第 1 章

UML 概述

UML（Unified Modeling Language，统一建模语言）的目标是以面向对象图形的方式来描述任何类型的系统。它通常用于建立软件系统的模型，但也同样可以用于描述非软件领域的系统，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，可以对任何具有静态结构和动态行为的系统进行建模。

- 了解 UML 的发展历程和特点。
- 理解什么是 UML。
- 掌握 UML 中的 14 种图以及系统开发的 5 个阶段。

1.1 什么是 UML

UML 是一种能够描述问题、描述解决方案、起到沟通作用的语言。通俗地说，它是一种用文本、图形和符号的集合来描述现实生活中各类事物、活动及其之间关系的语言。

UML 是一种很好的工具，可以贯穿软件开发的整个周期，适用于数据建模、业务建模、对象建模和组件建模。UML 作为一种模型语言，它使开发人员专注于建立产品的模型和结构，而不是选用什么程序语言和算法实现。当模型建立之后，可以被 UML 工具转换成指定的程序语言代码。

1.2 UML 的发展历程

UML 起源于多种面向对象（Object Oriented，OO）建模方法，由 OMG 开发，目前已经成为工业标准。面向对象建模语言最早出现于 20 世纪 70 年代中期。在 1989—1994 年，其数量从不到 10 种增加到了 50 多种。各种语言的创造者努力推广自己的产品，并在实践中不断完善。但是，面向对象方法的用户并不了解不同建模语言的优缺点及相互之间的差异，因而很难根据应用特点选择合适的建模语言，于是爆发了一场“方法大战”。20 世纪 90 年代中期，出现了一批新方法，其中最引人注目的是 Booch 1993、OMT-2 和 OOSE 等。

面向对象软件工程的概念最早是由 Booch 提出的，他是面向对象方法最早的倡导者之一。后来，Rumbaugh 等人提出了面向对象的建模技术（OMT）方法，采用了面向对象的概念，并引入各种独立于语言的表示符。这种方法使用对象模型、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，软件开发人员在开发过程的不同阶段不需要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 OOSE 方法，其最大特点是面向用例（Use Case），并在用例的描述中引入了外部角色的概念。用例是精确描述需求的重要武器，但用例贯穿于整个开发过程，包括对系统的测试和验证。OOSE 比较适合支持商业工程和需求分析。

此外，还有 Coad/Yourdon 方法，即著名的面向对象分析（OOA）/面向对象设计（OOD），它是最早的面向对象（OO）的分析和设计方法之一。该方法简单、易学，适合面向对象技术的初学者使用，但由于该方法在处理能力方面具有局限性，因此用得很少。

总结起来，首先，面对众多的建模语言，用户没有能力区别不同语言之间的差别，因此很难找到一种比较适合其应用特点的语言；其次，众多的建模语言实际上各有特色；最后，虽然不同的建模语言大多类同，但仍存在某些细微的差别，极大地妨碍了用户之间的交流。因此，需要统一建模语言。

1994 年 10 月，Grady Booch 和 Jim Rumbaugh 开始致力于这一工作。他们首先将 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，称之为统一方法 UM 0.8（Unitied Method）。1995 年秋，OOSE 的创始人 Jacobson 加入这一工作。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，即 UML 0.9 和 UML 0.91，并将 UM 重新命名为 UML。

1996 年，UML 的开发者成立了 UML 成员协会，以完善、加强和促进 UML 的定义工作。当时的成员有 DEC、HP、I-Logix、Itellicorp、IBM、ICON Computing、MCI Systemhouse、Microsoft、Oracle、Rational Software、TI 以及 Unisys。这一机构对 UML 1.0（1997 年 1 月）及 UML 1.1（1997 年 11 月 17 日）的定义和发布起到了重要的促进作用。

在美国，截至 1996 年 10 月，UML 获得了工业界、科技界和应用界的广泛支持，有 700 多个公司表示支持采用 UML 作为建模语言。1996 年年底，UML 已稳占面向对象技术市场的 85%，成为可视化建模语言事实上的工业标准。1997 年 11 月 17 日，OMG 采纳 UML 1.1 作为基于面向对象技术的标准建模语言。UML 代表了面向对象方法的软件开发技术的发展方向，具有巨大的市场前景，也具有重大的经济价值和国防价值。此后不断地对 UML 进行修订，并

产生了 UML 1.2、1.3 和 1.4 版本。2000 年，UML 1.4 在语义上添加了动作语义的定义，使得 UML 规格说明在计算上更加完整。2005 年，UML 2.0 规范形成，定义了许多可视化语法，特别是元模型的定义，至此，融合早期最好思想的 UML 已经呈现在我们面前。到 2011 年 8 月，正式发布的版本已经是 UML 2.4.1 了。2015 年 6 月，UML 2.5 版本正式发布。

1.3 UML 的特点

标准建模语言 UML 的主要特点可以归结为以下 3 点：

- UML 统一了 Booch、OMT 和 OOSE 等方法中的基本概念和符号。
- UML 吸取了面向对象领域中各种优秀的思想，其中也包括非面向对象方法的影响。UML 符号表示考虑了各种方法的图形表示，删掉了很多容易引起混乱的、多余的和极少使用的符号，同时添加了一些新符号。因此，在 UML 中凝聚了面向对象领域中很多人的思想。这些思想并不是 UML 的开发者们发明的，而是开发者们依据最优秀的面向对象方法和丰富的计算机科学实践经验综合提炼而成的。
- UML 在演变过程中还提出了一些新的概念。在 UML 标准中新添加了模板（Stereotypes）、职责（Responsibilities）、扩展机制（Extensibility Mechanisms）、线程（Threads）、过程（Processes）、分布式（Distribution）、并发（Concurrency）、模式（Patterns）、合作（Collaborations）、活动图（Activity Diagram）等概念，并清晰地区分类型（Type）、类（Class）、实例（Instance）、细化（Refinement）、接口（Interfaces）和组件（Components）等概念。

因此，可以认为UML是一种先进实用的标准建模语言，但其中某些概念尚待实践来验证，UML 也必然存在一个进化过程。

1.4 UML 的组成

UML 主要由事物、图和关系组成。事物是 UML 的核心元素，关系把元素紧密联系在一起，图是很多有相互关系的事物构成的集合。

1.4.1 UML 中的事物

UML 包含 4 种事物：构件事物、行为事物、分组事物和注释事物。

1）构件事物

构件事物是 UML 模型的静态部分、描述概念或物理元素，它包括以下几种。

（1）类：类是对一组具有相同属性、相同操作、相同关系和相同语义的对象的抽象。在 UML 中，类用一个矩形表示，它包含 3 个区域，最上面是类名，中间是类的属性，最下面是类的方法。

（2）接口：接口是指为类或组件提供特定服务的一组操作的集合，因此，一个接口描述了类或组件的对外可见的动作。一个接口可以实现类或组件的全部动作，也可以只实现一部分。在 UML 中，接口用一个圆和它的名字表示。

（3）协作：协作描述了一组事物间的相互作用的集合。

（4）用例：用例用于描述一系列的动作，这些动作是系统对一个特定角色执行的。在模型中，用例是通过协作来实现的。在 UML 中，用例用一个实线椭圆和它的名字表示。

（5）构件：构件也称为“组件”，是物理上可替换的系统部分，它实现了一个接口集合。在一个系统中，可以使用不同种类的组件，例如 COM+ 或 Java Beans。

（6）节点：为了能够有效地对部署的结构进行建模，UML 引入了节点这一概念，它可以用来描述实际的 PC、打印机、服务器等软件运行的基础硬件。节点是运行时存在的物理元素，代表了一种可计算的资源，通常其有存储空间和处理能力。

此外，参与者、文档库、页表等都是上述构件事物的变体。

2）行为事物

行为事物是UML模型图的动态部分，描述跨越空间和时间的行为，主要包括以下两部分。

（1）交互：交互是指实现某功能的一组构件事物之间的消息的集合，涉及消息、动作序列、链接。

（2）状态机：状态机用于描述事物或交互在生命周期内响应事件所经历的状态序列。

3）分组事物

分组事物是 UML 模型图的组织部分，用于描述事物的组织结构，主要由包来实现。

包：把元素编成组的机制。

4）注释事物

注释事物是 UML 模型的解释部分，用来对模型中的元素进行说明、解释。

注解：对元素进行约束或解释的简单符号。

1.4.2 UML 中的关系

在 UML 中有 4 种关系：依赖（Dependency）、关联（Association）、泛化（Generalization）和实现（Realization）。

1）依赖

依赖是两个模型元素间的语义关系，其中一个元素（独立元素）发生变化会影响另一个元素（依赖元素）的语义。在图形上，把依赖画成一条可能有方向的虚线，偶尔在其上还带有一个标记，如图 1.1 所示。

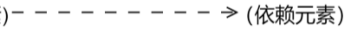

图 1.1 依赖

2）关联

关联指明了一个对象与另一个对象间的关系。在图形上，关联用一条实线表示，它可能有方向，偶尔在其上还有一个标记。例如，读者可以去图书馆借书和还书，图书管理员可以管理图书，也可以管理读者的信息，显然在读者、图书、管理员之间存在着某种联系。那么在用 UML 设计类图的时候，就可以在读者、图书、管理员三个类之间建立关联关系，如图 1.2 所示。

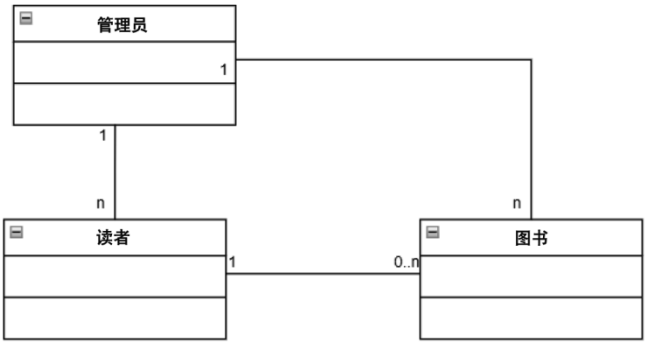

图 1.2　关联

3）泛化

泛化是一种一般化—特殊化的关系，是一般事物（父类）和该事物较为特殊的种类（子类）之间的关系，子类继承父类的属性和操作。除此之外，子类还能添加新的属性和操作。在图形上，把泛化关系画成带有空心箭头的实线，箭头指向父类，如图 1.3 所示。

图 1.3　泛化

4）实现

实现是类之间的语义关系，其中的一个类指定了另一个类必须执行的约定。在两种地方会遇到实现关系：一种是在接口和实现它们的类或构件之间，另一种是在用例和实现它们的协作之间。在图形上，把实现关系画成一条带有空心箭头的虚线，它是泛化和依赖关系两种图形的结合，如图 1.4 所示。

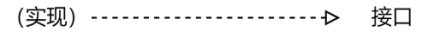

图 1.4　实现

1.4.3　UML 2.5 的图

UML 中的图是描述 UML 视图内容的图形。对比 UML 1.x，UML 2.0 增加了包图、组合结构图、交互概览图和定时图，UML 2.2 增加了概要图，一共有 14 种不同的图，通过它们的相互组合可以为被建模系统提供所有视图。UML 2.0 在可视化建模方面进行了许多改革和创新，可以描述现今软件系统中存在的许多技术，例如模型驱动架构（MDA）和面向服务的架构（SOA）。UML 2.5 中的 14 种图一一介绍如下。

1. 用例图

用例图从用户角度描述系统功能，并指出各功能的操作者。用例图是 UML 中最简单也是最复杂的一种图。说它简单是因为它采用了面向对象的思想，基于用户角度来描述系统，绘制非常容易，图形表示直观并且容易理解。说它复杂是因为用例图往往不容易控制，要么过于复杂，要么过于简单。用例图展示了一组用例、参与者以及它们之间的关系，如图 1.5 所示。

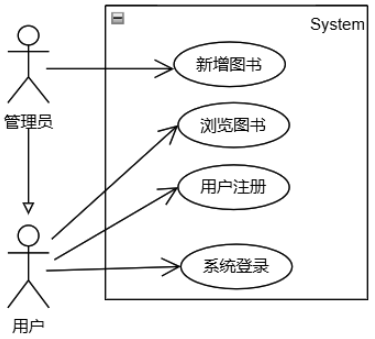

图 1.5　用例图

用例图中的主体内容（用例、参与者、关联）并没有变化。在 UML 1.x 中，只能用用例图所归属的包来表达一组用例的逻辑组织关系，即用用例在模型中所处的物理位置表达逻辑组织关系。在 UML 2.0 中，为每个用例增加了一个称为“Subject”的特征，这项特征的取值可以作为在逻辑层面划分一组用例的一项依据。用例所属的“系统边界”就是“Subject”的一种典型例子。

2. 类图

类图是 UML 面向对象中最常用的一种图，可以帮助我们更直观地了解一个系统的体系结构。通过关系和类表示的类图，可以图形化地描述一个系统的设计部分，如图 1.6 所示。

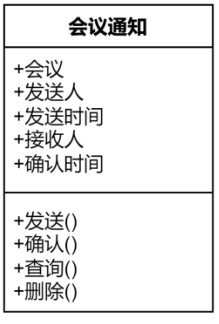

图 1.6　类图

3. 对象图

在面向对象中，对象图是类图的实例，使用与类图几乎完全相同的标识。它们的不同点在于对象图显示类的多个对象实例，而不是实例的类。由于对象存在生命周期，因此对象图只能在系统的某一时间段存在。

4. 状态机图

状态机图用于描述一个实体基于事件响应的动态行为，显示了该实体如何根据当前所处的状态对不同的事件做出响应，如图 1.7 所示。

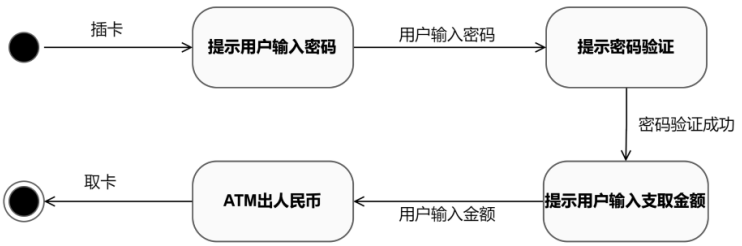

图 1.7　状态机图

5. 活动图

在面向对象中，活动图记录了单个操作、方法的逻辑，或者单个业务流程的逻辑，描述系统中各种活动的执行顺序，通常用于描述一个操作中所要进行的各项活动的执行流程。同时，它也常被用来描述一个用例的处理流程，或者某种交互流程。

活动图由一些活动组成，图中同时包括了对这些活动的说明。当一个活动执行完毕之后，将沿着控制转移箭头转向下一个活动。在活动图中还可以很方便地描述控制转移的条件以及并行执行等要求，如图 1.8 所示。

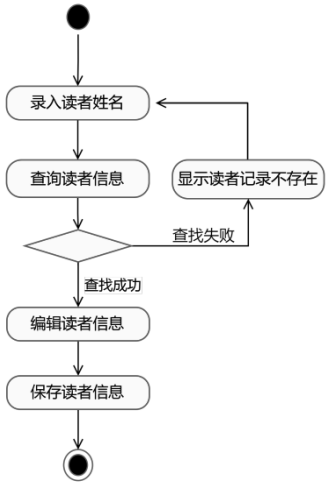

图 1.8　活动图

活动图是一种比较常用的图，接近于流程图。在 UML 2.0 中，活动图增加了许多新特性，例如，泳道可以划分成层次、增加丰富的同步表达能力、在活动图中引入对象等特性。

6. 顺序图

顺序图描述了对象之间动态的交互关系，主要体现对象之间进行消息传递的时间顺序。

顺序图由一组对象构成，每个对象分别带有一条竖线，称作对象的生命线，它代表时间轴，

时间沿竖线向下延伸。顺序图描述了这些对象随着时间的推移相互之间交换消息的过程。消息用从一个对象的生命线指向另一个对象的生命线的水平箭头表示。图中还可以根据需要增加有关时间的说明和其他注释，如图 1.9 所示。

对于顺序图，UML 2.0 主要做了如下改进：

- 允许顺序图中明确表达分支判断逻辑，这样能够将以前要通过两幅图才能表达的意思通过一幅图就表达了，但这并不意味着顺序图擅长表达这种逻辑，因此并不需要在顺序图中展现所有的分支判断逻辑。
- 允许“纵向”与“横向”地对顺序图进行拆分与引用，这样就解决了以前一幅图由于流程过多而造成的幅面过大、浏览不方便的困难。

7. 通信图

在面向对象中，通信图显示了交互中各个对象之间的组织交互关系以及对象彼此之间的链接。与顺序图不同，通信图显示的是对象之间的关系，而且它没有将时间作为一个单独的维度，因此序列号就决定了消息及并发线程的顺序。通信图用带有编号的箭头来描述特定的方案，以显示在整个方案过程中消息的移动情况，而非交互的顺序，如图 1.10 所示。

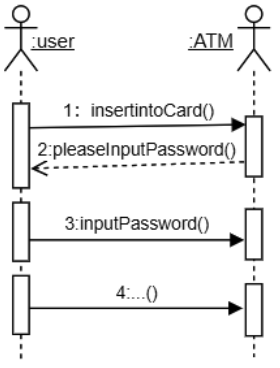

图 1.9　顺序图

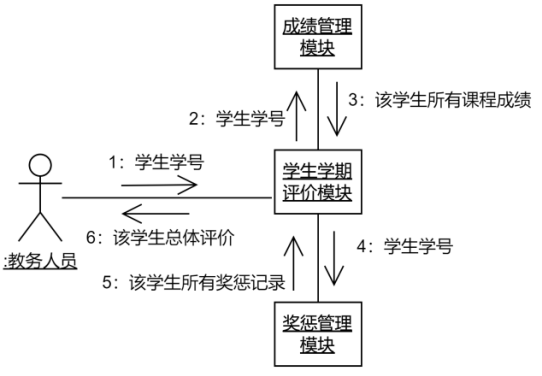

图 1.10　通信图

8. 组件图

组件图也称为构件图，用于描述代码部件的物理结构及各部件之间的依赖关系。组件图有助于我们分析和理解部件之间的相互影响程度。从组件图中，可以了解各软件组件（如源代码文件或动态链接库）之间的编译器和运行时依赖关系。使用组件图可以将系统划分为内聚组件，并显示代码自身的结构，如图 1.11 所示。

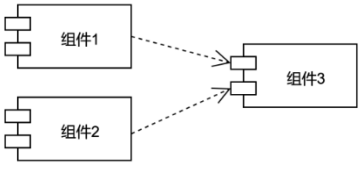

图 1.11　组件图

组件图是在物理层面对系统结构及内容的直观描述，最接近于通常意义上的模块结构图。

在 UML 2.0 中，组件图得到了显著改进，组件本身的内容描述更为清晰，组件所提供的接口、所要求的接口以及组件之间的依赖关系可以通过“组装连接器”（Assembling Connector）进行更加明确的表达。

9. 部署图

部署图也称为配置图，用于描述系统中硬件和软件的物理配置情况和系统体系结构。

在部署图中，用节点表示实际的物理设备，如计算机和各种外部设备等，并根据它们之间的连接关系，将相应的节点连接起来，并说明其连接方式。在节点里面，说明分配给该节点运行的可执行构件或对象，从而说明哪些软件单元被分配在哪些节点上运行，如图 1.12 所示。

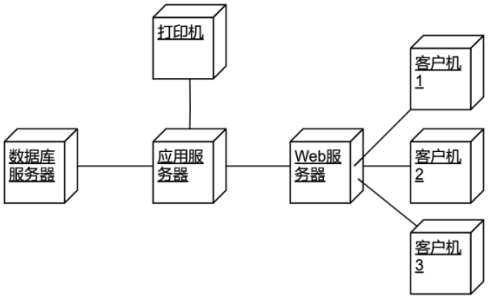

图 1.12　部署图

10. 包图

包图展现模型要素的基本组织单元，以及这些组织单元之间的依赖关系，包括引用关系和扩展关系。在通用的建模工具中，一般可以用类图描述包图中的逻辑内容，如图 1.13 所示。

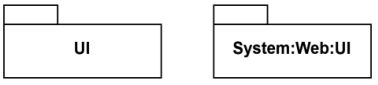

图 1.13　包图

11. 组合结构图

组合结构图用于描述系统中的某一部分（即组合结构）的内部内容，包括该部分与系统其他部分的交互点。这种图能够展示该部分内容内部参与者的配置情况。

组合结构图中引入了一些重要的概念，例如：端口（Port），它将组合结构与外部环境隔离，实现了双向的封装，既涵盖了该组合结构所提供的行为，同时也指出了该组合结构所需要的服务；协议（Protocol），它基于 UML 中的协作（Collaboration）的概念，展示那些可复用的交互序列，其根本目的是描述那些可以在不同上下文环境中复用的协作模式。协议中所反映的任务由具体的端口承担，如图 1.14 所示。

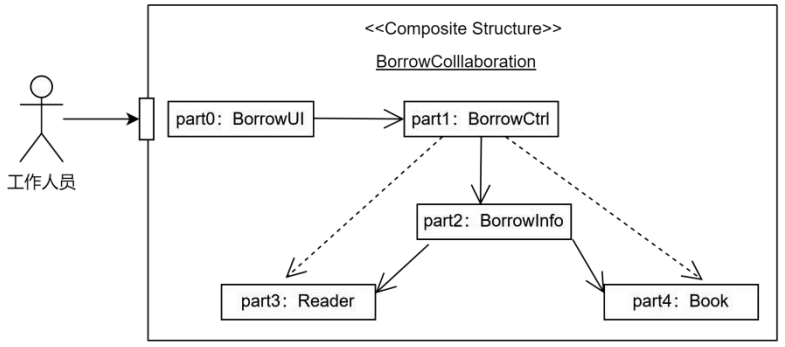

图 1.14　组合结构图

12. 定时图

定时图是一种可选的交互图，展示交互过程中的真实时间信息，具体描述对象状态变化的时间点以及维持特定状态的时间段。定时图是 UML 2.0 新增的建模图，实质上，它是一种特殊的顺序图。对一些实时性较强的系统功能进行建模时，可以选用定时图。

定时图是一个二维图，在纵轴方向，处在不同位置的水平线表示对象的不同状态；横轴用来表示时间，时间由左向右延伸。定时图包含的基本元素有对象、状态、表示状态的水平线、表示状态迁移的垂直线、表示时间的横轴和时间刻度，如图 1.15 所示。

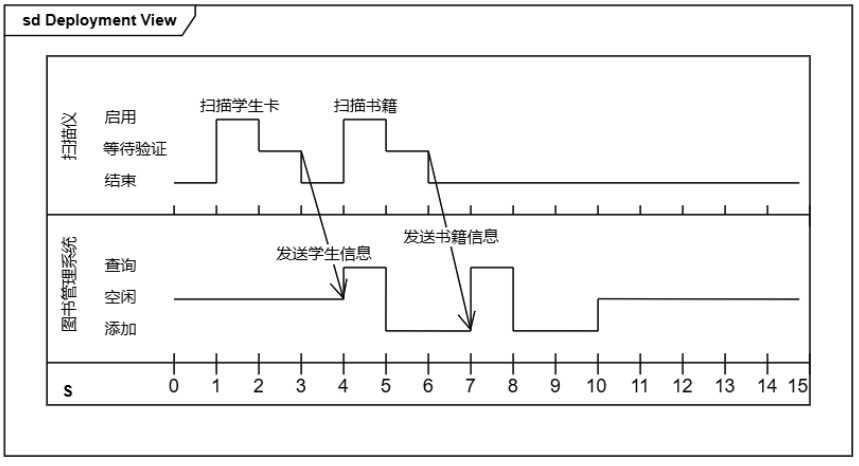

图 1.15　定时图

13. 交互概览图

交互概览图（InteractionOverviewDiagram）与活动图类似，只是将活动图中的动作元素改为交互概览图的交互关系。如果概览图内的一个交互涉及时序，则使用顺序图；如果概览图中的另一个交互需要关注消息次序，则可以使用通信图。交互概览图将系统内单独的交互结合起来，并针对每个特定交互使用最合理的表示法，以显示出它们是如何协同工作来实现系统的主要功能的，如图 1.16 所示。

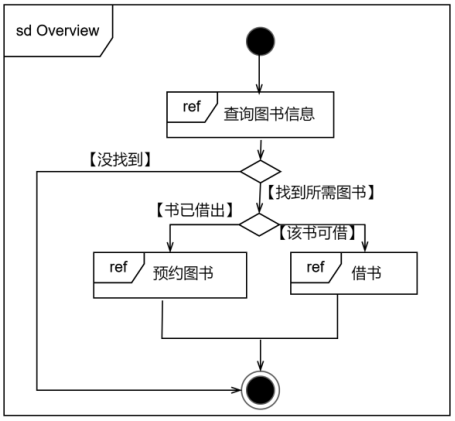

图 1.16　交互概览图

顺序图、通信图和定时图主要关注特定交互的具体细节，而交互概览图则将各种不同的交互结合在一起，形成针对系统某种特定要点的交互整体图。

14. 概要图

概要图（Profile Diagram），又称外廓图，或者剖面图，是一种结构图，它通过定义自定义原型、标记值和约束来描述 UML 的轻量级扩展机制，用来辅助其他 UML 图。概要图不是一种传统的图表类型，而是一种用于为 UML 图表创建新语义的机制。创作者可以使用此功能来标记值和关键字，添加条件和约束，或者设计全新的 UML 元素，而不仅仅是 UML 图表绘制工具中通常可用的元素。

概要图中主要包含的元素有概要（profile）、元类（metaclass）、构造型（stereotype）等，元素之间的关系有引用（reference）、应用（application），还有 UML 中常用的泛化等关系。

概要图的基本示例如图 1.17 所示。

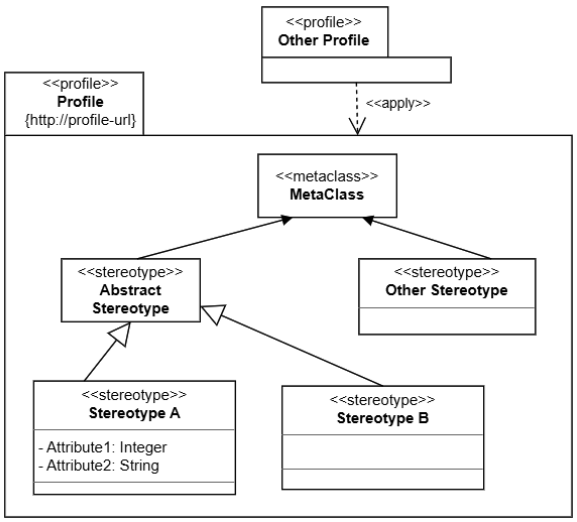

图 1.17　概要图

上述 14 种图可归纳为 5 类，如表 1.1 所示。

表 1.1　UML 图分类

类　　型	包　　含
静态图	类图、对象图、包图、组合结构图、概要图
行为图	状态机图、活动图
用例图	用例图
交互图	顺序图、通信图、定时图、交互概览图
实现图	组件图、部署图

从应用的角度看，当采用面向对象技术设计系统时，第一步是描述需求；第二步根据需求建立系统的静态模型，以构造系统的结构；第三步是描述系统的行为。其中，在第一步与第二步中所建立的模型都是静态的，包括用例图、类图、包图、对象图、组合结构图、组件图、部署图和概要图 8 个图形，是标准建模语言 UML 的静态建模机制。第三步所建立的模型或者可以执行，或者表示执行时的时序状态或交互关系，它包括状态机图、活动图、顺序图、通信图、定时图和交互概览图 6 个图形，是标准建模语言 UML 的动态建模机制。因此，标准建模语言 UML 的主要内容也可以归纳为静态建模机制和动态建模机制两大类。

1.5　UML 2.5 图的分类

UML 2.5 在 UML 2.4.1 的基础上进行了结构性的调整，简化和重新组织了 UML 规范文档。UML 规范被重新编写，使其“更易于阅读”，并且“尽可能减少前向引用”。

UML 2.5 发展了 UML 2.0 规范，并迅速成为建立软件系统可视化、规范、文档的标准。统一建模语言（UML）也被用于非软件系统的建模，并在金融、军事、工程等领域有着广泛应用。

在 UML 2.5.1 规范中，UML 图的类型有 14 种，比 UML 2.0 多了一种概要图，它们被分成两大类：结构建模图（Structure Diagram）和行为建模图（Behavior Diagram）。

1.5.1　结构建模图

结构建模图显示了系统及其各个部分在不同抽象层和实现层上的静态结构，以及它们如何相互关联。它们通常用来对那些构成模型的“要素”进行建模，比如类、对象、接口和物理组件。另外，它们也用来对元素间关联和依赖关系进行建模。

结构建模图分类如表 1.2 所示。

表 1.2　结构建模图分类

图　　名	功　　能	主要名词
包图（Package Diagram）	用来将模型划分成不同的逻辑容器或包，并在更高层次上描述它们之间的交互关系	包（Package），可封装的元素（Packageable Element），依赖（Dependency），元素导入（Element Import），包导入（Package Import），包合并（Package Merge）
类图（Class Diagram）	显示模型的静态结构，特别是模型中存在的类、类的内部结构以及它们与其他类的关系等	类（Class），接口（Interface），特性（Feature），约束（Constraint），关联（Association），泛化（Generalization），依赖（Dependency）
对象图（Object Diagram）	显示结构元素的实例间如何关联，以及在运行时如何使用	实例规范（Instance Specification），对象（Object），属性（Property），关联关系（Association）
组合结构图（Composite Structure Diagram）	提供了一种对元素结构进行分层的方法，并着重体现了元素内部的细节、结构、关系和行为	端口所需的接口（Required Interface） 委托连接器（Connector Delegate）
组件图（Component Diagram）	又称为构件图，用来构造更高层次或更复杂的结构，通常由一个或多个类构成，并提供一个定义明确的接口	组件（Component），接口（Interface），提供的接口（Provided Interface），所需的接口（Required Interface），类（Class），端口（Port），连接器（Connector），工件（Artifact），组件实现（Component Realization），作用关系（Usage）
部署图（Deployment Diagram）	显示现实环境中重要物件的物理配置	部署（Deployment），工件（Artifact），部署目标（Deployment Target），节点（Node），设备（Device），执行环境（Execution Environment），通信路径（Communication Path），部署规范（Deployment Specification）
概要图（Profile Diagram）	作为 UML 标准的轻量级扩展机制的辅助图，它容许定义定制的原型、标记值和约束。概要文件容许对不一样的 UML 元模型进行调整	概要（Profile），概要类（Profile Metaclass），模板（Stereotype），概要扩展（Profile Extension），概要参考（Profile Reference），概要应用程序（Profile Application）

结构建模图并没有使用时间相关的概念，也没有显示动态行为的细节，但是它们可能会显示与结构图中展现的分类器行为的关系。

1.5.2　行为建模图

行为建模图显示了系统中对象的动态行为，可以将其描述为随着时间的推移对系统进行的一系列更改，并跟踪系统在真实环境下如何表现，以及观察系统对一个操作或事件的响应及其造成的结果。

行为建模图分类如表 1.3 所示。

表 1.3　行为建模图分类

图　　名	功　　能	主要名词
用例图（UseCase Diagram）	用来对用户/系统的交互关系建模。 用脚本和情形的形式来定义行为、要求和约束。UML 2.4.1 规范指出，用例图是类图的一个特例	用例（Use Case），参与者（Actor），主体（Subject），扩展（Extend），包含（Include），关联（Association）
活动图（Activity Diagram）	广泛用于定义基本程序的流程和一般化过程中，记录判断点和动作	活动（Activity），分区（Partition），行动（Action），对象（Object），控制（Control），活动传递（Activity Edge）
状态机图（State Machine Diagram）	对于了解模型执行时的瞬时状态，即模型的运行状态是重要的	行为状态（Behavioral State），行为转换（Behavioral Transition），伪状态（Pseudostate）
通信图（Communication Diagram）	显示协作实例中，对象间实时消息和通信的网络结构与顺序	生命线（Lifeline），消息（Message）
顺序图（Sequence Diagram）	与通信图联系紧密，并在垂直时间线上显示对象间消息传递的顺序	生命线（Lifeline），执行申明（Execution Specification），消息（Message），组合片段（Combined Fragment），交互使用（Interaction Use），状态不变式（State Invariant），销毁（Time Limit）
定时图（Timing Diagram）	融合顺序图和状态图，以提供观察对象随时间变化的状态和改变这个状态的消息	生命线（Lifeline），状态或情况时间表（State or Condition Timeline），销毁事件（Destruction Event），持续约束（Duration Constraint），时间限制（Time Limit）
交互概览图（Interaction Overview Diagram）	融合活动图和顺序图，使交互部分容易与判断点和流程结合	初始节点（Initial Node），流最终节点（Flow Final Node），活动最终节点（Activity Final Node），决策节点（Decision Node），合并节点（Merge Node），分叉节点（Fork Node），链接节点（Join Node），交互（Interaction），交互使用（Interaction Use），持续约束（Duration Constraint），时间限制（Time Limit）

1.6　系统开发阶段

系统开发共有 5 个阶段：需求分析、系统分析、系统设计、程序实现和测试阶段。

软件过程对于组织的重要性，就如同算法对子程序运行一般。合适的算法可以提高运行的效率，不合适的算法则不仅无法提高效率，还会浪费组织资源的使用率。软件开发过程涉及的

是更为复杂的人、事、物，而算法则是纯粹的机器代码执行。

软件开发过程主要描述开发软件系统所涉及的相关活动，以及如何循序渐进地执行这些活动。不同的系统、组织及开发，其管理工具所采用的流程都有可能不同。例如，有些系统适合采用按部就班的方式，从分析、设计、实现、测试到移交逐步地进行；有些系统则适合采用反复循环的方式，不断地重复执行分析、设计、实现、测试等活动。

需求分析的主要内容是了解客户的需求、分析系统的可行性、分析需求的一致性及正确性等。

系统分析是根据在需求分析阶段建立的用户需求模型，进一步对系统进行分析，将用户需求转换为系统功能。在实际开发中，一般不存在系统分析这一个阶段，在需求分析阶段完成所有的分析。

系统设计是将需求转换为系统的重要过程，包含架构设计、模块间的接口设计、数据库设计、算法设计与数据结构设计等。许多软件工程师常会忽略规划的重要性，认为自己可以立即编写程序而不需要分析需求和撰写设计。此种做法对于软件系统而言，可能会造成种种问题。例如，如果没有架构设计，就会缺乏整体性的思考，系统可能因此而无法满足接口需求以及非功能性的需求（如性能、可维护性等）；还可能会因为忽略事先的规划与分析而造成重复工作；等等。

程序实现指的是通过程序语言，将所设计的内容转化为可以执行的软件系统。除错是实现活动中不可避免的工作，主要是修改程序编写过程中产生的错误。除此之外，单元测试通常也会在实现阶段进行，目的是要确认单元程序代码的正确性。当程序有错误时，需要进行除错，将错误排除。

测试是对实现的程序代码模块进行检测，检验其功能是否正确、性能是否符合要求。一般而言，测试可以分为单元测试、集成测试、系统测试与验收测试。

- 单元测试：测试单元模块功能是否能正常运行。
- 集成测试：测试模块或子系统的接口集成是否能正常运行。
- 系统测试：测试系统的整体性能、安全性、稳定度等非功能性需求是否符合预期目标。
- 验收测试：测试系统的整体性能是否符合使用者的要求。

软件系统的特性之一就是需求会经常发生变动，许多系统每隔半年甚至两三个月就会改版。软件维护的目的是要确保已经发行的软件系统可以持续满足客户的需要。一般而言，维护可以有如下几种情况：修复错误、增加或变更功能，以及因为平台改变所做的调整。

1.7 小　　结

UML 是一种遵循特定规则的建模语言，它允许创建多种模型，但不指定设计者应创建哪些模型，也不提供具体的开发流程。作为一种可视化的图形语言，UML 用于系统构建或理解系统结构。

UML 的组成共包括 3 部分：事物、图和关系。事物是 UML 的核心元素，关系把事物紧

密联系在一起，图是很多具有相互关系的事物构成的集合。

UML 中的事物主要有类、接口、用例、组件、节点、消息、连接、状态、事件、活动等。UML 2.5 中的图有 14 种，通过它们的相互组合可以为被建模系统提供所有视图。这 14 种图分为两大类：结构建模图，包括包图、类图、对象图、组合结构图、组件图、部署图和概要图；行为建模图，包括用例图、活动图、状态机要图、通信图、顺序图、定时图、交互概览图。

1.8 习　　题

1．UML 的事物有哪些？
2．UML 的关系有哪些？
3．UML 的图有哪些？其中哪些是静态图？哪些是动态图？
4．为什么要学习 UML?
5．简述什么是 UML?
6．在 Internet 上查询 UML 图的知识，写出你自己关于 UML 的认识和体会。