面向 MVC + +的测试驱动开发研究

Research on Test Driven Development Facing MVC + + Architecture

黎 利　刘振宇　（南华大学计算机科学与技术学院 湖南 衡阳 421001）

摘要：测试驱动开发是以测试作为软件开发过程中心的编程技术。由于其有助于提高产品代码质量，近几年来受到软件开发人员的推崇。然而，测试驱动开发在许多系统中应用还存在一定的难度，比如具有图形用户界面和多层架构的系统。本文提出了在 MVC + +架构下进行测试驱动开发的过程模型和测试用例的设计策略，最后介绍了该模型在具体项目中的应用。

关键词：MVC + +　测试驱动开发　测试用例

1　前言

随着对软件质量要求的提高，软件测试技术越来越受到重视，测试驱动开发则是近几年来受到软件开发人员推崇的技术之一。它是编程时使用的技术，要求在编写任何产品代码之前，首先编写用于定义产品代码行为的测试。采用测试驱动开发，我们将得到简单、清晰、高质量的代码。但是，测试驱动开发在许多系统中应用还存在一定的难度，比如具有图形用户界面和多层架构的系统。考虑到这类系统的特殊需求，测试驱动开发也需要遵循系统相应的设计模式的特点来进行开发。本文将介绍在 MVC + +架构下如何进行测试驱动开发和测试用例的设计。

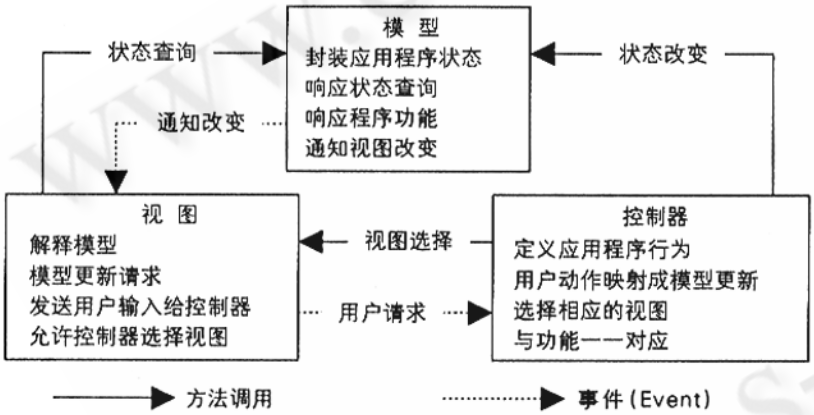

图 1　MVC 模式各部分的关系和功能

2　MVC + +架构

20 世纪 70 年代，MVC 架构在 Smalltalk80 的 GUI 设计中被提出[2]。MVC 架构把数据处理、程序输入输出控制以及数据表示分离开来，并且描述了不同部分的对象之间的通信方式，使它们不必卷入彼此的数据模型和方法中。由于 MVC 架构结构清晰，广泛应用于软件开发过程中，提高了软件的可复用性和可维护性，并且提高了软件开发的整体效率。

MVC 设计模式包含三个组件：Model（模型）– View（视图）– Controller（控制器）。（如图 1）

随着 MVC 的广泛应用，出现了很多可用的变种，如 MVC + +，HMVC（Hierarchical MVC），MVC Model 2，MVC Push，and MVC Pull。它们每一个都有不同之处。MVC + +由 Ari Jaaksi 提出，并成功应用在 Nokia 通信公司开发的网络管理系统中[3]。它与 MVC 有同样的三层，但在 MVC + +架构中 View 层与 Model 层没有直接的联系，由中间层 Controller 来处理两者之间的交互（如图 2）。

View 实现了应用程序的所有对话框，接收用户的操作并将表示模型数据、逻辑关系和状态的信息及特定形式展示给用户。但是它并不处理用户的操作，只是将这些操作信息传递给 Controller。

Controller 处理 View 与 Model 的交互操作，其职责

是控制提供 Model 中任何变化的传播，确保用户界面与模型间的对应联系；它接受用户的输入，将输入反馈给模型，进而实现对模型的计算控制，使模型和视图协调工作。

Model 将问题领域中的对象抽象为应用程序的对象，在这些抽象的对象中封装了对象的属性和这些对象所隐含的逻辑。它不需要知道用户接口和应用程序的具体逻辑，可以独立执行相关工作。它接受 Controller 的数据查询请求并返回查询结果。

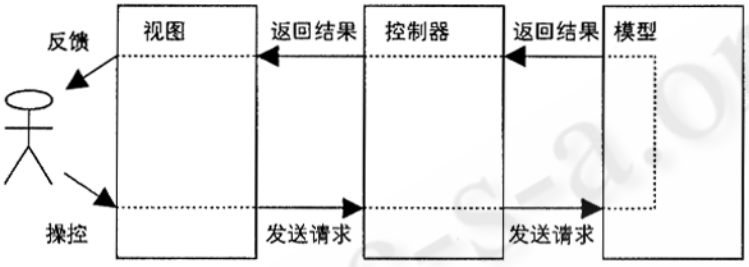

图 2　MVC + + 模式三个组件之间的通信

3　测试驱动开发(Test Driven Development)

测试驱动开发是极限编程(Extreme programming)的最佳实践之一，也是在极限编程中处于核心地位的技术。即便选用的开发过程不是 XP，但采用测试驱动开发也能让你从中获益。它的基本思想是在开发功能代码之前，先编写测试代码[1]。也就是说在明确要开发某个功能后，首先思考如何对这个功能进行测试，并完成测试代码的编写，然后编写相关的代码满足这些测试用例。如此继续添加其他功能，直到全部的功能开发完成(如图 3)。

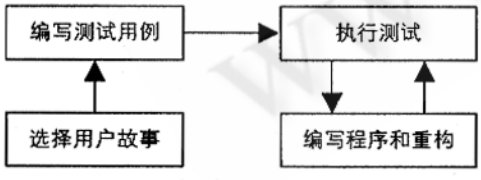

图 3　测试驱动开发模型

测试驱动开发通过编写测试用例，先考虑代码的使用需求(包括功能、过程、接口等)，通过编写这部分代码的测试用例，对程序功能的分解、使用过程、接口等都进行了设计，这种从使用角度对代码的设计通常更符合后期开发的需求和对程序可测试性的要求，对代码的内聚性的提高和复用都非常有益。测试驱动开发还要求测试可以完全自动化地运行，其结果就是可以拥有一套伴随产品代码的详尽的自动化的测试集[1]。将来无论出于什么原因需要对产品代码进行维护时，都在这套测试集的驱动下工作，会给我们的工作带来极大地便利。

4　基于 MVC 架构的测试驱动开发过程

根据对 MVC + + 架构各层的特点的分析，三层中 Controller 反映了应用程序的功能和流程，并且清楚 Model 和 View 的功能，所以测试驱动开发应该从 Controller 出发，首先将开发的重点放在实现程序的功能上，更早地实现需求。由于在 Controller 的开发过程中需要不断地对 Model 和 View 进行重构，为了减少重构的代价，可以将 Model 中未实现的对象用 Mock Object 来代替。实现一个用户故事的 Controller 后，将 View 和 Model 的接口提取出来，接口描述了与 Controller 通信的参数和方法，Mock Object 和应用程序对象都继承这些接口。图 4 简单描述了它们之间的关系。

接下来实现 Model 层。由于 Model 对象是根据问题域中的对象构建的，所以 Model 层的每个接口类描述了对象的属性和方法，Model 只负责实现接口中的方法，而可以不用考虑程序的控制逻辑。这些对象之间还存在着这样或那样的关系，如一般特殊关系、整体部分关系、实例关联、消息关联等，所以应当优先实现独立性强的对象，即应首先选择依赖程度低的用户故事来实现。

最后实现 View 层。View 接口描述了与 Controller 之间数据传递的信息，View 层的实现需要考虑以怎样的视图来显示这些信息，以及设计怎样的事件来触发 Controller 中相应的方法。需要注意的是视图应当保持统一的风格。

5　测试用例的设计策略

通过以上对 MVC + + 架构的测试驱动开发过程的分析，我们可以将测试活动分为三个不同的场景来进行：

View 测试：由于 View 层只负责显示图形用户界面，不涉及任何的功能代码，所以 View 层不适合作测试驱动开发，而是将 View 的测试集中在用户界面风格上，比如用户界面的一致性、界面的布局、用户界面之间的切换是否顺畅、颜色的使用是否适当、以及界面的设计是否考虑不同用户的需求等等。在 Web 界面测试方面，配合使用 ASPUnit 和 HttpUnit 等自动化测试工具可以提高测试效率。

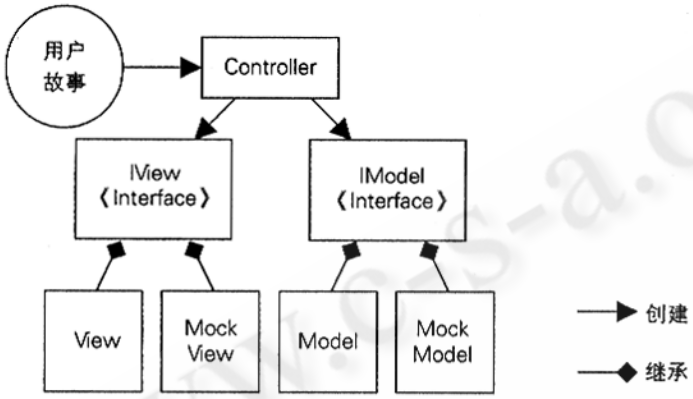

图 4　MVC 架构、Mock、object、接口关系图

Controller 测试：由于 Controller 层涉及程序的控制流和功能，所以 Controller 测试可以完全根据测试驱动开发的过程来进行。根据 Controller 层的特点，该层测试用例应当关注程序的控制流程是否正确地实现。首先依据场景法分析用户故事的基本流和备选流，并建立正面和负面的测试框架，再根据等价类划分、边界值分析等单元测试方法对每个测试框架设计测试数据。测试程序可以在测试框架下编写，如. Net 的 NUnit 框架、Java 的 JUnit 框架等。

Model 测试：Model 是由许多模型对象及这些对象之间各种各样的关系构成，Model 的测试集中在每个对象的方法是否返回正确的结果，所以可以采用传统的方法进行。然而，在绝大部分系统中，这些对象的方法主要是对数据库的操作，包括增加、删除、查询、修改以及数据库的连接等。通过引用 Controller 层的测试数据，还可以预先编写生成这些操作测试用例的程序，加速 Model 的测试。

6　测试驱动开发实践

下面通过一个论坛系统中用户注册功能的测试驱动开发过程，说明论文提出的方法应用。该系统采用 ASP. Net 开发。

6.1　设计测试用例

通过对用户故事“用户注册”的分析，注册时应该输入用户名和密码，以及重复密码输入。设计一个正面的测试用例，即设计合法的输入返回注册成功的信息。下面是测试程序：

```
[Test]
// 正面的测试用例，输入合法的参数能返回注册成功的信息
public void CanPassTest( )
{
  Register Dialog = new Register ( );
    Dialog. Password = " 123567" ;
  Dialog. RePassword = " 123567" ;
  Dialog. Username = " xiaofang" ;
  Assert. IsTrue( Dialog. SystemRegister( ) ) ;
}
```

6.2　编写通过测试用例的程序代码

显然，上面的测试程序是不能通过编译的，因为 Register 类还未编写。

```
public class Register
  {
   // Register 类属性
   ……
  // Register 类方法
    public bool SystemRegister( )
    {
     //调用 Model 层的方法
      return UserInsert( username, password) ;
    }
```

由于用户注册的行为需要将用户的姓名和密码存入数据库中，所以 UserInsert 方法应该交由 Model 层来处理。然而，此时设计数据库是相当不适合的，我们用 Mock object 来代替。

6.3　设计 Mock object

```
  public class MockUser
    {
```

```
public bool UserInsert(string username, string password)
{
  //模拟数据插入数据库成功返回
  return true;
}
```

此时在 Register 类中创建 MockUser 类的对象，并调用 UserInsert 的方法就可以通过 CanPassTest( ) 的测试了。

6.4 回到第一步继续设计测试用例完成 Register 的编写，并提取 Model 和 View 的接口

```
// 设计用户故事"用户注册"Controller 层代码
public class Register
{
  private IUserModel m_user;
   private IRegisterView v_userview;
  public Register(IUserModel user IRegisterView userview;)
  {
    this.m_user = user;
     this.v_userview = userview;
  }
  public bool SystemRegister( )
  {
    //判断用户名是否已经注册过
     if (! (m_user.CheckUsername(m_user.username)))
     {
       v_userview.errmeg = "用户名已经存在!";
       return false;
     }
    //判断用户两次输入的密码是否相同
    else if (m_user.password! = m_user.repassword)
     {
       v_userview.errmeg = "输入的两个密码不相同!";
       return false;
     }
    //用户输入的密码长度不能超过六位
    else if (m_user.password.Length <6)
     {
       v_userview.errmeg = "密码的个数大于六位";
       return false;
     }
    //输入有效性验证完成，可插入进数据库
    else
     {
       return m_user.UserInsert(m_user.username, m_user.password);
     }
  }
 //用户故事"用户注册" Model 接口
public interface IUserModel
 {
   string Username{get;set;}
   string Password {get;set;}
  bool CheckUsername(string username);
  bool UserInsert(string username, string password);
 }
 //用户故事"用户注册" View 接口
public interface IRegisterView
 {
   string Username {get;set;}
   string Password { get; set;}
   string RePassword{get;set;}
   string ErrMeg {get;set;}
}
```

接下来，就可以根据提取出来的接口实现 Model 和 View

7 总结和展望

本文提出了基于 MVC 架构的测试驱动开发过程和测试用例设计策略，将该方法应用在具有图形用户界面和多层架构的系统中，可以提高系统的开发效率和质量。

（下转第 50 页）

（上接第 46 页）

参考文献

1 Kent Beck. Extreme programming explained: Embrace change [M]. Addison – Weslley, 1999.

2 John Deacon. Model – View – Controller (MVC) Architecture [EB/OL]. http://www.jdl.co.uk/bridfings/index.html#mvc, 2000.

3 Ari Jaaksi. Implementing Interactive Applications in C++ [J]. Software Practice & Experience, 1995, 25 (3), 271 – 289.

4 Jens Uwe Pipka. Test – Driven Web Application Development in Java [EB/OL]. http://www.old.netobjectdays.org/pdf/02/papers/node/0389.pdf, 2002.

5 Stewart Baird, 限编程基础、案例与实施[M], 袁国忠译, 北京:人民邮电出版社, 2003.