天天学习 好好向上 rainbow...

Everything is possible

日志

测试相关 WEB类项目相关环境的安装与部署

JMock单元测试

2011-06-09 21:47:28 | 分类： 软测

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 {
```

关于我

Rainbow

加博友 关注她

```
    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：

```
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);
```

```
}
```

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) {
```

```
    }

    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 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

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。

例如，有 这样两个类，一个是Dao.java，用于数据库访问成操作的，一个是Business.java，需要调用Dao进行业务处理。Dao.java是已经 在有数据库的环境测试通过的，现在需要测试Business.java。通常情况下，我们需要在测试环境配

现在如果通过JMock就可以不配置数据库环境，也可以完成测试。

JMock网站链接：http://www.jmock.org

```
//Dao.java
package com.raistlin.test.jmock
public class Dao
{
    public Dao()
     {
     }
    public List execute(String sql)
    {
    //数据库操作...
    }
}

//Business.java
package com.raistlin.test.jmock
public class Business
{
    private Dao dao ;
    public void setDao(Dao dao)
    {
        this.dao = dao;
    }
```

```
    {
        List list = dao.execute("select * from tbl_test");
        return (String) list.get(0);
    }
}

//BusinessTest.java
package com.raistlin.test.jmock
import org.jmock.cglib.MockObjectTestCase;
import org.jmock.cglib.Mock;

public class BusinessTest extends MockObjectTestCase
{
  private Mock mockDao = null;
  private Business business = null;

  protected void setUp() throws Exception
  {
      super.setUp();
      mockDao = new Mock(Dao.class);
      business = new Business();
  }

  protected void tearDown() throws Exception
```

```
            mockDao = null;
            business = null;
            super.tearDown();
        }

        public void testOperate()
        {
            ArrayList stubList = new ArrayList();
            stubList.add("test");
            // 定义Mock剧本
            mockDao.expects(once())// 期待次数
                .method("execute")// 调用方法
                .with(eq("select * from tbl_test"))// 传入参数等于
                .will(this.returnValue(stubList));// 返回对象
            Dao dao = (Dao) mockDao.proxy();
business.setDao(dao);
            assertEquals(business.operate(), "test");
        }
    }
```

在工程中导入jmock-1.0.1.jar ，jmock-cglib-1.0.1.jar ，cglib-full-2.0.jar 三个jar包，然后在JUnit中执行BusinessTest.java，会显示绿棒，这样，在没有数据库的情况下，单元测试成功。

使用JMock模拟类的条件：

1、要有无参数的构造器；

这样对于private的构造器或单例模式的类应该是无法模拟了。这样就要求我们养成针对接口编程的习惯，对接口用JMock进行模拟将不受这些限制。只需要将

import org.jmock.cglib.MockObjectTestCase;

import org.jmock.cglib.Mock;

改为

import org.jmock.MockObjectTestCase;

import org.jmock.Mock;

即可。

对于Mock类，有很多API可供使用，比如：

isA(Class clz) 表示为某个类

isVoid() 将will()换成isVoid()表示无返回值

详细API可以参考一下DOC，还会找到很多有用的东西。