高等教育规划教材

UML 基础与建模实践教程

王先国　主编

机 械 工 业 出 版 社

本书是一本关于 UML 建模的实践教程，以大量案例为基础，重点介绍了 UML 体系结构、UML 元素语义、语法、UML 建模方法和 RUP 统一过程。全书分为 3 篇，共 17 章。第 1 篇（第 1 ~ 13 章）为 UML 语言基础，内容包括：UML 语言体系结构、UML 组成元素、UML 图的表示方法、UML 图的作用；第 2 篇（第 14 ~ 15 章）为 UML 高级技术，内容包括：Rose 双向工程、RUP 统一软件过程；第 3 篇（第 16 ~ 17 章）为 UML 建模实践，内容包括：网上书店建模和气象站数据建模，本篇重点介绍了领域建模与分析过程、用例建模与分析过程、动态建模与分析过程、RUP 分析和设计过程。

本书重点突出了 UML 语言的表示方法、系统建模方法和建模过程。书中所有的概念、技术、建模方法都通过实例来演示，内容精炼，表达简明，实例丰富，非常适合作为高等院校计算机专业及相关专业的教材，也可以作为培训机构相关专业的培训教材。

本书配套授课电子课件，需要的教师可登录 www. cmpedu. com 免费注册，审核通过后下载，或联系编辑索取（QQ：2966938356，电话：010 - 88379739）。

图书在版编目(CIP)数据

UML 基础与建模实践教程/王先国主编 . —北京:机械工业出版社,2015. 3
高等教育规划教材
ISBN 978-7-111-51554-8

Ⅰ. ①U… Ⅱ. ①王… Ⅲ. ①面向对象语言 - 程序设计 - 高等学校 - 教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2015)第 214330 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)
策划编辑：郝建伟 责任编辑：郝建伟
责任校对：张艳霞
责任印制：

印刷(装订)

2016 年 1 月第 1 版 · 第 1 次印刷
184mm × 260mm · 14. 25 印张 · 349 千字
0001 - 3000 册
标准书号：ISBN 978-7-111-51554-8
定价：36. 00 元

出 版 说 明

当前，我国正处在加快转变经济发展方式、推动产业转型升级的关键时期。为经济转型升级提供高层次人才，是高等院校最重要的历史使命和战略任务之一。高等教育要培养基础性、学术型人才，但更重要的是加大力度培养多规格、多样化的应用型、复合型人才。

为顺应高等教育迅猛发展的趋势，配合高等院校的教学改革，满足高质量高校教材的迫切需求，机械工业出版社邀请了全国多所高等院校的专家、一线教师及教务部门，通过充分的调研和讨论，针对相关课程的特点，总结教学中的实践经验，组织出版了这套“高等教育规划教材”。

本套教材具有以下特点：

1）符合高等院校各专业人才的培养目标及课程体系的设置，注重培养学生的应用能力，加大案例篇幅或实训内容，强调知识、能力与素质的综合训练。

2）针对多数学生的学习特点，采用通俗易懂的方法讲解知识，逻辑性强、层次分明、叙述准确而精炼、图文并茂，使学生可以快速掌握，学以致用。

3）凝结一线骨干教师的课程改革和教学研究成果，融合先进的教学理念，在教学内容和方法上做出创新。

4）为了体现建设“立体化”精品教材的宗旨，本套教材为主干课程配备了电子教案、学习与上机指导、习题解答、源代码或源程序、教学大纲、课程设计和毕业设计指导等资源。

5）注重教材的实用性、通用性，适合各类高等院校、高等职业学校及相关院校的教学，也可作为各类培训班教材和自学用书。

欢迎教育界的专家和老师提出宝贵的意见和建议。衷心感谢广大教育工作者和读者的支持与帮助！

机械工业出版社

前　言

UML 基础与建模实践是计算机科学专业和软件工程专业学生的必修课程，也是一门非常重要的课程。尽管市面上介绍 UML 语言的图书不少，但是很少有一本书的内容在系统分析、设计过程中全面涵盖领域建模、用例建模、动态建模。也很少有教材在系统分析、设计过程中采用 RUP 统一过程开发方法。一般的教材对建模过程的分析和设计与建模方法是脱节的，因此，学生不能真正理解建模技术和建模方法，在实践中更谈不上正确地运用 UML 语言实现面向对象的分析和设计。

学生在建模实践中重点解决的问题主要集中在以下三点：第一，真正理解 UML 表示法，知道如何使用它们；第二，理解统一开发过程（RUP），知道在哪种情况下采用哪种模型来构造系统；第三，知道如何运用建模技术和建模方法。

本书在内容组织上完全针对上述的三个问题，体系结构安排合理，知识表达通俗易懂，知识讲解深入浅出，以两个完整的案例为大、中型软件系统的建模提供了开发步骤、技术提示和表示方法。具体特点如下。

1）在体系结构的安排上强调内容的系统性、连贯性、逻辑性和实用性。对 UML 元素的语义、语法和建模方法的讲解由易到难逐层展开，便于读者学习和理解。

2）对 UML 语言的讲解充分体现了文字描述和图形描述的结合。通过文字描述，详细地定义了 UML 元素的语义、语法；通过图形将 UML 元素可视化、规范化；对每个 UML 元素的讲解采用实例演示 UML 元素的语义和使用方法，使读者易于理解。

3）从框架到细节表达知识。首先对知识进行概要描述，然后分解知识、简化知识，对知识进行详细描述，这样，将复杂的建模技术、建模方法简单化，抽象问题具体化。

4）提供完整的建模实例。本书以网上书店为例，为读者提供了详细的建模过程、建模技术和建模方法。整个建模流程是可以操作的，也是可以模拟的，学生能真正做到学以致用。

5）在写作上，本书以 UML 设计元素为主线，以系统建模为目标，运用实例系统地阐明了 UML 语言基础、建模技术和建模方法。本书技术、方法和实践结合生动，知识表达通俗易懂，既可作为高等院校计算机专业及相关专业的教材，也可以作为培训机构相关专业的培训教材。

本书由王先国主编，杨仕范、蔡木生参与了本书的编写工作。本书作者在大型软件公司从事应用系统的分析和设计工作，在开发系统过程中积累了丰富的系统建模方法，能熟练地运用 UML 语言把系统需求分析和系统设计形式化为标准的需求分析文档和设计文档。如有建议或在学习中遇到疑难问题，欢迎大家给作者发送电子邮件（作者邮箱：wangxg588@sohu.com）。本书配备了教学大纲和课件，如果需要，请与出版社联系。

本书中的分析和设计案例，虽然经过了多次修改和审核，但难免会存在疏漏和错误，恳请读者批评指正。

编　者

目　　录

第1章　UML概述

UML是一种图形符号系统，是一种通用的软件设计语言，可以用该种语言对任何具有静态结构和动态行为的系统进行表示、描述、模拟、可视化和文档化。

1.1　UML简介

UML（Unified Modeling Language，统一建模语言）是用来对业务系统和软件系统进行可视化建模的一种语言。在面向对象的软件开发过程中，常采用本语言对系统的产品进行说明、可视化和文档编写。

1.1.1　UML简史

公认的面向对象建模语言出现于20世纪70年代中期。从1989年到1994年，这种设计语言的数量从不到十种增加到了五十多种。

20世纪90年代，出现了一批新的软件开发方法，其中，Booch、OMT和OOSE等方法得到了广泛的认可。然而，采用不同方法进行建模不利于开发者之间的交流。而UML则统一了Booch、OMT和OOSE的表示方法，而且对其作了进一步的发展。1997年，UML被国际对象组织OMG采纳为面向对象的建模语言的国际标准，它融入了软件工程领域的新思想、新方法和新技术。UML不仅支持面向对象的分析与设计，而且还支持从需求分析开始的软件开发的全过程。数年来，UML凭借其简洁明晰的表达方式、超凡脱俗的表达能力，为业界所广泛认同。

Booch是面向对象方法最早的倡导者之一，他提出了面向对象软件工程的概念。1991年，Booch将之前面向Ada的工作扩展到面向整个对象设计领域。Booch方法较适用于系统的设计和构造。

20世纪90年代，Rumbaugh等人提出了面向对象的建模技术（OMT，Object Modeling Technology，一种软件开发方法），该方法采用了面向对象的概念，并引入独立于语言的表示符，同时使用对象模型、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。该方法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，但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OOSE（Object-Oriented Software Engineering）方法，该方法最大的特点是面向用例（Use-Case），并在用例描述中引入了外部角色的概念。用例的概念是精确描述需求的“重要武器”，同时用例贯穿于整个开发过程，包括对系统的测试和验证。

此外，同一时期还有Coad/Yourdon方法产生，即著名的OOA/OOD（Object-Oriented Analysis/Object-Driented Design），它是最早的面向对象的分析和设计方法之一。该方法简单、易学，适合于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的局限，目前已很少使用。

面对众多的建模语言，用户首先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千秋；第三，虽然不同的建模语言大多雷同，但仍存在某些细微的差别，这极大地妨碍了用户之间的交流。因此，在客观上极有必要在精心比较不同建模语言的优缺点及总结面向对象技术应用实践的基础上，组织联合设计小组，并根据应用需求，取其精华，去其糟粕，求同存异，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 加盟到这一工作。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（Unified Modeling Language）。

1996 年，一些机构将 UML 作为其商业策略。UML 的开发者得到了来自公众的正面反应，并倡议成立了 UML 成员协会，以完善、加强和促进 UML 的定义工作。当时的企业成员有 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 月发布）及 UML 1.1（1997 年 11 月 17 日发布）的定义和发布起了重要的促进作用。

1.1.2 UML 定义

首先，UML 融合了 Booch、OMT 和 OOSE 方法中的基本概念，而且这些基本概念与其他面向对象技术中的基本概念大多相同，因而，UML 必然成为这些方法以及其他方法的使用者乐于采用的一种简单一致的建模语言；其次，UML 不仅是上述方法的简单汇合，而是在这些方法的基础上广泛征求意见，集众家之长，几经修改而完成的，它扩展了现有方法的应用范围；第三，UML 是标准的建模语言，而不是标准的开发过程。尽管 UML 的应用必然以系统的开发过程为背景，但由于不同的组织和不同的应用领域，需要采取不同的开发过程。

UML 的定义包括 UML 语义和 UML 表示法两个部分。

- UML 语义：指 UML 元素符号代表的含义，UML 的所有元素在语法和语义上提供了简单、一致、通用的定义和说明，使开发者能在语义上取得一致，消除了因人而异的最佳表达方法所造成的影响。此外，UML 还支持元素语义的扩展。
- UML 表示法：对 UML 每个元素符号的表示方法进行了规范。开发者或开发工具在使用这些图形符号时都遵循相应的 UML 符号的表示准则。

1.1.3 UML 的特点

UML 统一了 Booch、OMT 和 OOSE 等方法中的基本概念，主要特点如下。

- UML 是非专利的第三代建模和规约语言。在开发阶段，UML 语言用于说明、可视化、构建和书写面向对象软件制品。
- UML 语言应用于软件开发周期中的每一个阶段。OMG（Object Management Group，对象管理组织）已将该语言作为业界的标准。
- UML 最适用于数据建模、业务建模、对象建模和组件建模。
- UML 作为一种模型语言，它可以使开发人员专注于建立产品的模型和结构。当模型建立之后，模型可以被 UML 工具转化成指定的程序语言代码。

UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言，它融入了软件工程领域的新思想、新方法和新技术，它支持从需求分析开始的软件开发的全过程。

1.2 模型

模型就是用图形对一个物体或系统的简化表示，如地球仪就是一个模型，它是对地球的简化表示。用户可以用模型来表示现实领域中的业务流程和工作流程，也可以用模型表示软件领域中的软件系统的组成和结构。

1. 描述系统的模型

在软件领域，将软件系统的模型分为逻辑模型和物理模型两大类。逻辑模型只描述未来系统应该做什么，并不规定采用什么技术来实现系统。逻辑模型不依赖任何技术，系统分析阶段的模型都是逻辑模型；物理模型不仅描述了未来系统做什么，还规定了实现系统所采用的技术，在设计阶段的模型都是物理模型。物理模型是逻辑模型的进一步细化。

2. 描述事物的模型

任何事物都可以用模型来简化表示，下面是生活中常常遇到的 4 种模型。

1）交通模型。道路交通图、道路交通标志等图示，如图 1-1 的模型就是对广州地铁的表示。

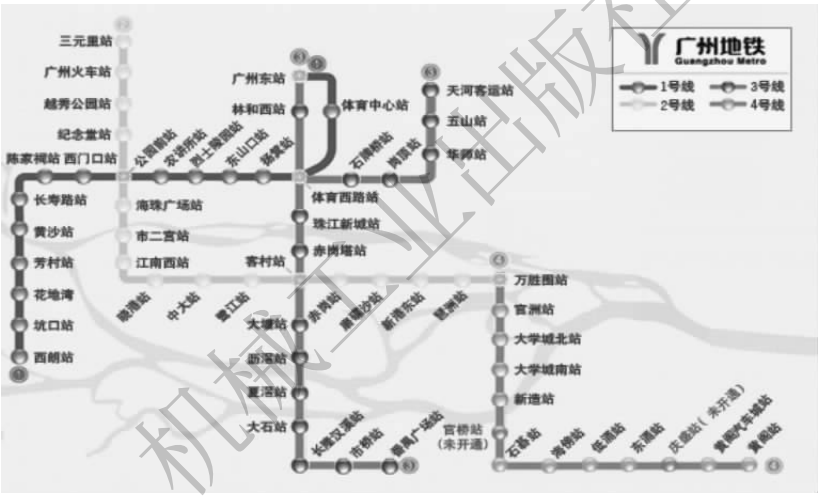


图 1-1　广州地铁模型

2）建筑模型。建筑物模型、沙盘等用来描述建筑物的图形，如图 1-2 就是描述某集团公司建筑物的模型。

图 1-2　建筑模型

3）设计模型。用来描述管线设计、电路板设计的图形。如图 1-3 就是描述某个局部电路的设计模型。

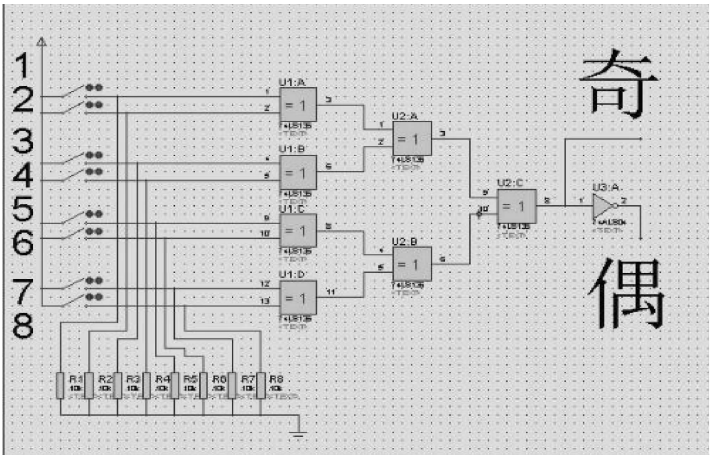


图 1-3　电路设计模型

4）数据分析模型。我们常见的条形图、饼状图。如图 1-4 就是描述某公司四种产品年销售所占份额。

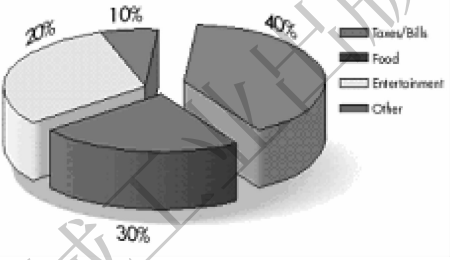


图 1-4　数据分析模型

上述模型是用图形符号对现实世界中某个事物的模仿或仿真。在软件开发工程中，模型主要用来描述问题域和软件域。问题域主要包括业务、业务规则、业务流程和工作流程等；软件域主要包括软件组成、软件结构和软件部署等。

1.2.1　模型的用途

开发软件活动包括两个方面的工作：第一是理解业务和需求（理解问题域）。即，对业务内容、业务过程和业务规则的理解和表示；第二是规划和设计软件系统（设计解决方案）。

由于业务规模和复杂度不断增加，软件的规模和复杂度也随之不断增加，因此，人们对业务的理解以及对软件的设计和构造也越来越困难。此时，在理解业务和需求时，只有借助 UML 这种建模语言来表示和理解业务；在规划和设计软件系统时，只有借助 UML 语言来设计和构造软件系统，以表示和展现系统的组成和交互。

总之，在软件开发活动中，UML 主要用于两个方面的建模：第一是，用 UML 语言对业务系统建模，便于分析师展现和理解业务；第二是，用 UML 对软件系统建模，便于设计师设计、修改软件系统。

1. 对业务系统建模

业务建模就是用 UML 符号表示业务内容和业务过程。用户对自己的业务过程建模，不仅是为了理解业务的内容中规定了要做什么，业务是如何进行的，同样也是为了识别业务的变更对业务造成的影响。对业务建模，有助于发现业务的优缺点，找出需要改进和优化的地方，在某些情况下还可以对几个可选的业务过程进行仿真。

2. 对软件系统建模

软件建模就是用 UML 符号表示软件的体系和组成，方便软件设计人员理解和修改软件方案，确保软件设计和计划能正确的实现，同时，一旦设计和计划需要修改时，修改后的软件系统同样经受得起时间的检验，例如当在一个软件系统中增加一个组件时，要保证系统不会因为增加了该组件而崩溃。

1.2.2 建模目标

对业务系统和软件系统进行建模，主要实现下面 5 个目标。

1）对业务系统进行可视化，建立业务模型。以业务模型为中介，领域专家、用户和需求分析师便于对业务内容和业务过程的理解和交流。

2）对软件架构进行可视化，建立软件体系模型。以体系模型为中介，设计师便于对软件系统宏观理解。

3）对软件系统的组成、结构和系统交互的行为进行建模，便于设计师和代码编写人员对软件深入理解。

4）用模型的方式为系统实现提供一个模板，开发人员可以依据该模板构造软件系统。

5）通过模型的方式将计划和决策文档化。

1.2.3 建模原则

对业务和软件建模是为了通过模型展示业务和软件，方便开发人员理解和交流。通过需求模型，用户与分析师共同理解业务需求；通过设计模型，分析师与设计师共同完成软件设计任务；通过设计模型，设计师能方便的构造和修改软件模型。为了实现这些目标，建模时应遵循以下几个原则。

1）仅当需要时才为业务或软件系统构建模型。简单的业务和软件系统不需要建立模型。

2）模型应该真实地反映业务系统的需求，或者模型能反映软件系统本身的组成和结构。

3）模型应该反映设计师的设计方案。

4）构建模型时，最好用一组相对独立的模型从不同的侧面描述重要的业务或软件系统。

1.2.4 为什么要建模

UML 是一种公共的、可扩展的、应用广泛的设计语言，它可应用于软件开发活动中的每个阶段（分析、设计、实现、测试），而且可以表示每个阶段的产品。

作曲家会将其大脑中的旋律谱成乐曲，建筑师会将其设计的建筑物绘制成蓝图，这些乐曲、蓝图就是模型（Model），而建构这些模型的过程就称为建模（Modeling）。软件开发如同音乐谱曲及建筑设计，其过程中也必须将需求、分析、设计、实现、部署等各项工作流程的构想和设计蓝图表示出来，供分析师、设计师、程序员、测试人员沟通、理解和修改。

建立大厦和搭建狗窝的区别是搭建狗窝不需要设计。因为建立大厦规模庞大，并且设计复

杂，所以，建立大厦前必须有大厦的设计蓝图，而搭建一所狗窝很简单，不需要特别的设计。同理，要开发大规模的复杂软件系统，必须首先了解系统需求，然后对未来的软件系统设计一个蓝图，即，对软件系统进行建模。

1.2.5 系统开发中的模型分类

软件系统从分析、设计到实现的整个过程，包含多种模型的开发。下面描述常见的两种模型分类方法。

1. 按模型的用途分类

如果按模型在软件开发过程中所起的作用，将它们分为3种，它们是：用来表示业务或软件系统的组成和结构的对象模型；用来表示业务系统或软件系统功能的用例模型；用来表示业务或软件系统中组件是如何交互的动态模型。

- 用例模型：从用户的角度展示系统的功能。用例模型常由用例图表示。
- 对象模型：模型展示了软件系统的组成和结构。对象模型由类图或对象图表示。
- 动态模型：展现系统的内部行为。动态模型常由顺序图、活动图和状态图表示。

2. 按产生模型的阶段性分类

在软件开发过程中，不同阶段产生不同的模型。模型按软件开发的阶段性可分为以下五种。

- 业务模型：展示业务过程、业务内容和业务规则的模型。常用对象模型表示业务模型。
- 需求模型：展示用户要求和业务要求的模型。需求模型常由用例模型表示（用例模型由用例图和用例描述组成）。
- 设计模型：设计模型包含架构模型和详细设计模型。架构模型展示软件系统的宏观结构和组成；详细设计模型展示软件的微观组成和结构。详细设计模型常由对象模型表示。
- 实现模型（也称为物理模型）：描述了软件组件及其关系（常由构件图或部署图组成）。
- 数据库模型：描述数据组成及其关系。

1.3 UML 工具与工具选择

UML 工具是帮助软件开发人员方便使用 UML 的软件，它的主要功能包括：支持各种 UML 模型图的输入、编辑和存储；支持正向工程和逆向工程；提供和其他开发工具的接口。不同的 UML 工具提供的功能不同，各个功能实现的程度也不同。在选择 UML 工具时应主要考虑的几方面因素是：产品的价格、产品的功能以及与自己的开发环境结合是否密切。

目前主要的 UML 工具有 Rational 公司的 Rose、Together Soft 公司的 Together 和 Microsoft 公司的 Visio 等。

1. UML 工具介绍

Rational 公司推出的 Rose 是目前最好的基于 UML 的 Case 工具之一，它把 UML 有机地集成到面向对象的软件开发过程中。不论是在系统需求分析阶段，还是在对象设计、软件的实现与测试阶段，它都提供了清晰的 UML 表达方法和完善的工具，方便建立其相应的软件模型。使用 Rose 可以方便地进行软件系统的分析和设计，并能很好地与常见的程序开发环境衔接在一起。

Rose 具有正向工程、逆向工程和对象模型更新等功能。用户修改模型后可以直接反映到代码上，同样，用户对代码框架的修改也可以反映到模型上。另外，它还提供对多种程序设计语言的支持，如 C ++ 、Java、Visual Basic 等。

Visio Professional 2000 提供了内建的 UML 支持，如 Visio 绘图工具提供绘制多种图形的功能，这是一个相当有价值的工具。

2. 如何选择 UML 工具

UML 支持的工具众多。当用户需要 UML 工具时，应该如何从中进行选择？如何选中符合自己要求，同时具有合适价格的工具？下面主要从技术方面来介绍在选择 UML 工具时应注意的几个方面。

（1）支持 UML 1.3

虽然许多工具声称完全支持 UML 1.3，但实际上很难做到这一点。目前很多工具并不能做到完全支持 UML 1.3，但至少应支持用例图、类图、合作图、顺序图、包图以及状态图。

（2）支持项目组的协同开发

对于一个大型项目，开发人员之间必须共享设计模型图。UML 工具应允许某个开发人员拥有整个模型，而其他人员只能以只读方式访问该模型，或者将这些组件结合到自己的设计中。需要注意的是，这种工具应允许从另一个模型中只引入所需要的组件，而不必引入整个模型。

（3）支持双向工程

支持正向工程和逆向工程是一项复杂的需求，不同厂商的工具在不同程度上支持这一点。正向工程在第一次从模型产生代码时非常有用，这项技术将节省许多用于编写类、属性以及方法代码等琐碎工作的时间。将代码转换成模型或重新同步模型和代码时，逆向工程就显得非常有用。一种好的建模工具应该支持双向工程，即支持以下五种功能。

- HTML 文档化

好的建模工具为对象模型及其组件无缝地产生 HTML 文档。而 HTML 文档应包括模型中的每个图形，以便开发者可以通过浏览器迅速查询。

- 打印支持

好的建模工具能够使用多个页面把一张大图准确地打印出来，同时提供打印预览和缩放功能，并且能够允许将每一张模型图放置在单页中打印。

- 健壮性

软件模型的健壮性很重要，在设计期间，应保证工具不发生崩溃。

- 开发平台

要慎重地考虑工具将运行在哪种平台上，UML 工具应与应用系统保持平台一致。

- 提供 XML 支持

XML 将成为各种工具之间数据交换的标准格式，支持 XML 将为软件的未来提供更好的兼容性。

以上介绍了选择 UML 工具应该考虑的主要因素。在实际选择时，还应综合考虑 UML 工具的价格、服务以及通用性等方面的因素。

1.4 UML 语言应用

UML 语言的目标是以图的方式来表示任何类型的系统。这种语言既可以用来为软件系统

建模，也可以用来对非软件系统建模。如，UML 语言可以对机械系统、企业机构或业务过程建模，以及对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建模。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。

此外，UML 适用于系统开发过程中从需求规格描述到系统完成后测试的不同阶段。在需求阶段，可以用用例来捕获用户需求；在静态分析阶段，主要识别系统中的类及其关系，并用 UML 类图来描述系统；在动态分析阶段，尝试组织多个对象，并构思对象的交互和协作，以实现和检验用例的可行性，此时可以用 UML 动态模型来描述。要注意的是，在分析阶段，只对问题域中的实体建模，而不考虑软件系统中类的定义和细节（如处理用户接口、数据库、通信和并行性等问题的类），这些技术细节将在设计阶段引入。因此设计阶段为编程（构造阶段）提供了更详细的规格说明。

编程（构造阶段）是一个独立的阶段，其任务是用面向对象编程语言将设计阶段的类转换成实际的代码。在用 UML 建立分析和设计模型时，应尽量避免考虑把模型转换成某种特定的编程语言。因为在早期阶段，模型仅仅是理解和分析系统结构的工具，过早考虑编码问题不利于建立简单、正确的模型。

UML 模型还可作为测试阶段的依据。系统通常需要经过单元测试、集成测试、系统测试和验收测试。不同的测试小组使用不同的 UML 图作为测试依据：单元测试使用类图和类规格说明；集成测试使用部件图和合作图；系统测试使用用例图来验证系统的行为；验收测试由用户进行，以验证系统测试的结果是否满足在分析阶段确定的需求。

总之，标准建模语言 UML 适用于以面向对象技术来描述任何类型的系统，而且适用于系统开发的不同阶段。

1.5 小结

本章介绍了 UML 的基本概念、主要内容和应用领域，还介绍了 UML 工具方面的知识。通过本章的学习，希望读者能够对 UML 有一定的认识和了解，为以后各章的学习打下基础。

本书主要从应用的角度来介绍 UML 的基本概念，不是一本完整详尽的用户手册，读者如果需要深入了解某些 UML 的概念和特殊用法，可以参考相关的手册。

1.6 习题

1. 什么是 UML?
2. UML 在软件开发中做什么用?
3. 指出 UML 的 3 个主要特性。
4. 简要说明建模的目标和建模的原则。
5. UML 的主要模型有几种？每种模型图的用途是什么?
6. 常用的 UML 工具有哪些？各自的特点是什么?

第 2 章　UML 语言体系

UML 语言是由符号系统组成的设计语言。它由构造块、规则和公共机制三个部分构成。本章将对 UML 语言体系进行系统介绍。

2.1　UML 语言组成

1. UML 语言

UML 语言由三个部分组成，它们是构造块、规则和公共机制，其结构如图 2-1 所示。

2. 构造块

UML 构造块又细分为三种：事物、关系和图。

1）事物：事物代表了系统中的简单实体（如，学生、老师、教师等等）。

2）关系：关系代表了实体间的联系（如，同学关系、同事关系等等）。

3）图：图代表了由实体按某种规则连接在一起组成的更大颗粒的实体（如，桌子这个实体由 4 条腿和一个桌面按照某种关系连接在一起）。图与图通过关系符号连接在一起组成更大的图，这种图代表更复杂的事物。如图 2-2 所示是构造块的三种类型。

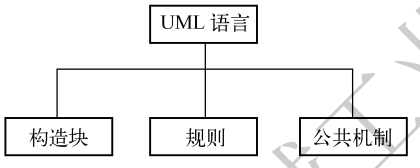


图 2-1　UML 语言组成

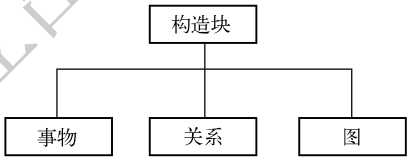


图 2-2　构造块的三种类型

3. 规则

规则是指构造块应该遵守的规定，即，用 UML 构造块描述软件系统或业务系统中事物时应该遵守的约束或规定。规则包括：名称、作用范围、可见性、完整性和可执行等属性。构造块应该遵守的规则如图 2-3 所示。

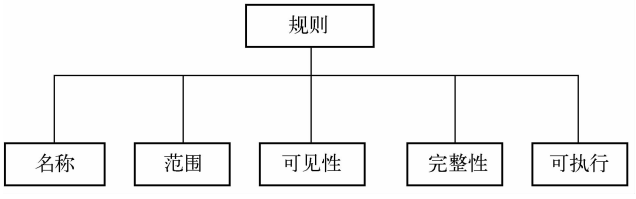


图 2-3　构造块应遵守的规则

1）名称：指每个构造块代表的事物应该有一个名字。

2）范围：每个构造块代表的事物的使用范围。

3）可见性：访问构造块代表的事物时，授予访问者的权限或者级别。

4）完整性：构造块代表的事物应该有完整的含义。

5）可执行：构造块代表的事物的行为具有实际意义和合理性。

4. 公共机制

公共机制是指每个事物必须遵守的通用规则。可以将公共机制进一步细分为：详述、修饰、通用划分以及扩展机制。公共机制的组成如图 2-4 所示。

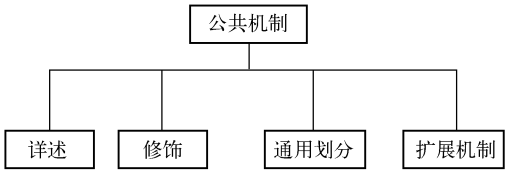


图 2-4　公共机制的组成

下面几节分别对事物、关系、图和视图的概念、表示法作详细介绍。

2.2　事物

事物进一步划分为 4 类：结构事物、行为事物、分组事物和注释事物。

2.2.1　结构事物

结构事物进一步细分为七种，分别是类和对象、接口、主动类、用例、协作、构件与结点。下面分别介绍 7 种结构事物的概念和表示法。

1. 类和对象

类是对具有相同属性、相同操作以及相同关系的一组对象的共同特征的抽象，即，类是对一组对象共同特征的描述。类是对象的模板，而对象是类的一个实例。

（1）类的表示

在 UML 中，类用一个长方形表示，如图 2-5 所示。垂直地把长方形分为三栏，第一栏列出类名，第二栏列出类的属性，第三栏列出类的操作。类名不能省略，属性和操作可以不列出。

图 2-5 是 Flight 类（航线）的 UML 表示法。第一栏，类名是 Flight；第二栏列出了 Flight 类的 3 个属性，它们分别是：flightNumber，departureTime 和 flightDuration；第三栏列出了 Flight 类的两个操作，它们分别是：delayFlight( ) 和 getArrivalTime( )。

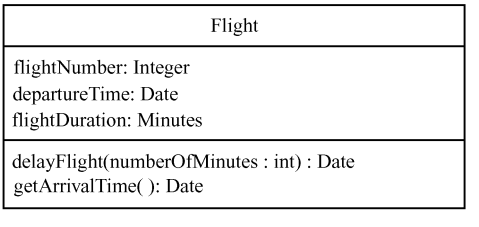


图 2-5　Flight 类的 UML 符号

（2）对象的表示

对象也是用一个长方形来表示。只是用“对象名：类名”的格式表示一个对象，并且，对象名和类名下面必须带下划线。表示对象时，第二和第三栏可以省去。例如，图 2-6 所示是对象“李世民”的 UML 表示法。

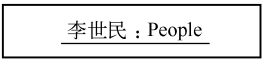


图 2-6　对象“李世民”的 UML 表示法

2. 主动类

一个对象可以是主动的也可以是被动的。主动对象可以改变自身状态，被动对象只有在接收到消息后才会改变自身的状态。例如，定时器和时钟就是主动对象，它们可以在没有外部事件触发的情况下改变它们自身状态。银行账户就是被动对象，银行账户的属性不会发生变化，除非银行账户接收到一条设置余额（一种用于更新账户余额的操作）的消息，账户才改变状态。

主动类是指创建主动对象的类。主动类的表示与一般类相似，只是矩形框用粗线表示而已，如主动类 Radio 的表示方法如图 2-7 所示。

3. 接口

因为访问类、对象、或构件是通过其方法来实现的，因此把类、对象、构件的方法集合称为接口。接口向外界声明了类（或构件）能提供的服务。

接口分为供给接口和需求接口两种，供给接口只能向其他类（或构件）提供服务，需求接口表示类（或构件）需要用到其他类（或构件）提供的服务。

上述两种接口的表示方法如图 2-8 所示。

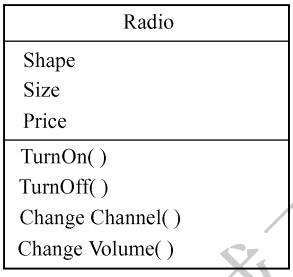


图 2-7　主动类 Radio 的表示方法

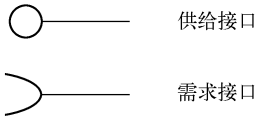


图 2-8　表示接口的 UML 符号

4. 用例

我们把完成某个任务而执行的一序列动作的集合称为场景。例如，客户小刘在柜员机上取款 500 元的动作集合就是一个场景，客户小王在柜员机上取款 300 元的动作集合也是一个场景。无论多少个客户，他们在柜员机上取款的一序列动作是相似的，即取款的场景相似，只是取款时，输入的密码、取款金额不同。

用例是对一组相似场景共同行为的描述，例如，我们可以用一个动作序列来描述所有取款客户的共同行为。因此，用例的每一次的具体执行就是一个场景，即，场景是用例的一个实例，是用例的一次具体执行。用例与用例实例的关系正如类与对象的关系。

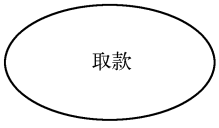


图 2-9　用例“取款”的表示方法

在 UML 中，用例是用一个实线椭圆来表示的，在椭圆中写入用例名称，如用例“取款”的表示方法如图 2-9 所示。

5. 协作

在系统中，把一组对象之间相互发送和接收消息的现象称为交互，把一组对象为了完成某个任务执行的交互现象称为协作。

我们知道，用例的一次具体执行就是一个场景，在某个场景中，存在多个对象间的交互。对象之间通过交互实现某个场景，即，通过对象间的协作来实现场景。本质上说，协作就是用例的实现。

协作用一个带两个分栏的虚线椭圆表示，例如，用例“销售”用协作图表示时，其对应的表示方法如图 2-10 所示。

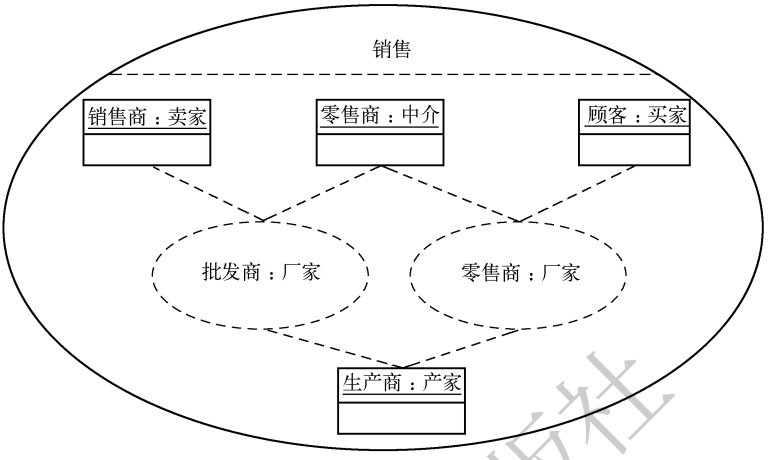


图 2-10　用例“销售”协作图

协作图“销售”表示的语义如下。

1）生产商生产出产品并以低价售给批发商和零售商，从中获得了利润。

2）批发商以比生产商较高的价格将产品出售给销售商或零售商，零售商在自己的商店得到更高利润。

3）顾客以较高的价格买到自己想要的商品。

6. 构件

构件也称组件，它是指软件系统设计中的一个相对独立的软件部件，它把功能实现部分隐藏在内部，对外声明了一组接口（包括供给接口和需求接口）。因此，两个具有相同接口的构件可以相互替换。

构件是比“对象”更大的软件部件，例如一个 COM 组件、一个 DLL 文件、一个 JavaBeans 以及一个执行文件都可以是构件。构件通常采用带有两个小方框的矩形表示，构件的名字写在方框中，如图 2-11 所示。

7. 结点

结点是指硬件系统中的物理部件，它通常具有存储空间或处理能力，如 PC（个人计算机）、打印机、服务器、显示器等都是结点。在 UML 中，用一个立方体表示一个结点，例如，结点“显示器”的表示方法如图 2-12 所示。

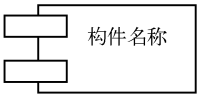


图 2-11　表示构件的 UML 符号

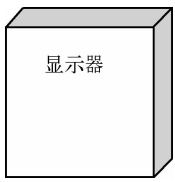


图 2-12　结点“显示器”的 UML 符号

2.2.2 行为事物

行为事物是用来表示业务系统或软件系统中事物之间的交互以及交互引起的事物状态的改变。行为事物描述了事物的动态特征。一般从两个方面描述事物的行为特征：事物之间的交互和事物的状态变化，描述这两个方面的符号也有两种：一种符号表示事物间的交互；一种符号表示事物的状态。

1. 交互

交互用来表示对象之间的相互作用，即，发送和接收消息的现象。

可以用一条有向直线来表示对象间的交互，并在有向直线上方标注消息名称即可，如图 2-13 所示。

2. 状态

事物处于某个特定属性值时的现象称为状态。在对象生命周期内，在事件驱动下，对象从一种状态迁移到另一状态，这些状态序列就构成了状态机，即一个状态机由多个状态组成。

在 UML 中，状态用一个圆角矩形表示，状态名称写在矩形框中。例如，手机处在“正在通话”状态的表示方法如图 2-14 所示。

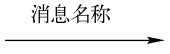


图 2-13　表示交互的 UML 符号

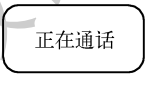


图 2-14　表示“正在通话”状态的 UML 符号

2.2.3 分组事物

在开发大型软件系统时，通常会包含大量的类、接口以及用例，为了能有效地对这些类、接口和用例进行分类和管理，就需要对其进行分组。在 UML 中可通过“包（Package）”来实现这一目标。

表示“包（Package）”的图形符号与 Windows 中表示文件夹的图符很相似，包的作用与文件夹的作用也很相似。如，java 语言中的“java. awt”包，用 UML 符号表示，则如图 2-15 所示。

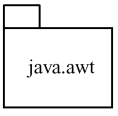


图 2-15　表示“java. awt”包的 UML 符号

2.2.4 注释事物

注释就是对其他事物进行解释，一般用文字进行注释。注释符号用一个右上角折起来的矩形表示，解释的文字就写在矩形框中，如图 2-16 所示。

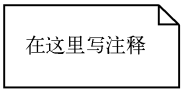


图 2-16　表示注释的 UML 符号

2.3 关系

上一节介绍了代表事物的构造块，本节将介绍代表事物之间联系的符号，即关系。在 UML 中，共定义了 24 种关系，相应的有 24 种关系符号，如表 2-1 所示。

表 2-1　UML 中的关系及其符号

关 系	关系细化	UML 中的关系	UML 符号	关 系	关系细化	UML 中的关系	UML 符号
抽象	派生	依赖关系	《derive》	导入	私有	依赖关系	《access》
	显现		《manifest》		公有		《import》
	实现	实现关系	虚线加空心三角	信息流			《flow》
	精化	依赖关系	《refine》	包含并			《merge》
	跟踪		《trace》	许可			《permit》
关联		关联关系	实线	协议符合			未指定
绑定		依赖关系	《bind》（参数表）	替换		依赖关系	《substitu－te》
部署			《deploy》	使用	调用		《call》
扩展	Extend		《extend》（扩展点）		创建		《create》
扩展	extension	扩展关系	实线加实心三角		实例化		《instanti－ate》
泛化		泛化关系	实线加空间三角		职责		《responsi－bility》
包含		依赖关系	《include》		发送		《send》

上述有 24 种关系，在 UML 中，可以归纳为关联关系、实现关系、泛化关系、扩展关系和依赖关系 5 种，下面介绍这些关系的表示方法。

1. 关联关系

只要 2 个类之间存在联系，我们就认为 2 个类之间存在关联。关联是人们赋予事物之间的联系。实现关系、泛化关系、扩展关系和依赖关系都属于关联关系，只是这些关系更具体，更明确。关联关系是对关系的最高层次的抽象。在所有关系中，关联的语义最弱。

在关联关系中，有两种比较特殊的关系，它们是聚合关系和组合关系。聚合和组合关系能通过 java 语言实现，关联关系不能通过 java 语言实现，所以，在设计阶段，我们必须把分析阶段的关联关系细化为更具体的关系，如，细化为聚合关系，或者组合关系，依赖关系等等。

（1）关联关系的表示

关联关系是比较抽象的关系，它包含的语义较少，聚合关系和组合关系是更具体的关联关系，它包含的语义更具体。在 UML 中，使用一条实线来表示关联关系，如图 2-17 所示。

图 2-17　表示关联关系的 UML 符号

（2）聚合关系

聚合（Aggregation）是整体与部分的关系，是一种特殊形式的关联。聚合关系是一种松散的对象间关系——计算机与它的外围设备就是聚合关系。一台计算机（整体）和它的外设（部分）之间松散地结合在一起，这些外设可以与其他计算机共享。即，部分可以离开整体而存在。

聚合的表示方法如图 2-18a 所示。其中棱形端表示事物的整体，另一端表示事物的部分。如计算机就是整体，外设就是部分。

（3）组合关系

如果发现“部分”类的存在是完全依赖于“整体”类的，那么就应使用组合关系来描述。组合关系是一种非常强的对象间关系，就像树和树叶之间的关系，树和它的叶子紧密联系在一起，叶子完全依赖树，它们不能被其他的树所分享，并且当树死去时，叶子也会随之死去——

这就是组合关系，在组合关系中，部分依赖于整体而存在。组合关系是一种强的聚合关系，它的表示方法如图 2-18b 所示。

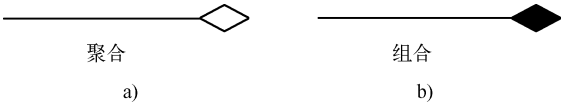


图 2-18　表示聚合关系和组合关系的 UML 符号

2. 泛化关系

泛化关系描述了从特殊事物到一般事物之间的关系，也就是子类到父类之间的关系，或者子接口到父接口的关系。表示泛化关系的符号是从子类指向父类的带空心箭头的实线，其表示方法如图 2-19 所示。而从父类到子类的关系则是特化关系。

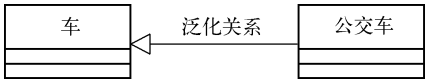


图 2-19　表示泛化关系的 UML 符号

3. 实现关系

实现关系是用来规定接口与实现接口的类之间的关系。接口是操作的集合，这些操作声明了类或组件所提供的服务。表示实现关系的符号是从类指向接口的带空心箭头的虚线，其表示方法如图 2-20 所示。

图 2-20　表示实现关系的 UML 符号

4. 依赖关系

假设有两个元素 X、Y，如果元素 X 的值发生变化，就会引起元素 Y 的值的变化，则称元素 Y 依赖于元素 X。依赖关系的表示如图 2-21 所示。

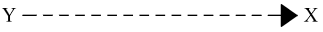


图 2-21　表示依赖关系的 UML 符号

如果两个元素是类，则类间的依赖现象有多种，如一个类向另一个类发送消息；一个类是另一个类的数据成员；一个类是另一个类的某个方法的参数。

从本质上说，聚合、组合、泛化以及实现关系都属于依赖关系，但是它们有更特别的语义。

5. 扩展关系

在 UML 中，用一个带箭头的实线表示扩展关系，如图 2-22 所示。这里的扩展含义是指对一个元类的扩展，即，通过扩展元类的语义，获得新的元类。

图 2-22　表示扩展关系的 UML 符号

2.4 图和视图

构造块代表了简单事物，简单事物通过一定关系组合成复杂事物，图就是用来表示复杂事物的。每个图是由代表简单事物的构造块和代表事物联系的关系构成。

2.4.1 UML 图

UML 中的图可分为两大类：结构图和行为图。结构图描绘系统中事物的组成及关系；行为图描述系统中事物间的交互行为。下面是 UML 图的组成，如图 2-23 所示。

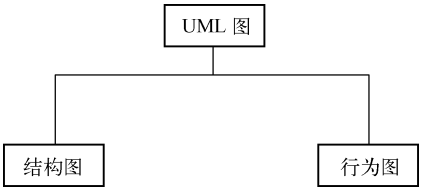


图 2-23　UML 图的组成

1. 结构图

结构图又分为 6 种，如图 2-24 所示。

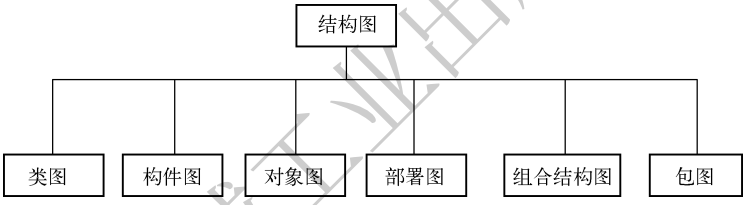


图 2-24　结构图组成

（1）类图

类图是使用 UML 建模时最常用的图，它展示了系统中的静态事物、它们的结构以及它们之间的相互关系。我们常用类图描述系统的逻辑设计和物理设计。

（2）构件图

构件图描述了一组构件组成和依赖关系。它用于说明软件系统中构件的组成、结构和如何协同工作。

（3）对象图

对象图可以展示系统中对象的组成和结构、是系统在某一时刻的快照。对象图是类图在某一时刻的快照。

（4）部署图

部署图可以展示系统中物理结点的组成、结构和运行时的体系结构。同时也展示了软件部件是如何部署在硬件上的。部署图展示了硬件和软件系统的体系结构。

（5）组合结构图

组合结构图展示系统、构件的内部结构及其内部关系。

（6）包图

包图可以展示系统的组成和包之间的依赖关系。包图常用来对系统中的元素进行分组，并用来表示软件的体系结构。

2. 行为图

行为图又细分为 7 种，如图 2-25 所示。

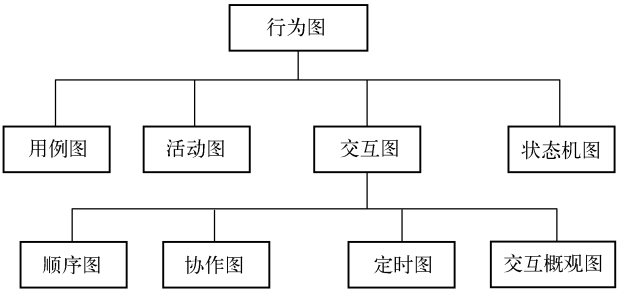


图 2-25　行为图组成

（1）活动图

活动图显示系统内部的活动控制流程。通常需要使用活动图描述不同的业务过程。

（2）状态图

状态图显示对象从一种状态迁移到其他状态的转换过程。例如可以利用状态图描述一个电话路由系统中交换机的状态迁移过程。不同的事件触发交换机转移到不同的状态。在 UML 2.0 中，状态图被称为状态机图。

（3）协作图

协作图（也称通信图）是交互图的一种。协作图突出对象之间的合作，以及交互时每个对象承担的职责。

（4）顺序图

顺序图是另一种交互图，它强调系统中对象相互作用时消息的先后顺序。

UML 2.0 中又增加了下列几种行为图。

（5）时间图

时间图也是一种交互图，它描述与交互对象的状态转换或条件变化有关的详细时间信息。

（6）交互概观图

交互概观图是一种高层视图，用于从总体上显示交互序列之间的控制流。

注意：在实际进行系统建模时，几乎没有人会使用到 UML 标准中定义的所有图。

（7）用例图

用例图描述了系统外部参与者如何使用系统提供的服务，描述了系统中用例之间的关系。

注意：组合结构图、包图及用例图是 UML 2.0 中新增的结构图。

3. 图的功能

在 UML 2.0 中共定义了 13 种图。表 2-2 列出了这 13 种图的功能。

表 2-2　UML 2.0 中的图

图　分　类	作　　用	描　　述
类图	描述系统中的类组成和类之间的关系	与 UML 1.0 相同
对象图	描述系统在某个时刻对象的组成和关系	UML 1.0 非正式图
组合结构图	描述复合对象的内部结构	UML 2.0 新增
构件图	描述构件的结构与组成	与 UML 1.0 相同
部署图	描述在系统中各个结点上的构件及其构件之间的关系	与 UML 1.0 相同
包图	描述系统的宏观结构，并用包来表示	UML 中非正式图
用例图	描述用户与系统如何交互及系统提供的服务	与 UML 1.0 相同
活动图	描述活动控制流程及活动结点的转换过程	与 UML 1.0 相同
状态机图	描述对象生命周期内，在外部事件的作用下，对象从一种状态如何转换到另一种状态	与 UML 1.0 相同
顺序图	描述对象之间的交互，重点在强调消息发送的顺序	与 UML 1.0 相同
协作图	描述对象之间的交互，重点在于强调对象的职责	UML 1.0 中的协作图
定时图	描述对象之间的交互，重点在于描述时间信息	UML 2.0 新增
交互概观图	是一种顺序图与活动图的混合嫁接	UML 2.0 新增

从使用的角度来看，可以将 UML 的 13 种图分为结构模型（也称为静态模型）和行为模型（也称为动态模型）两大类。

2.4.2　UML 视图

图描述系统某个方面的局部特征，多个相关的图可以描述系统的某个方面的全部特征，我们把描述系统某个方面全部特征的多个图的集合称为视图。

在 UML 参考手册第 2 版中，将 UML 图划分为 4 大应用类型和 9 种视图，如表 2-3 所示。

表 2-3　UML 图和视图

应 用 类 型	视　　图	组　　成
结构领域	静态视图	类图，对象图
	设计视图	复合结构图、协作图、构件图，对象图
	用例视图	用例图
动态领域	状态视图	状态机图
	活动视图	活动图
	交互视图	顺序图、通信图，时间图，交互概述图
物理领域	部署视图	部署图
模型管理	模型管理视图	包图
	特性描述	包图

其中，结构领域的视图描述了系统中的成员及其相互关系；动态领域的视图描述了系统随时间变化的行为；物理领域的视图描述了系统的硬件结构和部署在这些硬件上的系统软件；模型管理领域的视图说明了系统的分层组织结构。

2.5 规则和公共机制

1. 规则

在 UML 中，代表事物的构造块在使用时应遵守一系列规则，每个构造块必须遵守的 3 个规则如下。

- 名称：代表事物和关系的每个构造块应该有一个名字，即，事物、关系和图都应该有一个名字。和任何语言一样，名字即是一个标识符。
- 范围：每个事物都有它的作用的范围。相当于程序设计语言中变量的“作用域”。
- 可见性：UML 元素（代表事物的构造块）属于一个类或包中，因此，所有事物都存在访问级别，即可见性。在 UML 中，为元素定义了 4 种可见性，如表 2-4 所示。

表 2-4 UML 元素的可见性

元素的可见性	规　　则（假设被访问的元素在包中）	标准表示法
public	任一元素若能访问包，则就可以访问包中的元素	+
protected	只有包中的元素或子包才能访问它	#
private	只有包中的元素才能访问它	-
package	只有声明在同一个包中的元素才能访问该元素	~

2. 公共机制

在 UML 语言中，定义了 4 种公共机制：规格描述、修饰、通用划分和扩展机制。

（1）规格描述

在 UML 语言中，每个元素都有一个对应的图形符号，同时，对每个图形符号的语义有一个详细的文字描述，这种对图形符号的语义进行的文字描述称为规格描述，也称为详述。

如图 2-26 所示，在左边的方框中有三个用图形符号表示的用例，分别是：“存款”“取

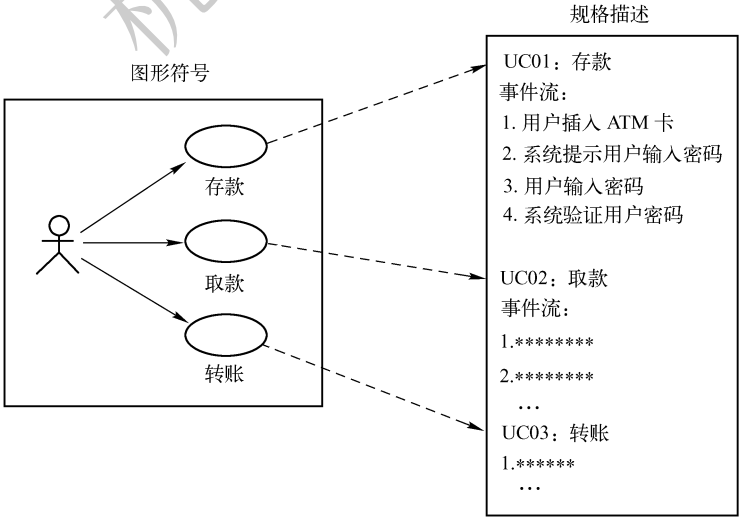


图 2-26 图形符号与对应的规格描述

款”“转账”。在右边的方框中，分别对每个图形符号表示的用例进行了详细的文字描述，即规格描述。

（2）修饰

在 UML 中，每个元素符号对事物的主要方面提供了可视化表示，而若想将事物的细节表示出来，则必须对元素符号加以修饰。例如，用斜体字表示抽象类，用 +，－符号表示元素的访问级别，这些都是通过修饰符号来表示事物的细节。所谓修饰就是增加元素符号的内涵，为被修饰的元素提供更多的信息。

（3）通用划分

在 UML 中，通过分组将事物分成两组，它们是：类与对象、接口与实现。

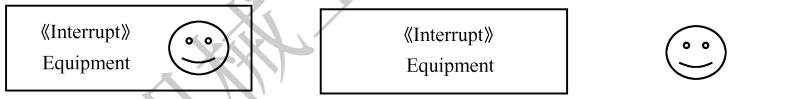
- 类与对象：类是对对象共同特征的描述、是对象的模板，而对象则是类的实例。
- 接口与实现：接口是一种声明、一个合同、是一组方法的集合，而实现则是完成一个合同、实现接口中的声明。

在 UML 中，用例就是一种对功能的声明和定义，是对事物功能的抽象描述；而协作则是实现用例的功能；操作名是声明服务的，而方法体则是实现服务的。因此，用例与协作、操作名与方法体之间的关系就是接口与实现的关系。

（4）扩展机制

由于 UML 中定义的元素符号不能将现实世界中所有事物的特征表示出来，因此需要通过一些方法对元素符号进行扩展。UML 提供的扩展机制有三种：构造型、标记值和约束。

1）构造型。构造型就是指分析师自己定义一种新的 UML 元素符号，给这种新的元素符号赋予特别的含义，例如，分析师可以定义一个元素符号《Interrupt》，用该元素符号代表“中断”。如图 2-27 所示，就是用自定义的符号《Interrupt》表示中断的三种不同表示方法（分析师赋予符号《Interrupt》的语义是：设备“中断”）。



第一种方法（构造符号加图标）　第二种方法（构造符号）　第三种方法（图标）

图 2-27　构造型的 3 种表示方法

- 第一种表示方法：创建一种新的 UML 元素符号《Interrupt》，表示“中断”，在构造元素符号右边放置一个图标，构造符号“《Interrupt》”与图标一起代表“中断”。
- 第二种表示法：创建一种新的 UML 元素符号《Interrupt》，表示“中断”，这是一种标准表示方法。
- 第三种表示方法：直接用一个图标表示新的构造元素符号，该符号的语义是“中断”。

2）标记值。标记值是为事物添加新特征，即，为事物增加一个属性。其格式是：“{标记名 = 标记值}”，标记名代表事物的属性，标记值表示了事物的属性值。例如，{name =“李小平”}。其中，标记名是 name；分隔符是 = ；标记值是“李小平”。

3）约束。约束是用来标识元素之间约束条件，是用来增加新的语义或改变已存在规则的一种机制（通过文本和 OCL 两种方法表示约束）。其中，OCLLObject Constraint Language）是

一种对象约束语言。约束的表示方法和标记值的表示方法类似，都是使用花括号括起来的字符串。

2.6 系统建模与视图

当用 RUP（Rational Unified Process，统一开发过程）软件开发模型开发软件系统时，可以从五个角度（五种视图）对软件系统进行建模，这五个视图分别是用例视图、设计视图、构件视图、并发视图和部署视图，即从 5 个角度来描述系统的五个方面。在这五个视图中，以用例视图为目标，分别构造其他四个视图。

下面是描述软件系统的 5 种视图，如图 2-28 所示。

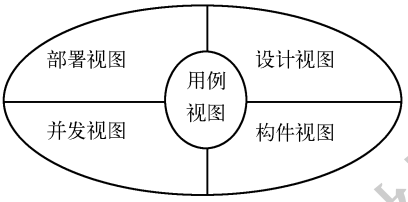


图 2-28 软件系统

1. 用例视图

描述了系统的功能和参与者。用例视图由多个用例图组成。

2. 设计视图

又称逻辑视图，描述了软件系统的组成、结构和行为，是软件系统的蓝图。设计视图常由类图、交互图、状态图和活动图组成。

3. 构件视图

描述了软件系统的组成和结构。视图描述了系统包含的软件构件和文件。该视图常由一组构件图组成。

4. 并发视图

描述系统各部分之间的同步和异步执行情况。该视图由状态图和活动图来描述。

5. 部署视图

描述了软件系统的各部分如何部署到各硬件结点上。该视图常用部署图、交互图、状态图和活动图描述。

2.7 小结

本章首先指出了 UML 是由构造块、规则和公共机制 3 个方面所组成的，然后对这 3 个方面展开进一步说明。

首先，阐述了事物构造块和关系构造块，它们是 UML 建模元素的主体。其中，事物构造块又包括结构事物、行为事物、分组事物和注释事物 4 种类型；关系构造块详细地描述了关联、泛化、依赖和实现 4 种主要的关系。

接着简要阐述了 UML 中公共的规则，并以命名规则、范围规则和可见性规则为例说明了它们对 UML 模型的影响。

然后系统地介绍了规格说明、修饰、通用划分和扩展机制。用户可以通过构造块添加新的事物，通过标记值添加新的特性，通过约束更好地体现模型，通过扩展机制为 UML 建模能力添加新的功能。

在本章的最后又介绍了“图”这个最重要的构造块，简要地阐述了 UML 2.0 中定义的 13 种图，以及不同图的划分和类别。同时还结合 RUP 中的“4 +1”视图来说明系统体系结构的表示方法。

2.8 习题

1. 填空题

（1）UML 中主要包含四种关系，分别是_____、_____、_____和_____。

（2）物理视图包含两种视图，分别是_____和_____。

（3）常用的 UML 扩展机制分别是_____、_____和_____。

（4）UML 的通用机制分别是_____、_____和_____。

2. 选择题

（1）UML 中的事物包括结构事物，分组事物，注释事物和_____。

（A）实体事物　　（B）边界事物

（C）控制事物　　（D）动作事物

（2）UML 中的四种关系是依赖、泛化、关联和_____。

（A）继承　　（B）合作

（C）实现　　（D）抽象

（3）下面不是 UML 中的静态视图是_____。

（A）状态图　　（B）用例图

（C）对象图　　（D）类图

3. 问答题

（1）在 UML 中定义了哪几种可见性规则？

（2）规格描述与元素有何区别？

（3）什么是构造型？为什么要引入构造型？

（4）约束有两种表示法，它们分别是什么？

第3章　类　　图

类图可以显示出类、接口以及它们之间的静态结构和关系。通常用类图来描述业务系统或软件系统的组成和结构。

3.1　类的表示

在 UML 中，要表示一个类，主要是标识出它的名称、属性和操作。类由一个矩形表示，矩形框分成三栏，在第一栏显示类的名称；在第二栏显示类的属性；在第三栏显示类的操作。BankAccount 类的表示如图 3-1 所示。

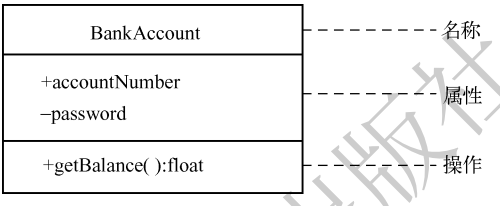


图 3-1　BankAccount 类

BankAccount 类包含两个属性，分别是：accountNumber，password，包含一个操作 getBalance( )。在类图中，属性和操作可以省略。BankAccount 类可以表示为图 3-2 和图 3-3 的形式。

图 3-2　省略属性和操作

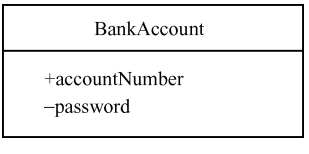


图 3-3　省略操作

1. 名称

每个类都有一个名称，类名是不能省略的，其他部分则可以省略。类名的书写格式有两种，即，简单名和全名。

1）简单名：见图 3-1 中的 BankAccount 类，表示类时没有将它所属的包表示出来。

2）全名：见图 3-2 中，在表示 BankAccount 类时，在类名的前面加了它所属的包名（java）。在包名和类名之间加上分隔符号::。在 UML 中，Date（类）表示为：java::awt::Date，而在 java 语言中表示为：java. awt. Date。

2. 属性

属性描述了类的静态特征，在面向对象编程中，把属性表示为成员变量。

在属性的前面有一个修饰，用来表示属性的可见性（在 java 语言中称为访问权限），在 java 语言中，表示属性的可见性分别是：public、private、protected 和默认权限，在 UML 中，相应的表示为 + 、 - 、#、 ~ ，如表 3-1 所示。

表 3-1　java 的可见性符号与 UML 中可见性符号对应表

Java 符号	UML 符号	Java 符号	UML 符号
public	+	protected	#
private	-	package	~

3. 操作

操作是类所提供的服务。在面向对象编程语言中，它通常表示为成员方法。对操作的表示方法说明如下。

1）一般来说，操作的可见性修饰应该声明为 public，否则其他类无权访问该类提供的服务。

2）操作名的参数可以省略不写。

3）如果属性和操作名之前没有可见性修饰符，则表示可见性是 package（包）级别。如果属性或操作名具有下划线，则说明它是静态的。

4. 职责

职责是指类承担的责任和义务。在矩形框中最后一栏中写明类的职责，如图 3-4 所示。

5. 约束

约束是指对类的属性值进行的限定或者要求。在 UML 中，约束是用花括号括起来的自由文本，如图 3-5 所示，表示 BankAccount 类的 password 的值不能为空。

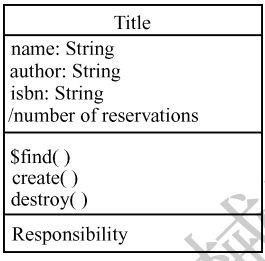


图 3-4　职责的表示

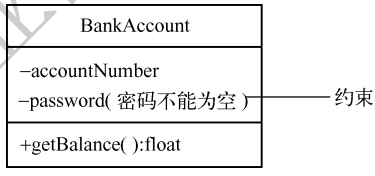


图 3-5　约束的表示

3.2　类图的概念

类图是描述类、对象、接口及其关系的图。与所有 UML 的其他图一样，类图还可以包括注释、约束和包。

1. 类图示例

图 3-6 是一个展示飞机（Plane 类）、引擎（Engine 类）和控制软件（ControlSoftware 类）关系的类图，该图表示一个 Plane 类如何由四个引擎和两个控制软件对象组成。

图 3-6 表示的含义如下。

1）图中包含三个类：Plane 类、Engine 类和 ControlSoftware 类。

2）图中包含的关系：2 个组合关系和一个关联关系。对关系的理解如下。

- 一架飞机拥有 4 个引擎。
- 一架飞机拥有 2 个控制软件。
- 一个引擎与 0 个到 2 个控制软件相关。

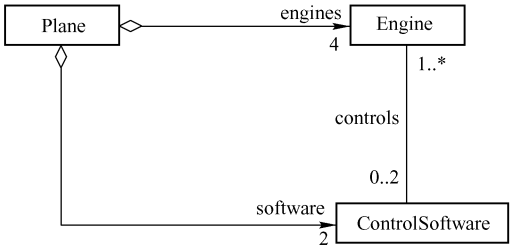


图 3-6　展示 3 个类的类图

2. 类图中的元素

类图中可以包含的元素有：类、接口、协作、注释、约束和包。关系把类、协作、接口连接在一起构成一个图，注释的作用是对某些类和接口进行解释，约束的作用是对某些类和接口进行约束。

3. 类图中的关系

一般来说，类图中的关系包括：依赖关系、泛化关系、关联关系和实现关系。

3.3　类图中的元素

前面介绍过，任何一个图是由元素和关系构成的。类图中的主要元素有：类、接口、包。其中，类还可以进一步细分为：抽象类、关联类、模板类、主动类、嵌套类。下面分别介绍。

1. 抽象类和抽象方法

抽象类是一种不能直接实例化的类，也就是说不能用抽象类创建对象。

Shape 类是一个抽象类，因为不能画出一个形状（Shape），只能画出它的子类（比如方形、圆形等、多边形等），因此，Shape 类中的 draw( ) 方法是一个抽象方法。如图 3-7 所示给出了抽象类 Shape 及其子类的示例的标准表示法。在这些子类中，都对父类 Shape 中的抽象方法 draw( ) 进行了重写，即实现了父类中的 draw( ) 方法。因此，可以调用 Retangle、Polygon 和 Circle 类中的 draw( ) 操作分别画矩形，多边形和圆。

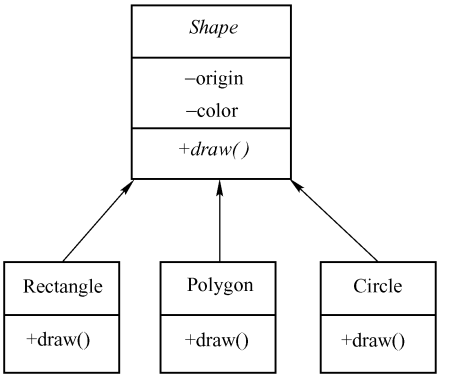


图 3-7　标准表示法

在 UML 中，抽象类和抽象方法有 2 种表示方式。

（1）UML 标准表示法

抽象类和抽象方法的名字用斜体表示，如图 3-7 所示。

（2）UML 草图表示法

在抽象类和抽象方法名前加符号《abstract》，如图 3-8 所示。

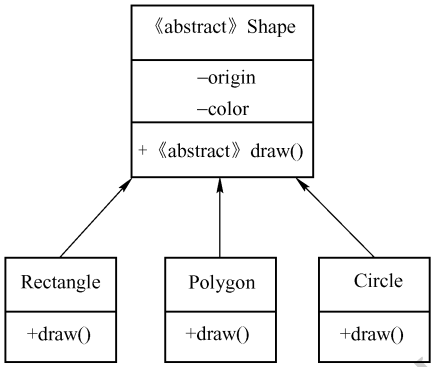


图 3-8　草图表示法

2. 接口

接口是抽象方法的集合（见 java 语言中接口的定义）。接口又细分为 2 种，即供给接口和需求接口。

在 UML 中，接口有两种表示方法，一种是用图标表示；另一种是用构造型《Interface》表示。

（1）图标表示法

供给接口用一个小圆表示，需求接口用一个半圆表示。用图标表示接口时，没有列出接口包含的方法，这种表示法适合于草图应用，如图 3-9 所示。

（2）构造符号表示法

构造符号《Interface》和接口名写在方形框的第一栏，在方形框的第二栏中列出多个常量（可以省去该栏），在第三栏中列出多个抽象方法。如图 3-10 所示，接口名是 IUnknown。

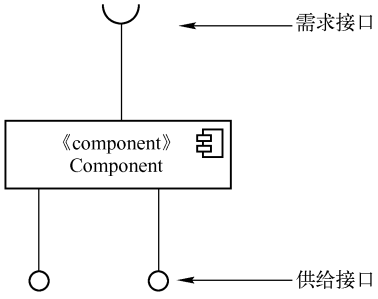


图 3-9　图标表示接口

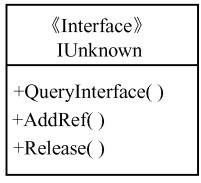


图 3-10　构造型表示接口

3. 关联类

在多对多的关系中，有些属性不属于关联两端任何一个类，例如，在工资管理系统中有两个类：person（人）和 Company（公司），显然一个人（person）可以在多个公司

（Company）工作，同时每个公司（Company）雇佣多个人（person），因此它们之间是多对多的关系。

如果要记录某个 person 在所属公司的工资（salary），应该把工资（salary）属性放在哪个类中呢？因为工资属性既不属于 person，也不属于 Company，只有某个人与公司建立了工作关系后工资属性（salary）才存在。因此，应该创建一个关联类 job，将工资属性加入这个 job 类中，如图 3-11 所示。

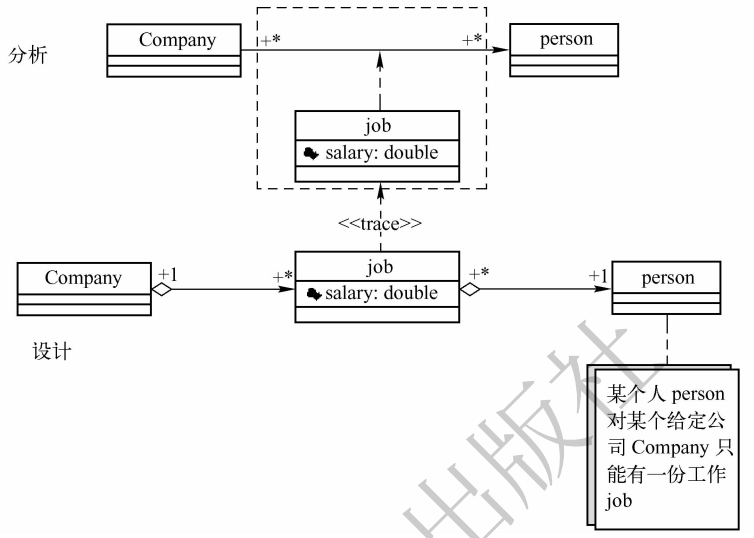


图 3-11　关联类 job

在分析阶段可以使用关联类。但是，任何 OO（Object - Oriented，面向接口）编程语言都不能直接将关联类映射为程序代码。因而，必须将分析阶段的关联类具体化为设计类（设计类是指可以用编程语言实现的类）。

设计师必须综合使用关联、聚合、组合，甚至依赖来捕获关联类的语义，将关联类具体转化成可以用编程语言实现的类，这可能需要对分析模型添加约束。通过确定某一端为关联的整体，以及使用相应的聚合和导航性，将关联类转换为具体可以实现的设计类。图 3-11 中，我们将分析阶段的关联类转换为了设计阶段的制品。

4. 模板类

模板类（template）又称作参数化类。模板类定义了一族类。模板类拥有一个参数表，这个参数表中的参数称作形参，当用实际的参数代替模板类中的形参后，才能创建一个具体的类。

模板类的 UML 表示法与一般类的表示法一样，只是在方形框的右上角拥有一个虚线框，在这个虚线框中列出了形参表。其表示方法如图 3-12 所示。

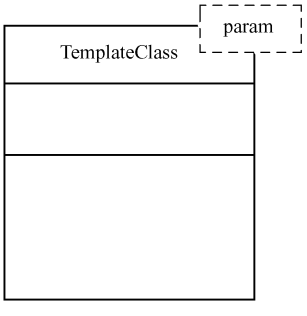


图 3-12　模板类的表示法

例如，某个应用要求定义一些类，以处理整型、字符串数组。一般的做法是为整型和字符型数组各创建一个类，这两个类除了数据类型不同之外，其他都相同。

设计师可以定义一个模板类，如图 3-13a 所示，在模板类中有 2 个形参（type,size）。形

参 size 的数据类型是 int，其默认值是 20。

为了创建整型数组类，用实参值（type = int，size = 50）代替形参后，就创建了类 IntArray。为了创建字符串数组类，用实参值（type = String，size = 10）代替形参后，就创建了类 StringArray。如图 3-13b 所示。

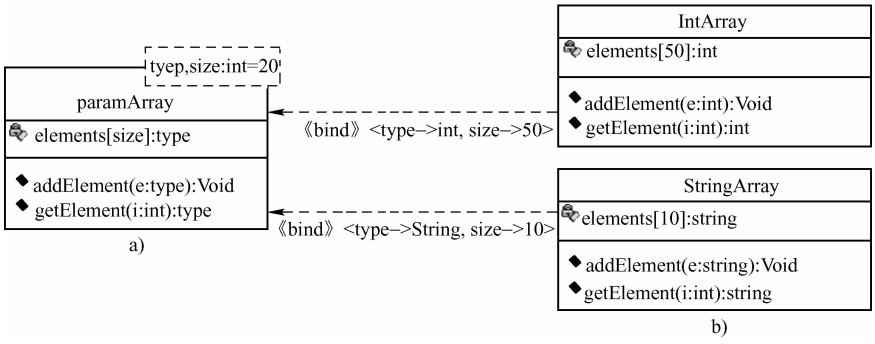


图 3-13　模板类示例

a）数据类　b）用模板创建的类

5. 主动类

程序执行时，能主动改变自身状态的对象就是主动对象，用于创建主动对象的类就是主动类。在程序执行期间，一个主动对象能够控制自身的活动，具有独立的控制权。

例如，时钟类（Clock）就是一个主动类，因为，用该类创建的时钟对象能改变自身的状态。在 UML 2.0 中，主动类的表示法是在方形框的两边各增加一条垂直线，如图 3-14 所示。

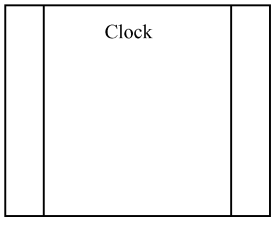


图 3-14　主动类的表示方法

6. 嵌套类

在 Java 的语言中，允许将一个类的定义放在另一个类定义的内部。在外层定义的类被称作外部类，在内部定义的类被称作内部类，类的这种关系就是嵌套类。在 UML 中，可以采用一个锚图标来表示这种关系，如图 3-15 所示。

例如，一个外部类是 HelloFrame，内部类是 Mouse 的嵌套类，定义格式如下。

```
public classHelloFrame extend Frame
{
    Class Mouse
    {
        …
    }
}
```

因为类 HelloFrame 是 Frame 的子类。用 UML 符号表示上面 3 个类之间的关系，如图 3-15 所示。

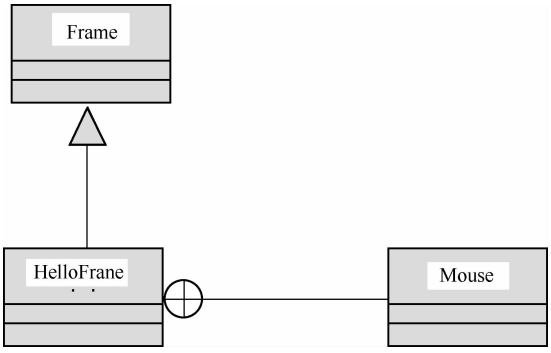


图 3-15　嵌套类表示法

3.4　类间关系

在 UML 中，按照关系的性质将关系分为 5 种，它们是依赖关系、泛化关系、关联关系、实现关系和扩展关系，其中，扩展关系在用例图中讲解，下面分别说明其他 4 种关系。

3.4.1　依赖关系

对于两个相对独立的对象，当一个对象的变化会影响另一个对象时，则这两个对象之间的关系表现为依赖关系。

比如说你要去拧螺丝，就必须借助（也就是依赖）螺丝刀（Screwdriver）来帮助你完成拧螺丝（screw()）的工作。那么你（Person）与螺丝刀（Screwdriver）的关系就是依赖关系。即，你依赖于螺丝刀，用 UML 来表示这种依赖关系，如图 3-16 所示。

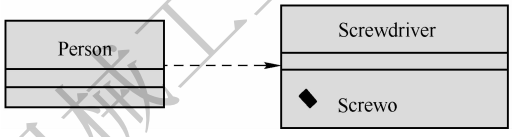


图 3-16　依赖关系的示例

上图的依赖关系用 java 语言来实现时，则代码如下。

```
public class Person{
    /**拧螺丝 */
    public void screw(Screwdriver screwdriver){
        screwdriver.screw();
    }
}
```

我们把提供服务的对象称为提供者，把使用服务的对象称为客户。因此，在图 3-16 中，我们称 Screwdriver 为提供者，称 Person 为客户。

依赖关系可以细分为 4 大类，即使用依赖、抽象依赖、授权依赖和绑定依赖。

（1）使用依赖

表示客户使用提供者提供的服务，以实现它的行为，下面都属于使用依赖的具体形式。

- 使用（《use》）
- 调用（《call》）
- 参数（《parameter》）
- 发送（《send》）
- 实例化（《instantiate》）

（2）抽象依赖

客户与提供者属于不同的抽象事物，具体依赖形式有以下几种。

- 跟踪（《trace》）
- 精化（《refine》）
- 派生（《derive》）

（3）授权依赖

表达一个事物访问另一个事物的能力，具体依赖形式有以下几种。

- 访问（《access》）
- 导入（《import》）
- 友元（《friend》）

（4）绑定依赖

用绑定模板以创建新的模型元素时，其依赖形式为绑定（《bind》）。

3.4.2 泛化关系

从父类到子类的关系称为继承关系；从子类到父类的关系称为泛化关系（从特殊到一般）。泛化关系中的事物可以是类、接口和用例。

比如说，Animal 类与 Tiger 类和 Dog 类的关系就是泛化关系。用 UML 模型来表示这种泛化关系，如图 3-17 所示。

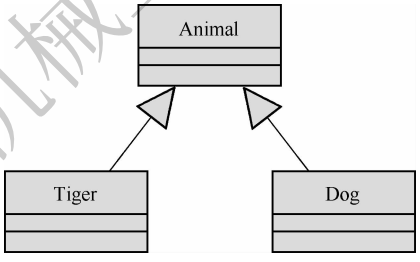


图 3-17　泛化关系

其中，Tiger 类和 Dog 类表示了特殊事物，Animal 类表示了一般性事物，因此，Tiger 类和 Dog 类分别是 Animal 类的特化，Animal 类是 Tiger 类和 Dog 类的一般化。从特殊事物中抽取共同特点构成一般性事物的过程是泛化的过程，在一般性事物中加入新特点的过程是继承的过程。

上图的泛化关系用 java 语言来实现时，则代码如下。

```
class Animal{}                          //定义一般性事物
class Tiger extends Animal{}            //定义 Tiger
classDog extends Animal{}               //定义 Dog
```

```
public class Test
{
    public void test( )
    {
        Animal a = new Tiger( );
        Animal b = new Dog( );
    }
}
```

3.4.3 实现关系

类与被类实现的接口、协作与被协作实现的用例都是实现关系。

比如说，Professor 和 Student 类与 Person 接口就是实现关系。Person 作为接口被定义，Professor 类和 Student 类分别实现了 Person 接口。用 UML 来表示这种实现关系时，其模型如图 3-18 所示。

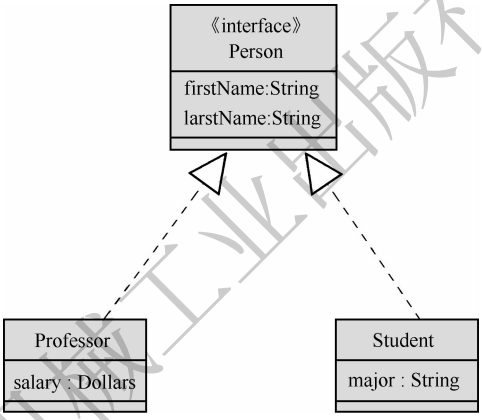


图 3-18　实现关系

3.4.4 关联关系

关联关系是一种最抽象的、最不具体的语义关系。只要建模者认为两个事物之间存在某种关系，我们就认为它们具有关联关系。但是，我们可能不知道它们是如何相关的，即，不知道关联的属性。

有五种关联，在这一部分中，将讨论它们中的四种：双向关联、单向关联、聚合关系和组合关系。

（1）双向关联

关联是两个类间的连接。关联总是被假定为双向的，即，两个类彼此知道它们间的联系。

图 3-19 显示了在 Flight 类和 Plane 类之间的一个标准类型的关联。

一个双向关联用两个类间的实线表示。在线的任一端，放置一个角色名和多重值。图 3-19 显示 Flight（航班）类与一个特定的 Plane（飞机）类相关联，而且 Flight 类知道这个

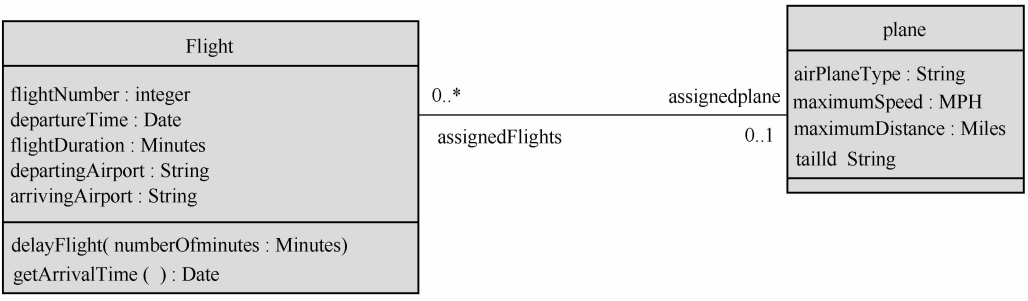


图 3-19　双向关联的示例

关联。因为角色名以 Plane 类表示，所以 Plane 承担关联中的“assignedPlane”角色。Plane 类后面的 0..1 表示当一个 Flight 实体存在时，可以有一个或没有 Plane 与之关联（即，Plane 可能还没有被分配给某个航班）。图 3-19 也显示 Plane 知道它与 Flight 类的关联。在这个关联中，Flight 承担“assignedFlights”角色；图 3-19 告诉我们，Plane 实体可以不与 Flight 关联（例如，它是一架全新的飞机，没有安排航班）。

（2）单向关联

在一个单向关联中，两个类是相关的，但是只有一个类知道另一个类的存在。图 3-20 显示单向关联的一个示例。

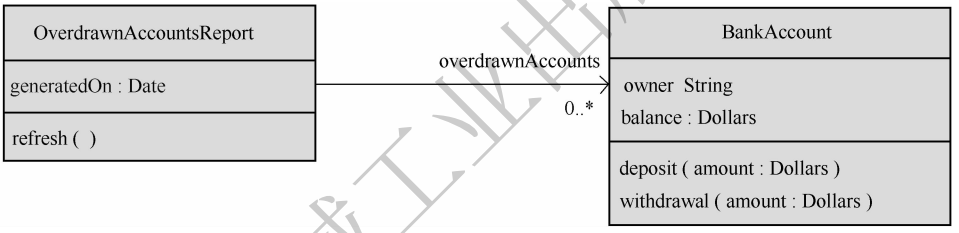


图 3-20　单向关联的示例

OverdrawnAccountsReport 类（透支财务报告）知道 BankAccount 类（账户）存在，而 BankAccount 类对 OverdrawnAccountsReport 类一无所知。

在构建 UML 模型时，应该把知道关联存在的类画在箭尾端，把不知道关联存在的类画在箭头端。

在单向关联中，应该在箭头端写出关联的角色名和一个多重值。在图 3-20 中，OverdrawnAccountsReport 知道 BankAccount 类，而且知道 BankAccount 类扮演“overdrawnAccounts”的角色。然而，和标准关联不同，BankAccount 类并不知道它与 OverdrawnAccountsReport 相关联。

（3）聚合关系

聚合是一种特别类型的关联，用于描述“总体与局部”的关系。在聚合关系中，部分的生命周期独立于整体的生命周期。

图 3-21 所示，车（Car）与车轮（Wheel）的关系就是整体与部分之间的关系。车是一个整体，而车轮是整个车的一部分。轮胎可以在安置到车时的前几个星期被制造，并放置于仓库中。在这个实例中，Wheel 实例清楚地独立于 Car 类实例而存在。

图 3-21　聚合关系示例

（4）组合关系

组合关系是关联关系的另一种形式，也是用于描述“总体与局部”的关系。但是，部分的生命周期依赖于整体的生命周期。

图 3-22 所示，公司（Company）与部门（Departement）的关系也是总体与部分的关系。在公司存在之前，部门不会存在。这里 Department 类的实例依赖于 Company 类的实例而存在。

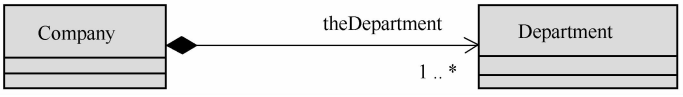


图 3-22　组合关系的示例

图 3-22 中，显示了 Company 类和 Department 类之间的组合关系，注意组合关系与聚合关系一样地绘制，不过这次菱形是被填充的。

注意： 在 Rose 工具中，表示组合的符号与在 UML 语言中表示组合的符号是不同的，如图 3-23 所示。

图 3-23　表示组合的符号

3.4.5　关联的属性

在类图中，依赖关系、泛化关系、实现关系已经是很具体的关系，而关联关系是对所有关系的一种泛指，是一种高度抽象的关系。为了对关联进一步具体化，确定关联的具体语义，就必须明确关联的属性。关联的属性包括名称、角色、多重性、导航性（方向）、限定性。

1. 名称

可以使用一个动词或动词短语给关联命名，用来描述关联的性质。关联名称应该描述关联两端类之间的关系。关联的名称不是必需的，在关联名和角色中选择一种即可。可以在关联的直线上方写上阅读方向的方向指示符，以消除阅读的歧义。

如图 3-24 所示，关联名称是“使用”，即用户使用计算机。

图 3-24　关联名称

2. 角色

在关联关系中，角色表明了关联的每一端在形成关联关系中所承担的职责，即关联发生时，关联的每一端在关联中扮演的角色。角色的名称应该是名词或名词短语，以解释对象是如

何参与关联的。

如图 3-25 所示的关联中，学生扮演的是学习者的角色，学校扮演的是教学者的角色。

3. 多重性

多重性就是某个类有多少个对象可以和另一个类的单个对象关联。

如图 3-26 所示的多重性表示：一个学校可以有 1 个或多个学生学习；一个学生可以到多个学校去学习，或不去任何学校学习。

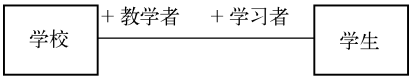


图 3-25　关联的角色

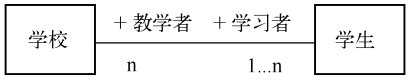


图 3-26　关联的多重性

4. 导航性（单向关联）

关联是双向的，即关联的两端都可以访问另一端。为了确定关联的方向，我们引进导航的概念。

导航性描述了以源类创建的对象可以将消息发送给由目标类创建的对象，反之，不能以目标类创建的对象的消息发送给源类创建的对象。导航性表示箭头从源类指向目标类。即，消息可以从源类对象发送到目标类对象，反之不可以，如图 3-27 所示。

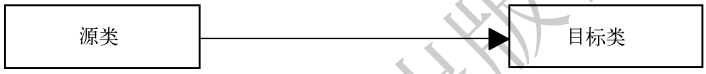


图 3-27　导航性

下面举一个例子，House（房子）与 Address（地址）是单向关联的，即房子对象可以访问地址，但是，地址是不能访问房子的。用 java 语言实现时其代码如下。

```
public class House
{
    private Addressaddre;    // House 引用 Address，角色名是 addre。
    …
}
```

从上面的代码可以看出，House 类创建的对象引用了 Address 类创建的对象，这样，House 对象可以访问 Address 对象，反之不可以。

用 UML 符号表示 House 类和 Address 类之间的关联关系如图 3-28 所示，显然，它们之间是单向关联，在这个关联中，目标类的角色名是 addre，在用 java 语言实现时，我们常把角色名作为成员变量。

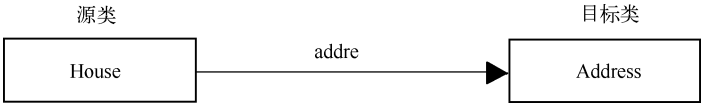


图 3-28　House 引用 Address

所有的面向对象的编程语言只能实现单向关联，不能实现双向关联，因此，我们在设计阶段的制品是不能有双向关联的，必须将双向关联转化成单向关联。

注意：如果我们标识了关联的导航方向，本质上表明了该关联是单向关联。

5. 限定符

如果源对象到目标对象是一对多关联，为了在多个目标对象的集合中查找到需要的目标对象，必须在目标对象集合中，选择一个唯一标识目标对象的查找键（限定符，从目标对象的属性中选择），它应该是目标对象中的某个属性，当然，也可以是表达式。图 3-29 所示，是一个标明了限定符号的关联关系。

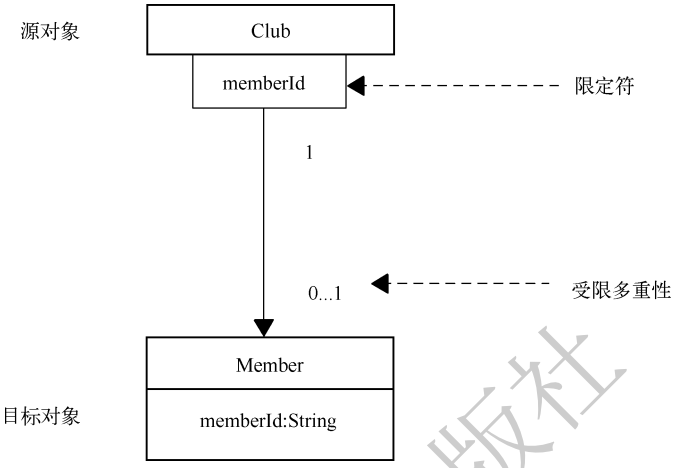


图 3-29　限定符

如图 3-29 所示说明一个俱乐部（Club）可以有多个成员（Member），为了在成员集合（目标对象）中找到需要的对象，从目标集合中选择 memberId 作为查找关键字，即限定符。

3.5　阅读类图

图 3-30 是一个描述汽车及旅客的类图。

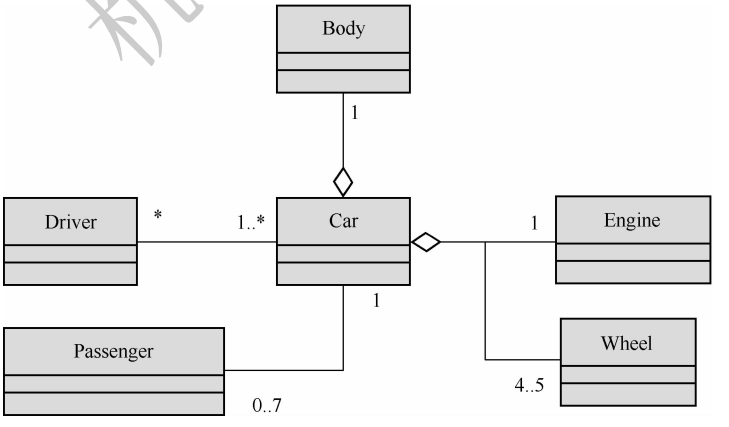


图 3-30　汽车与旅客关系类图

我们从关联最复杂的类开始理解上面的类图。

- Car 有一个 Engine。

- Engine 是 Car 的一个组成部分。
- Car 有 4 或 5 个 Wheel。
- 每个 Wheel 都是 Car 的一部分。
- 每个 Car 总是由一个 Body 组成。
- Body 总是 Car 的一部分，它随着 Car 一起存在和消亡。
- Car 可以有任意多个 Driver。
- Driver 至少可以驾驶一辆 Car。
- 一个 Car 承载 0 个 ~ 7 个 Passenge。
- 一个 Passenger 在某一时刻只能在一辆 Car 上。

3.6 小结

本章详细说明了 UML 中类图的概念、类的表示方法和类关系。在此基础上，还讲述了关系、多重性、导航、角色名称、限定符和约束等概念和表示方法，最后还讨论了高级概念：接口/抽象类、关联类、模板类、主动类、嵌套类。

3.7 习题

1. 填空题

（1）类之间的关系包括_____关系，_____关系，_____和_____关系。

（2）在 UML 的图形表示中，_____的表示法是一个矩形，这个矩形由三个部分构成。

（3）类中方法的可见性包含三种，分别是_____、_____和_____。

2. 问答题

（1）用实际例子绘制一个类，并指出它主要包含哪 3 个部分及其语义。

（2）在对类名、属性/方法命名时，通常会遵循什么规则？举例说明。

（3）举例说明嵌套类的概念？如果类 People 包含类 Student，请使用学过的、支持嵌套类的面向对象编程语言将其表示出来。