文章编号:1008-0570(2011)02-0211-03

基于 JAVA 的动态代理实现的 AOP 的研究

The Research Of AOP Based On Java Dynamic Proxy

(长沙理工大学)肖 露 龙鹏飞
XIAO Lu LONG Peng-Fei

摘要:面向方面编程(Aspect-oriented Programming,AOP)是一种新的软件开发思想,面向方面编程的思想是对于面向对象编程思想在处理横切关注点时的缺点而提出来的。本文首先介绍了 AOP 的基本思想,然后针对它在 JAVA 语言中的实现,具体介绍了 JAVA 的动态代理来怎样实现一个简单的 AOP 容器,并且分析了它的优缺点,最后展望未来分析了 AOP 的发展前景。

关键识: 面向方面编程; AOP; JAVA 动态代理; 反射机制
中图分类号: T **文献标识码: A**

Abstract: AOP (Aspect-oriented Programming, AOP) is a new software development ideas, the idea of aspect-oriented programming object-oriented programming for thinking in dealing with cross-cutting concerns raised at the time of the short comings of . This article first introduces the basic idea of the AOP, and then against it in the JAVA language to achieve specific JAVA introduced the dynamic proxy to how to implement a simple AOP container, and an analysis of its strengths and weaknesses, and finally look forward to the future, analysis of the AOP prospects for development.

Key words: Aspect-oriented programming; AOP; JAVA dynamic agents. Reflex mechanism

技术创新

1 引言

软件编程方式和编程语言从机器码编程、汇编语言编程、面向过程编程到面向对象编程 (Object-orientedProgramming, OOP) 的每一次飞跃都使得软件编程的过程更加接近人类的思维习惯和认识规律。而面向方面编程 (Aspect-orientedProgramming,AOP) 是一种新的软件开发思想 , 也是随着软件系统复杂程度的不断增加 ,为了降低开发难度 ,提高开发人员分解复杂问题的能力而出现的。面向对象的编程思想把客观世界看成是由许多对象组成的 ,这在一定程度上符合了人类的自然思维方式。它很好地解决了软件系统中角色划分的问题 ,使软件开发中的许多关注点都模块化了 ,将这些关注点的具体实现细节封装在类中 ,以实现信息的隐藏、数据的抽象和封装。但是 , 在软件系统中还有另一类关注点 ,它们并不是某一个模块或者类所特有的 ,它们可能横跨多个模块或者类 ,如日志管理,事务处理,权限管理等方面,当应用 OOP 将这些内容封装为对象的行为时,会产生大量的重复代码,虽然通过一些方法可以减少这种重复,但是却不能彻底的解决该问题,于是 AOP 出现了,AOP 思想提供改善关注点分解的技术, 它提供了模块化横切关注点的能力,支持业务逻辑代码与关注点即侧面代码的分离,即将横切关注点模块化后织入到面向对象的软件系统中, 也就是从面向对象的一维思考方式中转变成二维空间的实现。这样大大的增强了代码的重用性,并极大的提高了软件的可扩展性。提高了软件开发的生产力。

2 AOP 的基本思想

AOP(Aspect-Oriented Programming)就是面向方面的编程, 这一概念最初由 Gregor?Kiczales 在施乐的 Palo?Alto 研究中心领导的一个研究小组于 1997 年提出。AOP 正是基于方面与模块形成横切,造成代码纠结这一观察提出的,并提出了一种新的编程思路来解决这一问题。之后,AOP 经由几个不同小组的努力在各自的方向上得到了发展,到目前为止,AOP 仍处于发展阶段。AOP 技术使得应用开发者仅仅关注于业务逻辑本身的开发,而不用纠缠于那些诸如安全、事务、日志等和业务逻辑无关但又是系统有效地执行业务逻辑所必须的通用性功能。AOP 以“动态织入”的方式大大提高应用开发效率,有效地降低软件的复杂性,代码的清晰性、模块化、可测试性方面也会取得很好的提升。AOP 技术中将这些通用性功能称为方面(Aspect)或切面(Pointcut, Advicor)。AOP 主要目标是用来解决分离横切关注点的问题,关注于提高软件的抽象程度和模块度,在很大程度上改善了软件的可扩展性、可复用性和可维护性等方面。 如图 1.所示,我们利用面向对象的设计思想设计业务逻辑方法,提取去日志,事务的横切关注点,将其模块化后,切入到我们的业务逻辑方法的前后,使其从原来面向对象的一维空间的实现,变成现在的二维空间实现 ,也就是说 ,原来日志,事务等功能的实现被限制在每一个需要调用它的模块中 ,而现在这些模块只需要关心自己模块特有的关注点即可 。从而避免了代码的混乱和分散 ,提高了程序的可读性 ,以及代码的重复利用率。

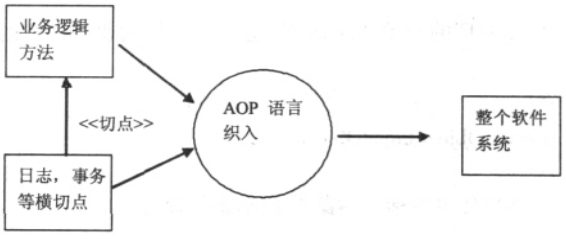

图 1 AOP 的过程

肖 露: 在读硕士

3 JAVA 动态代理实现 AOP 方案

AOP 的实现技术有静态和动态两类,静态的实现有用代理模式来实现,如果被代理的接口比较多,那我们不得不去编写很多代理类,这是一件非常麻烦的事情,并且代理类与目标对象之间的耦合程度比较大。尤其是项目非常大的时候,对于我们来说的工作量非常大,并且很难实现好。还有一种实现 AOP 技术的方式就是动态来实现, 有 CGLIB,AspectJ,java 动态代理等技术。本文讲的就是用 JAVA 的动态代理技术来实现一个简单的 AOP 容器。JAVA 的动态代理功能主要通过 java.lang.reflect.Proxy 类与 java.lang.reflect.InvocationHandler 接口完成, 这里正是通过它们实现一个简单的 AOP 容器的。其实, 像 JBoss AOP 等其他动态 AOP 框架也都是通过 Proxy 和 InvocationHandler 来实现的。Java 从 JDK1.3 开始提供动态代理(Java Proxy)功能。所谓动态代理,即通过代理类:Proxy 的代理,接口和实现类之间可以不直接发生联系,而可以在运行期(Runtime)实现动态关联。AOP(Aspect Oriented Programming):面向切面编程,其中的一种实现方法便是用 Proxy 来实现的。 Java Proxy 只能代理接口,不能代理类。本段讲解了用 JAVA 动态代理来实现一个简单的 AOP 容器。这是一个简单的 AOP 容器的实现方案。

首先我们定义逻辑处理类登陆的接口:ILogin

ILogin.java

```
package org.csust.reflectaop;
 /**
   业务逻辑的接口
   */
public interface ILogin {
   public void login();
}
```

然后编写针对本接口的实现类: LoginImpl.java

```
package org.csust.reflectaop;
 /**
   业务逻辑的接口的实现类
   *
   */
public class LoginImpl implements ILogin{
  public void login(){
   System.out.println("Welcome User ! ");
  }
}
```

接着编写 AOPContainer 类,这个类是我们 AOP 实现的核心类,也是 JAVA 动态代理机制的实现类。

```
package org.csust.reflectaop;
public class AOPContainer implements InvocationHandler {
 /**
  * 要处理的对象,目标对象(也就是我们的业务逻辑的对象)
  */
 private Object targetObject;
 /**
  * 参数传进目标源对象,生成动态代理对象。
        */
 public Object newProxy(Object targetObject) {
      this.targetObject = targetObject;
      return Proxy.newProxyInstance(
this.targetObject.getClass().getClassLoader(), this.targetObject.getClass().getInterfaces(),
  this);
 }
 /**
  * 此方法是动态的。自动执行的.
  */
 public Object invoke (Object proxy, Method method, Object[] args)
        throws Throwable {
      Object ret = null;
      try {
          // JAVA 通过这条语句执行原来的方法(java 的反射机制)
          ret = method.invoke(this.targetObject, args);
          // 执行原来的方法之后记录日志
          userLog();
      } catch (Exception e) {
          e.printStackTrace();
      }
      // 返回方法返回值给调用者
      return ret;
 }
 //用户登陆后的日志记录方法,也就是我们的横切关注点。
 private void userLog(){
      System.out.println("User enter at time  "+ new Date());
 }
 }
```

笔者讲解一下上面的一些方法的意思。Proxy. newProxyInstance 的参数: 必须传送以下 3 个参数给 Proxy. newProxyInstance 方法:ClassLoader,Class[],InvocationHandler。其中参数 1 为代理类的类加载器,参数 2 为被代理的接口 Class,参数 3 为实现 AOPContainer 接口的实例。该方法返回一个带有代理类的指定调用处理程序的代理实例,它由指定的类加载器定义,并实现了指定的接口。当程序显示调用接口的方法时,其时是调用该实例的方法, 此方法又会调用与该实例相关联 InvocationHandler 的 invoke 方法。这样我们可以在实现了 InvocationHandler 接口的类中来调用 InvocationHandler.invoke 方法来调用某些处理逻辑或真正的逻辑处理实现类。 在本例中, 我们通过在 AopContainer.invoke 方法 method. invoke (targetObject, args) 来调用 targetObject 类的与被代理接口的同名方法。其实在这里我们是采用了 JAVA 的反射机制来实现的,JAVA 反射机制是在运行状态中,对于任意一个类,都能够知道这个类的所有属性和方法;对于任意一个对象,都能够调用它的任意一个方法;这种动态获取的信息以及动态调用对象的方法的功能称为 java 语言的反射机制。在本例中我们在运行状态中动态的调用了目标源对象的一个方法, 生成动态代理。targetObject 必须实现了被代理的接口。因为 java 的动态代理只能代理接口,不能代理类。在我们这里的例子里, 我们在实际上被调用的业务逻辑方法的前后输出

技术创新

了日志信息。

最后编写测试类:TestAOP.java

```
package org.csust.reflectaop;
public class TestAop {
 public static void main(String[] args) {
  // 初始化目标对象
     LoginImpl login=new LoginImpl();
     //调用容器类方法生成动态代理
     ILogin iLogin=(ILogin) AopContainer.newProxy(login);
     //用代理访问目标源的方法
     iLogin.login();
 }
}
```

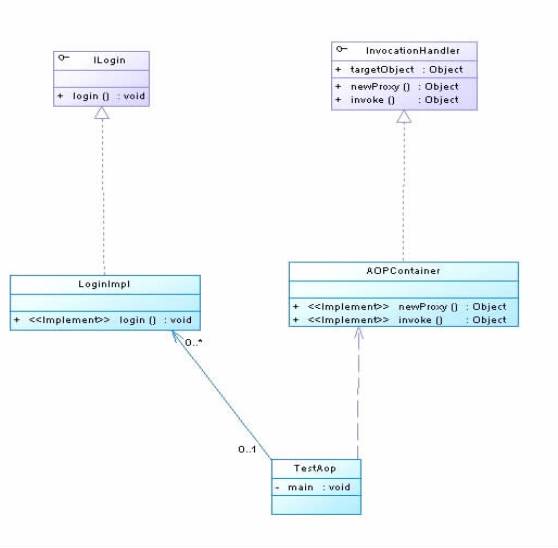

图 1.1 类关系图

至此整个 AOP 容器全部类编写完毕，我们采用的是 JAVA 动态代理机制来实现的 AOP,动态代理源于代理模式,即通过接口实现对业务对象的访问，但动态代理无需为每一个需代理的业务对象静态的生成代理对象，只需提供需要代理的接口与代理实现,就可以在运行时动态的生成代理对象,代理对上述接口的访问,这时我们利用 JAVA 的反射机制,在运行期间,将我们的横切关注点动态的织入进去。,动态织入可以在运行上下文动态的决定插入的方面代码,在程序运行时,根据上下文决定调用的方面,它们的先后顺序,增加或删除一个方面等来调用。运行时织入则在运行时，根据对方法的调用执行适当的方面代码以实现织入。

在本例中我们实现目标源对象方法后执行了日志记录,其实我们也可以在目标源对象前，或者抛出异常后等方式时执行该日志记录方法,这里让笔者又想到了 SpringFramework 的五种 Adivce，笔者也正在考虑当中对本例进行适当的补充来实现它的五种 Adivce。通过对 JAVA 动态代理机制实现的 AOP 的分析，我们发现与我们传统的用代理模式来写的代理类来实现的代理机制我们发现少了很多代码，也就是代码的重用性得到了极大地提高,并且系统的可扩展性也是得到了极大地改善,在本例中，因为我们没有采用硬编码的方式来实现在 Login 中实现日志记录，以后假如我们要在 Login 类中增加一个安全性检查，我们只要在 AOPContainer 的 invoke 方法里面加入安全性检查即可实现,这时会动态织入。也并不用去修改我们的 Login 类。当然在本例中我将关注点放到了我们的代理类中，这不是一种好的编程风格。这里主要是为了讲解的方便。我们应该将我们的关注点模块化,也就是实现我们的操作类与代理类实现解耦。笔者前几天看到过一篇类似的文章,实现了这样的分离,是采用了 JAVA 的反射机制来实现的. 其实 JAVA 的动态代理机制是有些缺陷的,即它只能代理接口,而不能代理类。在这里如果我们要代理类，我们可以采用 CGLIB 来实现。而不能用 JAVA 的动态代理，但是在实际开发中我们大部分对我们的业务逻辑都会去编写接口,这也是面向接口编程的一种思想。所以我们也一般推荐使用 JAVA 动态代理机制来实现 AOP。而不使用 CGLIB。目前非常流行的 SpringFramework 默认使用 JAVA 提供的动态代理机制,此时,业务对象通过接口编程,若需要代理对类的访问,则需要使用 CGLIB,这是一个开源的动态代理实现。

4 结束语

AOP 不是一种取代 OOP 编程技术的技术,而是对于 OOP 技术的补充,它解决了一些面向对象技术无法很好解决的问题。它与面向对象技术能起到相互补充的作用，解决横切关注分散在核心代码中,无法模块化的问题,对软件系统中不同的关注点进行分离。面向方面编程还能自动将横切关注点织入到面向对象的软件系统中,在一定程度上弥补了面向对象编程的一些缺陷。本文比较详细的叙述了利用 JAVA 动态代理机制来实现一个简单的 AOP 容器,即利用 JAVA 的反射(Reflection),基于动态代理(Dynamic?Proxy)等机制来将关注点织入到面向对象的系统中。

作者对本文版权全权负责,无抄袭。

参考文献

[1]KiczalesG. An overview of aspectJ [C].ECOOP,Object-Oriented Programming, Springer-Verlag, LNCS, 2001.327-353.

[2] JBoss aspect oriented programming [EB/OL]. http://www. jboss. org/products/aop.

[3] Grundy J. Aspect-oriented requirements engineering for component-based software system [C]. 4th IEEE International Symposiumon RE,IEEE Computer Society Press, 1999.84-91.

[4]Joseph D. Gradecki Nicholas Lesiecki. 精通 AspectJ 2005-1-1

[5] 田志,魏志强,贾东宁,:对象/关系映射持久化技术的研究及应用[J];微计算机信息;2008 年 09 期

作者简介:肖露(1985-),男(汉族),湖南岳阳人,长沙理工大学 08 级计算机软件与理论在读硕士,主要研究方向为软件工程,高级编译器设计;龙鹏飞(1960-),男(汉族),教授,硕士生导师,主要研究方向软件工程以及 web 应用.

Biography: Xiao Lu (1985 -), male (Han), Hunan Yueyang, Changsha University of Science and Technology 08 Computer Software and Theory in the Master, The main research directions for software engineering, advanced compiler design;Long Pengfei (1960 -), male (Han), Professor, Master teacher, the main research direction of software.

(410076 湖南长沙 长沙理工大学计算机与通信工程学院)
肖 露 龙鹏飞

(School of Computer and Communication Engineering, Changsha University of science and technology, Hunan Changsha, 410076, China) XIAO Lu LONG Peng-fei

通讯地址:(410076 湖南长沙 长沙理工大学计算机与通信工程学院) 肖 露

(收稿日期:2010.07.12)(修稿日期:2010.10.12)

技术创新