软件工程系列教材

面向对象技术UML教程

UML

http://www.tup.com.cn

王少锋 编著

清华大学出版社

软件工程系列

面向对象技术UML教程

王少锋 编著

清华大学出版社
北京

内容简介

本书主要介绍统一建模语言 UML 及其应用。全书内容丰富，包括 UML 的用例图、顺序图、协作图、类图、对象图、状态图、活动图、构件图和部署图等 9 个图中所涉及的术语、规则和应用，以及数据建模、OCL、业务建模、Web 建模、设计模式、OO 实现语言、RUP 等方面的内容，同时介绍了 Rose 开发工具中的一些用法。本书最后是一个课程注册系统的实例研究，以及一些思考题和设计题。附录中是两套模拟试题及答案，模拟试题中的题目可以作为 UML 应用的实例，完成这些练习题可以使读者加深对 UML 的认识。

本书可作为大专院校计算机软件专业研究生和高年级本科生学习 UML 和面向对象技术的教材，也可作为广大软件开发人员自学 UML 和面向对象技术的参考书。

图书在版编目(CIP)数据

面向对象技术 UML 教程/王少锋编著. —北京：清华大学出版社，2004.1
(软件工程系列教材)
ISBN 7-302-07740-1

Ⅰ. 面… Ⅱ. 王… Ⅲ. 面向对象语言，UML—程序设计—教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字 (2003) 第 112445 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
http://www.tup.com.cn　　邮　　编：100084
社 总 机：010-62770175　　客户服务：010-62776969
责任编辑：刘映欣
印 刷 者：清华大学印刷厂
装 订 者：三河市新茂装订有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×260　印张：17.5　字数：399 千字
版　　次：2004 年 2 月第 1 版　2004 年 2 月第 1 次印刷
书　　号：ISBN 7-302-07740-1/TP・5657
印　　数：1～5000
定　　价：24.00 元

前　言

本书以介绍面向对象的统一建模语言 UML 为主，目的是使读者了解面向对象技术的基本概念，掌握面向对象的分析和设计方法，以及与面向对象技术相关的一些软件开发技术，同时掌握在 Rose 环境下用 UML 进行分析和设计的技术。

UML 是由著名的面向对象技术专家 Grady Booch、James Rumbaugh 和 Ivar Jacobson 在各自方法的基础上，汲取其他面向对象方法的优点，广泛征求意见，几经修改完成的一种建模语言。UML 是一种定义良好、易于表达、功能强大且适用于各种应用领域的建模语言，已被 OMG 采纳为标准，目前 UML 已成为面向对象技术领域内占主导地位的标准建模语言。掌握 UML 语言，不仅有助于理解面向对象的分析与设计方法，也有助于对软件开发全过程的理解。

本书包括 19 章的内容及附录。第 1 章面向对象技术概述主要是对所涉及的软件工程中一些知识点的回顾；第 2 章是 UML 概述，目的是让读者对 UML 有一个总体的印象；第 3 章讲述用例和用例图，用例是 UML 中一个非常重要的概念，用例驱动的软件开发方法已得到广泛的认同；第 4 章讲述 UML 的动态建模机制中所用到的两个图，即顺序图和协作图；第 5 章讲述类图和对象图，UML 中的类图具有很充分的表达能力和丰富的语义，是建模时非常重要的图；第 6 章讲述数据建模的概念，任何稍具规模的系统都会涉及数据库设计的问题，数据建模可以看作类图的一个应用；第 7 章讲述 UML 中的包，包是一种很有用的建模机制，除了在 OO 分析设计中对建模元素进行分组外，在数据建模、Web 建模、支持团队开发等方面有不可替代的作用；第 8 章讲述状态图和活动图，状态图和活动图都用于对系统的动态行为建模；第 9 章和第 10 章分别讲述 UML 中对 OO 系统的物理方面建模的两个图，即构件图和部署图；第 11 章讲述对象约束语言 OCL，OCL 已成为 UML 规范说明的一部分，它可以表示施加于模型元素本身或模型元素的属性、操作等的约束条件，用于表示一些用图形符号很难表示的细微的意义；第 12 章讲述业务建模，对一个机构的业务过程进行建模可以更好地理解、分析、改善和替换机构中的业务过程，同时也可以作为软件系统开发的基础，使得软件系统能更好地支持机构中的业务过程，可以把业务建模看作 UML 的一个具体应用；第 13 章讲述 Web 建模，主要介绍如何用 UML 中的扩展机制对 Web 应用系统建模；第 14 章讲述设计模式，在软件设计过程中，设计模式的作用非常大，本章介绍了 3 个设计模式，可使读者对设计模式有一定的了解；第 15 章讲述面向对象的实现技术，主要介绍一些对 OO 技术影响比较大、或本身很有特色的 OO 语言；第 16 章讲述 RUP 软件开发过程，UML 是一个建模语言，它需要在具体的软件开发过程中使用，RUP 总结了一些好的开发经验，学习 RUP 有助于理解软件开发过程；第 17 章讲述与 UML 有关的一些开发工具及其使用；第 18 章是一个课程注册系统的例子，本章对这个例子的模型内部结构做了分析；第 19 章是一些综合练习。书末附录是两套模拟试题及答案，对于部分答案给出了解释，模拟试题中的一些题目可以

作为 UML 应用的实例，完成这些题目会使读者加深对 UML 的认识。

本书的主要内容曾作为清华大学计算机系研究生（一届）和软件学院研究生（三届）“面向对象技术与应用”课程的讲义，本书是在讲义的基础上做了进一步的丰富和完善后完成的，希望本书能够帮助读者全面、细致地了解 UML 及其应用。在写作过程中，作者对书中的内容反复多次修改，以求尽量减少错误，但由于时间关系，加之 UML 涉及的内容非常广泛，在编写过程中难免会有各种错误和疏漏，敬请广大读者批评指正。作者 email 信箱为：wangsf@tsinghua.edu.cn。

王少锋

2003 年 12 月于清华大学

目　录

第 1 章　面向对象技术概述

1.1　软件危机及软件工程

20 世纪 60 年代中期开始爆发的软件危机，使人们认识到大中型软件系统与小型软件有本质的不同：大型软件系统的开发周期长、开发费用昂贵、开发出来的软件质量难以保证、开发生产率低，它们的复杂性已远远超出人脑所能直接控制的程度。就像用制造小木船的方法不能生产航空母舰一样，大型软件系统的开发不能再沿袭早期手工作坊式的开发方式，而必须立足于科学的理论基础上，实行大兵团式的工程化作业，这一认识导致了软件工程学的诞生。1968 年，北大西洋公约组织(NATO)科技委员会在当时的联邦德国 Garmisch 召开了有近五十名一流的计算机科学家、编程人员和工业界人士参加的研讨会，商讨摆脱软件危机的办法，在这次会议上第一次提出了软件工程的概念，这是软件开发史上重要的里程碑，它标志着软件开发进入了划时代的新阶段。

经过三十多年的探索和发展，软件工程这门学科有了长足进展，但软件危机依然存在，而且有越来越严重的趋势。大量事实说明，软件的质量和生产率问题远没有得到解决，与三十多年前的软件相比，现在的软件在规模、复杂性等方面远远超过以前的软件，大型软件开发中许多问题，如开发效率低、产品质量差、产品难以维护、软件可移植性差、开发费用超过预算、开发时间超期等依然存在。一般说来，软件项目越大，情况越坏，所有的大型系统中，大约有四分之三的系统有运行问题，要么不像预料的那样起作用，要么根本就不能使用。如美国丹佛新国际机场自动化行李系统软件投资 1.93 亿美元，原计划在 1993 年万圣节前启用，但系统开发人员一直为系统中的错误所困扰，推迟到 12 月，系统仍无法交付使用。为了排除系统中存在的故障，一直拖延到 1994 年 3 月，最后到 6 月份，机场的计划者承认，他们无法预测行李系统何时能启用[YYW97]。

就国内外软件开发现状而言，对于小型软件系统，有比较好的开发方法，成功率也较高，但对于中大型软件系统的开发，情况则不尽如人意，在开发效率、开发费用、产品质量等重要方面都不能令人满意。

针对大型软件系统开发中存在的问题，人们提出了各种各样的软件开发方法，如瀑布式软件开发方法、快速原型方法、螺旋式软件开发方法、变换式软件开发方法、增量式软件开发方法、净室(cleanroom)软件开发方法、喷泉式软件开发方法等。但这些方法并未完全解决软件危机的问题，都存在这样或那样的问题，软件危机依然存在。

1.2 对软件开发的基本认识

大型软件系统的特点是：开发代价高，开发时间长，参加开发的人员多，软件生命周期长。采用传统的软件工程方法开发大型软件存在开发效率低、产品质量差、产品难以维护、软件可移植性差、软件可重用性低等问题。

一个软件系统的开发可以从两个方面进行刻画，一方面是软件开发过程，从软件需求、总体设计、详细设计、代码实现、测试到最终产品的提交，以及后期的软件维护及再开发过程，这方面要求软件开发具有连续性，开发各阶段得到的产品要求在逻辑上相互一致；另一方面是软件开发过程中所涉及的各种资源，它们包括参与开发的各种工作人员、硬件资源和软件资源，这些资源在使用过程中需要进行协调和管理。正是这两个方面之间的相互作用，形成了完整的软件开发活动。目前软件开发中存在的问题，究其原因，往往是由于在这两个方面上控制不当，或协调不一致造成的。

软件工程的目的就是要在规定的时间、规定的开发费用内开发出满足用户需求的高质量的软件系统。这里所说的高质量不仅是指错误率低，还包括好用、易用、可移植、易维护等要求。当初提出软件工程就是希望采用工程的概念、原理、技术和方法，把经过时间考验而证明有效的管理技术和当前能够得到的最好的技术方法结合起来，以指导计算机软件的开发和维护。

为了深入理解软件工程，有必要探讨软件的特点。软件是一个逻辑部件，而不是一个物理部件，所以软件具有与硬件不同的特点：

- 表现形式不同。硬件属于客观实体，看得见、摸得着，而软件是人的思想的产物，没有颜色、形状，看不见，摸不着，它的正确与否，是好是坏，一般要到软件在计算机上运行后才能知道，这就给软件的开发和管理带来许多困难。
- 生产方式不同。尽管软件开发与硬件制造两者之间有许多共同点，但这两种活动是根本不同的，在硬件制造过程中可能出现的质量问题在软件开发中可能不会出现，反之亦然。这两种活动都依靠人，但人的作用和相互之间的关系是完全不同的。由于软件是逻辑产品，软件的开发和人的智力活动紧密相关，在很多人共同完成一个软件项目时，人与人之间就有一个思想交流的问题，即沟通问题。沟通不但要花费时间，而且由于沟通中的疏忽会使错误出现的可能性增大。
- 产品要求不同。硬件产品允许有误差，生产时，只要达到规定的精度要求就算合格，而软件产品却不允许有误差。因此软件的生产要求有很高的质量保证体系。
- 维护方式不同。硬件在使用过程中，由于磨损、振动、腐蚀、空气污染等原因用旧或用坏，可以使用备用件。而对于软件来说不存在备用件，软件中任何缺陷都会在计算机上导致同样的错误。如果软件在使用过程中发现有缺陷，则需要进行修改。随着某些缺陷的改变，很可能引入一些新的缺陷，因而使软件的故障率增加，品质变坏，所以软件的维护比硬件复杂得多。

1.3　软件的固有复杂性

软件的特点说明了软件开发的复杂性和困难性。著名的计算机专家、被称之为 IBM 360 系列计算机之父的 F. Brooks 认为软件的复杂性是固有的，软件可能是人类所能制造出来的最复杂的实体[Bro87]。导致软件复杂性的原因很多，下面列出一些主要的原因。

首先，软件的复杂性和计算机的体系结构有关。计算机的体系结构从计算机诞生以来，尽管有了很大的进步，如采用流水线、超高速缓冲存储器等，但主要仍然是冯·诺依曼式的，虽然在计算机的发展过程中，也出现过一些新型的体系结构，但这些体系结构并没有获得主导地位。

冯·诺依曼所提出的存储程序方式的计算机体系结构的主要特点是硬件(存储器、运算器和控制器等)简单，而软件却很复杂，所需全部功能由软件来完成。在一些很单纯的应用中(如数值计算)，这种体系结构尚且可行，但在目前需要庞大而复杂的软件系统的情况下，这种体系结构存在难以克服的缺点。如果计算机继续采用冯·诺依曼式的体系结构，则软件的复杂性将很难消除。

其次，软件开发是人的一种智力活动，软件系统从本质上来说是由许多相互联系的概念所组成的结构。这种概念结构很难用一组数学公式或物理定律来描述，也就是说，很难找到一种好的方法或工具来刻画软件系统的内在本质特征或规律。

第三，造成软件系统复杂性的另一个原因是，软件系统中各元素之间的相互作用关系具有不确定性。从理论上讲，任何两个元素之间都可以存在交互关系，几乎不受任何外界因素的限制，而且随着元素数目的增加，元素之间的交互关系呈非线性递增的趋势。

第四，由于软件没有固定的形式与坚硬的外壳，人们普遍认为软件系统是“软”的，似乎可以随意扩充和修改。因此软件系统还面临不断变化的压力，不同的软件系统需要满足不同用户的工作方式和习惯。用户总是尝试用更合理和更方便的方式使用软件，并且希望系统为他们完成更多种类和更大数量的工作。新的功能或变化的功能不断地向软件系统提出新的要求，这种持续的变化又增加了软件系统的复杂性。

第五，规模较大的软件系统的生命周期一般都超过相应硬件系统的生命周期。在此期间，硬件系统可能在不断变化，原有的软件系统将不得不根据实际应用环境的要求随时做出调整与变化，以适应不同的硬件系统，这又给软件系统本身带来许多新的复杂性。

由于软件的固有复杂性，使得开发成员之间的沟通变得困难，开发费用超支，开发时间延期，等等；复杂性也导致产品有缺陷、不易理解、不可靠、难以使用、功能难以扩充，等等。

在一些传统的工程领域，设计人员往往有好的理论帮助其进行设计。如桥梁专家在设计桥梁时有完整的力学理论帮助其进行设计，硬件设计师在设计芯片时有微电子学理论的指导。但对于软件设计人员，几乎没有任何类似的数学或物理理论帮助或制约设计人员对软件系统进行设计，即使存在这种约束，那也是设计人员为了达到控制软件复杂性

的目的，人为地强加给软件系统的。软件开发过程中这种巨大的自由性使得软件系统可以具有极大的无序度，使得软件系统难以理解、认识、掌握与控制。

软件系统复杂到一定程度，人的智力将很难考虑到其中包含的所有问题。尽管可以人为地给软件强加某些约束关系，但毕竟是人为的。软件设计人员所面临问题的复杂性远远超过了设计一座桥梁、设计一个芯片等所面临问题的复杂性，软件设计人员既要为自己建立设计与实现的准则，又要利用这些准则构造符合要求的软件系统，因此所面临的困难比其他设计领域更多。

1.4 控制软件复杂性的基本方法

软件的复杂性不是因为某个软件系统要解决一个特定的复杂问题而偶然产生的，它是大型软件系统的一个固有的本质特征，软件的开发过程必然会受到软件复杂性的影响。

软件的固有复杂性是导致软件开发与维护过程中众多问题的根源，它使得软件开发过程难以控制，造成软件项目的延期及预算超支，达不到预定的设计要求。但由于软件的复杂性是固有的，人们无法彻底消除这些复杂性，因此只能采用控制复杂性的方法，尽量减少软件复杂性对软件开发过程的影响，而分解、抽象、模块化、信息隐蔽等是控制软件复杂性的有效方法。

1. 分解

人类解决复杂问题时普遍采用的一个策略就是“各个击破”，也就是对问题进行分解，然后再分别解决各个子问题。著名的计算机科学家 Parnas 认为，巧妙地分解系统可以有效地划分系统的状态空间，降低软件系统的复杂性所带来的影响[Par72]。对于复杂的软件系统，可以逐步将它分解为越来越小的组成部分，直至不能分解为止。这样就可以使系统的复杂性，在特定的层次与范围内不会超过人的理解能力。UNIX 中的 shell 和管道即是采用分解思想的例子。

2. 抽象

抽象指的是抽取系统中的基本特性而忽略非基本的特性，以便更充分地注意与当前目标有关的方面。现实世界中的大多数系统都有其内在的复杂性，远远超出人当时所能处理的程度。当使用抽象这个概念时，我们承认正在考虑的问题是复杂的，但我们并不打算理解问题的全部，而只是选择其中的主要部分，我们知道这个问题还应包括附加的细节，只是此时不去注意那些细节而已。

Miller 在一篇经典的文献“神奇的数字 7”中提到，人在同一时间里，一般只能集中于 7 项左右的信息，而不受信息的内容、大小等因素的影响[Mil56]。大型软件系统所包含的元素数目远远超过了这一数字。虽然我们仍然受 Miller 规则的限制，但可以利用抽象来克服这一困难。通过忽略系统内许多非本质的细节，仍然有可能理解和控制各种复杂的系统。

一般说来，抽象又可分为过程抽象和数据抽象。

过程抽象是广泛使用的一种抽象形式。任何一个有明确功能的操作都可被使用者作为单个的实体看待，尽管这个操作实际上可能由一系列更低级的操作来完成。在实际应用中，将处理分解成子步骤是对付复杂性的一个基本方法。

数据抽象定义了数据类型和施加于该类型上的操作，并限定了数据类型的值只能通过这些操作来修改和读取。数据抽象是一个强有力的抽象机制，是控制复杂性的一个重要方法。

3. 模块化

Parnas对模块化的原则有精辟的论述[Par76]。一般地，对模块的要求是高内聚(cohesion)、低耦合(coupling)。高内聚指在一个模块中应尽量多地汇集逻辑上相关的计算资源，低耦合指的是模块之间的相互作用应尽量少。

4. 信息隐蔽

也称封装。信息隐蔽的原则是把模块内的实现细节与外界隔离，用户只需知道模块的功能，而不需了解模块的内部细节。即将每个程序的成分隐蔽或封装在一个单一的模块中，定义每一个模块时尽可能少地暴露其内部的处理。信息隐蔽的基本思想是，无论喜欢或不喜欢，我们是生活在一个瞬息万变的环境中，如果将系统中极不稳定的部分封装起来，那么系统不可避免的变化对整体结构的威胁就减少了。

信息隐蔽能帮助人们在开发新系统时减少不必要的工作，将来如果需要对模块进行修改，则只需修改模块的内部结构，其外部接口可以不做变动。信息隐蔽原则提高了软件的可维护性，且模块内的错误不易蔓延到其他模块，极大地降低了模块间的耦合度，是控制软件复杂性的有效手段，现在信息隐蔽原则已成为软件工程学中的一条重要原则。

1.5　面向对象技术

面向对象(object-oriented，OO)技术充分体现了分解、抽象、模块化、信息隐蔽等思想，可以有效地提高软件生产率、缩短软件开发时间、提高软件质量，是控制软件复杂性的有效途径。

传统的结构化方法的着眼点在于一个信息系统需要什么样的方法和处理过程。以过程抽象来对待系统的需求，其主要思想就是对问题进行功能分解，如果分解后得到的功能过大，那么再对这些功能进行分解，直到最后分解得到的功能能比较方便地处理和理解为止。所以结构化方法也称作功能分解法(functional decomposition)。

与传统的结构化软件开发方法相比，面向对象软件开发方法在描述和理解问题域时采用截然不同的方法。其基本思想是，对问题域进行自然分割，以更接近人类思维的方式建立问题域模型，从而使设计出的软件尽可能直接地描述现实世界，具有更好的可维护性，能适应用户需求的变化。