基于AOP的工厂模式研究

周庆泉,郝克刚
(西北大学 信息科学与技术学院,陕西 西安 710127)

摘 要:工厂方法模式是面向对象软件开发中十分常见的设计模式之一,但其也存在着可扩展性和重复维护的问题。面向方面思想及技术的发展为解决上述问题提供了技术基础。应用AspectJ实现了基于面向方面技术的工厂模式,较之面向对象编程实现的工厂方法模式,本实现具有良好的可扩展性和可维护性。并通过分析认为本实现完全可以完成抽象工厂模式的任务,实现了两种工厂模式的统一。

关键词:面向方面编程;工厂方法模式;织入
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2008)08-0047-03

A Research on Factory Pattern Based on AOP

ZHOU Qing-quan, HAO Ke-gang
(College of Information Science and Technology, Northwest University, Xi'an 710127, China)

Abstract: Although the factory method model is one of the very common design patterns in the development of object-oriented software, it has such problems as bad extensibility and redundant maintenance. Through the application of AspectJ, achieved the factory model based on aspect-oriented technologies, and eliminated the defect existing in object-oriented factory method model. Through analysis, it can be confirmed that the model not only can entirely complete the tasks of abstract factory pattern, but also can achieve the reunification of the two factory models.

Key words: aspect-oriented programming; factory method pattern; weaver

0 引 言

面向对象编程(OOP)能够较好地将问题领域中的实体映射到软件中的对象,利用封装、继承和多态等技术完成系统的角色划分及模块化,在很大程度上提高了软件的可扩展性和可复用性。同时GOF的"设计模式"[1]的出现为人们使用面向对象技术提供了解决相似问题的编码典范。但是面向对象技术同样有它的局限性,实践证明面向对象技术不能很好地解决系统分离关注点[2]问题,在解决这类问题时往往会造成代码纠缠、分散,使模块难以复用。

面向方面编程(AOP)[3,4]技术的出现,为这些问题的解决带来了希望。方面通过编织过程打破了类和对象的封装性,为人们在避免修改源代码的情况下提供了修改类属性和方法的能力。面向方面技术的特点为解决存在于面向对象技术实现的经典设计模式中的可扩展性和可维护性等问题提供了契机。如工厂方法模式(Factory Method Pattern)。

文中应用AOP方法,使用AspectJ[5]的面向方面语法实现了基于Java语言的工厂模式。较之OOP实现,该实现具有良好的可扩展性和可维护性。

收稿日期:2007-11-18
基金项目:国家863计划资助项目(2004AA115090)
作者简介:周庆泉(1982-),男,山东滨州人,硕士研究生,研究方向为软件工程与理论;郝克刚,教授,博士生导师,研究方向为软件工程与理论。

1 面向对象编程实现的工厂方法模式

工厂方法模式中核心工厂类不再负责所有产品的创建,而是将具体创建的工作交给子类去做,成为一个抽象工厂角色,仅负责给出具体工厂类必须实现的接口,而不接触哪一个产品类应当被实例化这种细节。

1.1 面向对象编程实现的工厂方法模式结构

面向对象编程实现的工厂方法模式如图1所示。包括Creator、ConcreteCreator、Product、ConcreteProduct四个模块。

(1)Product定义了工厂方法多要创建的产品的接口。

(2)ConcreteProduct为一个Product接口的实现。

(3)Creator可为抽象类或者接口,声明工厂方法,该方法返回一个Product类型的对象。可以通过调用

工厂方法来得到一个 Product 对象。

(4)ConcreteCreator 继承(或实现)了 Creator,重定义了工厂方法以返回一个特定的 Product。

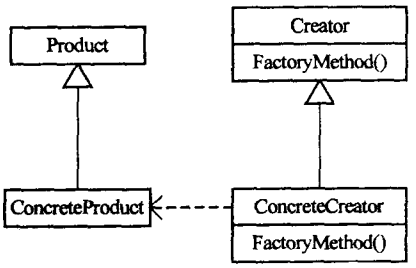

图 1 面向对象编程实现的工厂方法模式

1.2 面向对象编程实现的工厂方法模式的缺点

工厂方法模式提供一个抽象的工厂接口,在不同的产品结构上实现不同的产品工厂具体类来创建该层次中的产品,通常一个具体工厂创建一个具体产品。在新增产品的时候,只需要实现新的对应工厂就可以满足需要,是符合开闭原则的。

工厂方法模式同样存在其固有的缺点:工厂方法只获得了相对的扩展性,当需要修改产品结构中的对应产品的时候,需要修改各个工厂。如:一个抽象工厂定义了三个产品的工厂方法,但是现在系统中需要加入第四种产品,可以有几种方式实现。第一种想到的方法就是在抽象工厂里添加生产第四种产品的工厂方法,但是正如上面所说的,需要修改每一个子类,在里面添加相关的生产实现,不符合"开-闭"原则,同时也因为有大量的代码修改工作而被人们诟病。

其次该模式由于具体工厂类依赖于具体的产品类,所以当产品类发生改变时,会影响到依赖于该产品的具体工厂,产生了重复维护的问题。

1.3 面向对象编程实现的工厂方法模式缺点的本质

工厂方法模式通过抽象出工厂方法来避免抽象工厂对具体产品的依赖。这一点是成功的,但是同时又产生了两个依赖关系,第一个就是具体工厂对抽象工厂的依赖关系,第二个是具体工厂对具体产品的依赖关系。第一个依赖关系导致了可扩展性问题。当需求改变,需要对抽象工厂添加新的工厂方法时,这种改变打破了"开闭"原则,并将改变传播到各个具体工厂。每个具体工厂都需要进行代码修改,导致了可维护性的问题。第二个依赖关系导致了重复维护的问题。每当有产品发生变化时,就有可能导致依赖该产品的具体工厂的修改工作。

2 基于面向方面的工厂模式

2.1 面向方面技术及 AspectJ

在 1997 年的欧洲面向对象编程大会(ECOOP97)上,施乐公司 Palo Alto 研究中心首席科学家、大不列颠哥伦比亚大学教授 Gregor Kiczales 等人首次提出了 AOP 的概念[3]。此后每年的 ECOOP 大会上都会有关于 AOP 的专题研讨会,各大研究机构、公司以及大学都纷纷投入大量人力资源进行研究。AOP 不是面向对象程序设计(OOP)的一种代替,而是为了弥补面向 OOP 的不足。OOP 本身存在着一些局限性,比如,在处理横穿多个类的非核心关注点问题时会导致代码散乱、设计和代码的重用性差、维护代价高等问题。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。

面向方面编程(AOP)已经经历了多年的理论和工具的研究,出现了很多优秀的编译工具,比如:AspectJ、Aspectwerkz、JBoss AOP 等。AspectJ 是一个比较早且成熟的编译器,它是对 Java 的扩展和 Java 无缝连接,引入了 AOP 的概念和关键字。文中将使用 AspectJ 来做相关实现。

AspectJ 的一些主要概念[5]如下:

(1)连接点(Join Points)。

连接点就是在程序中可以识别的变量或者方法,是预先定义的方法开始执行的起始点。这些连接点既可以是静态的也可以是动态的。

(2)切入点(Pointcuts)。

切入点是用于选择连接点。它就像一个过滤器,按照满足执行内容的条件选择一些连接点。

(3)通知(Advice)。

通知是一些可以在连接点(Join Points)处执行的程序块,但是连接点(Join Points)必须满足切点(Pointcuts)。

(4)类型间声明(Inter-Type Declarations)。

AOP 除了支持动态的通知(Advice),也提供了静态的方法 Inter-Type Declarations,它可以向类中添加方法和属性。

(5)方面(Aspect)。

方面是一个模块单元,它像类一样包含有方法和属性,同时还可以包含切点(Pointcuts)、通知(Advice)、类型间声明(Inter-Type Declarations)。

2.2 基于面向方面的工厂模式的结构

使用面向方面编程实现的工厂模式与面向对象实现的工厂方法模式有了很大的不同,如图 2 所示。

(1)Factory 类可以为一个没有任何属性和方法的空类,也可以做成单件模式(Singleton)的空类。

(2)FactoryMethodAspect 方面为 Factory 定义了一个带参数的工厂方法,并该方法建立了一个联结点。

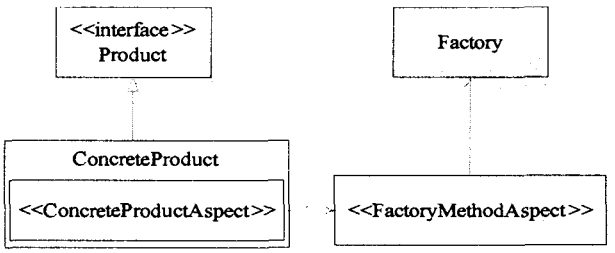

图 2　面向方面的工厂模式

(3)Product 为一产品接口。

(4) ConcreteProduct 为一具体的 Product 产品。ConcreteProductAspect 产品方面内置于 ConcreteProduct 类中。

2.3　面向方面工厂模式分析

在面向方面的实现中,工厂模式中仅有空工厂类和相关 aspect。核心为工厂类,该工厂类没有任何工厂方法和对产品的知识。

添加工厂方法是使用 AspectJ 的类型间声明实现的,为了适应面向方面的特点工厂方法被设计为带参数的方法。同时该方面提供一个联结点,用于捕获对该工厂方法的调用。

产品类和产品方面在一个文件里。产品方面提供了产品类到工厂方法的参数绑定。当有多个 advice 绑定到同一个联结点时,方面编译器将自动产生一个功能类似于 if - else 的结构。当工厂方法被调用时,调用被捕获并转到 advice 执行,判断参数与产品绑定参数是否相符,选择返回产品或者通过 proceed()方法返回调用,并被可能存在的其他 advice 捕获,只有不再有 advice 捕获调用时才执行该调用。

面向方面工厂模式的可扩展性。工厂方法方面将经典模式里的接口依赖转化为工厂对方法方面的依赖,当需要添加接口时,只需要添加一个相关方法方面即可,不需要修改工厂类。产品类所包含的方面反转了具体工厂对产品的依赖,添加产品只需要添加产品类(以及包含在其内的产品方面)。

面向方面工厂模式的可维护性。当产品实现发生改变时,此时只需要修改产品类文件(以及包含在其内的产品方面)。

通过分析可认为该模式具有良好的可扩展性和可维护性,解决了面向对象编程实现的经典工厂方法模式的弊端。

3　面向方面工厂模式与抽象工厂模式

抽象工厂模式提供一个创建一系列相关或相互依赖对象的接口而无需指定它们具体的类。它与工厂方法模式的区别在于工厂方法只创建一个产品接口的不同实现而抽象工厂则创建多个产品接口的多个实现,并保证每个具体工厂创建的多个产品之间的约束条件。当只有一个产品类时,抽象工厂可以退化成工厂方法模式。

面向方面的工厂模式可以通过改变参数意义完成抽象工厂的功能。抽象工厂模式通过实例化不同的具体工厂来实现条件约束,而在面向方面的工厂中使用同一个工厂,通过传递给不同的工厂方法相同的参数来完成条件约束。比如一个支持多种视感标准的用户界面工具包[1],完全可以通过传入平台信息参数作为创建相关产品的约束条件,从而得到满足约束条件的产品对象。

4　结束语

文中应用 AspectJ 面向方面技术实现的工厂模式,能够很好地完成 GOF 的"设计模式"中工厂方法模式和抽象工厂模式的任务,解决了困扰这两个模式的可扩展性和重复维护的问题,实现了两种模式的统一。

参考文献:

[1] Gamma E, Helm R, Johnson R, et al. Design Patterns Elements of Reusable Object - Oriented Software[M]. Boston: Addison Wesley Professional, 1995.

[2] Hürsch W, Lopes C. Separation of Concerns[R]. [s. l.]: College of Computer Science, Northeastern University, 1995.

[3] Filman R E, Elrad T, Clarke S, et al. Aspect - Oriented Software Development[M]. Boston: Addison Wesley Professional, 2004.

[4] 高海洋,陈　平. AOP 综述[J]. 计算机科学,2002,29(10): 133 - 135.

[5] Colyer A, Clement A, Harley G, et al. Eclipse AspectJ: Aspect - Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools[M]. [s. l.]: Addison Wesley Professional, 2004.

(上接第 46 页)

[10] Yang X, Cao J, Megson G M, et al. Minimum Neighborhood in a Generalized Cube[J]. Information Processing Letters, 2006, 97(3): 88 - 93.

[11] Chang G Y, Chang G J, Chen G H. Diagnosabilities of Regular Networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2005, 16(4): 314 - 323.