文章编号:1001 - 9081(2005)03 - 0620 - 03

利用产生式编程构建低耦合的软件模块——AOP 的原理和实践

张　强[1],谭　博[2],谭成翔[1]

(1. 同济大学 计算机科学与工程系,上海 200092; 2. 复旦大学 软件学院,上海 200433)

(yuqzhang2002@hotmail.com)

摘　要:分析了面向对象理论遇到的难以解决的问题。针对此类问题提出了利用产生式编程构建通用领域模型和低耦合的模块的思想。以 Aspect Oriented Programming (AOP)为例,列举了其主要实现手段,分析了它们的利弊,对比了传统 OO 方法的 Observer 模式实现和利用 AOP 的 Observer 模式实现。

关键词:产生式编程;AOP;复用;设计模式;横切关注点

中图分类号: TP311.5　**文献标识码**:A

Implementing low-coupling module with generative programming methods: the theory and practice of AOP

ZHANG Qiang[1], TAN Bo[2], TAN Cheng-xiang[1]

(1. *Department of Computer Science and Engineering, Tongji University, Shanghai* 200092, *China*;
2. *School of Software Engineering, Fudan University, Shanghai* 200433, *China*)

Abstract: With the development of programming technology and theory, some problems in practice, which can not be solved by the traditional OO theory, are attracting more and more interests of researchers. A generative programming-based approach was proposed to solve such kind of problems by constructing domain-neutral models and low-coupling modules. One implementation of our proposed approach, AOP, was analyzed to demonstrate the advantages and shortcomings of our approach. Finally, a comparison between AOP-based and OO-based Observer models was conducted to show the superiority of our approach over traditional OO approaches.

Key words: generative programming; AOP; reuse mechanism; design pattern; crosscut

关注点,是一个为了满足系统整体目标必须被处理的特定需求或考虑。一个软件系统是一组关注点的实现。例如一个银行系统是以下这些关注点的实现:客户与账户管理、利息计算、银行间事务、自动柜员机事务、账单生成等。我认为编程技术的本质就是分离关注点。

1　面向对象技术及其遇到的难题与原因

编程技术虽然在不断的发展,但始终都在关注这样一件事情就是分离关注点。在一个模块里面关注一个事情,然后把这一个事情做好。所有的模块组合在一起就变成了系统的功能。从结构化到基于对象,再到面向对象都是如此,只不过抽象的层次越来越高。

1.1　面向对象技术的本质

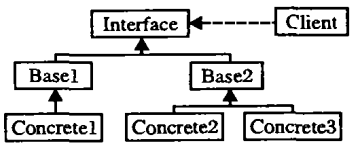

图 1　面向对象的基本图示

面向对象技术是现在主流的编程技术,图 1 就是面向对象技术一个最基本的表示。应面向接口编成,不要面向类编程。

我们定义一个接口,接口里面声明几个方法。这就是我们这个类体系所要做的事情,然后我们把这个接口暴露给用户。用户通过这个接口的时候,这个接口可能有 N 种实现的方式,每一种方式提供一种具体实现的办法。但用户并不关心这点。他关心的是调用这个接口是否可以解决他的问题。这就是面向对象的本质。

面向对象的实质是一个对解空间所建的模型。每一个对象体系实际上在解决一个问题。整个对象体系就描述了一个对问题的解决方案。

1.2　面向对象理论遇到的难题与产生根源

在实际的应用中,面向对象技术常常遇到一些解决不了或者很难解决的问题。

比如契约保证,以栈为例:你如果做 push 操作,那么这个栈一定不是满的,如果这个栈是满的,那么这个 push 操作是不合法的;如果做 pop 操作,这个栈不能是空的。这就是栈这个对象它的契约。那我们怎么用一组统一的方式来保证它的契约,在运行一个方法以前,要检查它的前置条件,方法运行完以后,要检查它的后置条件。整个系统里面有很多的对象,每一个对象的契约都要去检查,怎么检查到。如何用一种全局统一的方式解决这样的问题?

与此相似的还有错误处理、安全性问题、并发问题、事物控制等。在面向对象方法中我们很难为上面的这类问题提供全局的解决方法。如果用关注点的眼光看,上面提到的是解决分布在多个模块中的全系统范围特性,我们称之为:横切关注点。

下面我们分析一下它产生的根源:

图 2 把需求比作一束穿过三棱镜的光。我们让需求之光

收稿日期:2004 - 08 - 29;**修订日期**:2004 - 11 - 16

作者简介:张强(1980 -),男,河南南阳人,硕士研究生,主要研究方向:计算机网络技术、信息管理; 谭博(1979 -),男,湖北人,硕士研究生,主要研究方向:软件工程; 谭成翔(1965 -),男,湖北人,研究员,博士生导师,主要研究方向:计算机网络安全.

通过鉴别关注点的三棱镜,就会区别出每个关注点。

关注点可以被分为两种类型:核心关注点,捕捉模块的中心功能;以及横切关注点,捕捉横跨多个模块的系统级外围需求。对于横切关注点,当前的技术倾向于使用一维的方法学来处理这种需求,把对应需求的实现强行限制在一维的空间里。这个一维空间就是核心模块级实现,其他需求的实现被嵌入在这个占统治地位的空间,换句话说,需求空间是一个 n 维空间,而实现空间是一维空间,这种不匹配导致了糟糕的需求到实现的映射。其结果是,典型的 OOP 实现所制造的核心关注点和横切关注点之间的耦合是很不理想的,因为如果要增加新的横切特性,或者修改现存的横切功能,都需要对相关的所有核心模块进行修改。

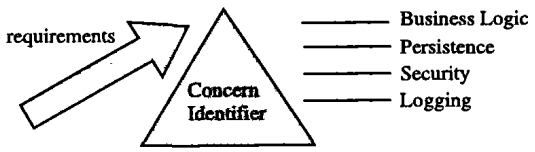

图 2　关注点分解:三棱镜法则

2　产生式思想的解决方法

现在有一种新的观点,叫做产生式编程,主要观点如图 3 所示。

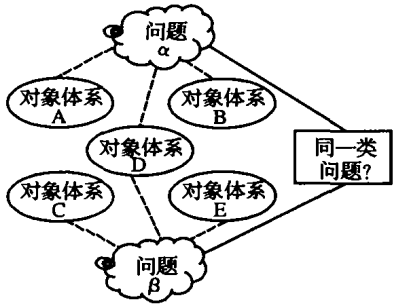

图 3　系统中的问题

问题 α 涉及 3 个对象体系,问题 β 又涉及另外的 3 个对象体系。以事务为例,问题 α 是说用户管理的时候它的事物保证,问题 β 可能是说在做文件审批的时候它的事务保证。显然这两个问题涉及到的对象体系是不一样的,但是它们会不会是同一类问题?我们有没有办法解决这种可能是同一类但是涉及不同的对象体系的问题,。这就是一个问题空间的视角,也就是横切关注点,我们不是从对象的角度来看,我们看到的是问题。

这是在整个企业应用里面随处出现的。产生式编程解决的问题就是如何进行领域建模。领域建模,是一个业务应用里面的问题,不是软件技术。技术可能是解决一个算法,领域建模描述的就是企业应用里面全局的问题。如果我们可以用一种方式,把事务管理、持久化、安全认证用领域建模的方式,把它建成一个可以重用的模型。然后在不同的应用里面为它配上不同的解决方案。对象模型就是一个解决方案,如果我们可以把一个领域模型抽取出来做成一个通用的模块,在不同的应用里面配上不同的解决方案,就可以定制出适用于不同场合的软件系统。也就是说要先有一个领域的概念,然后针对客户的具体需求,把解决方案接插上去,就可以生产出可用的一个系统。这就是所谓的软件生产线。

实现产生式编程的技术,现在主要有以下几种:1)泛型。2)基于模板的组件技术。这两种主要用在 C 和 C++里面,用了 C++的模板机制。3)代码生成,另一种说法是面向属性的编程。这个在 C#语言里面提供了支持,java1.5 提供了属性的概念。4)元程序设计。5)微软提出的意图编程(IP)。6)面向方面的程序设计,即 AOP。

3　AOP 的思想和概念

AOP 涉及的概念:

Aspect　就是横切了多个对象体系的问题,它描述的不是一个业务而是一个问题。比如说事务管理,安全认证,这些都是跨在多个功能模块的。

植入　把 Aspect 和对象融合成产品代码的过程,就叫植入。

插入点　插入点就是执行植入的地点。比如说事物植入,在开始业务操作之前是一个植入点,植入开始事务的代码;在结束以后是另一个植入点,植入执行 commit 提交事务或者 rollback 回滚事务的代码。

Advice　植入对象体系的代码片断。

Introduce　使植入后的产品代码具有另外一个接口。使产品代码除了提供业务接口外,同时还是一个事务的管理器,还可以执行事务管理,使组件在运行时具有了另外一组接口。

AOP 的核心概念就是正交分离关注点,又称横切关注点。AOP 是一套新的方法,通过提供一种新的能够横切其他模块的模块化单位:aspect(方面),达到了分隔横切关注点的目的。在 AOP 中,在 Aspect 中实现横切关注点,而不是把它们融合到核心模块当中去。利用 Aspect Weaver(方面编织器,类似于编译器),通过一个称为 Weaving(织入)的过程把核心模块和横切模块合并到一起,从而构造出最终的实际系统。

最终,AOP 用一种边界清晰的方式把横切关注点模块化,产生出一个更容易设计、实现和维护的系统架构。

4　主流的 AOP 产品及其优缺点分析

现在主要的 AOP 产品可以分为静态增强和动态增强两大类。

最著名的是 AspectJ,现在它已经合并到了 eclipse 项目中。它是一个静态的植入,工作原理是提供一个预编译器。在产品代码和 Aspect 都做出来以后,用预编译器做一次编译。这样生成的产品代码就同时包含了业务的逻辑和产品的逻辑。这种静态植入方式的明显缺点是:是一种新的语言,语法和 Java 差不多,但还是要学习的;预编译后的代码很难调试(在 eclipse 中要对编译过的代码做单步跟踪很难,因为它植入 Aspect 以后,行号就乱掉了);一旦对 Class 进行了静态增强之后,就不能取消这个预编译。要取消的话,必须重新编译重新生成,不能在运行时进行。

其他的 AOP 实现都是动态的增强,用 Java 的动态代理机制,运行时增强 Java 的二进制代码,例如:AspectWerkz,BCEL,JAC,Javassist,Nanning,SpringAop,Prose 等。

Jboss-Aop 是一个特殊的不是动态代理的动态增强,也不是预编译器的静态增强。它修改了 ClassLoader,在加载类的时候进行植入,可以看成半动态的。因为 Jboss 是一个应用服务器,有权管理它的 ClassLoader,其他的产品作为一个框架,不能管理 ClassLoader。

现在 AOP 的事实标准是 AspectJ,但是上面提到的静态增强和相对动态增强有很多的问题;而动态增强,只要在某个配置文件里面修改一行代码就可以取消某个类的增强。现在 AOP 的标准化组织 AOP - Alliance,提出了标准的实现的 API。

5　传统 Observer 和 AOP 的 Observer

Observer 模式的基本前提是包含两个角色:观察者

(Observer)和主体(Subject),如图 4 所示。它的逻辑模型是,观察者向主体注册,表明它观察主体的意愿。在某种状态发生变化时,主体向观察者通知这种变化情况。当观察者不再希望观察主体时,观察者向主体撤销注册。这些步骤分别称为观察者注册、通知和撤销注册。

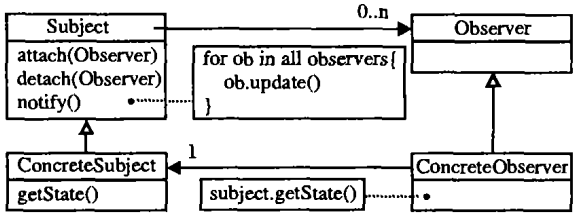

图 4 观察者和主体类图

5.1 传统的面向对象的 Oberver 实现

传统的 Observer 与业务逻辑混淆,不便维护。以它的 notify 为例,observer 是这样给 subject 发 message 的。

```
// Case1 : in every business method
...                                    // some business logic
Message message = populateMessage();
broadcast(message);
// Case2 : in every client
service.doSomething();
Message message = populateMessage();
service.broadcast(message);
```

可见,Case1 与业务的逻辑相混淆,还隐藏了业务逻辑知道自己被观察,要取消观察的话还要修改这一段代码;Case2 干扰客户端的逻辑,而且做法很不安全。

5.2 AOP 的 Observer 实现

我们使用 Spring Framework 中的 AOP 实现方法。Observer 逻辑不与业务逻辑混淆;独立模块管理 Observer,修改 Observer 时只需要关心这一个模块。

我们以用户管理为例。它将要观察的接口及其方法:

```
public interface IUserManager extends IService {
  public UserDTO registerNewUser
    (String userName, String password, String mail);
  public UserDTO userLogin
    (String userName, String password);
  public Serializable unregisterUser
    (String userName, String password);
}
```

它的业务模块和客户端代码都看不到使用 Observer 模式的地方。它的实现是在 ApplicationContext 中装配所需要的 bean,粗体为与拦截有关的代码。使用了两种拦截器,一种是拦截具体的方法,一种是拦截一个接口的所有方法。如下:

```
- <beans default-lazy-init="false" default-dependency-check="none" default-autowire="no">
- <bean id="userManagerTarget" class="TestUserManager" singleton="true" lazy-init="default" autowire="default" dependency-check="default">
- <property name="listeners">
- <list>
<value>net.sf.groller.observer.impl.TestListener</value>
</list>
</property>
- <property name="observedMethodNames">
- <list>
<value>registerNewUser</value>
<value>unregisterUser</value>
</list>
</property>
</bean>
- <bean id="userManager" class="org.springframework.aop.framework.ProxyFactoryBean" singleton="true" lazy-init="default" autowire="default" dependency-check="default">
- <property name="proxyInterfaces">
<value>service.IUserManager</value>
</property>
- <property name="interceptorNames">
<value>observerInterceptor,userManagerTarget</value>
</property>
</bean>
<bean id="observerInterceptor" class="net.sf.groller.interceptor.ObserverInterceptor" singleton="true" lazy-init="default" autowire="default" dependency-check="default" />
</beans>
```

Junit 测试代码为:

```
InputStream in = getClass().getClassLoader().
  getResourceAsStream("application.bean.xml");
BeanFactory factory = new XmlBeanFactory(in);
IUserManager manager =
  (IUserManager) factory.getBean("userManager");
manager.registerNewUser
  ("rocky", "password", "yuqzhang@sina.com");
```

运行结果显示 TestListener 对 IUserManager 的 registerNewUser, unregisterUser 方法; ObserverInterceptor 对 IUserManager 的所有方法成功进行了拦截。

它的实现方法比传统的 OO 实现方法更加优雅,实现了模块间的松耦合,增加了系统的灵活性,当修改 Observer 模式的监听器和监听方法时,只需要修改配置文件就可以了,不需要在相关的业务模块上进行修改。

6 结语

从演化的角度来看编程方法,过程性编程引入了功能抽象,OOP 引入了对象抽象,而现在 AOP 引入了关注点抽象。当前,OOP 是绝大多数新软件开发项目所选择的方法,OOP 的力量来自于共同行为的建模。但是,它在处理那些分布于许多个通常还是没有关联的模块之间的行为上做得并不好。AOP 填补了这一空白。

AOP 的前途是很光明的,它潜在的发展前景有:

1)另外一种体系结构的出现,使更多的内容可以抽象成 Aspect,以便使用 AOP 的方法;

2)对业务代码不产生干扰的模式,比如上面提到的 Observer 模式;

最终达到的目标是无限制的业务组件,这样业务组件完全可以重用,重用的时候只用配备不同的 Aspect 就可以了。

参考文献:

[1] (美)GAMMA E, HELM R, JOHNSON R, 等. 设计模式:可复用面向对象软件的基础[M]. 李英军, 马晓星, 蔡敏, 等译. 北京:机械工业出版社,2000.

[2] Aspect-Oriented Programming [A]. Proceedings European Conference on Object-Oriented Programming[C],1997.

[3] 轻量级框架[EB/OL]. http://www.springframework.org/, Spring Framework, 2004-02.

[4] 基于 J2EE 的 AOP 标准化组织网站[EB/OL]. http://aopalliance.sourceforge.net, 2004-03.

[5] groller 开源项目[EB/OL]. http://cosoft.org.cn/projects/groller, 2004-03.