方面编织策略研究进展及图形化方面编织框架

王 斌,周 亮,桂卫华
WANG Bin,ZHOU Liang,GUI Wei-hua

中南大学 信息科学与工程学院,长沙 410083
College of Information Science & Engineering,Central South University,Changsha 410083,China
E-mail:wb_csut@mail.csu.edu.cn

WANG Bin,ZHOU Liang,GUI Wei-hua.Research & development of aspect weaving strategy and graphic aspect weaving framework.Computer Engineering and Applications,2007,43(33):92-97.

Abstract: At present,as an extension and complementary of OOP(Object Oriented Programming),AOP(Aspect Oriented Programming) becomes the research hot topic of software engineering.The main idea of AOP is separation of concerns and modularization of these cross-cut concerns.One of the key technologies of implementing this goal in AOP systems is weaving aspect technology. This paper explains the concrete aspect weaving strategies,including aspect implementing strategies and weaving time strategies. Moreover,it also presents the weaving mechanism and technology of four mainstream AOP systems:AspectJ,AspectWerkz,Spring AOP and JBoss AOP.After analyzing and summarizing the pros and cons of present weaving strategies and technology,this paper proposes a graphic aspect weaving framework based on XML——XbGAWF and its integrating pattern with other AOP platform.
Key words: aspect;weaving strategy;weaving framework;JMangler

摘 要:目前 AOP 作为 OOP 的扩展和补充,成为软件工程领域中的研究热点,其核心思想是分离关注点,实现横切关注点的模块化。实现关注点分离的关键技术之一是方面的编织。重点阐述了 AOP 系统中方面的具体编织策略,包括方面编织实现策略和织入时间策略。详细介绍了目前四种主流 AOP 系统:AspectJ、AspectWerkz、Spring AOP 以及 JBoss AOP 的编织机制和技术,通过分析和总结目前 AOP 系统方面编织策略的优缺点,提出了一种基于 XML 的图形化方面编织框架——XbGAWF,以及 XbGAWF 与不同 AOP 平台的集成模式。
关键词:方面;编织策略;编织框架;JMangler
文章编号:1002-8331(2007)33-0092-06 **文献标识码**:A **中图分类号**:TP311.52

1 前言

软件开发过程是一个不断分离、模块化关注点的过程。软件系统的关注点分为两种:核心关注点和系统关注点(横切关注点)。其中核心关注点是针对系统的核心业务、商务逻辑、基本的业务处理逻辑,而系统关注点是指横切多个核心模块的关注点,如:认证、持久性、日志记录、路径优化等。面向对象程序设计(Object Oriented Programming,OOP)已能很好地实现核心关注点,但不能较好地实现系统关注点。这是由于系统关注点横切了多个核心关注点,是多维的,而 OOP 将系统看作一些合作对象的集合,其实现方法却是一维的[1,2],即 OOP 是把多维的需求映射到一维的核心模块实现中,这样会带来两类问题:(1)代码交织(Code Tangling):使用某一个方法或类来实现多个关注点;(2)代码扩散(Code Scattering):实现某个关注点的代码扩散到实现其他关注点的类中。

代码交织和扩散破坏了模块的封装,为软件设计人员带来了诸多麻烦。要解决这个问题,需要分离关注点。关注点分离是软件工程中的一个基本原则,它是指鉴别、封装、处理软件当中与特定概念或目标相关部分的能力。通过分离核心和系统关注点,软件开发人员可以相对独立地开发软件模块,降低系统模块间的耦合度,提高模块内部的内聚性,使得模块的可重用性、稳定性、易理解性、系统架构的可演化性等都有着很大的改善。目前有许多分离关注点的技术,包括面向方面程序设计(Aspect Oriented Programming,AOP)[3]、基于容器平台(Container Based Platforms,CBP)[4]、模型驱动工程(Model Driven Engineering,MDE)[5]、元对象协议(Meta-Object Protocols,MOP)[6,7]、自适应程序设计(Adaptive Programming,AP)[8]等。

AOP 是目前分离关注点技术的研究热点,本文在第 2 章和第 3 章中重点讨论 AOP 中方面编织策略和相关实现技术。AOP 分离关注点的技术的核心思想是引入模块化的系统关注点——方面(Aspect),在编译时、加载时或运行时静态或动态地将方面织入系统。目前有许多基于 Java 的 AOP 系统,其中较为成熟的主要有 AspectJ[9]、AspectWerkz[10,11]、Spring AOP[12]、

基金项目:湖南省自然科学基金(the Natural Science Foundation of Hunan Province of China under Grant No.05JJ40132);中南大学理科发展基金;中南大学博士后科学基金。
作者简介:王斌(1973-),男,博士,副教授,主要研究领域为构件组装;周亮(1983-),男,硕士研究生,主要研究领域为面向方面软件工程、方面编织;桂卫华(1950-),男,教授,博导,主要研究领域为工业过程控制、鲁棒控制。

JBoss AOP[13,14]等,由于上述系统方面编织实现方式的不同,决定了这些系统的灵活性以及横切粒度的不同。本文分析比较这些开发平台或语言的方面编织机制,从方面编织策略上对AOP系统进行了分类。同时,针对目前AOP开发平台难于手工编写方面代码的缺陷,并在综合比较各AOP系统方面编织能力后,提出了一种建立在JMangler[15]之上的基于XML的图形化方面编织框架——XbGAWF(XML based Graphic Aspect Weaving Framework)。

2　AOP的方面编织策略

方面的编织策略决定AOP开发平台的性能和特征,是AOP开发平台的核心支撑技术。方面编织策略可分为方面实现策略和方面织入时间策略。方面实现策略是指如何实现方面的行为,而方面织入时间策略是指方面行为何时被织入目标系统中。

2.1　方面实现策略

实现方面的织入和运行,关键是实现方法引用的插入。程序流执行到某个连接点(Join Point)时,系统应识别这个连接点,并在该连接点插入相应的横切行为。目前有许多策略实现这一插入,共分为四种:类包裹器、类替代、类修改以及解释器修改[16]。这四种实现策略阐述如下:

(1)类包裹器(Class Wrapper):类包裹器实现的思想是在客户端(调用类)和服务端(被调用类)之间插入一个中间类。类包裹器策略的实现方式有代理模式(Proxy)和继承(Inheritance)。类包裹器策略存在两个问题:自身包裹问题,反包裹(超类被修改时)和封装问题。

(2)类替代(Class Substitution):这种策略下原始类由新类代替,客户端无须添加其他特殊代码。与类包裹器相比,类替代与原始类没有直接关系。新类不会继承原始类,也不会引用原始类。类替代的实现方式有多种:改变类路径或创建自定义的类加载器。类替代的缺点是新类要实现原始类的所有方法。

(3)类修改(Class Modification):类修改直接修改被调用类。在代码生命周期的不同阶段通过修改源代码或字节代码实现类修改。由于不需要添加额外的类,也不需要使用动态的方法调用,这种策略具有较好的执行效率。

(4)解释器修改(Interpreter Modification):这种策略不需要修改应用程序而只需修改解释器。例如:利用JVMDI(Java Virtual Machine Debugger Interface)实现方面编织[17]。程序执行到某一断点时,JVMDI通知调试器,这时可在此断点上插入附加行为。这种方式的缺点是程序必须在调试模式下运行。解释器修改的另一种实现方式是修改虚拟机使之具备插入方法调用的能力[18]。这种方式也会产生一些问题:由于要创建特殊的虚拟机,这种策略与标准JVM规格说明的演化相关联,应用程序也只能限制在特定虚拟机之内。

2.2　方面织入时间策略

方面织入时间策略分为三类:编译时织入、加载时织入以及运行时织入,这三类策略阐述:

(1)编译时编织(Compile-Time Weaving):编译时编织策略可分为编译前策略和编译后策略。编译前策略通过修改源代码实现方面织入,它需要获得应用程序的源代码。编译后策略则是在字节代码级别上对类文件(.class文件)进行修改。目前有许多Java类库提供对字节代码的操作,例如:JOIE(Java Object Instrumentation Environment)[19],BCEL(ByteCode Engineering Libary)[20]和Javassist(Java Programming assist)[21]。编译时织入策略具有较高的执行效率,但前提是会牺牲较多的编译时间。

(2)加载时编织(Load-Time Weaving):这种方式无须修改任何文件,而是当类进行加载时在字节流层次上对方法调用进行修改以插入横切行为。实现这种策略的一种方法是创建自定义类加载器,它主要用来修改字节流而不是类文件,从而无需修改应用程序的类文件,但是它又产生了两个问题:首先,由于每次在运行应用程序的时候都需要修改类,执行效率较低;其次,如果目标应用程序同样需要一种自定义类加载器,这种技术则不能凑效。加载时编织的另外一种方法是修改类路径,使虚拟机能够最先获得新类。这种方法需要一个完全实现所有方法的替代类。

(3)运行时编织(Run-Time Weaving):对于运行时编织,类已经被加载至虚拟机。为实现横切行为,可采用的技术有:元对象协议、解释器修改、反射和动态代理、JIT(Just In Time)技术等。

2.3　方面的静态编织和动态编织

方面编织的不同策略决定了AOP系统所支持的方面编织类型。方面编织类型可分为静态编织和动态编织两种。静态编织是指通过编译时在连接点插入方面代码以修改源文件实现方面织入的编织方式。静态编织方式下方面内嵌(Inline)于源代码之中,具有较高的执行效率。其缺陷是方面一旦织入就不能修改,缺乏对方面的热部署能力。动态编织是指一种允许方面在程序运行过程中动态部署的编织方式。动态编织方式下方面可以由开发人员动态加载、卸载及重组[22]。表1阐述了方面静态编织、动态编织与方面实现策略、方面织入时间的关系。

3　四种AOP系统的编织机制

为了更好地研究方面编织技术,下面选取四种基于Java的AOP系统:AspectJ、AspectWerkz、Spring AOP、JBoss AOP,探讨它们的方面编织技术。

3.1　AspectJ方面编织机制

实现横切关注点关键是能够在程序流执行过程中发现已定义好的连结点区域,并在该区域插入相应的横切行为。因此,方面编织的主要工作集中在程序中切入点(Pointcut)的匹配以及通知(Advice)的插入。

AspectJ提供了一个编译器AJC(AspcetJ Compiler),它能

表1　方面静态编织、动态编织与方面编织策略关系表

织入时间策略	采用技术支持类型 实现策略			
	类包裹器	类替代	类修改	解释器修改
编译时编织	利用静态代理支持静态编织	支持静态编织	利用自定义编译器支持静态编织	不支持
加载时编织	利用静态、动态代理支持动态编织	利用自定义类加载器、修改类路径支持动态编织	利用自定义类加载器支持动态编织	不支持
运行时编织	利用动态代理支持动态编织	不支持	利用元对象支持动态编织	利用修改虚拟机支持动态编织

识别 AspectJ 字节代码和源代码,并以纯 Java 字节代码作为输出。因此,编译后的 AspectJ 程序如同普通的 Java 程序。具体而言,AspectJ 编译器分为两个阶段,即前端(Front-end)和后端(Back-end)。前端将 AspectJ 和纯 Java 代码编译成字节代码。由于 AspectJ 定义了一些不符合 Java 规范的结构,如:通知和切入点声明,前端的一个重要作用是将 AspectJ 中不符合 Java 规范的结构转换为符合其规范的字节代码。后端实现字节代码转换,在前端已编译的字节代码中插入对通知的调用代码,最终生成编织类文件[9]。如图 1 所示:

```
//helloWorld.java                    //goAspect.java
public class helloWorld {             public aspect goAspect {
 public static void main(String[] args){  before():execution(void go()){
   helloWorld.go();                     System.out.println("before the go function");
  System.out.println("HelloWorld,       }
   the main function");                }//goAspect
 }
 public static void go(){ }
}//helloWorld
```

图 1 AspectJ 示例

图 1 中 goAspect.java 的作用是捕获程序执行过程中返回值为 void 的 go()函数的执行,并在 go()函数执行前打印相关语句。其方面织入过程如图 2。

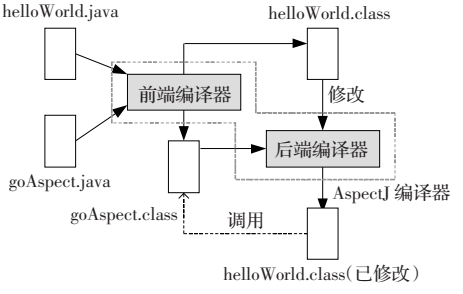

图 2 AspectJ 方面编织过程

AspectJ 方面的插入过程主要在后端编译阶段,最终生成包含方面的通知调用代码的标准 Java 字节代码,修改了程序的类文件。因此,在程序运行过程中,方面不能动态添加和卸载。这种编织方式具有较高的执行效率,但缺少对方面动态部署的支持。

3.2 AspectWerkz 方面编织机制

AspectWerkz 是基于 Java 的动态的、轻量级 AOP 框架。它通过加载时修改字节代码实现编织,能够向除基类加载器外的其它类加载器所加载的类插入钩子。AspectWerkz 框架使用松散耦合和委托机制,实现了动态方面模型。这个动态方面模型中的方面、通知和导言(Introduction)注册在系统中,通过句柄被引用。这种设计方式使得方面、通知和导言以不同的生命周期存在于不同的虚拟内存空间,同时运行时较容易添加、移除和重构方面、通知和导言,无需重新加载或编织目标类[10]。

AspectWerkz 支持动态和静态两种编织方式,通过 BCEL 在字节代码级编织 Java 类。AspectWerkz 通过改进虚拟机类加载内核,使得类加载到虚拟机前向类中织入方面。图 3 是一种基于 JMangler 的 AspectWerkz 方面编织框架。

如图 3 所示,Aspectwerkz 包裹脚本(bin/aspectwerkz)发起 JVM1(Java Virtual Machine,JVM),JVM1 触发目标应用程序以 debug 模式运行于 JVM2 之上。此时,JVM2 中的 java.lang.

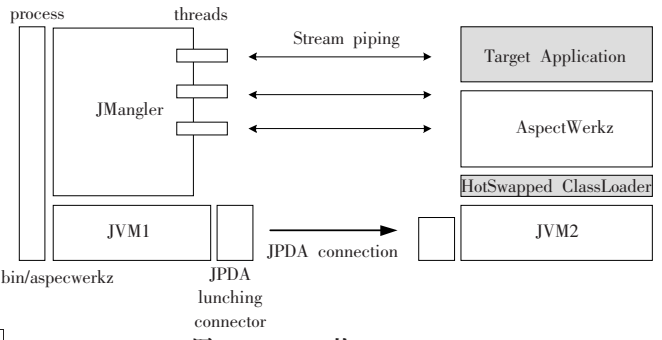

图 3 JMangler 的 AspectWerkz

ClassLoader 更改为一种可热插拔(HotSwapped)的类加载器。JVM2 通过三个后台线程——stderr、stdout、stdin 与 JVM1 通信,以保证正常运行。JVM2 成为具有编织功能的虚拟机,可以在编织过程中加载类。

目前 AspectWerkz 支持五种不同类型的加载时编织,共分为两组:热插拔(HotSwap)和基类路径(Bootclasspath)[10]。其中热插拔又分三种方式:规范热插拔(Regular HotSwap)、本地热插拔(Native HotSwap)和远程热插拔(Remote HotSwap)。规范热插拔类似上述基于 JMangler 方式,它需要两个虚拟机:运行 AspectWerkz 内核的 JVM1,运行 AspectWerkz 以及应用程序的 JVM2。JVM1 以调试方式触发 JVM2,使之具备方面编织能力,JVM2 通过三个后台线程与 JVM1 通信以保证正常运行;本地热插拔在虚拟机初始化时通过 JVMPI(JVM Profiler Interface)API 对其扩展,由一个本地 JVM 扩展机处理类加载器的替代;远程热插拔下应用程序虚拟机以调试模式运行,同时具备一个 JPDA(Java Platform Debugger Architecture)侦听连接器。此时,AspectWerkz 内核通过 JPDA 连接应用程序虚拟机,热交换类加载器并重启应用程序虚拟机。

基类路径采用"Xbootclasspath/p:"选项修改基类路径。基类路径实现加载时修改主要有两种:透明基类路径(Transparent bootclasspath)和预备基类路径(Prepared bootclasspath)。透明基类路径中 JVM1 更改 java.lang.ClassLoader 类并保存,JVM1 以"-Xbootclasspath"选项触发 JVM2,使得 JVM2 使用已更改的 java.lang.ClassLoader 类。预备基类路径中 AspectWerkz 内核修改 java.lang.ClassLoader 类并以 JAR 格式保存。虚拟机以"X-bootclasspath"选项运行应用程序,使用已修改的类加载器。

如上所述,AspectWerkz 提供了一系列编织开关。通过编译时字节代码修改,方面能够编织在普通的对象中,实现离线编织(Offline Weaving);通过加载时和运行时修改字节代码实现在线编织(Online Weaving)。运行时编织是 AspectWerkz 所提供的功能最强的可定制编织机制,开发者能在运行时添加、删除和重构通知,替换导言的实现,其缺陷是目前版本不支持运行时添加切入点。

3.3 Spring AOP 方面编织机制

Spring 是一个非入侵式轻量级 J2EE 框架,目的是最小化组件对框架的依赖程度。其核心思想是控制反转 IoC(Invert of Control)和动态代理。其中,控制反转可分为两类:依赖注入(Dependency Injection)和依赖查询(Dependency Lookup)。

Spring AOP 基于动态代理模式实现。利用反射和动态代理,在方法调用前后插入方面处理代码。Spring AOP 代理有两种方式:基于 JDK 动态代理和基于 CGLib(Code Generation Li-

brary)[23]代理。Spring AOP 默认使用基于 JDK 动态代理模式实现,从而任何接口都可被代理。CGLib 与动态代理的代理机制类似,只是其动态生成的代理对象并非是某个接口的实现,而是针对目标类扩展的子类。两者区别如图 4 所示。

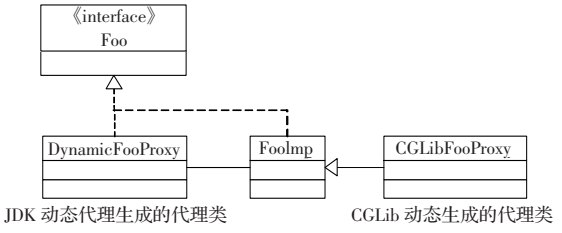

图 4　JDK 动态代理与 CGLib 动态代理比较

Spring AOP 获得代理对象后,原对象的调用首先分派给调用处理器,通过反射,应用程序可在调用处理器的 invoke() 方法中截获原对象的方法调用,从而为方面的插入提供了可能,其调用执行过程如图 5 所示。

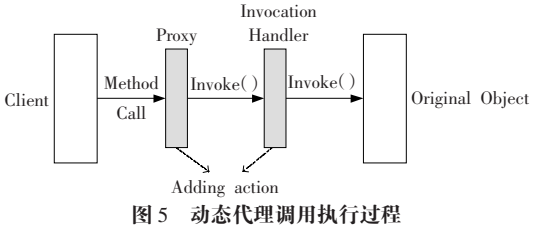

图 5　动态代理调用执行过程

具体来说,java.lang.reflect.Proxy.newProxyInstance 方法依据传入的接口类型动态构造代理类实例返回,该代理类在 JVM 内部动态构造,实现了传入接口列表包含的接口。Invocation-Handler.invoke 方法在被代理类的方法调用之前触发,因此,被代理类方法调用前后既可插入额外处理代码,也可通过 Method.invoke 方法调用被代理类的原始方法,为方面的实现提供了基础。以 Spring 事务管理机制为例,只需通过动态代理加载所有需要事务管理的 Bean,根据配置文件,在 invoke 方法中判定当前调用的方法名,为其加上相应事务管理代码,从而以 AOP 方式实现事务管理。

如上所述,Spring 优势在于它的非入侵轻量级框架,其核心技术是控制反转/依赖注入以及基于动态代理的 AOP。但是,由于采用动态代理,Spring 支持的方面粒度较大,目前仅支持方法调用这一种连接点类型,支持 Before,After returning,Around 通知,Throws 异常引入和导言。同样由于动态代理,Spring AOP 不能编织 final 类型的类。

3.4　JBoss AOP 方面编织机制

JBoss 是一个可扩展的,基于反射的,能够动态重配置的 Java 应用程序服务器。JBoss 是一个开放的中间件平台,用户可以通过动态部署新组件到运行服务器扩展中间件服务,目前还没有其他应用服务器能提供如此程度的扩展性。

由于 Java 利用元数据来存储有关类型、方法、字段的相关信息,因此,可以通过反射获得模块相关的元数据,对方法进行拦截,并将被拦截的方法与 aspect 逻辑关联。JBoss AOP 框架正是借助这种思想。JBoss AOP 同样需要 Interceptor 拦截器实现方法拦截,所有 Interceptor 实现 org.jboss.aop.Interceptor 接口,该接口最重要的方法是 invoke()。invoke 方法利用反射拦截调用方法的消息直接操作元数据,Interceptor 相当于 AOP 的 Advice,而 Pointcut 则在 XML 配置文件中配置。

JBoss AOP 通过加载时使用 Javassist 修改字节代码,实现动态 AOP。其过程如图 6。

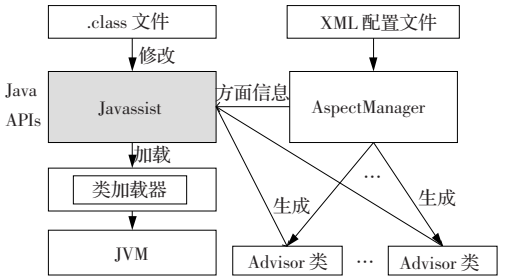

图 6　JBoss AOP 加载时方面编织过程

图 6 中 XML 配置文件中关于切入点、拦截器、元数据以及混合类的信息在应用程序部署时读入、解析,并生成相应的对象,这些信息与实例化的对象都由 AspectManager 管理。当需要织入方面的类加载时,Javassist 依据方面相关信息以及由 AspectManager 创建的 Advisor 类,修改类的字节代码,最后将已修改的字节代码交给类加载器完成类的装载。

综上所述,以上四种 AOP 系统的方面编织策略各有不同,决定了它们所支持的方面类型。如表 2 所示:

表 2　四种 AOP 系统方面编织策略及支持方面类型

	AspectJ	AspectWerkz	Spring AOP	JBoss AOP
实现策略	自定义编译器	自定义类加载器	动态代理	元对象自定义类加载器
织入时间	编译时编织	加载时编织	加载时编织	加载时编织
支持类型	支持静态编织	支持动态编织	支持动态编织	支持动态编织

4　基于 XML 的图形化方面编织框架——XbGAWF

面向方面软件开发中,系统关注点的横切工作由编织器解析织入方面完成,方面决定了系统横切关注点的模块化程度,关系着系统性能和软件质量。然而,现有 AOP 开发平台或语言中,方面的编写需要了解整个系统细节,如:模块函数名称、类的域等,这些细粒度的方面织入给软件开发带来了很大障碍。特别是系统引入第三方控件而需要对该控件插入细粒度方面(如对某个类的域进行处理)时,由于不了解控件内部结构,软件开发人员对此无能为力,因此需要一个图形化的方面编织框架解决手工编写方面代码带来的诸多问题。基于 XML 描述方面已经成为众多 AOP 开发平台的首选技术,为兼容其它方面编织器和 AOP 开发平台,图形化方面编织框架基于 XML 描述方面。

利用对 XML 描述方面的实践和研究工作有许多,其中较为成功的是 XWeaver[24]。XWeaver 是针对 Java 和 C++语言的、独立的 AOP 技术。XWeaver 将方面用 XML 来建模,同时将目标代码也用 XML 建模,通过将 XML 文件进行编织,生成编织后的 XML 模型,然后,将其转换成为 Java 或者 C++编写的代码。在上述过程中,XWeaver 需要将目标代码用 XML 描述,这个过程对于真正的软件开发是难以接受的。同时,上述过程是源代码级别的静态编织过程,而不是针对目标代码(.class,.dll)的动态编织过程,这样使得该技术的应用受到制约。同时,考虑到目前存在多种成熟的,并在软件工程中被广泛使用的面向方面开发平台,如:AspectJ,Spring AOP,JBoss AOP 等,基于 XML 的图形化方面编织框架 XbGAWF(XML based Graphic Aspect Weaving Framework,XbGAWF)应该能够被灵活地集成到上述开发平台中,而 XWeaver 不提供集成到上述平台的能力。

4.1 加载时编织技术选择

为了使 XbGAWF 能够支持动态 AOP,XbGAWF 选取加载时编织策略,通过扩展类加载器在类加载时对其修改以插入方面。这需要加载时修改类文件工具的支持,目前较为成熟的加载时修改类文件的工具主要有 BCA (Binary Component Adaptation)[25]、JOIE、Javassist 以及 JMangler。图 7 为上述加载时类文件修改工具的区别示意图。

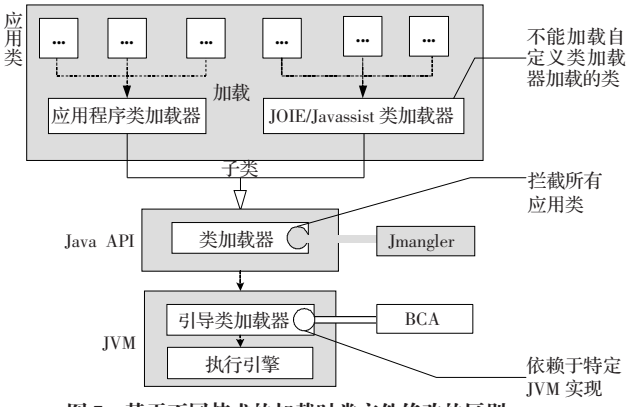

图 7 基于不同技术的加载时类文件修改的区别

其中,BCA 是一个二进制组件自适应编织器,它是第一个允许加载时对已编译的 Java 类修改的编织器。这些修改规约由类 Java 语言书写并保存于 Detal 文件,通过引入编译后的 Detal 文件参数化解释器,修改目标类文件。不足的是,BCA 作为 JDK1.1 在 Solaris 上的 JVM 的定制版,不能应用于其它 JVM,因此它是依赖 JVM 的。

JOIE 是一个加载时类文件转换框架,通过定制自定义类加载器,JOIE 为每个类创建 ClassInfo 对象,并将这些对象传递给已注册的转换器,转换结束后,类加载器再将 ClassInfo 转换成最终的类文件字节数组提交给 JVM 验证。JOIE 提供了较为安全的转换框架,转换过程中需要自定义类加载器,但不能转换已具有自定义类加载器的应用程序。

Javassist 是一个加载时提供结构化反射的类库,不同于 BCA 和 JOIE,Javassist 集中于元对象协议的设计,将加载时的适应性作为实现重点,它同样需要加载时修改类文件。其实现方式类似于 JOIE,通过特殊类加载器和基于对象表示的类文件来修改类文件,因此它也不能转换已具有自定义类加载器的应用程序。

JMangler 也是一个加载时类文件转换框架,通过修改 java.lang.ClassLoader 中 defineClass()方法实现通用拦截机制。与上述技术不同,JMangler 是一个开放的架构,独立于 JVM,支持 JDK1.4,同时也能修改具有自定义类加载器的应用程序。在此基础之上,开发人员可以自定义转换器,包括代码转换器和接口转换器,扩展功能。虽然 JMangler 不能修改系统类文件,但对于一般方面编织器而言是可以忽略的。

通过上述分析,XbGAWF 采用 JMangler 作为类加载时的通用插入实现基础,由于 JMangler 最新版本已包含一个功能较为完整的 BCEL 编织器,XbGAWF 采用 BCEL 字节代码转换框架对其进行改进和扩展。图 8 为 XbGAWF 框架。

如图 8 所示,JMangler 为类文件转换提供了一个基础钩子平台,而 BCEL 字节代码转换库支持字节代码级的修改与转换。XMLAspect 是基于 XML 的为 AOP 的通用概念——方面,通知,切点等建模的语言,其详细设计规范将在另文中阐述。XbGAWF 具备如下功能:图形化视图(Graphic View)利用 BCEL,解析.class 文件,映射出整个项目的依赖关系以及类的基本结构;方面化 UML(Aspectual UML)生成项目的 UML 类关系图,支持方面的 UML 表示;方面编辑器(Aspect Editor)提供图形化的方面编辑界面,由系统统一生成 XML 方面表示文件;编织器(XML based Graphic Aspect Weaver,XbGAW)依据 XMLAspect 方面描述文件,动态织入方面;方面管理器(Aspect Manager)负责管理方面信息,提供动态部署方面功能。

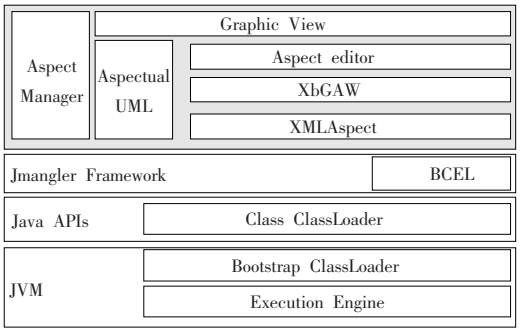

图 8 基于 XML 的图形化方面编织框架

4.2 XbGAWF 与不同 AOP 平台的集成模式

XbGAWF 具有广泛的工程应用能力,在 XMLAspect 的基础之上,XbGAWF 可以以 XMLAspect 语言的 Schema 为核心,为不同 AOP 支持平台扩展 XMLAspect。例如:针对 Spring Framework,可以生成 SpringXMLAspect,针对 JBoss 可以生成 JBossXMLAspect 等。图 9 给出了 XbGAWF 集成到主流 AOP 开发平台中的集成模式示意图。

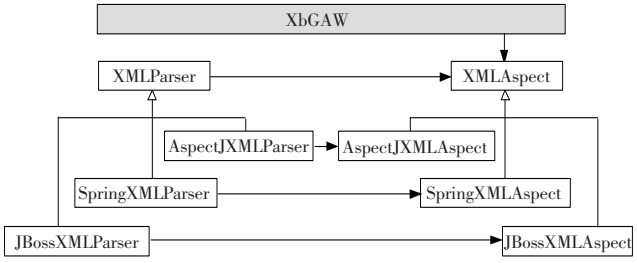

图 9 XbGAWF 与不同 AOP 平台的集成模式

图 9 中采用了抽象工厂模式。XMLParser 是一个接口,针对不同 AOP 平台的 XMLAspect 可实现不同的 Parser。例如:针对 SpringXMLAspect 有 SpringXMLParser 等。

以 XMLAspect 为基础的 XbGAWF 和主流 AOP 平台的集成开发有机的结合在一起,具有如下技术优势:(1)XMLAspect 与XbGAWF 可以成为独立的 AOP 开发技术,并可成为支持按需计算的中间件平台的支撑技术之一;(2)XMLAspect 与 XbGAWF 可以集成目前存在的 AOP 平台中,利用 AOP 平台中已存在的对方面支持能力,使之具有更广泛的应用。

5 总结及展望

本文阐述了 AOP 系统中方面的编织策略,从方面编织实现策略和织入时间策略对 AOP 系统进行了分类。详细介绍了目前 4 种主流 AOP 系统:AspectJ、AspectWerkz、Spring AOP、JBoss AOP 的方面编织机制和技术。通过分析和比较编译时、加载时、运行时方面编织策略,本文提出了一种建立在 JMangler 之上的基于 XML 的图形化方面编织框架—XbGAWF,并给出了该框架与不同 AOP 平台的集成模式。

在上述研究的基础上,本文将对下列内容做进一步研究:(1)方面编织器的实现机制研究:包括 BCEL 字节代码库的分析和应用,连接点查找机制,切入点匹配问题,分布式切入点查找机制等;(2)XMLAspect 语言规范描述:包括如何利用 XML 表示方面、切入点、通知、导言,对 XMLAspect 语言的解析与验证,优化与各 AOP 平台的集成模式等;(3)方面管理环境的研究和开发。扩展 UML 以支持方面的表示,以图形化方式管理方面信息,寻找适合大型软件开发中的方面管理和使用策略。

(收稿日期:2007 年 3 月)

参考文献:

[1] Ossher H,Tarr P L.Using multi-dimensional separation of concerns to reshape evolving software[J].Communications of the ACM,2001,44(10):43-50.

[2] Tarr PL,Ossher H,William H,et al.Degrees of separation:multi-dimensional separation of concerns [C]//International Conference on Software Engineering 1999(ICSE 1999),Las Angeles,1999:107-119.

[3] Kiczales G,Lamping J,Mendhekar A,et al.Aspect-oriented programming[C]//Proceedings of the European Conference on Object-Oriented Programming(ECOOP),Jyväskylä,Finland,1997.

[4] Duclos F,Estublier J,Morat P.Describing and using non functional aspects in component based applications[C]//Proceeding of the 1st International Conference on Aspect-Oriented Software Development(AOSD 2002),Enschede,2002:65-75.

[5] Kent S.Model driven engineering [C]//Proceedings of the 3rd Integrated Formal Methods(IFM 2002),Turku,Finland,2002.

[6] Kleinöder J,Golm M.MetaJava:an efficient run-time meta architecture for JavaTM[C]//Proceedings of the 5th International Workshop on Object-Orientation in Operating Systems(IWOOOS 1996),Seattle,WA,USA,1996:54-61.

[7] Zang Haiyun,Feng Zhiyong.Aspect-oriented programming using reflection and meta-object protocol[J].Computer Applications,2004,24(4):34-37.

[8] Gouda M,Herman T.Adaptive programming[C]//IEEE Transactions on Software Engineering(TSE 1991),1991,17(9):911-921.

[9] Hilsdale E,Hugunin J.Advice weaving in aspectJ[C]//The 3rd International Conference on Aspect-Oriented Software Development(AOSD 2004),Lancaster,UK,2004:26-35.

[10] Bonér J.Aspectwerkz-dynamic AOP for java[C]//Proceeding of the 3rd International Conference on Aspect-Oriented Software Development(AOSD 2004),Lancaster,UK,2004.

[11] Vasseur A.Dynamic AOP and runtime weaving for Java how does aspectWerkz address It?[C]//Proceedings of the 2004 Dynamic Aspects Workshop(DAW 2004),Lancaster,UK,2004.

[12] Johnson R,Hoeller J,Arendsen A.The spring framework-reference documentation:aspect oriented programming with spring[EB/OL].[2006].http://www.springframework.org/docs/reference/aop.html.

[13] JBoss AOP.JBoss AOP-Aspect-Oriented Framework for Java[EB/OL].[2006].http://labs.jboss.com/portal/jbossaop/docs/1.5.0.GA/docs/aspect-framework/reference/en/html/index.html.

[14] Fleury M,Reverbel F.The JBoss extensible server[C]//Proceedings of Middleware 2003,Rio de Janeiro,Brazil,2003:344-373.

[15] Kniesel G,Costanza P.JMangler-a-framework for load-time transformation of Java class files[C]//Proceedings of the 1st IEEE International Workshop Source Code Analysis and Manipulation(SCAM 2001),Florence,Italy,2001:98-108.

[16] Frédéric D,Jacky E,Rémy S.Separation of concerns and the extended object machine[J].ADVICE:Journal of Aspect Orientation,2004,1(1).

[17] Popovici A,Gross T,Alonso G.Dynamic weaving for aspect-oriented programming[C]//Proceedings of the 1st International Conference on Aspect-Oriented Software Development Enschede,The Netherlands,2002:141-147.

[18] Nicoar A,Alonso G.Dynamic AOP with PROSE[C]//Proceedings of 1st International Workshop on Adaptive and Self-Managing Enterprise Applications(ASMEA2005),Porto,Portugal,2005.

[19] Geoff A C,Jeffrey S C.Automatic program transformation with JOIE[C]//Proceedings of the 1998 USENIX Annual Technical Conference,New Orleans,Louisiana,1998.

[20] Apache Software Foundation.The Byte Code Engineering Library[EB/OL].[2006].http://jakarta.apache.org/bcel/.

[21] Chiba S.Javassist-a reflection-based programming wizard for Java[C]//OOPSLA 1998 Workshop on Reflective Programming in C++ and Java,Vancouver,Canada,Oct 1998.

[22] Bollert K.On weaving aspects[C]//Proceedings of the European Conference on Object-Oriented Programming(ECOOP1999)workshop on Aspect-Oriented Programming,Lisbon,Portugal,1999.

[23] Code Generation Library.CGLib[EB/OL].[2004].http://cglib.sourceforge.net.

[24] Weaver X.The XWeaver Project[EB/OL].[2005-06].http://www.xweaver.org/Home.html.

[25] Keller R,Hölzle U.Binary component adaptation[C]//Proceedings of European Conference on Object-Oriented Programming(ECOOP1998),Brussels,Belgium,1998.

(上接 47 页)

[4] Tarski A.A lattice theoretical fixpoint theorem and its applications [J].Pacific J Math,1955,5:285-309.

[5] Kozen D.On Kleene algebras and closed semirings[C]//Rovan.Lecture Notes in Computer Science,MFCS'90,Berlin:Springer,1990,452:26-47.

[6] Leiβ H.Kleene modules and linear languages [J].The Journal of Logic and Algebraic Programming,2006,66:185-194.

[7] Pastijn F,郭聿琦.幂等元半环簇的格[J].中国科学:A 辑,1999,29(5):391-407.