面向方面测试技术的研究

The Research of Aspect Oriented Testing Techniques

张琼声 夏守姬 黄亭宇 (中国石油大学(华东)计算机与通信工程学院 山东东营 257061)

摘要:面向方面编程运用方面模块化横切关注点,构建出易于理解、易于扩展以及高质量的软件。然而,软件测试是软件质量保证的关键因素,那么开发有效的测试方法来检验 AOP 的正确性就显得十分重要。本文简述了面向方面编程的基本概念,详细介绍了四种 AOP 测试方法,并根据检验错误类型的能力对比了其中部分方法的性能,分析了 AOP 测试方法的研究现状、面临的技术问题以及未来的研究工作,最后总结了开发 AOP 测试技术和工具的重要性。

关键词:面向方面编程 软件测试 横切关注点 方面 模块性

1 引言

软件开发的根本目的是生产高质量的软件,而软件测试的目标就是要在软件投入运行前,对软件需求分析、设计规格说明以及编码进行检验,力求在测试代价最小的前提下尽可能多地发现软件中存在的错误,以便及时进行修改和弥补,从而保证软件质量。统计表明,在典型的软件开发项目中,软件测试工作量往往占软件开发总工作量的 40% 以上,而在软件开发的总成本中,用在测试上的开销占 30% 到 50% 。由此可见,软件测试是软件生命周期中的一个重要阶段,是软件质量保证的关键步骤。

面向方面编程(aspect - oriented programming, AOP)[1]是一种新兴的编程思想,它能够提高软件的模块性和内聚性,降低软件的复杂度,为软件质量的提高提供了一个新的途径。但是,AOP 并不能保证程序员不犯错误,而且它本身也不提供正确性验证,那么开发有效的测试方法来检验 AOP 的正确性至关重要。

本论文阐述了开发 AOP 测试方法面临的困难,从基本思想、实现过程等方面详细介绍了四种 AOP 测试方法并对部分方法的性能进行了比较分析,论文最后分析了 AOP 测试方法的研究现状、存在的问题以及未来的研究工作。

2 AOP 测试面临的困难

2.1 AOP 概述

AOP 是一种关注点分离技术,它运用方面(Aspect)机制捕获横切关注点,将分散的应用组成单独的模块,有效地解决了横切关注点造成的代码交织和散布问题。所谓横切关注点是指横跨系统各个对象层次的模块,但与它所跨越的对象代码在功能上没有相关性。为了描述和实现 Aspect 机制,AOP 引入了一些新的构造:连接点(Joint Point)是程序执行中明确定义的点,在这些点中可执行 Aspect 代码;切入点(Pointcut)是捕获、识别程序中连接点的结构,是通知的激发条件,它决定了各种 AOP 特征将如何被运用到类中;通知(Advice)用于声明在切入点表达式中定义的连接点被调用时应执行的动作,它包括 before、around 以及 after 三种类型;导言(Introduce)是一个增加方法到存在类中的途径;方面是 AOP 的核心,类似于 OOP 中类的概念,是把切入点和通知封装在一起体现横切关系的模块单元,它也可以包含方法和属性、从其他类或方面扩展以及实现接口等。

2.2 AOP 测试面临的困难

与面向过程和面向对象语言不同,AOP 引入了一些新的语言构造,如连接点、切入点、导言以及通知等,它们不仅改变了软件的开发过程,还严重影响了程序的状态和行为(如导言可以把一些新方法和实例变量引入核心关注点中;通知则能够为类引入一些额外的不可见行为),从而导致方面和类之间的交互更加复杂,那么传统的测试方法不能直接应用到 AOP 中,它需要开发专门的测试技术和方法来支持这些特殊构造,这为 AOP 测试带来了困难。

此外，AOP 本身的特性问题也不容忽视：首先方面没有独立的实体或存在物，它们完全依赖其他一些类的上下文；其次，方面的实现同它们的织入上下文紧密关联；再者，方面或类代码中的控制和数据依赖关系不明显；最后就是特殊的织入过程可能还会产生一些突发行为。这些问题都给 AOP 的测试带来了困难。

因此，开发适合面向方面的软件测试技术是 AOP 的一项迫切而艰巨的任务。

3 AOP 测试方法的介绍

AOP 作为一种新的编程范例，目前还缺乏成熟的测试方法。下面主要介绍四种不同的测试方法，它们在实现上各有其特点。

3.1 基于错误模型的系统测试方法

任何系统测试方法都是针对程序中出现过的一些错误类型，如果对于错误类型没有任何概念，那么开发的系统测试方法也是没有意义的。Alexander、Bieman 和 Andrews 在文献[2]中提出了一个错误模型，它包括六种类型的错误，反映了 AOP 与面向对象以及面向过程截然不同的特征。

(1) 错误的切入点结构约束。切入点包含识别、捕捉特定类型连接点的表达式描述。对于一个切入点 p，关注点 C 中每个匹配的连接点 j 将被织入一个与 p 关联的通知。p 表达式中定义的匹配函数用于确定被选择的连接点。如果定义的匹配函数约束过强，一些必需的连接点会被忽略；如果约束太弱，一些本应该忽视的连接点将被捕获。以上任何一种情况都很可能造成织入程序的错误行为，这也是 AOP 中最明显的一类错误。

(2) 错误的方面优先级。在面向方面程序中，多个方面有可能同时影响一个连接点，那么通知的不同织入次序会影响系统的行为。在这种情况下，控制通知织入次序是很重要的，它可以通过为不同的方面指定不同的优先级来确定，高优先级的通知能够被优先织入。

(3) 未能建立期望的后条件。方面会引起类代码中控制流的改变，从而导致类不能实现其类契约的后条件。而其使用者希望关注点能根据它们的契约运行，不管通知是否被织入，方法的后条件都必须满足，也就是在织入过程之后应该保留关注点的行为契约。因此，对于正确的行为，被织入通知必须允许核心关注点的方法满足其后条件。然而，在所有可能的织入上下文以及方面组合中定义不会造成行为契约违背的通知是一个困难的挑战，本身也是一个极可能的错误源。

(4) 未能保存状态不变量。关注点的行为是根据其状态的物理表示以及作用于此状态上的方法来定义的。除了要建立方法的后条件外，方法也必须保证其状态不变量的保留。方面的通知和导言能够为关注点引入新的状态，那么要保证织入不会造成状态不变量的违背也是一个艰难的挑战，同时也是一个错误的来源。

(5) 错误的控制流焦点。一个切入点表达式指定了要选择的连接点。然而，这种选择的确定不仅仅是在织入阶段，很多情况取决于运行时期。因此，要诊断出由于运行时上下文造成的错误是很困难的。

(6) 错误的控制依赖变更。around 通知能够为方法引入新的分支控制流，从而极大地改变语句之间的依赖，因此要保证程序的正确执行，语句之间的依赖关系也需要测试。

虽然这仅仅是一个理论上的错误模型，还没有形成实际完整的测试方法，但是它有助于开发测试工具以及确定测试覆盖策略和标准，为面向方面程序的系统测试做出了重要的第一步。

3.2 基于状态的测试方法

从面向对象程序的 FREE[3] 测试设计模型中受到启发，Dianxiang Xu，Weifeng Xu 和 Kendall Nygard 在文献[4]中提出了一种基于状态的测试方法。它的基本思想是：首先，为了详细说明类和方面的状态和行为，把类的 FREE 模型扩展成方面状态模型（Aspectual State Model，ASM）；然后将 ASM 转化成一棵转换树，该树包含一个可充分测试对象行为以及类与方面之间交互的测试套件。

FREE 模型是面向对象软件开发使用的一种状态图，它描述了对象的状态和动态行为，并提供了实现的指导。ASM 模型是 FREE 状态模型的一个扩展，为了表示 AOP 中基本的语言构造——连接点、切入点和通知，ASM 增加了一些新的标志。图 1 显示了一个 ASM 例子。其中状态（A）表示初始状态，它接受一个具有约束条件（[cond1]）（值为真时）的消息（M1）后转入状态（B）。消息（M1）同时受三个通知影响：具有约束

条件([cond2])的 before 通知将(A)转入状态(C);具有约束条件([cond4])的 after 通知将(A)转入状态(E);具有约束条件([cond3])的 around 通知将(A)转入状态(D)。

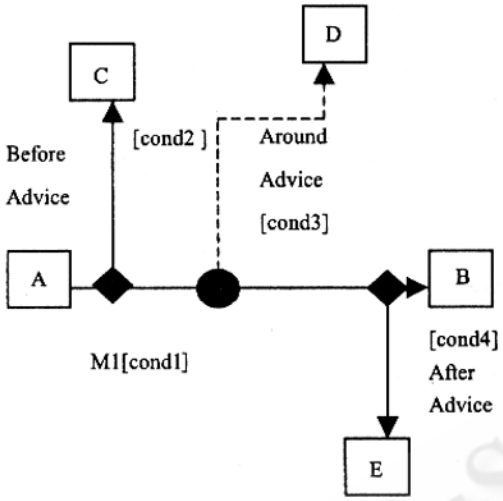

图 1　一个 ASM 例子

转换树测试的核心思想就是通过一定的算法将 ASM 转变成一棵转换树,然后根据从根到叶节点的每条路径所表示的消息序列,生成相应的测试套件。在转换树中,每条从根到叶节点的路径表示检验对象行为的一个测试需求。如果此路径代表的消息序列上的变量被指定为满足相应约束条件的值,那么这个测试需求就变成了一个具体的测试用例。

3.3 基于方面流图的测试方法

Weifeng Xu,Dianxiang Xu 和 Vivek Goel 等人在文献[5]提出了一个基于方面流图的测试方法。该方法包括:首先把类状态模型和方面状态模型合并为一个方面域状态模型(Aspect Scope State Model, ASSM);然后,使用通知和方法流图替换 ASSM 中相应的类与方面之间的转化以及对应的动作,从而构造一个方面流图(Aspect Flow Graph,AFG);此外,根据 ASSM 和一组定界参数连同决定行为的参数关系生成一棵转换树;最后,利用 AFG 和转换树,产生一个基于代码的可运行测试套件。

方面域状态模型 ASSM 的构造是通过合并两个状态模型完成的:(1)类状态模型。它描述了特定类的状态转换行为;(2)方面状态模型。它包括四个状态:a 状态和 ω 状态分别对应方面的入口和出口;其它两个状态依次称为 Action Prior 和 Action Posterior。此外,方面还包括 before 通知、after 通知以及类的 arbitrary 动作,它们是方面中的特定动作。其中 before 通知将方面中的状态从 a 状态转变到 Action Prior 状态;而 after 通知将其从 Action Posterior 转变到 ω 状态;arbitrary 动作则将状态从 Action Prior 变化到 Action Posterior。构造算法的基本思想是检查类状态模型中边表示的每个动作,该动作对应类中一个定义良好的连接点。如果它被方面中的一个切入点捕获,那么使用相应的方面状态模型取代此边。

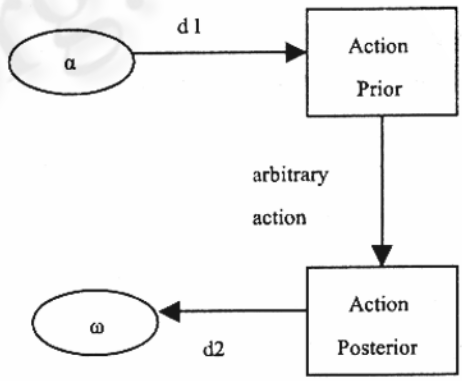

图 2　方面状态模型

方面流图(AFG)类似类流图,仅有的差别在于 AFG 也可以表示由方面引入的控制流。它是通过合并 ASSM 以及方法和通知流图构造的。具体来说,ASSM 中任何导致方面状态改变的动作(通知和方法)都被相应的通知和方法流图替换。AFG 不仅显示了运行时通知(before 和 after)是如何被附加到一个选中的连接点上,也展示了类行为是如何受附加的通知影响。

转换树显示了所有可能的状态转换序列,根据这些序列就可以生成满足数据流覆盖标准的测试用例。

3.4 基于数据流的单元测试方法

Jianjun Zhao 在文献[6]中提出了一个基于数据流的 AOP 单元测试方法。该方法测试 AOP 中的两类单元:(1)方面,程序横切实现的模块化单元;(2)类,行为受一个或多个方面影响。它使用控制流图来寻找被测试单元的 def-use 对,然后利用此信息指导选择测试用例。

"单元测试"的意思就是测试程序的每个单元(基

本组件),从而检验单元的详细设计是否正确实现。然而,它在 AOP 中的含义更难理解。由于方面可以打破类的封装,那么类不再被认为是一个良好封装的内聚单元。此外,一个类可能受多个方面的影响而一个方面也可能影响多个类,方面与类之间的交互也变得很复杂。因此,单独测试 AOP 中的类和方面是不现实的,应该(1)测试方面时连同那些其行为受通知影响的方法(从方面角度);(2)测试类时连同那些能影响该类行为的通知以及能引入新成员到该类中的导言(从类角度)。

为了能正确的测试方面和类,将 AOP 分为 5 类:

(1) 聚集方面。它是单个方面连同一个或多个类中的方法,这些方法受方面中通知的影响,表示为 c – aspect。

(2) 聚集类。它是单个类连同一个或多个方面中的通知或导言,通知影响类方法的行为,而导言改变类的类型结构,表示为 c – class。

(3) 聚集方法。它是单个方法连同一个或多个影响其行为的通知,表示为 c – method。

(4) 正常类。它是单个类,行为不受任何方面的影响,表示为 n – class。

(5) 正常方法。它是单个方法,不受任何通知影响,表示为 n – method。

以上每类的流图是单独构造并且其测试方法也是分开执行的。然而,论文中只介绍了 c – aspect 和 c – class 的测试。

c – aspect 和 c – class 流图的构造过程分为 3 步:

首先构造 c – aspect 和 c – class 的调用图。它是一个有向图,表示 c – aspect 和 c – class 中各模块之间的调用关系。对于 c – aspect 来说,其顶点表示组成 c – aspect 的模块,它们可以是行为受方面影响的 c – method、通知或 n – method。边表示 c – aspect 中模块间的调用次序。

第二步,使用专用顶点(包括框架入口顶点(frame entry vertex)、框架循环顶点(frame loop vertex)、框架出口顶点(frame exit vertex)、框架返回顶点(frame return vertex)和框架调用顶点(frame call vertex))来构造 c – aspect 或 c – class 的框架,并将其插入对应的调用图中。此框架表示 c – aspect 或 c – class 的测试驱动,它可以模拟 c – aspect 或 c – class 中某些模块间的任意调用顺序,支持对 c – aspect 或 c – class 的数据流分析。

最后用模块的控制流图替换调用图中的相应顶点。

三步产生的结果就是一个称为框架式控制流图(Framed Control Flow Graph)的 c – aspect 图。c – class 的 FCFG 构造方法类似。

对于每个待测试的类或方面,此方法执行一个三层测试策略:

(1) 模块内测试。仅仅测试单个模块。对于 c – aspect,其模块可以为一个 c – method、n – method 或 c – aspect 的导言;对于 c – class,其模块可以是 c – class 的一个 n – method 或 c – method。它只检验模块体内出现的 def – use 对。

(2) 模块间测试。测试一个公共模块连同其在一个方面或类中直接或间接调用的其他一些模块。它需要测试涉及多个模块的 def – use 对。

(3) 方面内/类内测试。检验那些可以在方面或类外被访问并能够被方面或类的使用者以任意次序调用的模块。不同的调用次序将检验不同的 def – use 对。

4 AOP 测试方法的对比分析

下面简单地总结各个测试方法的特点,同时根据文献[2]中提出的错误模型比较各方法的测试性能:

(1) 基于状态的测试方法捕获方面对类状态模型的影响。它运用转换树来测试 AOP 中的所有路径,能够达到 N + 的测试覆盖率。它可以显示程序中所有的状态控制错误、潜在路径以及许多误用的状态错误。同时,该方法还可以反映错误模型中的两类错误——错误的切入点结构约束和未能保存状态不变量。然而,此方法经常会遇到状态爆炸的问题,因此,如何确定最感兴趣或最重要的路径是值得研究的下一步工作。

(2) 基于方面流图的测试方法构造方面流图来描述方面与类之间的交互。这有助于运用多种白盒测试方法以及测试覆盖标准,而且如果类代码已经被测试,那么只要把注意力集中在由方面引入的新行为和交互上,就可以重用类的测试用例并且能够更多的降低测试成本。此外,该方法还可以检测错误模型中的一类错误——错误的控制依赖变更。

(3) 基于数据流的单元测试方法通过组合单元测试和数据流测试技术来检验 AOP 中的类和方面。它没有针对任何特定的错误类型,但能反映错误模型中的第四类错误——未能保存状态不变量,因为状态错误通常是与错误的数据流相关的。不过,该方法只针对方面和类本身,没有考虑其集成测试问题,也没有考虑检验扩展类和方面的测试方法。

表格1总结了三个测试方法与文献[2]中错误类型之间的关系。由此可以得出:已提出的测试方法只针对 AOP 中出现的局部错误类型,测试覆盖面太窄,检验效率不高,缺乏全面的考虑。AOP 测试技术要不断发展,必须要在测试方法的深度和广度上进行研究。

表 1 测试方法与错误类型之间的关系
(1-6 分别表示 3.1 中的六类错误)

	基于状态的测试	基于方面流图的测试	基于数据流的单元测试
1	否	是	否
2	否	否	否
3	否	否	否
4	是	是	否
5	否	否	否
6	否	否	是

5 结论与展望

目前,AOP 仍没有成熟的测试方法支持,一方面由于人们对 AOP 的研究主要集中在问题分析、软件设计以及实现技术上,很少关注其测试技术和方法的研究;另一方面由于 AOP 中方面的特殊行为以及与类之间的复杂依赖关系,为 AOP 测试方法的开发提出了许多新的技术问题,例如如何独立地测试方面、如何定义织入程序的测试度量标准和覆盖率、如何测试方面与类之间的复杂交互等。传统的基于类的单元测试、集成测试方法已经完全不适合 AOP 的测试,需要研究新的适合 AOP 特性的专门测试方法。这些因素都制约了 AOP 测试技术的发展。

参考文献

1 G. Kiczales, J. Lamping, A. Mendhekar et al. Aspect - Oriented Programming. Proc. 11th European Conference on Object - Oriented Programming, LNCS, Vol. 1241, Spring - Verlag, June 1997, pp220 - 242.

2 R. T. Alexander, J. M. Bieman, and A. A. Andrews. Towards the Systematic Testing of Aspect - Oriented Programs. Technical Report, Colorado State University. http://www.cs.colostate.edu/~rta/publications/CS-04-105.pdf.

3 R. V. Binder. Testing Object - Oriented Systems: Models, Patterns, and Tools. Addison - Wesley, 2000.

4 D. Xu, W. Xu and K. Nygard. State Based Approach to Testing Aspect - Oriented Programs. Technical Report NDSU - CS - TR04 - XU03. Department of Computer Science North Dakota State University, September 2004.

5 W. Xu, D. Xu, and V. Goel et al. Aspect flow graph for testing aspect - oriented programs. http://www.cs.ndsu.nodak.edu/~wxu/research/436-1111jd.pdf.

6 J. Zhao. Data - flow - based unit testing of aspect oriented programs. In Proc of the 27th Annual IEEE International Computer Software and Applications Conference (COMPSAC'03), December 2003, pp188 - 197.

7 G. Denaro and M. Monga. An experience on verification of aspect properties. 4th international workshop on Principles of software evolution, ACM Press, 2002, pp186 - 189.

8 N. Ubayashi and T. Tamai. Aspect oriented programming with model checking. 1st international conference on Aspect oriented software development, ACM Press, 2002, pp. 148 - 154.

9 H. Li, S. Krishnamurthi, and K. Fisler. Verifying cross - cutting features as open systems. 10th ACM SIGSOFT symposium on Foundations of software engineering, ACM Press, 2002, pp89 - 98.