软件测试 实验报告—SSHTest

1、实验目的

使用 EasyMock+Junit 对附件中基于 SSH 框架的注册登录小程序进行测试：分别对 UserService、RegisterAction 和 LoginAction 进行测试，测试类名分别为 UserServiceTest、RegisterActionTest 和 LoginActionTest。

2、测试范围

2.1 UserService 的 userRegister 和 loginVerify 方法

2.2 RegisterAction 的 validate 和 execute 方法

2.3 LoginAction 的 validate 和 execute 方法

3、测试过程

由于我们需要在不运行的情况下进行单元测试，因此数据库层就需要模拟，这里使用的是内存数据库模拟法。

3.1 编写 MockUserDao 及其代理类

类 InMemorySet，提供了内存的存储单元，作为 UserService 的实际存储结构

```
package org.tty.test3.dao;

import org.tty.test3.entity.User;

import java.util.HashMap;

public class InMemoryUserSet {
    HashMap<String, User> sets = new HashMap<>();

    public void addUser(User user) {
        sets.remove(user.getUsername());
        sets.put(user.getUsername(), user);
    }

    public boolean verifyUserName(String username) {
        return sets.keySet().stream().anyMatch(user -> user.equals(username));
    }
```

```
    public boolean verifyUserNameAndPassword(String username, String password) {
        return sets.values().stream().anyMatch(user ->
                user.getUsername().equals(username)
                        && user.getPassword().equals(password)
        );
    }
}
```

类 MockUserDao，作为模拟的 UserDao，并使用了 EasyMock 来创建模拟。

```
package org.tty.test3.dao;

import org.easymock.EasyMock;
import org.tty.test3.entity.User;

public class MockUserDao implements IUserDao {
    public void setInMemoryUserSet(InMemoryUserSet inMemoryUserSet) {
        this.inMemoryUserSet = inMemoryUserSet;
    }

    private InMemoryUserSet inMemoryUserSet;


    @Override
    public boolean addUser(User user) {
        if (!verifyUsername(user.getUsername())) {
            inMemoryUserSet.addUser(user);
            return true;
        }
        return false;
    }

    @Override
    public boolean verifyUsername(String username) {
        return inMemoryUserSet.verifyUserName(username);
    }

    @Override
    public boolean verifyPassword(String username, String password) {
        return inMemoryUserSet.verifyUserNameAndPassword(username, password);
    }


    /**
     * 获取代理类
```

```
     * @return 实际代理类
     */
    public static IUserDao getProxy() {
        MockUserDao realUserDao = new MockUserDao();
        realUserDao.setInMemoryUserSet(new InMemoryUserSet());
        return realUserDao;
    }

    public static IUserDao getMocked(String username, String password) {
        IUserDao realUserDao = getProxy();
        User wrappedUser = new User();
        wrappedUser.setUsername(username);
        wrappedUser.setPassword(password);

        IUserDao mockUserDao = EasyMock.createMock(IUserDao.class);
        EasyMock.expect(mockUserDao.addUser(wrappedUser))
                .andReturn(realUserDao.addUser(wrappedUser));
        EasyMock.expect(mockUserDao.verifyUsername(wrappedUser.getUsername()))
                .andReturn(realUserDao.verifyUsername(wrappedUser.getUsername()));
        EasyMock.expect(mockUserDao.verifyPassword(wrappedUser.getUsername(),
wrappedUser.getPassword()))
                .andReturn(realUserDao.verifyPassword(wrappedUser.getUsername(),
wrappedUser.getPassword()));
        EasyMock.replay(mockUserDao);

        return mockUserDao;
    }
}
```

类 MockFactory，用于快速获取模拟对象

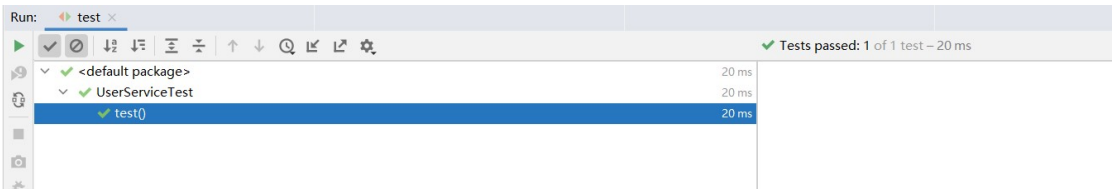
```
package org.tty.test3;

import org.tty.test3.action.LoginAction;
import org.tty.test3.action.RegisterAction;
import org.tty.test3.dao.MockUserDao;
import org.tty.test3.service.UserService;

import java.util.Map;

public class MockFactory {
    public static UserService mockService() {
        UserService userService = new UserService();
        userService.setUserDao(MockUserDao.getProxy());
        return userService;
    }
```

```
    public static LoginAction mockLoginAction(Map session) {
        LoginAction loginAction = new LoginAction();
        loginAction.setUserService(mockService());
        loginAction.setSession(session);
        return loginAction;
    }

    public static RegisterAction mockRegisterAction(Map session) {
        RegisterAction registerAction = new RegisterAction();
        registerAction.setUserService(mockService());
        return registerAction;
    }
}
```

3.2 编写 UserServiceTest 的代码

```
package org.tty.test3;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.tty.test3.service.UserService;


public class UserServiceTest {
    private UserService userService;

    @BeforeEach
    public void initialize() {
        userService = MockFactory.mockService();
    }

    @Test
    public void test() {
        String password = "password";
        String username = "test1";
        Assertions.assertFalse(userService.loginVerify(username, password));
        userService.userRegister(username, password);
        Assertions.assertTrue(userService.loginVerify(username, password));
    }
}
```

执行结果：



3.3 编写 RegisterActionTest 的代码

```
package org.tty.test3;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.tty.test3.action.RegisterAction;

public class RegisterActionTest {
    private RegisterAction registerAction;

    @BeforeEach
    public void initialize() {
        registerAction = MockFactory.mockRegisterAction();
    }

    /**
     * 测试用户名为空的情况
     */
    @Test()
    public void testUserNameEmpty() {
        String username = "";
        String password = "123456";
        registerAction.setUsername(username);
        registerAction.setPassword(password);
        registerAction.validate();

Assertions.assertTrue(registerAction.getActionErrors().stream().anyMatch((msg) ->
msg.equals("用户名不能为空！")));
    }

    /**
     * 测试密码为空的情况
     */
    @Test
    public void testPasswordEmpty() {
```

```
        String username = "test1";
        String password = "";
        registerAction.setUsername(username);
        registerAction.setPassword(password);
        registerAction.validate();

Assertions.assertTrue(registerAction.getActionErrors().stream().anyMatch((msg) ->
msg.equals("密码不能为空！")));
    }

    /**
     * 测试注册
     */
    @Test
    public void testRegister() {
        String username = "test1";
        String password = "123456";
        registerAction.setUsername(username);
        registerAction.setPassword(password);
        registerAction.validate();
        registerAction.execute();
        // 没有错误
        Assertions.assertFalse(registerAction.hasActionErrors());
        // 有对应的提示消息

Assertions.assertTrue(registerAction.getActionMessages().stream().anyMatch((msg) ->
msg.equals("注册成功！")));
    }

    /**
     * 测试用户重复的情况
     */
    @Test
    public void testRegisterRedundant() {

        String username = "test1";
        String password = "123456";
        registerAction.setUsername(username);
        registerAction.setPassword(password);
        registerAction.validate();
        registerAction.execute();
        registerAction.validate();
        registerAction.execute();
```

```
Assertions.assertTrue(registerAction.getActionErrors().stream().anyMatch((msg) ->
msg.equals("注册失败,该用户名已存在！")));
    }
}
```

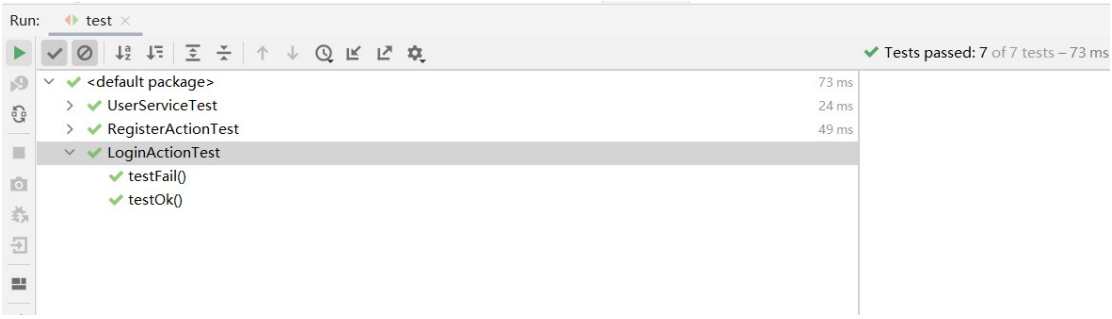
测试结果

3.4 编写 LoginActionTest 的代码

```
package org.tty.test3;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.tty.test3.action.LoginAction;
import org.tty.test3.entity.User;
import org.tty.test3.service.UserService;

import java.util.HashMap;
import java.util.Map;

public class LoginActionTest {
    Map<String, Object> session;
    LoginAction loginAction;
    UserService userService;

    @BeforeEach
    public void initialize() {
        session = new HashMap<>();
        userService = MockFactory.mockService();
        loginAction = new LoginAction();
        loginAction.setSession(session);
        loginAction.setUserService(userService);

        // 先存入一个用户
        userService.userRegister("test", "123456");
    }
```

```
    @Test
    public void testOk() {
        String username = "test";
        String password = "123456";

        loginAction.setUsername(username);
        loginAction.setPassword(password);

        String result = loginAction.execute();
        User user = (User) session.get("user");
        Assertions.assertEquals(username, user.getUsername());
        Assertions.assertEquals(password, user.getPassword());
        Assertions.assertEquals("success", result);
    }

    @Test
    public void testFail() {
        String username = "test";
        String password = "12345";

        loginAction.setUsername(username);
        loginAction.setPassword(password);

        String result = loginAction.execute();
        Assertions.assertEquals("input", result);
        Assertions.assertTrue(loginAction.getActionErrors().stream().anyMatch((msg)
-> msg.equals("账号或密码错误！")));
    }
}
```

测试结果



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。

在本次实验中，难点在于模拟数据库，本实现使用了模拟代理的方法来完成测试，使用了实现代理类的方式委托原有的类将其存储导向到内存数据库以完成测试的目的。