AOP综述

An Overview of AOP

高海洋　陈　平

（西安电子科技大学软件工程研究所　西安710071）

Abstract The development of software parallel to the degree it is abstracted. Traditional procedural or OO methodology now cannot meet the expanding software scale and the aggravating software intricacy very well. In recent years, a new methodology on programming named AOP (Aspect-Oriented Programming) attracts developers all over the world. With this approach, cross-cutting code scattered around the whole program is encapsulated. Therefore, the abstraction level is raised and the quality of software is improved. In this paper, the background of AOP is illustrated, as well as its main ideas, application domains and current status.

Keywords Tangled code, Aspect, Weave

1. 引言

随着计算机越来越广泛地应用于社会各个行业，应用软件的规模不断扩大，复杂度不断提高。传统的软件开发方法，如过程化程序设计、面向对象程序设计等已渐渐不能适应这种变化。近年来，一种新的程序开发方法，AOP（Aspect Oriented Programming，面向特征编程）引起了国内外的广泛关注，并被《MIT技术评论》杂志评为21世纪十种对经济和人类生活工作方式最具影响力的技术之一[1]。

本文先概述AOP产生的背景，接着分析AOP的核心思想及其应用实例，然后介绍AOP发展和应用现状，最后总结并展望AOP的发展趋势。

2. AOP的产生的背景

2.1 面向过程编程面临的问题

面向过程编程是一种自顶向下的编程方法，其实质是对软件进行功能性分解。它适用于小型软件系统，例如某一算法的实现。在大型应用系统中，自顶向下逐步求精的方法无论在系统体系结构的确立，系统的进化和维护，以及软件重用性方面都存在其固有的不足之处[2]。

2.2 传统面向对象编程面临的问题

传统的面向对象语言由于其良好的封装性、层次化性以及继承性等特性而取得了很大的成功，并且对象模型可以很好地映射到实际领域。但是在软件的生命周期中，它存在以下不足之处：

1）设计阶段，由于以类为单位组织建模，因此它不能全面地反映软件系统的需求。

2）编码阶段，将数据和方法封装到类中的思想增强了数据的安全性和软件的模块化，但是有一些数据和方法是特定于应用的，因此这种编码阶段的封装减少了代码重用的可能性。

3）维护阶段，由于类中夹杂了各种特定于应用的代码，使得维护人员难以理解代码。此外，完成某个特定需求的代码分散在各个类中，当这些代码需要改变时，很难把它们全部找到，这就给程序的健壮性带来了隐患。

由于上述这些问题的产生，需要一种新的程序设计方法从更高的层次上对软件系统进行抽象，来将传统的按功能或按对象划分程序模块的方法转化为按系统特征划分程序模块，这就是AOP的基本思想。

2.3 AOP的产生

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上都有AOP相关的专题研讨会，各大公司、大学、研究机构纷纷投入人员进行研究。2001年3月15日，Palo Alto研究中心发布了首种支持AOP的语言，AspectJ。

3. AOP的核心思想及其应用领域

3.1 什么是Aspect

所谓的Aspect，从抽象意义上讲，是对系统组件的性能和语法产生一定影响的一些属性[3]；从设计上讲，是横切系统的一些软件系统关注点；从实现上讲，Aspect是一种程序构造单元，它支持将横切系统的关注点封装到单独的模块单位中[5]。典型的Aspect如系统异常和出错处理、同步和并发控制、内存访问模式以及特定于应用的程序关注点等。

考虑一个电信模拟系统，该系统的功能包括用户打电话，接电话，电信部门根据通话性质（长途电话或本地电话）计时收费。用面向对象的方法分析该系统，基本的类图如下所示。

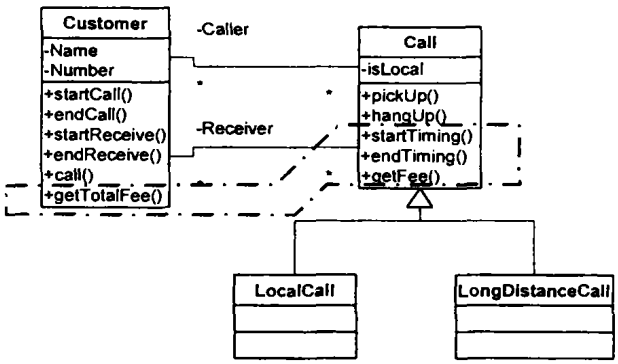

图1　用面向对象方法分析一个简单电信模型系统

由图可见，特定于应用系统的关注点，如计时、计费等代

高海洋　硕士生，研究方向为面向对象技术、分布式计算。陈　平　博导，研究方向为面向对象技术。

码分散在系统的各个类中，由此造成系统代码的缠结(如虚线框所示)。Aspect 的核心思想就是将这种横切于系统的程序关注点抽取出来，并作为软件系统的一个模块，在设计、实现阶段独立实现，从而提高系统的抽象性和模块性。

3.2 AOP 特性

衡量软件质量高低的要素主要包括可靠性、可扩展性、可重用性、兼容性以及易用性和易维护性等。AOP 作为一种程序设计方法学，关注于提高软件的抽象程度和模块性，从而在很大程度上改善了软件的可扩展性、重用性、易理解性和易维护性，并由此提高影响软件质量的其它因素。

下面通过对 OOP 和 AOP 在提高软件可扩展性、可重用性和易理解性、易维护性等方面的能力比较来阐述 AOP 特性：

1)可扩展性　可扩展性指软件系统在需求更改时程序的易更改能力。面向对象主要通过提供继承和重载机制来提高软件的可扩展性，因此它的扩展性体现在类一级。面向特征编程提供系统级的扩展机制，通过扩展 Aspect(AspectJ 支持 Aspect 继承机制)或增加 Aspect，系统相关的各个部分都随之产生变化。由此带来的另一个好处是在软件测试中，通过屏蔽某些 Aspect，可以大大简化软件的测试复杂度，提高测试精度。

2)可重用性　可重用性指某个应用系统中的元素被应用到其他应用系统的能力。面向对象的类机制作为一种抽象数据类型，提供了比过程化更好的重用性[2]。泛化机制也使可重用性得到很大提高。面向对象所提供的重用性对非特定于系统的功能模块有很好的支持，如对于堆栈的操作或窗口机制的实现等。但在特定于系统的功能模块中，一个类通常包含很多应用系统相关的数据及对其的操作，此时类的重用变得十分困难。此外，面向对象的重用也限于类一级，对于不能封装成类的元素，如异常处理等，很难实现有效的重用。面向特征编程中的系统模块包括系统组件和影响这些组件的特性，通过将实现基本功能的组件和特定于应用的系统特性分离，使得组件(包括类或者函数)的重用性得到提高，并使不能封装为类或函数的系统元素(Aspect)的重用成为可能。

3)易理解性和易维护性　易理解性和易维护性是影响软件质量的内在因素，它对软件开发人员和维护人员产生影响。在面向对象中，类机制的引入使其具有比过程化编程更好的模块性，因此也更易于被程序员理解和维护。但是如上所述的代码缠结问题的存在，使面向对象技术在易理解性和易维护性方面都难有更大的提高。Kiczales 经过统计发现："如果一个他人写的程序有37处需要改动，对于一个最优秀软件开发人员，也大概只能找到35个"。而对于 AOP，对一个 Aspect 的修改可以通过联结器影响到系统相关的各个部分，从而大大提高系统的易维护性。另外，对系统特征的模块化封装无疑也能提高程序的易理解性。

3.3 AOP 程序设计的一般步骤

基于 AOP 的应用程序结构和传统的高级语言的应用程序结构基本类似。传统的高级语言系统实现由三部分组成：(1)一种语言，(2)特定于这种语言的编译器，(3)利用这种语言写的应用程序。基于 AOP 的系统实现也有三部分，但是可以进一步细化为：(1.1)一种组件语言，(1.2)一种或多种 aspect 语言，(2)一个用来合并前两者的 aspect 联结器(weaver)，(3.1)利用组件语言实现的系统组件，(3.2)利用 aspect 实现的 aspect 语言。

因此一个基于 AOP 的应用可以分以下几步来实现：

1)确定哪些功能是组件语言必须实现的，哪些功能可以以 aspect 的形式动态加入到系统组件中去。

2)构造系统的组件。此处所指的系统的组件，是实现该系统的基本模块，对于 OOP 语言，这些组件可以是类，对于过程化程序设计语言，这些组件可以是各种函数和 API。

3)构造系统的 aspect。

4)用联结器将组件代码和 aspect 代码进行组合。

上述过程如图2所示。

3.4 代码联结

代码联结是指对输入的组件语言和 aspect，根据联结点的语法定义，生成相应的中间文件或目标代码。这个过程可以分成三个阶段进行。首先，为组件语言和 Aspect 语言构造相应的语法树；然后依据 aspect 中的联结点定义对语法树进行联结；最后在联结的语法树上生成中间文件或目标代码。

3.5 AOP 的应用实例

目前已有许多研究机构将 AOP 思想结合到实际应用中，下面是几个典型的例子：

1)a-kernel　加拿大大不列颠哥伦比亚大学的 a-kernel 项目组将 AOP 思想引入到操作系统设计中，试图提高操作系统代码的模块性。他们利用 AspectC，对 FreeBSD v3.3操作系统内核作了修改，将操作系统的内存缺页处理作为一个 aspect 进行处理，收到了很好的效果[6]。

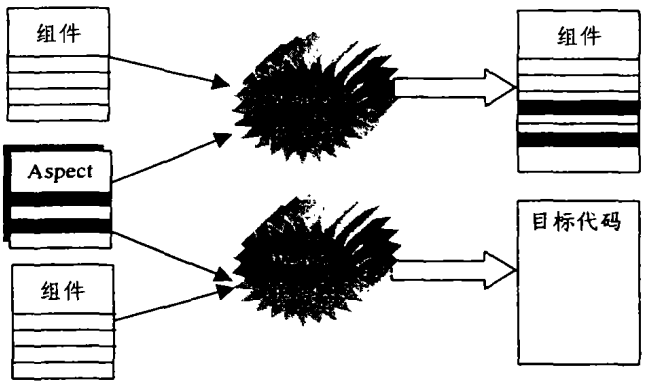

图2　AOP 编程模型

2)Lasagne　比利时 Katholieke 大学的 Lasagne 是一种基于 aspect 的中间件，主要用于分布式系统的服务定制。采用 Aspect 思想，Lasagne 可以在不改变代码的情况下，实现动态程序扩充。服务作为一种 aspect 以 Wrapper 的形式存在，利用运行时联结思想在运行时被动态选取[7]。

3)FACET　华盛顿大学的 FACET(Framework for Aspect Composition for an EvenT channel)项目组利用 AOP 方法来实现可定制中间件。他们使用 AspectJ 开发了一个实时事件信道，该信道和 TAO 实时事件信道相比，具有更好的模块性、更少的代码以及简单和易于扩展等特点[8]。

4. AOP 的发展现状

从1997年正式被提出至今，AOP 技术得到了很大的发展。作为一种编程方法学，目前已经出现了许多支持语言和工具；面向特征(Aspect-Oriented)作为一种新的思想也开始渗入到计算机应用的各个领域。

4.1 AspectJ

AspectJ 是目前较为成熟和流行的 AOP 语言，它是标准

Java 语言的扩展。AspectJ 采用四种语言单位来支持 AOP：Pointcut、advice、code introduction 以及 Aspect。

a)Pointcut：是 Aspect 在系统中的联结点，如对象生成点，方法入口点等，aspect 通过在这些良好定义(well-defined)的点上对系统产生影响。

b)Advice：定义在 pointcut 点所要完成的功能。根据不同的语法定义，这些功能可以在 pointcut 之前或之后执行，也可以替换 pointcut 执行。

c)Code Introduction：在程序中的原有的类添加新的变量和方法。

d)Aspect：一种类似于类的语言单位，是包含一定功能的类型化实体，用于实现对横切于系统的系统特征的封装，由 pointcut，advice 和 code introduction 组成。

图3是一个 Aspect 的例子。在该 aspect 中，首先为系统的每个类添加一个 flag 属性并提供设置方法，可以由此来确定是否需要为这个类添加异常处理功能；Pointcut 声明被检测类的每一个方法的执行都要被检查；在 advice 中简单地将捕获的异常进行打印，AspectJ 的具体语法参见文[5,11]。

4.2 其它语言和工具

除了 AspectJ，目前已经出现或正在进行开发的支持语言主要包括 AspectC，AspectC++，AspectS(for smalltalk)等。

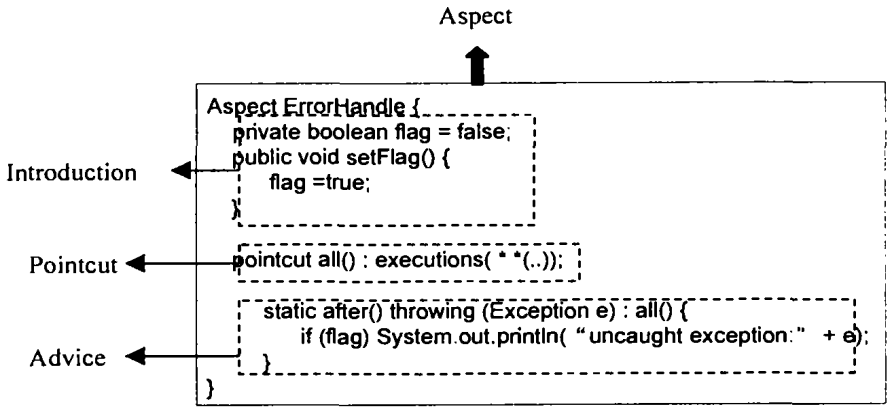

图3 一个 Aspect 实例

IBM 的 MDSOC(Multi-Dimensional Separation of Concerns，关注点多维分解)也是 AOP 领域目前的一个研究热点。利用该研究小组的 Hyper/J 工具，开发人员可以认定和封装各种关注点，认定和管理关注点直接的关系，还可以重新组合关注点来生成新的软件系统[9]。

此外，英国兰开斯特大学开发了一个名为 SADES 的对象数据库进化系统，它利用了 AOP 的基本思想，将横切于数据的特征进行分离，从而支持面向特征应用系统的特征持久性模型，并可以将不同面向特征技术的特征映射到一个公共持久性表示中去[10]。

4.3 现有的问题

作为一种新的编程方法，AOP 的广泛应用还有待于更多工作的完成。支持的语言需要进一步丰富并保证其正确性；必须有更多的工具支持 AOP，满足其从软件设计到维护各个阶段的需要；还必须有一套行之有效的方法来培养一批能充分利用 AOP 各种优点的程序员。

结束语 面向特征编程方法实现了从更高层次上对软件系统的抽象，有利于软件的开发、维护，在软件规模日益庞大，软件结构日益复杂的今天，必将发挥越来越重要的作用。

参 考 文 献

1 The Emerging Technologies That Will Change the World. MIT Technology Review January/February 2001 issue

2 Bertrand Meyer. Object-Oriented Software Consstuction

3 Kiczales G, et al. Aspect-Oriented Programming. In: Proc. of the European Conf. on Object-Oriented Programming (ECOOP), June 1997

4 Miller S K. Aspect-Oriented Programming Takes Aim at Software Complexity. Computer 2001 IEEE Vol. 34, No. 4

5 The AspectJ team. Aspect-Oriented Programming with AspectJ. Available at: http://aspect.org

6 Coady Y, et al. Exploring an Aspect-Oriented Approach to OS Code. Available at : http://WWW.cs.ubc.ca/~ ycoady/akernel.html

7 Truyen E, et al. Dynamic and Selective Combination of Extensions in Component-based Applications. In: Proc. of the 23rd Intl. Conf. on Software Engineering (ICSE'2001), May 2001, Toronto, Canada

8 Hunleth F, Cytron R, Gill C. Building Customizable Middleware using Aspect Oriented Programming. In: Proc. of Conf. on Object-Oriented Programming, Systems, Languages, and Applications 2001(OOPSLA2001)

9 Ossher H, Tarr P. Multi-Dimensional Separation of Concerns using Hyperspaces: [IBM Research Report 21452]. April, 1999

10 Rashid A, Sawyer P. Object Database Evolution using Separation of Concerns. ACM SIGMOD Record, 2000, 29(4)

11 The AspectJ team. The AspectJ programming guide. Available at: http://aspect.org