基于面向方面的软件重用模式设计

毛 凯
(重庆工商大学计算机科学与信息工程学院 重庆 400067)

摘 要 针对传统的软件重用设计模式,分析了在采用当前流行的面向对象开发方法进行软件重用设计的不足之处,阐述了面向方面的程序设计理论,提出了一种新型的面向方面软件重用模式设计方法,并结合实际应用给出了具体的设计步骤以及面向方面的程序代码架构。

关键词 面向方面,重用模式,设计模式

Software Reused Model Design Based on Aspect-oriented

MAO Kai
(College of Computer Science and Information Engineering, Chongqing Technology and Business University, Chongqing 400067)

Abstract Aimed at traditional software reused model, this paper analyses the shortage of current popular objected-oriented development method on software reused design, and explains design theory of aspect-oriented proramming, and provides a new software design method on aspect-oriented. And then the paper gives design of a concrete practical application and aspect-oriented framework of program code.

Keywords Aspect-oriented, Reused model, Design model

在软件工程领域由于广泛地采用面向对象的开发方法,已经引发了一场深刻的技术变革。通过面向对象的分析和设计为我们提供了各种开发方法和工具,引入诸如封装、继承、抽象和多态的概念来减少软件的复杂性。面向对象开发技术主导软件开发领域以后,它的优势和劣势也逐渐显现出来,虽然它是编程技术革新的先锋,但是在软件重用设计方面仍存在一些无法克服的问题和障碍。这主要表现在:

1. 代码交织:当一个模块或代码片段同时管理多项重用的功能易发生代码交织情况;

2. 代码分散:当一个重用的功能扩散到许多模块,并且没有很好地进行局部化和模块化时易发生代码分散情况。

如在构建带有大量 Java 类的 Web 应用程序,我们就需要对异常处理功能进行重用设计,其重用模块程序代码如下面的定义形式:

```
try{
    // Code that throws Exception
    } catch (Exception e){
    System. out. println("Exception:
    " e. getMessage());
}
```

按照面向对象设计方法针对各个重用功能编写的重用代码片段是无法将开发人员从一遍遍重写应用程序的烦琐中解脱出来。正是基于这一原因,本文按照面向方面的程序设计理论,提出了一种面向方面的程序重用设计模式,期望通过新的程序重用设计模式来解决面向对象设计方法所不能够解决的代码混乱与分散的软件重用问题。克服对象和组件技术的局限性,使系统拥有更好的可维护性和可扩展性。

面向方面(Aspect-oriented Programming, AOP)编程是一种新的编程技术,从一定的角度来说也是面向对象开发技术的延伸。它结合了方面(aspect)和现有代码,为应用程序添加新的行为,模块化横切系统或关注点(感兴趣或有需求的领域)。这些模块称为"方面"(Aspect),把切点(Pointcut)及建议(Advice)关联起来。其中切点就是一系列连接点(Joint-point)的集合,而连接点是执行程序流中定义良好的点(如方法调用),建议是与特定的切点有关,用于指定代码在一个连接点处执行。面向方面编程就是通过切点声明了一个事件,通过建议声明触发事件时要执行的一个动作。在实际开发设计过程中,代码编写可以直接表示并组织横切关注点,将横切关注点的实现代码分离出来,并模块化成为"方面",然后将其组合到各个类中。

单独来说,软件重用技术和面向方面开发技术都有着共同一致的目标,即创建易于理解和维护,同时无需付出很大开发代价的软件系统。采用面向方面的重用技术是一种全新的模块性构造技术,超出了传统意义上的重用技术并结合了两者的共同点,帮助重组对应于重用功能的代码,以便进一步提高模块化程度,并根除常见的问题症状:代码交织(Code-tangling)和代码分散(Code-scattering)。

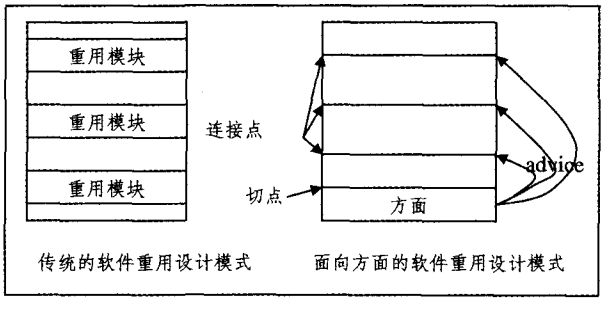

图 1 传统的软件重用设计模式与面向方面的软件重用设计模式的比较

传统的重用技术是把代码模块转化为一种基于组件的面

毛 凯 讲师,硕士,主要从事信息系统和网络数据库系统研究。

向对象代码模块来实现，从本质上说，就是将需求分配到类、包、服务等组件中。尽管许多需求能够有效地局部化于单个组件对象，但也有一些需求无法局部化于单个组件对象，甚至有时会影响到全系统。而面向方面开发技术则可排出那些不用于重用的代码，可以通过更少的代码来执行相同的功能，从而显示出面向方面编程的优势，即程序易于理解，代码紧凑，而且修改起来相当简单，二者的比较如图 1 所示。

下面给出面向方面的软件重用模式设计方法的具体步骤。

1. 分离切点

系统进行需求分析时结构化各个模块，用于识别、捕获切点，并在整个设计和编码过程中保持分离，一般可将系统划分为两个层次：应用层和领域层，通过用例加以实现。其中应用层主要包括系统参与者的工作流，并通过领域层中的元素来实现用例，涉及到的类有一个或多个特定的参与者类、一个或多个特定的用例类、某些功能区域的类等。领域层主要包括有描述重要领域概念的元素，涉及到系统中需要维护、跟踪或者控制的信息以及产生这些信息的相关行为。这些元素通常在用例实现中被共享和重用，因此处于比应用层更低的层中。系统用例可以从〈执行事务〉用例开始，包括处理表示(Handle Presentation)模板用例、处理分布式(Handle Distribution)模板用例、处理持久化(Handle Persistence)模板用例等各个基础用例。整个系统分析的层次结构模型如图 2 所示。

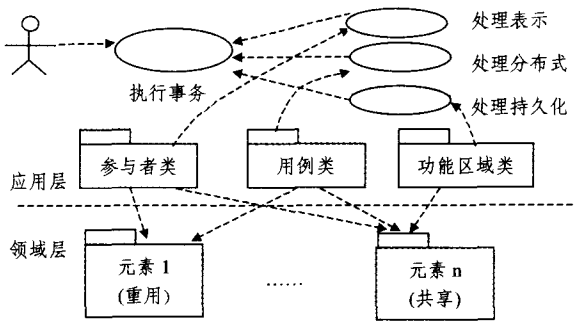

图 2　系统分析的结构层次模型

2. 定义切点

定义一个切点来捕捉重用后功能的连接点，其中定义切点的名称应符合命名切点关键字的要求。

如在对异常处理功能进行面向方面的重用设计时，可定义一个 exceptionHandler 的切点，该切点指定一个调用 Exception 的异常处理程序或者它的子类连接点，这个定义在程序中指定一个可被建议引用的命名连接点。

```
pointcut exceptionHandler(Exception e);
Handler(Exception+)&& args(e);
```

3. 为切点定义建议

根据横切逻辑所需的位置，使用一种正确的建议形式(如 before、after)，为切点定义建议。

如上例在对异常处理功能 exceptionHandler 切点定义建议时，可引用 exceptionHandler 切点的名称，编写异常处理程序，即对异常编写日志消息的代码块。

```
After(Exception e): exceptionHandler(e){
Log. error("Exception:"+e. toString);
}
```

4. 创建方面，将类方法和切点集成起来

通过面向方面编程定义横切行为，创建一个方面，把类的方法和切点集成起来，并将其封装到该单独的方面里面，为应用程序添加新的行为，其中该行为即可为重用功能。创建方面时需要考虑类的方法和切点之间如何交互来实现用例，聚焦于理解交互过程，从中寻找类的方法和切点之间相互的职责。

如上例对异常处理功能可创建一个相应的 LogPoints 方面，同时为该方面添加切点和程序代码，假定该应用程序的主程序体代码为 main_programme_code，其中包含有 method_1()到 method_n()共 n 个方法，可以通过使用一个日志记录接口，在执行相应的方法前后以及调用 Exception 或其任意子类之一的异常处理程序时，创建对异常编写日志消息的日志功能项，从而为 LogPoints 方面在应用程序添加新的重用行为功能，完成面向方面的软件重用模式设计的要求。所创建方面的结构如图 3 所示。

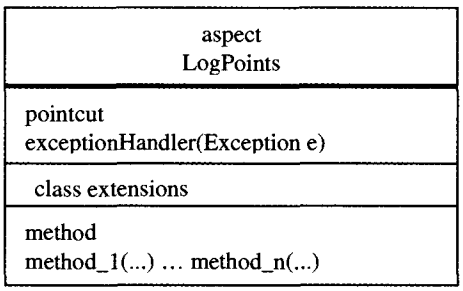

图 3　方面的结构层次模型

其程序框架代码如下：

```
import org. apache. commons. logging. Log;
import org. apache. commons. logging. LogFactory;
aspect LogPoints{
private static Log log=LogFactory. getLog
(main_programme_code);
   before(): call(public * method_1(…)){
   log. info("Enter:"+ exceptionHandler(Exception e));
   }
   after(): call(public * method_1(…)){
   log. info("Exit:"+ exceptionHandler(Exception e));
   …
   before(): call(public * method_n(…)){
   log. info("Enter:"+ exceptionHandler(Exception e));
   }
   after(): call(public * method_n(…)){
   log. info("Exit:"+ exceptionHandler(Exception e));
   }
   //The Exception handler pointcut and
   pointcut exceptionHandler(Exception e):
        handler(Exception+)&& args(e);
   after(Exception e): exceptionHandler(e){
        log. error("Exception:"+e. toString());
   }
}
```

对开发效率的影响

采用面向方面的软件重用模式能够带来巨大的开发效率的提高。为了证明这个观点，我们来看一个包括 N 个需求的系统。假设这 N 个需求是独立的，而每个需求的工作量为 X，那么开发 N 个需求的工作量就是 NX。

假设这 N 个需求并未采用面向方面的重用模式开发，这就意味着每个需求的实现将与其它需求的实现相互交迭。而最差的情况就是每个需求的实现都与 $N-1$ 个需求(也就是其它所有的需求)实现相关，则需要集成交迭部分的工作量为 Y，那么开发每个需求的工作量就变成了 $X+(N-1)Y$，因此完成这个任务所需的工作量就是 $NX+N(N-1)Y$。

对采用和未采用面向方面的重用模式开发进行比较，如图 4 所示，我们假定 $X=20$，$Y=1$，也就是每个需求所需时间是每个集成所需时间的 20 倍。

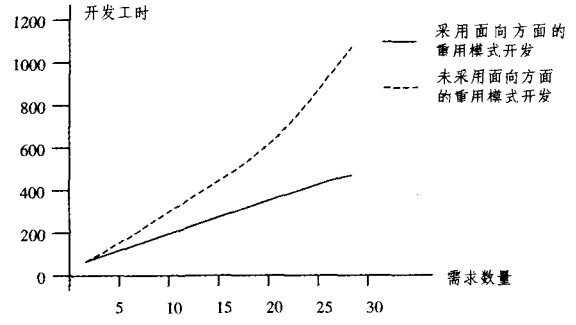

图 4　采用和未采用面向方面的重用模式开发比较

从这个简单的计算中，就可以发现通过采用面向方面的重用模式开发，能够大大增加软件的开发效率，从而达到事半功倍的效果。

参 考 文 献

1　GHEZZI C. 软件工程基础[M]. 北京：清华大学出版社，2003
2　杨雪涛. 基于 UML 的软件开发模型[J]. 重庆工商大学学报(自然科学版)，2004，21(6)：580～582
3　BUDD T. 面向对象的 JAVA 编程思想[M]. 北京：清华大学出版社，2002
4　Kurt Bittner 和 Ian Spence. 用例建模[M]. Addison-Wesley，Boston，2002

(上接第 248 页)

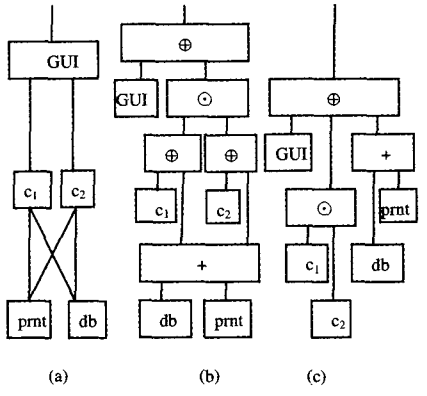

图 2　系统组合模型图

需求、特征、特征迹、方法、构件、连接件和 SA 之间的关系。对于每一个变化的需求，可以根据与特征迹相关连的构件和连接件，特别是增加一个新的需求时，可能需要插入新的构件，特征迹有利于找到这个新的构件应该挂接的连接件。

设 $Req=\{r_1, r_2, \cdots, r_n\}$ 是可变化的需求集合，并假定每一个需求对应唯一一条特征迹。下面给出基于领域的 SA 可演化性度量方法：

(1) $\forall r_i \in Req$，确定特征迹 tr_i，$1 \leqslant i \leqslant n$，

(2)确定 tr_i 经过的构件 $coms=\{c_{i1}, c_{i2}, \cdots, c_{ik1}\}$ 和连接件 $conns=\{conn_{i1}, conn_{i2}, \cdots, conni_{k2}\}$，

(3) $\forall c \in coms$，计算 $evol_{del}(c)$ 和 $evol_{rep}(c)$，$\forall conn \in conns$，计算 $evol_{ins}(conn)$，

(4)计算特征迹 tr_i 的可演化性 $evol(tr_i)$：

tr_i 的可插入性 $evol_{ins}(tr_i)=(evol_{ins}(cnn_{i1})+evol_{ins}(cnn_{i2})+\cdots+evol_{ins}(cnn_{ik2}))/k_{k_2}$，

tr_i 的可删除性 $evol_{del}(tr_i)=(evol_{del}(c_{i1})+evol_{del}(c_{i2})+\cdots+evol_{del}(c_{ik1}))/k_1$，

tr_i 的可修改性 $evol_{rep}(tr_i)=(evol_{rep}(c_{i1})+evol_{rep}(c_{i2})+\cdots+evol_{rep}(c_{ik1}))/k_1$，

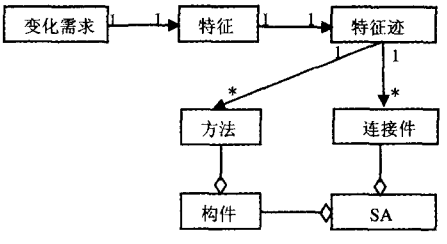

图 3　需求与 SA 和构件的关系模型

tr_i 的可演化性 $evol(tr_i)=(evol_{ins}(tr_i)+evol_{del}(tr_i)+evol_{rep}(tr_i))/3$，

(5)计算软件体系结构 sa 的可演化性 $evol(sa)=(evol(tr_1)+evol(tr_2)+\cdots+evol(tr_n))/n$。

总结　基于构件组合运算的 SA 具有层次性，构件操作对 SA 的影响，或者波及到其上层的使用连接件，或者波及到 SA 的顶层，这为计算构件操作的波及效应提供了可能；基于构件组合运算的构件操作，保证了组合构件演化的一致性。根据构件操作影响到的连接件数目来度量 SA 可演化性，具有较好的直观性；示例中应用度量算法的结果与直观结果一致。根据领域工程，区分基本需求和可变需求，利用特征迹，可以更精确地度量 SA 的可演化性，也可根据 SA 可演化性度量方法来判断 SA 对一次需求变化的适应性。本文假定了构件操作必须保持构件演化特性，并且演化扩展了构件接口，而实际中存在接口不变的情况，这只需要对算法进行适当修改。

参 考 文 献

1　Cook S, He J, Harrison R. Dynamic and static views of software evolution. In: Proc. of the Int'l Conf. of Software Maintenance, IEEE, 2001. 592～601
2　赵会群，王国仁，高远. 软件体系结构抽象模型[J]. 软件学报，2002，25(7)：730～736
3　任洪敏，钱乐秋. 构件组装及其形式化推导研究[J]. 软件学报，2003，14(6)：1077～1074
4　王映辉，张世琨，刘瑜，等. 基于可达矩阵的软件体系结构演化波及效应分析[J]. 软件学报，2004，15(08)：1000～9825
5　Taylor R, Medvidovic N, Anderson K. Acomponent- and message-based architectural style for GUI software[J]. IEEE Transactions on Software Engineering, 1996, 22(6): 390～406
6　张友生. 构件运算与软件演化研究[J]. 计算机应用，2004，4(24)：20～24
7　Mens T, Wermelinger M. Challenges in Software Evolution. In: Proceedings of the 2005 Eighth International WorkshoP on Principles of Software Evolution (IWPSE'05), IEEE, 2005
8　Sadou N, Tamzalit D, Oussalah M. A unified Approach for Software Architecture Evolution at different abstraction levels. In: Proceedings of the 2005 Eighth International Workshop on Principles of Software Evolution (IWPSE'05)
9　梅宏，申峻嵘. 软件体系结构研究进展[J]. 软件学报，2006，6(17)：1257～1275
10　Wang Yingxu. A New Mathematiccal Notation for Describing Notion and Thought in Software Design. In: Proceedings of the First IEEE International Conference on Cognitive Informatics (ICCI'02)
11　Stoermer C, Bachmann F. SACAM: The software architecture comparison analysis method: [Tech Rep]. CMU Software Engineering Institute, CMU/SEI-2003-TR2006 ESC-TR-2003-006, 2003
12　Burd E, Munro M. An Initial Approach towards Measuring and Characterising Software Evolution
13　Shaffer C A. A practical Introduction to Data Structures and Algorithm Analysis. Prentice Hall, 2001
14　张世琨，王立福，常欣，等. 基于层次消息总线的软件体系结构描述语言[J]. 电子学报，2001，5(29)：581～584
15　覃国蓉，张世琨. 基于层次消息总线的软件构架动态模拟和演化研究[J]. 计算机科学，2001，28(93)：75～77
16　Greevy O, Ducasse S. Correlating Features and Code Using A Compact Two-Sided Trace Analysis Approach. In: Proceedings of CSMR 2005 (9th European Conference on Software Maintenance and Reengineering). 314～323