设计模式在电子商务交易网站中的应用

Design Patterns Using in Establishing Electroic Business Sites

林碧英　曲俊华　（华北电力大学（北京）计算机科学与技术系 102206）

摘要：近年来，设计模式已经成为面向对象软件开发领域中的一个热门话题。本文从现实出发引入设计模式的概念和一些有关模式的定义，分析比较了模式和框架的异同。最后用实例说明了如何应用设计模式，使软件具备良好的可靠性、可扩展性、可复用性和可维护性。

关键词：设计模式　框架　Observer 模式　Iterator 模式　Proxy 模式

1　引言

随着面向对象的系统开发数量的增加，我们会注意到有越来越多相似的结构出现。这并不令人惊奇：软件解决的是确定领域的问题，当两个不同的系统要解决类似的问题时就会有相似的内部结构。对这些功能相似的、通用的结构模块进行抽象总结而成的就是设计模式。简言之，一个设计模式就是对象（类）的集合，它具有解决某一类特定问题的功能，而且是解决这类问题的最优方案。应用设计模式可以使软件具备良好的可靠性、可扩展性、可复用性和可维护性。

设计模式是软件工程师设计开发软件经验的总结，是一种专家知识。

下面是有关设计模式的不同定义：

（1）每个模式都是由三部分组成的规则。它们表达了特定环境、问题和解决方案之间的一种关系（C. Alexander）。

（2）一个模式是由三部分组成的规则。这个规则描述特定环境、特定系统作用力以及特定软件配置之间的关系，其中特定系统作用力可以在特定环境下反复出现，并且特定软件配置可以是特定系统有能力解决自身存在的问题。（R. Gabriel）

随着互联网的普及，人们已经越来越习惯享受它所提供的巨大便利，足不出户，只靠鼠标的轻轻点击就可在网上实现轻松购物，这几乎已经成为我们所熟知的广告词。2003 年初，我们在基于 Java 及 Jsp 技术的电子商务网站的开发中，用到了一些成熟的设计模式，如观察者模式、遍历器模式和代理模式等。正是由于采用了这种基于设计模式的对象建模方法，提高了软件的通用性，并为将来的代码复用打好了基础。

2　框架和模式的比较

框架是一套详细而精确的类，类与类之间功能各异，彼此调用，相辅相成，形成了对某一类重现问题的可重用、易扩展的解决方案。这里需要强调的一点是框架经常是相对于某一特定平台来讨论的。MVC 在 SmallTalk 中的实现是一个框架，CORBA 是一个框架，JavaBean 也是一个框架。在实际的情况中，我们可以通过派生框架类的子类和组织框架类的实例为一特定的应用程序定制框架。

框架是特定于某一应用领域的，它依据不同的平台呈现出不同的表现形式，有时两个貌似不同的框架也许完全出于同一种设计模式，模式相对于框架就更加抽象，可以用到任何一种应用当中。框架可以由程序设计语言编写，不仅可以被学习，还可以直接执行和复用。对于模式只有它的实例可以用代码体现。框架包含模式，一个框架可以看成是一个或多个设计模式解决方案的物理实现，而模式用来指导如何实现这个方案。框架的存在形态是特定于某一环境、某类模式的可执行软件，就像我们熟知的 Apache 公司的 Struts，就是一个有名的实现 MVC 模式的框架。而设计模式是软件开发经验的总结，它是一个概念，是一个逻辑实体。

3　模式的应用

3.1　Observer（观察者）模式的应用

Observer 模式是设计模式的一个代表之作。它的目的是定义对象间的一种一对多的关系。即当一个对象改变时，所有依赖于它的对象都会接到通知并且自动更新。

网上商店与传统的面对面交易有所区别，它强调要充分利用互联网的便利性。为使商品交易更具人性化，给用户购物提供最大的便利性，每当有新品上架时都要通知给会员，这也是会员所享有的一种服务。相应的代码如下：

```
public class productDB extends Observable{
    private String name = "";
    public Insert( ){
    ……;//形成新品插入数据库的 sql 语句执行
```

数据库的插入操作

```
    setChanged( ) ;//设置变化点
    notifyObservers( name) ;//通知所有的观察者
    } }
public class NewObserver implements Observer{
    private String name = null;
    public void update( Observable obj,Object arg) {
    if ( arg instanceof String) {
    ……;//此间执行相应的当添加新品后的操作,如通知各会员
      } } }
```

在 Jsp 页面中实现代码时,用一个 form 表单记录管理员输入的新产品的相关信息,如品名、价格、简介等。在处理表单提交信息的页面中调用 productDB 的 addObserver(NewObserver) 方法加入观察者 NewObserver。这样当维护人员用 submit 按钮提交信息的时候观察者就开始起作用了,只要新品信息被插入数据库,观察者就会被触发执行 update 方法中的代码。此时,会员就会知道有哪些新产品上架。

3.2 Iterator 模式的应用

数据值以一种复合结构组织时,经常需要遍历数据结构去处理每一个数值。Iterator 模式提供了统一的接口操作来实现对一个数据结构的遍历,使得当数据结构的内部算法发生变化时,客户代码不需要任何的改变,只需要改变相应的 Iterator 实现,就可以无缝的集成在原来的程序中。在 Java 中,Iterator 模式已经被集成到接口 Collection 中,在使用时,只要将数据装入 Collection 中,直接使用 Iterator 遍历即可。

既然是网上商店,就有可能向用户展示产品详细信息;同时当用户有查询商品的请求时,要向用户展示查询结果的列表;用户要查看购物车的信息,就要向用户展示购物车中存放的所有内容。这些都涉及不同的数据结构,所以程序中多次采用 Iterator 模式来遍历数据结构,取得用户所要求的信息。例如:

```
Collection c = ……;//将所有数据装入到 Collection 中
Iterator it = c. iterator( ) ;//得到本 Collection 的一个遍历器
while( it. hasNext( ) ) {
    NewsDetail nd = ( NewsDetail) it. next( ) ;//取得 Collection 中的一个数并进行类型转换
System. out. print( nd. getTitle( ) + DateFormat. change( nd. getDate( ) )
    + nd. getTime( ) + " \n" ) ;//以后台输出为例,代表对取出的数据可以进行操作}
    }
```

3.3 Proxy 模式的应用

Proxy 在设计模式中的定义是为其他对象提供一种代理以控制对这个对象的访问。它包含一个客户端对象以及与之进行交互的一个媒介对象。显然媒介对象必须为客户端对象提供一些服务,在实际的情况当中,媒介对象一般只充当客户端与服务器端的接口:他收集客户端的请求,然后把他送到合适的服务器去处理。它隐藏了网络传输的细节,使得事务处理看起来更简单。在这个模式的有些实例当中,媒介对象可以对客户请求信息中的数据进行控制,如改变它们的格式或结构等。同样,在服务器端的信息返回到客户端之前,媒介对象也可以对它进行过滤。

网上商店综合了前台处理和后台维护系统,故而要对用户的权限做出严格的控制。以订单处理为例:只有负责订单的管理员可以进入订单的后台管理系统,也只有他可以确定订单是否有效,对有效订单进行处理并设置订单处理状态,将无效订单从系统删除。下面是它的代码实现:

```
public class Permissions {
……;//一些其他有关权限设置的实例变量
private static Boolean Admin = false;//用一个布尔变量标识是否为系统管理员
public setAdmin( Boolean Admin) { this. Admin = Admin; }//为这个 boolean 赋值
public Boolean isAdmin( ) { return Admin; }
……; }
public class PopedomProxy{
private Permissions permission;
public PopedomProxy( permissions) {
}
……
public void OrderProcess( String state) {
    //只有是系统管理者才可以查看新生成的订单和修改数据库中的订单状态
    if ( permissions. isAdmin( ) ) {
        OrderDB. OrderProcess( state) ;//真正的对订单的操作发生在
        } //类 OrderDB 的 OrderProcess 方法中
    }
……}
```

当用户有查看订单的要求时,总要先经过 PopedomProxy 这个代理类的 OrderProcess 方法来判断用户是否有此权限的管理员,然后通过代理调用真正与数据库发生交互的类 OrderDB 中的 OrderProcess 方法,来达到修改订单状态的目的。

(下转第 46 页)

(上接第 42 页)

4 结束语

当我们面对的应用越来越庞大,要处理的客户端越来越复杂时,设计模式就成为我们需要甚加考虑的重点,在这种情况下也更能体现模式的优越性。一个成功的模式不仅能够保证软件良好的结构,同时也为未来的软件维护和代码重用打下了良好的基础,将人们从简单的重复劳动中解放出来,这也是我们研究模式的主要目的。

参考文献

1 Deepak Alur, John Crupi, Dan Malks, J2EE 核心模式,机械工业出版社,2002。

2 Gamma E. 设计模式可复用面向对象软件的基础,机械工业出版社,2000。

3 Design Patterns. http://www.cs.unc.edu .