使用JMock来实现孤立测试

发表日期:2007-5-25
更新日期:2007-5-25
作者文章阅读次数:3901

源自：baggio785 (个人网站) 标签:

JMock是帮助创建mock对象的工具，它基于Java开发，在Java测试与开发环境中有不可比拟的优势，更重要的是，它大大简化了虚拟对象的使用。本文中，通过一个简单的测试用例来说明JMock如何帮助我们实现这种孤立测试。

JMock是帮助创建mock对象的工具，它基于Java开发，在Java测试与开发环境中有不可比拟的优势，更重要的是，它大大简化了虚拟对象的使用。本文中，通过一个简单的测试用例来说明JMock如何帮助我们实现这种孤立测试。

我们在测试某类时，由于它要与其他类发生联系，因此往往在测试此类的代码中也将与之联系的类也一起测试了。这种测试，将使被测试的类直接依赖于其他类，一旦其他类发生改变，被测试类也随之被迫改变。更重要的是，这些其他类可能尚未经过测试，因此必须先测试这些类，才能测试被测试类。这种情况下，测试驱动开发成为空谈。而如果其他类中也引用了被测试类，我们到底先测试哪一个类？因此，在测试中，如果我们能将被测试类孤立起来，使其完全不依赖于其他类的具体实现，这样，我们就能做到测试先行，先测试哪个类，就先实现哪个类，而不管与之联系的类是否已经实现。

虚拟对象(mock object)就是为此需要而诞生的。它通过JDK中的反射机制，在运行时动态地创建虚拟对象。在测试代码中，我们可以验证这些虚拟对象是否被正确地调用了，也可以在明确的情况下，让其返回特定的假想值。而一旦有了这些虚拟对象提供的服务，被测试类就可以将虚拟对象作为其他与之联系的真实对象的替身，从而轻松地搭建起一个很完美的测试环境。

JMock是帮助创建mock对象的工具，它基于Java开发，在Java测试与开发环境中有不可比拟的优势，更重要的是，它大大简化了虚拟对象的使用。

本文中，通过一个简单的测试用例来说明JMock如何帮助我们实现这种孤立测试。有三个主要的类，User，UserDAO，及UserService。本文中，我们只需测试UserService，准备虚拟UserDAO。对于User，由于本身仅是一个过于简单的POJO，可以不用测试。但如果你是一个完美主义者，也可以使用JMock的虚拟它。在这领域，JMock几乎无所不能。:)

User是一个POJO，用以在视图中传输数据及映射数据库。其代码如下:

```
package com.sarkuya.model;

public class User {
    private String name;

    public User() {
    }

    public User(String name) {
        this.name = name;
```

```
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

UserDAO负责与数据库打交道，通过数据库保存、获取User的信息。尽管我们可以不用知道JMock如何通过JDK的反射机制来实现孤立测试，但至少应知道，JDK的反射机制要求这些在运行时创建的动态类必须定义接口。在使用JMock的环境中，由于我们要虚拟UserDAO，意味着UserDAO必须定义接口。代码如下:

```
package com.sarkuya.dao;

import com.sarkuya.model.User;

public interface UserDAO {
    public void saveUser(User user);
    public User getUser(Long id);
}
```

UserService存有UserDAO的引用，通过其对外提供应用级的服务。相应地，我们先定义了其接口(尽管在本文中，作为被测试类，UserService不需要有接口，但如果以后此类需要被虚拟，也应该带有接口，基于此原因，我们也为其定义了接口)。

```
package com.sarkuya.service;

import com.sarkuya.dao.UserDAO;
import com.sarkuya.model.User;

public interface UserService {
    public void setUserDAO(UserDAO userDAO);

    public void saveUser(User user);
    public User getUser(Long id);
}
```

可以看到，除了setUserDAO()外，其另外的方法与UserDAO一样。这是设计模式中门面模式的典型应用，应用只通过UserService提供服务，而UserService在内部通过调用UserDAO来实现相应的功能。

根据测试先行的原则，你应该先写测试，再编写实现。这里先编写实现的原因，主要是使读者更加清楚我们接着要测试什么。由于本文是着重介绍JMock的使用，加上UserServiceImpl比较简单，因此先列出其代码如下：

```
package com.sarkuya.service.impl;

import com.sarkuya.dao.UserDAO;
import com.sarkuya.model.User;
import com.sarkuya.service.UserService;

public class UserServiceImpl implements UserService {
    private UserDAO userDAO;

    public UserServiceImpl() {
    }

    public void setUserDAO(UserDAO userDAO) {
        this.userDAO = userDAO;
    }

    public User getUser(Long id) {
        return userDAO.getUser(id);
    }

    public void saveUser(User user) {
        userDAO.saveUser(user);
    }
}
```

下面是UserService的测试代码：

```
package com.sarkuya.service;

import com.sarkuya.dao.UserDAO;
import com.sarkuya.model.User;
import com.sarkuya.service.impl.UserServiceImpl;
import junit.framework.*;
import org.jmock.Mock;
import org.jmock.MockObjectTestCase;

public class UserServiceTest extends MockObjectTestCase {
    private UserService userService = new UserServiceImpl();
```

```
    private Mock userDAO = null;

    public UserServiceTest(String testName) {
        super(testName);
    }

    protected void setUp() throws Exception {
        userDAO = new Mock(UserDAO.class);
        userService.setUserDAO((UserDAO)userDAO.proxy());
    }

    protected void tearDown() throws Exception {
    }

    public static Test suite() {
        TestSuite suite = new TestSuite(UserServiceTest.class);

        return suite;
    }

    public void testGetUser() {
        User fakeUser = new User("John");
        userDAO.expects(once()).method("getUser").with(eq(1L)).will(returnValue(fakeUser));

        User user = userService.getUser(1L);
        assertNotNull(user);
        assertEquals("John", user.getName());
    }

    public void testSaveUser() {
        User fakeUser = new User("John");

        userDAO.expects(once()).method("getUser").with(eq(1L)).will(returnValue(fakeUser));
        User user = userService.getUser(1L);
        assertEquals("John", user.getName());

        userDAO.expects(once()).method("saveUser").with(same(fakeUser));
        user.setName("Mike");
        userService.saveUser(user);

        userDAO.expects(once()).method("getUser").with(eq(1L)).will(returnValue(user));
        User modifiedUser = userService.getUser(1L);
        assertEquals("Mike", user.getName());
    }
}
```

此段代码有几点应注意:

1、此测试类继承了JMock的MockObjectTestCase

2、private Mock userDAO = null;说明userDao是一个准备虚拟的对象

3、在setup()中，将userDAO.class传入Mock()后，再通过proxy()方法返回一个UserDAO的代理类实例(即虚拟对象实例)，并赋值于userService

4、在testGetUser()方法中，如果我们先将第一行及第二行代码屏蔽掉，可以看出，这是一个真实环境下的测试代码。先获取一个User，然后确认其非空值，再确认其姓名为“John”。此时，在真实环境下，这段代码要测试成功的前提必须是UserDAO已经连接到了数据库，然后返回一个User后传给UserService。
但问题是，到目前为止，且不说UserDAO还未经历连接数据库这一系列繁琐而痛苦的过程，我们甚至还未实现UserDAO的接口！那么，为何加上第一行及第二行代码后就可以了呢？这正是JMock的威力所在。先实例化一个测试用的fakeUser，然后通过一系列的指令，在第二行代码中告诉JMock应该如何“做假”。尽管这句代码很长，我们可作如下理解：
1) userDAO.expects(once())：我们期望userDAO的某方法被执行一次，如果此方法未被执行，或者执行了二次以上，测试就不会通过
2) method("getUser")：这个期望被执行一次的方法名为userDAO.getUser()
3) with(eq(1L))：执行getUser()方法时，确认其传入的参数值为“1L”
4) will(returnValue(fakeUser))：上述条件均满足后，返回一个虚假的对象，即我们前面实例化的fakeUser
总体来说，当设定好第二行语句后，JMock就在后台监控着，确保userDAO.getUser()必须，且只被执行一次，且参数“1L”已经正确地传给了此方法，一旦这些条件被满足，就返回fakeUser。
而在第三行，User user = userService.getUser(1L)将触发所有这些条件，作为奖励，它接受了奖品fakeUser并赋值于user对象。而下面第四行及第五行均对此user对象进行测试，不通过才怪。

5) testSaveUser()方法中的原理类似。其思路是，将id为“1”的user从数据库中取出，将其名改为“Mike”，再存回数据库，然后再从数据库中取出此user，确保其名字已被改变。
第五行userDAO.expects(once()).method("saveUser").with(same(fakeUser))比较特殊。首先，with(same(fakeUser))说明，传入参数必须是fakeUser此实例，尽管我们在下面的语句中通过user.setName("Mike")，但只是改变了其name的属性，而fakeUser的实例引用并未发生改变，因此可以满足条件。其次，其后没有.will(returnValue(fakeUser))，因为userDAO.saveUser()不需要返回任何对象或基本数据类型。
另外，当再次执行userDAO.expects()时，JMock将重设其监控条件。我们也可以通过userDAO.reset()来显式是清除监控条件。

通过以上实例代码及其说明，我们看出，用好JMock的关键是先设置监控条件，再写相应的测试语句。一旦设好监控条件后，在某段代码块执行完毕时，如果监控条件未得到满足，或是没有通过expects()再次重设条件，或通过reset()来显式是清除监控条件，测试将无法通过。

以上介绍了JMock的基本使用方法。而这种基本用法，占了全面掌握JMock所需学习的知识70%以上。关于JMock的更多细节，感兴趣的读者可以访问JMock的网站进一步学习。

不断学习

跨越边界：活动记录和 Java 编程中特定于域的语言
GOOGLE服务列表
让你的JSP支持Ubbcode
不同配置文件的初始化Hibernate心得
Struts的巨大烦恼 真的不适合大系统？
Struts配置文件详解
详细介绍在tomcat中配置数据源以及数据源的原理
使用 DHTML 与 XML 制作 Ajax 幻灯片
利用反射机制实现XML-RPC
面向方面的编程：它的好处是什么？

评论

wayttcjl 2008年03月21日 222.65.4.*

%u6B63%u5728%u770B%u8FD9%u4E2A%uFF0C%u70B9%u62E8%u4E86%u6211%u4E00%u4E0B%uFF0C%u8C22%u8C22%u4E86%uFF01

更多评论

标 题： Re：使用JMock来实现孤立测试
姓 名： huapuyu8
校验码： R0826V
内 容：
发表评论