文章编号: 1000-8608(2007)02-0270-06

一种基于AOP的软件演化策略设计与实现

郭 禾*, 刘天阳, 陈 锋, 王宇新, 刁华丽

(大连理工大学 电子与信息工程学院, 辽宁 大连 116024)

摘要: 软件演化通常需要向系统的各个模块中添加新的通用功能, 一般是在每个有需求的模块中直接插入相应的功能代码, 但这种做法工作量大、出错率高, 同时容易破坏系统的结构. 为此介绍了一种基于面向方面的编程(AOP)的软件演化策略, 该策略通过使用关注点, 向演化系统中注入新的代码, 而不改变系统的结构. 除此, 还介绍了基于此方法的支持软件演化所实现的一个工具, 并通过实例对该方法加以了验证和说明. 该方法只是应用了AOP的思想, 因此不局限于支持AOP的程序设计语言, 对于一般的面向对象的程序设计语言都适用.

关键词: 面向方面的编程(AOP); 软件演化; 动态编织
中图分类号: TH166 文献标识码: A

0 引 言

软件系统的演化通常需要向系统中添加新的通用功能, 这些功能分布于系统的各个组成模块之中. 实现这些功能, 最直接的做法是在每一个有需求的模块中直接插入或调用相应的功能代码. 但是这种方法工作过于烦琐, 增加了出错几率, 破坏了系统的封装性.

作者将面向方面的编程(aspect-oriented programming, AOP)思想运用到软件演化中, 提出一种基于AOP的软件演化策略, 即将新的通用功能作为方面分离, 独立进行配置, 并在需要时将新的通用功能注入原系统, 以避免对原代码的直接修改, 降低出错几率, 维护系统的结构.

1 软件演化和AOP

1.1 软件演化

软件演化被定义为对成功开发的原始系统的软件维护, 分三大类: 修复系统潜在漏洞的校正行为; 处理不断变化环境的适应行为; 满足新的用户需求的完善行为.

现在存在着大量的、对用户来说非常重要的软件系统, 它们日益变得难以维护和改进, 难以满足快速变化的用户需求. 如何使系统动态地适应不断变化的需求, 顺利实现软件演化, 对延长系统的生命周期非常有意义.

目前的软件演化研究多集中于如何对软件系统进行分析, 从而找出需要演化的构件. 怎样定义一个较为通用的演化策略, 高效地修改这些构件, 依然是一个还在研究的问题[1]. Devanbu等给出了一种AspectJ下扩充软件功能的实现方法, 但该方法也有一定局限性.

1.2 AOP

AOP是由Kiczales等在1997年提出的[2]. 其基本思想是将横跨多个模块的行为分离出来, 并封装为一个新的模块[2]. 可单独对它进行编程、修改, 在需要时使用编织器把这个模块编织到原有的代码中[3]. 它克服了OOP不能处理跨越多个不相关模块的行为的缺陷[4].

AOP的核心概念是关注点, 指在程序执行过程中定义好的点, 如方法调用、循环开始和对象构建等[5]. 具有横切多个模块的行为的一类特殊关注点称为横切关注点. AOP提供了一种描述横切关注点的机制, 能够自动将横切关注点织入到面向对象的软件系统中, 从而实现模块化. 被模

收稿日期: 2005-08-10; 修回日期: 2007-02-10.
作者简介: 郭 禾* (1955-), 男, 副教授.

块化的横切关注点就称为方面. 开发者可以在编译时更改、插入或删除系统的方面,甚至重用系统的方面.

现有的 AOP相关研究侧重于在项目的前期设计阶段,对横切关系进行分离,应用 AOP[6 7]. 将 AOP思想运用到软件演化中,是对软件演化策略的一个新的尝试.

2　基于 AOP的软件演化策略

2.1　具体策略

下面以一个需要演化的系统为例,用以说明基于 AOP的软件演化策略.

图 1描述了一个需要演化的系统[8]. 根据用户需求,要向多个实体类中添加 SQL字符串校验功能. 采用基于 AOP的软件演化策略,该功能被作为方面类单独实现,并插入方面消息链中. 当程序运行时,实体类不是显式调用方面类,而是通过类似于编译器的编织器,将方面类和目标类合并在一起.

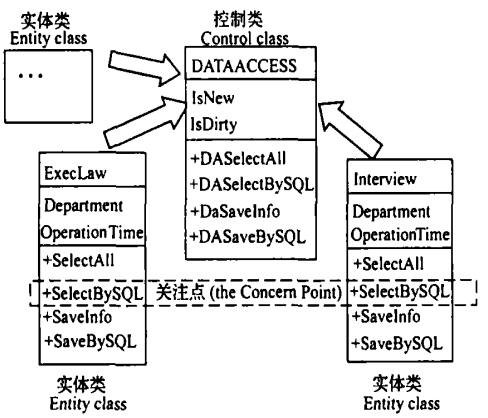

图 1　需维护的系统

Fig. 1　The project need to be maintained

图 2为基于 AOP的软件演化策略的实现框架. 具有相同横切关注点的类聚合为目标类集合,其中的所有对象都具有相同的方法,即关注点. 模块化的横切关注点称为方面类,不同方面类组成方面消息链. 方面消息链的起点为监视器,终点为目标类.

对目标类中的关注点进行调用时,编织器中的监视器捕获对关注点的调用消息,将其沿方面消息链传递. 在消息到达目标类之前,方面类对消息进行处理,即编织器将需要的方面功能织入目标类中. 这样,在不修改系统原代码,维护系统封装性的前提下,实现了功能的添加.

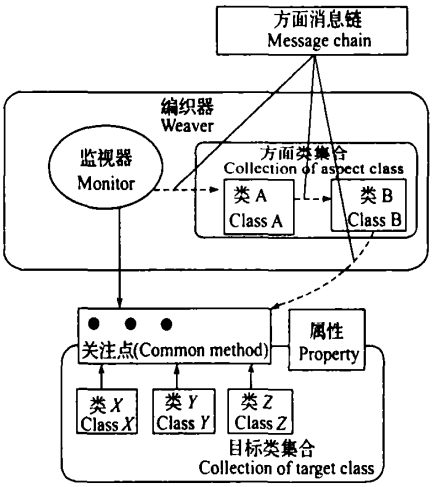

图 2　一种 AOP的实现框架

Fig. 2　The implementation structure of AOP

关注点被定义为目标类中的公共成员函数,当公共成员函数被调用时,将自动引发方面类. 为在 Microsoft. NET实现这种机制,引入了代理类子类 (RealProxy). 类似 CORBA 中的 Skeleton,作为一种透明代理对远程对象的公共成员函数进行调用.

如图 3所示, RealProxy是代理类的子类,在代理类中定义的调用方法在 RealProxy中重写. 客户端将消息传给代理类实例而不是真正的远程对象. 通过这种方式,客户端把远程对象当做本地对象处理. 代理类捕获所有的调用消息并决定是否执行方面功能,起到编织器中监视器的作用.

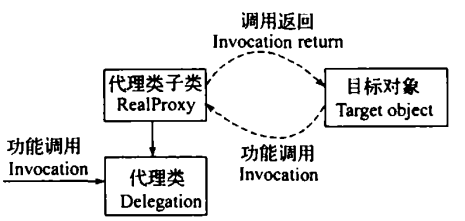

图 3　代理类框架

Fig. 3　Delegation structure

方面功能在各自的方面类中实现,而不在代理类中实现. 运行时,所有的方面类实例连接成方面消息链.

如图 4所示, Microsoft .NET中提供 ImessageSink接口构建方面消息链. 方面消息链中的对象使用 SyncProcessMessage传递消息给下一个对象,而不直接调用其函数. 对象使用 SyncProcessMessage 传递消息, 抽象类

AspectAttribute作为基类被创建,所有的方面类都从该基类派生,保证系统结构清晰.

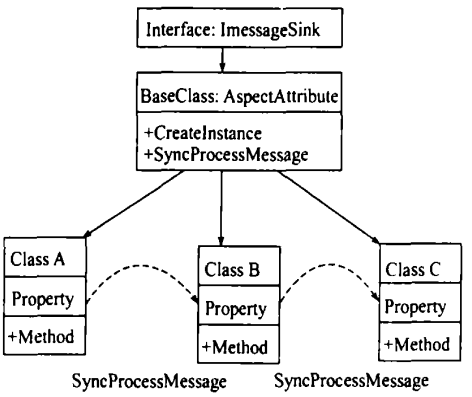

图 4 方面消息链的结构

Fig. 4 Message chain structure

所有的消息,包括构造器调用都通过代理类捕获. 当代理类捕获构造器消息时,目标类的实例被创建. 然后创建方面消息链,其中代理类为起点,目标类实例为终点. 此后,所有的消息将通过方面消息链传给目标类,见图 5.

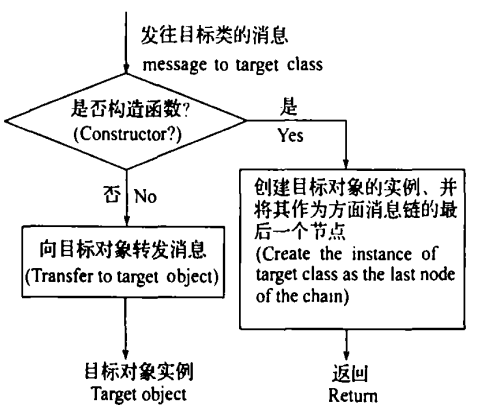

图 5 代理类中的消息传递

Fig. 5 Message process in delegation

方面类对调用者应当是透明的. 这通过使用元数据的相关属性实现. 在 Microsoft .NET环境中,上下文指的是一个对象的边界. 处在同一上下文中的所有对象共享相同的类库. 新的类实例创建后,系统自动检验当前上下文与实例是否一致. 不一致时,系统自动创建新的上下文. 类可以是基于上下文的,称为基于上下文的类. 如图 6,基于上下文的类从基类派生,如ContextBoundObject,任何基于上下文的实例都有各自的上下文对象. 基于上下文的实例通过.NET运行时创建私有的上下文,这为扩展方面消息链中的方面类提供了方法. 因为私有上下文的建立会引起代理类的创建[9].

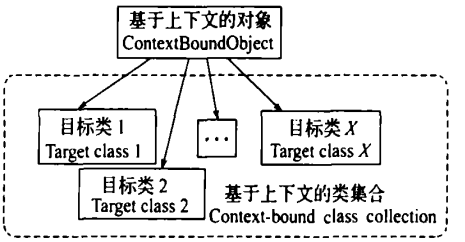

图 6 基于上下文的类集合

Fig. 6 The collection of context-bound class

基于上下文的类具有属性类. 属性被添加到基于上下文的类中描述元数据[7]. 实际上,方面类也是一种属性类,从代理属性派生.

如果一个属性类从代理属性派生,用来描述基于上下文的类,那么属性类的 CreateInstance事件将在目标类的实例创建之前被引发. 在目标类的实例创建过程中,系统将构建代理类并建立方面消息链.

2.2 规则分析

并非所有类都适合使用上述提到的方法添加AOP功能. 因此,本节定义了一些规则来帮助设计人员进行分析,作出更好的选择. 这些规则来源于 2.1中所描述项目的演化中的分析,并在项目的后续演化工作中进行了验证. 规则能够从类或者类的方法的层面帮助开发人员筛选出适合应用 AOP方式演化的类,但这种筛选只是一种充分性筛选,对于少数业务功能比较特殊、实现形式较为复杂的类模块并不适用.

2.2.1 目标类选择的规则

Rule 1 应用方面功能的类必须派生于基于上下文的对象.

只有客户端和对象处于同一个应用域中,才能建立二者之间的代理. 此时,方面类将作为方面消息链中的一个链接事件被实现,而不参与客户端的任何行为[10]. 根据此规则,编译好的组件或 COM对象不能应用方面函数.

Rule 2 应用方面功能的类不应该包含递归公共成员函数.

因为代理检查所有发送到目标类实例的消息,如果目标类中包含递归的公共成员函数,那么所有对这些功能的调用都引起代理类的校验,这将大大降低系统的性能[11].

利用规则 1和规则 2,开发人员可以判断出一

个类是否适合作为目标类,应用方面功能.

2.2.2 关注点选择规则

Rule 3 如果公共成员函数在目标类实例创建之前被调用,它不能被定义为关注点.

基于 AOP的软件演化策略的原理是捕获对目标类的公共成员函数调用,并作出相应的修改和控制. 如果一个成员函数,如静态函数,在代理创建之前被调用,它将不能被定义为一个关注点.

2.2.3 针对效率和性能的规则

Rule 4 代码精简的倍数 C_r 至少应该大于 1.

代码精简的倍数 C_r 可用以下公式计算:

$$C_r = \frac{m\times k\times l + s}{m + s + W} \tag{1}$$

式中: m 为目标类的数量; k 为目标类中关注点的数量,通常,每个目标类包含相同数量的关注点; l 为调用方面函数的源代码的行数; s 为方面函数源代码的行数; W为方面基类的源代码行数; $m\times k\times l + s$ 代表采用普通方法实现的代码长度; $m + s + W$ 代表采用基于 AOP的策略之后的代码长度. C_r 值越大,意味着代码简化程度越大.

Rule 5 系统性能的提高率 E_r 不能小于某一下限.

使用上述提到的 AOP的实现方法,代理将检验发往目标对象的所有消息,这将导致系统性能的下降.

系统性能的提高率 E_r 可用如下公式计算:

$$E_r = \frac{1}{\sum_{i=1}^{m} k\times C_i} \tag{2}$$

式中: C_i 为第 i 个目标类的公共成员函数的数量(注意,将公共属性当做一种特殊的公共成员函数处理); $\sum_{i=1}^{m} k\times C_i$ 为应用方面功能后进行消息检验的次数. E_r 用来评估演化系统的性能.

如果系统性能不能满足需求,方面函数将不能被使用.

3 工具支持

为了验证上述方法,设计并开发了一个原型工具 EvoWeaver.

EvoWeaver是一个半自动化的工具,可以辅助软件开发者实现基于 AOP的软件演化.

图 7展示了 EvoWeaver工具的主界面. EvoWeaver工具首先加载和分析系统的源代码.

在工具窗口中,可通过 4个方面对方面类进行配置:

(1) 命名空间: 选择常用的命名空间.

(2) 代理: 这里有很多可被用做代理的组件. Frei提出了很多利用不同组件创建代理的方法. 默认代理为 RealProxy.

(3) 消息传递方法: 有两种传递消息的方法,异步消息传递和同步消息传递. 默认选项为同步消息传递. 使用同步消息传递意味着只有在一个函数调用返回之后,消息才能被传递到下一个节点.

(4) 命令选项: 默认选项是强制的,这意味着演化系统将应用方面类而不考虑系统效率.

图 7中还包含了 3个结构图. 结构图 1显示了演化系统中的所有类. 双击树上的节点将弹出显示类代码的窗口. 结构图 2中显示了满足规则 1和规则 2的所有类. 这些类按照它们之间的继承关系形成树状结构. 当结构图 2中的节点被选中,应用规则 3这个类的所有属性以及被选中的公共成员函数将在结构图 3中显示.

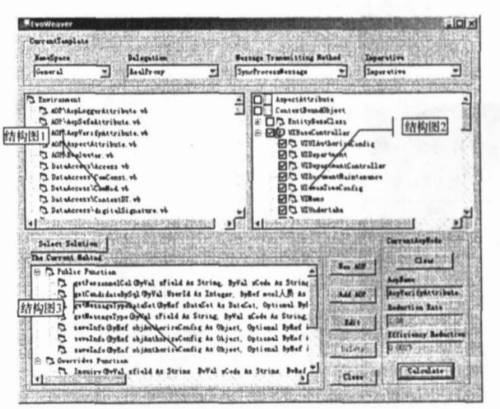

图 7　EvoWeaver工具

Fig. 7　EvoWeaver tool

当目标类和方面函数选定后,演化系统的代码精简率和性能效率即被计算出来. 开发者可以据此判断基于 AOP的软件演化方法是否能被采用.

为方便添加方面功能,还提供了 AOPEdit编辑工具. 开发者通过 AOPEdit工具编写方面功能代码. 通过 .NET编译器验证的代码,作为方面类自动插入方面消息链中.

为验证策略的可行性,引入 2.1节中 SQL字符串校验实例和事件日志实例. 事件日志是指系统跟踪事件上下文,如操作者、操作和操作时间,

并将其记录到日志文件中. SQL字符串校验为向少数类中添加新的功能,而事件日志为向较多的类中添加新功能. 由于篇幅有限,仅给出分析结果.

4 实例结果及系统性能分析

4.1 整体分析

表 1是对上述两个实例的整体分析,数据来源于 2.1节所描述的演化例子. 结果表明 AOP技术在精简代码方面有积极的意义[3].

为了判断成员函数是否为关注点,第一个实例中的所有目标类在运行时,需要增加额外判断,这将导致系统性能的下降.

表 1 性能分析

Tab. 1 The performance analysis

	SQL校验	事件日志
目标类数量	39	64
目标类中关注点的数量	1	6
不使用 AOP方法的情况下的代码行数	408	1 824
使用 AOP方法的情况下的代码行数	296	295
代码精简率	1.38	6.18
接口校验减少的次数	38	363
额外增加的判断次数	439	5 380
性能提高率	0.002 3	0.000 2

4.2 系统性能分析

影响系统性能的因素很多,如方面类的数量、目标类的数量以及类中公共成员函数的数量. 同时,方面消息链管理和额外增加的检查都将影响系统的性能.

假设每一个目标类具有相同数量的公共成员函数 C_i. 这里取实例中的公共成员函数的数量的最大值,$C_i = C_{max} = 20$.

根据式 (1) 可以得出

$$E_r = \frac{1}{\sum_{i=1}^{m} k \times C_i} \xrightarrow{C_i = C_{max}} E_r = \frac{1}{m \times C_{max} \times k} \tag{3}$$

式 (3) 表示了系统性能和目标类数量的函数关系. 图 8显示了系统性能提高率随目标类数量变化而变化的曲线. 可以得出以下结论:

(1) 系统性能将随着目标类数量的增加而降低. 在目标类中应用方面功能将导致系统性能的下降.

(2) 当目标类数量大于 20时,系统性能提高率迅速降低. 当目标类的数量达到 48后,系统性能提高率几乎不变. 这说明,当目标类的数量到达某个临界点之后,性能提高率将主要依赖于方面函数的性能而不是目标类的数量.

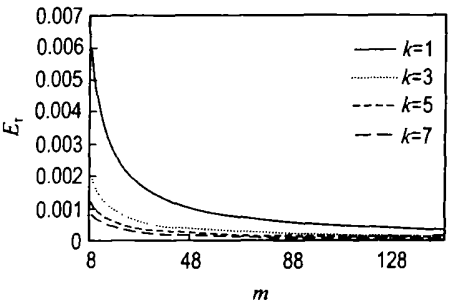

图 8 系统性能提高率随目标类数量变化而变化的函数曲线 (不同 k)

Fig. 8 Efficiency rate as a function of the amount of target classes for different k

(3) 关注点的数量即式 (3) 中的 k,通过曲线线形的不同加以体现. 当 k 值增大时,系统的性能提高率将减小,即系统性能降低.

假设式 (3) 中的 E_r 等于经验中的最小值 0.000 1,将得到如下表达式:

$$E_r = \frac{1}{m \times C_{max} \times k} = 0.000\ 1 \Rightarrow C_{max} = \frac{10\ 000}{m \times k} \tag{4}$$

式 (4) 表明了目标类中公共成员函数的数量和目标类数量之间的关系.

图 9显示了每个目标类中函数数量随目标类数量变化的曲线.

正如图 9显示的那样,当性能提高率保持不变的情况下,目标类的数量和每个目标类中函数的数量成反比. 目标类中的成员函数越少,系统扩展的能力将越强.

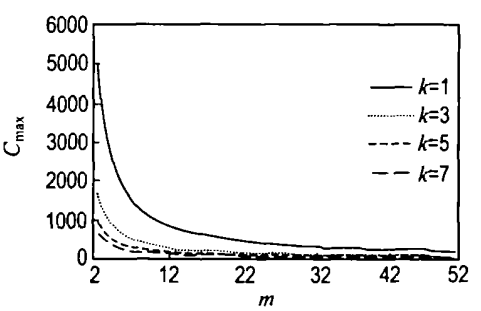

图 9 C_{max} 和 m 之间的函数关系

Fig. 9 Relationship between C_{max} and m

5 结　论

研究表明,当需要添加横切功能时,基于 AOP的软件演化策略是适用的. 新的横切功能将不会破坏系统的结构和封装性. 根据上面提出的分析规则, EvoWeaver工具辅助应用方面功能的目标类选择恰当的成员函数,并能评价应用 AOP软件演化策略后系统的性能. 相对于现有的软件演化方法,这种策略适用面更宽、效率更高.

最后,该工具支持在运行时创建执行代码,即动态编译. 如何利用该机制构建编织器克服工具的上述缺点,还有待于进一步研究.

参考文献:

[1] MENS T, WERMELINGER M, DUCASSE S, *et al*. Challenges in software evolution [C] // **Proceedings of the Eighth International Workshop on Principles of Software Evolution**. Lisbon: IEEE, 2005: 13-22

[2] KICZALES G, LAMPING J, MENDHEKAR A, *et al*. Aspect-oriented programming [C] // **Proceedings of ECOOP'97**. Finland: Springer-Verlag, 1997: 220-242

[3] KICZALES G, HILSDALE E, HUGUNIN J, *et al*. An overview of aspectJ [C]// **European Conference on Object-oriented Programming**. Budapest: Springer-Verlag, 2001: 327-353

[4] LOPES C V, KICZALES G. Recent developments in aspectJ [C] // **European Conference on Object-oriented Programming (ECOOP 98)**. Brussels: Springer-Verlag, 1998: 398-401

[5] PAHLSSON N. Aspect-oriented programming [R] // Topic Report for Software Engineering. Sweden: Department of Technology, University of Kalmar, 2002

[6] ATKINSON C, KUEHNE T. Aspect-oriented development with stratified frameworks [J]. **IEEE Software**, 2003, **20**(1): 81-89

[7] MENDHEKAR A, KICZALES G, LAMPING J. RG: A case-study for aspect-oriented programming [R]// Technical Report SPL97-009P9710044. Palo Alto: Xerox PARC, 1997

[8] GUO H, CHEN F, WANG Y. A reusable software architecture model for manufactory management information system [C]// **26th IEEE International Conference on Computer Software and Application**. Oxford: IEEE, 2002

[9] SCHULT W, POLZE A. Aspect-oriented programming with C# and .NET [C]// **Fifth IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'02)**. Washington D C: IEEE, 2002: 241-249

[10] GARSON E, THOMAS D. Aspect-oriented programming in C# /.NET [J]. **Visual Syst J**, 2004(2): 33-38

[11] ISHIO T, KUSUMOTO S, INOUE K. Program slicing tool for effective software evolution using aspect-oriented technique [C]// **The Sixth IEEE International Workshop on Principles of Software Evolution (IWPSE'03)**. Helsinki: IEEE, 2003: 3-12

Design and implementation of a software evolution strategy based on AOP

GUO He*, LIU Tian-yang, CHEN Feng, WANG Yu-xin, DIAO Hua-li

(School of Electr. and Inf. Eng., Dalian Univ. of Technol., Dalian 116024, China)

Abstract Software system evolution is often required to add some new public functions, which could be distributed in many components on the system. A normal method is to insert code into each corresponding class, which may be just a trivial task but may also increase the risk of introducing errors and destroy the structure of the system. To solve this problem, an aspect-oriented programming (AOP) based software evolution approach based on .NET is introduced. By using 'Joinpoints', the proposed approach can insert new code into the evolving system without any modifications to the existing class structures. A tool in AOP for supporting the system evolution was developed and illustrated in instances. The method can be used in general object-oriented programme design language because it only applies the idea of AOP.

Key words aspect-oriented programming (AOP); software evolution; dynamic weaving