AOP技术及其在J2EE中的动态代理实现

曹晓利,郭顺生
(武汉理工大学,湖北 武汉 430074)

摘 要:随着软件技术的发展及需求的增加,OOP逐渐表现出其不足之处,AOP在继承OOP基础之上很好地解决了OOP所面临的困难。针对OOP编程思想的一种补充,AOP编程思想通过分离出与系统核心业务实现无关的模块,减少模块间的耦合度从而提高开发效率。AOP使得需要编写的代码量大大缩减,节省了时间,控制了开发成本。为使用AOP技术解决在软件开发中OOP难以解决的问题以及提高开发效率,利用JAVA的反射机制,研究了AOP的动态代理实现原理,说明了其可行性,体现了AOP技术应用价值和发展前景。

关键词:AOP;动态代理;横切关注点
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2008)11-0120-03

AOP Technology and Its Dynamic Proxy Implementation in J2EE

CAO Xiao-li, GUO Shun-sheng
(Wuhan University of Technology, Wuhan 430074, China)

Abstract: With the developing of software technology and the increasing of demand, there have been many disadvantages in OOP technology. As AOP inherits OOP, it can solve difficulty with which OOP is confronted. As a supplement of OOP, AOP can improve the efficiency during the software development process by isolating the modules which are not relevant with the core modules and decreasing the coupling among modules. AOP can decrease the codes, save on times and control the cost of exploitation. To use AOP to improve the efficiency and to solve the problems which OOP can not solve, makes a study of the implementation principle of AOP dynamic proxy, using JAVA reflectibility mechanism and explains its feasibility and incarnates the value on application of AOP and its foreground.

Key words: AOP; dynamic proxy; crosscutting concern

0 引 言

近年来,在面向过程编程和面向对象的编程之后,一种新的技术即AOP技术得到快速发展。OOP(Object-Oriented Programming,面向对象编程)技术解决了软件系统中角色划分的问题,通过模块化和分析系统,可以把一系列相关的问题抽象成对象,这些对象具有属性和行为。面向对象的技术可以把一个庞大的系统拆分成各个模块,解决了大型软件实现的系列问题,并且在当今技术中得到了淋漓尽致的应用。但随着OOP技术应用实践的增多,也暴露出它的问题来。

比如一个系统中有几十个或几百个数据库查询的函数,每个地方都要求把数据库查询的语句记录下来,在OOP技术中除了为每个函数增加记录日志的功能还能有什么更好的办法呢?面对这种重复劳动OOP技术束手无策。AOP技术通过方法拦截的方式解决了这个问题,在每个方法调用前后或抛出异常时记录日志信息[1]。

AOP在继承OOP基础之上很好地解决了OOP所面临的困难,实现核心关注点和横切关注点的松耦合,程序员只需要把精力放在最关心的业务逻辑上,把通用的模块放到横切关注点中,提高了代码的复用性。实现AOP技术可以采用动态代理技术也可以采用静态织入的方式。笔者将主要围绕AOP技术及其动态代理实现原理展开[2]。

1 AOP技术简介

AOP全名是Aspect-Oriented Programming,中文就是面向切面(方面)的编程。AOP的核心思想就是"将应用程序中的商业逻辑同对其提供支持的通用服务进行分离。"AOP从关注点的角度考虑问题,关注点也就是要考察或解决的问题。AOP把一个系统分解成不同的关注点,核心的业务逻辑称为核心关注点,而

收稿日期:2008-02-26
基金项目:湖北省国际科技合作重点项目(2007CA008)
作者简介:曹晓利(1983-),女,湖北人,硕士研究生,研究方向为现代通信网络理论与技术等;郭顺生,教授,研究方向为ERP/CAD/CAM、制造业信息化等。

一些分散在各个部分辅助的关注点称为横切关注点。核心关注点通过面向对象的方式来实现,横切关注点通过 AOP 的方式来实现,比如记录日志、安全验证、字段校验等横切关注点就可以通过 AOP 的方式来实现[3]。

AOP 除了可以把一些共性的功能通过拦截的方式融合在系统中外,还可以动态地为一个类添加属性和方法,增加实现的接口。

AOP 它将"关注"封装在"方面(aspect)"中。实现 AOP 的技术,主要分为两大类:一是采用动态代理技术,利用截取消息的方式,对该消息进行装饰,以取代原有对象行为的执行;二是采用静态织入的方式,引入特定的语法创建"方面",从而使得编译器可以在编译期间织入有关"方面"的代码[4]。

AOP 包括以下三个开发步骤:

(1)方面分解:分解需求提取出横切关注点和一般关注点。在这一步,核心模块级关注点和系统级的横切关注点被分离开来。就一般 Web 应用系统来说,可以分解出:核心事务处理、日志、验证等多个关注点。

(2)关注点实现:各自独立地实现这些关注点。

(3)方面的重新组合:在这一步里,方面集成器通过创建一个模块单元——方面来指定重组的规则。重组过程也叫织入或结合,使用这些信息来构建最终系统。在一般的 Web 应用系统中,每个操作的开始和结束需要记录,并且每个操作在涉及到业务逻辑之前必须通过验证[2]。

AOP 技术在 J2EE 领域中的发展最迅速,这是由于语言本身的特点来决定的,Java 语言具有反射的功能,这是大部分语言所不具备的特点。下面介绍如何利用 Java 的反射机制来实现 AOP。

2　AOP 的动态代理实现原理

GOF 介绍了 23 种经典的设计模式,其中有一种模式就是代理模式,代理对象提供了一种代理的方式来控制对原对象的访问。比如访问对象 ObjectA 时,实际访问的是代理对象 proxy,然后代理对象再把请求委托给实际对象 ObjectA,这就提供了一种实现 AOP 的思路,在把请求委托给实际对象前后可以运行其他的功能,从而实现对方法的拦截[4]。如图 1 所示。

由代理模式的序列图可以看出在对实际对象的调用前后可以加入一些特定的处理,即 AOP 中所说的切面,对实际对象的调用点对应 AOP 中的连接点[5]。

现在已经知道了这种方式的 AOP 实现原理,JDK 从 1.3 版本以来就从代码级对这种代理模式提供了支持,在 JDK 的 java.lang.reflect 库中提供了三个类:Proxy, InvocationHandler 和 Method。现在用这三个类,来实现方法调用的 AOP 的具体实例代码:

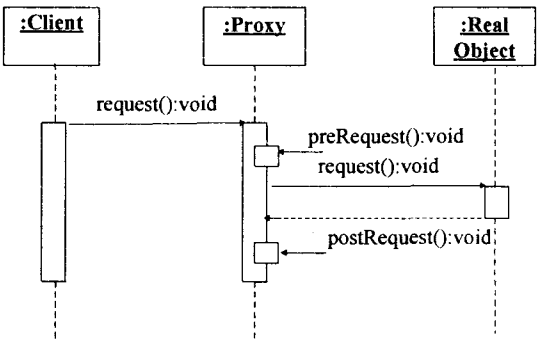

图 1　代理模式调用序列图

实际对象需要实现的接口,该接口只提供了一个方法 operateA,就以对该方法为切入点来展示 AOP 的动态实现原理。

```
package aop;
public interface IObjectA {
public void operateA ();
}
```

实现接口 IObjectA 的实际调用类:

```
package aop;
public class ObjectA implements IObjectA {
public void operateA () {
  System.out.println("Calling operateA method");
  }
}
```

下面这段代码就生成了实际对象 aop.ObjectA 的代理类 test,通过代理实现对原对象的调用。

```
Class cls = Class.forName("aop.ObjectA ");
IObjectA test = (IObjectA)cls.newInstance();
    //生成实际目标对象的实例
test = (ITest) LogHandler.createProxy(test);
    //生成代理对象的实例
test.operateA ();
//代理方法的调用,实际调用时会委托给实际对象,
  //可以在实际方法的前后加入通知
```

下面是代理类 LogHandler 的源代码,在代理中实现了对方法的拦截,在调用方法前后加入横切关注点。

```
package aop;
import java.lang.reflect.*;
public class LogHandler implements
InvocationHandler {
private Object target=null;
public static Object createProxy(Object obj){
return Proxy.newProxyInstance(
obj.getClass().getClassLoader(),
obj.getClass().getInterfaces(),
  new LogHandler(obj));
```

```
//代理的创建
}
public LogHandler (Object target){
  this.target =target;
}
public Object invoke(
Object proxy,
Method method,Object[] args)
throws Throwable {
  //对方法的拦截,通知的添加,都是在该函数中
  Object rs = null;
  try{
  System.out.println("Before calling the operateA method");
  //前置通知
  rs = method.invoke(target,args);
  //调用实际对象的方法
  }
  catch(InvocationTargetException ex)
  {
    System.out.println(ex);
    //异常通知
  }
  System.out.println("Finishing calling the operateA method);
    //后置通知
  return rs;
  }
}
```

由以上的实现代码可以看出,动态代理中的接口InvocationHandler实现是最关键的一步,InvocationHandler接口只定义了一个方法invoke方法,AOP的具体实现就体现在该方法的定义之中。切入点和通知的添加主要在method.invoke函数的前后,函数中的代码System.out.println("Before calling the operateA method");就是AOP中的前置通知;System.out.println("Finishing calling the Ope- rateA method);是后置通知;对方法method.invoke的异常捕捉就是异常通知。在实际应用中需要一个管理切入点和通知的容器,提供一个XML文件配置需要代理的类,容器根据配置文件动态实现切入点和通知的装配。

3 结束语

以上介绍了通过动态代理的方式实现AOP的原理和方法,除此之外实现AOP还有一些其他的方法,如动态字节码生成、编译期织入和类加载织入,这些都需要特殊的工具来协助实现。AOP是一种比较新的技术,可以很大程度地提高代码的复用,但AOP不可能代替OOP,它是OOP的一种补充和完善。AOP能提高效率和复用,便于维护,因此AOP的应用也会越来越广泛,现在的主流J2EE产品都提出了对AOP的支持,相信在未来的应用中,会逐渐采用这种技术。

参考文献:

[1] Walls C,Breidenbach R. Spring In Action[M]. Beijing: Manning Publications,2007.

[2] 李 刚. Spring 2.0 宝典[M]. 北京:电子工业出版社,2006.

[3] 王申源,董传良,刘英丹. 面向方面的编程的研究与实现[J]. 计算机应用研究,2004(11):220-223.

[4] 石丹丹. 面向方面编程模式的探讨[J]. 武汉理工大学学报,2005,27(1):93-95.

[5] 张广红,陈 平. 关于AOP实现机制和应用的研究[J]. 计算机工程与设计,2003(8):14-17.

(上接第119页)

运动情况下更具有优势,能够更快地搜索到大运动向量的匹配点,但是有时会导致冗余甚至错误,这是由于采用的搜索点分布过稀所造成的;而ADS算法由于摒弃了大模板,所以更适合小运动图像,对于运动比较激烈的图像,编码后图像的质量不是很理想。

4 结束语

随着网络和数字化时代的飞速发展,对于视频图像多媒体的处理要求越来越高,追求高压缩比、高传输效率、高清晰图像质量是视频技术发展的方向。运动估计对视频编码器是否能实现实时编码和编码的视频质量有着重要影响。因此,有必要对运动估计做深入研究,在传统运动估计算法基础上寻求改进方法,寻找到运动估计速度与精度的平衡点,以提高编码效率。

参考文献:

[1] Zhu S,Ma K K. A New Diamond Search Algorithm for Fast Block-matching Motion Estimation[J]. IEEE Trans Image Processing,2000,9(2):287-290.

[2] 刘 翔. 基于光流场算法的运动估计方法研究[J]. 无线电工程,2006,36(4):17-20.

[3] 姚 晨,沙济彰. 用于块匹配运动估计的SGDS算法[J]. 计算机与现代化,2006(1):8-12.

[4] 莫路锋,熬 山,顾洁宇. 一种基于MPEG4的运动估计模板选择算法[J]. 传感技术学报,2006,19(3):36-39.

[5] 倪 伟,郭宝龙,丁贵广. 基于六边形的运动矢量场自适应搜索算法[J]. 计算机工程,2005,31(13):10-12.

[6] Zhu Ce, Lin Xiao, Chau Lap-Pui. Hexagon-Based Search Pattern for Fast Block Motion Estimation[J]. IEEE Transactions on Circuits and Systems for Video Technology,2002,12(5):349-355.