面向方面编程技术及应用研究

黄 天 开
(柳州师范高等专科学校 数学与计算机科学系,广西 柳州 545004)

摘 要:AOP的目标是通过把问题分解成一系列功能模块和一系列贯穿多个功能模块的方面,然后再通过工具自动化的重新组织这些模块和方面,以获得系统的实现。本文主要介绍了 AOP的编程思想及其技术特点,并结合 Java给出在日志和并发访问中的应用。

关键词:面向对象;AOP;横切关注点;通知;方面
中图分类号:TP311.1 **文献标识码**:A **文章编号**:1003-7020(2006)03-0104-04

1 AOP的产生背景

计算机编程技术的发展经历了面向过程、面向对象和面向方面。面向对象将问题领域中的“名词”转化为软件系统中的“对象”,抽象出对象的类型(类),通过类的继承体系实现代码的重用[1]。面向对象技术也因此获得了巨大的成功。但随着软件设计的进一步深入,人们逐渐发现对象的抽象并不能完全解决问题。因为问题域并不是全部由“名词”组成,而是和特定的应用相关联,如系统日志之类的服务。在软件系统的设计过程中,人们除了关注与系统业务相关的模块,如银行系统中账户的存取模块、库存管理系统中的货物购置和入库模块,也要考虑分布在多个核心模块中的公共行为,比如日志记录、安全性、缓存和权限控制等。人们把前者称为核心关注点,后者称为横切关注点。面向对象技术可以很好地完成对核心关注点的设计与开发,而对横切关注点却有些力不从心,其原因在于,横切关注点会跨越多个模块,是多维的,而面向对象技术的设计方法却是一维的,把多维的需求映射到一维上,便产生了许多需要探讨的问题。

面向方面的编程技术 AOP(Aspect-Oriented Programming)就是为了解决横切关注点而诞生的,它由施乐公司帕洛阿尔托研究中心(Xerox Palo Alto Research Center)的 Gregor Kiczales等在 1997年提出,并开发了第一个 AOP开发环境 AspectJ。AOP为开发者提供了一种描述横切关注点的方法,人们可以通过它单独实现横切模块。此外,AOP提供了一种机制,使得核心模块和横切模块能够融合在一起,从而构造出最后的实际系统。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。使用 AOP技术横切软件开发,可以将需要出现多次的公用代码集中到一处实现,从而大大减少代码的冗余度和耦合度,增强可读性。

2 AOP的编程语言

2.1 Java的 AOP实现

和其它编程模式一样,AOP面向方面编程思想的实现取决于一个良好的语言环境,而 AspectJ是当前实现 AOP编程的基础,它是在 Java语言基础上,扩展了若干语言元素,加入了一些植入规则而形成的。它还允许为指定的类引入新的方法和属性以及重新指定父类型。AspectJ的编织器把方面和类编译在一起,成为纯 Java字节码,并且提供了可视化的开发工具,同时给流行的 IDE(如 Jbuilder、Eclipse、Forte等)提供插件。

2.2 AspectJ语言基础

AspectJ在 Java语言中加入了如下语言元素:(1)连接点(Join Point)。它是在程序执行过程中的明确定义的点,它可以定义在方法调用、条件检测或是赋值动作处。连接点有一个与之相关联的上下文。(2)切入点(Pointcut)。它是用来指明所需连接点的语言元素。切入点可能包括一系列的连接点,同时它还可以为在连接点上执行的通知提供上下文[2-4]。例如,pointcut callSetter(): call(public void HelloWorld.Set*(..))。其中,pointcut说明其后声明的是一个命名的切入点 callSetter(),后面的空括号表示该切入点不需要上下文信息。call表示该切点捕获的是对指定方法的调用。以上指定的方法是在类 helloWorld中声明的公有的、返回值为空、以 set开头、拥有任意参数的方法。其中,* 和.. 都是

收稿日期:2006-05-08
作者简介:黄天开(1972-),男,广西宾阳人,讲师。

通配符。(3)通知(Advice)。指定到达特定切入点处应执行的代码[2]4。AspectJ提供了3种把通知关联到连接点的方式:before、after及around。before和after分别表示通知在连接点的前面或者后面运行,around则表示通知在连接点的外面运行,并可以决定是否运行此连接点。(4)类型间声明(Inter-type declaration)。AspectJ的类型间声明指的是那些跨越类和它们的等级结构的声明。AspectJ可以在方面中声明在其他的类中加入数据或函数成员,或者声明一个已有的类继承特定接口或类。如果要在指定的多个类中加入相同的多个数据和方法,比较好的做法就是把这多个数据和方法在一个接口或类中声明,然后声明这多个类都继承于该接口或类。(5)方面(Aspect)。方面是AspectJ的模块单元,它包装所有的切点、通知和类型间声明。方面和类非常相似,它们也可以被定义为抽象的,被子方面继承。根据以上的介绍,一个连接点是程序流中指定的一点。切点收集特定的连接点集合和在这些点中的值。一个通知是当一个连接点到达时执行的代码,这些都是AspectJ的动态部分。AspectJ中也有许多不同种类的类型间声明,这就允许程序员修改程序的静态结构、名称、类的成员以及类之间的关系。AspectJ中的方面是横切关注点的模块单元。它们的行为与Java语言中的类很像,但是方面还封装了切点、通知以及类型间声明。

3 AOP的开发步骤[3]

AOP的开发依次分为3个阶段:

(1)*方面分解*。分析系统需求,提取出模块级的核心关注点和系统级的横切关注点。例如,在一个银行信息系统中,帐户的存取模块就是核心关注点,而权限验证、日志记录是横切关注点。

(2)*关注点实现*。各自独立的实现这些关注点。沿用上面的例子,需要实现帐户存取模块、权限验证模块和日志记录模块。帐户存取模块可用OOP技术来实现。其他的模块可采用AOP技术,首先定义捕获相应方法的切点,分别定义该切点的around、before、after通知,在around通知中验证权限,只有验证通过才运行该连接点,在before和after通知中就输出日志记录。

(3)*方面的重新组合*。编织器通过创建一个模块单元(aspect)来指定重组的规则。重组的过程也叫植入或集成,就是将通过方面实现的模块单元植入通过基于OOP实现的基础代码,以构建最终系统。在上面的例子中,编织器帮把权限验证语句插入到相应方法调用的外围,把日志输出语句插入到相应方法调用的前面和后面。

4 AOP的实现

4.1 AOP实现机理

使用AOP技术的面向对象的软件系统由类和方面组成。其中,类实现核心关注点,方面语法将会描述横切关注点。到目前为止,方面语法还不能由编译器识别,因此,在用编译器生成可运行程序之前,必须先由某种工具将方面与类合并,合并的过程被称为编织,而用来编织的工具被称为方面编织器。编织主要分为两种:静态编织和动态编织。利用静态编织技术,编织器将方面代码嵌入到类代码中,然后再编译成class文件执行。利用动态编织技术,编织器无需改变类文件的源代码,编织器利用反射技术,在运行时编织方面代码。

4.2 两种编译器

采用静态编织技术会改变类的源代码,动态编织技术则不会。静态编织意味着通过在连接点插入特定的方面语句来修改类的源代码。例如,要编织一个持久性的方面,就是在每个实例变量的值发生更改时插入一段数据库更新语句。AspectJ是当前最流行的AOP语言,它就是一个静态编织语言。它是对Java语言的扩展,使用几个新的结构提供了模块化横切关注点的合成机制。Java编译器不能解析AspactJ的方面语法,因此它自带了一个编织器。AspectJ的编织器是这样工作的:它先解析文件,为文件建立一棵静态语法树,然后在原来的连接点的代码体的适当位置嵌入方面代码中相应于这个连接点的代码,接着用这个嵌入后产生的代码代替语法树上原来的连接点的代码体,这样就实现了方面代码和类的编织[3]。因为静态编织后的代码被高度优化,因此执行速度与没有使用方面的相差无几,对性能没有影响。

AOP/ST是一个动态编织器,它给VisualWorks/Smalltalk扩充了面向方面编程的内容。它编织方面时没有修改类的源码,而是利用Smalltalk的反射接口实现编织。如果某个类行为需要调整,它会创建该类的子类,重载需要改变的方法,然后把该类的实例声明为子类的实例,当查找该类的函数时,就在编织后的类中查找。并且,当将来有该类的对象被创建时,同样的事情也会发生。而动态编织由于引入了额外的抽象层,因此性能会有一定降低。

静态编织需要在设计时就确定所有的方面,而动态编织可以在运行时增加、修改、删除方面。静态编织使得在编织代码中推迟指定方面语句变得困难。相应的,在运行中动态的指定或者修改方面变得费时或者根本不可能,即使使用的程序语言支持用反射接口手动控制程序。虽然大多数方面不需要这种灵活性,仍然有方面将因此而获利。例如,一个装载平衡的方面可以根据当前服务器的负载,用更好的装载策略取代以前的编织。

5 AOP的应用

5.1 AOP在日志记录方面的应用

AOP主要应用于日志记录,性能统计,安全控制,事务处理等方面。它的主要意图就要将日志记录,性能统计,安全控制等等代码从商业逻辑代码中清楚的划分出来,我们可以把这些行为一个一个单独看作系统所要解决的问题,就是所谓的面向方面的编程。通过对这些行为的分离,我们希望可以将它们独立地配置到商业方法中,而要改变这些行为也不需要影响到商业方法代码。假设系统由一系列的 BusinessObject所完成业务逻辑功能,系统要求在每一次业务逻辑处理时要做日志记录。这里我们略去具体的业务逻辑代码。以下是我们通常的做法:

```
public interface BusinessInterface {
  public void processBusiness();
}

public class BusinessObject implements BusinessInterface {
  private Logger logger=Logger.getLogger(this.getClass().getName());
  public void processBusiness() {
    try {
      logger.info("start to processing...");
      //business logic here.
      System.out.println("here is business logic");
      logger.info("end processing...");
    } catch (Exception e) {
      logger.info("exception happends...");
      //exception handling
    }
  }
```

这里处理商业逻辑的代码和日志记录代码混合在一起,这给日后的维护带来一定的困难,并且也会造成大量的代码重复。完全相同的 log代码将出现在系统的每一个 BusinessObject中。按照 AOP的思想,我们应该把日志记录代码分离出来。要将这些代码分离就涉及到一个问题,我们必须知道商业逻辑代码何时被调用,这样我们好插入日志记录代码。在此我们可以使用动态代理,通过实现 java.Lang.reflect.InvocationHandler接口提供一个执行处理器,然后通过 java.Lang.reflect.Proxy得到一个代理对象,通过这个代理对象来执行商业方法,在商业方法被调用的同时,执行处理器会被自动调用。我们所要做的仅仅是提供一个日志处理器:

```
public class LogHandler implements InvocationHandler {
  private Logger logger=Logger.getLogger(
  this.getClass().getName());
  private Object delegate;
  public LogHandler(Object delegate) {
    this.delegate=delegate;
  }

  public Object invoke(Object proxy,
  Method method, Object args)
  throws Throwable {
    Object o=null;
    try {
      logger.info("method starts..."+method);
      o=method.invoke(delegate,args);
      logger.info("method ends..."+method);
    } catch (Exception e) {
      logger.info("Exception happends...");
      //exception handling.
    }
    return o;
  }
}
```

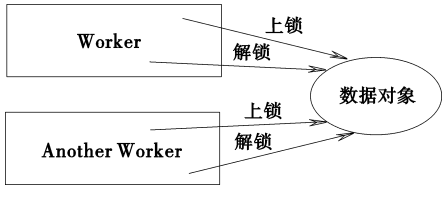
现在我们可以把 BusinessObject里面的所有日志处理代码全部去掉了

```
public class BusinessObject implements BusinessInterface {
  private Logger logger=Logger.getLogger(
  this.getClass().getName());
  public void processBusiness() {
    //business processing
    System.out.println("here is business logic");
  }
}
```

5.2 AOP在并发访问中的应用

多个访问类同时访问一个共享数据对象时,每个访问类在访问这个数据对象时,需要将数据对象上锁,访问完成后,再实行解锁,供其它并发线程访问,这是我们处理并发访问资源的方式。

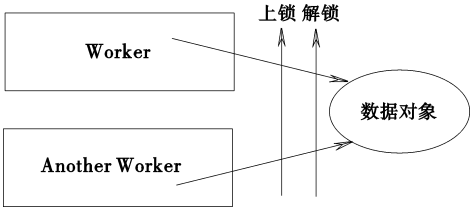
下图为传统的编程思路:



锁的实现方式是在每个具体类中实现。

使用 AOP概念来重新实现上述需求,AOP给我

们提供了观察问题的一个新视角度。解决思路如下图:



具体做法:先建立一个类似 Class的 Aspect(如下),

```
public class MyClass{
  /属性和方法…
  }
```

同样,建立一个 Aspect的代码如下:

```
public aspect MyAspect{
  /属性和方法…
  }
```

再建立一个名为 Lock的 Aspect

```
import EDU.oswego.cs.dl.util.concurrent.*;
public aspect Lock {
  ……
   ReentrantWriterPreferenceReadWriteLock rwl =
     new ReentrantWriterPreferenceReadWriteLock();
   public pointcut writeOperations():
    execution(public boolean Worker.createData()) ||
    execution(public boolean Worker.updateData()) ||
    execution(public boolean AnotherWorker.updateData());
   before(): writeOperations() {
     rwl.writeLock().acquire(); //上锁 advice body
   }
   after(): writeOperations() {
     rwl.writeLock().release(); /解锁 advice body
   }
  ……
  }
```

before代表触发之前上锁,after代表触发之后解锁。通过这样定义就可以在 Worker的实现中把锁的代码去掉,使得在 Worker中只有关于数据操作的方法。减少代码的冗余,更大程度地实现代码重用。

6 结束语

目前 AOP在系统维护、性能优化等方面得到了广泛的应用,但面向方面在很多理论的实际应用中还没有给出最佳实践的指导,对这些方法学的应用完全依赖组织的技术能力。如何在 OOP和 AOP之间寻找最佳的平衡点,并很好地结合起来,也是 AOP技术发展中面临的难题。要成为一种主流、成熟的技术,还有许多有待深入探讨的问题。

参考文献:

[1]凌晨,陈芳莉.面向方面程序设计技术[J].计算机系统应用,2006,(2):34.

[2]徐宝文,等.面向方面的程序设计:概念、实现与未来[J].计算机与数字工程,2005,(8).

[3]Ramnivas Laddad[EB/OL]. http://jit.blogbus.com/logs/2004/05/175571.html

(责任编辑:梁文杰)

A Research on the Aspect-Oriented Programming Technology and Its Applications

HUANG Tian-kai

(Department of Mathematics and Computer Science, Liuzhou Teachers College, Liuzhou, Guangxi 545004, China)

Abstract: The purpose of Aspect-Oriented Programming(AOP) is to divide the problem automatically reassembled into a series of function models and aspects crossing many function models by a reorganization of these tools the models and aspects into a whole program. This paper introduces the programme idea and the technical trait of AOP, and delivers its applications in log and parallel accessing by Java.

Key words: OOP; AOP; crosscutting concerns; advice; aspect