第 4 章　面向对象方法

结构化分析和设计方法在一定程度上缓解了“软件危机”。但随着人们对软件提出的要求越来越高，结构化方法已经无法承担快速高效开发复杂软件系统的重任。20 世纪 80 年代逐渐成熟的面向对象方法学，使软件开发者对软件的分析、设计和编程等方面都有了全新的认识。由于“对象”概念的引入，将数据和方法封装在一起，提高了模块的聚合度，降低了模块的耦合度，更大程度上支持了软件重用，从而十分有效地降低了软件的复杂度，提高了软件开发的生产率。目前，面向对象方法学已成为软件开发者的第一选择。

根据考试大纲，本章要求考生掌握以下知识点：

- 面向对象的基本概念；
- 统一建模语言 UML；
- 可视化建模；
- 面向对象系统分析；
- 面向对象系统设计。

4.1　面向对象的基本概念

为了讨论面向对象（Object-Oriented，OO）的技术和方法，必须首先明确什么是“面向对象”？为什么要讨论面向对象的方法？什么是对象？对于这些问题，有许多不同的看法。其中 Booch、Coad/Yourdon 和 Jacobson 的方法在面向对象软件开发界得到了广泛的认可。特别值得一提的是统一建模语言（UML，Unified Modeling Language），该方法结合了 Booch、OMT 和 Jacobson 方法的优点，统一了符号体系，并从其他的方法和工程实践中吸收了许多经过实践检验的概念和技术。

Peter Coad 和 Edward Yourdon 曾提出了下列等式：

面向对象＝对象（Objects）＋类（Classes）＋继承（Inheritance）＋消息通信（Communication with Messages）

4.1.1　对象与封装

对象（Object）是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。面向对象的软件系统是由对象组成的，复杂的对象由比较简单的对象组合而成。也就是说，面向对象方法学使用对象分解取代了传统方法的功能分解。

对象三要素：对象标志、属性和服务。

对象标志（Object Identifier），也就是对象的名字，供系统内部唯一地识别对象。定义或使用对象时，均应指定对象标志。

属性（Attribute），也称状态（State）或数据（Data），用来描述对象的静态特征。在某些面向对象的程序设计语言中，属性通常被称为成员变量（Member Variable）或简称变量（Variable）。

服务（Service），也称操作（Operation）、行为（Behavior）或方法（Method）等，用来描述对象的动态特征。在某些面向对象的程序设计语言中，服务通常被称为成员函数（Member Function）或简称函数（Function）。

封装（Encapsulation）是对象的一个重要原则。它有两层含义：第一，对象是其全部属性和全部服务紧密结合而形成的一个不可分割的整体；第二，对象是一个不透明的黑盒子，表示对象状态的数据和实现操作的代码都被封装在黑盒子里面。使用一个对象的时候，只需知道它向外界提供的接口形式，无须知道它的数据结构细节和实现操作的算法。从外面看不见，也就更不可能从外面直接修改对象的私有属性了。

4.1.2　类与类库

类（class）是对象的抽象定义，是一组具有相同数据结构和相同操作的对象的集合。类的定义包括一组数据属性和在数据上的一组合法操作。类定义可以视为一个具有类似特性与共同行为的对象的模板，可用来产生对象。

类与对象是抽象描述与具体实例的关系，一个具体的对象被称为类的一个实例（Instance）。它们都可使用类中提供的函数。一个对象的状态则包含在它的实例变量中。

从物理特征上来看，类库和传统例程库是类似的，它们都是一种预先定义的程序库。类库是一种预先定义的程序库，它以程序模块的形式，按照类层次结构把一组类的定义和实现组织在一起。较上层的类代表了较一般的事物，相反，较下层的类代表了较具体的事物，很好地体现了面向对象机制的继承性、重载等许多特征。

类属类（Generic Class）仅描述了适用于一组类型的通用样板，由于其中所处理对象的数据类型尚未确定，因而程序员不可用类属类直接创建对象实例，即一个类属类并不是一种真正的类类型。

类属类必须经过实例化后才能成为可创建对象实例的类类型。类属类的实例化是指用某一数据类型替代类属类的类型参数。类属类定义中给出的类型参数称为形式类属参

数，类属类实例化时给出的类型参数称为实际类属参数。如果类属类实例化的实际类属参数可以是任何类型，那么这种类属类称为无约束类属类。然而在某些情况下，类属类可能要求实际类属参数必须具有某些特殊的性质，以使得在类属类中可应用某些特殊操作，这种类属类称为受约束类属类。类属类对类库的建设提供了强有力的支持。

4.1.3　继承与多态

继承（Inheritance）是使用已存在的定义作为基础建立新定义的技术，继承是面向对象方法学中的一个十分重要的概念。新类的定义可以是现存类所声明的数据、定义与新类所增加的声明的组合。新类复用现存类的定义，而不要求修改现存类。因为这种类的一部分已经实现和测试，故开发费用较少。现存类可当作父类（泛化类、基类或超类）来引用，则新类相应地可当作子类（特化类、子女类或派生类）来引用。

在面向对象技术中，多态考虑的是类与类之间的层次关系，以及类自身内部特定成员函数之间的关系问题，是解决功能和行为的再抽象问题。多态是指类中具有相似功能的不同函数是用同一个名称来实现，从而可以使用相同的调用方式来调用这些具有不同功能的同名函数。这也是人类思维方式的一种直接模拟，比如一个对象中有很多求两个数最大值的行为，虽然可以针对不同的数据类型，写很多不同名称的函数来实现，但事实上，它们的功能几乎完全相同。这时，就可以利用多态的特征，用统一的标志来完成这些功能。这样，就可以达到类的行为的再抽象，进而统一标志，减少程序中标志符的个数。

严格地说，多态性可分为四类，分别为过载多态（重载多态），强制多态，包含多态和参数多态，其中前两种统称为专用多态（特定多态），后面两种称为通用多态。

包含多态是研究类族中定义于不同类中的同名成员函数的多态行为，主要是通过虚函数来实现。包含多态最常见的例子就是子类型化，即一个类型是另一类型的子类型。

参数多态的应用比较广泛，被称为最纯的多态。这是因为同一对象、函数或过程能以一致的形式用于不同的类型。参数多态与类属（类模板）相关联，类属是一个可以参数化的模板，其中包含的操作所涉及的类型必须用类型参数实例化。这样，由类模板实例化的各类都具有相同的操作，而操作对象的类型却各不相同。

过载多态是同一算子（操作符、函数名等）被用来表示不同的功能，通过上下文以决定一个算子所代表的功能，即通过语法对不同语义的对象使用相同的算子，编译能够消除这一模糊。

强制多态是通过语义操作把一个变元的类型加以变换，以符合一个函数的要求，如果不做这一强制性变换将出现类型错误。类型的变换可在编译时完成，通常是隐式地进行，当然也可以在动态运行时来做。

从实现的角度来看，多态可划分为两类，分别是编译时的多态和运行时的多态。前者是在编译的过程中确定了同名操作的具体操作对象，而后者则是在程序运行过程中才

动态地确定操作所针对的具体对象。这种确定操作的具体对象的过程就是联编（编联，束定或绑定）。联编是指计算机程序自身彼此关联的过程，也就是把一个标志符名和一个存储地址联系在一起的过程。用面向对象的术语讲，就是把一条消息和一个对象的方法相结合的过程。

按照联编进行的阶段的不同，可以分为两种不同的联编方法，分别为静态联编和动态联编，这两种联编过程分别对应着多态的两种实现方式。

联编工作在编译连接阶段完成的情况称为静态联编。因为联编过程是在程序开始执行之前进行的，因此有时也称为早期联编或前联编。在编译和连接过程中，系统就可以根据类型匹配等特征确定程序中操作调用与执行该操作代码的关系，其确定了某一个同名标志到底是要调用哪一段程序代码。有些多态类型，其同名操作的具体对象能够在编译、连接阶段确定，通过静态联编解决，比如过载，强制和参数多态等。

与静态联编相对应，联编工作在程序运行阶段完成的情况称为动态联编，也称为晚期联编或后联编。在编译、连接过程中无法解决的联编问题，要等到程序开始运行之后再来确定，包含多态的操作对象的确定就是通过动态联编完成的。

4.1.4　消息通信

消息（Message）是指向对象发出的服务请求，它应该含有下述信息：提供服务的对象标志、消息名、输入信息和回答信息。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，相反，它是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。

消息通信（Communication with Messages）也是面向对象方法学中的一条重要原则，它与对象的封装原则密不可分。封装使对象成为一些各司其职、互不干扰的独立单位；消息通信则为它们提供了唯一合法的动态联系途径，使它们的行为能够互相配合，构成一个有机的系统。

只有同时使用对象、类、继承与消息通信，才是真正面向对象的方法。

4.1.5　面向对象方法学的优点

与面向过程相比，面向对象方法学具有以下优点。

（1）与人类习惯的思维方法一致：面向对象方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使软件开发的方法与过程尽可能接近人类认识世界解决问题的方法与过程，也就是使描述问题的“问题域”与解决问题的“解域”在结构上尽可能一致。

（2）稳定性好：传统的软件开发方法基于功能分析与功能分解，软件结构紧密依赖于系统所要完成的功能，当功能需求发生变化时将引起软件结构的整体修改。由于用

户需求变化大部分是针对功能的，因此这样的系统是不稳定的。

面向对象的方法用对象模拟问题域中的实体，以对象为中心构造软件系统，当系统的功能需求变化时并不会引起软件结构的整体变化。由于现实世界中的实体是相对稳定的，因此以对象为中心构造的软件系统也是比较稳定的。

（3）可重用性好：面向对象方法学在利用可重用的软件成分构造新的软件系统时有很大的灵活性。继承机制与多态性使得子类不仅可以重用其父类的数据结构与程序代码，并且可以方便地修改和扩充，而这种修改并不影响对原有类的使用。

（4）较易开发大型软件产品：由于用面向对象方法学开发软件时，构成软件系统的每个对象相对独立。因此，可以把一个大型软件产品分解成一系列相互独立的小产品来处理。这不仅降低了开发的技术难度，而且也使得对开发工作的管理变得容易多了。

（5）可维护性好：面向对象的软件比较容易理解、容易修改、容易测试。

4.2　UML 概述

在 20 世纪的 80～90 年代，面向对象的分析与设计（OOA&D）方法获得了长足的发展，而且相关的研究也十分活跃，涌现了一大批新的方法学。其中最著名的是 Booch 的 Booch 1993、Jacobson 的 OOSE 和 Rumbaugh 的 OMT 方法。而 UML 正是在融合了 Booch、Rumbaugh 和 Jacobson 方法论的基础上形成的标准建模语言。

4.2.1　UML 是什么

UML（Unified Modeling Language，统一建模语言）是用于系统的可视化建模语言，尽管它常与建模 OO 软件系统相关联，但由于其内建了大量扩展机制，还可以应用于更多的领域中，例如工作流程、业务领域等。

（1）UML 是一种语言：UML 在软件领域中的地位与价值就像“1、2、3、+、–、…”等符号在数学领域中的地位一样。它为软件开发人员之间提供了一种用于交流的词汇表，一种用于软件蓝图的标准语言。

（2）UML 是一种可视化语言：UML 只是一组图形符号，它的每个符号都有明确语义，是一种直观、可视化的语言。

（3）UML 是一种可用于详细描述的语言：UML 所建的模型是精确的、无歧义和完整的，它适合于对所有重要的分析、设计和实现决策进行详细描述。

（4）UML 是一种构造语言：UML 虽然不是一种可视化的编程语言，但其与各种编程语言直接相连，而且有较好的映射关系，这种映射允许进行正向工程、逆向工程。

（5）UML 是一种文档化语言：它适合于建立系统体系结构及其所有的细节文档。

4.2.2　UML 的发展历史

面向对象建模语言，最早出现于 20 世纪 70 年代中期，而在 20 世纪 80 年代末开始进入快速发展阶段，截止到 1994 年，就从不到 10 种发展到 50 多种。由于每种语言、方法的创造者都极力推崇自己的成果，于是爆发了“面向对象技术的方法大战”，也从此流传着一句戏言：“方法学家和恐怖分子的差别在于，方法学家不能谈判。”

而在 1994 年之后，各种方法论逐渐拉开了差距，以 Grady Booch 提出的 Booch 方法和 James Rumbaugh 提出的 OMT（Object Modeling Technique，对象建模技术）成为了可视化建模语言的主导。而 Ivar Jacobson 的 Objectory 方法则成为最强有力的方法。

Booch 是面向对象方法最早的倡导者之一。他在 1984 年《Ada 软件工程》(“Software Engineering with Ada”）一书中就描述了面向对象软件开发的基本问题。1991 年，他在《面向对象的设计》(“Object-Oriented Design”）一书中，将以前针对 Ada 的工作扩展到整个面向对象设计领域。他对继承和类的阐述特别值得借鉴。Booch1993 比较适合于系统的设计和构造。

Runbaugh 等人提出了面向对象的建模技术（OMT），采用了面向对象的概念并引入各种独立于程序设计语言的表示符号。这种方法用对象模型、动态模型、功能模型和用例模型共同完成对整个系统的建模，所定义的概念和符号可用于软件开发的分析、设计和实现的全过程，软件开发人员不必在开发过程的不同阶段进行概念和符号的转换。OMT-2 特别适用于分析和描述以数据为中心的信息系统。

Jacobson 于 1994 年提出了面向对象软件工程（OOSE）的方法。其最大特点是面向用例，并在用例的描述中引入了外部角色的概念。用例的概念贯穿于整个开发过程（包括对系统的测试和验证），是精确描述需求的重要武器。目前在学术界和工业界已普遍接受用例的概念，并认为是面向对象技术走向第二代的标志。OOSE 比较适合支持商务工程和需求分析。

面对众多的建模语言，首先，用户无力区分不同建模语言之间的差别和使用范围。其次，各种建模语言其实各有长短。第三，由于不同的用户使用不同的建模语言和不同建模语言表达方式上的差异，使得用户之间的沟通方面出现了困难。要解决以上问题就必须在综合分析各种不同建模语言的优缺点及适用情况的基础上统一各种不同的建模语言。

1994 年 10 月，Grady Booch 和 Jim Rumbaugh 开始了这项工作。首先他们将 Booch 1993 和 OMT-2 统一起来，并于 1995 年 10 月发布了第一个公开版本称为标准方法 UM0.8（Unified Method）。1995 年秋，OOSE 的创始人 Ivar Jacobson 加盟到这项工作中，经过 Booch、Rumbaugh 和 Jacobson 三人的共同努力，于 1996 年 6 月和 10 月分别发布了两个新的版本（UML0.9 和 UML0.91），并将 UM 重新命为 UML。

1996 年，UML 被 OMG 提议为 OO 可视化建模语言的推荐标准，UML 被提交。1997 年，OMG 采纳了 UML，一个开放的 OO 可视化建模语言工业标准诞生了。UML 在经

历了 1.1、1.2 和 1.4 三个版本的演变之后进行了一次大的调整，升级为 2.0 版标准。目前 UML 的最新版是 2010 年 11 月发布的 2.4 版。

4.2.3　UML 结构

UML 的结构如图 4-1 所示。

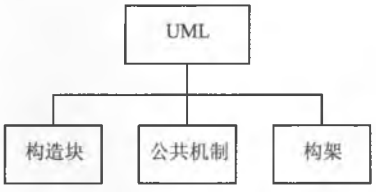

图 4-1　UML 结构示意图

1．构造块

构造块也就是基本的 UML 建模元素、关系和图。

（1）建模元素：包括结构元素（类、接口、协作、用例、活动类、组件、节点等）、行业元素（交互、状态机）、分组元素（包）、注解元素。

（2）关系：包括关联关系、依赖关系、泛化关系、实现关系。

（3）图：在 UML 1.x 中包括 9 种不同的图，当升级为 2.x 之后，增加至 14 种图。分为表示系统静态结构的静态模型（包括类图、对象图、复合结构图、构件图、部署图、包图），以及表示系统动态结构的动态模型（包括用例图、活动图、状态机图、顺序图、通信图、定时图、交互概观图、制品图）。

2．公共机制

公共机制是指达到特定目标的公共 UML 方法，主要包括规格说明、修饰、公共分类和扩展机制四种。

（1）规格说明：规格说明是元素语义的文本描述，它是模型真正的核心。

（2）修饰：UML 为每一个模型元素设置了一个简单的记号，还可以通过修饰来表达更多的信息。

（3）公共分类：包括类元与实体（类元表示概念，而实体表示具体的实体）、接口和实现（接口用来定义契约，而是实现就是具体的内容）两组公共分类。

（4）扩展机制：包括约束（添加新规则来扩展元素的语义）、构造型（用于定义新的 UML 建模元素）、标记值（添加新的特殊信息来扩展模型元素的规格说明）。

3．构架

UML对系统构架的定义是：系统的组织结构，包括系统分解的组成部分、它们的关联性、交互、机制和指导原则，这些提供系统设计的信息。具体来说，是指五个系统视图。

（1）逻辑视图：以问题域的语汇组成的类和对象集合。

（2）进程视图：可执行线程和进程作为活动类的建模，它是逻辑视图的一次执行实例。

（3）实现视图：对组成基于系统的物理代码的文件和组件进行建模。

（4）部署视图：把组件物理地部署到一组物理的、可计算节点上。

（5）用例视图：最基本的需求分析模型。

4.2.4　UML的主要特点

UML统一了Booch、OMT、OOSE和其他面向对象方法的基本概念和符号，同时汇集了面向对象领域中很多人的思想，是从优秀的面向对象方法和丰富的计算机科学实践中总结而成的。

目前UML是最先进、实用的标准建模语言，而且还在不断发展进化之中。

UML是一种建模语言而不是一种方法，其中并不包括过程的概念，它本身是独立于过程的，可以在使用过程中使用它。不过与UML结合最好的是用例驱动的、以体系结构为中心的、迭代的、增量的开发过程。

4.2.5　UML的应用领域

作为一种标准建模语言，UML的核心是以面向对象的思想来描述客观世界，具有广阔的应用领域。目前主要应用领域是在软件系统建模，但是它同样可以应用于其他领域，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。总之，UML是一个通用的标准建模语言，可以对任何系统的动态行为和静态行为进行建模。同时，标准建模语言UML可以对信息系统提供从需求分析到系统维护的整个生命周期提供有效的支持。在需求分析阶段，可以通过用例模型来捕获和组织用户的需求，分析系统对于用户的价值。通过用例建模，描述对系统感兴趣的外部角色及其对系统（用例）的功能要求。分析阶段主要关心问题域中的主要概念（如抽象、类和对象等）和机制，以及这些概念之间的相互协作，并用UML类图来描述。至于类之间的协作关系则可以用交互图和顺序图来描述。在分析阶段，只对问题域的对象（现实世界的概念）建模，而不考虑定义软件系统中技术细节的类（如处理用户接口、数据库、通信和并行性等问题的类）。由于这些技术细节将在设计阶段引入，因此设计阶段为构造阶段提供更详细的规格说明。

编码阶段的主要任务是将设计阶段的设计结果转换成为实际的代码。在设计阶段需要注意的是不要过早地考虑设计结果要用哪种编程语言实现。如果过早地考虑这个问题，会使设计工作陷入细节的泥潭，不利于对模型进行全面理解。

标准建模语言 UML 还可以对测试阶段提供有效的支持。不同的测试阶段可以使用不同的 UML 图作为测试的依据。比如，在单元测试阶段，可以使用类图和类的规格说明来进行测试；在集成阶段，可以使用合作图，活动图和部署图；系统测试和验收测试阶段则可以使用顺序图和用例图来验证系统的外部行为。

总之，标准建模语言 UML 能够用面向对象的方法描述任何类型的系统，并对系统开发从需求调研到测试和维护的各个阶段进行有效的支持。

4.3　UML 的建模机制

UML 是一个通用的可视化建模语言，用于对软件进行描述、可视化处理、构造和建立软件系统的文档。它记录了对必须构造的系统的决定和理解，可用于对系统的理解、设计、浏览、配置、维护和信息控制。UML 适用于各种软件开发方法、软件生命周期的各个阶段、各种应用领域，以及各种开发工具，UML 是一种总结了以往建模技术的经验并吸收当今优秀成果的标准建模方法。UML 包括概念的语义，表示法和说明，提供了静态、动态、系统环境及组织结构的模型。它可被交互的可视化建模工具所支持，这些工具提供了代码生成器和报表生成器。UML 标准并没有定义一种标准的开发过程，但它适用于迭代式的开发过程。它是为支持大部分现存的面向对象开发过程而设计的。

UML 不是一种可视化的编程语言，但是 UML 描述的模型可与各种编程语言直接相连，即可把用 UML 描述的模型映射成编程语言。

UML2.x 中包括 14 种不同的图，分为表示系统静态结构的静态模型（包括类图、对象图、复合结构图、构件图、部署图、包图），以及表示系统动态结构的动态模型（包括用例图、活动图、状态机图、顺序图、通信图、定时图、交互概观图、制品图）。

4.3.1　用例图

用例实例是在系统中执行的一系列动作，这些动作将生成特定参与者可见的价值结果。一个用例定义一组用例实例。它确定了一个和系统参与者进行交互、并可由系统执行的动作序列。用例模型描述的是外部执行者（Actor）所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。

在 UML 中，用例表示为一个椭圆。图 4-2 显示了一个个人图书管理系统的用例图。其中，“新增书籍信息”、“查询书籍信息”、“修改书籍信息”、“登记外借情况”、“查询外借情况”、“统计金额与册数”等都是用例的实例。

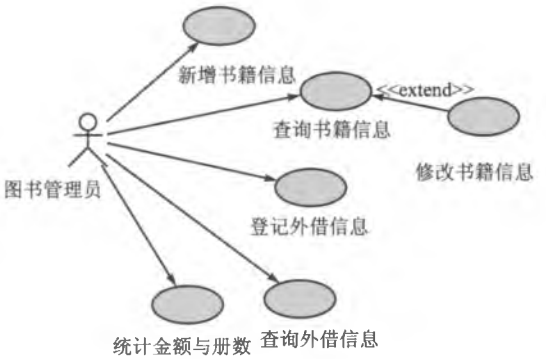

图 4-2　用例图示例

1. 参与者

参与者（Actor）代表与系统接口的任何事物或人，它是指代表某一种特定功能的角色，参与者都是虚拟的概念。在 UML 中，用一个小人表示参与者。

图 4-2 中的“图书管理员”就是参与者。对于该系统来说，可以充当图书管理员角色的可能有多个人，由于他们对于系统而言均起着相同的作用，扮演相同的角色，因此只使用一个参与者表示。切忌不要为每一个可能与系统交互的真人画出一个参与者。

可以通过以下问题来帮助你寻找到系统的相关参与者。

- 谁是系统的主要用户？
- 谁从系统获得信息？
- 谁向系统提供信息？
- 谁从系统删除信息？
- 谁支付、维护系统？
- 谁管理系统？
- 系统需要与哪些其他系统交互？
- 系统需要操纵哪些硬件？
- 在预设的时间内，有事情自动发生吗？
- 系统从哪里获得信息？
- 谁对系统的特定需求感兴趣？
- 几个人在扮演同样的角色吗？
- 一个人扮演几个不同的角色吗？
- 系统使用外部资源吗？
- 系统用在什么地方？

2. 用例

用例（Use Case）是对系统行为的动态描述，它可以促进设计人员、开发人员与用户的沟通，理解正确的需求，还可以划分系统与外部实体的界限，是系统设计的起点。在识别出参与者之后，可以使用下列问题帮助识别用例：

- 每个参与者的任务是什么？
- 有参与者将要创建、存储、修改、删除或读取系统中的信息吗？
- 什么用例会创建、存储、修改、删除或读取这个信息吗？
- 参与者需要通知系统外部的突然变化吗？
- 需要通知参与者系统中正在发生的事情吗？
- 什么用例将支持和维护系统？
- 所有的功能需求都对应到用例中了吗？
- 系统需要何种输入输出？输入从何处来？输出到何处？
- 当前运行系统的主要问题是什么？

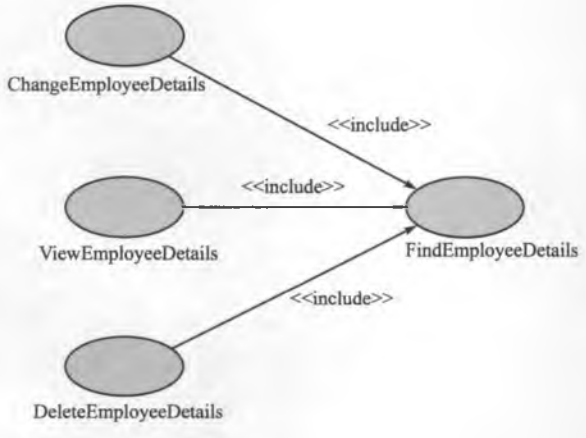

图 4-3　包含关系示例图

3. 包含和扩展

两个用例之间的关系可以主要概括为两种情况。一种是用于重用的包含关系，用构造型<<include>>表示；另一种是用于分离出不同的行为，用构造型<<extend>>表示。

（1）包含关系：当你可以从两个或两个以上的原始用例中提取公共行为，或者发现能够使用一个组件来实现某一个用例的部分功能是很重要的事时，应该使用包含关系来表示它们。

（2）扩展关系：如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种事情。我们可以断定将这个用例分为一个主用例和一个或多个辅用例描述可能更加清晰。

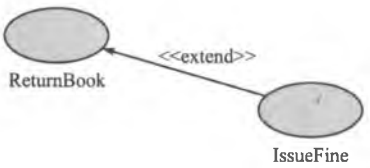

图 4-4　扩展关系示例图

4.3.2　类图和对象图

在面向对象建模技术中，对象是指现实世界中有意义的事物具有封装性和自治性的特点，而类是指具有相同属性和行为的一组对象。类（Class）、对象（Object）和它们之间的关联是面向对象技术中最基本的元素。对于一个想要描述的系统，其类模型和对象模型揭示了系统的结构。在 UML 中，类和对象模型分别由类图和对象图表示。类图技术是 OO 方法的核心。图 4-5 显示了一个小型图书管理系统的类图。

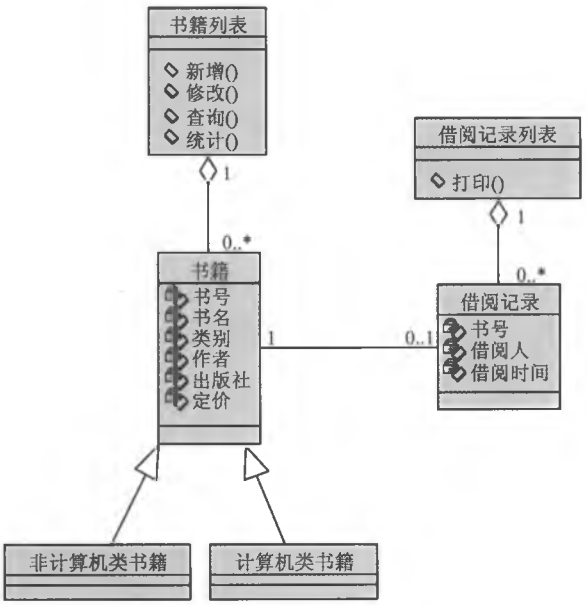

图 4-5　类图

1．类和对象

对象与我们对客观世界的理解相关。它通常用来描述客观世界中的某个具体的事物。因为类（Class）是对一组具有相同属性，表现相同行为的对象的抽象。因此，对象是类的实例（Instance）。在 UML 中，类的可视化表示为一个划分成 3 个格子的长方形（下面两个格子可省略）。图 4-5 中，“书籍”、“借阅记录”等都是一个类。

（1）类的命名：最顶部的格子包含类的名字。类的命名应尽量用应用领域中的术语，应明确、无歧义，以利于开发人员与用户之间的相互理解和交流。

（2）类的属性：中间的格子包含类的属性，用以描述该类对象的共同特点。该项可省略。图 4-5 中“书籍”类有“书名”、“书号”等特性。UML 规定类的属性的语法为：“可见性属性名：类型=默认值{约束特性}”。

- 可见性包括 Public、Private 和 Protected，分别用+、–、#号表示。
- 类型表示该属性的种类：它可以是基本数据类型，例如整数、实数、布尔型等，也可以是用户自定义的类型。一般它由所涉及的程序设计语言确定。
- 约束特性则是用户对该属性性质一个约束的说明。例如“{只读}”说明该属性是只读属性。

（3）类的操作（Operation）：该项可省略。操作用于修改、检索类的属性或执行某些动作。操作通常也被称为功能，但是它们被约束在类的内部，只能作用到该类的对象上。操作名、返回类型和参数表组成操作界面。UML 规定操作的语法为“可见性：操作名（参数表）：返回类型{约束特性}”。

类图描述了类和类之间的静态关系。定义了类之后，就可以定义类之间的各种关系。

2. 类之间的关系

在建立抽象模型时，由于很少有类会单独存在，大多数都将会以某种方式彼此通讯，因此我们还需要描述这些类之间的关系。关系是事物间的连接，在面向对象建模中，有四个很重要的关系。

（1）依赖关系。有两个元素 A、B，如果元素 A 的变化会引起元素 B 的变化，则称元素 B 依赖（Dependency）于元素 A。在 UML 中，使用带箭头的虚线表示依赖关系，如图 4-6 所示。

在类中，依赖关系有多种表现形式，如：一个类向另一个类发消息；一个类是另一个类的成员；一个类是另一个类的某个操作参数等。

（2）泛化关系。泛化关系描述了一般事物与该事物中的特殊种类之间的关系，也就是父类与子类之间的关系。继承关系是泛化关系的反关系，也就是说子类是从父类中继承的，而父类则是子类的泛化。在 UML 中，使用带空心箭头的实线表示，箭头指向父类，如图 4-7 所示。

图 4-6　依赖关系的图示

图 4-7　泛化关系的图示

在 UML 中，对泛化关系有三个要求：

- 子类应与父类完全一致，父类所具有的关联、属性和操作，子元素都应具有；
- 子类中除了与父类一致的信息外，还包括额外的信息。

- 可以使用父类实例的地方，也可以使用子类实例。

在如图 4-5 所示的例子中，“书籍”与“非计算机类书籍”之间就是泛化关系。

（3）关联关系。关联（Association）表示两个类的实例之间存在的某种语义上的联系。例如，一个老师在某学校工作，一个学校有多间教室。我们就认为教室和学校、学校和教室之间存在着关联关系。

关联关系为类之间的通信提供了一种方式，它是所有关系中最通用、语义最弱的。关联关系通常可以再细分成以下几种。

- 聚合关系：聚合关系（Aggregation）是关联关系的特例。聚合关系是表示一种整体和部分的关系。如一个电话机包含一个话筒，一个电脑包含显示器，键盘和主机，这些都是聚合关系的例子。在 UML 中，聚合关系用一个带空心菱形的实线表示，空心菱形指向的是代表“整体”的类，如图 4-8 所示。
- 组合关系：如果聚合关系中的表示“部分”的类的存在，与表示“整体”的类有着紧密的关系，例如“公司”与“部门”之间的关系，那么就应该使用“组合”关系来表示。在 UML 中，使用带有实心菱形的实线表示。

（4）实现关系。实现关系是用来规定接口和实现接口的类或组件之间的关系。接口是操作的集合，这些操作用于规定类或组件的服务。在 UML 中，使用一个带空心箭头的虚线表示，如图 4-9 所示。

图 4-8　聚合关系的图示

图 4-9　实现关系的图示

3．类图

类图（Class Diagram）描述类和类之间的静态关系。与数据模型不同，它不仅显示了信息的结构，同时还描述了系统的行为。类图是面向对象建模中最重要的模型。

4．对象图

UML 中对象图与类图具有相同的表示形式。对象图可以看做是类图的一个实例。对象是类的实例，对象之间的链（Link）是类之间的关联的实例。对象与类的图形表示相似，均为划分成两个格子的长方形（下面的格子可省略）。上面的格子是对象名，对象名下有下画线；下面的格子记录属性值。链的图形表示与关联相似。对象图常用于表示复杂的类图的一个实例。

4.3.3　交互图

交互图（Interactive Diagram）是表示各组对象如何依某种行为进行协作的模型。通常可以使用一个交互图来表示和说明一个用例的行为。在 UML 中，包括两种不同形式

的交互图，分别是强调对象交互行为顺序的顺序图，强调对象协作的协作图，它们之间没有什么本质不同，只是排版不尽相同而已。

1．顺序图

顺序图（Sequence Diagram）用来描述对象之间动态的交互关系，着重体现对象间消息传递的时间顺序。顺序图允许直观地表示出对象的生存期。在生存期内，对象可以对输入消息做出响应，并且可以发送信息。图 4-10 是顺序图示例。

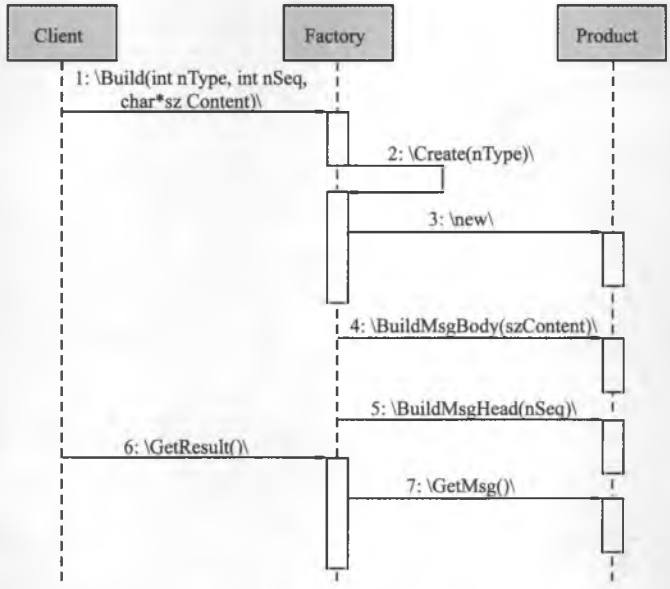

图 4-10　顺序图示例

正如图 4-10 所示，顺序图存在两个轴，水平轴表示不同的对象，即图中的 Client、Factory、Product 等；而垂直轴表示时间，表示对象及类的生命周期。

对象间的通信通过在对象的生命线间画消息来表示。消息的箭头指明消息的类型。顺序图中的消息可以是信号、操作调用或类似于 C++中的 RPC（Remote Procedure Calls）和 Java 中的 RMI（Remote Method Invocation）。当收到消息时，接收对象立即开始执行活动，即对象被激活了。通过在对象生命线上显示一个细长矩形框来表示激活。

消息可以用消息名及参数来标志，消息也可带有顺序号。消息还可带有条件表达式，表示分支或决定是否发送消息。如果用于表示分支，则每个分支是相互排斥的，即在某一时刻仅可发送分支中的一个消息。

2．协作图

协作图（Collaboration Diagram）用于描述相互合作的对象间的交互关系和链接关系。虽然顺序图和协作图都用来描述对象间的交互关系，但侧重点不一样。顺序图着重

体现交互的时间顺序，协作图则着重体现交互对象间的静态链接关系。图 4-11 是与图 4-10 相对应的协作图。

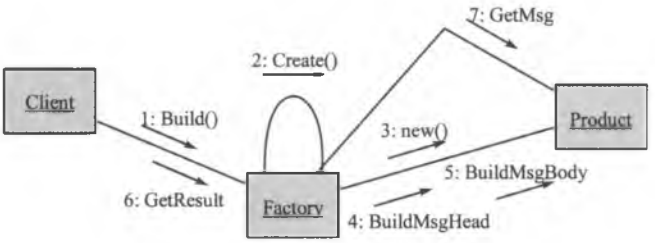

图 4-11　协作图示例

4.3.4　状态图

状态图（State Diagram）用来描述对象状态和事件之间的关系。我们通常用状态图来描述单个对象的行为。它确定了由事件序列引出的状态序列，但并不是所有的类都需要使用状态图来描述它的行为。只有那些具有重要交互行为的类，我们才会使用状态图来描述。

图 4-12 是一个数码冲印店的订单状态图实例，正如图 4-12 所示，状态图包括以下部分。

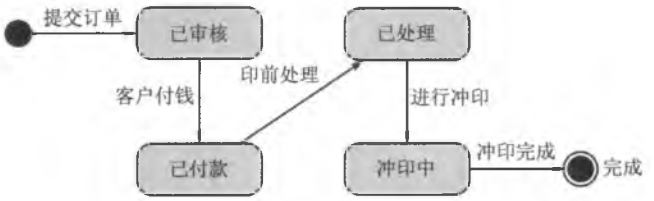

图 4-12　状态图示例

（1）状态：又称为中间状态，用圆角矩形框表示；

（2）初始状态：又称为初态，用一个黑色的实心圆圈表示，在一张状态图中只能够有一个初始状态；

（3）结束状态：又称为终态，在黑色的实心圆圈外面套上一个空间圆，在一张状态图中可能有多个结束状态；

（4）状态转移：用箭头说明状态的转移情况，并用文字说明引发这个状态变化的相应事件是什么。

一个状态也可能被细分为多个子状态，如果将这些子状态都描绘出来的话，那这个状态就是复合状态。

状态图适合用于表述在不同用例之间的对象行为，但并不适合于表述包括若干协作的对象行为。通常不会需要对系统中的每一个类绘制相应的状态图，而会在业务流程、控制对象、用户界面的设计方面使用状态图。

4.3.5　活动图

活动图用来表示系统中各种活动的次序，它的应用非常广泛，既可用来描述用例的工作流程，也可以用来描述类中某个方法的操作行为。活动图是由状态图变化而来的，它们各自用于不同的目的。活动图依据对象状态的变化来捕获动作（将要执行的工作或活动）与动作的结果。活动图中一个活动结束后将立即进入下一个活动（在状态图中状态的变迁可能需要事件的触发）。

1. 基本活动图

图 4-13 给出了一个基本活动图的例子。

正如图 4-13 所示，活动图中与状态图类似，包括了初始状态、终止状态，以及中间的活动状态，每个活动之间，也就是一种状态的变迁。在活动图中，还引入了以下几个概念。

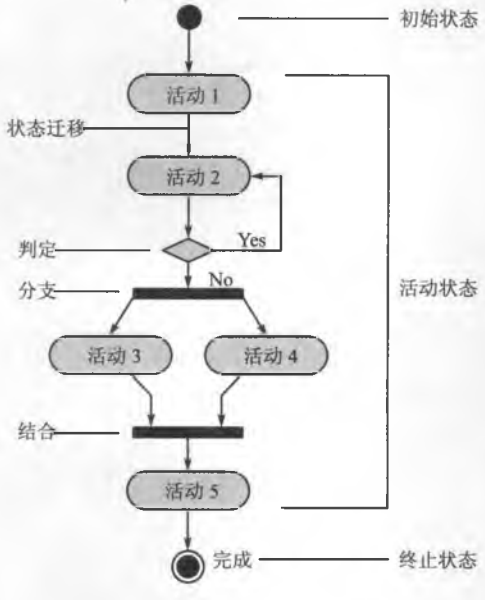

图 4-13　活动图示例

（1）判定：说明基于某些表达式的选择性路径，在 UML 中使用菱形表示。

（2）分叉与结合：由于活动图建模经常会遇到并发流，因此在 UML 中引入了如图 4-13 所示的粗线来表示分叉和结合。

2. 带泳道的活动图

在前面说明的基本活动图中，虽然能够描述系统发生了什么，但无法说明完成这个活动的对象。针对 OOP 而言，这就意味着活动图没有描述出各个活动由哪个类来完成。要想解决这个问题，可以通过泳道来解决这一问题。它将活动图的逻辑描述与顺序图、协作图的责任描述结合起来。下面是一个使用了泳道的例子，如图 4-14 所示。

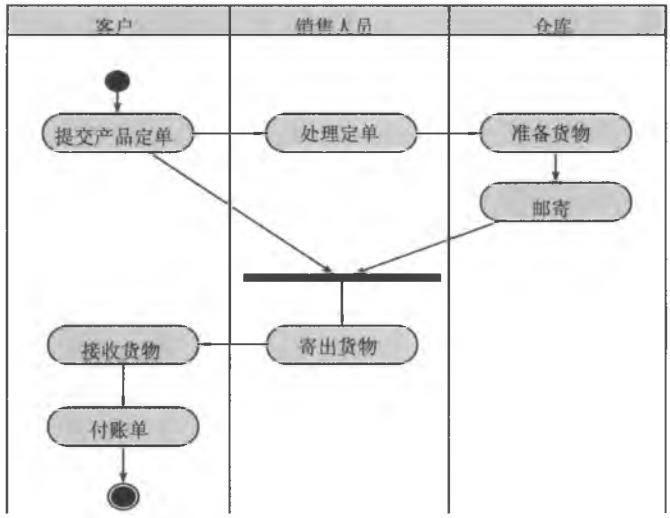

图 4-14　带泳道活动图示例

3. 对象流

在活动图中对象可以作为活动的输入或输出，对象与活动间的输入/输出关系由虚线箭头来表示。如果仅表示对象受到某一活动的影响，则可用不带箭头的虚线来连接对象与活动。

4. 信号

在活动图中可以通过信号的发送和接收标记来表示信号的发送和接收，发送和接收标志也可与对象相连，用于表示消息的发送者和接收者。

4.3.6　构件图

构件图是面向对象系统的物理方面进行建模时要用的两种图之一。它可以有效地显示一组构件，以及它们之间的关系。构件图中通常包括构件、接口，以及各种关系。图 4-15 所示是一个构件图的例子。

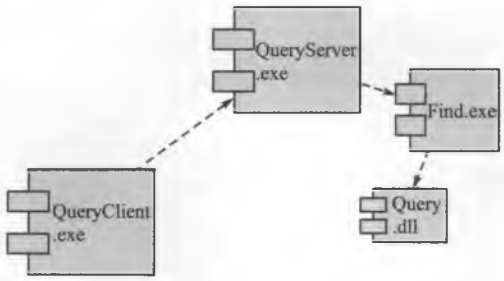

图 4-15　构件图示例

通常构件指的是源代码文件、二进制代码文件和可执行文件等。而构件图就是用来显示编译、链接或执行时构件之间的依赖关系。例如，在图 4-15 中，就是说明 Query Client.exe 将通过调用 QueryServer.exe 来完成相应的功能，而 QueryServer.exe 则需要 Find.exe 的支持，Find.exe 在实现时调用了 Query.dll。

通常，我们使用构件图可以完成以下工作。

（1）对源代码进行建模：这样可以清晰地表示出各个不同源程序文件之间的关系。

（2）对可执行体的发布建模：如图 4-15 所示，将清晰地表示出各个可执行文件、DLL 文件之间的关系。

（3）对物理数据库建模：用来表示各种类型的数据库、表之间的关系。

（4）对可调整的系统建模：例如，对于应用了负载均衡、故障恢复等系统的建模。

在绘制构件图时，应该注意侧重于描述系统的静态实现视图的一个方面，图形不要过于简化，应该为构件图取一个直观的名称，在绘制时避免产生线的交叉。

4.3.7　部署图

部署图，也称为实施图，它和构件图一样，是面向对象系统的物理方面建模的两种图之一。构件图是说明构件之间的逻辑关系，而部署图则在此基础上更近一步，描述系统硬件的物理拓扑结构，以及在此结构上执行的软件。部署图可以显示计算结点的拓扑结构和通信路径、结点上运行的软件构件，常常用于帮助理解分布式系统。

图 4-16 就是与图 4-15 对应的部署图，这样的图示可以使系统的安装、部署更为简单。在部署图中，通常包括以下一些关键的组成部分。

1．节点和连接

节点（Node）代表一个物理设备，以及其上运行的软件系统，如一台 Windows NT 主机、一个 PC 终端、一台打印机、一个传感器等。

如图 4-16 所示，“客户端：PC”和“服务器”就是两个节点。在 UML 中，使用一个立方体表示一个节点，节点名放在左上角。节点之间的连线表示系统之间进行交互的

通信路径，在UML中被称为连接。通信类型则放在连接旁边的“□□”之间，表示所用的通信协议或网络类型。

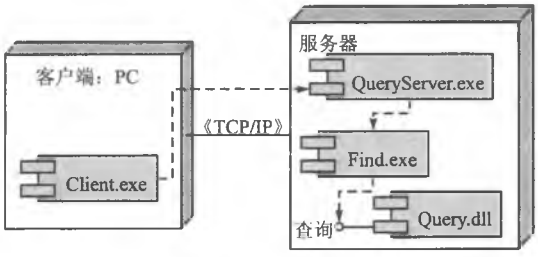

图4-16　部署图示例

2．构件和接口

在部署图中，构件代表可执行的物理代码模块，如一个可执行程序。逻辑上构件可以与类图中的包或类对应。例如在图4-15中，“服务器”节点中包含“QueryServer.exe”、“Find.exe”和“Query.dll”3个构件。

在面向对象方法中，类和构件等元素并不是所有的属性和操作都对外可见。它们对外提供了可见操作和属性，称为类和构件的接口。界面可以表示为一头是小圆圈的直线。在图4-16中，“Query.dll”构件提供了一个“查询”接口。

4.4　面向对象分析

综观计算机软件发展史，许多新方法和新技术都是在编程领域首先兴起，进而发展到软件生命周期的前期阶段——分析与设计阶段。结构化方法经历了从“结构化编程”、“结构化设计”到“结构化分析”的发展历程，面向对象的方法也经历了从“面向对象的编程”（Object-Oriented Programming，OOP）、“面向对象的设计”（Object-Oriented Design，OOD）到“面向对象的分析”（Object-Oriented Analysis，OOA）的发展历程。1989年之后，面向对象方法的研究重点开始转向软件生命周期的分析阶段，并将OOA和OOD密切地联系在一起，出现了一大批面向对象的分析与设计（OOA&D）方法。到目前为止，公开发表并具有一定影响的OOA&D方法已达数十种。

由于各种OOA方法所强调的重点与该方法的主要特色不同，因此所产生的OOA模型从整体形态、结构框架到具体内容都有较大的差异。

4.4.1　OMT方法简介

1991年，James Rumbaugh在《面向对象的建模与设计》（*Object-Oriented Modeling*

and Design）一书中提出了面向对象分析与设计的 OMT（Object Modeling Technique）方法。20 世纪 90 年代中期，笔者曾使用 OMT 方法开发了“印典”、“书林”等排版系统。本书的 OOA 模型主要依据 OMT 方法，同时参考了 Peter Coad 和 Edward Yourdon 的 OOA 模型。

OMT 方法的 OOA 模型包括对象模型、动态模型和功能模型。

- 对象模型表示静态的、结构化的系统的“数据”性质。它是对模拟客观世界实体的对象及对象彼此间的关系的映射，描述了系统的静态结构。通常用类图表示。
- 动态模型表示瞬时的、行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。通常用状态图表示。
- 功能模型表示变化的系统的“功能”性质，由于它指明了系统应该“做什么”，因此更直接地反映了用户对目标系统的需求。通常用数据流图表示。

OMT 方法的三个模型，分别从三个不同侧面描述了所要开发的系统：功能模型指明了系统应该“做什么”；动态模型明确了什么时候做（即在何种状态下接受了什么事件的触发）；对象模型则定义了做事情的实体。这三种模型相互补充、相互配合，三者之间具有下述关系。

（1）动态模型展示了对象模型中每个对象的状态及它接受事件和改变状态时所执的操作；而功能模型中的处理则对应于对象模型中的对象所提供的服务。

（2）对象模型展示了动态模型中是谁改变了状态和经受了操作；而功能模型中的处理则可能产生动态模型中的事件。

（3）对象模型展示了功能模型中的动作者、数据存储和流的结构；而动态模型则展示了功能模型中执行加工的顺序。

1. 建立对象模型

Peter Coad 和 Edward Yourdon 在 1991 年出版的《面向对象的分析》（*Object-Oriented Analysis*）一书中指出，复杂系统的对象模型通常由下述五个层次组成：类及对象层、结构层、主题层、属性层和服务层。上述五个层次对应着建立对象模型的五项主要活动：确定类与对象、确定结构与关联、划分主题、定义属性和定义服务。但这五项活动完全没必要顺序完成，也无需彻底完成一项活动之后再开始另外一项活动。

（1）确定类与对象。类与对象是在问题域中客观存在的，系统分析的重要任务之一就是找出这些类与对象。首先找出所有候选的类与对象，然后进行反复筛选，删除不正确或不必要的类与对象。

（2）确定结构与关联。结构与关联反应了对象（或类）之间的关系，主要有以下几种。

- 一般-特殊结构（Generalization-Specialization Structure），又称分类结构（Classification Structure），是由一组具有一般-特殊关系（继承关系）的类所组成的结构。一般-特殊关系（Generalization-Specialization Relation）的表达式为 is a kind of。

- 整体-部分结构（Whole-Part Structure），又称组装结构（Composition Structure），是由一组具有整体-部分关系（组成关系）的类所组成的结构。整体-部分关系（Whole-Part Relation）的表达式为 has a。
- 实例关联（Instance Connection），即一个类的属性中含有另一个类的实例（对象），它反映了对象之间的静态联系。
- 消息关联（Message Connection），即一个对象在执行自己的服务时需要通过消息请求另一个对象为它完成某个服务，它反映了对象之间的动态联系。

（3）划分主题。在开发大型、复杂软件系统的过程中，为了降低复杂程度，需要把系统划分成几个不同的主题。注意，应该按问题域而不是用功能分解方法来确定主题，应该按照使不同主题内的对象相互间依赖和交互最少的原则来确定主题。

（4）定义属性。为了发现对象的属性，首先考虑借鉴以往的 OOA 结果，看看相同或相似的问题域是否有已开发的 OOA 模型，尽可能复用其中同类对象的属性定义。然后，按照问题域的实际情况，以系统责任为目标进行正确的抽象，从而找出每一对象应有的属性。

（5）定义服务。发现和定义对象的服务，也应借鉴以往同类系统的 OOA 结果并尽可能加以复用。然后，研究问题域和系统责任以明确各个对象应该设立哪些服务，以及如何定义这些服务。

2．建立动态模型

建立动态模型的第一步，是编写典型交互行为的脚本。虽然脚本中不可能包括每个偶然事件，但至少必须保证不遗漏常见的交互行为。第二步，从脚本中提取出事件，确定触发每个事件的动作对象及接受事件的目标对象。第三步，排列事件发生的次序，确定每个对象可能有的状态及状态间的转换关系，并用状态图描绘它们。最后，比较各个对象的状态图，检查它们之间的一致性，确保事件之间的匹配。

3．建立功能模型

OMT 方法中的功能模型实际上就是结构化方法中的数据流图。从这点看，OMT 方法并不是“纯”面向对象的。这是 OMT 方法的一大缺陷。

1992 年，Ivar Jacobson 在《面向对象的软件工程——用例驱动的途径》（*Object-Oriented Software Engineering, A Use Case Driven Approach*）中首次提出了“用例”（use case）的概念。随后，有人提出以用例图取代数据流图进行需求分析和建立功能模型，这应该被看做是对 OMT 方法的重大改进。使用用例图建立起来的系统模型也被称为用例模型。

4.4.2　用 UML 进行分析

首先，我们结合图 4-17 来领会一下 OO 的世界观。

在结构化的理论基础下，我们会将应用分解成为一个个功能模块、子功能模块、功能接口等，它完全与现实问题域的东西没有具体的联系。从图 4-17 可以看出，使用 OO 的思想所建立的系统模型就是对问题域的完整的、直接的映射。也就是从现实世界中抽象出一个模型，然后在计算机中实现出来。

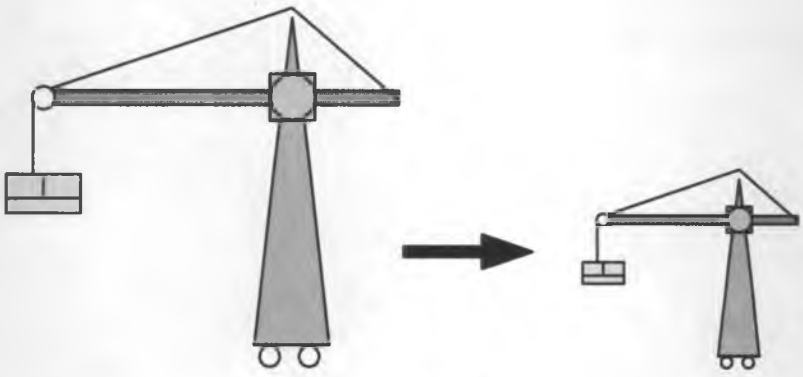

图 4-17　OO 世界观

也就是说，从面向对象的角度来看，世界是由对象组成的。由于任何给出的业务功能都是由一整套协作的对象所支持实现的。因此，采用面向对象分析方法时，我们需要识别出与系统相关的对象，并且描述这些对象的属性，以及它们之间的关系；另一方面，我们还需要了解这些对象之间是如何协作，从而完成系统功能的。

综上所述，采用面向对象分析方法，整个分析阶段通常包括以下两个工作任务：建立一个反映问题域静态关系的概念模型，通常使用类图来表示；建立一个反应系统行为的动态模型，即用例模型。

1．建立域模型

“问题域”是指一个包含现实世界事物与概念的领域，这些事物和概念与所设计的系统要解决的问题有关。而建立概念模型，又称为问题域建模或域建模，也就是找到代表那些事物与概念的“对象”。

（1）寻找类。寻找类的方法有很多种，其中最广泛应用的莫过于“名词动词法”，也就是阅读需求文档，找出名词和名词短语，从中提取对象与属性，通常带有所有格的名词是属性而非对象。找出动词与动词短语，从中提取操作与关联。

第一步：列出所有的备选类。即将需求中所有的名词和名词短语列举出来。

第二步：决定候选类。很显然，不是所有的名词和名词短语都是系统中的一个合适的候选类，因为有的在系统之外，有的与系统不相关，有的名词概念较小，只适合于作为某个候选类的属性。因此，我们必须对其进行筛选，将不合适的滤掉。

不过，在采用名词动词法寻找类的时候，有些团队会陷入一个误区，那就是花费过多的时间，甚至到了“咬文嚼字”的地步，这样会使分析迷失方向。

（2）确定类之间的关联。当我们完成了类的寻找工作之后，需要理清这些类之间的层次关系，如关联、继承、聚合等，然后通过 UML 的类图工具将这些关系记录下来。如图 4-18 所示是一个与个人藏书管理系统相关的领域模型。

当完成了这些关联关系的建模之后，需要细化这些关系的描述。例如，我们从图 4-18 中就会产生一些疑问，如一本书可以有几条借阅记录等。这就需要将这些关系之间的多重性进行一些描述，当然这些描述必须是来源于业务规则，与领域相关的，如果还不清晰的可以暂时放在一边。如图 4-19 所示是一个修改过后的模型。

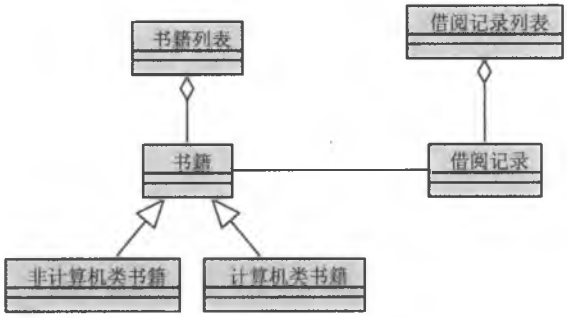

图 4-18　域模型示例（1）

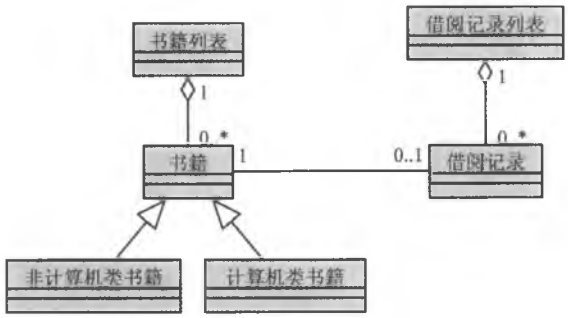

图 4-19　域模型示例（2）

通过关联关系的识别与建模，我们可以对问题域中的各个类之间层次结构关系、协作关系有一个相对完整的认识与理解。

（3）为类添加职责。在前面两步中，我们找到了与问题域本质相关的主要概念类，而且还理清了它们之间的协作关系，此后就应该为这些类添加其相应的主要职责。什么是类的职责呢？类的职责包括以下两方面的内容：

- 类所维护的知识即成员变量，属性；
- 类能够执行的行为也就是成员方法。

不过要注意的是，为类添加职责与找到类之间的关联关系一样，这个阶段也只是找到那些主要的、明显的、与业务规则相关的部分。切忌在这个阶段不断地细化，甚至引入一些与具体实现相关的技术内容，如数据库、分布式对象之类的东西。

你可以通过 CRC 技术来发现这些类的职责，然后在原来的类模型的基础上进行完善，图 4-20 是一个完成了主要类职责之后的概念模型。

（4）域模型的详细度。在前面，我们一再强调，在做域模型时要适度，那么到底应该详细到什么程度呢？有些关于 OO 的书中建议只列出类，以及类之间的主要关联关系，不要对关联关系进行描述，更不要体现类的职责；而有些 OO 的书则认为应该将这些东西都列出来。其实干巴巴地讨论这个问题是没有任何意义的，根据笔者长期的实践，总结了以下两点。

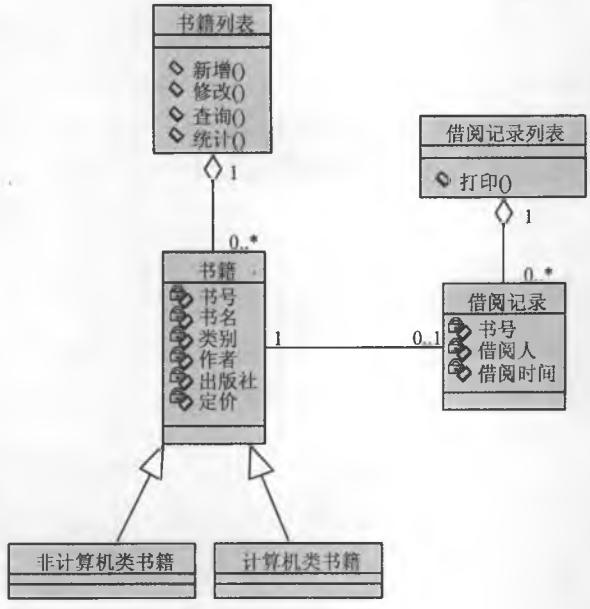

图 4-20　完整的域模型示例

- 由于概念模型的目的是让开发团队对问题域建立一个全貌式的了解，并作为今后进一步深化建立的基础，因此不妨取中庸之道，将需求描述中所谈到的主要内容都反馈到概念模型中去，而无须顾及其是关联关系还是职责描述；
- 概念模型是在开发过程中生产出来的第一个系统的静态模型，随着开发活动的推进，该模型将会随之加入新内容，改掉旧内容，逐渐完善，演变完整。

总之，模型不是我们要生产的目标产物，而是过程中的一个辅助工作，只要能够利用其帮助团队更好的开发，那就是详细也罢、简约也罢，都是好模型。

2. 建立用例模型

有些制作精细的“模型车”不管从外观还是内部结构上都与真车一模一样，但是却不能够像真车那样行驶，缺了什么呢？缺的是每个零件只是“神似”，而非“真是”，换一句话说就是处于静态状态下是相像的，但是无法动起来，无法实现这些零件本该实现的功能，这就使得模型车无法真正地开起来。

当我们完成了概念模型的建立时，仅仅是打造完成了一辆“模型车”而已，只能够帮助开发团队更好地理解系统所涉及的问题领域，帮助对要开发系统相关的业务知识建立正确、完整、清晰的理解。但还无法开始构建系统。要想让“模型车”开起来，最重要的就是建立反映系统行为的动态模型，也就是用例模型。

（1）用例是什么？ Ivar Jacobson 是这样描述的：“用例实例是在系统中执行的一系列动作，这些动作将生成特定参与者可见的价值结果。一个用例定义一组用例实例。”

首先，我们从定义中得知用例是由一组用例实例组成的，用例实例也就是常说的“使用场景”，是用户使用系统的一个实际的、特定的场景。其次，我们可以知道，用例应该给参与者带来可见的价值，这点很关键。最后，用例是在系统中的。

（2）用例模型是如何产生的？ 用例技术自从诞生起，就被广为关注，现在已经成为现代软件工程公认的最佳需求分析技术之一。近几年来，在国内的开发团队中也开始逐渐被接受，不过由于国内认识用例是从 UML 开始的，因此许多人误把用例图当作用例模型。另外，还有许多初用用例分析的开发组织，误将其当作一种分解技术，致使做出来的分析与功能分解酷似，以致失去了用例分析技术带来的益处。

其实，用例分析技术是一种需求合成技术，它的合成过程如图 4-21 所示。

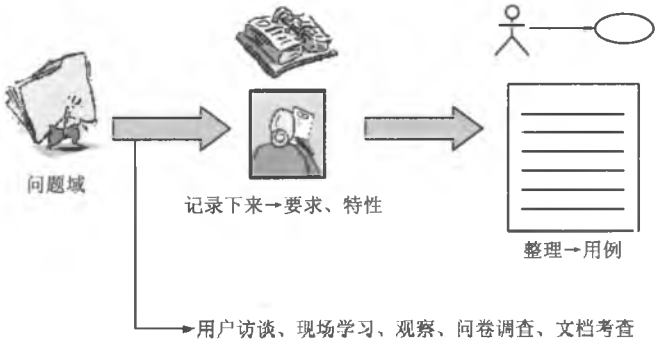

图 4-21　用例建模过程示意图

用例分析技术也就是采用现有的需求捕获技术从客户、原有系统、文档中找到需求，记录下来，然后从这个零散的要求、特性中进行整理、提炼，从而建立用例模型。千万记住，不要尝试向你的客户询问诸如“你还有什么用例吗？”之类的问题。

（3）识别参与者。 参与者（Actor）是同系统交互的所有事物，该角色不仅可以由人承担，还可以是其他系统、硬件设备、甚至是时钟。

- 其他系统：当你的系统需要与其他系统交互时，如在开发 ATM 柜员机系统时，银行后台系统就是一个参与者。
- 硬件设备：如果你的系统需要与硬件设备交互时，如在开发 IC 卡门禁系统时，IC 卡读写器就是一个参与者。
- 时钟：当你的系统需要定时触发时，时钟就是一个参与者，如在开发 Foxmail 中的“定时自动接收”功能时，就需要引入时钟作为参与者。

要注意的是，参与者一定在系统之外，不是系统的一部分。我们可以通过谁使用这个系统？谁安装这个系统？谁启动这个系统？谁维护这个系统？谁关闭这个系统？哪些其他的系统使用这个系统？谁从这个系统获取信息？谁为这个系统提供信息？是否有事情自动在预计的时间发生？等一系列问题来帮助发现系统的参与者。

（4）合并需求获得用例。将参与者都找到之后，接下来就是仔细地检查参与者，为每一个参与者确定用例。而其中的依据主要来源于已经获取得到的“特征表”。

- 将特征分配给相应的参与者：首先，要将这些捕获到的特征，分配给与其相关的参与者，以便针对每一个参与者进行工作，而无遗漏。
- 进行合并操作：在合并之前，我们首先还要明确为什么要合并，知道了合并的目的，也就会使我们选择正确的合并操作。一个用例就是一个对参与者来说可见的价值结果，因此合并的根据就是使其能够组合为一个可见的价值结果。

合并后，将产生用例，而用例的命名应该注意采用“动词（短语）+名词（短语）”的形式，而且最好能够对其进行编号，这也是实现跟踪管理的重要技巧，通过编号可以将用户的需求落实到特定的用例中去。

（5）绘制成用例图。最后将识别到的参与者，以及合并生成的用例通过用例图的形式整理出来，以获得用例模型的框架，也算是得到一个中间的成果，如图 4-22 所示。

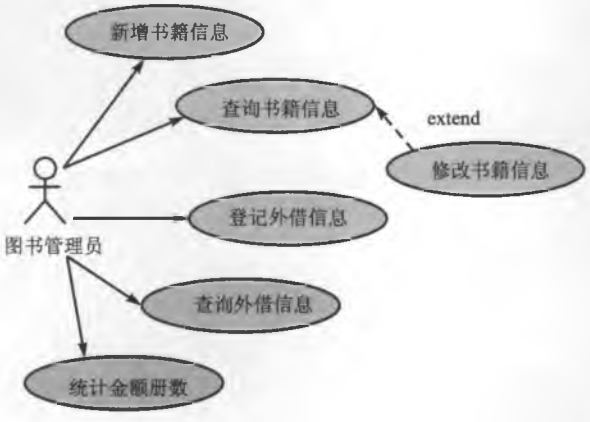

图 4-22　用例图示例

千万不要以为到此用例分析就结束了。这仅仅是一个好的开端，接下来的工作才是最重要的一环，也是用例发挥作用的关键。

（6）细化用例描述。用例的描述可以迭代完成，先对一些重要的用例编制相对细致的用例描述，对于一些不重要的，可以留待以后再补充完成。

用例描述通过包括以下几个部分完成。

① 用例名称：应该与用例图相符，并写上其相应的编号；

② 简要说明：对该用例对参与者所传递的价值结果进行描述，应注意语言简要，使用用户能够阅读的自然语言。

③ 事件流：也就是该用例所完成的工作步骤，在编写时应该注意以下几点。

- 使用简单的语法：主语明确，语义易于理解。
- 明确写出“谁控制球”：即在事件流描述中，让读者直观地了解是参与者在控制还是系统在控制。
- 从俯视的角度来编写：指出参与者的动作，以及系统的响应，也就是第三者的角度。
- 显示过程向前推移：也就是第一步都有前进的感觉（例如，用户按下 tab 键作为一个事件就是不合适的）。
- 显示参与者的意图而非动作（光有动作，让人不容易直接从事件流中理解用例）；
- 包括“合理的活动集”（带数据的请求、系统确认、更改内部、返回结果）。
- 用“确认”而非“检查是否”，例如“系统确认所输入的信息中书名未有重名”。
- 可选择地提及时间限制。

另外，事件流的编写过程也是可以分阶段、迭代进行的，对于优先级高的用例花更多的时间进行细化；对优先级低的用例可以先简略地将主要事件流描述清楚。另外，对于一些事件流较为复杂的，可以在用例描述中引用顺序图、状态图、协作图等手段进行描述。

④ 非功能要求：主要对该用例所涉及的非功能性需求进行描述。由于其通常很难在事件流中进行表述，因此单列为一小节进行阐述。这些需求通过包括法律法规、应用程序标准、质量属性（可用性、可靠性、性能、支持性等）、兼容性、可移植性，以及设计约束等方面的需求。在这些需求的描述方面，一定要注意使其可度量、可验证，否则就容易流于形式，形同摆设。

⑤ 前置条件：是执行用例之前必须存在的系统状态，这部分内容如果在当前不容易确定可以在后面再细化。

⑥ 后置条件：用例执行完毕系统可能处于的一组状态，这部分内容如果在当时不容易确定也可以在后面再细化。

⑦ 扩展点：如果包括扩展或包含用例，则写出扩展或包含用例名，并说明在什么情况下使用。在本例中，由于用例图里没有相应的内容，因此可以直接写无。如果有，则应该在编写事件流的同时进行编写。

⑧ 优先级：说明用户对该用例的期望值，可以为以后开发时制订先后顺序。可以采用满意度/不满意度指标进行说明，其中满意度的值为 0～5，指如果实现该功能，用户的满意程度；而不满意度的值也为 0～5，是指如果不实现该功能，用户的不满意程度。

下面是图 4-22 中所对应的其中一个较重要的用例“新增书籍信息”的用例描述，这里给出的是一个相对完整的版本，这些内容不一定要一次完成。

1．用例名称：

新增书籍信息（UC01）

2．简要说明：

录入新购书籍信息，并自动存储建档。

3．事件流：

3.1　基本事件流

1）图书管理员向系统发出“新增书籍信息”请求；

2）系统要求图书管理员选择要新增的书籍是计算机类还是非计算机类；

3）图书管理员做出选择后，显示相应界面，让图书管理员输入信息，并自动根据书号规则生成书号；

4）图书管理员输入书籍的相关信息，包括：书名、作者、出版社、ISBN 号、开本、页数、定价、是否有 CDROM；

5）系统确认输入的信息中书名未有重名；

6）系统将所输入的信息存储建档。

3.2　扩展事件流

5a）如果输入的书名有重名现象，则显示出重名的书籍，并要求图书管理员选择修改书名或取消输入；

5a1）图书管理员选择取消输入，则结束用例，不做存储建档工作；

5a2）图书管理员选择修改书名后，转到 5）

4．非功能需求

无特殊要求。

5．前置条件

用户进入图书管理系统。

6．后置条件

完成新书信息的存储建档。

7．扩展点

无

8．优先级

最高（满意度 5，不满意度 5）

4.5 面向对象设计

与结构化方法不同，面向对象的方法并不强调分析与设计之间严格的阶段划分。因为OOA与OOD所采用的概念、原则和表示法都是一致的，二者之间不存在鸿沟，不需要从分析文档到设计文档的转换，所以有些工作无论在分析时进行还是在设计时进行都不存在障碍。当然，OOA与OOD仍然有不同的分工和侧重点。

关于OOA与OOD的关系，目前有两种不同的观点。

一种观点是继续沿用传统的分工——分析着眼于系统“做什么”，设计解决“怎么做”的问题。而Peter Coad和Edward Yourdon的OOA&D方法则采用了另外一种分工方式——分析阶段只考虑问题域和系统责任，建立一个独立于实现的OOA模型；设计阶段考虑与实现有关的因素，对OOA模型进行调整并补充与实现有关的部分，形成OOD模型。

4.5.1 Coad/Yourdon方法

Coad/Yourdon方法严格区分了面向对象分析OOA和面向对象设计OOD。该方法利用五个层次和活动定义和记录系统行为输入和输出。这五个层次的活动包括：

（1）发现类及对象。描述如何发现类及对象。从应用领域开始识别类及对象，形成整个应用的基础，然后，据此分析系统的责任。

（2）识别结构。该阶段分为两个步骤。第一，识别一般-特殊结构，该结构捕获了识别出的类的层次结构；第二，识别整体-部分结构，该结构用来表示一个对象如何成为另一个对象的一部分，以及多个对象如何组装成更大的对象。

（3）定义主题。主题由一组类及对象组成，用于将类及对象模型划分为更大的单位，便于理解。

（4）定义属性。其中包括定义类的实例（对象）之间的实例连接。

（5）定义服务。其中包括定义对象之间的消息连接。

在面向对象分析阶段，经过五个层次的活动后的结果是一个分成五个层次的问题域模型，包括主题、类及对象、结构、属性和服务五个层次，由类及对象图表示。5个层次活动的顺序并不重要。

面向对象设计模型需要进一步区分以下4个部分。

1. 问题域的设计

问题域部分的设计是任何OOD方法都必须完成的工作，它主要是对OOA结果进行改进和精化，并将其由问题域转化到解域，具体来说，有以下几个方面。

（1）属性：由于有些属性在分析阶段有助于问题的理解，而到了设计阶段则可以由其他属性导出或根本没必要保留。因此，应将去掉它们。相反地，为了实现服务算法还需要增加相应的一些属性。

（2）服务：OOA 只给出了服务的接口，其具体实现算法要在 OOD 阶段完成。

（3）类及对象：在 OOA 阶段有助于问题理解的一些类在 OOD 阶段成为冗余，需要删除，而为了优化调整继承关系还要增加一些类。所有的类都确定以后还要明确哪些类的对象会引发哪些类创建新对象。

（4）结构：对类间结构进行优化调整。

（5）对象行为：明确对象间消息传递的实现算法，依据动态模型确定对象间消息发送的先后顺序，并设计相应算法，协调对象的行为。

2．人-机交互界面的设计

有些设计方法并没有提到交互界面的设计，一方面是因为这些系统中交互界面不十分重要；另一方面是因为这部分的设计很有规律，设计方法也比较成熟，但为完整起见，仍将其列出。主要工作包括。

（1）交互界面子系统的设计：与界面有关的类及类间结构的设计，以及有关算法的设计。

（2）交互界面子系统和应用之间接口的设计。

3．应用控制的设计

这部分对象主要完成应用的驱动工作。这部分对象不同于从现实世界中抽象出来的对象，在现实世界和问题域中没有原型，它们同界面子系统中的对象及问题对象发生作用，控制系统的运行。

4．与问题领域有关的设计

一些系统具有与应用领域有关的特点，如多任务、分布式计算等，该项工作主要是针对这些特点完成相应设计的。

4.5.2　Booch 方法

动态逻辑模型描述对象之间的互相作用。互相作用通过一组协同的对象，对象之间消息有序的序列，参与对象的可见性定义，定义系统运行时的行为。Booch 方法中的对象交互作用图被用来描述重要的互相作用，显示参与的对象和对象之间按时间排序的消息。可见性图用来描述互相作用中对象的可见性。对象的可见性定义了一个对象如何处于向它发送消息的方法的作用域之中。例如，它可以是方法的参数、局部变量、新的对象或当前执行方法的对象的部分。静态物理模型通过模块描述代码的布局。动态物理模型描述软件的进程和线程体系结构。

Booch 方法的过程包括以下步骤：

（1）在给定的抽象层次上识别类和对象；

（2）识别这些对象和类的语义；

（3）识别这些类和对象之间的关系；

（4）实现类和对象。

这四种活动不仅仅是一个简单的步骤序列，而是对系统的逻辑和物理视图不断细化的迭代和渐增的开发过程。

类和对象的识别包括找出问题空间中关键的抽象和产生动态行为的重要机制。开发人员可以通过研究问题域的术语发现关键的抽象。语义的识别主要是建立前一阶段识别出的类和对象的含义。开发人员确定类的行为（即方法）和类及对象之间的互相作用（即行为的规范描述）。该阶段利用状态转移图描述对象的状态的模型，利用时态图（系统中的时态约束）和对象图（对象之间的互相作用）描述行为模型。

在关系识别阶段描述静态和动态关系模型。这些关系包括使用、实例化、继承、关联和聚集等。类和对象之间的可见性也在此时确定。

在类和对象的实现阶段要考虑如何用选定的编程语言实现，如何将类和对象组织成模块。

在面向对象的设计方法中，Booch 强调基于类和对象的系统逻辑视图与基于模块和进程的系统物理视图之间的区别。他还区别了系统的静态和动态模型。然而，他的方法偏向于系统的静态描述，对动态描述支持较少。

Booch 方法的力量在于其丰富的符号体系，包括：

（1）类图（类结构-静态视图）；

（2）对象图（对象结构-静态视图）；

（3）状态转移图（类结构-动态视图）；

（4）时态图（对象结构-动态视图）；

（5）模块图（模块体系结构）；

（6）进程图（进程体系结构）。

用于类和对象建模的符号体系使用注释和不同的图符（如不同的箭头）表达详细的信息。Booch 建议在设计的初期可以用符号体系的一个子集，随后不断添加细节。对每一个符号体系还有一个文本的形式，由每一个主要结构的描述模板组成。符号体系由大量的图符定义，但是，其语法和语义并没有严格的定义。

4.5.3　OMT 方法

Rumbaugh 的 OMT 方法从三个视角描述系统，相应地提供了三种模型，对象模型，动态模型和功能模型。对象模型描述对象的静态结构和它们之间的关系。主要的概念包括：

（1）类；
（2）属性；
（3）操作；
（4）继承；
（5）关联（即关系）；
（6）聚集。

动态模型描述系统那些随时间变化的方面，其主要概念有：
（1）状态；
（2）子状态和超状态；
（3）事件；
（4）行为；
（5）活动。

功能模型描述系统内部数据值的转换，其主要概念有：
（1）加工；
（2）数据存储；
（3）数据流；
（4）控制流；
（5）角色（源/潭）。

OMT 方法将开发过程分为 4 个阶段：

1．分析

基于问题和用户需求的描述，建立现实世界的模型。分析阶段的产物有：
（1）问题描述；
（2）对象模型=对象图+数据词典；
（3）动态模型=状态图+全局事件流图；
（4）功能模型=数据流图+约束。

2．系统设计

结合问题域的知识和目标系统的体系结构（求解域），将目标系统分解为子系统。

3．对象设计

基于分析模型和求解域中的体系结构等添加的实现细节，完成系统设计。主要产物包括：
（1）细化的对象模型；
（2）细化的动态模型；
（3）细化的功能模型。

4. 实现

将设计转换为特定的编程语言或硬件，同时保持可追踪性、灵活性和可扩展性。

4.5.4 Jacobson 方法

Jacobson 方法与上述 3 种方法有所不同，它涉及整个软件生命周期，包括需求分析、设计、实现和测试等四个阶段。需求分析和设计密切相关。需求分析阶段的活动包括定义潜在的角色（角色指使用系统的人和与系统互相作用的软、硬件环境），识别问题域中的对象和关系，基于需求规范说明和角色的需要发现用例，详细描述用例。设计阶段包括两个主要活动，从需求分析模型中发现设计对象，以及针对实现环境调整设计模型。第一个活动包括从用例的描述发现设计对象，并描述对象的属性、行为和关联。在这里还要把用例的行为分派给对象。

在需求分析阶段的识别领域对象和关系的活动中，开发人员识别类、属性和关系。关系包括继承、熟悉（关联）、组成（聚集）和通信关联。定义用例的活动和识别设计对象的活动，两个活动共同完成行为的描述。Jacobson 方法还将对象区分为语义对象（领域对象）、界面对象（如用户界面对象）和控制对象（处理界面对象和领域对象之间的控制）。

在该方法中的一个关键概念就是用例。用例是指行为相关的事务（transaction）序列，该序列将由用户在与系统对话中执行。每一个用例就是一个使用系统的方 式，当用户给定一个输入，就执行一个用例的实例并引发执行属于该用例的一个事务。基于这种系统视图，Jacobson 将用例模型与其他 5 种系统模型关联：

（1）领域对象模型。用例模型根据领域来表示；

（2）分析模型。用例模型通过分析来构造；

（3）设计模型。用例模型通过设计来具体化；

（4）实现模型。该模型依据具体化的设计来实现用例模型；

（5）测试模型。用来测试具体化的用例模型。

OOD 详细设计阶段应该注意属性、消息和服务的如下特性和问题。

（1）是类属性（所有同类对象的共同特征）还是实例属性？

（2）该属性对实例是常数性的还是参数性的？在物资类的一个实例中，编码等是常数，而仓位、储量等是参数。常数在实例构造时确定，而参数在实例构造时最多可以有默认值，在系统运行时可以被改变。

（3）服务是主动还是被动的？若业务规则决定对所有预定代码的物资必须进行质检，则自动请求质检服务就也可能是主动服务。而系统如果要求质检请求经过库房管理用户发出，则物资类的“请求质检”服务就是被动服务了。

（4）是对内还是对外服务？对内服务是对实例本身的属性的操作，对外服务是对其他实例或系统发出消息或操作其他类实例的属性。

（5）消息的分析对 OOA 的企业经营模型有特别意义。因为消息是对象间动态关系的要素，也是信息隐蔽的重要手段。在顺序系统中，消息是向其他对象或系统发出的服务请求，在并行系统中消息表述为对象之间在一次交互中所传递的信息。在 C/S 结构的系统中，客户端向服务器请求数据属于并行的线程之间的消息，而在单机数据库系统中的用户请求数据，则一般是顺序执行的线程内的消息。接受消息的对象是什么？请求的是什么服务？消息发送者是否要与接收者同步？消息是定向的还是广播的？接收消息者如何处理消息？发送者如何对待消息处理的结果？发出消息的条件是什么？消息是在线程之间还是线程内部传递？

（6）属性、服务的性质是公共的、保护的还是私有的？这些特性的设置对封装性能影响很大，应该考虑全面和一致。应用面向对象的分析、设计方法时还要求考虑类、对象之间的关系和联系。类和对象间有：整体-部分，一般-特殊关系，以及实例连接、消息连接等联系。

多元连接使系统分析和设计复杂。可以采用增加类的方法解决。例如“出版计划-编辑-图书”这样的三元关联的语意为某编辑在某出版计划中负责某图书，增加图书项目组成员类。

扩充的 OOA 模型包括用例图和交互图，它们是在类图基本完成的基础上建立的。对企业信息系统应用而言，用例就是一个用户或一个活动者与系统进行交互的一个过程的描述，是一个用类似自然语言的语言或伪代码表述的语序，近似于功能模块描述，包含与系统责任有关的企业经营规则描述的信息化版本。用例所指的一个交互过程应该是原子过程。而活动者就是指系统外与系统交互作用的事物，活动者可能不是类图中的类、对象。在出版社系统中，“编辑”作为人员不是待实现的对象，而是要和系统进行交互的一种事物。总之，活动者是要与系统交互的事物，而其本身不是系统成分，即使可能有与活动者同名的类在类图中。

附于类图的一个重要文档是详细说明，它包含了属性、服务、消息、联系等成分的准确语意。