms
- > [7: {username=admin} fail] 6 ms
- > [8: {} fail] 1 ms
- > [9: {password=01234567, _gmcode=1, username=admin} fail] 0 ms
- > test[9: {password=01234567, _gmcode=1, username=admin}] 0 ms

java.lang.AssertionError:
Unexpected method call HttpServletRequest.get...

at org.easymock.internal.MockInvocationHan
at org.easymock.internal.ObjectMethodsFilt
at org.tty.test2.LoginServlet.doPost(Login:
at javax.servlet.http.HttpServlet.service(!
at javax.servlet.http.HttpServlet.service(!
at org.tty.test2.test.LoginTest.test(Login:

4、总结与分析

在执行单元测试中，有时候待测的模块需要依赖与各种桩模块，如果此时桩模块并没有编写好，则需要到 EasyMock 来模拟桩模块的运行情况。

对于返回结果的函数 EasyMock 模拟起来相对比较简单。但是如果需要使用 EasyMock 来模拟 void 函数就会比较复杂。且当函数体对测试环境造成影响时将会更加难以测试。这也启示我们在设计模块时，需要考虑到模块的分离，这样可以对测试带来便利。

HttpServlet 是待测的模块，因此测试起来需要手动创建对象。而且需要查看源代码。通过查看代码发现 HttpServlet 依赖于 ServletConfig，ServletContext，RequestDispatcher，

HttpServletRequest 和 HttpServletResponse，这些都是需要模拟构造的。