基于 AOP 的物流遗留系统安全进化

Security Promotion in Logistic Legacy System Based on AOP

杨晓燕 雷 霞 (西安工程大学管理学院 陕西西安 710048)

摘要:本文结合物流遗留系统的实际安全状态,分析了面向对象的编程思想在横切关注点和核心关注点处理上的不足,指出面向方面的编程思想解决方案对系统进行分离关注点处理的优势,并对面向方面的编程的一种具体实现 AspectJ 进行分析,提出了一种依据 AspectJ 对遗留物流系统进行 IC 卡安全进化的方法。

关键词:AOP OOP AspectJ 遗留系统 安全进化

1 引言

随着计算机网络技术的飞速发展,很多企业,尤其是物流企业,急需对企业原有的管理信息系统进行升级改造。根据对企业的不同遗留系统的综合评价,可采取不同的策略,如进行集成、改造、淘汰和继承等。有些企业部分遗留系统的技术含量比较高,运行仍很稳定,基本能满足目前的业务需求,还具有很大的生命力。由于原来设计开发时,对于安全方面的考虑较少,采用系统安全管理方面的手段也较少,企业对核心信息的保密性要求和目前业务的安全性要求,迫切需要对原企业遗留系统进行安全方面的升级改造,以使得这部分系统经过安全性能提升后既可保持原遗留系统的稳定运行,又可提高业务安全系数,以降低企业在管理信息系统升级改造方面的成本。对这种遗留系统可采用进化改造策略。

但在遗留系统演进过程中,随着许多功能性和非功能性需求的增多可能已经向遗留系统添加许多第三方的库或框架,它们全都彼此交互,并在系统日常工作的表面之下相互配合。安全性模块并不实现主要的业务功能,只辅助这些功能的实现,并分散在遗留系统的许多模块中。若采用面向对象的编程思想对遗留系统进行安全进化,则需对很多遗留系统代码模块进行改造升级。而由于大多数遗留系统的规模较大,内在的难以理解性及可能添加后续功能升级模块,向遗留系统的程序代码添加新的安全提升功能模块,可能要面临很大的未知风险,并且改造后的系统运行状态很难保证,企业投入面临较大风险。

面向方面编程方法(Aspect Oriented Programming,AOP)给我们提供了一个研究和解决该问题的新视角。AOP 技术是面向对象编程(Object Oriented Programming,OOP)技术的一种有益补充。本文应用 AOP 技术解决物流遗留系统的安全进化,实现了新老功能的集成,并描述了其实现过程。

2 物流遗留系统的安全问题

物流是指物品从供应地向接收地的实体流动过程,现代物流系统是从供应、采购、生产、运输、仓储、销售到消费的供应链。开发物流信息系统的目标就是要帮助业务人员提高工作效率,同时使管理人员能够实时了解企业的经营状况,为辅助决策提供科学依据和参考,提高企业的管理水平。

该物流遗留系统是采用 Java 语言开发的多用户的软件系统,目前仍运转良好。在该系统中,为安全起见和职责明确,用户所能操作的功能根据角色的不同进行区分,以防止未经授权的访问。所采用的安全权限验证方式为常见的用户名和密码。在实际运行的过程中却发现,很多用户为避免忘记密码,直接把密码贴在自己的电脑上,以致密码公开,不同角色的用户相互之间都可操作,失去了权限管理的作用,且责任不易区分。希望对该系统进行改造,实行 IC 卡加密码验证机制,每个用户拥有一张和自己的角色对应的 IC 卡,在登录的时候需要在 IC 卡读写器中插入 IC 卡然后在系统中输入自己的密码。这就需要对系统进行安全改造,以实现安全性能的提升。

3 AOP 的基本思想

目前，OOP 已成为系统开发的主流。采用 OOP 技术可实现组件的可重用性、模块化、系统实现复杂度的降低、后期维护成本减少等优点。

若采用 OOP，则一个软件系统可以由不同的类的集合来构建。每一个对象都有一个明确的任务，是对某个具体系统局部目标进行属性和方法的封装，即对象应该是一种自包含的模块，对象完全不理解其所处的外在状态，而其所处的环境也只是了解对象所公开的方法，对象内部的隐藏起来的属性或方法仅仅在对象内部可见，其示意图如图 1 所示。

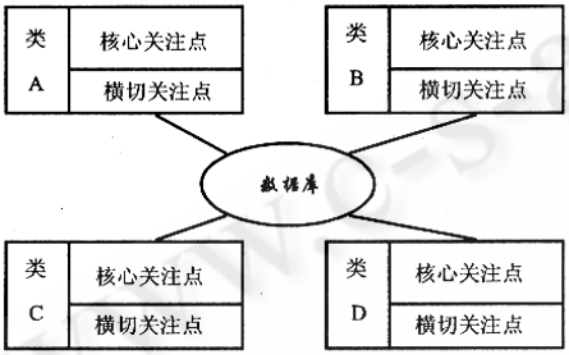

图 1　OOP 处理横切点示意图

在面向对象的应用中，系统中所有的类一同协作来实现应用系统的总体任务。在系统中实现业务功能的模块是应用系统的核心，我们称之为核心关注点。面向对象的编程技术对于实现核心关注点的功能非常有效。然而在实际的应用中，存在这样的一些任务，它们不能只是被看作单个类的责任。比如：分布式系统中的锁机制、日志管理、长和短事务处理、对上下文敏感的异常处理、整体性能调优等功能。这些功能和方法通常并不实现系统的业务功能，而是辅助这些业务功能的实现。这些功能模块和实现系统业务功能的模块产生了横切，因此称之为横切关注点。横切关注点代码散布在实现系统功能的诸多模块中，从而造成代码的缠绕。这使得实现系统功能模块的代码难于阅读、理解、调试、维护和扩展等，同时纠结的代码也很难于被复用。系统功能之间产生的横切关系如图 2 所示。

鉴于此，AOP 提出的解决方法是对这两种相互横切的关注点分别进行编码，使用传统的编程语言对核心关注点编程，使用面向方面的编程语言对横切关注点（即方面）进行编程。然后使用一种机制将这两种代码自动混合起来，形成最终的代码。因此，对方面单独编码，并通过适当的织入机制使两种代码混合，构成了 AOP 解决代码纠结问题的基石。AOP 的编程模型如图 3 所示。

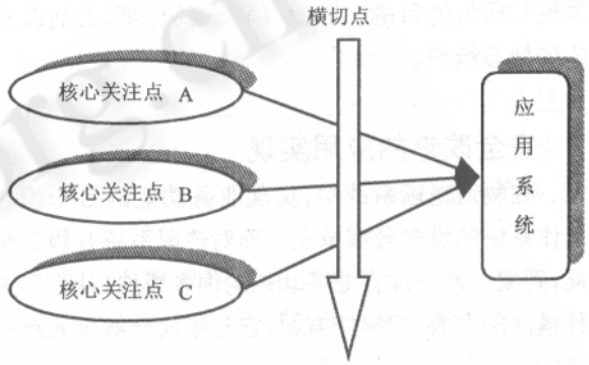

图 2　系统的横切关系

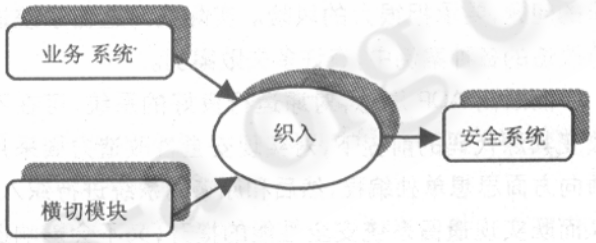

图 3　AOP 编程模型

4 AspectJ

AspectJ 是目前最完善的 AOP 语言，是对 Java 编程语言的扩展，通过增加了一些新的构造模块支持对横切关注点的模块化封装。在业务功能代码中的适当位置，比如某段代码执行前，或执行后，将方面模块中的编码织入，从而形成混合的编码。AspectJ 提供了两种横切实现机制，一种称为动态横切（Dynamic? Crosscutting），另一种称为静态横切（Static? Crosscutting）。

动态横切是指在程序执行的某一个明确的点上运

行额外的，预先定义好的实现。在编译时，方面中的通知将被转化为标准的方法，类代码中匹配切点的连接点将被转化为一个静态的标记点，然后，这些静态点将被通知所转化成的方法的调用所取代，由此完成两种代码的织入，最后对织入完成的代码编译为字节码，即完成了整个编译过程。

静态横切则允许我们通过注入（inject in）附加的方法或属性来直接改变对象的结构，达到改变对象的属性和行为的目的。因此，静态横切实际上是改变对象的静态结构。

5 安全改造的应用实现

在物流遗留系统中，实现业务功能的各个模块可看作系统的纵向分解单元。而对遗留系统升级安全性能，则要涉及分散在业务功能的很多模块，从而形成一种横向的方面，即横切方面，它与纵向分解单元产生了横切。

因此对于该物流遗留系统提升其安全性能时，若仍采用面向对象编程技术实现，可能对很多的核心关注点模块都要实行改造，不仅工作量巨大，且会产生大量的问题，要承担很大的风险。实际在对遗留系统进行改造的各种案例中，有许多失败案例。

而采用 AOP 思想，对原运行良好的系统，可在不改变其源代码的前提下，对实现安全性改造方面采用面向方面思想单独编程，然后和原来的系统进行织入，从而既实现遗留系统安全性能的提升，又不会影响遗留系统运行。

AOP 编程的模型如图 4 所示。

在该遗留系统中，实现系统登录功能的模块类为：

```
public class UserLoginTest
{
  //系统其余功能模块
  //……
  //获取用户名
  public String GetUserName( );
  //获取用户密码
  public String GetUserPassword( );
  //验证用户名和密码
  public boolean TestValidate( );……
  //系统其余的功能
  //……
}
```

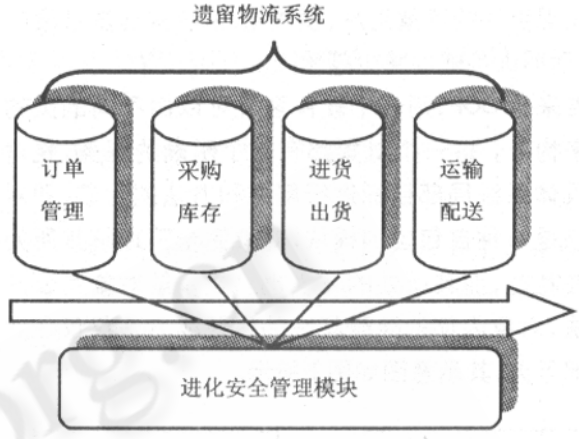

图 4　进化安全模块和系统横切图

该功能在系统中的很多模块中都有使用。为添加 IC 卡验证机制，首先需要评估增加该功能可能对系统带来的影响，故我们采用动态横切技术。此技术在发现特定的连接点可达时会发出编译时警告，然后编写连接点来捕捉针对该特定接口的所有方法的调用，就可以查找出有可能影响的遗留系统程序代码。我们首先在本地区域中提取出应用程序代码库，修改生成文件以使其使用 AspectJ 编译器以及在适当的位置包括方面类，然后运行系统的完整生成文件。编写 Aspect 代码如下

```
public Aspect UserLoginTestAspect {
    //捕捉对 GetUsername( ) 的调用
    pointcutGetUserNamePoint( ) :
      call ( public String UserLoginTest. GetUserName
( ) )
      && ! within ( UserLoginTest + ) ;
    //捕捉对 GetUserPassword( ) 的调用
    pointcutGetUserPasswordPoint( ) :
      call ( public String UserLoginTest. GetUserPass-
word( ) ) && ! within ( UserLoginTest + ) ;
    //捕捉对 TestValidat( ) 的调用
    pointcutTestValidatePoint( ) :
      call ( public void UserLoginTest. TestValidate( ) )
&&
      ! within ( UserLoginTest + ) ;
```

```
    //切点对应的连接点发生签名匹配时,编译器发出警告
    declare warning: GetUsernamePoint( ):
      " Call to UserLoginTest. GetUserName( )" ;
    declare warning: GetUserPasswordPoint( ):
      " Call to UserLoginTest. GetUsePassword( )" ;
    declare warning: TestValidatePoint ( ):
      " Call to UserLoginTest. TestValidate( )" ;
    }
```

对于 UserLoginTestAspect 方面 ,项目在进行编译时候,当有别的模块调用这三个方法时,编译器可给出警告,从而可有效地检查出在哪些模块中使用了这三个方法。采用类似的方法,我们能够彻底搞清系统登录模块在整个系统中的使用方式及与别的模块之间的相互影响。而完成之后,就可把此 aspect 删掉。

然后再采用静态横切技术,根据获得的对系统模块之间的相互影响的知识,对遗留系统的原有的登录模块添加属性和方法,改变其静态结构。建立 IC 卡验证 aspect:

```
public aspect ICCardValidate
{
    private String UserLoginTest. ICCardInoformation;
    private boolean UserLoginTestValidateResult;
    public String UserLoginTest. GetICCardInformation
( )
    {
      //通过 IC 读写器,获取 IC 卡信息中的安全代码
    }
    public Boolean UserLoginTest. ICCardValidate
(String s)
    {
      //根据数据库中的信息验证读取的 IC 卡信息
    }
   //捕捉对于 GetUserName( )的调用
   Pointcut GetUserNamePoint( UserLoginTest p):
     call ( public String GetUserName ( )) && target
(p);
    before( UserLoginTest p) :GetUserNamePoint( p)
    {
      ICCardInformation = p. GetICCardInformation ( );
ValidateResult = p. ICCardValidate(
          ICCardInoformation) ;
      //.....
      //根据验证结果进行处理的代码
    }
}
```

5 总结和展望

通过 AOP 技术,在不改变原有系统代码的前提下,我们成功地为遗留物流系统增加了新的横切功能点,实现了系统的平稳运行,提升该系统的安全性能,延长了该系统的使用周期,省降低了企业的成本支出。该系统能够稳定地与企业更新后管理信息系统其余模块协作运行。

故采用 AOP 思想,能够提高遗留系统的可维护性和可重用性。在此方面,AOP 是一种比 OOP 思想更为优秀的方法。从编程方法学角度讲,为遗留系统保持活力增添了新的方法。在当今软件规模日益扩大,结构日益复杂,系统更新换代速度日益加快的今天,AOP 将会在对遗留系统的维护和进化中发挥愈来愈重要的作用。

参考文献

1 Laddad R. AspectJ in Action[M]. America: Manning ublica – tions,2003.

2 The AspectJ Programming Guide, http: // aspect. jorg.

3 (爱尔兰) SiobhánClarke, ElisaBaniassad. Aspect – Oriented Analysis And Design – The Themeap-proach[M]. 北京:机械工业出版,2006.

4 曹东刚、梅宏,面向 Aspect 的程序设———种新的编程范型[J],计算机科学, 2003, 30 (9):5 –10.

5 Kiczales G, Lamping J, Mendhekar A, et al. Aspect – oriented Pro – gramming[C]. Proc. of ECOOP' 97, Springer – Verlag, 1997: 220 – 242.

6 AspectJ Team. The AspectJTM Programming Guide [Z]. http://eclipse. org/aspectj/

7 http: //www. eclipse. org/aspectj/, 2007.