高等院校计算机应用系列教材

UML面向对象设计与分析教程

(第二版)(微课版)

薛均晓　石　磊　李庆宾　主编

清華大學出版社
北　京

内 容 简 介

本书全面讲述面向对象设计与分析技术和统一建模语言(UML)的基本内容和相关知识。全书共分为11章，深入介绍面向对象的基本概念、UML视图、UML模型图、需求分析、静态分析、动态分析、用例图模型、类图和对象图建模、交互模型、行为模型、系统设计模型、软件开发过程等内容。

本书采用微课形式配合视频讲解和实践操作，帮助读者全面了解面向对象设计与分析的理论知识及实践方法，并掌握UML建模工具的使用技巧。本书内容丰富，结构合理，语言简练流畅，示例翔实，适合初学者使用。本书可作为高等院校软件开发技术及相关专业、软件工程专业的教材，也可作为软件系统开发人员的参考资料。

本书配套的电子课件、实例源文件和习题答案可以到http://www.tupwk.com.cn/downpage网站下载，也可以扫描前言中的“配套资源”二维码获取。扫描前言中的“看视频”二维码可以直接观看教学视频。

图书在版编目(CIP)数据

UML 面向对象设计与分析教程：微课版 / 薛均晓，石磊，李庆宾主编 . —2 版 . —北京：清华大学出版社，2024.1

高等院校计算机应用系列教材

ISBN 978-7-302-64496-5

Ⅰ. ①U… Ⅱ. ①薛… ②石… ③李… Ⅲ. ①面向对象语言—程序设计—高等学校—教材 Ⅳ. ①TP312.8

中国国家版本馆 CIP 数据核字 (2023) 第 162405 号

责任编辑：胡辰浩
封面设计：高娟妮
版式设计：孔祥峰
责任校对：成凤进
责任印制：沈　露

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-83470000　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市人民印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**17.5　　**字　　数：**415 千字
版　　次：2020 年 1 月第 1 版　　2024 年 1 月第 2 版　　**印　　次：**2024 年 1 月第 1 次印刷
定　　价：79.00 元

产品编号：099643-01

前言

自20世纪40年代计算机问世以来，计算机在人类社会的各个领域得到了广泛应用。为了解决计算机软件开发的低效率，以及传统过程式编程方法在处理复杂问题时所遇到的难维护、重用性差等问题，计算机业界提出了软件工程的思想和方法。面向对象技术是一种系统开发方法，是软件工程学的一个重要分支。面向对象设计与分析是使用现实世界的概念模型来思考问题的一种方法。对于理解问题、与应用领域专家交流、建模企业级应用、编写文档、设计程序和数据库来说，面向对象模型都非常有用。

统一建模语言(unified modeling language，UML)是一种功能强大且普遍适用的面向对象建模语言。它融入了软件工程领域的新思想、新方法和新技术。它的作用域不限于支持面向对象分析与设计，还支持从需求分析开始的软件开发全过程。

UML的应用贯穿于软件开发的五个阶段。

- 需求分析阶段。UML 的用例视图可以表示客户的需求。通过用例建模，可以对外部的角色以及它们所需要的系统功能建模。
- 分析阶段。分析阶段主要考虑所要解决的问题，可用UML的逻辑视图和动态视图来描述。
- 设计阶段。在设计阶段，把分析阶段的结果扩展成技术解决方案。加入新的类来提供技术基础结构，如用户界面、数据库操作等。分析阶段的领域问题类被嵌入这个技术基础结构中。
- 构造阶段。在构造(或程序设计)阶段，把设计阶段的类转换成某种面向对象程序设计语言的代码。
- 测试阶段。不同的测试小组使用不同的UML图作为其工作的基础：单元测试使用类图和类的规格说明，集成测试典型地使用组件图和协作图，而系统测试通过实现用例图来确认系统的行为符合这些用例图中的定义。

UML模型在面向对象软件开发中的使用非常普遍。本书全面讲述面向对象设计与分析技术UML的相关知识，主要内容包括面向对象的基本概念、UML视图、UML模型图、需求分析、静态分析、动态分析、系统设计模型及软件开发过程等，并且运用大量实例对各种关键技术进行深入浅出的分析。从相关内容中，读者能感受到UML在描述软件系统方法方面十分有效，以及使用UML建模工具开发面向对象设计与分析模型的便捷性和高效性。为了提高学习效率，在每一章的末尾还提供了一定数量的思考练习题。

本书采用微课形式配合视频讲解和实践操作，帮助读者全面了解面向对象设计与分析的理论知识及实践方法。全书由具体案例贯穿始终，并由案例引入相关的操作和模型创建过程。同时，本书在讲解相关概念时，列举了大量实例。利用这些实例，读者可以更快地掌握UML的基

本元素和建模技巧，也能让读者更好地理解面向对象技术的基本原理。

本书主要针对面向对象技术的初学者，适合作为高等院校软件开发技术及相关专业、软件工程专业的教材，也可作为软件系统开发人员的参考资料。

由于作者水平有限，本书难免有不足之处，欢迎广大读者批评指正。我们的电子邮箱是992116@qq.com，电话是010-62796045。

本书配套的电子课件、实例源文件和习题答案可以到http://www.tupwk.com.cn/downpage网站下载，也可以扫描下方的“配套资源”二维码获取。扫描下方的“看视频”二维码可以直接观看教学视频。

扫描下载

配套资源

扫一扫

看视频

作　者

2023年10月

目录

第 1 章

面向对象与 UML

面向对象是一种软件系统开发方法。本章通过介绍面向对象的主要概念，帮助读者理解面向对象开发方法的本质。同时，通过介绍面向对象软件开发过程、UML与面向对象软件开发的关系、UML建模工具等内容，帮助读者理解面向对象分析和设计建模的含义及目的，为学习UML面向对象设计与分析打下基础。

本章的学习目标：

- 理解面向对象的含义
- 理解软件工程过程框架
- 掌握类和对象的关系
- 掌握封装、继承、多态
- 掌握UML的含义和特点

1.1 面向对象介绍

自20世纪40年代计算机问世以来，计算机在人类社会的各个领域得到了广泛应用。20世纪60年代以前，计算机刚刚投入实际使用，软件设计往往只是为了特定的应用而在指定的计算机上设计和编制，软件的规模比较小，文档资料通常也不存在，很少使用系统化的开发方法，设计软件往往等同于编制程序，基本上是个人设计、个人使用、个人操作、自给自足的私人化的软件生产方式。到了20世纪60年代中期，随着计算机性能的提高，计算机的应用范围迅速扩大，软件开发急剧增长。软件系统的规模越来越大，复杂程度越来越高，软件可靠性问题也越来越突出。原来的个人设计、个人使用的方式不再能满足要求，迫切需要改变软件生产方式，提高软件生产率。

为了解决长期以来计算机软件开发的低效率问题，计算机业界提出了软件工程的思想和方法。面向对象技术是一种系统开发方法，是软件工程学的一个重要分支。在面向对象

编程中，数据被封装(或绑定)到使用它们的函数中，形成的整体称为对象，对象之间通过消息相互联系。面向对象建模与设计是使用现实世界的概念模型来思考问题的一种方法。对于理解问题、与应用领域专家交流、建模企业级应用、编写文档、设计程序和数据库来说，面向对象模型都非常有用。

1.1.1 软件系统概述

计算机是一种复杂的设备，通常包含的要素有硬件、软件、人员、数据库、文档和过程。其中，硬件是提供计算能力的电子设备；软件是程序、数据和相关文档的集合，用于实现所需要的逻辑方法、过程或控制；人员是硬件和软件的用户与操作者；数据库是通过软件访问的大型的、有组织的信息集合；文档是描述系统使用方法的手册、表格、图形及其他描述性信息；过程是一系列步骤，它们定义了每个系统元素的特定使用方法或系统驻留的过程性语境。

软件是指由系统软件、支撑软件和应用软件组成的软件系统。系统软件用于管理计算机的资源和控制程序的运行，主要功能是调度、监控和维护计算机系统，管理计算机系统中各种独立的硬件，使得它们可以协调工作。系统软件使得计算机使用者和其他软件将计算机当作整体，而不需要顾及底层的每个硬件是如何工作的。Windows、Linux、DOS、UNIX等操作系统都属于系统软件。支撑软件包括语言处理系统、编译程序以及数据库管理系统等。应用软件是用户可以使用的、由各种程序设计语言编制的应用程序的集合。应用软件是为满足用户不同领域、不同问题的应用需求而提供的，可以拓宽计算机系统的应用领域，使其为人们的日常生活、娱乐、工作和学习提供各种帮助。Word、Excel、QQ等都属于应用软件。

软件是用户与硬件之间的接口。用户主要通过软件与计算机进行交流。如果把计算机比喻为人，那么硬件就表示人的身躯，而软件则表示人的思想、灵魂。一台没有安装任何软件的计算机称为“裸机”。

软件是计算机系统设计的重要依据。为了方便用户，为了使计算机系统具有较高的总体效用，在设计计算机系统时，必须通盘考虑软件与硬件的结合，以及用户的要求和软件的要求。软件的正确含义应该是：①运行时，能够提供所要求功能和性能的指令或计算机程序的集合；②程序能够满意地处理信息的数据结构；③为了描述程序功能需求以及程序如何操作和使用所要求的文档。

软件与硬件的不同主要体现在以下几点。

(1) 表现形式不同。硬件有形，看得见，摸得着；而软件无形，看不见，摸不着。软件大多存在于人们的头脑里或纸面上，它们的正确与否，是好是坏，一直要到程序在机器上运行才能知道。这就给设计、生产和管理带来许多困难。

(2) 生产方式不同。硬件是设备制造，而软件是设计开发，是人的智力的高度发挥，不是传统意义上的生产制造。虽然软件开发和硬件制造存在某些相似点，但二者根本不同：两者均可通过优秀的设计获得高品质产品，然而硬件在制造阶段可能会引入质量问题，这在软件中并不存在(或者易于纠正)。二者都依赖人，但是人员和工作成果之间的对应关系是完全不同的。它们都需要构建产品，但是构建方法不同。软件产品的成本主要在于开发设

计，因此不能像管理制造项目那样管理软件开发项目。

(3) 要求不同。硬件产品允许有误差，而软件产品却不允许有误差。

(4) 维护不同。硬件会用旧、用坏，理论上，软件不会用旧、用坏，但实际上，软件也会变旧、变坏。因为在软件的整个生存周期中，一直处于改变维护状态。

现在的计算机软件系统具有产品和产品交付载体的双重作用。作为产品，软件显示了由计算机硬件体现的计算能力，更广泛地说，显示的是由可被本地硬件设备访问的计算机网络体现的计算潜力。无论驻留在移动电话还是大型计算机中，软件都扮演着信息转换的角色：产生、管理、获取、修改、显示或传输各种不同的信息，简单的如几位信息的传递，复杂的如从多个独立的数据源获取的多媒体演示。而作为产品生产的载体，软件提供了计算机控制(操作系统)、信息通信(网络)以及应用程序开发和控制(软件工具和环境)的基础平台。

软件提供了我们这个时代最重要的产品——信息。它会转换个人数据(如个人财务交易)，使信息在一定范围内发挥更大的作用；它通过管理商业信息提升竞争力；它为世界范围的信息网络提供通路(如互联网)，并对各类格式的信息提供不同的查询方式。

在最近半个世纪里，软件的作用发生了很大的变化。硬件性能的极大提高、计算机结构的巨大变化、内存和存储容量的大幅增加，还有种类繁多的输入和输出方法，都促使计算机系统的结构变得更加复杂，功能更加强大。如果系统开发成功，复杂的结构和功能可以产生惊人的效果，但是同时复杂性也给系统开发人员带来巨大的挑战。

现在，庞大的软件产业已经成为工业经济中的主导因素。早期的独立程序员也已经被专业的软件开发团队代替，团队中的不同专业技术人员可分别关注复杂的应用系统中某一技术部分。然而同过去的独立程序员一样，开发现代计算机系统时，软件开发人员依然面临同样的问题：

- 为什么软件需要如此长的开发时间？
- 为什么开发成本居高不下？
- 为什么在将软件交付顾客使用之前，我们无法找到所有的错误？
- 为什么维护已有的程序要花费高昂的时间和人力代价？
- 为什么软件开发和维护的过程仍旧难以度量？

种种问题验证了业界对软件以及软件开发方式的关注，这种关注促使业界对软件工程实践方法的采纳。

1.1.2　软件工程

要使计算机执行某一复杂的任务，而这一任务运行着一项业务，就不得不编写出一系列指令来明确规范计算机应该完成的事情。这些指令以诸如C++、Java或C#之类的编程语言编写，形成了我们所熟知的计算机软件。

然而，随着技术和应用的发展，计算机软件越来越复杂，程序员不能简单地制定业务操作的规则，或是猜测需要输入系统中的数据类型，这些数据类型会被存储并且稍后会被访问，以屏幕显示或报告的形式提供给用户。

要构建能够适应各种挑战的软件产品，就必须认识到以下几个简单的事实。

软件已经深入我们生活的各个方面，对软件应用所提供的特性和功能感兴趣的人显著增多。当要开发新的应用领域或嵌入式系统时，一定会听到很多不同的声音。很多时候，每个人对发布的软件应该具备什么样的软件特性和功能似乎都有些许不同的想法。因此，在制定软件解决方案前，必须尽力理解问题。

年复一年，个人、企业和政府的信息技术需求日臻复杂。过去一个人可以构建的计算机程序，现在需要由一支庞大的团队来共同实现。曾经运行在可预测、自包含、特定计算环境下的复杂软件，现在可以嵌入消费类电子产品、医疗设备、武器系统等各种环境中执行。这些基于计算机的系统或产品的复杂性，要求对所有系统元素之间的交互非常谨慎。因此，设计已经成为关键活动。

个人、企业、政府在进行日常运作管理以及战略战术决策时越来越依靠软件。软件失效会给个人或企业带来诸多不便，甚至灾难性的失败。因此，软件必须保证高质量。随着特定应用感知价值的提升，其用户群和软件寿命也会增加。随着用户群和使用时间的增加，其适应性和可扩展性需求也会同时增加。因此，软件需要具备可维护性。

由这些简单事实可以得出如下结论：各种形式、各个应用领域的软件都需要工程化。软件工程是研究如何以系统性的、规范化的、可定量的过程化方法开发和维护软件，以及如何把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来的学科。

软件工程的基础是过程。过程将各个技术层次结合在一起，使得合理、及时地开发计算机软件成为可能。过程定义了一个框架，构建该框架是有效实施软件工程技术必不可少的。过程构成软件项目管理控制的基础，建立工作环境以便于应用技术方法、提交工作产品(模型、文档、数据、报告、表格等)、建立里程碑、保证质量及正确管理变更。

过程是构建工作产品时执行的一系列活动、动作和任务的集合。活动主要实现宽泛的目标(如与利益相关者进行沟通)，与应用领域、项目大小、结果复杂性或者实施软件工程的重要程度没有直接关系。动作(如体系结构设计)包含主要工作产品(如体系结构设计模型)生产过程中的一系列任务。任务关注小而明确的目标，能够产生实际产品(如构建单元测试)。

在软件工程领域，过程不是对如何构建计算机软件的严格规定，而是一种可适应性调整的方法，以便于工作人员(软件团队)可以挑选适合的工作动作和任务集合。其目标通常是及时、高质量地交付软件，以满足软件项目资助方和最终用户的需求。

过程框架定义了若干框架活动，为实现完整的软件工程过程建立基础。这些活动可广泛应用于所有软件开发项目，无论项目的规模和复杂性如何。此外，过程框架还包含一些适用于整个软件过程的普适性活动。通用的软件工程过程框架通常包含以下5种活动。

- 沟通：在技术工作开始之前，和客户(及其他利益相关者)的沟通与协作是极其重要的；其目的是理解利益相关者的项目目标，并收集需求以定义软件特性和功能。
- 策划：如果有地图，任何复杂的旅程都可以变得简单。软件项目好比复杂的旅程，策划活动就是创建“地图”，以指导团队的项目旅程，这种地图称为软件项目计划，里面定义和描述了软件工程工作，包括需要执行的技术任务、可能的风险、资源需求、工作产品和工作进度计划。

- 建模：无论是庭院设计家、桥梁建造者、航空工程师、木匠还是建筑师，每天的工作都离不开模型。你会画一张草图来辅助理解整个项目大的构想——体系结构、不同的构件如何结合，以及其他一些特征。如果需要，可以把草图不断细化，以便更好地理解问题并找到解决方案。软件工程师也是如此，利用模型来更好地理解软件需求，并完成符合这些需求的软件设计。
- 构建：包括编码(手写的或自动生成的)和测试以发现编码中的错误。
- 部署：将软件(全部或部分增量)交付给用户，用户对其进行评测并给出反馈意见。

上述5种通用框架活动既适用于简单小程序的开发，也可用于大型网络应用程序的建造以及基于计算机的大型复杂系统工程。不同的应用案例中，软件过程的细节可能差别很大，但是框架活动都是一致的。

对许多项目来说，随着项目的开展，框架活动可以迭代应用。也就是说，在项目的多次迭代过程中，沟通、策划、建模、构建、部署等活动不断重复。每次项目迭代都会产生软件增量，每个软件增量实现了部分的软件特性和功能。随着每一次软件增量的产生，软件逐渐完善。

开发可靠软件是一种高成本的劳动密集型活动。已经有无数关于软件项目失败的报告，因此软件开发是一项高风险投资。在过去的几十年间，软件工业经历了快速增长，这使得大型软件系统的开发对规范方法的需求变得非常突出。今天，开发者采用已验证的软件工程方法和完善的项目管理实践，来保证所构建的软件不但满足客户的功能需求，还要及时交付，并且不会超出项目预算。

因为软件是不可见的，所以找出软件中存在的所有缺陷是相当困难的。实际上，不管准备投入多少资源和精力，完全没有缺陷的复杂软件系统只能是软件开发者的梦想。虽然人们必须接受这类系统不可能完全没有缺陷的事实，但是必须考虑软件故障所造成的后果。现在，软件系统广泛应用于政府、商业组织和日常活动的方方面面，支撑着这些系统的平稳运行。因此，软件系统故障总会影响人们的生活，并可能引发很多灾难，甚至关系到生命安全。因而，质量是软件工业的一个非常重要的问题。解决这个问题的最常见办法是采用已验证的过程来开发软件系统。

尽管质量问题非常重要，但是开发大型软件系统面临的最大挑战是开发这些项目所需的时间很长，系统需求总是会由于各种原因发生改变。经常发生这样的事情：在项目开始的时候，客户并不清楚他们具体想要的是什么。这样一来，当交付项目的时候，客户可能会发现系统并不像他们预期的那样。技术的快速变化本身也可能是个问题。如果项目开发时间很长，那么在项目开发期间可能会发生技术变更的情况。项目经理经常左右为难，一方面不得不修改系统设计以采用新技术，但是另一方面会导致费用超出预算，并且延长了开发时间，或者构建了无关紧要的系统，仅仅是完成了交付，但是系统并没有真正用起来。

开发大型系统的另一个困难是开发过程通常会涉及一支由众多专业人员构成的团队，这些专业人员都是各自领域的专家。因此，团队成员之间进行高效沟通是极其重要的。实际上，很多时候项目失败的首要原因是不良沟通和人员因素问题，而非技术问题。

软件工程将系统化的、规范的、高质量的方法应用到软件开发、运行和维护过程中。

软件工程建立在良好的工程概念之上，并且实际上很多人将软件工程规范开发比作建筑行业，已经从以人工活动为主转变为细化的工业流程。

与其他工程规范一样，软件工程师在真正实现系统之前要构建软件系统的模型。在软件开发过程中，建模是一项非常重要的活动，通常在最终设计和实现软件系统之前，软件工程师要花费很多时间在不同的抽象层次上开发模型。模型是一种高效的沟通手段，特别是在那些不需要详细信息的场合。例如，通常使用高度抽象的拓扑图来表示运输系统中的行车路线。在软件系统中，不同的涉众(stakeholder)总是需要物理系统的不同方面的信息。例如，乘客需要知道车费和某条公交路线的停靠站位置；公交车司机则需要某项特定公交车服务的精确路线信息；公交车站管理员需要知道所有从本站发出以及到达本站的公交车的时间表。为了迎合这些涉众的不同需要，需要为他们创建不同的模型，具体如下。

- 为乘客建立的模型：可以用一条直线并在上面画一个圆圈来表示，给出公交车停靠站名，可能还有相关的车费。
- 为公交车司机建立的模型：可以是一张显示公交车服务覆盖区域的简化路线图。为了向司机提供更多的信息，图中还包括街道名称和道路。
- 为行车路线规划者建立的模型：这个模型中可能包含由各条行车路线构成的详细道路图。每条行车路线都被标记出来，并用不同的颜色显示。

模型可以包含一个或多个视图，每个视图表示系统的某个特定方面。例如，乘客模型包含车费视图和路径视图。车费视图为某条路线上的不同停靠站提供车费信息，而路径视图则提供包含相关街道的路线信息。基于不同视图的模型必须是一致的。例如，建筑物的三维模型必须与同一建筑物的不同正视图(模型)保持一致。进而应该能够使用某种合适的表示法(语言)来表达模型，涉众能够理解这种表示法(notation)。对于软件开发而言，可以使用3个正交的视图来适当地描述系统：

- 包含软件系统中数据转换的功能视图；
- 包含系统结构以及与之相关的数据的静态视图；
- 包含软件系统中事务顺序或过程的动态视图。

从广义上讲，有两大基本的软件开发方法：结构化方法和面向对象方法。前者自20世纪70年代就非常流行，因为传统的过程式语言为其提供足够的支持。随着20世纪90年代面向对象编程语言(如C++和Java)的发明，面向对象方法已经变得越来越流行。

可以按照对系统的不同视图进行建模的方式及其相关过程，对这两种软件开发方法进行比较。结构化方法以系统的功能视图为中心，在开发的不同阶段使用不同的模型。当从一个阶段进行到下一阶段时，当前阶段的模型也将转换到下一阶段的模型。结构化方法有三大弱点。

首先，因为结构化方法以系统的功能视图为中心，所以当系统的功能发生改变时，系统的分析、设计模型以及实现都要发生很大的变化。

其次，在结构化方法中，只要早期创建的模型发生改变(因为需求发生改变或者纠正前面的错误)，就需要进行模型变换工作。在分析阶段，数据流图(data flow diagram，DFD)用来系统建模，系统由一组函数构成，函数之间有数据流。在设计阶段，系统被建模成结构图，由函数层次结构组成。如果系统的功能发生改变，就需要重新来一遍，即重新经历分

析和设计阶段，而在这些阶段需要投入相当多的时间和精力。

最后，在结构化方法中基本上不存在动态视图。DFD由两个视图构成：功能视图和静态视图。图形界面的使用以及软件系统日益增加的复杂度使得动态视图变得越来越重要。软件模块的结构由结构图指定，这些图可由DFD变形得到。然而，系统的很多动态行为不能由函数之间的数据依赖关系推导出来。使用结构化方法很难做出动态模型，即使引入控制流图(control flow diagram，CFD)也同样如此，因为并没有在动态视图中正确地为系统建模。

由于结构化方法具有上述不足，因此在与面向对象方法进行比较时，结构化方法的性价比很低。面向对象方法将软件系统建模为一组相互协作的对象。对象通过发送和接收消息的方式与其他对象交互，在这些过程中操作对象的数据。面向对象方法使得软件工程师进行软件系统一致模型的开发变得更加简单，因为在整个开发过程中，只使用同样的一组模型。因而，在不同的阶段不需要浪费时间和精力进行模型转换和更新。

而且，使用面向对象方法开发的系统结构要比使用结构化方法开发的更加稳定。这是因为针对面向对象系统的改变已经被限定，仅仅发生在对象内部，修改工作要比使用结构化方法设计的系统更加容易。所以，使用面向对象方法开发的软件系统可以降低开发和维护成本。

1.1.3　面向对象的含义

在现实世界中，复杂的事物往往是由许多部分组成的。例如，一辆汽车是由发动机、底盘、车身和车轮等部件组成的。当人们生产汽车时，分别设计和制造发动机、底盘、车身和车轮等，最后把它们组装在一起。组装时，各部分之间有一定联系，以便协同工作。

面向对象系统开发的思路和人们在现实世界中处理问题的思路是相似的，是一种基于现实世界来设计与开发软件系统的方式。面向对象技术以对象为基础，使用对象抽象现实世界中的事物，以消息来驱动对象执行处理。和面向过程的系统开发不同，面对对象技术不需要一开始就使用主函数来概括整个系统的功能，而是从问题域的各个事物入手，逐步构建出整个系统。

在程序结构上，人们常常使用下面的公式来表述面向过程的结构化程序：

面向过程程序 = 算法 + 数据结构

算法决定了程序的流程以及函数间的调用关系，也就是函数之间的相互依赖关系。算法和数据结构二者相互独立，分开设计。在实际问题中，有时数据是全局的，很容易超出权限范围修改数据，这意味着对数据的访问是不能控制的，也是不能预测的，如多个函数访问相同的全局数据，因为不能控制访问数据的权限，程序的测试和调试就变得非常困难。另外，面向过程程序中的主函数依赖于子函数，子函数又依赖于更小的子函数。这种自顶向下的模式，使得程序的核心逻辑依赖于外延的细节，一个小小的改动，有可能带来一系列连锁反应，引发依赖关系的一系列变动，这也是面向过程程序设计不能很好处理需求变化、代码重用性差的原因。在实践中，人们慢慢意识到算法和数据是密不可分的，通过使用对象将数据和函数封装(或绑定)在一起，程序中的操作通过对象之间的消息传递机制

实现，就可以解决上述问题。因此，就形成了面向对象程序设计：

面向对象程序 = (对象 + 对象 + …) +消息

面向对象程序设计的任务包括两个方面：一方面是决定把哪些数据和函数封装在一起，形成对象；另一方面是考虑怎样向对象传递消息，以完成所需任务。各个对象的操作完成了，系统的整体任务也就完成了。

在面向对象程序设计中，所有人都使用相同的概念和表示方法，而且要处理的概念和表示方法都比较少。

- 数据和过程不是人为分离的。在传统方法中，需要存储的数据早早便与操作这些数据的算法分离开来，算法是独立开发的。从需要访问这些数据的过程来看，这会导致数据的格式不合适，或者数据的放置位置不方便。而利用面向对象的开发方式，数据和过程一起放在容易管理的小软件包中，数据从来不与算法分开。最后得到的代码不太复杂，对客户需求的变化也不太敏感。
- 代码更容易重用：在传统方法中，是从要解决的问题开始，利用问题来驱动整个开发，最后得到当前问题的单个解决方案，但明天总是会有另一个问题要解决。无论新问题和已解决的旧问题多么接近，都不能分解单个系统，对它进行调整，因为一个小问题就会影响系统的每个部分。在面向对象的开发方式中，我们总是要在类似的系统中寻找有用的对象。即使新系统的区别非常小，也可以修改已有的代码，因为对象就类似于智力拼图玩具中的各块拼图。在建立面向对象的系统时，在考虑自己编写代码之前，一般都应寻找已有的对象(由自己、同事或第三方编写)。

1.1.4 什么是对象

在关于面向对象分析和设计方面最早的一本书中，Coad和Yourdon将对象定义为：对问题域中某些事情的抽象，反映了系统保持信息，并且与信息交互，或者二者兼有的能力。

在这一背景下，抽象是一种表示形式，只是涵盖从某一角度来说重要的或关注的东西。地图就是一种大家所熟知的抽象表示。没有哪幅地图能显示设计区域中的所有细节(除非地图与区域一样大，并且使用同样的材质制作，否则就不能表示)。地图的目的旨在指导地图上显示细节的选择以及所强调的内容。道路地图关注道路及位置，通常会忽略地形特征，除非它们有助于导航。地理地图显示岩石和其他地貌特征，但是会忽略城镇和道路。不同的投影以及地图比例尺，强调区域的不同位置或者更值得关注的特征。每一幅地图都是抽象的，部分是因为地图揭示(或强调)的相关特征，并且也因为地图所隐藏(或淡化)的非相关特征。对象作为抽象，方式也大体相似。对象只代表与分析目的相关的事情的特征，并且隐藏非相关的特征。

Coad和Yourdon所讲的系统是基于面向对象的软件系统，并且考虑到了开发。然而，应该注意，面向对象也会涉及其他系统，特别是人类活动系统。我们在详述合适的软件系统之前必须理解这一点。在需求和分析工作流中，在对应用域(尤其是部分人类活动系统)的理解进行建模时，会使用到对象。在设计和实现待建模的工作流(并且部分会产生软件系

统)时，也会使用到对象。这些目的很特别，我们在弄明白所需要完成的工作的性质时，仍会使用到对象。

Rumbaugh等人特别说明了这种双重目的。我们将对象定义为概念、抽象或者对于手头问题的解决来说边界和含义都很清晰的事情。对象有两种目的：它们促进对实际世界的了解，同时又提供计算机实现的现实基础。

对象(object)是一件事、一个实体或名词。一些对象是活的，一些对象不是。现实世界中的例子有汽车、人、房子、桌子、狗、植物、支票簿或雨衣等。通常需要对人群、组织和诸如合同、销售或协议之类的关系进行建模。虽然关系是无形的，但这些关系是长期存在的，对人群和其他应用域中的事情有着复杂的影响。

考虑在超市购买一管牙膏。从某个层面来说，这只是一笔销售，是货-钱交易。从更深层次来说，会介入商店和厂商之间的复杂关系。这取决于其他因素。例如，根据购买商品时因所处地区不同而五花八门的产品质量保证书，或许这笔销售会为会员卡赢得积分；商品包装可能包括用于下次购买的优惠券，或者购买记录表格，填满之后会给优惠。现在假定发现牙膏有问题，需要退款或换货。顾客可能会以合适的方式因造成的损失而起诉商店。如果我们没有合理地理解因销售而造成的后果，就无法理解上述业务。此时几乎肯定，真实世界中的“销售”可以被建模为系统中的对象。

在合适的抽象层次，当选择我们期望建模的对象时——实际上，抽象层次对应地图制作者——我们需要提问：“这是什么样的地图，显示什么样的细节，为了强调什么？”信息系统模型中所有的对象与其他对象都有一定的相似性，Booch将此总结为一句话：对象是状态、行为和标识。这里，标识是每个对象独一无二的东西，而状态和行为是对象互相联系的东西。状态表示对象在某一时刻所处的情况，通过对象的状态可以判断对象可以执行某一事件的行为或动作。对于软件对象来说，状态是对象数据值的汇总(更广泛地说，包括与其他对象的链接)，而行为表示对象对事件响应的方式。可以使用的动作是由对象的状态判断的，但是通常所说的对象可以“操作”，只是指改变数据或者对其他对象发送消息。软件对象的很多行为(但不是全部行为)会导致状态发生改变。

事实上，对象是对问题域中某些事物的抽象。显然，任何事物都具有两方面特征：一方面是该事物的静态特征，如某个人的姓名、年龄、联系方式等，这种静态特征通常称为属性；另一方面是该事物的动态特征，如兴趣爱好、学习、上课、体育锻炼等，这种动态特征称为操作。因此，面向对象技术中的任何一个对象都应当具有两个基本要素：属性和操作。对象往往是由一组属性和一组操作构成的。

许多现代编程语言在很大程度上或完全依赖于对象的概念：将一些数据通过语法紧密绑定到可在这些数据上执行的操作。在面向对象的语言(如C++、C#、Java、Eiffel、Smalltalk、Visual Basic .NET、Perl等)中，程序员需要创建类，每个类定义了许多相似对象的行为和结构。然后，他们需要编写代码来创建并操作那些作为类的实例的对象。

之所以说对象是一种功能强大的编程技术，主要原因在于程序中对象的概念自然地对应到真实世界中的对象。例如，假设公司必须处理订单。这些订单可能包括产品ID号以及产品的相关信息；可能需要创建Order对象(对应于真实世界中的对象：订单)，使其包括ID和ProductList等属性；还可能希望能够将产品添加到订单中，并且能提交订单，因此，就需

要编写AddProduct和SubmitOrder方法。真实世界中的对象和抽象的代码对象之间的这种对应关系促使程序员在问题域中，而不是在计算机科学术语中进行思考。然而，这种优点可能有点言过其实，除非正在建立真实世界过程的模拟器，否则，这种代表“真实世界”的对象仅仅是系统的表面内容。设计的复杂性实际上取决于真实对象表面之下的东西，需要用代码来反映的业务规则、资源分配、算法和其他计算机科学的细节。如果只是用对象来反映真实世界，那么还有很多工作要做。

在面向对象的软件中，真实世界中的对象会转变为代码。在编程术语中，对象是独立的模块，有自己的知识和行为(也可以说它们有自己的数据和进程)。把软件对象看作机器人、动物或人是很常见的：每个对象都有一定的知识，表现为属性；并且知道如何为程序的其他部分执行某些操作。例如，Person对象知道自己的头衔、姓名、出生日期和地址，还可以改名、搬到新地址、告诉我们年龄有多大等。

在为Person对象编写代码时集中一个人的特性，就可以想象出系统的其余部分，这会使编程比其他方式更简单(还有助于从真实世界中的概念开始)。如果Person对象以后需要知道身高，就可以把身高信息(和相关的行为)直接添加到Person代码中。在系统的其余部分，只有需要使用身高属性的代码才需要修改，其他代码都保持不变。改变的简单性和本地化是面向对象软件的重要特性。

把生物看成某类机器人是很简单的，但把没有生命的物体看成有行为的对象就有点怪异。我们一般不认为电视能自己改变价格或给自己打广告。但是，在面向对象的软件中，这就是我们需要做的工作。关键是，如果电视不做这些工作，系统的其他部分就要做。这就会把电视的特性泄露给代码的其他部分，丧失我们一开始追求的简单性和本地化(又回到“做事的旧方式”了)。不要被面向对象开发中常见的把软件对象想象成人的理论、把人的特性赋予没有生命的对象或动物吓住。

1.1.5 类

为了表示一组事物的本质，人们往往采用抽象方法将众多事物归纳、划分成一些类。例如，我们常用的名词“人”，就是一种抽象表示。因为现实世界中只有具体的人，如王安、李晓、张明等。把所有国籍为英国的人归纳为一个整体，称为“英国人”，也是一种抽象。抽象的过程是对有关事物的共性进行归纳、集中的过程。依据抽象的原则进行分类，即忽略事物的非本质特征，只注意那些与当前目标有关的本质特征，从而找出事物的共性，把具有共同性质的事物划分为一类，所得出的抽象概念称为类。

在面向对象的方法中，类的定义如下：类是具有相同属性和操作的一组对象的集合，它为属于该类的全部对象提供统一的抽象描述。

事实上，类与对象的关系如同模具和铸件的关系。类是对象的抽象定义，或者说是对象的模板；对象是类的实例，或者说是类的具体表现形式。类封装了一组对象的公共属性。

类用来描述一组按照相同方式进行规范的对象。这里，我们将对象作为软件系统中的抽象(既可以是模型，也可以是最终的软件)，而不是它们所表示的真实世界。

给定类的所有对象，就特征、语义和基于对象的约束来说，都有共同的规范(一般来说，语义与单词或符号的含义相关，但是对计算机科学来说，语义通常是使用编程语言实现操作的正式数学描述。这里，语义只是表示对象大致的行为；或者从其他方面来说，在应用域中工作的人所表示的对象被赋予的含义)。这并不是指同一类中的所有对象在任何方面都相同，但是它们的规范是相同的。互相类似的对象属于同一类。类是这些对象之间所规范的逻辑相似性的一种抽象描述符(如图1-1所示)。

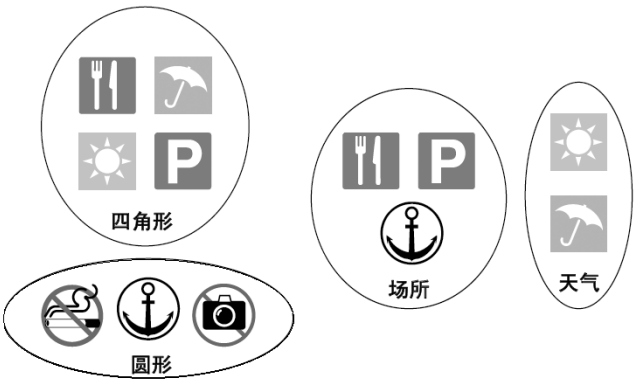

图1-1　类

类的思想根源于面向对象编程。例如，在Java程序中，类是某种模板，可以在需要的时候从模板中构建单个对象(这也不是全部的内容，因为软件类也可以做这里没有考虑的其他事情)。单个对象也称为实例。实例解释了对象所属类的含义，每一个对象都是类的一个实例。

类和类的类型会有进一步的区分。类型类似于类，但是更为抽象，既不包含物理实现，也不包含操作的物理规范。类型可以由多个类实现，例如，在语法和特征不同的两种编程语言的情况下。一些作者建议，分析模型可以只包含类型而不包含类。然而，术语“类”的标准使用应具有所有的含义：真实世界中类似的一组对象，分析和设计模型中规范的一组类似对象，以及面向对象编程语言中的软件结构。同时，“对象”是“实例”的同义词，虽然后者通常更多用在对象所属类的讨论中。

面向对象的开发人员使用类来描述某种对象拥有的编程元素。下面的代码显示了一个用Java编写的完整类，其中包含类的6个基本元素(参见表1-1)。

```
1  // An actor with "name" and "stage name" attributes
2  public class Actor {
3
4     //Fields
5     private String name, stageName;
6
7     //Create a new actor with the given stage name
8     public Actor(String sn) {
9           name = "<None>";
10       stageName = sn;
11    }
12
```

```
13  //Get the name
14  public String getName() {
15     return name;
16   }
17
18  //Set the name
19  public void setName(String n) {
20     name = n;
21  }
22
23  //Get the stage name
24     public String getStageName() {
25        return stageName;
26  }
27
28  //Set the stage name
29     public void setSatgeName(String sn) {
30        stagename = sn;
31     }
32
33     // Reply a summary of this actor’s attributes, as a string
34     public String toString() {
35        return "I am known as " + getStageName() +
36                    ", but my real name is + "getName();
37     }
38  }
```

表1-1　类定义的信息

元素	作用	对应代码
类名	在代码的其他地方引用类	第2行的Actor
字段	描述这类对象存储的信息	第5行的name和stageName
构造函数	控制对象的初始化	第8行的Actor()
消息	以使用对象的方式提供其他对象	第14行的getName()，第19行的setName()，第24行的getStageName()，第29行的setStageName()，第34行的toString()
操作	告诉对象如何操作	第15、20、25、30、35、36行
注释	告诉程序员如何使用或维护类(编译器忽略)	以//开头的行，如第1行和第4行

新对象是由Actor操作创建的，这是一种特殊的操作，称为构造函数，只在创建类的实例时使用。在Java中，使用下面的表达式创建对象：

```
New Actor("Charlie Chaplin");
```

在这个例子中，表达式会生成Actor的一个新实例，它的参与者名字是Charlie Chaplin，名称是<None>。getName()和setName()等操作称为获取(getter)和设置(setter)操作，因为它们获取和设置信息。

除了上面列出的元素之外，纯面向对象编程语言还允许程序员指定系统的哪些部分可以访问元素，通常至少可以指定元素是公共的(在其他地方可见)还是私有的(仅对对象

本身可见)，因此要使用Java中的关键字public和private。一些语言允许程序员添加断言(assertion)，即必须总是为true的逻辑语句，例如这个类的对象总是有正的结果，或者这条消息总是返回非空字符串。断言对于可靠性、调试和维护比较有用。

1.1.6　封装、信息隐藏和消息传递

封装是面向对象编程的特征，用于分析模型，指的是在对象中放置数据，同时将操作应用于数据。对象其实就是一些数据以及处理这些数据的操作。数据存储于对象的特性中，数据和特性一起组成对象的信息结构。这些处理过程就是对象的操作，每一个操作都有特定的签名。操作的签名定义了作为有效请求而传递的消息的结构和内容，包括操作的名称，以及需要运行操作的任何参数(通常是数据值)。为了调用操作，必须给出操作的签名。签名有时也称为消息协议。对象的操作签名的完整集合称为接口。所以，每一个操作都提供了接口，允许系统的其他部分通过发送消息来调用操作。操作可以访问存储于对象中的数据。接口独立于数据和操作的实现。这一思想在于数据只能被同一对象的操作访问和修改(然而大部分面向对象编程语言都提供了回避封装的方法)。信息隐藏是更强大的设计原则，指明没有对象或子系统可以对其他对象或子系统公布实现的细节。不管是封装还是信息隐藏，都表明为了让对象相互协作，必须交换消息。

对象通常表示真实世界系统中的事情，而系统需要协作以完成共同的任务。协作的事务和人互相发送消息，例如，这些消息包括：我们对家人和朋友所说的事情、登录网站阅读的电子邮件、公共汽车上的广告贴画、电视中的娱乐节目和动画片、交通指示灯信号、笔记本计算机的电源指示灯，甚至我们穿的衣服、说话的语气和姿态，这些都是某种类型的消息。使得这些消息有用的是它们遵守的协议，而这些协议是我们对消息意义的解释。比较明显的例子就是国际上就“红灯停，绿灯行”达成的一致。软件对象也需要一致的协议，以便能互相通信。

直到最近，软件才按照这样的方式构建。早期的系统开发方法一般是将系统中的数据和数据的处理过程分割开来。这样做缘于合理的原因分析，并且在一些应用程序中依然使用，但是逐渐出现了困难。最主要的困难是，对于设计系统过程的人来说，很难理解系统使用数据的组织情况。对于这样的系统，数据处理依赖于数据结构。

数据处理和数据结构的依赖关系会引起问题。数据结构的改变通常会迫使使用数据的处理过程发生改变。这加大了构建可靠系统的难度，也增加了系统升级和修改的难度，而在系统崩溃时难以进行修复。

相比之下，设计合理的面向对象系统会被模块化，以便每个子系统在设计和实现的时候，尽可能地与其他子系统独立。通过对数据分配处理过程，封装就可以做到这点。

在实际中，处理过程通常不能被分配给由它们处理的所有数据，而处理过程会分布于不同的对象中。一些操作甚至会被编写为访问对象中已封装的数据。因为对象要使用另一个对象的操作，就必须发送消息。

信息隐藏比封装更进一步，使得对象不可能以任何方式访问另一对象的数据，除非调用操作。这避免了每个对象都需要“知道”其他对象的所有外部细节。

本质上，对象只需要知道自身的数据和操作。然而，很多处理过程也因此必须包括知道如何请求来自其他对象的服务。服务是对象或子系统执行的代表其他对象或子系统的动作，包括对存储于其他对象中的数据的索引。此时，对象必须“知道”该向其他对象提问什么，以及如何提问。但是也不需要对象“知道”其他对象发布服务方式的所有信息。这一“知识”要求负责实现对象的程序员有其他对象实现方式的详细信息。

可以将对象看作防护层，就像洋葱头的表皮。封装放置了数据以及可以直接使用数据的操作。信息隐藏使得对象的内部细节对其他对象不可访问。对需要访问对象数据的其他对象，必须发送消息。当对象接收到消息时，可以区分消息是否与自己有关。如果消息中包括操作的有效签名，对象就会响应。如果没有包括，对象就不响应。操作只能被给出有效操作签名的消息调用。对象的数据甚至处于更深层，只能被自己对象的操作访问。因此，对象中的操作运行和数据的组织方式可以被改变而不影响协作的对象。只要操作的签名不变，这样的改变对外部就是不可见的。图1-2阐述了信息隐藏和封装的对比情况。

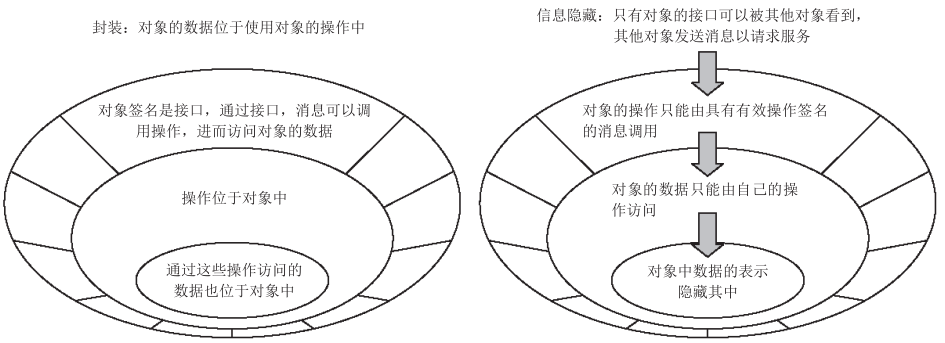

图1-2　封装和信息隐藏

这种软件设计方式有很多实际优势。考虑公司职员工资支付的简单系统，假设存在类Employee，这个类的实例表示工资单上的每一个职员。Employee对象负责了解表示的真实职员工资水平。假定还有PaySlip对象，负责打印职员每月的工资单。为了打印工资单，PaySlip对象必须知道对应职员的工资水平。解决该问题的一种面向对象方法是，对每一个PaySlip对象，发送与Employee对象关联的消息，询问应该支付多少工资。PaySlip对象不必知道Employee对象是如何计算出工资的，也不需要知道存储的数据，只需要了解可以向Employee对象询问工资的数值，并且会给出合理的响应即可。消息传递允许对象对系统的其他部分隐藏自己的内部信息，因此能够最小化因设计或实现改变而造成的连锁效应。

1.1.7　继承与多态

继承是面向对象编程语言实现一般化和特殊化的一种机制。当两个类通过继承机制进行关联的时候，更一般化的类称为超类，而另一个相关的、更特殊化的类称为子类。面向对象的继承规则通常如下。

- 子类继承超类的所有特性。
- 子类的定义通常至少包括一个不是从超类派生出的细节。

继承与一般化的关系很紧密。一般化描述了共享一些特性的元素之间的逻辑关系，而

继承则描述了允许这些共享关系出现的面向对象机制。

注意，面向对象中的继承与生物学和逻辑概念上的继承之间只是表面上相像。一些主要的区别如下。

- 生物学中的继承(至少对于哺乳动物来说)是复杂的，因为子代继承了父代的特征。从每个父本或母本继承的特征部分是随机判断的，部分是由基因和染色体工作机制判断的。逻辑继承主要与原法人的死亡造成的财产转移有关，而不是原法人特征的转移。逻辑继承的规则因地而异，通常也很复杂，但是在大部分国家，法定继承人一般都是生物学意义上的后代。
- 对于面向对象中的继承，类通常只有一个父类，并且继承父类的所有特性。两个特例是所继承特性的多重继承和重写。多重继承是指子类同时是多个层级结构中的成员，继承了每个层级结构中自己所在超类的特性。重写是指继承的特性在子类中被重新定义。重写在操作需要按照不同的方式定义不同的子类时很有用。此时，操作可能只是做了大致描述或是采用超类中的默认形式，之后会被重新指定，但是在每一个子类中都有所不同。

在层级结构中，相邻层中两个元素之间的关系可以在更特殊化的一级传递下去。因此，在图1-3中，动物的定义可以适用于所有的哺乳动物，因此经过一系列传递操作之后，也适用于家猫。因此，我们可以完善上述继承规则，具体如下。

- 子类通常会继承超类的所有特性。
- 子类的定义通常至少包括一个不是从超类派生出的细节。

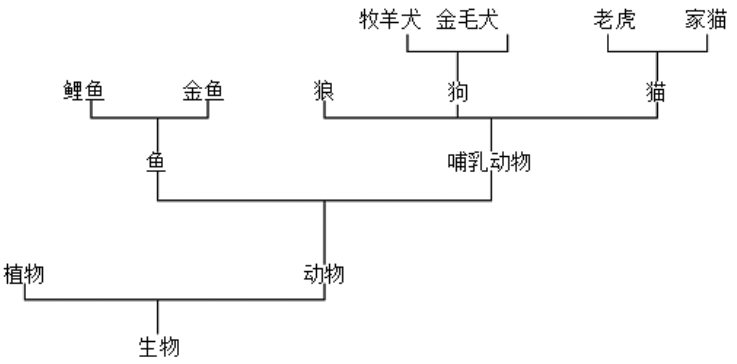

图1-3　继承

在层级结构系统中，随着远离根部而更靠近叶子，树的分支也会更加发散。它们不允许汇聚。这意味着，例如，家猫不可能既是哺乳动物，又是爬行动物。换言之，层级结构的每一个元素都处于给定层级结构的某一层，都只是某个分类的成员(当然在其他层中，也有可能是其他分类的成员，因为传递特性的存在)。

一般化的互斥特性说明有时候我们需要小心选择表达一般化的特性。例如，我们不能将“拥有4条腿”作为定义哺乳动物的特征，甚至上述特征假定对所有哺乳动物都成立也不行。因为很多蜥蜴也有4条腿，否则会将蜥蜴也归类为哺乳动物。在定义类的时候，特征集合必须是独一无二的，从而将类与同一层的其他类区分开来。

一般化的层级结构是互斥的这一事实，不能被误解为类只能属于层级结构。一般化结

构是我们选择的一种抽象，因为这种结构表达了我们对应用域某些方面的理解。这说明只要能够表达情况的相关方面，就可以选择将多个一般化结构应用于同一域中。因此，一个人可能会被同时归类为人(智人)、市民(公民选举人)和职员(Agate创意部门的财务经理)。如果每个层级结构都可以使用面向对象模型表示，那么人就是多重继承的例子。

真实世界中的结构不会强迫遵从面向对象建模中的逻辑规则。有时它们既不是传递的，也不是互斥的，因此不是严格的层级结构关系。但是这并不妨碍层级结构在面向对象开发中的用处。

多态的字面意思是“表现为多种形式的状态”，是指相同的消息被发送到不同类对象的可能性，每一个类对象都会以不同但合理的方式响应消息。这意味着原始对象不需要知道在特定场合即将接收消息的类。多态中关键的一点是，每一个对象都知道如何响应接收到的有效消息。同一条消息被不同的对象接收到时，可能产生完全不同的行为，这就是多态性。多态性支持“同一接口，多种方法”的面向对象原则，使高层代码只写一次而在低层可以多次复用。

实际上，在现实生活中可以看到多态性的许多例子。如学校发布一条消息：8月25日新学期开学。不同的对象接收到该条消息后会做出不同的反应：学生要准备好开学上课的必需物品，教师要备好课，教室管理人员要打扫干净教室，准备好教学设备和仪器，宿舍管理人员要整理好宿舍，等等。显然，对于同一条消息，不同的接收对象做出不同的反应，这就是多态性。可以设想，如果没有多态性，那么学校就要分别给学生、教师、教室管理人员和宿舍管理人员等许多不同对象分别发开学通知，分别告知需要做的具体工作，显然这是一件非常复杂的事情。有了多态性，学校在发消息时，不必一一考虑各类人员的特点，不断发送各种消息，而只需要发送一条消息，各类人员就可以根据学校事先安排的工作机制有条不紊地工作。

从编程角度看，多态提升了代码的可扩展性。编程人员利用多态性，可以在少量修改甚至不修改原有代码的基础上，轻松加入新的功能，使代码更加健壮、易于维护。

这与人群进行通信类似。当一个人向另一个人发送消息的时候，我们通常会忽略另一个人可能的响应方式。例如，一位母亲可能会使用同样的表述告诉她的孩子：“现在去睡觉”，但是根据孩子年龄或者其他性格原因，完成该任务的方式也不一样。5岁的孩子会直接自己上床睡觉，但是之后可能会要求帮他洗脸、刷牙、换睡衣，并且还会期望有人给他阅读睡前故事，而13岁的孩子不会要求这么多，只要到了时间就会去睡觉。

多态在使用面向对象方法鼓励子系统的解耦合方式方面是重要元素。图1-4使用通信图阐述了多态在某种业务场景下的工作方式。图1-4假定存在不同的方式用于计算职员的工资。全职职员的工资取决于员工等级；兼职职员的工资部分取决于员工等级，但是也取决于工时；临时工的工资不需要扣除养老金，除此之外，工资的计算方式与全职职员相同。计算这些职员工资的面向对象系统可能包括表示每一类型职员的单独类，每个类都能进行合适的工资计算。然而，根据多态原则，所有calculatePay操作的消息签名可能是相同的。假定系统的某个输出是打印显示本月总的工资：为了计算总的工资数额，需要对每一个职员发送消息，计算他们各自的工资。既然消息签名在每种情况下都是一样的，因此请求类(这里是MonthlyPayPrint)不需要知道每一个接收对象的类，也仍然能够进行计算。

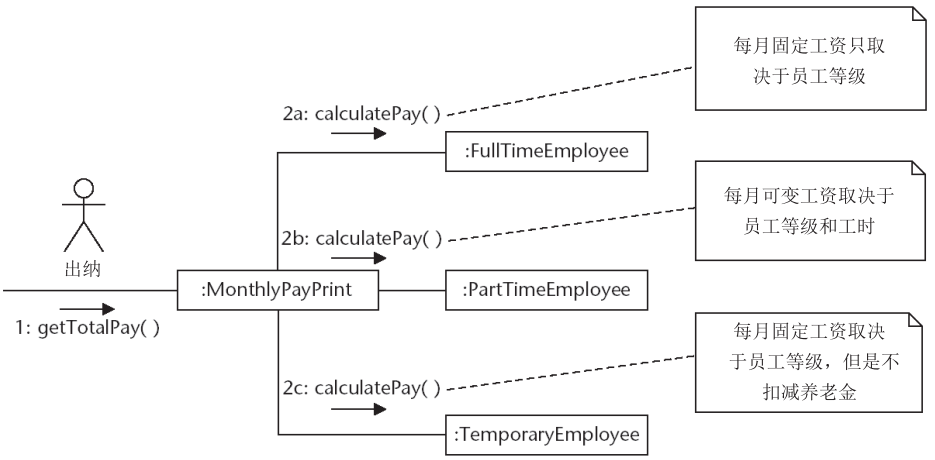

图1-4　多态

多态对于软件系统开发者来说是十分强大的概念。多态与封装和信息隐藏一起，允许子系统之间明确分离，能够按照表面看起来类似的方式处理不同形式的任务。这表明系统修改起来很简单，并且能够扩展以包含附加特征，因为在进行修改的时候，只需要知道类之间的接口。对于开发者来说，除非是自己要开发的部分，否则不需要关注系统任何部分的实现方式(内部结构和行为)。

1.2　面向对象的开发模式

面向对象开发是一种基于现实世界以及程序中的抽象来思考软件的方式。在此背景下，开发(development)指的是软件生命周期，即分析、设计和实现。面向对象开发的本质是识别和组织应用领域中的概念，而不是以一种编程语言最终表示这些概念。因为问题内在的复杂性，软件开发中困难的部分是对其本质进行的操作，而不是将它映射成某种语言的这些次要方面。

1.2.1　面向对象程序的工作原理

面向对象程序在工作时，要创建对象，把它们连接在一起，让它们彼此发送消息，相互协作。谁启动这个过程？谁创建第一个对象？谁发送第一条消息？为了解决这些问题，面向对象程序必须有一个入口点(entry point)。例如，Java在启动程序时，要在用户指定的对象上找到main操作，执行main操作中的所有指令，当main操作结束时，程序就停止。

main操作中的每个指令都可以创建对象，把对象连接在一起，或者给对象发送消息。对象发送消息后，接收消息的对象就会执行操作。这个操作也可以创建对象，把对象连接在一起，或者给对象发送消息。这样，该机制就可以完成我们想完成的任何任务。

图1-5显示了一个面向对象程序。main操作中一般没有很多代码，大多数动作都在其他对象的操作中。如图1-5所示，对象给自己发送消息是有效的，我们也可以问自己一个问

题：“我昨天干了什么？”

main操作的理念不仅可以应用于在控制台上执行的程序，也可以应用于更复杂的程序，例如图形用户界面(GUI)、Web服务器和服务小程序(servlet)。下面是针对其工作方式的一些提示。

- 用户界面的main操作创建顶级窗口，告诉它显示自己。
- Web服务器的main操作有一个无限循环，告诉socket对象监听某个端口的入站请求。

servlet是由Web服务器拥有的对象，它接收从Web浏览器传送来的请求。注意，Web服务器有main操作。

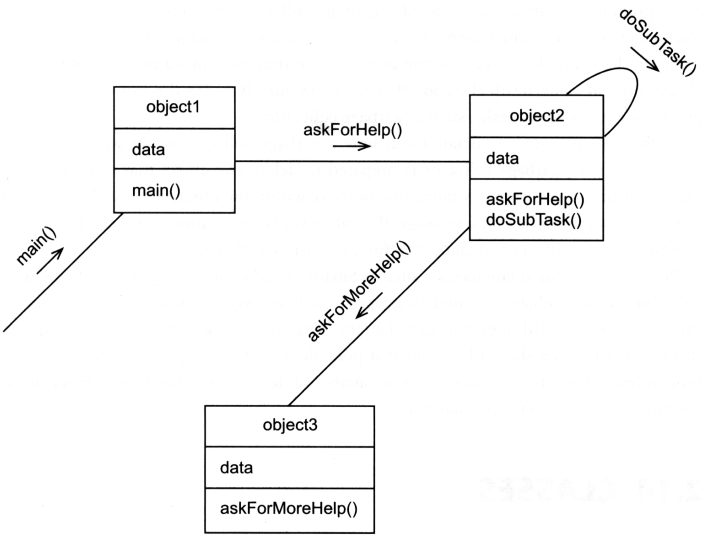

图1-5　面向对象程序

1.2.2　面向对象方法论

根据Budgen(1994年)的定义，软件开发方法主要由表示法、过程和技术3部分组成，如图1-6所示。

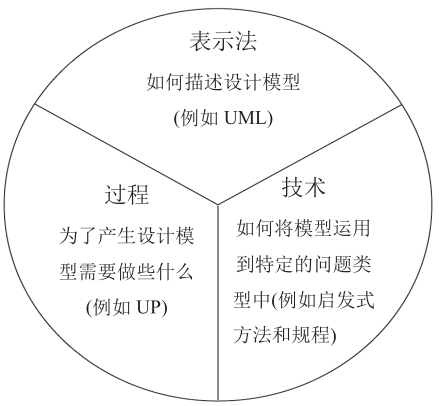

图1-6　软件开发方法

过程对应为从问题空间转到解空间。在这个过程中，为设计师提供了一些选项，必须做出选择或判断。软件开发方法的技术部分为他们提供了一些关于正确选择的启发式方法和规程，通过这种方式可以辅助设计师完成设计。典型的是，这个过程中的每个任务或活动会在结束的时候，产生系统工件(artifact)。这些工件使用推荐的表示法从一个或多个视点(模型)，在不同的抽象级别进行表示，这种表示法既可以为初始问题(需求)的结构建模，也可以为将要实现的解决方案建模。

如果没有规划好规程，那么系统开发将导致开发时间延期、费用超支，甚至导致项目不能完工。因此，开发者特别是初级程序员，在开发系统时一定要遵循经过验证的过程或规程，这样才能够在合理的预算和时间要求下完成可用的系统。然而，没有哪个合适的过程能够适合于所有环境。因此，选中的过程只能用来指导开发者在系统开发过程中应用适当的技术。与此同时，应该允许熟练的开发者能够按照他们自己的方式组织开发步骤，这样能够鼓励创造和创新。理想情况下，过程应该具备以下特性。

- 具有良好管理的迭代和增量式生命周期，在不影响创新的情况下提供必要的控制。
- 在整个软件开发生命周期(software development life cycle，SDLC)内为开发者嵌入方法、规程和启发式方法以完成分析、设计和实现。
- 在迭代和增量式开发过程中，为复杂问题的解决提供指导。
- 根据已知的或已经建模的需求识别那些不是非常明显的需求。

表示法是系统不同涉众之间使用的公共语言。人类大脑在同一时刻只能处理有限的信息。表示法模型通过创建真实系统的抽象层次化描述，有助于降低复杂性。通过抽象创建模型是人类认识世界的一项基本技能，而对于开发大型软件系统而言，这是重要的第一步。对于软件开发而言，表示法模型有助于开发者进行以下活动。

- 捕获系统需求。使用的表示法应该可以被用户和开发者理解。
- 通过开发合适的分析模型分析系统。使用恰当的表示法表达模型，这样开发者就可以快速、方便地从中提取信息。
- 开发系统设计。使用恰当的表示法开发和表达设计模型，使设计师和程序员都可以理解。系统设计师可能要操作分析模型，并在此过程中做出设计决策。
- 实现、测试和部署系统。这也需要使用合适的表示法来表示这些活动的工件，使之可以被系统设计师、程序员和系统测试人员所理解。

为了支持上面这些活动，理想的表示法应该满足以下条件。

- 有利于团队成员和客户之间的高效沟通。
- 能够无歧义地描述用户的需求。
- 提供丰富的语义，以捕获所有重要的战术和战略决策。
- 为人们提供逻辑框架，以便于对模型进行推理。
- 有利于使用工具实现自动化，至少要实现模型构建过程的自动化。

在创建模型时，根据那些经过仔细设计的规则将信息分为不同的层次，使得它们既不太抽象也不会有太多局限。尽管建模对于人类而言是一个很自然的过程，但是为软件系统开发适当的模型则可能是软件工程领域最困难的一个方面。这是因为常常会有多个解决方

案，独立工作的不同观察者总会得到不同的模型。因此，开发一个系统化过程来判断在不同级别进行抽象以构造合理的一致模型是非常有用的。如果能够遵循一个经过验证的步骤检查列表来构造模型，就不会忽略重要的特性或关键的需求。

在软件开发过程开始时，一般是从客户那里捕获系统需求，并使用合适的建模表示法(比如UML)来表示这些需求。因为建模表示法提供了丰富的模型，所以很多开发者遇到的一个共同问题是：他们不知道确定设计需要哪些模型，以及在过程中如何创建模型。

技术部分的主要目的是提供一组指导意见和启发式方法，辅助开发者系统地开发必需的设计模型和实现。软件开发方法的技术部分应该包含以下内容。

- 一组用于产生和验证设计、初始需求和规约的指南。
- 一组设计师可用来确保设计结构和设计模型的一致性的启发式方法。如果设计由一支设计团队完成，那么这一点尤为重要，因为这些设计师需要确保他们的模型是一致和连贯的。
- 一个能够捕获设计关键特性的系统，以补充设计师的领域知识。

面向对象的开发模式使用图形化表示法来表示面向对象的概念。这个过程首先构建一种应用模型，然后在设计中增加细节。从分析到设计，再到实现，使用的是统一的可视化模型表示法，这样在某一开发阶段增加的信息在下一阶段就不会丢失或转换。这种方法论包括下面几个阶段。

(1) 系统构思：软件开发始于业务分析人员或用户构思一项应用，并制定临时性需求。

(2) 分析：通过创建模型，分析人员仔细审查并严格地重新描述系统构思阶段的需求。因为问题描述很少会是完整的或正确的，所以分析人员必须与请求者一起工作以理解问题。分析模型是一种简明准确的抽象，它描述目标系统要做哪些事情，而不是要如何来做这些事情。分析模型不应该包含任何实现决策。例如，工作站窗口系统中的Window 类可以用可视化属性和操作来描述。分析模型有两部分：领域模型(domain model)，描述系统内部反映的现实世界中的对象；应用模型(application model)，描述用户可见的应用系统本身的组成部分。例如，对于股票经纪人系统来说，领域对象包括股票、债券、交易和佣金，应用对象会控制交易的执行过程，并给出结果。一些应用专家虽然本身不是程序员，但他们能够理解并评议好的模型。

(3) 系统设计：开发团队设计出一种高层策略，即系统架构(system architecture)，用于解决应用程序的问题。他们也制定策略，默认作为后面更加详细的设计内容。系统设计人员必须确定优化哪些性能特性，选择何种策略来解决问题，完成哪些临时性的资源分配。例如，系统设计人员决定工作站屏幕上的变化必须快速且平滑，即使在窗口移动或删除的情况下也是如此，他们要选择一种适当的通信协议和内存缓冲策略。

(4) 类的设计：根据系统设计策略，类的设计者给分析模型添加细节。类的设计者使用相同的面向对象(object oriented，OO)概念和表示法阐释领域对象和应用对象，尽管它们存在于不同的概念层面上。类设计的焦点在于实现每个类的数据结构和算法。例如，类的设计者现在也要为Window类的每项操作确定数据结构和算法。

(5) 实现：实现人员将类的设计阶段开发的类及其关系转换到某种编程语言、数据库或硬件上。程序设计应该是简单直接的，因为所有困难的决策都已经完成。在实现阶段，

遵循良好的软件工程实践是很重要的，这样，设计的可追溯性就非常清晰，系统将保持灵活性和可扩展性。例如，实现人员会借助工作站下层图形系统的调用，用某种语言编写出 Window 类。

从分析到设计，再到实现，面向对象的概念会被应用到系统开发的整个生命周期中。可在各个阶段使用相同的类，不用改变符号记法，只在后续阶段增添细节。分析和设计模型服务于不同的目标，表示不同的抽象。但在整个开发过程中使用的标识、分类、多态和继承等概念都是相同的。

1.2.3　面向对象建模

建模在所有工程实践中都已得到广泛接受，这主要是因为建模引证了分解、抽象和层次结构的原则。设计中的每个模型都描述了被考虑的系统的某个方面。我们尽可能地在老模型的基础上构建新模型，因为我们对那些老模型已经建立起了信心。模型让我们有机会在受控制的条件下失败。我们分别在预期的情况和特殊的情况下评估每个模型，当它们没能按照我们的期望工作时，我们就修改它们。

为了表达一个复杂系统，必须使用多种模型。例如，当设计一台个人计算机时，电子工程师必须考虑系统的组件视图以及线路板的物理布局。组件视图构成了系统设计的逻辑视图，它帮助工程师思考组件间的协作关系。线路板的物理布局代表了这些组件的物理封装，它受到线路板尺寸、可用电源和组件种类等条件的限制。通过组件视图，工程师可以独立地思考散热和制造等方面的问题。线路板的设计者还必须考虑在建系统的动态方面和静态方面。因此，电子工程师利用一些图示来展示单个组件之间的静态连接，也用一些时间图来展示这些组件随时间变化的行为。然后，工程师就可以利用示波器和数字分析设备来验证静态模型和动态模型的正确性。

面向对象建模是一种新的思维方式，是一种关于计算和信息结构化的新思维。面向对象建模把系统看作相互协作的对象，这些对象是结构和行为的封装，都属于某个类，那些类具有某种层次化的结构。系统的所有功能通过对象之间相互发送消息来获得。面向对象建模可被视为包含以下元素的概念框架：抽象、封装、模块化、层次、分类、并行、稳定、可重用和可扩展性。

面向对象建模的出现，并不能算是一场计算革命。更恰当地讲，它是面向过程和严格数据驱动的软件开发方法的渐进演变结果。软件开发的新方法受到来自两个方面的推动：编程语言的发展以及日趋复杂的问题域的需求驱动。尽管在实际中分析和设计在编程阶段之前进行，但从发展历史看却是编程语言的革新带来了设计和分析技术的改变。同样，语言的演变也是对计算机体系的增强和需求的日益复杂的自然响应。

影响面向对象开发产生的诸多因素中，最重要的可能要算是编程方法的进步了。在过去的几十年中，编程语言中对抽象机制的支持已经发展到一个较高的水平。这种抽象的进化从地址(机器语言)到名字(汇编语言)，再到表达式(第一代高级语言，如FORTRAN)，到控制(第二代高级语言，如COBOL)，到过程和函数(第二代和早期第三代高级语言，如Pascal)，到模块和数据(晚期第三代高级语言，如Modula)，最后到对象(基于对象和面向对

象的语言)。Smalltalk和其他面向对象语言的发展，使得新的分析和设计技术的实现成为可能。

这些新的面向对象技术实际上是结构化和数据库方法的融合。面向对象的方法中，小范围内对面向数据流的关注，如耦合和聚合，也是很重要的。同样，对象内部的行为最终也需要面向过程的设计方法。数据库技术中实体-关系(E-R)图的数据建模思想也在面向对象的方法中得以体现。

计算机硬件体系结构的进步，性价比的提高和硬件设计中对象概念的引入，都对面向对象的发展产生了一定的影响。面向对象的程序通常要更加频繁地访问内存，需要更高的处理速度。它们需要并且也正在利用强大的计算机硬件功能。哲学和认知科学的层次和分类理论也促进了面向对象的产生和发展。最后，计算机系统不断增长的规模、复杂度和分布性都对面向对象技术起了或多或少的推动作用。

因为影响面向对象发展的因素很多，面向对象技术本身还未成熟，所以在思想和术语上有很多不同的提法。所有的面向对象语言并非生而平等，它们在术语、概念的运用上也各不相同。尽管也存在统一的趋势，但就如何进行面向对象的分析、设计而言还没有完全达成共识，更没有统一的符号来描述这些活动(说明：UML正在朝这方面努力)。但是，面向对象的开发已经在以下领域被证明是成功的：空中交通管理、动画设计、银行、商业数据处理、命令和控制系统、计算机辅助设计(CAD)、计算机集成制造(CIM)、数据库、专家系统、图像识别、数学分析、音乐合成、操作系统、过程控制、空间站软件、机器人、远程通信、界面设计和超大规模集成电路(VLSI)设计。毫无疑问，面向对象技术的应用已经成为软件工业发展的主流。

1.2.4 对概念而非实现建模

随着计算科学的发展，软件开发的抽象级别爆炸性增长。所谓“抽象”，是指程序员与执行程序的计算机在物理细节上的隔离。程序员可以写出单个指令，指令的含义通常接近于英语单词的意义，指令会被翻译为冗长的“机器码”指令序列。此时，执行程序的计算机的目的就变得相当复杂，需求也急剧增加，因此大大增加了系统自身的复杂度。

GUI在20世纪80年代和90年代的快速普及，给现代开发方法带来了特殊的困难。GUI的引入使得软件开发遇到新的难题。原因在于，展示给GUI用户的是计算机显示屏上高度视觉化的界面，一次性提供很多选项操作，每一种选项都可以通过单击鼠标实现。通过下拉菜单、列表框和其他对话框技术，很多其他的选项也可以通过两次或三次单击鼠标来实现。界面开发者自然会探索新技术带来的机遇。结果是，系统设计者几乎不可能预测用户通过系统界面执行的任何可能的任务。这意味着计算机应用程序现在很难以过程方式进行设计或控制。面向对象为设计软件提供了自然而然的方法，软件的每一个组件都提供明确的服务，而这些服务可以被系统的其他部分通过任务序列或控制流独立使用。

在精心设计的面向对象系统中，信息隐藏表明类有两种定义。外在地，类可以根据接口定义。其他的对象(以及程序员)只需要知道类对象能提供的服务，以及用于请求服务的签名即可。内在地，类可以根据知道的和完成的事情进行定义，但是只有类的对象(以及程序员)需要知道内部定义的详情。遵循以下思路：面向对象的系统可以被构建，以便每一部分

的实现基本上独立于其他部分的实现。这就是模块化的思想，并且有助于解决信息系统开发中最棘手的一些问题。模块化的方法可以帮助以多种方式解决软件开发面临的问题。

- 按照模块化方式构建的系统易于维护，子系统的改变很可能会对系统的其他部分产生不可预测的影响。
- 基于同样的原因，更新模块化的系统也更加简单。只要替换之前的模块，根据规范采用新的模块，就不会对其他模块产生影响。
- 构建可靠的系统更加容易。子系统可以被单独完整测试多次，而在之后系统集成的时候，需要解决的问题就会少得多。
- 模块化系统可以被开发为小型可管理的增量。假定每一个增量被设计用来提供有用且前后一致的功能包，就可以依次进行部署。

众所周知，编码并非软件开发中问题的主要来源。相比之下，需求和分析的问题更加普遍，而且它们的纠错代价更加昂贵。因此，对面向对象开发技术的关注就不能仅仅集中在编码上面，更应集中关心系统开发的其他方面。过去，面向对象界的大多数人都专注于编程语言，而不是分析和设计。最初在解决传统开发语言中那些不灵活的问题时，面向对象编程语言显示出强大的威力。对于软件工程来说，这种专注可以说是某种意义上的倒退，因为其过度注重实现机制，而不是它们所支持的底层思维过程。

真正的高效来源于解决前端的概念性问题，而不是后端的实现细节。修正在实现过程中显现出来的设计缺陷要花费高昂的代价，不如在早期发现它们。过早专注于实现会限制设计决策，常常会导致劣质产品的出现。面向对象开发方法鼓励软件开发者在软件生命周期内应用其概念来工作和思考。只有较好地识别、组织和理解应用领域的内在概念，才能有效表达出数据结构和函数的细节。

面向对象开发只有到了最后几个阶段才不是独立于编程语言的概念过程。从根本上讲，面向对象开发是一种思维方式，而不是一种编程技术。它的最大好处在于，帮助规划人员、开发者和客户清晰地表达抽象的概念，并将这些概念互相传达。它可以充当规约、分析、文档、接口以及编程的一种媒介。

1.2.5　面向对象分析与面向对象设计

面向对象分析(object oriented analysis，OOA)建立于以前的信息建模技术的基础之上，可以定义为一种分析方法，通过从问题域词汇中发现的类和对象的概念来考查需求。OOA的结果是一系列从问题域导出的“黑箱”对象。OOA通常使用“剧情”来帮助确定基本的对象行为。剧情是发生在问题域的连续的活动序列。在对给定的问题域进行面向对象分析时，“框架”的概念非常有用。框架是应用或应用子系统的骨架，包含一些具体或抽象的类。或者说，框架是一种特定的层次结构，包含描述某一问题域的抽象父类。当下流行的所有OOA方法的一个缺点就是都缺乏一种固定的模式。

在面向对象设计(object oriented design，OOD)阶段，注意的焦点从问题空间转移到解空间。OOD是一种包含对所设计系统的逻辑和物理的过程描述，以及系统的静态和动态模型的设计方法。

在OOA和OOD中，都存在着对重用性的关注。目前，面向对象技术的研究人员正在尝试定义“设计模式”这一概念。它是一种可重用的“财富”，可以应用于不同的问题域。通常，设计模式指的是一种多次出现的设计结构或解决方案。如果对它们进行系统的归类，就可以构成不同设计之间通信的基础。

OOD技术实际上早于OOA技术出现。目前OOA和OOD之间已经很难画出一条清晰的界线。因此，下面的描述给出一些常用OOA/OOD技术的(联合)概貌。

Meyer用语言作为表达设计的工具。

Booch的OOD技术扩展了他以前在Ada方面所做的工作。他采用一种“反复综合”的方法，包括以下过程：识别对象，识别对象的语义，识别对象之间的关系，实施，同时包含一系列迭代。Booch是最先使用类图、类分类图、类模板和对象图来描述OOD的人。

Wrifs-Brock的OOD技术是由职责代理驱动的。类职责卡(class responsibilities card)被用来记录负责特定功能的类。在确定类及其职责之后，再进行更详细的关系分析和子系统实施。

Rumbaugh使用3个模型来描述系统：①对象模型，描述系统中对象的静态结构；②动态模型，描述系统状态随时间变化的情况；③功能模型，描述系统中各个数据值的转变。对象图、状态转换图和数据流图分别被用于描述这3个模型。

Coad和Yourdon采用以下OOA步骤来确定多层的面向对象模型(包含5个层次)：找出类和对象，识别结构和关系，确定主题，定义属性，定义服务。这5个步骤分别对应模型的5个层次：类和对象层、主题层、结构层、属性层和服务层。他们的OOD方法既是多层次的，又是多方面的。层次结构和OOA一样。多方面包括：问题域，人与人的交互，任务管理和数据管理。

Ivar Jacobson 提出了Objectory方法(或Jacbson方法)，这是一种在瑞典Objective系统中开发的面向对象软件工程方法。Jacbson方法特别强调“Use Case”的使用。Use Case成为分析模型的基础，用交互图(interaction diagram)进一步描述后就形成设计模型。Use Case同时也驱动测试阶段的测试工作。到目前为止，Jacbson方法是最为完整的工业方法。

以上所述方法还有许多变种，无法一一列出。近年来，随着各种方法的演变，它们之间也互相融合。1995年，Booch、Rumbaugh和Jacbson联手合作，提出了第一版的UML(unified modelling language，统一建模语言)，目前已经成为面向对象建模语言的事实标准。

当组织向面向对象开发技术转向时，支持软件开发的管理活动也必然要有所改变。承诺使用面向对象技术意味着要改变开发过程、资源和组织结构。面向对象开发的迭代、原型以及无缝性，消除了传统开发模式不同阶段之间的界限。新的界限必须重新确定。同时，一些软件测度方法也不再适用。“代码行数”(lines of code，LOC)绝对过时了。重用类的数目，继承层次的深度，类与类之间关系的数目，对象之间的耦合度，类的个数以及大小显得更有意义。在面向对象的软件测度方面的工作还是相当新的，但也已经有了一些参考文献。

资源分配和人员配置都需要重新考虑。开发小组的规模逐步变小，擅长重用的专家开始吃香。重点应该放在重用而非LOC上。重用的真正实现需要一套全新的准则。在执行软

件合同的同时，库和应用框架也必须建立起来。长期的投资策略，以及对维护这些可重用财富的承诺和过程，变得更加重要。

至于软件质量保证，传统的测试活动仍是必需的，但它们的计时和定义必须有所改变。例如，将某个功能“走一遍”将牵涉激活剧情、一系列对象互相作用、发送消息、实现某个特定功能。测试面向对象系统是另一个需要进一步研究的课题。发布稳定的原型需要不同于以往控制结构化开发的产品的配置管理。

另一个关于管理方面需要注意的问题是合适的工具支持。面向对象的开发环境是必需的。同时需要的东西还包括：类库浏览器、渐增型编译器、支持类和对象语义的调试器、对设计和分析活动的图形化支持和引用检查、配置管理和版本控制工具，以及像类库一样的数据库应用。

除非面向对象开发的历史足以提供有关资源和消耗的数据，否则成本估算也是问题。计算公式中应该加入目前和未来的重用成本。最后，管理也必须明白在向面向对象方法转变的过程中可能遇到的风险，如消息传递、消息传递的爆炸性增长、动态内存分配和释放的代价。还有一些起步风险，比如对合适的工具、开发战略的熟悉，以及适当的培训、类库的开发等。

1.3　UML带来了什么

目前，面向对象已经成为软件开发的最重要方法。面向对象的兴起是从编程领域开始的。第一门面向对象语言 Smalltalk 的诞生宣告了面向对象开始进入软件领域。最初，人们只是为了改进开发效率，编写更容易管理、能够重用的代码，在编程语言中加入了封装、继承、多态等概念，以求得代码的优化。但分析和设计仍然以结构化的面向过程方法为主。

在实践中，人们很快就发现了问题：编程需要的对象不但不能从设计中自然而然地推导出来，而且强调连续性和过程化的结构化设计与事件驱动型的离散对象结构之间有着难以调和的矛盾。由于设计无法自然推导出对象结构，使得对象结构到底代表什么样的含义变得模糊不清；同时，设计如何指导编程，也成为困扰在人们心中的一大疑问。

为了解决这些困难，一批面向对象设计(OOD)方法开始出现，例如Booch 86、GOOD(通用面向对象开发)、HOOD(层次化面向对象设计)、OOSE(面向对象结构设计)等。这些方法作为面向对象设计方法的奠基者和开拓者，在不同的范围内拥有着各自的用户群，它们的应用为面向对象理论的发展提供了非常重要的实践和经验。

然而，随着应用程序变得进一步复杂，需求分析成为比设计更为重要的问题。这是因为人们虽然可以写出漂亮的代码，却常常被客户指责不符合需要而推翻重来。事实上如果不符合客户需求，再好的设计也没用。于是OOA(面向对象分析)方法开始走上舞台，其中最为重要的方法便是UML的前身：由Booch创造的Booch方法，由Jacobson创造的OOSE、Martin/Odell方法，以及由Rumbaugh创造的OMT、Shlaer/Mellor 方法。这些方法虽然各不相同，但它们的理念却是非常相似的。于是三位面向对象大师决定将他们各自的方法统一起

来，在1995年10月推出了第一个版本，称为“统一方法”(Unified Method 0.8)。随后，又以“统一建模语言”(UML)作为正式名称提交到OMG(对象管理组织)，在1997年1月正式成为一种标准建模语言。

1.3.1 什么是UML

UML(unified modeling language，统一建模语言)是一种建模语言，是一种用来为面向对象开发系统的产品进行可视化说明和编制文档的建模方法。在面向对象编程中，数据被封装(或绑定)到使用它们的函数中，形成整体，称为对象，对象之间通过消息相互联系。面向对象建模与设计是使用现实世界中的概念模型来思考问题的一种方法。对于理解问题、与应用领域专家交流、建模企业级应用、编写文档、设计程序和数据库来说，面向对象模型都非常有用。

UML 的应用领域很广泛，可以用于商业建模、软件开发建模的各个阶段，也可以用于其他类型的系统。UML是一种通用的建模语言，具有创建系统的静态结构和动态行为等多种结构模型的能力。UML本身并不复杂，具有可扩展性和通用性，适合为各种多变的系统建模。

UML是一种图形化建模语言，是面向对象分析与设计模型的一种标准表示。UML的目标如下。

- 易于使用，表达能力强，能进行可视化建模。
- 与具体的实现无关，可应用于任何语言平台和工具平台。
- 与具体的过程无关，可应用于任何软件开发过程。
- 简单并且可扩展，具有扩展和专有化机制以便于扩展，无须对核心概念进行修改。
- 为面向对象设计与开发中涌现出的高级概念(例如协作、框架、模式和组件)提供支持，强调在软件开发中对框架模式和组件的重用。
- 与最好的软件工程实践经验集成。
- 可升级，具有广阔的适用性和可用性。
- 有利于面对对象工具的市场成长。

需要说明的是，UML不是一种可视化的程序设计语言，而是一种可视化的建模语言；UML不是工具或知识库的规格说明，而是一种建模语言规格说明，是一种表示标准；UML既不是过程也不是方法，但允许任何一种过程和方法使用。

1.3.2 UML与面向对象软件开发

UML是一种建模语言，是一种标准表示，而不是一种方法(或方法学)。方法是一种把人的思考和行动结构化的明确方式，方法需要定义软件开发的步骤，告诉人们做什么，如何做，什么时候做，以及为什么要这么做。UML只定义了一些图以及它们的意义，UML的思想与方法无关。因此，我们会看到人们用各种方法来使用UML，而无论方法如何变化，它们的基础是UML视图，这就是UML的最终用途：为不同领域的人们提供统一的交流

标准。

软件开发的难点在于项目的参与者包括领域专家、软件设计开发人员、客户以及用户，他们之间交流的难题成为软件开发的最大难题。UML的重要性在于，表示方法的标准化有效地促进了拥有不同背景的人们的交流，有效地促进软件设计、开发和测试人员的相互理解。无论分析、设计和开发人员采取何种不同的方法或过程，他们提交的设计产品都是用UML描述的，这促进了相互的理解。

UML尽可能结合了世界范围内面向对象项目的成功经验，因而UML的价值在于体现了世界上面向对象方法实践的最好经验，并以建模语言的形式把它们打包，以适应开发大型复杂系统的要求。

在众多成功的软件设计与实现经验中，最突出的有两条，一条是注重系统架构的开发，另一条是注重过程的迭代和递增性。尽管UML本身对过程没有任何定义，但UML对任何使用它的方法(或过程)提出的要求是：支持用例驱动(use-case driven)、以架构为中心(architecture-centric)以及递增(incremental)和迭代(iterative)地开发。

注重架构意味着不仅要编写出大量的类和算法，还要设计出这些类和算法之间简单而有效的协作。所有高质量的软件中似乎含有大量这类协作，而近年来出现的软件设计模式也正在为这些协作起名和分类，使它们更易于重用。最好的架构就是概念集成(conceptual integrity)，它驱动整个项目注重开发模式并力图使它们简单。

迭代和递增的开发过程反映了项目开发的节奏。不成功的项目没有进度节奏，因为它们总是机会主义的，在工作中是被动的；成功的项目有自己的进度节奏，反映在它们有定期的版本发布过程，注重于对系统架构进行持续的改进。

UML的应用贯穿于软件开发的5个阶段，具体如下。

- 需求分析阶段。UML的用例视图可以表示客户的需求。通过用例建模，可以对外部的角色以及它们所需要的系统功能建模。角色和用例是用它们之间的关系、通信建模的。每个用例都指定了客户的需求：他或她需要系统干什么。不仅要对软件系统，对商业过程也要进行需求分析。
- 分析阶段。分析阶段主要考虑所要解决的问题，可用UML 的逻辑视图和动态视图来描述。类图描述系统的静态结构，协作图、序列图、活动图和状态图描述系统的动态特征。在分析阶段，只为问题域的类建模，不定义软件系统的解决方案的细节(如用户接口的类、数据库等)。
- 设计阶段。在设计阶段，把分析阶段的结果扩展成技术解决方案。加入新的类来提供技术基础结构，如用户接口、数据库操作等。分析阶段的领域问题类被嵌入这个技术基础结构中。设计阶段的结果是构造阶段的详细规格说明。
- 构造阶段。在构造(或程序设计)阶段，把设计阶段的类转换成某种面向对象程序设计语言的代码。在对UML表示的分析和设计模型进行转换时，最好不要直接把模型转换成代码。因为在早期阶段，模型是理解系统并对系统进行结构化的手段。
- 测试阶段。对系统的测试通常分为单元测试、集成测试、系统测试和接受测试几个不同级别。单元测试是针对几个类或一组类的测试，通常由程序员进行；集成测试集成组件和类，确认它们之间是否能够恰当地协作；系统测试把系统当作

“黑箱”，验证系统是否具有用户所要求的所有功能；接受测试由客户完成，与系统测试类似，验证系统是否满足所有的需求。不同的测试小组使用不同的UML图作为他们工作的基础：单元测试使用类图和类的规格说明，集成测试典型地使用组件图和协作图，而系统测试实现用例图来确认系统的行为符合这些图中的定义。

UML模型在面向对象软件开发中的使用非常普遍。软件开发通常按以下方式进行：一旦决定建立新的系统，就要写一份非正式的描述，说明软件应该做什么，这份描述被称作需求说明书(requirements specification)，通常与系统未来的用户磋商制定，并且可以作为用户和软件供应商之间正式合同的基础。

将完成后的需求说明书移交给负责编写软件的程序员或项目组，他们相对隔离地根据需求说明书编写程序。幸运的话，程序能够按时完成，不超出预算，而且能够满足最初方案下目标用户的需要。但在许多情况下，事情并不是这样。

许多软件项目的失败引发人们对软件开发方法的研究，他们试图了解项目为何失败，结果得到许多对如何改进软件开发过程的建议。这些建议通常以过程模型的形式，描述了开发所涉及的多个活动及其应该执行的次序。

软件开发过程模型可以用图解的形式表示。例如，图1-7表示一个非常简单的软件开发过程模型，其中直接从系统需求开始编写代码，没有中间步骤。图1-7中除了圆角矩形表示的过程之外，还显示了过程中每个阶段的产物。如果过程中的两个阶段顺次进行，一个阶段的输出通常就作为下一个阶段的输入，如虚线箭头所示。

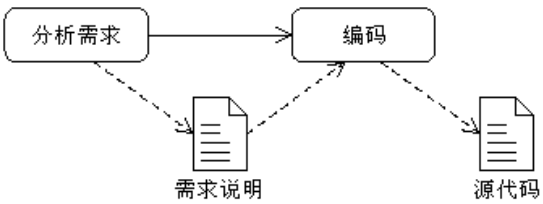

图1-7　软件开发过程模型

开发初期产生的需求说明书可以采取多种形式。书面的需求说明书可以是所需系统的非常不正规的概要轮廓，也可以是非常详细、井井有条的功能描述。在小规模的开发中，最初的系统描述甚至可能不会写下来，而只是程序员对需要什么的非正式理解。在有些情况下，可能会和未来的用户一起合作开发原型系统，成为后续开发工作的基础。上面所述的所有可能性都包括在“需求说明书”这个一般术语中，但并不意味着只有书面的文档才能够作为后继开发工作的起点。还要注意的是，图1-7没有描述整个软件生命周期。完整的项目计划还应该提供诸如项目管理、需求分析、质量保证和维护等关键活动。

单个程序员在编写简单的小程序时，几乎不需要相比图1-7更多的组织开发过程。有经验的程序员在写程序时心中会很清楚程序的数据和子程序结构，如果程序的行为不像预期的那样，他们能够直接对代码进行必要的修改。在某些情况下，这是完全适宜的工作方式。

然而，对比较大的程序，尤其是不止一个人参与开发时，在过程中引入更多的结构通

常是必要的。软件开发不再被看作单独的自由活动，而是分割为多个子任务，每个子任务一般都涉及一些中间文档资料的产生。

图1-8描述的是相比图1-7稍微复杂一些的软件开发过程模型。在这种情况下，程序员不再只是根据需求说明书编写代码，而是先创建结构图，用以表示程序的总体功能如何划分为一些模块或子程序，并说明这些子程序之间的调用关系。

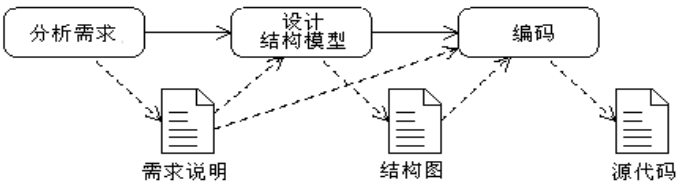

图1-8　稍复杂的软件开发过程模型

这个软件开发过程模型表明，结构图以需求说明书中包含的信息为基础，需求说明书和结构图在编写最终代码时都要使用。程序员可以使用结构图使程序的总体结构清楚明确，并在编写各个子过程的代码时参考需求说明书来核对所需功能的详细说明。

在软件开发期间产生的中间描述或文档称为模型。图1-8中给出的结构图在此意义上就是模型。模型展现系统的抽象视图，突出系统设计的某些重要方面，如子程序和它们的关系，而忽略大量的底层细节，如各个子程序代码的编写。因此，模型比系统的全部代码更容易理解，通常用来阐明系统的整体结构或体系结构。上面的结构图中包含的子程序调用结构就是这里所指的结构。

随着开发的系统规模更大、更复杂以及开发组人数的增加，需要在过程中引入更多的规定。这种复杂性不断增加的外部表现就是在开发期间使用更广泛的模型。实际上，软件设计有时就定义为构造一系列模型，这些模型越来越详细地描述系统的重要方面，直到获得对需求的充分理解，能够开始编程为止。

因此，使用模型是软件设计的中心，模型具有两个重要的优点，有助于处理重大软件开发中的复杂性。第一，系统要作为整体来理解可能过于复杂，模型则提供了对系统重要方面的简明描述。第二，模型为开发组的不同成员之间以及开发组和外界(如客户)之间提供了一种颇有价值的通信手段。

1.4　UML建模工具

使用建模语言需要相应的工具支持，即使手工在白板上画好了模型的草图，建模者也需要使用工具。因为模型中很多图的维护、同步和一致性检查等工作，人工做起来几乎是不可能的。

1.4.1　UML建模工具概述

自从用于产生程序的第一个可视化软件问世以来，建模工具(又称CASE 工具)一直不很

成熟，许多CASE 工具几乎和画图工具一样，仅提供建模语言和很少的一致性检查，增加一些方法的知识。经过人们不断地改进，今天的CASE 工具正在接近图的原始视觉效果，比如Rational Rose 工具，就是一种比较现代的建模工具。但是还有一些工具仍然比较粗糙，比如一般软件中很好用的“剪切”和“粘贴”功能，在这些工具中尚未实现。另外，每种工具都有属于自己的建模语言，或至少有自己的语言定义，这也限制了这些工具的发展。随着统一建模语言(UML)的发布，工具制造者现在可能会花较多的时间来提高工具质量，减少定义新的方法和语言所花费的时间。

现代的CASE 工具应提供下述功能。

- 画图(draw diagram)：CASE工具中必须提供方便作图和为图着色的功能，也必须具有智能，能够理解图的目的，知道简单的语义和规则。这样的特点带来的便利是，当建模者不适当或错误地使用模型元素时，工具能自动告警或禁止其操作。
- 积累(repository)：CASE工具中必须提供普通的积累功能，以便系统能够把收集到的模型信息存储下来。如果在某个图中改变某个类的名称，那么这种变化必须及时地反映到使用该类的所有其他图中。
- 导航(navigation)：CASE工具应该支持易于在模型元素之间导航的功能，也就是使建模者能够容易地从一个图到另一个图，跟踪模型元素或扩充对模型元素的描述。
- 多用户支持：CASE工具应能够使多个用户可以在一个模型上工作，且彼此之间没有干扰。
- 产生代码(generate code)：高级的CASE 工具一定要有产生代码的能力，该功能可以把模型中的所有信息翻译成代码框架，把代码框架作为实现阶段的基础。
- 逆转(reverse)：高级的CASE 工具一定要有阅读现成代码并依代码产生模型的能力，即模型可由代码生成。它与产生代码是互逆的两个过程。对开发者来说，可以用建模工具或编程方法建模。
- 集成(integrate)：CASE工具一定要能与其他工具集成，也就是与开发环境(比如编辑器、编译器和调试器)和企业工具(比如配置管理和版本控制系统)等的集成。
- 覆盖模型的所有抽象层：CASE工具应该能够容易地从对系统最上层的抽象描述向下导航至最低的代码层。这样，如果需要获得类中某个具体操作的代码，只需要在图中单击这个操作的名字即可。
- 模型互换：模型或来自某个模型的个别图应该能够从一个工具输出，然后再输入另一个工具。就像Java代码可在一个工具中产生，而后用在另一个工具中一样。模型互换功能也应该支持用明确定义的语言描述的模型之间的互换(输出/输入)。

1.4.2 常用的UML建模工具

目前，Rational Rose、PowerDesigner、Visio是三个比较常用的建模工具软件。

1. Rational Rose

Rational Rose是直接伴随UML发展而诞生的设计工具，它的出现就是为了对UML建模

提供支持，Rational Rose一开始没有对数据库端建模提供支持，但是现在的版本中已经加入数据库建模功能。Rational Rose对开发过程中的各种语义、模块、对象以及流程、状态等描述得比较好，主要体现在能够从各个方面和角度进行分析和设计，使软件的开发蓝图更清晰、内部结构更加明朗(但是仅仅对那些掌握UML的开发人员有效，对客户了解系统的功能和流程等并不一定很有效)，对系统的代码框架生成有很好的支持，但对数据库的开发管理和数据库端的迭代不是很好。

2. PowerDesigner

PowerDesigner原来是伴随数据库建模而发展起来的一种数据库建模工具。直到7.0版本才开始对面向对象开发提供支持，后来又引入对UML的支持。但是由于PowerDesigner侧重不一样，因此对数据库建模的支持很好，支持能够看到90%左右的数据库，对UML建模用到的各种图的支持比较滞后，但是最近得到加强。所以使用PowerDesigner进行UML开发的人并不多，很多人都用它进行数据库建模。如果使用UML，PowerDesigner的优点是生成代码时对Sybase产品PowerBuilder的支持很好(其他UML建模工具则不支持或者需要一定的插件)，对其他面向对象语言(如C++、Java、VB、C#等)的支持也不错。但是PowerDesigner好像继承了Sybase公司的一贯传统，对中国市场不是很看好，所以对中文的支持总是有这样或那样的问题。

3. Visio

UML建模工具Visio原来仅仅是一种画图工具，能够用来描述各种图形(从电路图到房屋结构图)，也是到了Visio 2000才开始引入软件分析和设计功能，直到代码生成的全部功能，可以说是目前能够用图形方式表达各种商业图形的最好工具(对软件开发中的UML支持仅仅是其中很少的一部分)。Visio跟微软Office产品能够很好兼容，能够把图形直接复制或内嵌到Word文档中。对于代码的生成，更多的是支持微软的产品，如VB、VC++、SQL Server等(这也是微软的传统)，所以Visio适合用于图形语义的描述，但是用于软件开发过程的迭代开发则有点牵强。

1.4.3　三种常用UML建模工具的性能对比

建模工具的基本功能就是作图。Rational Rose、PowerDesigner、Visio三种建模工具都支持UML模型图。其中，Rational Rose支持全系列的UML模型图，而且很容易体现迭代、用例驱动等特性，相关性最好；缺点是图形质量差，逻辑检查与控制差，没有Name和Code的区分(PowerDesigner的特性)，不太适合中国人，生成的文档不好也不适合自定义，也没有设计对象的字典可以快速查找。PowerDesigner也支持全系列的UML模型图，优点是图形质量好，生成的文档容易自定义，逻辑检查与控制好，有设计对象的字典可以快速查找和快速在图形中定位；缺点是相互之间的衔接比较麻烦，对UML和RUP不熟练的人使用PowerDesigner，体现不出迭代和用例驱动。相比起来，Visio的图形质量是最好的，但是衔接和相关性也是最差的，逻辑检查和控制也比较差。

另外，好的建模工具支持模型文档与代码、模型文档与数据库之间的双向转换。常用

的UML工具Rational Rose通过中间插件能够实现文档与代码、数据库的双向转换，该功能是通过中间插件实现的。PowerDesigner支持模型文档与代码、模型文档与数据库之间的双向转换，而且不需要插件。Visio通过VBA和宏实现模型文档与代码、模型文档与数据库之间的双向转换，用起来稍微麻烦。

Rational Rose提供相对最新、最完整的UML支持，PowerDesigner和Visio稍微滞后一点。Rational Rose有RUP体系的支持和一系列支持RUP的软件与Rational Rose协作，这一点PowerDesigner和Visio望尘莫及。

1.5 小结

本章介绍了面向对象和UML的基本概念，并结合软件系统和软件工程的概念，介绍了面向对象软件开发过程、UML与面向对象软件开发、UML建模工具等内容。希望通过本章的学习，读者能够理解并掌握面向对象、UML等概念，了解面向对象软件开发的过程，理解面向对象分析和设计建模的含义及目的，为学习统一建模语言(UML)打下基础。

1.6 思考练习

1. 面向对象的含义和特点是什么？
2. 比较面向过程方法和面向对象方法，并说明面向对象方法的有效性？
3. 通用的软件工程过程框架通常包含哪些活动？
4. 什么是对象？
5. 类和对象的关系是什么？
6. 封装的含义是什么？
7. 什么是信息隐藏？
8. 面向对象编程语言如何实现一般化和特殊化？
9. 举例说明多态的含义。
10. 什么是UML？
11. 常用的UML建模工具有哪些？各自的特点是什么？