China-pub.com
下载

第一部分

背景知识

这一部分介绍了 UML的基本原理，包括 UML建模的性质和目标以及UML覆盖的所有功能领域。

China-pub.com
下载

第1章 UML 综述

本章是UML及其应用的一个快速浏览。

1.1 UML简介

统一建模语言（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制品的文档。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，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描述了一个系统的静态结构和动态行为。UML将系统描述为一些离散的相互作用的对象并最终为外部用户提供一定功能的模型结构。静态结构定义了系统中重要对象的属性和操作以及这些对象之间的相互关系。动态行为定义了对象的时间特性和对象为完成目标而相互进行通信的机制。从不同但相互联系的角度对系统建立的模型可用于不同的目的。

UML还包括可将模型分解成包的结构组件，以便于软件小组将大的系统分解成易于处理的块结构，并理解和控制各个包之间的依赖关系，在复杂的开发环境中管理模型单元。它还包括用于显示系统实现和组织运行的组件。

UML不是一门程序设计语言，但可以使用代码生成器工具将 UML模型转换为多种程序设计语言代码，或使用反向生成工具将程序源代码转换为 UML。UML不是一种可用于定理证明的高度形式化的语言，这样的语言有很多种，但它们通用性较差，不易理解和使用。UML是一种通用建模语言。对于一些专门领域，例如用户图形界面（GUI）设计、超大规模集成电路（VLSI）设计、基于规则的人工智能领域，使用专门的语言和工具可能会更适合些。UML是一种离散的建模语言，不适合对诸如工程和物理学领域中的连续系统建模。它是一个综合的通用建模语言，适合对诸如由计算机软件、固件或数字逻辑构成的离散系统建模。

China-pub.com 下载

1.2 UML 的历史

UML是为了简化和强化现有的大量面向对象开发方法这一目的而开发的。

1.2.1 面向对象的开发方法

利用传统程序设计语言（如Cobol和 Fortran语言）的软件开发方法出现于20世纪70年代，在80年代被广泛采用，其中最重要的是结构化分析和结构化设计方法[Yourdon-79]和它的变体，如实时结构化设计方法[Ward-85]等。这些方法最初由Constantine、DeMarco、Mellor、Ward、Yourdon和其他一些人发明和推广，在一些大型系统，特别是和政府签约的航空和国防领域的系统中取得了一定突破，在这些系统中，主持项目的政府官员强调开发过程的有组织性和开发设计文档的完备和充分。结果不总是像预料的那么好 —— 许多计算机辅助软件工程系统（CASE）只是摘录一些已实现的系统设计的报表生成器 —— 尽管如此，这些方法中仍包含一些好的思想，有时在一些大系统中是很有效的。商业应用软件更不愿采用大型的 CASE系统和开发方法。大部分商业企业都独立开发本企业内部使用的软件，客户和缔约人之间没有对立关系，而这种关系正是大型政府工程的特征。一般都认为商用系统比较简单，不论这种看法是否正确，反正它不需要经过外界组织的检查。

普遍认为，诞生于1967年的Simula-67是第一个面向对象的语言。尽管这个语言对后来的许多面向对象语言的设计产生了很大的影响，但是它没有后继版本。 80年代初，Smalltalk语言的广泛使用掀起了一场“面向对象运动”，随之诞生了Objective C、C++、Eiffel和CLOS等语言。起初，尽管面向对象语言在实际使用中有一定的局限性，但它仍然吸引了广泛的注意力。在Smalltalk语言成名约5 年后，第一批介绍面向对象软件开发方法的书籍出现了，包括Shlaer/Mellor [Shlaer-88]和Coad/Yourdon [Coad-91]，紧接着又有 Booch[Booch-91]、Rumbaugh/Blaha/Premerlani/Eddy/Lorensen [Rumbaugh-91]和Wirfs-Brock/Wilkerson/Wiener [Wirfs-Brock-90]（注意：图书的版权年代往往包括了上一年度 7月份以后出版的书）。这些著作再加上Goldberg/Robson[Goldberg-83]、Cox[Cox-86]和Meyer[Meyer-88] 等有关程序语言设计的著作，开创了面向对象方法的先河。第一阶段在 1990年末完成。稍晚，讲述Objectory方法的[Jacobson-92]出版了，它建立在以前的成果的基础上，介绍了一种稍微不同的方法，即以用例和开发过程为中心。

在以后的5年中，大批关于面向对象方法的书籍相继问世，各有自己的一套概念、定义、表示法、术语和适用的开发过程。有些书提出了一些新概念，但总的来说各个作者所使用的概念大同小异。许多后继出版的书都照搬前人，自己再做一些小的扩充或修改。最早的著作者也没闲着，他们大部分人都更新了自己前期的著作，采纳了其他人一些好的思想。总之，出现了一些被广泛使用的核心概念，另外还有一大批被个别人采纳的概念。即使在被广泛接受的核心概念里，在各个面向对象方法中也有一些小的差异。这些面向对象方法之间的细微比较常使人觉得这些概念不知依据哪个为好，特别是对于非专业的读者。

China-pub.com
下载

1.2.2 统一工作

在UML之前，已经有一些试图将各种方法中使用的概念进行统一的初期尝试，比较有名的是Coleman和他的同事们[Coleman-94]所创造的，包括OMT[Rumbaugh-91]、Booch[Booch-91]、CRC[Wirfs-Brock-90]方法的Fusion方法。由于这项工作没有这些方法的原作者参与，实际上仅仅形成了一种新方法，而不能替换现存的各种方法。第一次成功合并和替换现存的各种方法的尝试始于1994 年在Rational 软件公司Rumbaugh与 Booch的合作。他们开始合并OMT和Booch 方法中使用的概念，于1995年提出了第一个建议。此时，Jacobson 也加入了Rational公司开始与Rumbaugh和Booch一同工作。他们共同致力于设计统一建模语言。三位最优秀的面向对象方法学的创始人共同合作，为这项工作注入了强大的动力，打破了面向对象软件开发领域内原有的平衡。而在此之前，各种方法的拥护者觉得没有必要放弃自己已经采用的概念并接受这种统一的思想。

1996年，OMG发布了向外界征集关于面向对象建模标准方法的消息。 UML的三位创始人开始与来自其他公司的软件工程方法专家和开发人员一道制订一套使 OMG感兴趣的方法，并设计一种能被软件开发工具提供者、软件开发方法学家和开发人员这些最终用户所接受的建模语言。与此同时，其他一些人也在做这项富有竞争性的工作。 1997年9月，所有建议终于被合并成一套UML方法提交到OMG。 最后的成果是许多人共同努力的结果。我们发起了创建UML的工作并提出了一些有益的建议，但是这些建议的最终成型是集体智慧的结晶。

1.2.3 标准化

1997年11月，UML被OMG全体成员一致通过，并被采纳为标准。 OMG承担了进一步完善UML标准的工作。在UML标准通过前，就已经有许多概括UML精华的书出版发行。许多软件开发工具供应商声称他们的产品支持或计划支持 UML，若干软件工程方法学家宣布他们将使用UML的表示法进行以后的研究工作。 UML的出现似乎深受计算机界欢迎，因为它是由官方出面集中了许多专家的经验形成的，减少了各种软件开发工具之间无谓的分歧。我们希望建模语言的标准化既能促进软件开发人员广泛使用面向对象建模技术，同时也能带来 UML支持工具和培训市场的繁荣，因为不论是用户还是供应商都不用再考虑到底应该采用哪一种开发方法。

1.2.4 核心组员

提出UML建议或进行UML标准修订工作的核心组员有下列人员：

数据存取公司：Tom Digre

DHR 技术公司：Ed Seidewitz

HP 公司：Martin Griss

IBM 公司：Steve Brodsky, Steve Cook, Jos Warmer

China-pub.com 下载

I-Logix 公司：Eran Gery, David Harel
ICON Computing 公司：Desmond D'Souza
IntelliCorp and James Martin 公司：Conrad Bock, James Odell
MCI 系统企业：Cris Kobryn, Joaquin Miller
ObjecTime 公司：John Hogg, Bran Selic
Oracle 公司：Guus Ramackers
铂技术公司：Dilhar Desilva
Rational 软件公司：Grady Booch, Ed Eykholt, Ivar Jacobson, Gunnar Övergaard, Karin Palmkvist, James Rumbaugh
SAP 公司：Oliver Wiegert
SOFTEAM 公司：Philippe Desfray
Sterling 软件公司：John Cheesman, Keith Short
Taskon 公司：Trygve Reenskaug
Unisys 公司：Sridhar Iyengar, GK K halsa

1.2.5 统一的意义

“统一”这个词在UML中有下列一些相互关联的含义：

在以往出现的方法和表示法方面。UML合并了许多面向对象方法中被普遍接受的概念，对每一种概念，UML都给出了清晰的定义、表示法和有关术语。使用 UML可以对已有的用各种方法建立的模型进行描述，并比原来的方法描述得更好。

在软件开发的生命期方面。UML对于开发的要求具有无缝性。开发过程的不同阶段可以采用相同的一套概念和表示法，在同一个模型中它们可以混合使用。在开发的不同阶段，不必转换概念和表示法。这种无缝性对迭代的增量式软件开发至关重要。

在应用领域方面。UML适用于各种应用领域的建模，包括大型的、复杂的、实时的、分布式的、集中式数据或计算的、嵌入式的系统。也许用某种专用语言来描述一些专门领域更有用，但在大部分应用领域中，UML 不但不比其他的通用语言逊色，相反更好。

在实现的编程语言和开发平台方面。UML可应用于运行各种不同的编程实现语言和开发平台的系统。其中包括程序设计语言、数据库、4GL组织文档及固件等。在各种情况下，前部分工作应当相同或相似，后部分工作因各种开发媒介的不同而有某种程度上的不同。

在开发全过程方面。UML是一个建模型语言，不是对开发过程的细节进行描述的工具。就像通用程序设计语言可以用于许多风格的程序设计一样，UML适用于大部分现有的或新出现的开发过程。尤其适用于我们所推荐的迭代式增量开发过程。

在内部概念方面。在构建UML元模型的过程中，我们特别注意揭示和表达各种概念之间的内在联系并试图用多种适用于已知和未知情况的办法去把握建模中的概念。这个过程会增强对概念及其适用性的理解。这不是统一各种标准的初衷，但却是统一各种标准最重要的结

China-pub.com
下载

果之一。

1.3 UML的目标

UML语言的开发有多个目标。首先，最重要的目标是使 UML成为一个通用的建模语言，可供所有建模者使用。它并非某人专有，且建立在计算机界普遍认同的基础上，即它包括了各种主要的方法并可作为它们的建模语言。至少，我们希望它能够替代 OMT，Booch，Objectory方法以及参与UML建议制订的其他人所使用的方法建立的模型。其次，我们希望UML 尽可能地采用源自OMT Booch, Objectory及其他主要方法的表示法，即使它能够很好地支持设计工作，如封装、分块、记录模型构造思路。此外，我们希望 UML 能准确表达当前软件开发中的热点问题，比如大规模、分布、并发、方式和团体开发等。

UML并不试图成为一个完整的开发方法。它不包括一步一步的开发过程。我们认为一个好的软件开发过程对成功的开发软件是至关重要的，我们向读者推荐一本书 [Jacobson-99]。UML和使用UML的软件开发过程是两回事，这一点很重要。我们希望 UML可以支持所有的，至少是目前现有的大部分软件开发过程。 UML包含了所有的概念，我们认为这些概念对于支持基于一个健壮的构架来解决用例驱动的需求的迭代式开发过程是必要的。

UML的最终目标是在尽可能简单的同时能够对实际需要建立的系统的各个方面建模。UML需要有足够的表达能力以便可以处理现代软件系统中出现的所有概念，例如并发和分布，以及软件工程中使用的技巧，如封装和组件。它必须是一个通用语言，像任何一种通用程序设计语言一样。然而，这样就意味着 UML必将十分庞大，不可能像描述一个近乎于玩具一样的软件系统那样简单。现代语言和操作系统比起 40年前要复杂得多，因为我们对它们的要求越来越多。 UML提供了多种模型，不是在一天之内就能够掌握的。它比先前的建模语言更复杂，因为它更全面。但是不必一下就完全学会它，就像学习任何一种程序设计语言、操作系统或是复杂的应用软件一样。

1.4 UML概念域

UML的概念和模型可以分成以下几个概念域。

静态结构。任何一个精确的模型必须首先定义所涉及的范围，即确定有关应用、内部特性及其相互关系的关键概念。 UML的静态组件称为静态视图。静态视图用类构造模型来表达应用，每个类由一组包含信息和实现行为的离散对象组成。对象包含的信息被作为属性，它们执行的行为被作为操作。多个类通过泛化处理可以具有一些共同的结构。子类在继承它们共同的父类的结构和行为的基础上增加了新的结构和行为。对象与其他对象之间也具有运行时间连接，这种对象与对象之间的关系被称为类间的关联。一些元素通过依赖关系组织在一起，这些依赖关系包括在抽象级上进行模型转换、模板参数的捆绑、授予许可以及通过一种

China-pub.com 下载

元素使用另一种元素等。另一类关系包括用例和数据流的合并。静态视图主要使用类图。静态视图可用于生成程序中用到的大多数数据结构声明。在 UML视图中还要用到其他类型的元素，比如接口、数据类型、用例和信号等，这些元素统称为类元，它们的行为很像在每种类元上具有一定限制的类。

动态行为。有两种方式对行为建模。一种是根据一个对象与外界发生关系的生命历史；另一种是一系列相关对象之间当它们相互作用实现行为时的通信方式。孤立对象的视图是状态机 —— 当对象基于当前状态对事件产生反应，执行作为反应的一部分的动作，并从一种状态转换到另一种状态时的视图。状态机模型用状态图来描述。

相互作用对象的系统视图是一种协作，一种与语境有关的对象视图以及它们相互之间的链，通过数据链对象间存在着消息流。视图点将数据结构、控制流和数据流在一个视图中统一起来。协作和互操作用顺序图和协作图来描述。对所有行为视图起指导作用的是一组用例，每一个用例描述了一个用例参与者或系统外部用户可见的一个功能。

实现构造。UML模型既可用于逻辑分析又可用于物理实现。某些组件代表了实现项目。构件是系统中物理上的可替换的部分，它按照一组接口来设计并实现。它可以方便地被一个具有同样规格说明的构件替换。节点是运行时间计算资源，资源定义了一个位置。它包括构件和对象。部署图描述了在一个实际运行的系统中节点上的资源配置和构件的排列以及构件包括的对象，并包括节点间内容的可能迁移。

模型组织。计算机能够处理大型的单调的模型，但人力不行。对于一个大型系统，建模信息必须被划分成连贯的部分，以便工作小组能够同时工作在不同部分上。即使是一个小系统，人的理解能力也要求将整个模型的内容组织成一个个适当大小的包。包是 UML模型通用的层次组织单元，它们可以用于存储、访问控制、配置管理以及构造包含可重用的模型单元库。包之间的依赖关系是对包的组成部分之间的依赖关系的归纳。系统整个构架可以在包之间施加依赖关系。因此，包的内容必须符合包的依赖关系和有关的构架要求。

扩展机制。无论一种语言能够提供多么完善的机制，人们总是想扩展它的功能。我们已使UML具有一定的扩展能力，相信能够满足大多数对 UML扩充的需求而不改变语言的基础部分。构造型是一种新的模型元素，与现有的模型元素具有相同的结构，但是加上了一些附加限制，具有新的解释和图标。代码生成器和其他的工具对它的处理过程也发生了变化。标记值是一对任意的标记值字符串，能够被连接到任何一种模型元素上并代表任何信息，如项目管理信息、代码生成指示信息和构造型所需要的值。标记和值用字符串代表。约束是用某种特定语言（如程序设计语言、专用语言或自然语言）的文本字符串表达的条件。 UML提供了一个表达约束的语言，名为 OCL。与所有其他扩展机制一样，必须小心使用这些扩展机制，因为有可能形成一些别人无法理解的方言。但这些机制可以避免语言基础发生根本性变化。

1.5 表达式和图表语法

本书列举了许多演示实际模型的表达式和图表，以及表达式的语法和图表的注释。为了

China-pub.com
下载

尽量避免将解释说明和实例弄混，本书采用了一些约定的格式。

在图表和文本表达式中实际的表示法部分用 Helvetica字体印刷。例如，模型中出现的Helvetica字体的类名是一个合法的名称。语法表达式中的括弧是一个可能出现在实际表达式中的括弧，它不是实际语法机构的一部分。例如：Order.create(customer,amount)

在正文中，关键词和模型元素名都用Helvetica字体印刷，如：Order或Customer。

在一个语法表达式子中，句法单元名可以被实际的一段文字替代，这样的句法单元名用斜体印刷，如：*name*。斜体和下划线说明替换文本具有给定的性质。例如：

name. Operation(argument,..)

object-name:class

在语法表达式中，下标和上划线用于指示某种语法性质。例如：

$expression_{opt}$ 这个表达式是任选的。

$expression_{list}$ 用逗号来分隔一系列表达式。如果出现了零个或者一个重复符号，则不需要分隔符。每个重复符号都要用一个单独的替换符号。如果一个除逗号之外的标点符号出现在下标中，则它是分隔符。

$\underline{=expression_{opt}}$ 用上划线来连接两个或多个属于同一单元的可选的或重复出现的项目。在这个例子中，等号和表达式构成一个可以使用或省略的单元。如果只有一个项目，可以不用上划线。

在图表中，楷体和箭头是注释，它们是解释性说明而不是实际表示法的一部分。其他文字和符号是实际表示法的一部分。

China-pub.com
下载

第2章
模型的性质与用途

本章将解释什么是模型，模型有何用途以及如何使用模型。本章还将解释模型的不同层次：理想的，部分的和基于工具的。

2.1 什么是模型

模型是用某种工具对同类或其他工具的表达方式。模型从某一个建模观点出发，抓住事物最重要的方面而简化或忽略其他方面。工程、建筑和其他许多需要具有创造性的领域中都使用模型。

表达模型的工具要求便于使用。建筑模型可以是图纸上所绘的建筑图，也可以是用厚纸板制作的三维模型，还可以用存于计算机中的有限元方程来表示。一个建筑物的结构模型不仅能够展示这个建筑物的外观，还可以用它来进行工程设计和成本核算。

软件系统的模型用建模语言来表达，如 UML。模型包含语义信息和表示法，可以采取图形和文字等多种不同形式。建立模型的目的是因为在某些用途中模型使用起来比操纵实物更容易和方便。

2.2 模型的用途

模型有多种用途。

捕获和精确表达项目的需求和应用领域中的有关知识，以使各方面的利益相关者能够理解并达成一致。建筑物的各种模型能够准确表达出这个建筑物在外观、交通、服务设施、抗风和抗震性能、消费及其他需求。各方面的利益相关者则包括建筑设计师、建筑工程师、合同缔约人、各个子项目的缔约人、业主、出租者和市政当局。

软件系统的不同模型可以捕获关于这个软件的应用领域、使用方法、度量手段和构造模式等方面的需求信息。各方面的利益相关者包括软件结构设计师、系统分析员、程序员、项目经理、顾客、投资者、最终用户和使用软件的操作员。在 UML中要使用各种各样的模型。

进行系统设计。建筑设计师可以用画在图纸上的模型图、存于计算机中的模型或实际的

China-pub.com
下载

三维模型使自己的设计结果可视化，并用这些模型来做设计方面的试验。建造、修改一个小型模型比较简单，这使得设计人员不需花费什么代价就可以进行创造和革新。

在编写程序代码以前，软件系统的模型可以帮助软件开发人员方便地研究软件的多种构架和设计方案。在进行详细设计以前，一种好的建模语言可以让设计者对软件的构架有全面的认识。

使具体的设计细节与需求分开。建筑物的某种模型可以展示出符合顾客要求的外观，另一类模型可以说明建筑物内部的电气线路、管线和通风管道的设置情况。实现这些设置有多种方案。最后确定的建筑模型一定是建筑设计师认为最好的一个设计方案。顾客可以对此方案进行检查验证，但通常顾客对具体的设计细节并不关心，只要能满足他们的需要即可。

软件系统的一类模型可以说明这个系统的外部行为和系统中对应于真实世界的有关信息，另一类模型可以展示系统中的类以及实现系统外部行为特性所需要的内部操作。实现这些行为有多种方法。最后的设计结果对应的模型一定是设计者认为最好的一种。

生成有用的实际产品。建筑模型可以有多种相关产品，包括建筑材料清单、在各种风速下建筑物的偏斜度、建筑结构中各点的应力水平等。

利用软件系统的模型，可以获得类的声明、过程体、用户界面、数据库、合法使用的说明、配置草案以及与其他单位技术竞争情况的对比说明。

组织、查找、过滤、重获、检查以及编辑大型系统的有关信息。建筑模型用服务设施来组织信息：建筑结构、电器、管道、通风设施、装潢等等。除非利用计算机存储，否则对这些信息的查找和修改没那么容易。相反，如果整个模型和相关信息均存储在计算机中，则这些工作很容易进行，并且可方便地研究多种设计方案，这些设计方案共享一些公共信息。

软件系统用视图来组织信息：静态结构视图、状态机视图、交互视图、反映需求的视图等等。每一种视图均是针对某一目的从模型中挑选的一部分信息的映射。没有模型管理工具的支持不可能使模型做得任意精确。一个交互视图编辑工具可以用不同的格式表示信息，可以针对特定的目的隐藏暂时不需要的信息并在以后再展示出来，可以对操作进行分组、修改模型元素以及只用一个命令修改一组模型元素等等。

经济地研究多种设计过程中的解决方案。对同一建筑的不同设计方案的利弊在一开始可能不很清楚。例如，建筑物可以采用的不同的子结构彼此之间可能有复杂的相互影响，建筑工程师可能无法对这些做出正确的评价。在实际建造建筑物以前，利用模型可以同时研究多种设计方案并进行相应的成本和风险估算。

通过研究一个大型软件系统的模型可以提出多个实际方案并可以对它们进行相互比较。当然模型不可能做得足够精细，但即使一个粗糙的模型也能够说明在最终设计中所要解决的许多问题。利用模型可以研究多种设计方案，所花费的成本只是实现其中一种方案所花费的成本。

全面把握复杂的系统。一个关于龙卷风袭击建筑物的工程模型中的龙卷风不可能是真实

China-pub.com 下载

世界里的龙卷风，仅仅是模型而已。真正的龙卷风不可能呼之即来，并且它会摧毁测量工具。许多快速激烈的物理过程现在都可以运用这种物理模型来研究和理解。

一个大型软件系统由于其复杂程度可能无法直接研究，但模型使之成为可能。在不损失细节的情况下，模型可以抽象到一定的层次以使人们能够理解。可以利用计算机对模型进行复杂的分析以找出可能的“问题点”，如时间错误和资源竞争等。在对实物做出改动前，通过模型研究系统内各组成部分之间的依赖关系可以得出这种改动可能会带来哪些影响。

2.3 模型的层次

针对不同目的，模型可以采取各种形式及不同的抽象层次。模型中所包含的信息量必须对应于以下几种目的。

指导设计思路。在项目早期所建立的高层模型用于集中利益相关者的思路和强调一些重要的选择方案。这些模型描述了系统的需求并代表了整个系统设计工作的起点。早期的模型帮助项目发起者在把精力放在系统的细节问题之前研究项目可能的选择方案。随着设计工作的进展，早期模型被更为精确的模型所替代。没有必要详细保存早期研究过程中的种种选择方案和返工情况。早期模型的目的是帮助获得思路，但最后得到的“思路模型”要在进行详细设计前记录下来。早期模型不需要达到实现阶段的模型的精确程度，无须涉及有关系统实现的一套概念。建立这种模型只使用了 UML定义的组件的一个子集，比后期的设计阶段的模型使用的组件要少得多。

当早期模型发展到具有一定精度的完整的视图模型时 —— 例如，分析系统需求的模型 —— 那么要在开发过程进入下一阶段时将其保存下来。不断向模型中填加信息的增量式开发（在这种情况下开发的推理过程也要保存和记录）与一般的针对“死端点”进行研究直到得出正确的解决方案的随意漫步式开发之间存在一个重要的区别。后一种情况通常使人不知怎么着手，并且根本没有必要对整个开发过程进行记录保存，除非遇到特殊情况需要对开发过程进行回溯。

系统基本结构的抽象说明。在分析阶段和初步设计阶段所建立的模型以关键概念和最终系统的各种机制为中心。这些模型以某种方式与最终系统匹配，但是模型中丧失了细节信息，在设计过程中必需显式地补充这些信息。使用抽象模型的目的是在处理局部细节问题前纠正系统高层次的普遍问题。通过一个仔细的开发过程，这些模型可以发展成最终模型，该过程保证最终获得的模型能够正确实现初期模型的设计意图。必须具备跟踪能力来跟踪从基本模型到完备模型的过程，否则无法保证最终系统正确包含了基本模型所要表达的关键特性。基本模型强调语义，它们不需要涉及系统实现方面的细节。有时确实会出现这种情况：在低层实现方面的差别会使逻辑语义模糊不清。从基本模型到最后的实现模型的途径必须清晰和简明，不论这个过程由代码生成器自动实现还是由设计者人工实现。

最终系统的详细规格说明。系统实现模型包含能够建造这个系统的足够信息，它不仅要

China-pub.com
下载

包括系统的逻辑语义和语法、算法、数据结构和确保能正确完成系统功能的机制，而且还要包括系统制品的组织决定，这些制品对个人之间的相互协作和使用辅助工具来说十分必要。这种模型必须包括对模型元素进行打包的组件以便于理解和用计算机进行自动处理。这不是目标应用系统的特性，而是系统构造过程应具有的特性。

典型或可能的系统范例。精心挑选的实例可以提高人们的观察能力并使系统的说明和实现有实际效果。然而，即使有非常多的例子，也起不到一段详细定义所起的效果。我们最终希望的是要让模型能够说明一般的情形，这也是程序代码所要做的事情。不过典型的数据结构、交互顺序或对象生命历程的例子对于理解复杂系统很有益处。必须小心使用例子。从逻辑上来说，从一大堆特例中归纳出最一般的情况是不可能的，但是大部分人思考某一问题时总是首先考虑一些被精心挑选出来的有关该问题的例子。范例模型仅是对模型的示例而不带有一般性的描述，因此，人们会觉得这两种模型之间有差异。范例模型一般只使用 UML定义的组件的子集。说明型模型和范例模型在建模中都很有用。

对系统全面的或部分的描述。模型可以完全描述一个独立系统，并且不需要参考外部信息。更通常的情况是，模型是用相互区别的、不连续的描述单元组织起来的，每个单元作为整体描述的一部分可以被单独进行存储和操纵。这种模型带有必须与系统其他模型联系在一起的“散件”。因为这些散件具有相关性和含义，因此它们能够与其他散件通过各种方式结合来构造不同的系统。获得重用是一个好的建模方法的重要目标。

模型要随时间发展变化。深度细化的模型源于较为抽象的模型，具体模型源于逻辑模型。例如，开始阶段建立的模型是整个系统的一个高层视图。随着时间的推进，模型中增加了一些细节内容并引入了一些变化。再随着时间的推进，模型的核心焦点从一个以用户为中心的前端逻辑视图转变成了一个以实现为中心的后端物理视图。随着开发过程的进行和对系统的深入理解，必须在各种层次上反复说明模型以获得更好的理解，用一个单一视角或线性过程理解一个大型系统是不可能的。对模型的形式无所谓“正确和错误”之分。

2.4 模型的内容

语义和表示法。模型包含两个主要方面：语义方面的信息（语义）和可视化的表达方法（表示法）。

语义方面用一套逻辑组件表达应用系统的含义，如类、关联、状态、用例和消息。语义模型元素携带了模型的含义即，它们表达了语义。语义建模元素用于代码生成、有效性验证、复杂度的度量等，其可视化的外观与大多数处理模型的工具无关。语义信息通常被称作模型。一个语义模型具有一个词法结构、一套高度形式化的规则和动态执行结构。这些方面通常分别加以描述（定义UML的文献即如此），但它们紧密相关，并且是同一模型的一部分。

可视化的表达方式以可使人观察、浏览和编辑的形式展示语义信息。表示方式元素携带了模型的可视化表达方式，即语义是用一种可被人直接理解的方式来表达的。它们并未增添

China-pub.com 下载

新的语义，但用一种有用的方式对表达方式加以组织，以强调模型的排例。因此它们对模型的理解起指导作用。表达方式元素的语义来自于语义模型元素。但是，由于是由人来绘制模型图，所以表达方式元素并不是完全来自于模型的逻辑元素。表达方式元素的排列可能会表达出语义关系的另外含义，这些语义关系很不明显或模棱两可，以至于在模型中不能形式化地表达，但可给人启迪。

语境。模型自身是一个计算机系统的制品，被应用在一个给出了模型含义的大型语境中。这个语境包括模型的内部组织、整个开发过程中对每个模型的注释说明、一个缺省值集合、创建和操纵模型的假定条件以及模型与其所处环境之间的关系。

模型需要有内部组织，允许多个工作小组同时使用某个模型而不发生过多的相互牵涉。这种对模型的分解并不是语义方面所要求的 —— 与一个被分解成意义前后连贯的多个包的模型相比，一个大的单块结构的模型所表达的信息可能会同样精确，因为组织单元的边界确定会使准确定义语义的工作复杂化，故这种单块模型表达的信息可能比包结构的模型表达得更精确。但是要想有效地工作于一个大的单块模型上的多个工作组不彼此相互妨害是不可能的。其次，单块模型没有适用于其他情况的可重用的单元。最后，对大模型的某些修改往往会引起意想不到的后果。如果模型被适当分解成具有良好接口的小的子系统，那么对其中一个小的、独立的单元所进行的修改所造成的后果可以跟踪确定。不管怎样，将大系统分解成由精心挑选的单元所构成的层次组织结构，是人类千百年来所发明的设计大系统的方法中最可靠的方法。

模型捕获一个应用系统的语义信息，但还需记录模型自身开发过程中的各种信息，如某个类的设计者、过程的调试状态和对各类人员的使用权限的规定。这些信息至多是系统语义的外围信息，但对开发过程非常重要。因此，建立一个系统的模型必须综合考虑两方面。最简便的实现方法是将项目管理信息作为注释加入到语义模型中，即可以对模型元素用非建模语言进行任意描述。在UML中用文本字符串来表示注释。

文本编辑器或浏览器所使用的命令不是程序设计语言的一部分，同样，用于创建和修改模型的命令也不是建模语言语义的一部分。模型元素的属性没有缺省值，在一个特定的模型中，它们均具有值。然而，对于实际的开发过程，人们要求建立与修改模型时无须详细说明有关的所有细节。缺省值存在于建模语言和支持这种语言的建模工具的边界处。在建模工具所使用的创建模型的命令中，它们是真正的缺省值，尽管它们可能会超过单个的建模工具并像用户所期望的那样成为建模工具所使用的通用语言。

模型不是被孤立地建造和使用的。它们是模型所处的大环境中的一部分，这个大环境包括建模工具、建模语言和语言编译器、操作系统、计算机网络环境、系统具体实现方面的限制条件等等。系统信息应该包括环境所有方面的信息。系统信息的一部分应被保存在模型中，即使这些信息不是语义信息，如项目管理注释（在上面已经讨论过）、代码生成提示、模型的打包、编辑工具缺省命令的设置。其他方面的信息应分别保存，如程序源代码和操作系统配置命令。即使是模型中的信息，对这些信息的解释也可以位于多个不同地方，包括建模语言、建模工具、代码生成器、编译器或命令语言等等。本书用 UML自身对模型进行解释。但是当

China-pub.com
下载

进行系统的具体物理实现时，要用到其他用于解释的资源，这些资源对 UML来说是不可见的。

2.5 模型的含义

模型是一个系统潜在配置的发生器，系统是它的外延或值。按照模型来进行系统配置是一种理想化的情况。然而，有时模型所要求的种种条件在现实中无法实现。模型还是对系统含义和结构的一般性说明。这种描述是模型的内涵或含义。模型总是具有一定的抽象层次，它包含了系统中的最基本的成分而忽略了其他内容。对于模型，以下几方面需要考虑。

抽象与具体。模型包含系统的基本成分而忽略了其他内容。哪些是基本内容需要根据建模的目的来判定。这并不是说要把模型所含的信息一分为二，可能会存在一系列精度递增的模型。建模语言不是程序设计语言，它所表达的内容可能很不精确，因为多余的详细内容与建模目的无关。具有不同精度级别的模型可应用于同一个项目的各个阶段。用于程序设计编码的模型要求必须与程序设计语言有关。典型的情况是，在早期分析阶段使用高层次的、表达精度低的模型。随着开发过程的深入，所用的模型越来越细化，最终所使用的模型包含了大量的细节内容，具有很高的精度。

说明与实现。模型可以告诉我们“做什么”（说明），也可以告诉我们“功能是如何实现的”（实现）。建模时要注意区分这两方面。在花大量时间研究怎么做之前很重要的一点是要正确分析出系统要做什么。建模的一个重要的方面是对具体实现细节进行抽象。在说明与实现之间可能有一系列相关关系，其中某层次中的说明就是对它上一层次的实现。

描述与示例。模型主要是描述性的。模型所描述的是一个个实例，这些实例仅作为例子出现在模型中。大部分实例仅在运行的一部分时间中才存在。有时，运行实例自身是对其他事物的描述。我们把这些混杂对象称做元数据。更深一层次地看，认为任何事物不是描述性的就是实例性的，这种观点是不符合实际情况的。某一事物是实例还是描述不是孤立区分的，与观察角度有关，大部分事物都可以用多种角度来考察。

解释的变更。在一种建模语言中模型可能会有多种解释。可以定义语义变更点 —— 可能会出现多种解释的地方 —— 给每一个解释一个语义变更名，以便可以标识究竟使用了哪种解释。例如， Smalltalk语言的使用者要避免在系统实现模型中出现多重继承关系，因为Smalltalk语言不支持多重继承。而其他程序设计语言使用者可能会在模型中使用多重继承。语义变更点支持多种具体的实现过程。

China-pub.com
下载

第二部分

基本概念

这一部分包括了UML中使用的各概念的综述，以说明在系统建模中如何综合运用这些概念。本部分不详细说明每一个概念，其详细说明可参见本书的第三部分。

China-pub.com
下载

第3章
UML 初览

本章使用一个简单的例子对UML中所使用的概念和视图进行初览。本章的目的是要将高层UML概念组织成一系列较小的视图和图表来可视化说明这些概念，说明如何用各种不同的概念来描述一个系统以及如何将各种视图组织在一起。概括性的说明不可能面面俱到，其中省略了许多概念。要想得到更详细的说明，可参见下一章对UML各视图的说明和本书大全部分的有关细节。

本章使用的例子是计算机管理的戏院售票系统。这是一个精心设计的例子，目的是用少量篇幅来强调说明UML的各个组件。这是一个经过有意简化的例子，忽略了有关细节。除非进行大量的反复说明，否则一个实际系统的完整模型不可能用这么少的篇幅来对 UML中使用的每种组件进行介绍。

3.1 UML视图

UML中的各种组件和概念之间没有明显的划分界限，但为方便起见，我们用视图来划分这些概念和组件。视图只是表达系统某一方面特征的 UML建模组件的子集。视图的划分带有一定的随意性，但我们希望这种看法仅仅是直觉上的。在每一类视图中使用一种或两种特定的图来可视化地表示视图中的各种概念。

在最上一层，视图被划分成三个视图域：结构分类、动态行为和模型管理。

结构分类描述了系统中的结构成员及其相互关系。类元包括类、用例、构件和节点。类元为研究系统动态行为奠定了基础。类元视图包括静态视图、用例视图和实现视图。

动态行为描述了系统随时间变化的行为。行为用从静态视图中抽取的系统的瞬间值的变化来描述。动态行为视图包括状态机视图、活动视图和交互视图。

模型管理说明了模型的分层组织结构。包是模型的基本组织单元。特殊的包还包括模型和子系统。模型管理视图跨越了其他视图并根据系统开发和配置组织这些视图。

UML还包括多种具有扩展能力的组件，这些扩展能力有限但很有用。这些组件包括约束、构造型和标记值，它们适用于所有的视图元素。

China-pub.com 下载

表3-1列出了UML的视图和视图所包括的图以及与每种图有关的主要概念。不能把这张表看成是一套死板的规则，应将其视为对UML常规使用方法的指导，因为UML允许使用混合视图。

表3-1 UML视图和图

主要的域	视图	图	主要概念
结构	静态视图	类图	类、关联、泛化、依赖关系、实现、接口
	用例视图	用例图	用例、参与者、关联、扩展、包括、用例泛化
	实现视图	构件图	构件、接口、依赖关系、实现
	部署视图	部署图	节点、构件、依赖关系、位置
动态	状态机视图	状态机图	状态、事件、转换、动作
	活动视图	活动图	状态、活动、完成转换、分叉、结合
	交互视图	顺序图	交互、对象、消息、激活
		协作图	协作、交互、协作角色、消息
模型管理	模型管理视图	类图	包、子系统、模型
可扩展性	所有	所有	约束、构造型、标记值

3.2 静态视图

静态视图对应用领域中的概念以及与系统实现有关的内部概念建模。这种视图之所以被称之为是静态的是因为它不描述与时间有关的系统行为，此种行为在其他视图中进行描述。静态视图主要是由类及类间相互关系构成，这些相互关系包括：关联、泛化和各种依赖关系，如使用和实现关系。类是应用领域或应用解决方案中概念的描述。类图是以类为中心来组织的，类图中的其他元素或属于某个类或与类相关联。静态视图用类图来实现，正因为它以类为中心，所以称其为类图。

在类图中类用矩形框来表示，它的属性和操作分别列在分格中。如不需要表达详细信息时，分格可以省略。一个类可能出现在好几个图中。同一个类的属性和操作可只在一种图中列出，在其他图中可省略。

关系用类框之间的连线来表示，不同的关系用连线上和连线端头处的修饰符来区别。

图3-1是售票系统的类图，它只是售票系统领域模型的一部分。图中表示了几个重要的类，如Customer、Reservation、Ticket和Performance。一个顾客可多次订票，但每一次订票只能由一个顾客来执行。有两种订票方式：个人票或套票，前者只是一张票，后者包括多张票。每一张票不是个人票就是套票中的一张，但是不能又是个人票又是套票中的一张。每场演出都有多张票可供预定，每张票对应一个唯一的座位号。每次演出用剧目名、日期和时间来标识。

类可用不同的精确度和抽象级别来描述。在设计的初期阶段，所建立的模型只是问题的逻辑模型，到了设计的后期，模型中会增添许多设计结论和有关系统实现的细节。 UML中的大部分视图都可不断进行类似的细化。

China-pub.com
下载

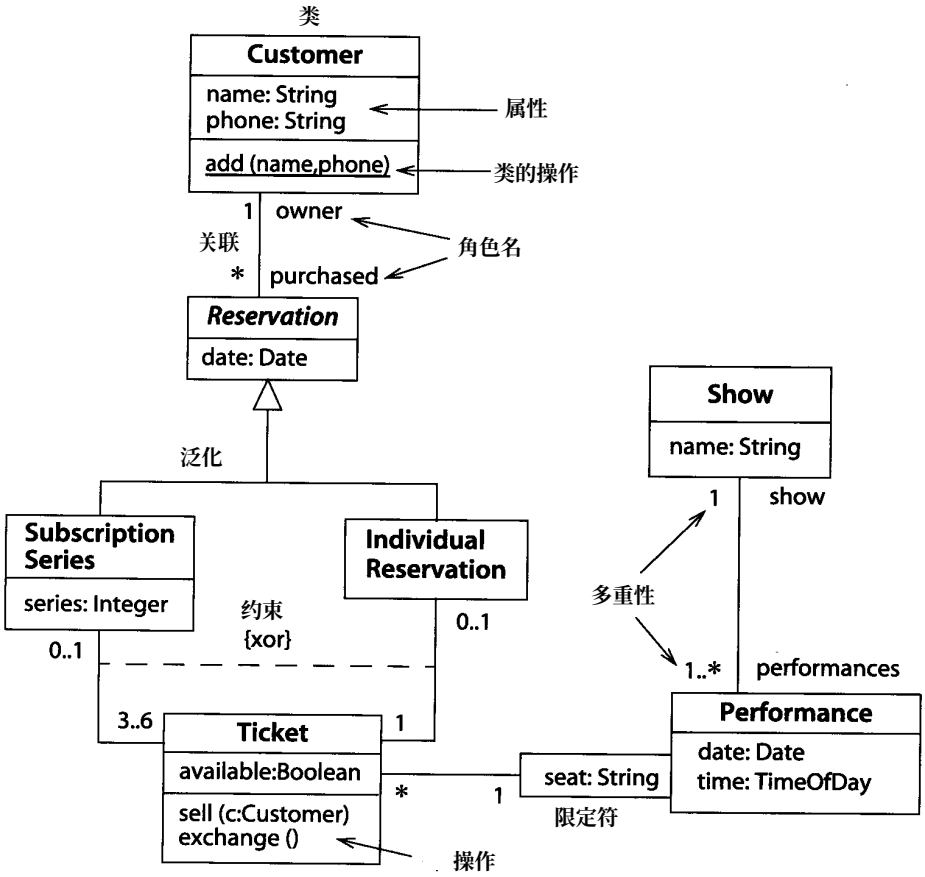


图3-1 类图

3.3 用例视图

用例视图是被称为参与者的外部用户所能观察到的系统功能的模型图。用例是系统中的一个功能单元，可以被描述为参与者与系统之间的一次交互作用。用例模型的用途是列出系统中的用例和参与者，并显示哪个参与者参与了哪个用例的执行。

图3-2是售票系统的用例图。参与者包括售票员、监督员和公用电话亭。公用电话亭是另一个系统，它接受顾客的订票请求。在售票处的应用模型中，顾客不是参与者，因为顾客不直接与售票处打交道。用例包括通过公用电话亭或售票员购票，购预约票（只能通过售票员）以及售票监督（应监督员的要求）。购票和购预约票包括一个共同的部分——即通过信用卡来付钱。（对售票系统的完整描述还要包括其他一些用例，例如换票和验票等。）

用例也可以有不同的层次。用例可以用其他更简单的用例进行说明。在交互视图中，用

China-pub.com 下载

例做为交互图中的一次协作来实现。

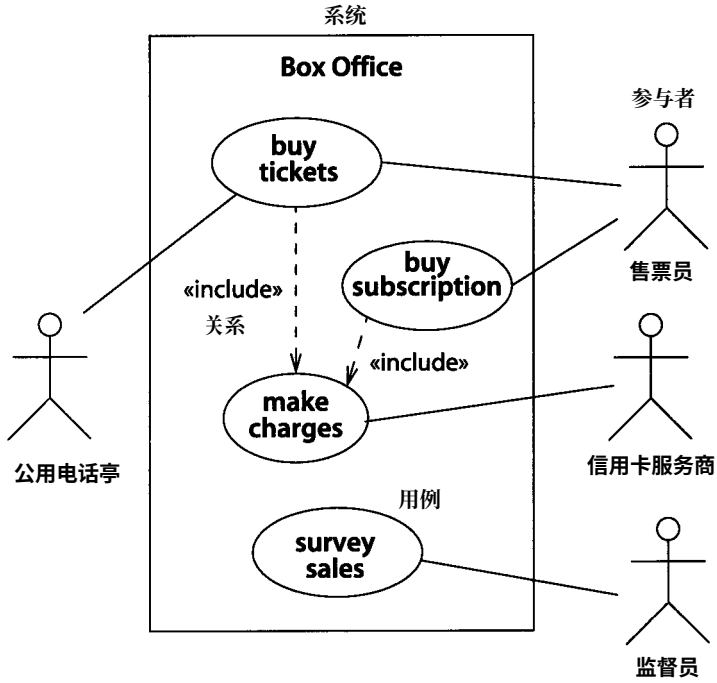


图3-2 用例图

3.4 交互视图

交互视图描述了执行系统功能的各个角色之间相互传递消息的顺序关系。类元是对在系统内交互关系中起特定作用的一个对象的描述，这使它区别于同类的其他对象。交互视图显示了跨越多个对象的系统控制流程。交互视图可用两种图来表示：顺序图和协作图，它们各有不同的侧重点。

3.4.1 顺序图

顺序图表示了对象之间传送消息的时间顺序。每一个类元角色用一条生命线来表示——即用垂直线代表整个交互过程中对象的生命期。生命线之间的箭头连线代表消息。顺序图可以用来进行一个场景说明——即一个事务的历史过程。

顺序图的一个用途是用来表示用例中的行为顺序。当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件。

图3-3是描述购票这个用例的顺序图。顾客在公用电话亭与售票处通话触发了这个用例的

China-pub.com
下载

执行。顺序图中付款这个用例包括售票处与公用电话亭和信用卡服务处的两个通信过程。这个顺序图用于系统开发初期，未包括完整的与用户之间的接口信息。例如，座位是怎样排列的；对各类座位的详细说明都还没有确定。尽管如此，交互过程中最基本的通信已经在这个用例的顺序图中表达出来了。

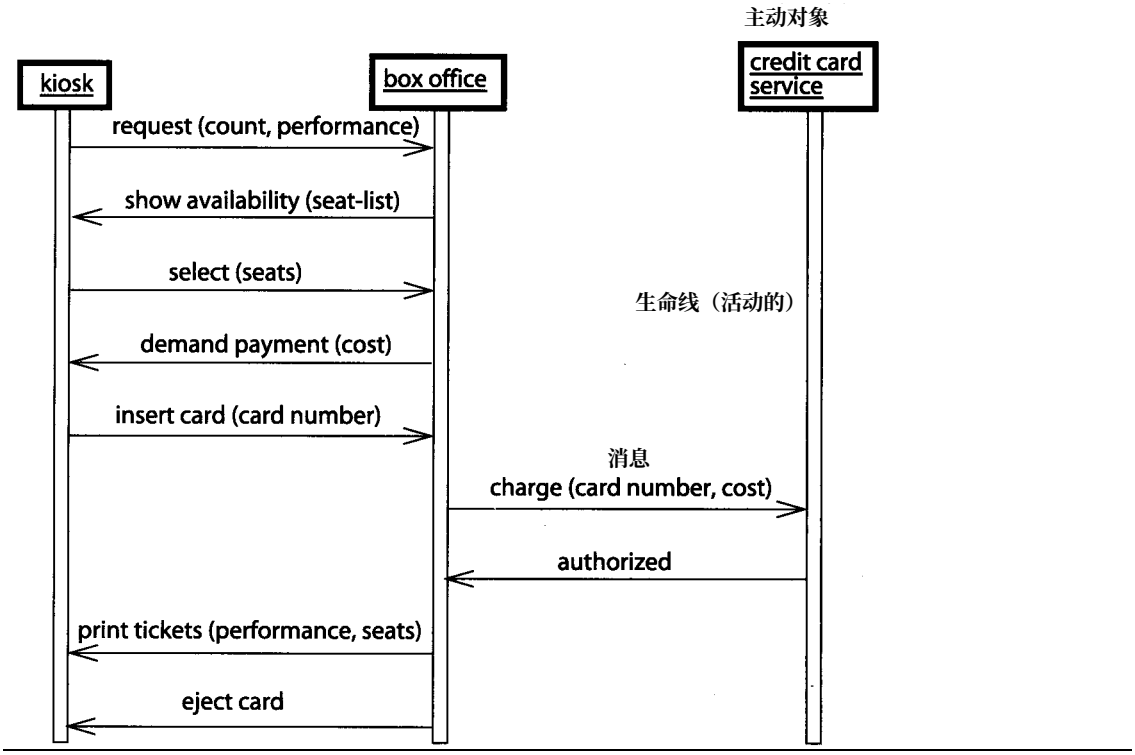


图3-3 顺序图

3.4.2 协作图

协作图对在一次交互中有意义的对象和对象间的链建模。对象和关系只有在交互的语境中才有意义。类元角色描述了一个对象，关联角色描述了协作关系中的一个链。协作图用几何排列来表示交互作用中的各角色（如图 3-4）。附在类元角色上的箭头代表消息。消息的发生顺序用消息箭头处的编号来说明。

协作图的一个用途是表示一个类操作的实现。协作图可以说明类操作中用到的参数和局部变量以及操作中的永久链。当实现一个行为时，消息编号对应了程序中嵌套调用结构和信号传递过程。

图3-4是开发过程后期订票交互的协作图。这个图表示了订票涉及的各个对象间的交互关

China-pub.com 下载

系。请求从公用电话亭发出，要求从所有的演出中查找某次演出的资料。返回给 ticketseller 对象的指针db代表了与某次演出资料的局部暂时链接，这个链接在交互过程中保持，交互结束时丢弃。售票方准备了许多演出的票；顾客在各种价位做一次选择，锁定所选座位，售票员将顾客的选择返回给公用电话亭。当顾客在座位表中做出选择后，所选座位被声明，其余座位解锁。

顺序图和协作图都可以表示各对象间的交互关系，但它们的侧重点不同。顺序图用消息的几何排列关系来表达消息的时间顺序，各角色之间的相关关系是隐含的。协作图用各个角色的几何排列图形来表示角色之间的关系，并用消息来说明这些关系。在实际中可以根据需要选用这两种图。

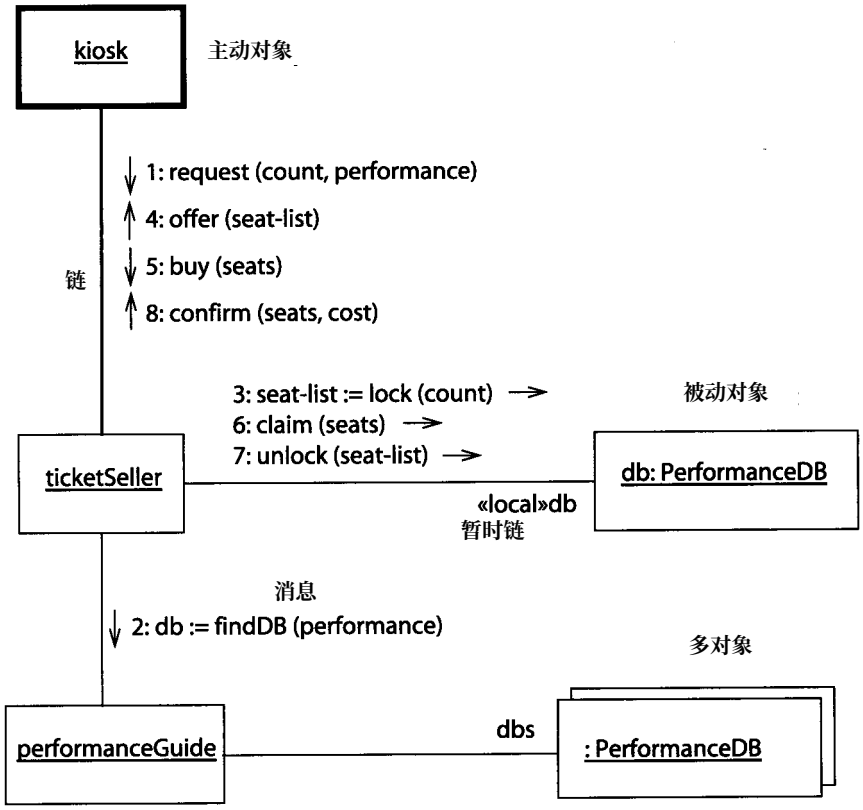


图3-4 协作图

3.5 状态机视图

状态机视图是一个类对象所可能经历的所有历程的模型图。状态机由对象的各个状态和

China-pub.com
下载

连接这些状态的转换组成。每个状态对一个对象在其生命期中满足某种条件的一个时间段建模。当一个事件发生时，它会触发状态间的转换，导致对象从一种状态转化到另一新的状态。与转换相关的活动执行时，转换也同时发生。状态机用状态图来表达。

图3-5是票这一对象的状态图。初始状态是 Available状态。在票开始对外出售前，一部分票是给预约者预留的。当顾客预定个人票时，被预定的票首先处于锁定状态，此时顾客仍有是否确实要购买这张票的选择权，故这张票可能出售给顾客也可能因为顾客不要这张票而解除锁定状态。如果超过了指定的期限顾客仍未做出选择，此票将被自动解除锁定状态。预约者也可以换其他演出的票，如果这样的话，最初预约票也可以对外出售。

状态图可用于描述用户接口、设备控制器和其他具有反馈的子系统。它还可用于描述在生命期中跨越多个不同性质阶段的被动对象的行为，在每一阶段该对象都有自己特殊的行为。

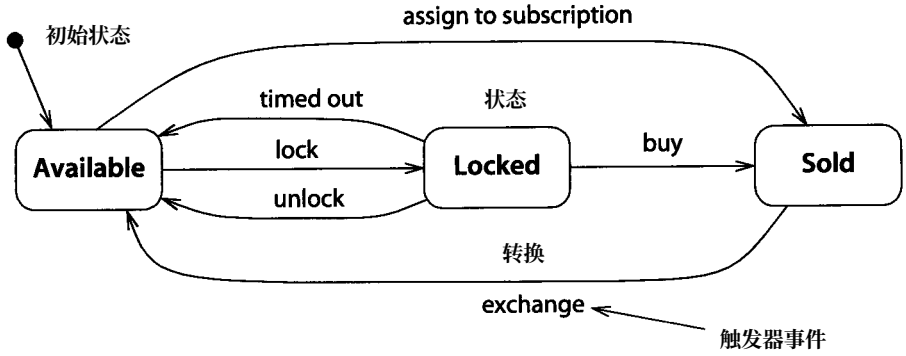


图3-5 状态图

3.6 活动视图

活动图是状态机的一个变体，用来描述执行算法的工作流程中涉及的活动。活动状态代表了一个活动：一个工作流步骤或一个操作的执行。活动图描述了一组顺序的或并发的活动。活动视图用活动图来体现。

图3-6是售票处的活动图。它表示了上演一个剧目所要进行的活动（这个例子仅供参考，不必太认真地凭着看戏的经验而把问题复杂化）。箭头说明活动间的顺序依赖关系 —— 例如，在规划进度前，首先要选择演出的剧目。加粗的横线段表示分叉和结合控制。例如，安排好整个剧目的进度后，可以进行宣传报道、购买剧本、雇用演员、准备道具、设计照明、加工戏服等，所有这些活动都可同时进行。在进行彩排之前，剧本和演员必须已经具备。

这个例子说明活动图的用途是对人类组织的现实世界中的工作流程建模。对事物建模是活动图的主要用途，但活动图也可对软件系统中的活动建模。活动图有助于理解系统高层活动的执行行为，而不涉及建立协作图所必须的消息传送细节。

China-pub.com 下载

用连接活动和对象流状态的关系流表示活动所需的输入输出参数。

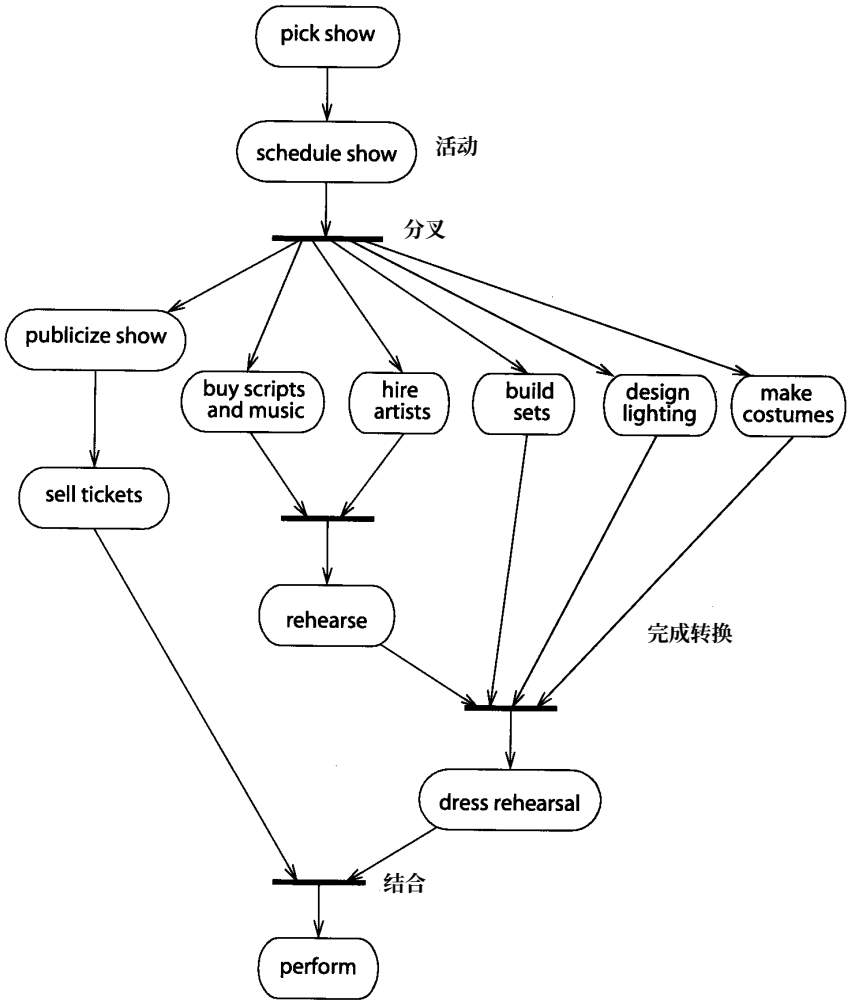


图3-6 活动图

3.7 物理视图

前面介绍的视图模型按照逻辑观点对应用领域中的概念建模。物理视图对应用自身的实现结构建模，例如系统的构件组织和建立在运行节点上的配置。这类视图提供了将系统中的类映射成物理构件和节点的机制。物理视图有两种：实现视图和部署视图。

实现视图为系统的构件建模——构件即构造应用的软件单元——还包括各构件之间的依

China-pub.com
下载

赖关系，以便通过这些依赖关系来估计对系统构件的修改给系统可能带来的影响。

实现视图用构件图来表现。图 3-7是售票系统的构件图。图中有三个用户接口：顾客和公用电话亭之间的接口、售票员与在线订票系统之间的接口和监督员查询售票情况的接口。售票方构件顺序接受来自售票员和公用电话亭的请求；信用卡主管构件用于处理信用卡付款；还有一个存储票信息的数据库构件。构件图表示了系统中的各种构件。在个别系统的实际物理配置中，可能有某个构件的多个备份。

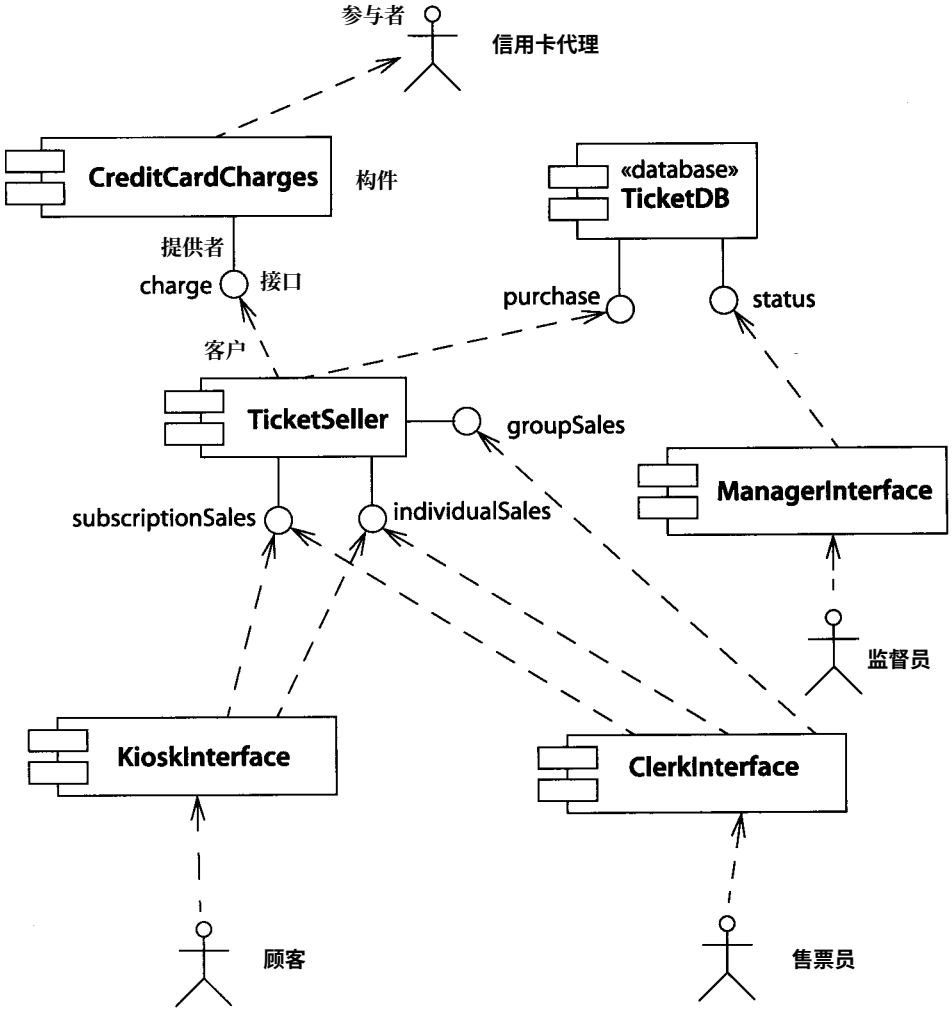


图3-7 构件图

图中的小圆圈代表接口，即服务的连贯集。从构件到接口的实线表明该构件提供的列在接口旁的服务。从构件到接口的虚线箭头说明这个构件要求接口提供的服务。例如，购买个

China-pub.com 下载

人票可以通过公用电话亭订购也可直接向售票员购买，但购买团体票只能通过售票员。

部署视图描述位于节点实例上的运行构件实例的安排。节点是一组运行资源，如计算机、设备或存储器。这个视图允许评估分配结果和资源分配。

部署视图用部署图来表达。图 3-8是售票系统的描述层部署图。图中表示了系统中的各构件和每个节点包含的构件。节点用立方体图形表示。

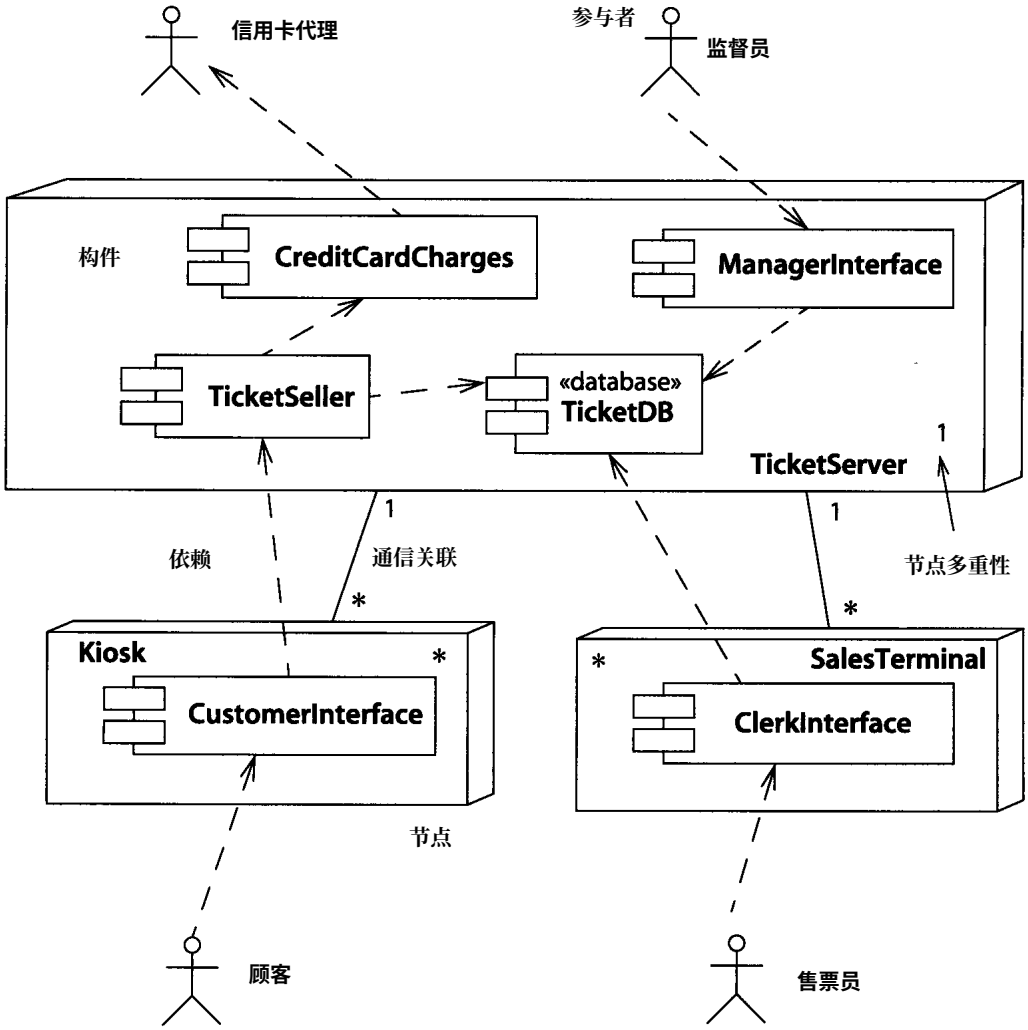


图3-8 部署图（描述层）

图3-9是售票系统的实例层部署图。图中显示了各节点和它们之间的连接。这个模型中的信息是与图3-8的描述层中的内容相互对应的。

China-pub.com
下载

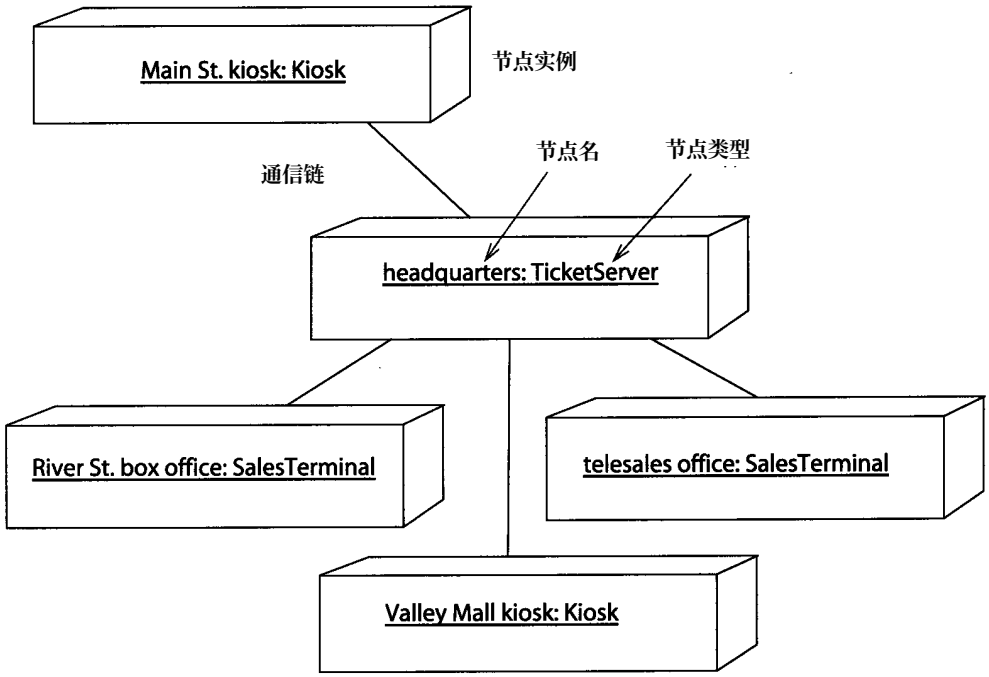


图3-9 部署图（实例层）

3.8 模型管理视图

模型管理视图对模型自身组织建模。一系列由模型元素（如类、状态机和用例）构成的包组成了模型。包可能包含其他的包，因此，整个模型实际上可看成一个根包，它间接包含了模型中的所有内容。包是操作模型内容、存取控制和配置控制的基本单元。每一个模型元素包含于包中或包含于其他模型元素中。

模型是从某一观点以一定的精确程度对系统所进行的完整描述。从不同的视角出发，对同一系统可能会建立多个模型，例如有系统分析模型和系统设计模型之分。模型是一种特殊的包。

子系统是另一种特殊的包。它代表了系统的一个部分，它有清晰的接口，这个接口可作为一个单独的构件来实现。

模型管理信息通常在类图中表达。

图3-10显示了将整个剧院系统分解所得到的包和它们之间的依赖关系。售票处子系统在前面的例子中已经讨论过了，完整的系统还包括剧院管理和计划子系统。每个子系统还包含了多个包。

China-pub.com 下载

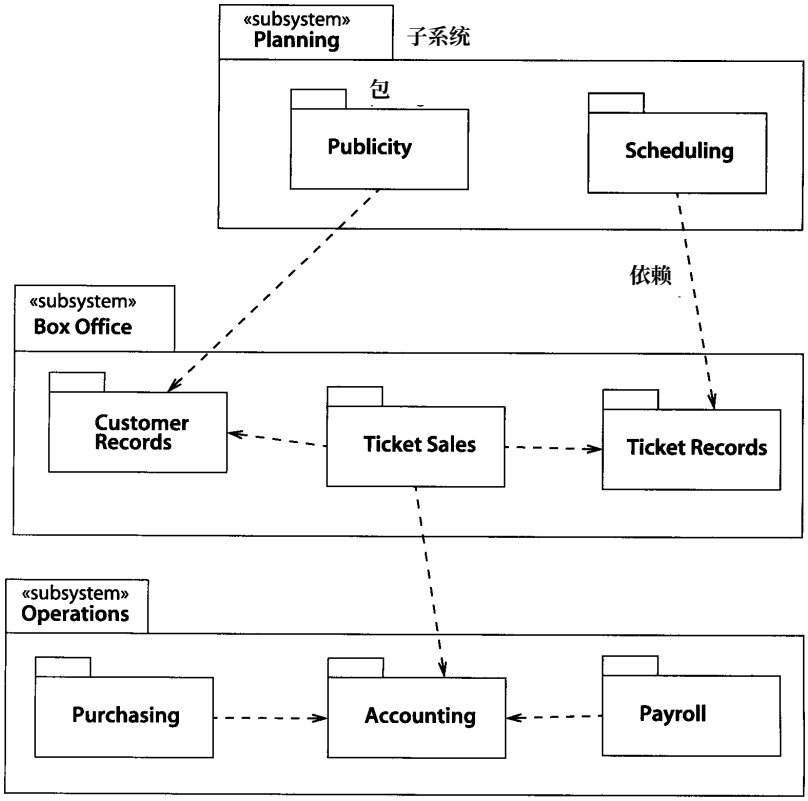


图3-10 包

3.9 扩展组件

UML包含三种主要的扩展组件：约束、构造型和标记值。约束是用某种形式化语言或自然语言表达的语义关系的文字说明。构造型是由建模者设计的新的模型元素，但是这个模型元素的设计要建立在UML已定义的模型元素基础上。标记值是附加到任何模型元素上的命名的信息块。

这些组件提供了扩展UML模型元素语义的方法，同时不改变UML定义的元模型自身的语义。使用这些扩展组件可以组建适用于某一具体应用领域的UML用户定制版本。

图3-11举例说明了约束、构造型和标记值的使用。对剧目类的约束保证了剧目具有唯一的名称。图3-1说明了两个关联的异或约束，一个对象某一时刻只能具有两个关联中的一个。用文字表达约束效果较好，但UML的概念不直接支持文字描述。

TicketdDB构件构造型表明这个是一个数据库构件，允许省略该构件的接口说明，因为这个接口是所有数据库都支持的通用接口。建模者可以增加新的构造型来表示专门的模型元素。一个构造型可以带有多个约束、标记值或者代码生成特性。如图所示，建模者可以为命名的构造型定义一个图标，作为可视化的辅助工具。尽管如此，可以使用文字形式说明。

China-pub.com
下载

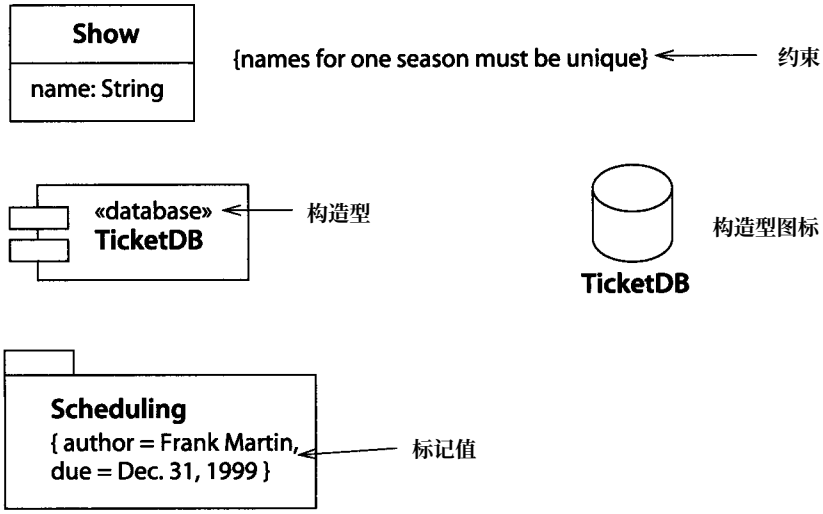


图3-11 扩展组件

Scheduling包中的标记值说明Frank Martin要在年底前完成计划的制定。可以将任意信息作为标记值写于一个模型元素中建模者选定的名字之下。使用文字有益于描述项目管理和代码生成参数。大部分标记值被保存为编辑工具中的弹出信息，在正式打印出的图表中通常没有标记值。

3.10 各种视图间的关系

多个视图共存于一个模型中，它们的元素之间有很多关系，其中一些关系列在表 3-2中。表中没有将各种关系列全，但它列出了从不同视角观察得到的元素间的部分主要关系。

表3-2 不同视图元素间的部分关系

元 素	元 素	关 系
类	状态机	拥有
操作	交互	实现
用例	协作	实现
用例	交互实例	样本场景
构件实例	节点实例	位置
动作	操作	调用
动作	信号	发送
活动	操作	调用
消息	动作	激发
包	类	拥有
角色	类	分类

China-pub.com
下载

第4章
静 态 视 图

4.1 概述

静态视图是UML的基础。模型中静态视图的元素是应用中有意义的概念，这些概念包括真实世界中的概念、抽象的概念、实现方面的概念和计算机领域的概念，即系统中的各种概念。举个例子，一个剧院的售票系统有各种概念，如票、预订、预约计划、座位分配规则、网络订票和冗余信息等。

静态视图说明了对象的结构。一个面向对象的系统使数据结构和行为特征统一到一个独立的对象结构中。静态视图包括所有的传统数据结构思想，同时也包括了数据操作的组织。数据和操作都可量化为类。根据面向对象的观点，数据和行为是紧密相关的。比如， Ticket对象可以携带数据，如价格、演出日期、座位号，该对象还可以有基于它的操作，例如：预留这张票或以一定折扣计算它的价格。

静态视图将行为实体描述成离散的模型元素，但是不包括它们动态行为的细节。静态视图将这些行为实体看作是将被类所指定、拥有并使用的物体。这些实体的动态行为由描述它们内部行为细节的其他视图来描述，包括交互视图和状态机视图。动态视图要求静态视图描述动态交互的事物 —— 如果不首先说清楚什么是交互作用，就无法说清楚交互作用怎样进行的。静态视图是建立其他视图的基础。

静态视图中的关键元素是类元及它们之间的关系。类元是描述事物的建模元素。有几种类元，包括类、接口和数据类型。包括用例和信号在内的其他类元具体化了行为方面的事物。实现目的位于像子系统、构件和节点这几种类元之后。

为了利于理解和模型的可重用性，大的模型必须由较小的单元组成。包是拥有和管理模型内容的一般的组织单元。任何元素都可被包所拥有。模型是用来描述完整的系统视图的包，并且使用时或多或少地独立于其他的模型 —— 这是掌握描述系统的更细节的包的基础。

对象是从建模者理解和构造的系统中分离出来的离散单元。它是类的实例 —— 即，它是一个其结构和行为都由类来描述的具有身份的个体。对象是一个可识别的状态，该状态的行为能被激发。

类元之间的关系有关联、泛化及各种不同的依赖关系，包括实现和使用关系。

China-pub.com
下载

4.2 类元

类元是模型中的离散概念，拥有身份、状态、行为和关系。有几种类元包括类、接口和数据类型。其他几种类元是行为概念、环境事物、执行结构的具体化。这些类元中包括用例、参与者、构件、节点和子系统。表 4-1列出了几种类元和它们的功能。元模型术语类元中包括了所有这些概念，因为类是我们所最熟悉的术语，所以先讨论它，再根据类与其他概念的区别来定义其他概念。

表4-1 各种类元

类 元	功 能	表示法
参与者	系统的外部用户	
类	模型系统中的概念	Name
状态类	局限于某个给定状态的类	Name[S]
类元角色	在合作中局限于某个使用的类元	role:Name
构件	系统的一个物理组成单元	
数据类型	无身份的一组原始值的描述符	Name
接口	刻划行为特征的操作命名集	Iname
节点	计算资源	
信号	对象间的异步通信	«signal»
子系统	作为具有规范、实现和身份的单元的包	«subsystem»
用例	与外部代理交互中的实体行为规范	

类。类表示被建模的应用领域中的离散概念——物理实体（如飞机）、商业事物（如一份订单）、逻辑事物（如广播计划）、应用事物（如取消键）、计算机领域的事物（如哈希表）或行为事物（如一项任务）。类是有着相同结构、行为和关系的一组对象的描述符号。所用的属性与操作都被附在类或其他类元上。类是面向对象系统组织结构的核心。

对象是具有身份、状态和可激发行为的离散实体。对象是用来构造实际运行系统的个

China-pub.com 下载

体；类是用来理解和描述众多个别对象的个别概念。

类定义了一组有着状态和行为的对象。属性和关联用来描述状态。属性通常用没有身份的纯数据值表示，如数字和字符串。关联则用有身份的对象之间的关系表示。个体行为由操作来描述，方法是操作的实现。对象的生命期由附加给类的状态机来描述。类的表示法是一个矩形，由带有类名、属性和操作的分格框组成。如图 4-1所示。

一组类可以用泛化关系和建立在其内的继承机制分享公用的状态和行为描述。泛化使更具体的类（子类）与含有几个子类公同特性的更普通的类（超类）联系起来。一个类可以有零个或多个父类（超类）和零个或多个后代（子类）。一个类从它的双亲和祖先那里继承状态和行为描述，并且定义它的后代所继承的状态和行为描述。

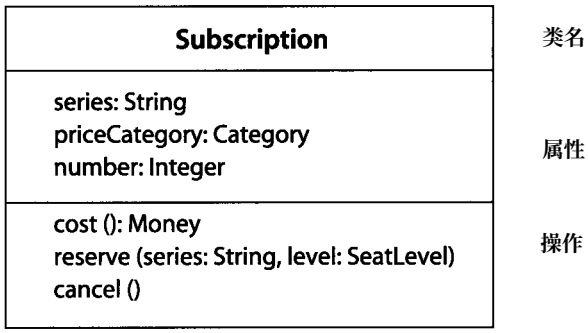


图4-1 类表示法

类在它的包含者内有惟一的名字，这个包含者通常可能是一个包但有时也可能是另一个类。类对它的包含者来说是可见的，可见性说明它如何被位于它的可见者之外的类所利用。类的多重性说明了有多少个实例可以存在，通常情况下，可以有多个（零个或多个，没有明确限制），但在执行过程中一个实例只属于一个类。

接口。接口是在没有给出对象的实现和状态的情况下对对象行为的描述。接口包含操作但不包含属性，并且它没有对外界可见的关联。一个或多个类或构件可以实现一个接口，并且每个类都可以实现接口中的操作。

数据类型。数据类型用以描述缺少身份的简单数据值。数据类型包括数字、字符串、枚举型数值。数据类型通过值来传递，并且是不可变的实体。数据类型没有属性，但是可以有操作。操作不改变数据值，但是可以把数据值作为结果返回。

含义分层。类可以存在于模型的几种含义层中，包括分析层、设计层和实现层。当表现真实世界的概念时，说明实际的状态、关系和行为是很重要的。但是像信息隐藏、有效性、可见性和方法这些实现方面的概念与真实世界的概念无关（它们是设计层的概念）。分析层类代表了在应用域中的逻辑概念或应用本身。分析层模型应该尽可能少地表示建模的系统，并在不涉及到执行和构造的情况下，充分说明系统的必要逻辑组成。

当表示高层次的设计时，有些概念与类直接相关，这些概念包括特定类的状态定位、对

China-pub.com
下载

象之间导航的效率、外部行为和内部实现的分离和准确操作的描述。设计层类表示了将状态信息和其上的操作封装于一个离散的单元内，它说明了关键的设计决定、信息定位和对象的功能。设计层类包含真实世界和计算机系统两方面的内容。

最后，当实现程序代码时，类的形式与所选择的语言紧密相关。如果一个通用类的功能不能直接用语言来实现，那么不得不放弃它们。实现层类直接与程序代码相对应。

同一个系统可以容纳多个层次的类，面向实现的类会在模型中实现更逻辑化的类。一个实现类表示用特定程序设计语言声明一个类，它得到了一个按照语言所要求的准确格式的类。然而，在许多情况下，分析、设计和实现信息可嵌套在一个单独的类中。

4.3 关系

类元之间的关系有关联、泛化、流及各种形式的依赖关系，包括实现关系和使用关系（参见表4-2）。

表4-2 关系的种类

关 系	功 能	表 示 法
关联	类实例间连接的描述	
依赖	两个模型元素间的关系	
流	在相继时间内一个对象的两种形式的关系	
泛化	更概括的描述和更具体的种类间的关系，适用于继承	
实现	说明和实现间的关系	
使用	一个元素需要别的元素提供适当功能的情况	

关联关系描述了给定类的单独对象之间语义上的连接。关联提供了不同类间对象可以相互作用的连接。其余的关系涉及到类元自身的描述，而不是它们的实例。

泛化关系使父类元（超类）与更具体的后代类元（子类）连接在一起。泛化有利于类元的描述，可以不用多余的声明，每个声明都需加上从其父类继承来的描述。继承机制利用泛化关系的附加描述构造了完整的类元描述。泛化和继承允许不同的类元分享属性、操作和它们共有的关系，而不用重复说明。

实现关系将说明和实现联系起来。接口是对行为而非实现的说明，而类之中则包含了实现的结构。一个或多个类可以实现一个接口，而每个类分别实现接口中的操作。

流关系将一个对象的两个版本以连续的方式连接起来。它表示一个对象的值、状态和位置的转换。流关系可以将类元角色在一次相互作用中连接起来。流的种类包括变成（同一个对象的不同版本）和拷贝（从现有对象创造出一个新的对象）两种。

China-pub.com 下载

依赖关系将行为和实现与影响其他类的类联系起来。除了实现关系以外，还有好几种依赖关系，包括跟踪关系（不同模型中元素之间的一种松散连接）、精化关系（两个不同层次意义之间的一种映射）、使用关系（在模型中需要另一个元素的存在）、绑定关系（为模板参数指定值）。使用依赖关系经常被用来表示具体实现间的关系，如代码层实现关系。在概括模型的组织单元，例如包时，依赖关系很有用，它在其上显示了系统的构架。例如编译方面的约束可通过依赖关系来表示。

4.4 关联

关联描述了系统中对象或实例之间的离散连接。关联将一个含有两个或多个有序表的类元，在允许复制的情况下连接起来。最普通的关联是一对类元之间的二元关联。关联的实例之一是链。每个链由一组对象（一个有序列表）构成，每个对象来自于相应的类。二元链包含一对对象。

关联带有系统中各个对象之间关系的信息。当系统执行时，对象之间的连接被建立和销毁。关联关系是整个系统中使用的“胶粘剂”，如果没有它，那么只剩下不能一起工作的孤立的类。

在关联中如果同一个类出现不止一次，那么一个单独的对象就可以与自己关联。如果同一个类在一个关联中出现两次，那么两个实例就不必是同一个对象，通常的情况都如此。

一个类的关联的任何一个连接点都叫做关联端，与类有关的许多信息都附在它的端点上。关联端有名字（角色名）和可见性等特性，而最重要的特性则是多重性，即一个类的多个实例与另一个类的一个实例相关。多重性对于二元关联很重要，因为定义 n元关联很复杂。

二元关联用一条连接两个类的连线表示。如图 4-2所示，连线上有相互关联的角色名而多重性则加在各个端点上。

如果一个关联既是类又是关联，即它是一个关联类，那么这个关联可以有它自己的属性（如图4-3）。如果一个关联的属性在一组相关对象中是唯一的，那么它是一个限定符（如图 4-4）。限定符是用来在关联中从一组相关对象中标识出独特对象的值。限定符对建模名称和身份代码很重要，同时它也是设计模型的索引。

在分析阶段，关联表示对象之间的逻辑关系。没有必要指定方向或者关心如何去实现它们。应该尽量避免多余的关联，因为它们不会增加任何逻辑信息。在设计阶段，关联用来说明关于数据结构的设计决定和类之间职责的分离。此时，关联的方向性很重要，而且为了提高对象的存取效率和对特定类信息的定位，也可引入一些必要的多余关联。然而，在该建模阶段，关联不应该等于 C++语言中的指针。在设计阶段带有导航性的关联表示对一个类有用的状态信息，而且它们能够以多种方式映射到程序设计语言当中。关联可以用一个指针、被嵌套的类甚至完全独立的表对象来实现。其他几种设计属性包括可见性和链的可变性。图 4-5表示了一些关联的设计特性。

China-pub.com
下载

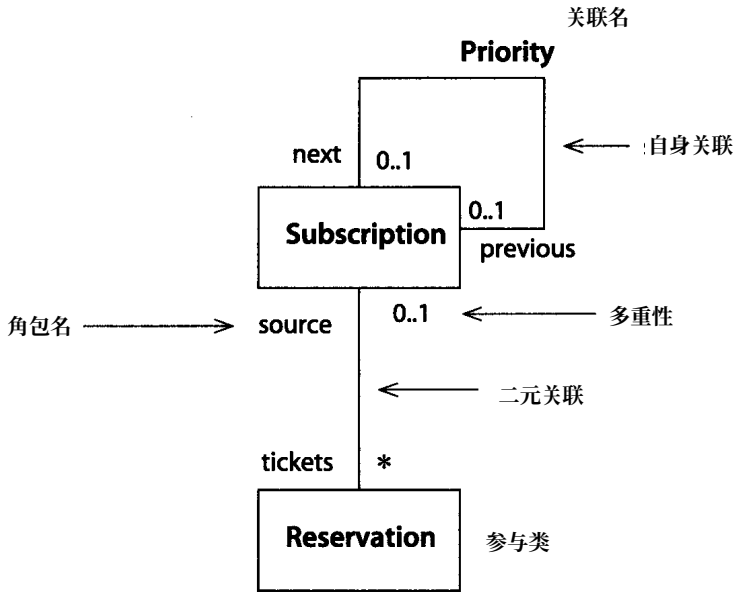


图4-2 关联表示法

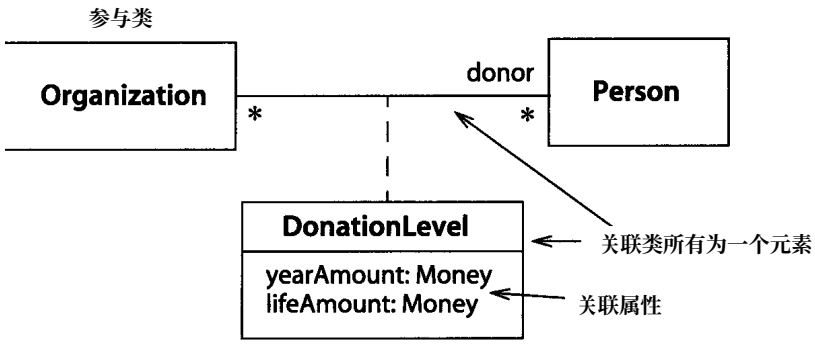


图4-3 关联类

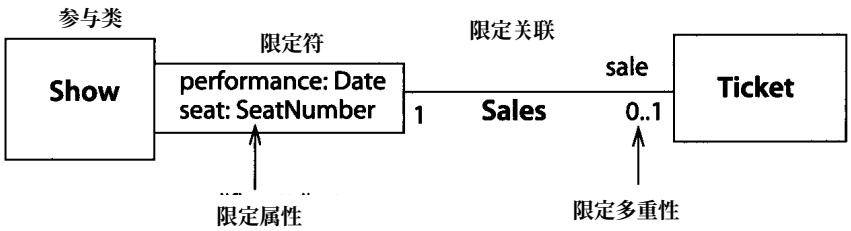


图4-4 限定关联

China-pub.com 下载

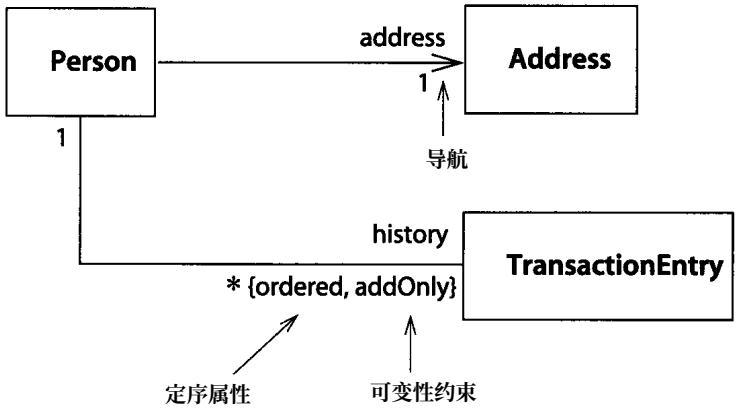


图4-5 关联的设计特性

聚集和组成。聚集表示部分与整体关系的关联，它用端点带有空菱形的线段表示，空菱形与聚集类相连接。组成是更强形式的关联，整体有管理部分的特有的职责，它用一个实菱形物附在组成端表示。每个表示部分的类与表示整体的类之间有单独的关联，但是为了方便起见，连线结合在一起，现在整组关联就像一棵树。图 4-6表示了聚集关联和组成关联。

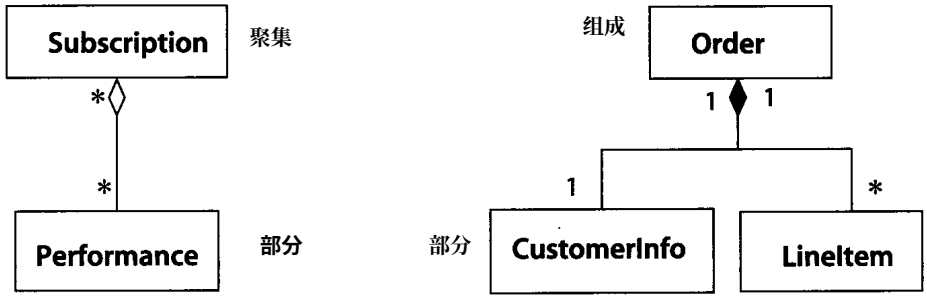


图4-6 聚集和组成

链。链是关联的一个实例。链即所涉及对象的一个有序表，每个对象都必须是关联中对应类的实例或此类后代的实例。系统中的链组成了系统的部分状态。链并不独立于对象而存在，它们从与之相关的对象中得到自己的身份（在数据库术语中，对象列表是链的键）。在概念上，关联与相关类明显不同。而在实际中，关联通常用相关类的指针来实现，但它们可以作为与其相连的类分离的包含体对象来实现。

双向性。关联的不同端很容易辨认，哪怕它们都是同一种类。这仅仅意味着同一个类的不同对象是可以相互联系的。正是因为两端是可区分的，所以关联是不对称的（除了个别的例子外），且两个端点也是不能互相交换的。在通常情形下这是一个共识：就像动词短语中的主语和宾语不能互换一样。关联有时被认为是双向性的，这意味着逻辑关系在两个方向上都起作用。这个观点经常被错误理解，甚至包括一些方法学家。这并不意味着每个类“了解”其他类，或者说在实现

China-pub.com
下载

中类与类之间可以互相访问。这仅仅意味着任何逻辑关系都有其反向性，无论这个反向性容不容易计算。如果关联只在一个方向横穿而不能在另一个方向横穿，那么关联就被认为有导航性。

为什么使用基本模型，而不用编程语言中流行的指针来表示关联？原因是模型试图说明系统实现的目的。如果两个类之间的关系在模型中用一对指针来表示，那么这两个指针仍然相关。关联方法表明关系在两个方向都有意义，而不管它们是如何实现的。将关联转化成一对用于实现的指针很容易，但是很难说明这两个指针是彼此互逆的，除非这是模型的一部分。

4.5 泛化

泛化关系是类元的一般描述和具体描述之间的关系，具体描述建立在一般描述的基础之上，并对其进行了扩展。具体描述与一般描述完全一致，具有其所有特性、成员和关系，并且包含补充的信息。例如，抵押是借贷中具体的一种，抵押保持了借贷的基本特性并且加入了附加的特性，如房子可以作为借贷的一种抵押品。一般描述被称作父，具体描述被称作子，如借贷是父而抵押则是子。泛化在类元（类、接口、数据类型、用例、参与者、信号等等）、包、状态机和其他元素中使用。在类中，术语超类和子类代表父和子。

泛化用从子指向父的箭头表示，指向父的是一个空三角形（如图 4-7表示）。多个泛化关系可以用箭头线组成的树来表示，每一个分支指向一个子类。

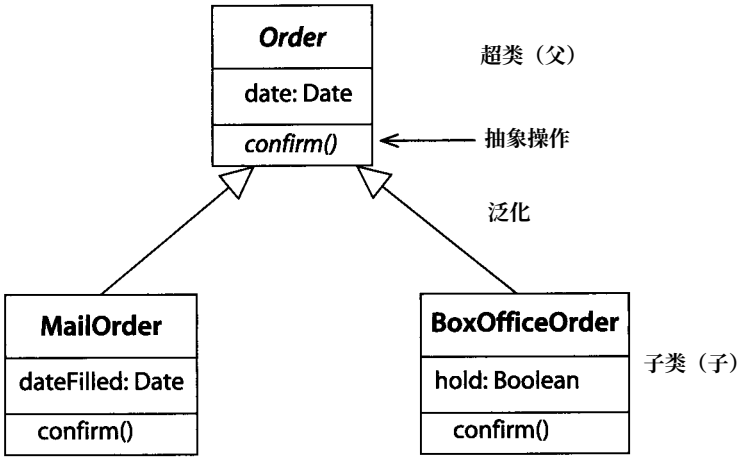


图4-7 泛化表示法

泛化的用途。泛化有两个用途。第一个用途是用来定义下列情况：当一个变量（如参数或过程变量）被声明承载某个给定类的值时，可使用类（或其他元素）的实例作为值，这被称作可替代性原则（由 Barbara Liskov提出）。该原则表明无论何时祖先被声明了，则后代的一个实例可以被使用。例如，如果一个变量被声明拥有借贷，那么一个抵押对象就是一个合

China-pub.com 下载

法的值。

泛化使得多态操作成为可能，即操作的实现是由它们所使用的对象的类，而不是由调用者确定的。这是因为一个父类可以有许多子类，每个子类都可实现定义在类整体集中的同一操作的不同变体。例如，在抵押和汽车借贷上计算利息会有所不同，它们中的每一个都是父类借贷中计算利息的变体。一个变量被声明拥有父类，接着任何子类的一个对象可以被使用，并且它们中的任何一个都有着自己独特的操作。这一点特别有用，因为在不需要改变现有多态调用的情况下就可以加入新的类。例如，一种新的借贷可被新增加进来，而现存的用来计算利息操作的代码仍然可用。一个多态操作可在父类中声明但无实现，其后代类需补充该操作的实现。这种不完整操作是抽象的（其名称用斜体表示）。

泛化的另一个用途是在共享祖先所定义的成分的前提下允许它自身定义增加的描述，这被称作继承。继承是一种机制，通过该机制类的对象的描述从类及其祖先的声明部分聚集起来。继承允许描述的共享部分只被声明一次而可以被许多类所共享，而不是在每个类中重复声明并使用它，这种共享机制减小了模型的规模。更重要的是，它减少了为了模型的更新而必须做的改变和意外的前后定义不一致。对于其他成分，如状态、信号和用例，继承通过相似的方法起作用。

4.5.1 继承

每一种泛化元素都有一组继承特性。对于任何模型元素均包括约束。对类元而言，它们同样包括一些特性（如属性、操作和信号接收）和关联中的参与者。一个子类继承了它的所有祖先的可继承的特性。它的完整特性包括继承特性和直接声明的特性。

对类元而言，没有具有相同特征标记的属性会被多次声明，无论直接的或继承的，否则将发生冲突，且模型形式错误。换而言之，祖先声明过的属性不能被后代再次声明。如果类的接口一致（具有同样的参数、约束和含义），操作可在多个类中声明。附加的声明是多余的。一个方法在层次结构中可以被多个类声明，附在后代上的方法替代（重载）在任何祖先中声明过的具有相同特征标记的方法。如果一个方法的两个或多个副本被一个类继承（通过不同类的多重继承），那么它们会发生冲突并且模型形式错误（一些编程语言允许显式选定其中的一种方法。我们发现如果在后代类中重新定义方法会更简单、安全）。元素中的约束是元素本身及它所有祖先的约束的联合体，如果它们存在不一致，那么模型形式错误。

在一个具体的类中，每一个继承或声明的操作都必须有一个已定义的方法，无论是直接定义或从祖先那里继承而来的。

4.5.2 多重继承

如果一个类元有多个父类，那么它从每一父类那里都可得到继承信息（如图 4-8）。它的特征（属性、操作和信号）是它的所有父类特征的联合。如果同一个类作为父类出现在多条路径上，那么它的每一个成员中只有它的一个拷贝。如果有着同样特征的特性被两个类声明，

China-pub.com
下载

而这两个类不是从同一祖先那里继承来的（即独立声明），那么声明会发生冲突并且模型形式错误。因为经验告诉我们设计者应自行解决这个问题，所以 UML不提供这种情形的冲突解决方案。像Eiffel这样的语言允许冲突被程序设计者明确地解决，这比隐式的冲突解决原则要安全，而这些原则经常使开发者大吃一惊。

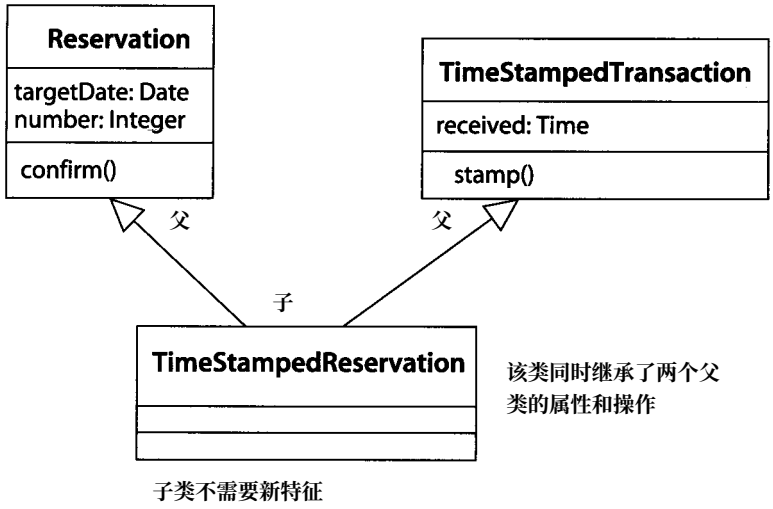


图4-8 多重继承

4.5.3 单分类和多重分类

在最简单的形式中，一个对象仅属于一个类，许多面向对象的语言有这种限制。一个对象仅属于一个类并没有逻辑上的必要性，我们只要从多个角度同时观察一下真实世界的对象就可以发现这一点。在 UML更概括的形式中，一个对象可以有一个或多个类。对象看起来就好像它属于一个隐式类，而这个类是每个直接父类的子类 —— 多重继承可以免去再声明一个新类，这可提高效率。

4.5.4 静态分类和动态分类

在最简单的形式中，一个对象在被创建后不能改变它的类。我们再次说明，这种限制并没有逻辑上的必要性，而是使面向对象编程语言的实现更容易些。在更普遍的形式下，一个对象可以动态改变它的类，这么做会得到或失去一些属性或关联。如果对象失去了它们，那么在它们中的信息也就失去了并且过后也不能被恢复，哪怕这个对象变回了原来的类。如果这个对象得到了属性或关联，那么它们必须在改变时就初始化，就像初始化一个新对象一样。

当多重分类和动态分类一起使用时，一个对象就可以在它的生命期内得到或失去类。动态类有时被称作角色或类型。一个常见的建模模式是每个对象有一个唯一的静态的固有类（即不

China-pub.com 下载

能在对象的生命期内改变的类），加上零个或多个可以在对象生命期内加入或移走的角色类。固有类描述了这个对象的基本特性，而角色类描述了暂时的特性。虽然许多程序设计语言不支持类声明中的多重动态分类，然而它仍然是一个很有用的建模概念，并且可以被映射到关联上。

4.6 实现

实现关系将一种模型元素（如类）与另一种模型元素（如接口）连接起来，其中接口只是行为的说明而不是结构或者实现。客户必须至少支持提供者的所有操作（通过继承或者直接声明）。虽然实现关系意味着要有像接口这样的说明元素，它也可以用一个具体的实现元素来暗示它的说明（而不是它的实现）必须被支持。例如，这可以用来表示类的一个优化形式和一个简单低效的形式之间的关系。

泛化和实现关系都可以将一般描述与具体描述联系起来。泛化将在同一语义层上的元素连接起来（如，在同一抽象层），并且通常在同一模型内。实现关系将在不同语义层内的元素连接起来（如，一个分析类和一个设计类；一个接口与一个类），并且通常建立在不同的模型内。在不同发展阶段可能有两个或更多的类等级，这些类等级的元素通过实现关系联系起来。两个等级无需具有相同的形式，因为实现的类可能具有实现依赖关系，而这种依赖关系与具体类是不相关的。

实现关系用一条带封闭空箭头的虚线来表示（如图 4-9），且与泛化的符号很相像。

用一种特殊的折叠符号来表示接口（无内容）以及实现接口的类或构件。接口用一个圆圈表示，它通过实线附在表示类元的矩形上（如图 4-10）。

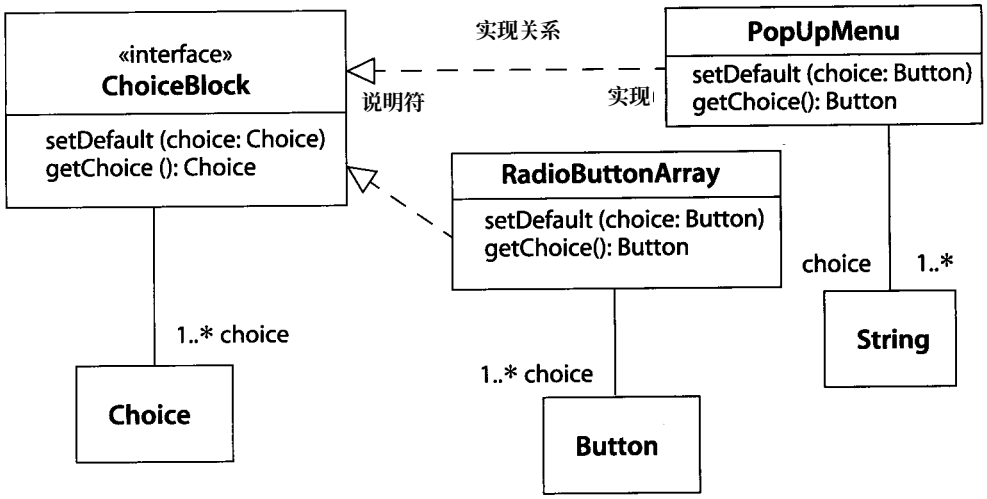


图4-9 实现关系

China-pub.com
下载

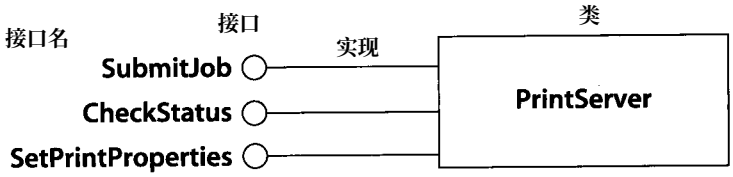


图4-10 接口和实现图标

4.7 依赖

依赖表示两个或多个模型元素之间语义上的关系。它只将模型元素本身连接起来而不需要用一组实例来表达它的意思。它表示了这样一种情形，提供者的某些变化会要求或指示依赖关系中客户的变化。

根据这个定义，关联和泛化都是依赖关系，但是它们有更特别的语义，故它们有自己的名字和详细的语义。我们通常用依赖这个词来指其他的关系。表 4-3列出了UML基本模型中的一些依赖关系。

表4-3 依赖关系种类

依赖关系	功能	关键字
访问	允许一个包访问另一个包的内容	access
绑定	为模板参数指定值，以生成一个新的模型元素	bind
调用	声明一个类调用其他类的操作的方法	call
导出	声明一个实例可从另一个实例导出	derive
友员	允许一个元素访问另一个元素，不管被访问的元素是否具有可见性	friend
引入	允许一个包访问另一个包的内容并为被访问包的组成部分增加别名	import
实例化	关于一个类的方法创建了另一个类的实例的声明	instantiate
参数	一个操作和它的参数之间的关系	parameter
实现	说明和其实之间的映射关系	realize
精化	声明具有两个不同语义层次上的元素之间的映射	refine
发送	信号发送者和信号接收者之间的关系	send
跟踪	声明不同模型中的元素之间存在一些连接，但不如映射精确	trace
使用	声明使用一个模型元素需要用到已存在的另一个模型元素，这样才能正确实现使用者的功能（包括调用、实例化、参数、发送）	use

跟踪是对不同模型中元素的连接的概念表述，通常这些模型是开发过程中不同阶段的模型。跟踪缺少详细的语义，它特别用来追溯跨模型的系统要求和跟踪模型中会影响其他模型的模型所起的变化。

精化是表示位于不同的开发阶段或处于不同的抽象层次中的一个概念的两种形式之间的

China-pub.com 下载

关系。这并不意味着两个概念会在最后的模型中共存，它们中的一个通常是另一个的未完善的形式。原则上，在较不完善到较完善的概念之间有一个映射，但这并不意味着转换是自动的。通常，更详细的概念包含着设计者的设计决定，而决定可以通过许多途径来制定。原则上讲，带有偏移标记的对一个模型的改变可被另一个模型证实。而实际上，现有的工具不能完成所有这些映射，虽然一些简单的映射可以实现。因此精化通常提醒建模者多重模型以可预知的方式发生相互关系。

导出表示一个元素可以通过计算另一个元素来得到（而被导出的元素可以被明确包含在系统中以避免花费太多代价进行迭代计算）。导出、实现、精化和跟踪是抽象的依赖——它们将同一个潜在事物的不同形式联系起来。

使用表示的是一个元素的行为或实现会影响另一个元素的行为或实现。通常，这来自于与实现有关的一些问题，如编译程序要求在编译一个类前要对另一个类进行定义。大部分使用依赖关系可以从代码中获得，而且它们不需要明确声明，除非它们是自顶向下设计风格的系统的一部分（如，使用预定义的构件或函数库）。特别的使用关系可以被详细说明，但是因为关系的目的是为了突出依赖，所以它常常被忽略。确切的细节可以从实现代码中获得。使用的构造型包括调用和实例。调用表示一个类中的方法调用另一个类的操作；实例表示一个类的方法创建了另一个类的实例。

若干种使用依赖允许某些元素访问其他元素。访问依赖允许一个包看到另一个包的内容。引入依赖能够达到更高要求，可以将目标包内容的名字加入到引入包的命名空间内。友员依赖是一种访问依赖，允许客户看到提供者的私有内容。

绑定是将数值分配给模板的参数。它是具有精确语义的高度结构化的关系，可通过取代模板备份中的参数实现。

使用和绑定依赖在同一语义层上将很强的语义包括进元素内。它们必须连接模型同一层的元素（或者都是分析层，或者都是设计层，并且在同一抽象层）。跟踪和精化依赖更模糊一些，可以将不同模型或不同抽象层的元素连接起来。

关系（一个元关系，不只限于依赖关系）实例表示一个元素（如对象）是另一个元素（如类）的实例。

依赖用一个从客户指向提供者的虚箭头表示，用一个构造型的关键字来区分它的种类，如图4-11所示。

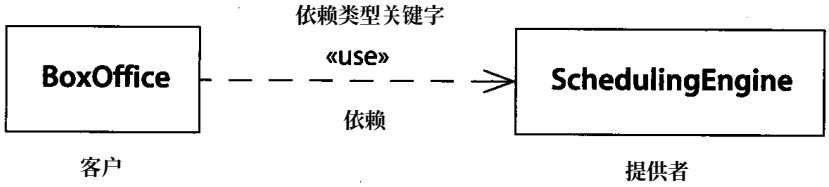


图4-11 依赖关系

China-pub.com
下载

4.8 约束

UML以模型元素图的形式为建模系统提供一组概念和关系，而它们中的一些用文字表达会更好一些，即利用文字语言的优势。约束是布尔表达式，可用由特定语言来解释的字符串表示。自然语言、集合论中的符号、约束语言或各种编程语言都可以用来表示约束。 UML中定义了一种约束语言， OCL，它有利于表达UML约束并且有希望得到广泛支持。详见《OCL入门》和[Warmer-99]等书。

约束可以用来表示各种非局部的关系，如关联路径上的限制。约束尤其可以用来表述存在特性（存在X则C条件成立）和通用特性（对于Y中的所有y，条件D必须成立）。

一些标准的约束已经被UML预先定义为标准元素，包括异或关系中的关联和泛化中子类关系中的各种约束。详见第14章。

约束用大括弧内的文字表达式来表示，可以使用形式语言或自然语言。文字字符串可以写成注释或附加在依赖关系的箭头旁。图4-12表示了一些约束。

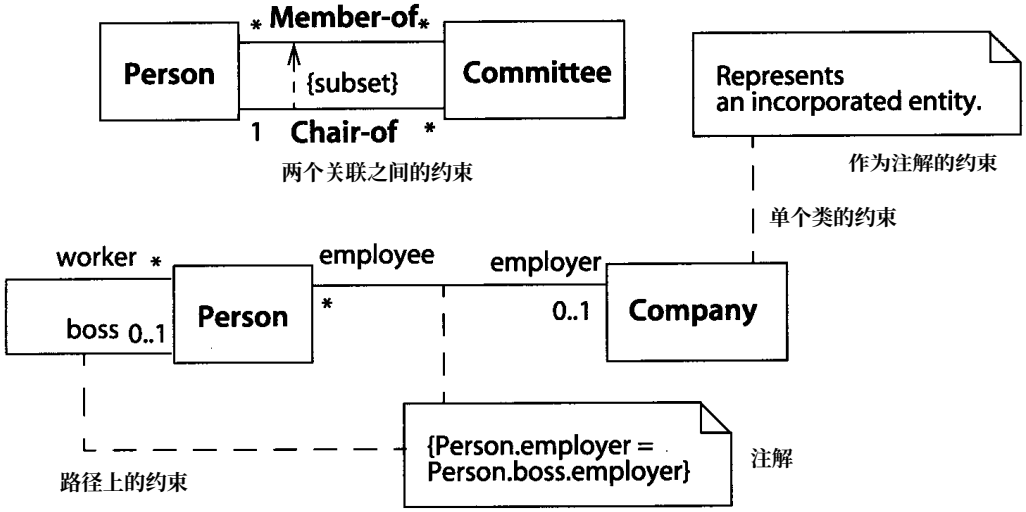


图4-12 约束关系

4.9 实例

实例是有身份标识的运行实体，即它可以与其他运行实体相区分。它在任何时刻都有一个值，随着对实例进行操作值也会被改变。

模型的用途之一是描述一个系统的可能状态和它们的行为。模型是对潜在性的描述，对可能存在的对象集和对象经历的可能行为历史的描述。静态视图定义和限制了运行系统值的可能配置。动态视图定义了运行系统从一个配置传递到另一个的途径。总之，静态视图和建

China-pub.com 下载

立在其上的各种动态视图定义了系统的结构和行为。

在某一时刻一个系统特定的静态配置叫做快照。快照包括对象和其他实例、数值和链。对象是类的实例，是完全描述它的类的直接实例和那个类的祖先的间接实例（如果允许多重分类，则对象可能是多个类的直接实例）。同样，链是关联的实例，数值是数据类型的实例。

对象对于它的类的每个属性有一个数据值，每个属性值必须与属性的数据类型相匹配。如果属性有可选的或多重的多重性，那么属性可以有零个或多个值。链包含有多个值，每一个值是一个给定类的或给定类的后代的对象的引用。对象和链必须遵从它们的类或关联的约束（其中既包括明确的约束又包括如多重性的内嵌的约束）。

如果系统的每个实例是一个形式良好的系统模型的一些元素的实例，并且实例满足模型的所有约束，则说明系统的状态是有效的系统实例。

静态视图定义了一组能够在单独快照中存在的对象、值和链。原则上，任何与静态图相一致的对象和链的结合都是一个模型可能的配置。但这不意味着每个可能的快照能够或将要出现。某些快照可能在静态下合法但在系统的动态图下可能不会被动态地达到。

UML的行为部分描述了快照的有效顺序，快照可能作为内部和外部行为影响的结果出现。动态图定义了系统如何从一个快照转换到另一个快照。

4.10 对象图

快照的图是系统在某一时刻的图像。因为它包含对象的图像，因此也被叫做对象图。作为系统的样本它是有用的，如可以用来说明复杂的数据结构或一系列的快照中表示行为（如图4-13）。请记住所有的快照都是系统的样本，而不是系统的定义。系统结构和行为在定义视图中定义，且建立定义视图是建模和设计的目标。

静态视图描述了可能出现的实例。除了样本外，实际的实例不总是直接在模型中出现。

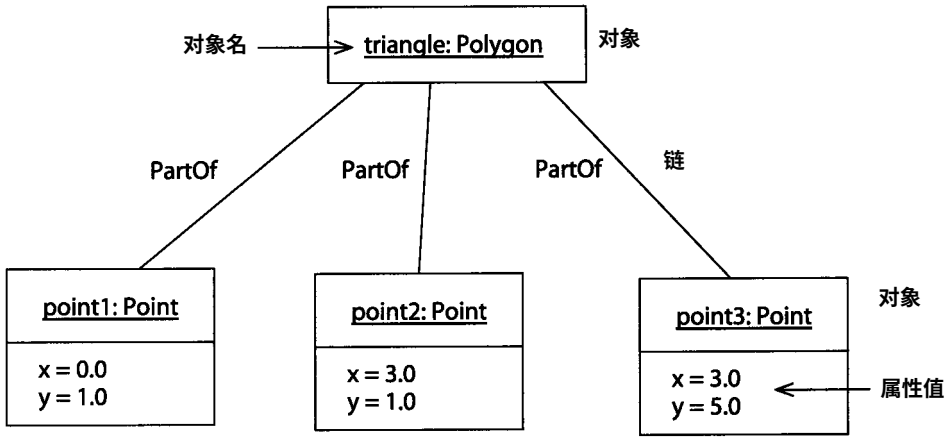


图4-13 对象图

China-pub.com
下载

第5章
用 例 视 图

5.1 概述

当用例视图在外部用户前出现时，它捕获到系统、子系统或类的行为。它将系统功能划分成对参与者(即系统的理想用户)有用的需求。而交互功能部分被称作用例。用例使用系统与一个或多个参与者之间的一系列消息来描述系统中的交互作用。参与者可以是人,也可以是外部计算机系统和外部进程。图5-1表述了一个电话目录销售的用例视图。此例是实际系统简化后的例子。

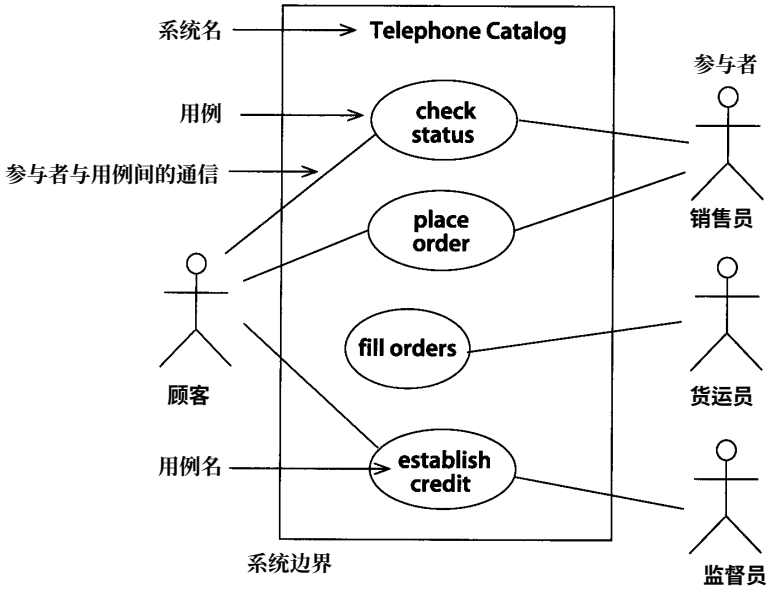


图5-1 用例视图

5.2 参与者

参与者是与系统、子系统或类发生交互作用的外部用户、进程或其他系统的理想化概念。

China-pub.com 下载

作为外部用户与系统发生交互作用，这是参与者的特征。在系统的实际运作中，一个实际用户可能对应系统的多个参与者。不同的用户也可以只对应于一个参与者，从而代表同一参与者的不同实例。

每个参与者可以参与一个或多个用例。它通过交换信息与用例发生交互作用（因此也与用例所在的系统或类发生了交互作用），而参与者的内部实现与用例是不相关的，参与者可以被一组定义它的状态的属性充分描述。

参与者可以通过泛化关系来定义，在这种泛化关系中，一个参与者的抽象描述可以被一个或多个具体的参与者所共享。

参与者可以是人、另一个计算机系统或一些可运行的进程。在图中，参与者用一个名字写在下面的小人表示。

5.3 用例

用例是外部可见的一个系统功能单元，这些功能由系统单元所提供，并通过一系列系统单元与一个或多个参与者之间交换的消息所表达。用例的用途是在不揭示系统内部构造的情况下定义连贯的行为。用例的定义包含用例所必需的所有行为 —— 执行用例功能的主线次序、标准行为的不同变形、一般行为下的所有异常情况及其预期反应。从用户角度来看，上述情况很可能是异常情况；从系统角度来看，它们是必须被描述和处理的附加情况。

在模型中，每个用例的执行独立于其他用例，虽然在具体执行一个用例功能时由于用例之间共享对象的缘故可能会造成本用例与其他用例之间有这样或那样的隐含的依赖关系。每一个用例都是一个纵向的功能块，这个功能块的执行会和其他用例的执行发生混杂。

用例的动态执行过程可以用 UML的交互作用来说明，可以用状态图、顺序图、协作图或非正式的文字描述来表示。用例功能的执行通过类之间的协作来实现。一个类可以参与多个协作，因此也参与了多个用例。

在系统层，用例表示整个系统对外部用户可见的行为。一个用例就像外部用户可使用的系统操作。然而，它又与操作不同，用例可以在执行过程中持续接受参与者的输入信息。用例也可以被像子系统和独立类这样的小单元所应用。一个内部用例表示了系统的一部分对另一部分呈现出的行为。例如，某个类的用例表示了一个连贯的功能，这个功能是该类提供给系统内其他有特殊作用的类的。一个类可以有多个用例。

用例是对系统一部分功能的逻辑描述，它不是明显的用于系统实现的构件。非但如此，每个用例必须与实现系统的类相映射。用例的行为与类的状态转换和类所定义的操作相对应。只要一个类在系统的实现中充当多重角色，那么它将实现多个用例的一部分功能。设计过程的一部分工作即在不引入混乱的情况下，找出具有明显的多重角色的类，以实现这些角色所涉及的用例功能。用例功能靠类间的协作来实现。

用例除了与其参与者发生关联外，还可以参与系统中的多个关系（如表 5-1）。

China-pub.com
下载

表5-1 用例之间的关系

关 系	功 能	表 示 法
关联	参与者与其参与执行的用例之间的通信路径	————
扩展	在基用例上插入基用例不能说明的扩展部分	«extend» - - - - ->
用例泛化	用例之间的一般和特殊关系，其中特殊用例继承了一般用例的特性并增加了新的特性	——▷
包含	在基用例之上插入附加行为，并且具有明确的描述	«include» - - - - ->

如图5-2，用例用一个名字在里面的椭圆表示，用例和与它通信的参与者之间用实线连接。

虽然每个用例的实例是独立的，但是一个用例可以用其他的更简单的用例来描述。这有点像一个类可以通过继承它的超类并增加附加描述来定义。一个用例可以简单地包含其他用例具有的行为，并把它所包含的用例行为做为自身行为的一部分，这被称作包含关系。在这种情况下，新用例不是初始用例的一个特殊例子，而且不能被初始用例代替。

一个用例也可以被定义为基用例的增量扩展，这叫做扩展关系。同一个基用例的几个扩展用例可以在一起应用。基用例的扩展增加了原有的语义，此时是基用例而不是扩展用例被作为例子使用。

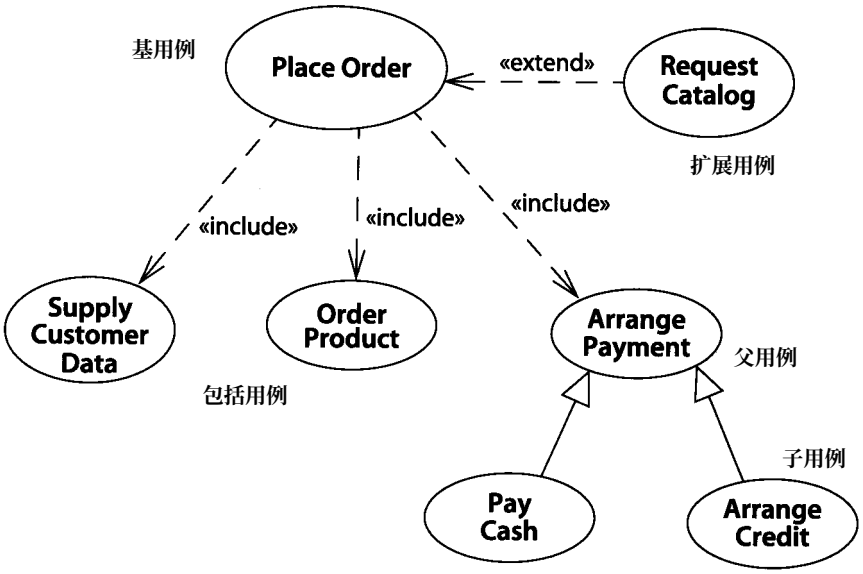


图5-2 用例之间的关系

包含和扩展关系可以用含有关键字《include》和《extend》的带箭头的虚线表示。包含

China-pub.com 下载

关系箭头指向被包含的用例，扩展关系箭头指向被扩展的用例。

一个用例也可以被特别列举为一个或多个子用例，这被称做用例泛化。当父用例能够被使用时，任何子用例也可以被使用。

用例泛化与其他泛化关系的表示法相同，都用一个三角箭头从子用例指向父用例。图 5-2 表示了销售中的用例关系。

China-pub.com
下载

第6章
状态机视图

6.1 概述

状态机视图通过对类对象的生存周期建立模型来描述对象随时间变化的动态行为。每一个对象都被看作是通过对事件进行探测并做出回应来与外界其他部分通信的独立的实体。事件表示对象可以探测到的事物的一种运动变化 —— 如接受到从一个对象到另一个对象的调用或信号、某些值的改变或一个时间段的终结。任何影响对象的事物都可以是事件，真实世界所发生的事物的模型通过从外部世界到系统的信号来建立。

状态是给定类的对象的一组属性值 ,这组属性值对所发生的事件具有相同性质的反应。换而言之，处于相同状态的对象对同一事件具有同样方式的反应，所以当给定状态下的多个对象接受到相同事件时会执行相同的动作，然而处于不同状态下的对象会通过不同的动作对同一事件做出不同的反应。例如，当自动答复机处于处理事务状态或空闲状态时会对取消键做出不同的反应。

状态机用于描述类的行为，但它们也描述用例、协作和方法的动态行为。对这些对象而言，一个状态代表了执行中的一步。我们通常用类和对象来描述状态机，但是它也可以被其他元素所直接应用。

6.2 状态机

状态机是展示状态与状态转换的图。通常一个状态机依附于一个类，并且描述一个类的实例对接受到的事件所发生的反应。状态机也可以依附于操作、用例和协作并描述它们的执行过程。

状态机是一个类的对象所有可能的生命历程的模型。对象被孤立地从系统中抽出和考察，任何来自外部的影响被概述为事件。当对象探测到一个事件后，它依照当前的状态做出反应，反应包括执行一个动作和转换到新状态。状态机可以构造成继承转换，也能够对并发行为建立模型。

China-pub.com 下载

状态机是一个对象的局部视图，一个将对象与其外部世界分离开来并独立考查其行为的图。利用状态机可以精确地描述行为，但不适合综合理解系统执行操作。如果要更好地理解整个系统范围内的行为产生的影响，那么交互视图将更有用些。然而，状态机有助于理解如用户接口和设备控制器这样的控制机。

6.3 事件

事件是发生在时间和空间上的一点的值得注意的事情。它在时间上的一点发生，没有持续时间。如果某一事情的发生造成了影响，那么在状态机模型中它是一个事件。当我们使用事件这个词时，通常是指一个事件的描述符号，即对所有具有相同形式的独立发生事件的描述，就像类这个词表示所有具有相同结构的独立类一样。一个事件的具体发生叫做事件的实例。事件可能有参数来辨别每个实例，就像类用属性来辨别每个对象。对类而言，信号利用泛化关系来进行组织，以使不同的类共享公用的结构。事件可以分成明确或隐含的几种：信号事件、调用事件、改变事件、时间事件等。表 6-1是几种事件类型及其描述。

表6-1 事件的种类

事件类型	描 述	语 法
调用事件	接受等待应答的对象的明确形式的同步请求	*op*(*a*:*T*)
改变事件	对布尔表达式值的修改	*When*(*exp*)
信号事件	接受一个对象间外在的、命名的、异步的通信	*Sname*(*a*:*T*)
时间事件	绝对时间的到达或一个相对时间段的终结	*After*(*time*)

信号事件。信号是作为两个对象之间的通信媒介的命名的实体，信号的接收是信号接受对象的一个事件。发送对象明确地创建并初始化一个信号实例并把它发送到一个或一组对象。最基本的信号是异步单路通信，发送者不会等待接收者如何处理信号而是独立地做它自己的工作。在双路通信模型中，要用到多路信号，即至少要在每个方向上有一个信号。发送者和接受者可以是同一个对象。

信号可以在类图中被声明为类元，并用关键字《signal》表示，信号的参数被声明为属性。同类元一样，信号间可以有泛化关系，信号可以是其他信号的子信号，它们继承父信号的参数，并且可以触发依赖于父信号的转换（如图 6-1）。

调用事件。调用事件是一个对象对调用的接收，这个对象用状态的转换而不是用固定的处理过程实现操作。对调用者来说，普通的调用（用方法实现的调用）不会被调用事件所辨别。接收者不是用方法来实现操作就是触发一个状态转换来实现这个操作。操作的参数即事件的参数。一旦调用的接收对象通过由事件触发的转换完成了对调用事件的处理或调用失败而没有进行任何状态转换，则控制返回到调用对象。不过，与普通的调用不同，调用事件的接收者会继续它自己的执行过程，与调用者处于并行状态。

China-pub.com
下载

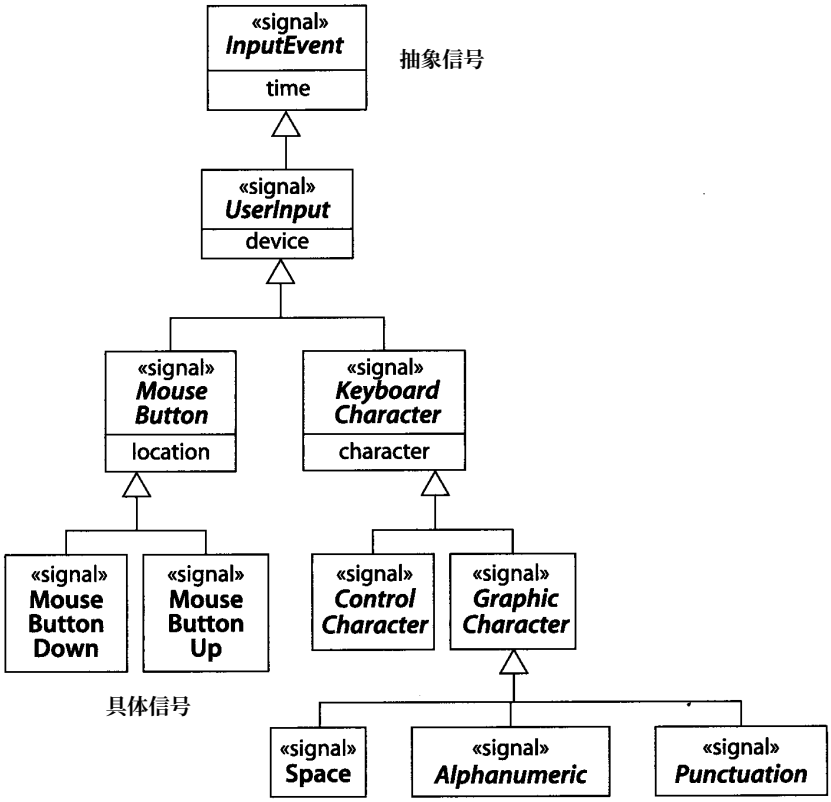


图6-1 信号的层次

改变事件。当是依赖于特定属性值的布尔表达式所表示的条件满足时，事件发生改变。这是等到特定条件被满足的一种声明途径，但是一定要小心使用它，因为它表示了一种具有时间持续性的并且可能是涉及全局的计算过程（是一种远距离的动作，因为被测试的值可能是远距离的）。这既有好处也有坏处，好处在于它将模型集中在真正的依赖关系上——一种当给定条件被满足时发生的作用——而不是集中在测试条件的机制上。缺点在于它使修改系统潜在值和最终效果的活动之间的因果关系变得模糊了。测试改变事件的代价可能很大，因为原则上改变事件是持续不断的。而实际上，又存在着避免不必要的计算的方法。改变事件应该仅用在当一个具有更明确表达形式的通信形式显得不自然时。

请注意监护条件与改变事件的区别。监护条件只是在引起转换的触发器事件触发时和事件接收者对事件进行处理时被赋值一次。如果它为假，那么转换将不会被激发，条件也不会被再赋值。而改变事件被多次赋值直到条件为真，这时转换也会被激发。

时间事件。时间事件代表时间的流逝。时间事件既可以被指定为绝对形式（天数），也可以被指定为相对形式（从某一指定事件发生开始所经历的时间）。在高层模型中，时间事件可

China-pub.com 下载

以被认为是来自整个世界的事件；在实现模型中，它们由一些特定对象的信号所引起，这些对象既可能是操作系统也可能是应用中的对象。

6.4 状态

状态描述了一个类对象生命期中的一个时间段。它可以用三种附加方式说明：在某些方面性质相似的一组对象值；一个对象等待一些事件发生时的一段时间；对象执行持续活动时的一段时间。虽然状态通常是匿名的，并仅用处于该状态时对象进行的活动描述，但它也可以有名字。

在状态机中，一组状态由转换相连接。虽然转换连接着两个状态（或多个状态，如果图中含有分支和结合控制），但转换只由转换出发的状态处理。当对象处于某种状态时，它对触发状态转换的触发器事件很敏感。

状态用具有圆形拐角的矩形表示。如图 6-2所示。

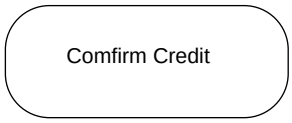


图6-2 状态

6.5 转换

从状态出发的转换定义了处于此状态的对象对外界发生的事件所做出的反应。通常，定义一个转换要有引起转换的触发器事件、监护条件、转换的动作和转换的目标状态。表 6-2 列出了几种转换和由转换所引起的隐含动作。

表6-2 转换的种类及隐含动作

转换的种类	描 述	语 法
入口动作	进入某一状态时执行的动作	entry/*action*
出口动作	离开某一状态时执行的动作	exit/*action*
外部转换	引起状态改变的转换或自身转换，同时执行一个具体的动作，包括引起入口动作和出口动作被执行的转换	*e*(*a*:*T*)[*exp*]/*action*
内部转换	引起一个动作的执行但不改变状态或不引起入口动作或出口动作的执行	*e*(*a*:*T*)[*exp*]/*action*

外部转换。外部转换是一种改变活动状态的转换，它是最普通的一种转换。它用从源状

China-pub.com
下载

态到目标状态的箭头表示，其他属性以文字串附加在箭头旁（如图 6-3）。

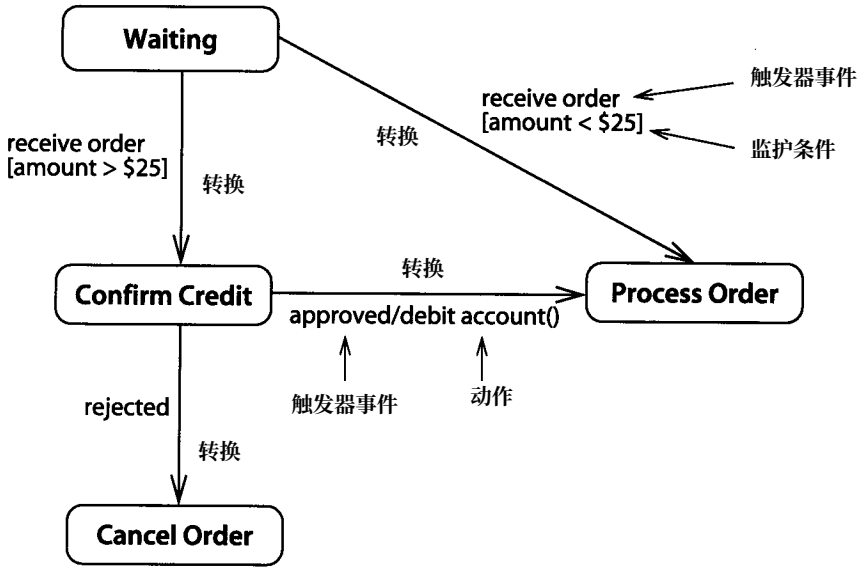


图6-3 外部转换

触发器事件。触发器事件是引起转换的事件。事件可以有参数，以供转换的动作使用。如果一个信号有后代，那么信号中的任一个后代都可以引起转换。例如，如果转换将 *Mouse Button* 作为触发器，那么*Mouse Button Down*可以触发这个转换（如图6-1）。

事件并不是持续发生的，它只在时间的一点上发生。当一个对象接收到一个事件时，如果它没有空闲时间来处理事件，就将事件保存起来。对象一次只处理一个事件，在对象处理事件时转换必须激发，事件过后是不会被记住的（某些特殊的延迟事件除外，在触发一个转换前或处延迟被解除前，这类事件被保存起来）。如果两个事件同时发生，它们被每次处理一个。没有触发任何转换的事件被简单地忽略或遗弃，这并不是一个错误，忽略不想要的事件要比详细指明所有事件容易得多。

监护条件。转换可能具有一个监护条件，监护条件是一个布尔表达式。监护条件可以引用对象的属性值和触发事件的参数。当一个触发器事件被触发时，监护条件被赋值。如果布尔表达式的值为“真”，那么触发事件，即使转换有效。如果布尔表达式的值为“假”，则不会引起转换。监护条件只能在触发事件发生时被赋值一次。如果在转换发生后监护条件由原来的“假”变为“真”，则因为赋值太迟而不能触发转换。

从一个状态引出的多个转换可以有同样的触发器事件，但是每个转换必须具有不同的监护条件。当其中一个监护条件满足时，触发器事件会引起相应的转换。通常，监护条件的设置要考虑到各种可能的情况以确保一个触发器事件的发生应该能够引起某些转换。如果有些情况没有考虑到，一个触发器事件没有引起任何转换，那么在状态机视图中要忽略这个事件。

China-pub.com 下载

一个事件的发生只能同时引起一个转换（在一个控制线程中）。如果一个事件可能引起多个转换，那么其中只有一个转换有效。如果两个相互矛盾的转换同时有效，则无法确定到底发生了哪个转换。这两个转换随机地发生一个，或者由系统的实现细节决定究竟发生哪一个，但是对建模者来说，无法预料这种转换产生的后果。

完成转换。没有标明触发器事件的转换是由状态中的活动的完成引起的（即完成转换）。完成转换也可以带一个监护条件，这个监护条件是在状态中的活动完成时被赋值的（而不是完成以后）。

动作。当转换被引起时，它对应的动作被执行。动作是原子性的，一般是一个简短的计算处理过程，通常是一个赋值操作或算术计算。另外还有一些动作，包括给另一个对象发送消息、调用一个操作、设置返回值、创建和销毁对象，没有被定义的控制动作用外部语言来进行详细说明。动作也可以是一个动作序列，即一系列简单的动作。动作或动作序列的执行不会被同时发生的其他动作影响或终止。按照 UML中的概念，动作的执行时间非常短，与外界事件所经历的时间相比是可以忽略的，因此，在动作的执行过程中不能再插入其他事件。然而，实际上任何动作的执行都要耗费一定时间，新到来的事件必须被安置在一个队列中。

整个系统可以在同一时间执行多个动作。我们说动作是原子性的，并不是说整个系统是原子性的。系统能够处理硬件的中断和多个动作的时间共享。动作在它的控制线程中是原子性的。一旦开始执行，它必须执行到底并且不能与同时处于活动状态的动作发生交互作用。但动作不能用于表达处理过程很长的事物。与系统处理外部事件所需要的反应时间相比，动作的执行过程应该很简洁，否则系统不能够做到实时响应。

动作可以使用触发器事件的参数和对象的属性值作为表达式的一部分。

表6-3列出了各种动作及描述。

表6-3 动作的种类

动作种类	描述	语法
赋值	对一个变量赋值	*target*:=*expression*
调用	调用对目标对象的一个操作；等待操作执行结束；可能有一个返回值	*opname*(*arg*, *arg*)
创建	创建一个新对象	new *Cname*(*arg*, *arg*)
销毁	销毁一个对象	*object*. destroy()
返回	为调用者指定返回值	return *value*
发送	创建一个信号实例并将这个信号发送到目标对象或一组目标对象	*sname*(*arg*, *arg*)
终止	对象的自我销毁	terminate
不可中断	用语言说明的动作，如条件和迭代	[语言说明]

状态改变。当动作执行完毕后，转换的目标状态被激活，这时会触发出口动作或入口动作的执行。

嵌套状态。状态可以被嵌套在其他的组成状态之内（看下一段）。从一个外部状态出发的转换可以应用于这个状态所有的内部嵌套状态。任何一个内部嵌套状态被激活时，转换都有

China-pub.com
下载

可能发生。组成状态可用于表达例外和异常，因为组成状态上的转换适用于所有它所嵌套的状态，不需要每个嵌套状态显式地单独处理异常。

入口和出口动作。一个跨越多个嵌套层次的转换可能会离开或进入某个状态。只要转换进入或离开某个状态，则该状态可能包含要被执行的动作。进入一个状态可能会执行一个依附于该状态的入口动作。如果转换离开初始状态，那么在转换的动作和新状态的入口动作被执行前，执行该状态的出口动作。

入口动作通常用来进行状态所需要的内部初始化。因为不能回避一个入口动作，任何状态内的动作在执行前都可以假定状态的初始化工作已经完成，不需要考虑如何进入这个状态。同样，无论何时从一个状态离开都要执行一个出口动作来进行善后处理工作。当出现代表错误情况的高层转换使嵌套状态异常终止时，出口动作特别有用。出口动作可以处理这种情况以使对象的状态保持前后一致。入口动作和出口动作原则上依附于进来的和出去的转换，但是将它们声明为特殊的动作可以使状态的定义不依赖状态的转换，因此起到封装的作用。

内部转换。内部转换有一个源状态但是没有目标状态。内部转换的激发规则和改变状态的转换的激发规则相同。由于内部转换没有目标状态，因此转换激发的结果不改变本状态。如果一个内部转换带有动作，它也要被执行，但是没有状态改变发生，因此也不需要执行入口和出口动作。内部转换用于对不改变状态的插入动作建立模型（如，记录发生的事件数目或建立帮助信息屏）。

尽管入口动作和出口动作的执行是由进入或离开某状态的外部转换所引起的，除了使用保留字entry 和exit 代替触发事件名称之外，入口和出口动作使用与内部转换相同的表示法。

一个自身转移会激发状态上的入口动作和出口动作的执行（从概念上来讲，自身转换从一个状态出发后又会到自身状态），因此，自身转换不等价于内部转换。图 6-4 说明了入口动作、出口动作和内部转换。

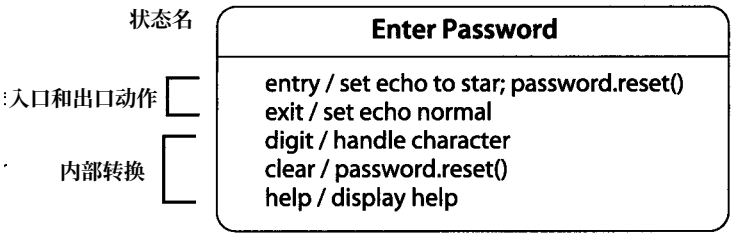


图6-4 内部转换、入口和出口动作

6.6 组成状态

一个简单状态没有子结构，只带有一组转换和可能的入口和出口动作。组成状态是一个被分解成顺序的或并发的子状态的状态。表 6-4 列出了各种状态。

China-pub.com 下载

表6-4 状态的种类

状态种类	描 述	表示法
简单状态	没有子结构的状态	
并发组成状态	被分成两个或多个并发子状态的状态，当组成状态被激活时，所有的子状态均被并发激活	
顺序组成状态	包含一个或多个不连接的子状态的状态，特别是当组成状态被激活时，子状态也被激活	
初始状态	伪状态，仅表明这是进入状态机真实状态的起点	
终止状态	特殊状态，进入此状态表明完成了状态机的状态转换历程中的所有活动	
结合状态	状态，将两个转换连接成一次就可以完成的转换	
历史状态	伪状态，它的激活保存了组成状态中先前被激活的状态	H
子机器引用状态	引用子机器的状态，该子机器被隐式地插入子机器引用状态的位置	include S
桩状态	伪状态，用来在子机器引用状态中标识状态	T

将状态分解成互斥的子状态是对状态的一种专门化处理。一个外部状态被细分成多个内部子状态，每一个子状态都继承了外部状态的转换。在某一时间只有一个子状态处于激活状态。外部状态表达了每一个内部状态都具有的条件。

进入或离开一个组成状态的转换会引起入口动作或出口动作的执行。如果有多个组成状态，跨越多个层次的转换会引起多重入口动作（最外层最先执行）和出口动作（最内层最先执行）的执行。如果转换带有动作，那么这个动作在入口动作执行后，出口动作执行前执行。

组成状态也可能在其内部具有一个初始状态。组成状态边界上的转换隐含为初始状态上的转换。一个新对象起始于它的最外层的初始状态。如果一个对象到达了它最外层状态的终止状态，那么该对象将被销毁。初始状态、终止状态、入口动作和出口动作封装了状态的定义，使状态的定义与进出状态的转换无关。

China-pub.com
下载

图6-5展示了一个状态的顺序分解，其中包括一个初始状态。这是售票系统的状态机模型。

将一个状态分解成并发的多个子状态代表相互独立的并行处理过程。当进入一个并发超状态时，控制线程的数目增加；当离开一个并发超状态时，控制线程的数目减少。对于每一个子状态而言，并发通常依靠不同的对象实现，但是，并发子状态还可以代表一个单独状态内部的逻辑并发关系。图6-6展示了选修一门大学课程的并发分解。

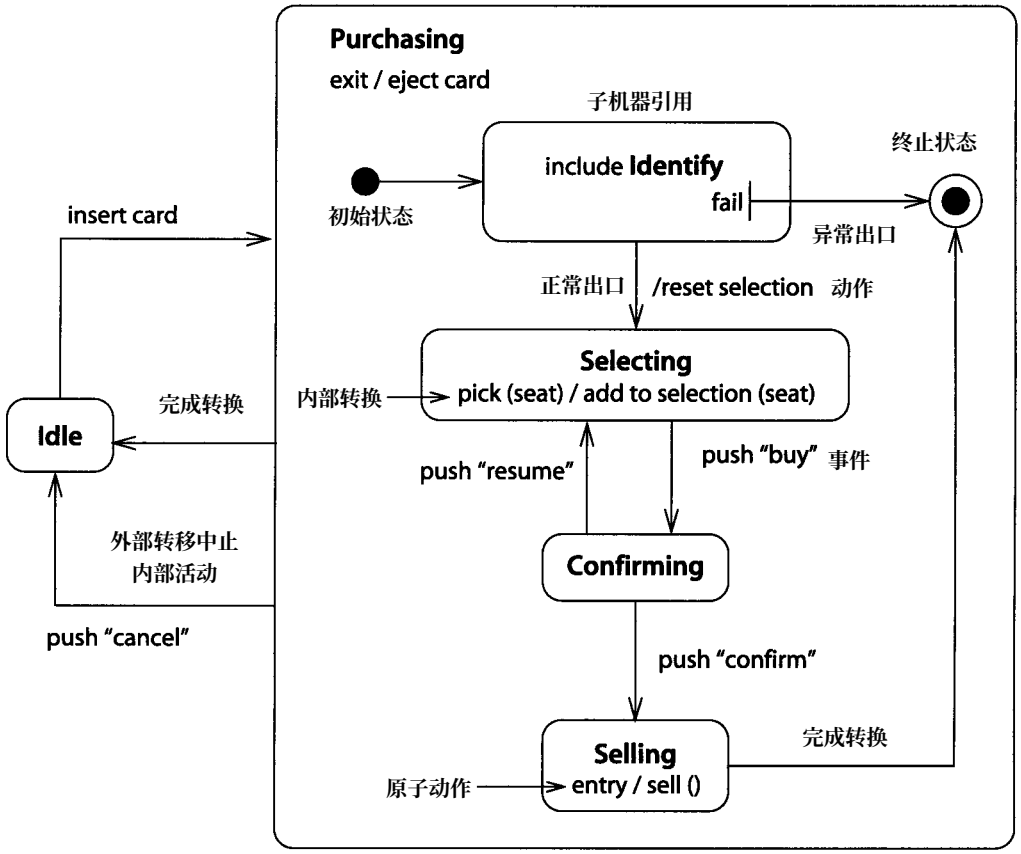


图6-5 状态机

通常，可以在一个状态机中使用其他状态机的一部分，这种可重用性会带来一些方便。状态机可以命名，并可以用名字引用其他一个或多个状态机。目标状态机是一个子机器，引用这个子机器的状态叫做子机器引用状态。它的含义是在需要引用的地方用一个状态机来替换原有内容。状态机可以包含一个活动，即可以包含一个处理过程或一个需要消耗时间才能完成的持续过程或是可以被中断的事件，而子机器不能。图 6-7演示了子机器的引用。

China-pub.com 下载

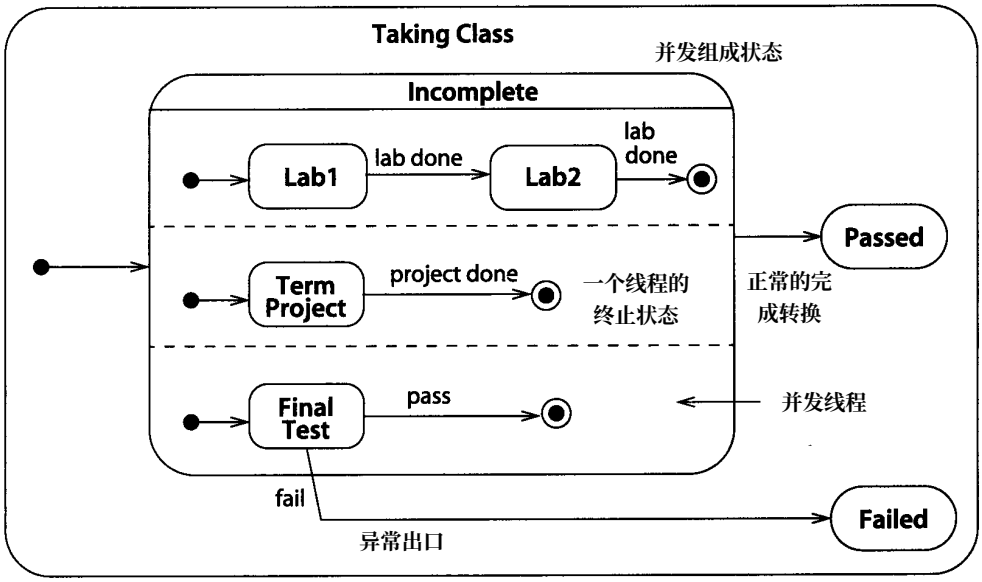


图6-6 带有并发组成状态的状态机

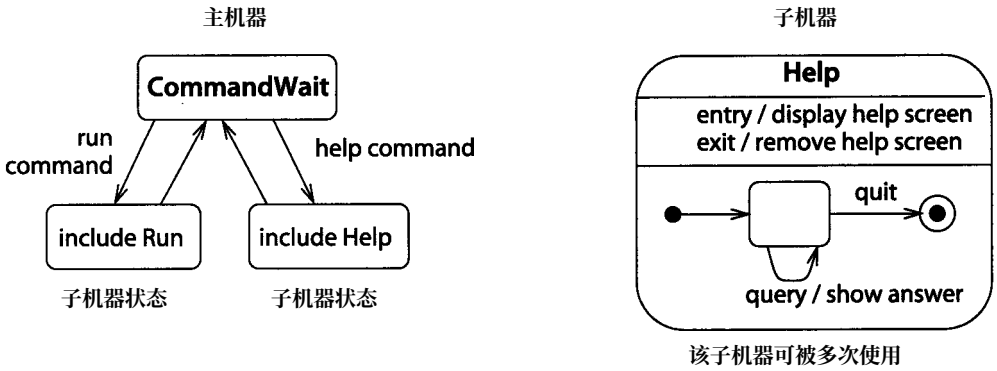


图6-7 子机器状态

进入一个子机器引用状态的转换会激活目标子机器的初始状态。要进入其他状态的子机器，需要在子机器引用状态中安置一个或多个桩状态。桩状态用于在子机器中标识状态。

China-pub.com
下载

第7章
活 动 视 图

7.1 概述

活动图是一种特殊形式的状态机，用于对计算流程和工作流程建模。活动图中的状态表示计算过程中所处的各种状态，而不是普通对象的状态。通常，活动图假定在整个计算处理的过程中没有外部事件引起的中断，否则，普通的状态机更适于描述这种情况。

活动图包含活动状态。活动状态表示过程中命令的执行或工作流程中活动的进行。与等待某一个事件发生的一般等待状态不同，活动状态等待计算处理工作的完成。当活动完成后，执行流程转入到活动图中的下一个活动状态。当一个活动的前导活动完成时，活动图中的完成转换被激发。活动状态通常没有明确表示出引起活动转换的事件，当转换出现闭包循环时，活动状态会异常终止。

活动图也可以包含动作状态，它与活动状态有些相似，但是它们是原子活动并且当它们处于活动状态时不允许发生转换。动作状态通常用于短的记帐操作。

活动图可以包含并发线程的分叉控制。并发线程表示能被系统中的不同对象和人并发执行的活动。通常并发源于聚集，在聚集关系中每个对象有着它们自己的线程，这些线程可并发执行。并发活动可以同时执行也可以顺序执行。活动图不仅能够表达顺序流程控制还能够表达并发流程控制，如果排除了这一点，活动图很像一个传统的流程图。

7.2 活动图

活动图是活动视图的表示法（如图 7-1）。它包括一些方便的速记符号，这些符号实际上可以用于任何状态图，尽管活动图和状态图的混合表示法多数时候都很难看。

活动状态表示成带有圆形边线的矩形框，它含有活动的描述（普通的状态盒为直边圆角）。简单的完成转换用箭头表示。分支表示转换的监护条件或具有多标记出口箭头的菱形。控制的分叉和结合与状态图中的表示法相同，是进入或离开深色同步条的多个箭头。图 7-1表示订单处理的活动图。

China-pub.com 下载

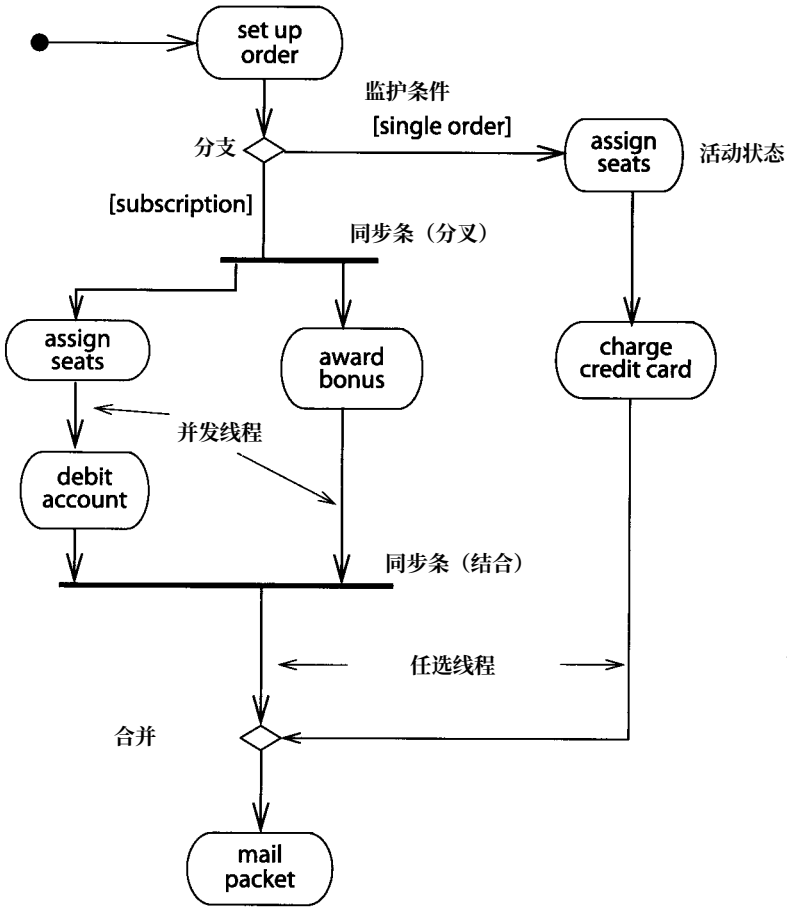


图7-1 活动图

为了表示外部事件必须被包含进来的情景，事件的接收可以被表示成转换的触发器或正在等待某信号的一个特殊内嵌符号。发送可同样表示。然而，如果有许多事件驱动的转换，那么用一个普通的状态图表示更可取。

泳道。将模型中的活动按照职责组织起来通常很有用。例如，可以将一个商业组织处理的所有活动组织起来。这种分配可以通过将活动组织成用线分开的不同区域来表示。由于它们的外观的缘故，这些区域被称作泳道。图 7-2表示了泳道。

对象流。活动图能表示对象的值流和控制流。对象流状态表示活动中输入或输出的对象。对输出值而言，虚线箭头从活动指向对象流状态。对输入值而言，虚线箭头从对象流状态指向活动。如果活动有多个输出值或后继控制流，那么箭头背向分叉符号。同样，多输入箭头指向结合符号。

China-pub.com
下载

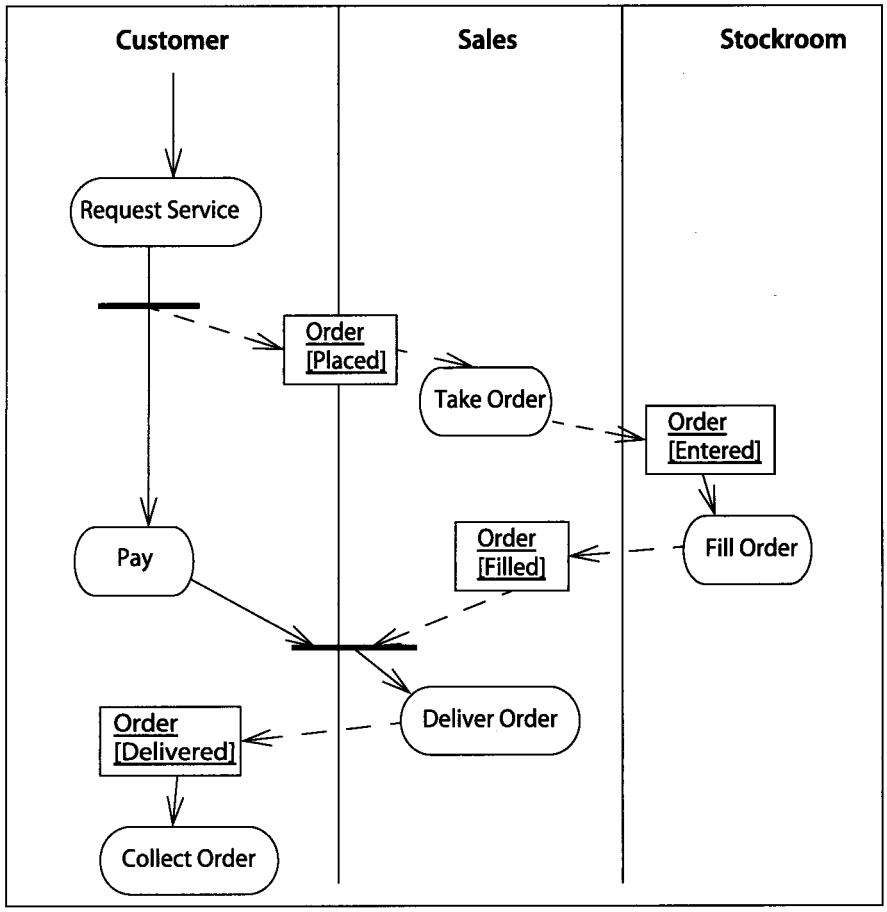


图7-2 泳道和对象流

图7-2表示一个活动和对象流状态都被分配到泳道中的活动图。

7.3 活动和其他视图

活动图没有表示出计算处理过程中的全部细节内容。它们表示了活动进行的流程但没表示出执行活动的对象。活动图是设计工作的起点。为了完成设计，每个活动必须扩展细分成一个或多个操作，每个操作被指定到具体类。这种分配的结果引出了用于实现活动图的对协作的设计工作。

China-pub.com
下载

第8章
交互视图

8.1 概述

对象间的相互作用体现了对象的行为。这种相互作用可以描述成两种互补的方式，一种以独立的对象为中心进行考察，另一种以互相作用的一组对象为中心进行考察。

状态机的描述范围不宽，但它描述了对象深层次的行为，是单独考察每一个对象的“微缩”视图。对状态机的说明是精确的并且可直接用于代码。然而，在理解系统的整个功能时存在困难，因为状态机一个时刻只集中描述一个对象，要确定整个系统的行为必需同时结合多个状态机进行考察。交互视图更适合于描述一组对象的整体行为。交互视图是对象间协作关系的模型。

8.2 协作

协作描述了在一定的语境中一组对象以及用以实现某些行为的这些对象间的相互作用。它描述了为实现某种目的而相互合作的“对象社会”。协作中有在运行时被对象和连接占用的槽。协作槽也叫做角色，因为它描述了协作中的对象或连接的目的。类元角色表示参与协作执行的对象的描述；关联角色表示参与协作执行的关联的描述。类元角色是在协作中被部分约束的类元；关联角色是在协作中被部分约束的关联。协作中的类元角色与关联角色之间的关系只在特定的语境中才有意义。通常，同样的关系不适用于协作外的潜在的类元和关联。

静态视图描述了类固有的内在属性。例如， Vehicle 需要有一个所有者。协作图描述了类实例的特性，因为它在协作中起特殊的作用，例如，在一个 RentalCar的协作中，rentalVehicle 需要 rentalDriver，它通常与交通工具不直接相关但它是协作的基本部分。

系统中的对象可以参与一个或多个协作。虽然协作的执行通过共享对象相连，但是对象所出现的协作不必直接相关。例如，在一个 Vacation模型中，某人可以既是rental Driver同时又是hotelGuest。不经常出现的情况是一个对象在同一个协作中可能担当多个角色。

China-pub.com
下载

协作包括结构和行为两个方面。结构方面与静态视图相似——包含一个角色集合和它们之间的关系，这些关系定义了行为方面的内容。行为方面是一个消息集合，这些消息在具有某一角色的各对象间进行传递交换。协作中的消息集合叫做交互。一个协作可以包含一个或多个交互，每个交互描述了一系列消息，协作中的对象为了达到目标交换这些消息。

状态机描述范围具有一定的局限性，但它的描述层次较深入，协作不受限制但描述层次较浅。它捕获了对象组成的网络结构中相互发送消息的整体行为。协作表示潜藏于计算过程中的三个主要结构的统一，即数据结构、控制流和数据流的统一。

8.3 交互

交互是协作中的一个消息集合，这些消息被类元角色通过关联角色交换。当协作在运行时，受类元角色约束的对象通过受关联角色约束的连接交换消息实例。交互作用可对操作的执行、用例或其他行为实体建模。

消息是两个对象之间的单路通信，从发送者到接收者的控制信息流。消息具有用于在对象间传值的参数。消息可以是信号（一种明确的、命名的、对象间的异步通信）或调用（具有返回控制机制的操作的同步调用）。

创建一个新的对象在模型中被表达成一个事件，这个事件由创建对象所引起并由该对象所在的类本身所接受。创建事件作为从顶层初始状态出发的转换的当前事件对于新实例是可行的。

消息可以被组织成顺序的控制线程。分离的线程代表并发的几个消息集合。线程间的同步通过不同线程间消息的约束建模。同步结构能够对分叉控制、结合控制和分支控制建模。

消息序列可以用两种图来表示：顺序图（突出消息的时间顺序）和协作图（突出交换消息的对象间的关系）。

8.4 顺序图

顺序图将交互关系表示为一个二维图。纵向是时间轴，时间沿竖线向下延伸。横向轴代表了在协作中各独立对象的类元角色。类元角色用生命线表示。当对象存在时，角色用一条虚线表示；当对象的过程处于激活状态时，生命线是一个双道线。

消息用从一个对象的生命线到另一个对象生命线的箭头表示。箭头以时间顺序在图中从上到下排列。

图8-1为带有异步消息的典型的顺序图。

China-pub.com 下载

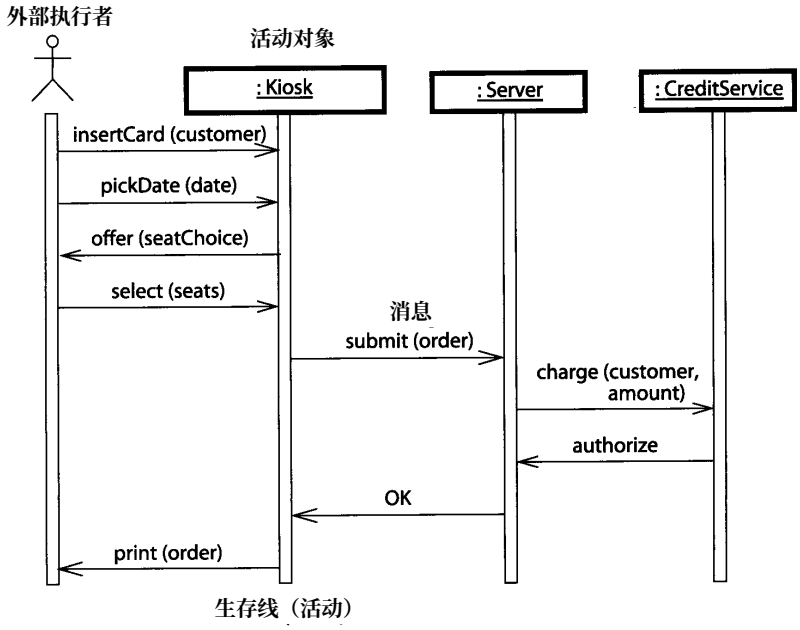


图8-1 顺序图

8.5 激活

激活是过程的执行，包括它等待嵌套过程执行的时间。在顺序图中它用部分替换生命线

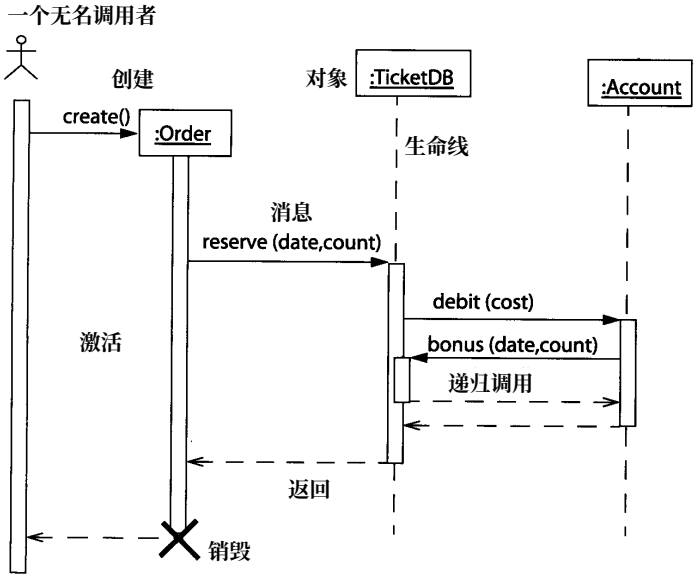


图8-2 带有激活的顺序图

China-pub.com
下载

的双道线表示。调用用指向由这个调用引起的激活的上部的箭头表示。当控制流程重新进入对象中的一个操作时，递归调用发生，但是第二个调用是与第一个调用分离的激活。同一个对象中的递归或嵌套调用用激活框的叠加表示。图 8-2为含有过程控制流的一个顺序图，包括一个递归调用和一个对象的创建。

主动对象是激活栈中一组激活对象中的根对象。每个主动对象有由它自己的事件驱动控制线程，控制线程与其他主动对象并行执行。被主动对象所调用的对象是被动对象。它们只在被调用时接受控制，而当它们返回时将控制放弃。

如果几个并行控制线程有它们自己的利用嵌套调用的过程控制流，那么不同的线程必须用不同的线程名、颜色或其他方式辨别，以避免当两个线程在同一个对象中产生混乱。通常，在一个单独的图中最好不要混合使用过程调用和信号。

8.6 协作图

协作图是一种类图，它包含类元角色和关联角色，而不仅仅是类元和关联。类元角色和关联角色描述了对象的配置和当一个协作的实例执行时可能出现的连接。当协作被实例化时，对象受限于类元角色，连接受限于关联角色。关联角色也可以被各种不同的临时连接所担当，如过程参量或局部过程变量。连接符号可以使用构造型表示临时连接（《parameter》或《local》）或调用同一个对象（《self》）。虽然整个系统中可能有其他的对象，但只有涉及到协作的对象才会被表示出来。换而言之，协作图只对相互间具有交互作用的对象和对象间的关联建模，而忽略了其他对象和关联。图 8-3为一个交互图。

可以将对象标识成四个组：存在于整个交互作用中的对象；在交互作用中创建的对象（使用约束{new}）；在交互作用中销毁的对象（使用约束 {destroyed}）；在交互作用中创建并销毁的对象（使用约束 {transient}）。设计时可以首先表示操作开始时可得的对象和连接，然后决定控制如何流向图中正确的对象去实现操作。

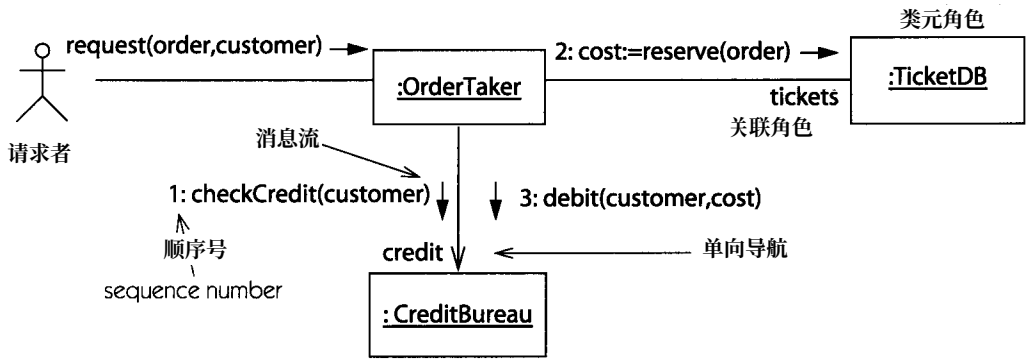


图8-3 协作图

虽然协作直接表现了操作的实现，它们也可以表示整个类的实现。在这种使用中，它表

China-pub.com 下载

示了用来实现类的所有操作的语境。这使得对象在不同的操作中可以担当多种角色。这种视图可以通过描述对象所有操作的协作的联合来构造。

消息。消息可以用依附于链的带标记的箭头表示。每个消息包括一个顺序号、一张可选的前任消息的表、一个可选的监护条件、一个名字和参量表、可选的返回值表。顺序号包括线程的名字（可选）。同一个线程内的所有消息按照顺序排列。除非有一个明显的顺序依赖关系，不同线程内的消息是并行的。各种实现的细节会被加入，如同步与异步消息的区别。

流。通常，在完整的操作中协作图包含对象的符号。然而，有时对象具有不同的状态并且必须弄明确表达出来。例如，一个对象可以改变位置，或者在不同的时刻它的关联有很大区别。对象可以用它的类与它所处的状态表示 —— 即具有状态类的对象。同一个对象可以表示多次，每次有不同的位置和状态。代表同一对象的不同对象符号可以用变成流联系起来。变成流是从一个对象状态到另一个的转换。它用带有构造型《become》的箭头表示，并且可以用顺序号标记表示它何时出现（如图 8-4）。变成流也可以用来表示一个对象从一个位置到另一个位置的迁移。

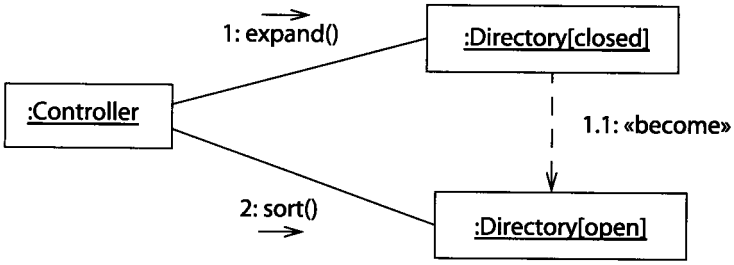


图8-4 变成流

构造型《copy》不经常出现，它表示通过拷贝另一个对象值而得到的一个对象值。

表8-1表示了几种对象流的关系。

表8-1 流关系的种类

流	功 能	表 示 法
变成	从一个对象值变化到另一个对象值	《become》
拷贝	拷贝一个对象，从此以后，该对象为独立对象	《copy》

协作图与顺序图。协作图和顺序图都表示出了对象间的交互作用，但是它们侧重点不同。顺序图清楚地表示了交互作用中的时间顺序，但没有明确表示对象间的关系。协作图清楚地表示了对象间的关系，但时间顺序必须从顺序号获得。顺序图常常用于表示方案，而协作图用于过程的详细设计。

China-pub.com
下载

8.7 模板

模板是一个参数化的协作，并有表示何时使用该协作的标线。参数可以被不同的值替代从而产生不同的协作。参数通常为类指定槽。当模板实例化时，它的参数受限于类图中的实际类或受限于更大的协作中的角色。

模板用一个虚线椭圆表示，椭圆用标记有角色名字的虚线与每个类联系。例如，图 8-5表示摘自[Gamma-95]的Observer模板的使用。在这个模板的使用中， CallQueue替代Subject角色，SlidingBarIcon替代handler角色。

模板可以出现在分析、结构设计、详细设计和具体实现等不同层次中，这是重用经常出现的结构的一种方法。图8-5表示了Observer模板的使用。

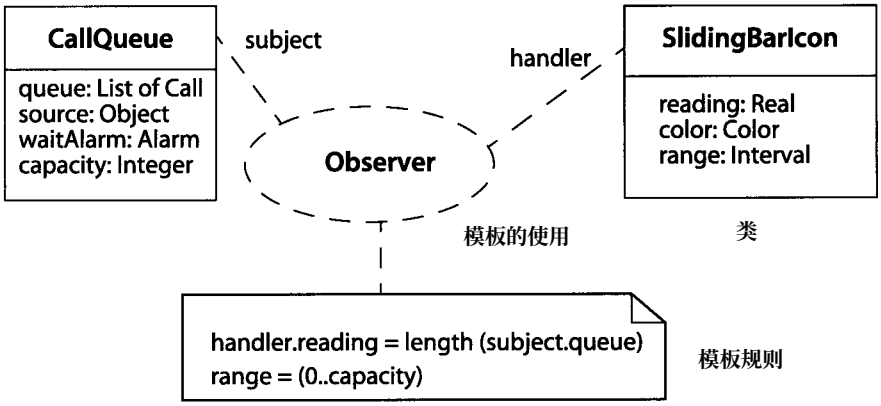


图8-5 模板的使用

China-pub.com
下载

第9章
物 理 视 图

9.1 概述

系统模型的大部分内容反映了系统的逻辑和设计方面的信息，并且独立于系统的最终实现单元。然而，为了可重用性和可操作性的目的，系统实现方面的信息也很重要。 UML使用两种视图来表示实现单元：实现视图和部署视图。

实现视图将系统中可重用的块包装成具有可替代性的物理单元，这些单元被称为构件。实现视图用构件及构件间的接口和依赖关系来表示设计元素（例如类）的具体实现。构件是系统高层的可重用的组成部件。

部署视图表示运行时的计算资源（如计算机及它们之间的连接）的物理布置。这些运行资源被称作节点。在运行时，节点包含构件和对象。构件和对象的分配可以是静态的，它们也可以在节点间迁移。如果含有依赖关系的构件实例放置在不同节点上，部署视图可以展示出执行过程中的瓶颈。

9.2 构件

构件是定义了良好接口的物理实现单元，它是系统中可替换的部分。每个构件体现了系统设计中特定类的实现。良好定义的构件不直接依赖于其他构件而依赖于构件所支持的接口。在这种情况下，系统中的一个构件可以被支持正确接口的其他构件所替代。

构件具有它们支持的接口和需要从其他构件得到的接口。接口是被软件或硬件所支持的一个操作集。通过使用命名的接口，可以避免在系统的各个构件之间直接发生依赖关系，有利于新构件的替换。构件视图展示了构件间相互依赖的网络结构。构件视图可以表示成两种形式，一种是含有依赖关系的可用构件（构件库）的集合，它是构造系统的物理组织单元。它也可以表示为一个配置好的系统，用来建造它的构件已被选出。在这种形式中，每个构件与给它提供服务的其他构件连接，这些连接必须与构件的接口要求相符合。

构件用一边有两个小矩形的一个长方形表示，它可以用实线与代表构件接口的圆圈相连（如图9-1）。

China-pub.com
下载

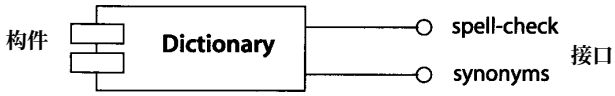


图9-1 带接口的构件

构件图表示了构件之间的依赖关系（如图9-2）。每个构件实现（支持）一些接口，并使用另一些接口。如果构件间的依赖关系与接口有关，那么构件可以被具有同样接口的其他构件替代。

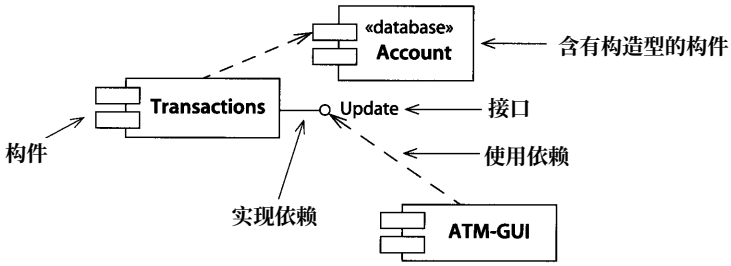


图9-2 构件图

9.3 节点

节点是表示计算资源的运行时的物理对象，通常具有内存和处理能力。节点可能具有用来辨别各种资源的构造型，如CPU、设备和内存等。节点可以包含对象和构件实例。

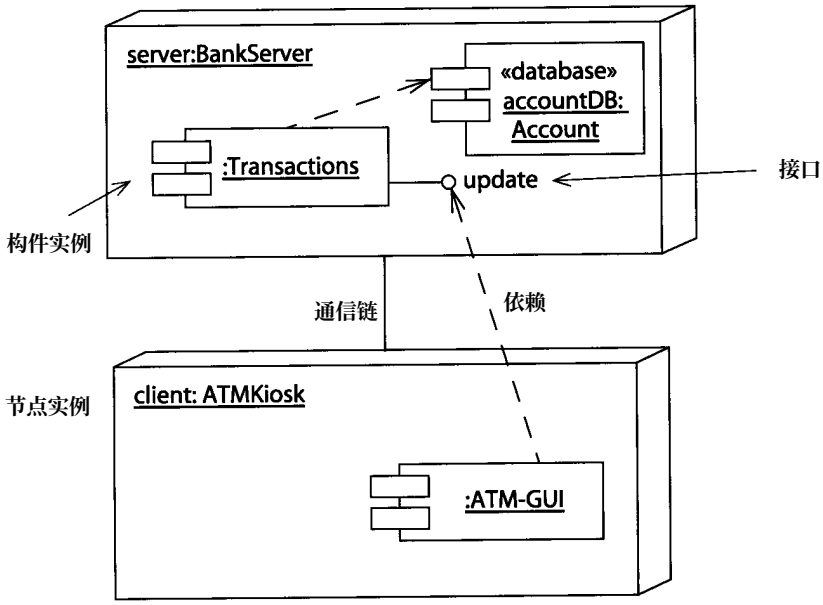


图9-3 部署图

China-pub.com 下载

节点用带有节点名称的立方体表示，可以具有分类（可选）（如图9-3）。

节点间的关联代表通信路径。关联有用来辨别不同路径的构造型。

节点也有泛化关系，将节点的一般描述与具体的特例联系起来。

对象在节点内的存在用嵌套在节点符号内的对象符号来表示。如果这样的表示不方便，对象符号可以包含表示它所在节点名称的 location标签。节点间对象或构件实例的迁移也可以表示出来。

见第13章“ become(变成) ”。

China-pub.com
下载

第10章
模型管理视图

10.1 概述

任何大的系统都必须被分成几个小的单元，这使得人们可以一次只处理有限的信息，并且分别处理这些信息的工作组之间不会互相干扰。模型管理由包及包之间的依赖关系组成。

10.2 包

包是模型的一部分，模型的每一部分必须属于某个包。建模者可以将模型的内容分配到包中。但是为了使其能够工作，分配必须遵循一些合理的原则，如公用规则、紧密耦合的实现和公用观点等。UML对如何组包并不强制使用什么规则，但是良好的解组会很大地增强模型的可维护性。

包包含顶层的模型元素，即任何不被其他元素所包含的元素，如类和它们之间的关系、状态机、用例图、交互和协作。有些元素如属性、操作、状态、生命线和消息被其他元素包含，而不在包中直接出现。每个顶层元素都有一个包，它在这个包中被声明，该包被称作元素的“家”包。元素的内容可能被别的包引用，但是其所有权属于家包。在一个配置好的控制系统中，建模者必须能够对家包进行访问以修改元素的内容，这为处理大的模型提供了访问控制机制。包也是任何版本出版机制的单元。

一个包可以包含其他的包，根包可间接地包含系统的整个模型。组织系统中的包有几种可能的方式，可以用视图、功能或建模者选择的其他基本原则来规划包。包是 UML模型中一般的层次组织单元，它们可以被用来进行存储、访问控制、配置管理和构造可重用模型部件库。

如果包的规划比较合理，那么它们能够反映系统的高层构架 —— 有关系统由子系统和它们之间的依赖关系组合而成。包之间的依赖关系概述了包的内容之间的依赖关系。

10.3 包间的依赖关系

依赖关系在独立元素之间出现，但是在任何规模的系统中，应从更高的层次观察它们。

China-pub.com 下载

包之间的依赖关系概述了包中元素的依赖关系，即包间的依赖关系可从独立元素间的依赖关系导出。

包间依赖关系的存在表示存在一个自底向上的方法（一个存在声明），或允许过后存在于一个自顶向下的方法（限制其他任何关系的约束）中，对应的包中至少有一个独立元素之间给定种类的依赖关系的关系元素。这是一个“存在声明”，并不意味着包中的所有元素都具有依赖关系。这对建模者来说是表明存在更进一步的信息的标志，但是包层依赖关系本身并不包含任何更深的信息，它仅仅是一个概要。

自顶向下方法反映了系统的整个结构，自底向上方法可以从独立元素自动生成。在建模中两种方法有它们自己的地位，即使是在单个的系统中也是这样。

独立元素之间属于同一类别的多个依赖关系被聚集到包间的一个独立的包层依赖关系中，独立元素包含在这些包中。如果独立元素之间的依赖关系包含构造型（如几种不同的使用），为了产生单一的高层依赖关系，包层依赖关系中的构造型可能被忽略。

包用附有标签的矩形表示，依赖关系用虚线箭头表示。

图10-1显示了订票系统的包结构图。外部包与 Seat selection 的两个变更之间存在依赖关系，子系统的任何一个实现都将只包括其中一个变。

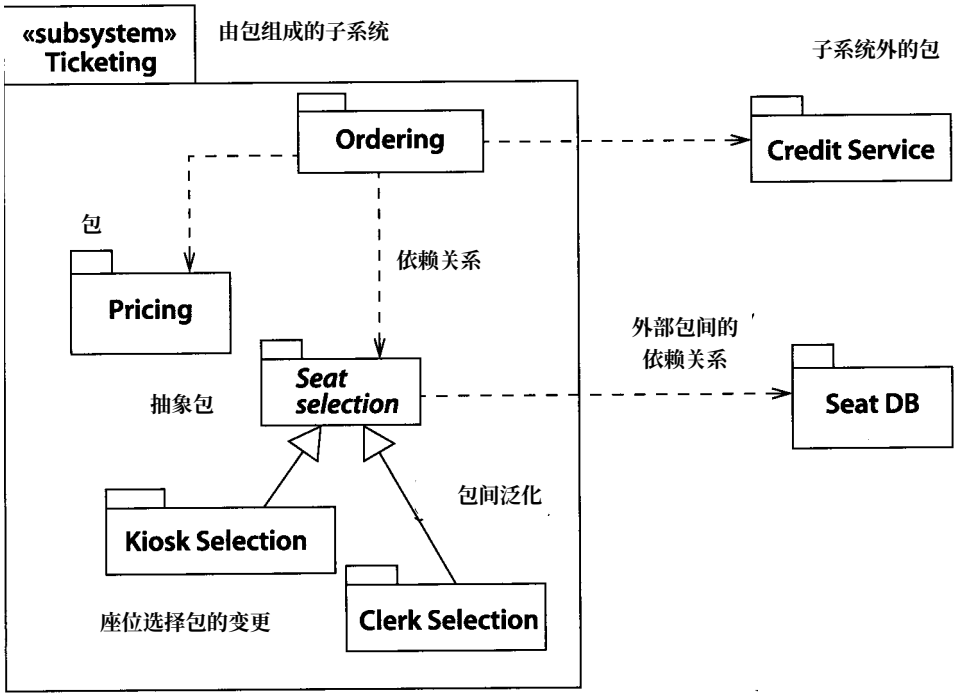


图10-1 包和包间的关系

China-pub.com
下载

10.4 访问与引入依赖关系

通常，一个包不能访问另一个包的内容。包是不透明的，除非它们被访问或引入依赖关系才能打开。访问依赖关系直接应用到包和其他包容器中。在包层，访问依赖关系表示提供者包的内容可被客户包中的元素或嵌入于客户包中的子包所引用。提供者中的元素在它的包中要有足够的可见性，使得客户可以看到它。通常，一个包只能看到其他包中被指定为具有公共可见性的元素。具有受保护可见性的元素只对包含它的包的后代包具有可见性。可见性也可用于类的内容（属性和操作）。一个类的后代可以看到它的祖先中具有公共或受保护可见性的成员，而其他的类则只能看到具有公共可见性的成员。对于引用一个元素而言，访问许可和正确的可见性都是必须的。所以，如果一个包中的元素要看到不相关的另一个包的元素，则第一个包必须访问或引入第二个包，且目标元素在第二个包中必须有公共可见性。

嵌套在另一个包中的包是包容器的一部分，而且可以完全访问包容器的内容。然而，对包容器来说，如果不访问嵌套包，则不能看到嵌套包的内部，其内容被封装了起来。

请注意访问依赖关系不改变客户的命名空间或以任何其他方式自动建立引用，它仅仅授予建立引用的权限。引入依赖关系用来将名字加入到客户包的命名空间中，以作为路径名的别名。

10.5 模型和子系统

模型是从某一个视角观察到的对系统进行完全描述的包。它从一个视点提供一个系统的封闭的描述。它对其他包没有很强的依赖关系，如实现依赖或继承依赖。跟踪关系表示某些连接的存在，是不同模型的元素之间的一种较弱形式的依赖关系，它不用特殊的语义说明。

通常，模型为树形结构。根包包含了存在于它体内的嵌套包，嵌套包组成了从给定观点出发的系统的所有细节。

子系统是具有单独说明和实现部分的包。它表示具有对系统其他部分存在干净接口的连贯模型单元，通常表示按照一定的功能要求或实现要求对系统进行的划分。模型和子系统都用具有构造型关键字的包表示（如图 10-1）。

China-pub.com
下载

第11章
扩展机制

11.1 概述

UML提供了几种扩展机制，允许建模者在不用改变基本建模语言的情况下做一些通用的扩展。这些扩展机制已经被设计好，以便于在不需理解全部语义的情况下就可以存储和使用。由于这个原因，扩展可以作为字符串存储和使用。对不支持扩展机制的工具来说，扩展只是一个字符串，它可以作为模型的一部分被导入、存储，还可以被传递到其他工具。我们期望后端工具设计成能够处理各种扩展，这些工具会为它们需要理解的扩展定义特定的语法和语义。

这种扩展的方法很可能不能满足出现的多种要求，但是它以一种易于实现的简单方式容纳建模者对UML裁制的大部分要求。

扩展机制包括约束、标记值和构造型。

一定要记住扩展是违反UML的标准形式的，并且使用它们会导致相互影响。在使用扩展机制之前，建模者应该仔细权衡它的好处和代价，特别是当现有机制能够合理工作时。典型地，扩展用于特定的应用域或编程环境，但是它们导致了 UML方言的出现，包括所有方言的优点和缺点。

11.2 约束

约束是用文字表达式表示的语义限制。每个表达式有一种隐含的解释语言，这种语言可以是正式的数学符号，如集合论表示法；或是一种基于计算机的约束语言，如 OCL；或是一种编程语言，如 C++；或是伪代码或非正式的自然语言。当然，如果这种语言是非正式的，那么它的解释也是非正式的，并且要由人来解释。即使约束由一种正式语言来表示，也不意味着它自动为有效约束。

约束可以表示不能用UML表示法来表示的约束和关系。当陈述全局条件或影响许多元素的条件时约束特别有用。

约束用大括弧内的字符串表达式表示。约束可以附加在表元素、依赖关系或注释上。图11-1表示了几种约束。

China-pub.com
下载

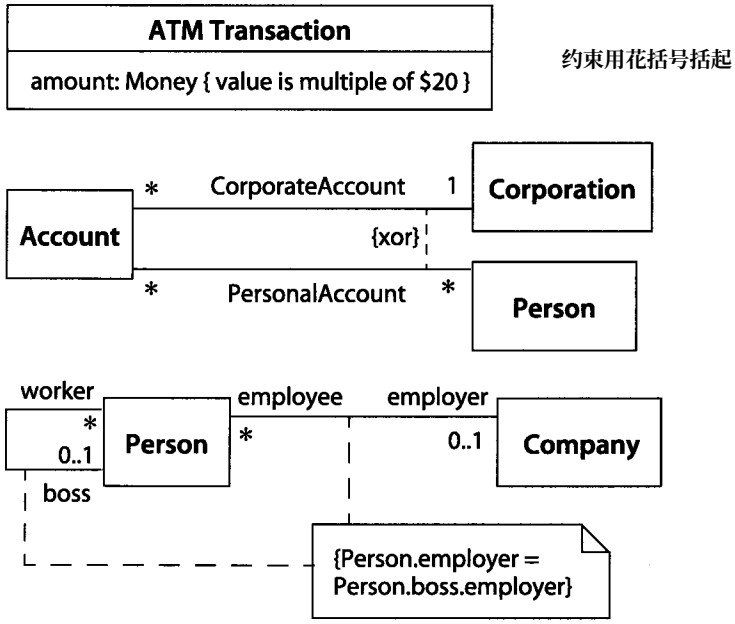


图11-1 约束

11.3 标记值

标记值是一对字符串——一个标记字符串和一个值字符串——存储着有关元素的一些信息。标记值可以与任何独立元素相关，包括模型元素和表达元素。标记是建模者想要记录的一些特性的名字，而值是给定元素的特性的值。例如，标记可以是 author，而值是对元素负责的人的名字，如Charles Babbage。

标记值可以用来存储元素的任意信息，对于存储项目管理信息尤其有用的，如元素的创建日期、开发状态、截止日期和测试状态。除了内部元模型属性名称外，任何字符串可以作为标记名（这是因为标记和属性在一起会被认为是一个元素的属性并且可以被工具一起访问），而一些标记名已经被预定义了（见第 14章）。

标记值还提供了一种方式将独立于实现的附加信息与元素联系起来。例如，代码生成器需要有关代码种类的附加信息以从模型中生成代码。通常，有几种方式可以用来正确地实现模型，建模者必须提供做出何种选择的指导。有些标记可以用做标志告诉代码生成器使用哪种实现方式。其他标记可为加入工具使用，如项目计划生成器和报表书写器。

标记值也可以用来存储有关构造型模型元素的信息（我们将在下面讨论）。

标记值用字符串表示，字符串有标记名、等号和值。它们被规则地放置在大括弧内（如图11-2）。在图表中标记值经常被省略，只显示在下拉表格中。

China-pub.com 下载

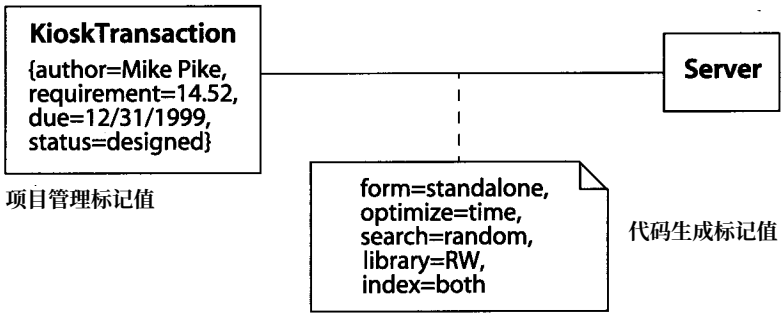


图11-2 标记值

11.4 构造型

许多建模者希望为了一种特定的应用域裁制一种建模语言，这会带来一些风险，因为被裁制的语言不易普遍为人理解，但人们仍然试图这么做。

构造型是在一个已定义的模型元素的基础上构造的一种新的模型元素。构造型的信息内容和形式与已存在的基本模型元素相同，但是含义和使用不同。例如，商业建模领域的建模者希望将商业对象和商业过程作为特殊的建模元素区别开来，这些元素的使用在特定的开发过程中是不同的。它们可以被看作特殊的类——它们有属性和操作，但是在它们与其他元素的关系上和它们的使用上有特殊的约束。

构造型建立在已存在的模型元素基础上，构造型元素的信息内容与已存在的模型元素相同。这样便可允许工具以相同的方式存储和使用新元素和已存在的元素。构造型元素可以有它自己的区别符号，并且这很容易由工具所支持。例如，一个“商业组织”可以有一个看起来像一组人的图标。构造型也可以有一组适用于它的使用的约束。例如，一个“商业组织”可能只能与另一个“商业组织”，而不能与任何其他类联合。不是所有的约束都能被多用途工具自动地确定，但是它们可以被用手动执行或被理解构造型的加入工具确定。

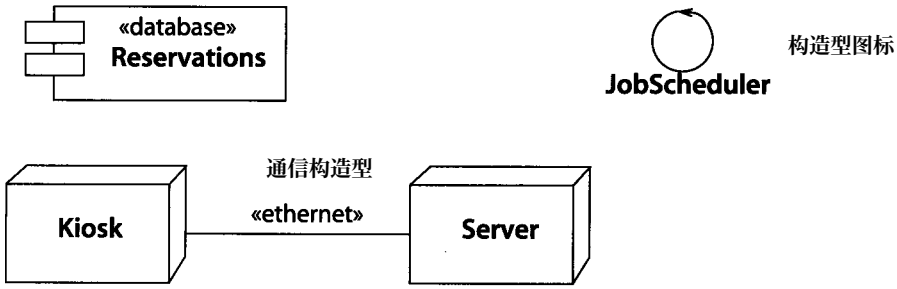


图11-3 构造型

China-pub.com
下载

构造型可以用标记值来存储不被基本模型元素所支持的附加特性。

构造型用双尖括号内的文字字符串表示（如图 11-3)，它可以放在表示基本模型元素的符号的里边或旁边。建模者也可以为特殊的构造型创建一个符号，这个符号替代了原来的基本模型元素的符号。

11.5 裁制UML

约束的扩展机制、标记值和构造型使得为了特殊的应用域而裁制 UML轮廓成为可能。已经形成了几种轮廓，其描述见附录 C。此外用户还提出了其他几种。这种裁制建模语言的能力意味着应用域的用户可以使建模语言适应应用域的需要，还能够共享在所有领域中通用的概念。

China-pub.com
下载

第12章 UML 环境

12.1 概述

UML 模型被用在环境中。多数人使用建模技术为了达到一个目的，即为了开发性能优良的系统，而不是为了使用模型本身。模型的目的和对模型的解释也受环境之外的因素影响。在广阔的外部环境中，另一些工具包括：跨越多种语言的元模型、模型编辑工具、程序设计语言、操作系统和主系统构件以及那些使用系统的商业和工程背景。确定模型的意义和使用目的取决于所有这些工具，其中也包括 UML语言。

模型在不同的具体层次中出现。 UML 是一种通用建模语言，包括语义和表示法，适用于不同的工具和实现语言。应用中的每个层次需要使用不同层次的 UML建模思路。

12.2 语义职责

元模型是对模型的一种描述。建模语言描述模型，因此，它可以用元模型描述。元模型试图通过定义语义使语言精确，但是为适应新情况它必须允许扩展。元模型的实际形式对于用工具实现模型和模型的互换很重要，但多数用户并不关心它。因此，我们在本书中未涉及到它。对此有兴趣的读者可以参阅配套光盘上的原始标准文献 [UML-98]。

元模型和语言必须能够覆盖很多的背景信息并具有很强的解释力。现存系统有不同的执行和存储模型，从中选择一种作为正确的解释是不可能的。实际上，甚至考虑这样的选择都可能产生误导。我们可以将执行模型的不同解释作为语义变更点。语义变更点是一个随执行的具体语义而不同的点，但它与系统的其他方面无关。例如，一个环境可以选择或不选择支持动态类元，即对象具有在运行时改变所属类的能力。现在，许多程序设计语言不允许这么做，主要是因为程序设计语言的实现方面的原因，但有些语言实现了这一功能。在静态语义中这种区别是不能辨认的。选择静态分类还是动态分类可以用一个具有两个选项（静态分类或动态分类）的语义变更点来标明。当这些选择存在时，人们经常辩论哪一种是正确的解释。相反，如果明确了这仅仅只是一种选择，并给选项起一个名字，这样任何一种选择都可以使用。

China-pub.com
下载

元模型描述一个结构良好的模型的内容，正如一种程序设计语言描述一个结构良好的程序一样。只有结构良好的模型才有意义和恰当的语义；询问一个结构糟糕的模型的意义是没有意义的。然而，多数开发中的模型的结构是不完善的，它们不完整并有可能相互矛盾。但是模型编辑工具必须支持不完整的模型，而不仅只支持完整的模型。 UML元模型描述正确的、结构完善的模型。分离的元模型能描述可能的模型片段。我们让工具开发者决定在哪里划定支持模型片段的界限和支持结构不完善的模型用哪种语义。

UML包含一些内置的扩展机制以适应特殊领域的应用。这些机制包括定义构造型和标记值的能力。通过定义一组构造型和标记值并采用相应的使用约定，这些机制可以用于裁制UML的变体。例如，可以开发出以不同程序设计语言的执行语义为核心的 UML变体语言。使用扩展机制会很有效，但也会带来潜在的危险。因为这些语义不是在 UML中定义的，UML不支持它们的含义，这种解释取决于建模者。此外，如果你不注意，某些含义可能是二义性的甚至是矛盾的。建模工具能够自动支持由工具定义的构造型和标记值，但不支持用户自定义的扩展。不论是否支持扩展，任何扩展的使用都会使用户偏离语言标准所提供标准定义，并损害模型的互换性和易理解性。当然，使用特殊的类库也会偏离所谓的最完美的互换性，所以不用担心这些。当扩展有帮助时就使用，但当扩展不必要时则避免使用它。

12.3 表示法职责

表示法并不是给模型增加含义，而是帮助用户理解模型的含义。表示法没有语义，但它经常为用户加入一些内在涵义，如在一张图中基于相近意义的两个概念的密切联系。

UML文档[UML-98]和本书定义了一套规范的 UML 表示法，可以称作模型的出版格式。这和许多程序设计语言类似，把期刊文章中的程序印刷成有吸引力的格式，包括仔细的规划、用黑体表示的保留字，且每个过程用不同的图形来说明。而实际的编译者不得不接受较为凌乱的输入。我们期望编辑工具能把表示法扩展成屏幕格式，包括诸如使用不同的字体和颜色加亮条目，简单地删除和过滤不需要使用的条目的能力，放大图像以展示图中的嵌套元素的能力，通过超文本热链转入其他模型和视图的能力，以及动画功能。试图将所有这些可能的项目都标准化是不可能的，也是愚蠢的，因为没有这个必要，而且这样还会限制有益的创新。这种表示法的扩展能力是工具制造者的工作。在交互式工具中，产生二义性的危险不大，因为用户总能找到一个明确的解释。这也许比坚持一种粗看上去没有任何二义性的表示法更有用。这个观点是说：当需要时，一个工具必须能生成规范化的表示法，特别是在印刷格式中，但在使用交互式工具中也应该采用合理的扩展。

我们仍期望工具能让用户有限制但又有效地扩展表示法。我们已经规定构型可以有自己的图标。对其他的表示法的扩展也是允许的，但用户要谨慎使用这些扩展机制。

注意，表示法不仅仅是图片，它还包括基于文本格式的信息和元素中间不可见的超链接。

China-pub.com 下载

12.4 程序设计语言职责

UML 必须在没有与不同的实现语言明确地合并时与它们共同使用。我们认为 UML应该允许任何程序设计语言（至少是多数）的使用，包括规格说明和目标代码生成。问题是每种程序设计语言都存在许多我们不想吸收到 UML中的语义，因为它们作为程序设计语言来说很好控制，而在执行语义中有相当大的变化。例如，并发的语义在不同的语言中存在不同的处理方法（如果能够处理这些语义）。

UML中没有详细地描述简单数据类型，这么做是经过深思熟虑的，因为我们不想把我们偏爱的一种程序设计语言的语义合并到其他所有语言中。对于多数的建模目的，这不是一个问题。应使用适合于你的目标语言的语义模型。这是语义变更点的一个例子。

详尽的语言实现特性的表示使得在不把实现特性的语义内置到 UML的情况下，带来了捕获其信息的问题。我们的方法是通过构造型和标记值捕获超越了 UML内置能力的语言特性，这些可以由工具或代码生成器分配给语言给性和代码生成选项。一般的编辑器不需要理解它们。事实上，用户可以用一个不支持目标语言的工具创建一个模型，然后把最终模型转换到适于最终处理的工具。当然，如果工具不能理解构造型和标记值，那么它不能对它们进行一致性检查。但这并不比用文本编辑器和编译器差。如果必要，可以创建一个工具以使用 UML语言的一组特定的扩展。

在可预知的未来，代码生成和逆向工程不仅需要 UML模型，还需要设计者的输入信息。代码生成器需要的指导和提示可以由标记值和构造型提供。例如，建模者可以指定哪种包容器类用于实现一个关联。当然，这种工具中的代码生成设置方法也许是矛盾的，但目前没有一种标准化的实际设置方法。目前不同的工具均使用对自己有竞争优势的代码生成器，但最终将会出现缺省设置并发展为成熟的标准。

12.5 使用建模工具建模

在实际的系统中模型需要工具支持，工具提供了观察和编辑模型的交互方式。工具提供了一层超出UML自身作用域的组织，可以帮助用户理解并获得信息。通过搜索和过滤已经存在的资料，工具有助于在大型模型中查找信息。

12.5.1 工具问题

工具处理模型的物理组织和存储。它必须支持一个项目的若干工作组同时工作，以及支持跨越多个项目的重用。以下几点问题超出了规范的 UML 的作用域，但在运用实际工具中必须予以考虑。

二义性和未详尽说明的信息。在初期阶段，许多事物不能用语言表达。工具必须能够调整模型的精确性并且不能强迫每个值都要进行详细说明。可参看以下两小节。

China-pub.com
下载

表示选项。用户不想在任何时候都看到所有的信息。工具必须能够过滤和隐藏那些某一时间不需要的信息。工具还要通过显示器硬件的功能提供交替的可视化支持。这一点已经在12.3节讲过了。

模型管理。模型单元的配置控制、访问控制和版本超出了 UML的作用域，但是它们对于软件工程过程十分重要，并且位于元模型的上层。

与其他工具的接口。模型需要由代码生成器、规格计算器、报表书写器、执行引擎和其他后台工具处理。其他工具所需要的信息要包含到模型中，但这不是 UML信息。标记值适合保存这些信息。

12.5.2 工作进展过程中产生的不一致模型

建模的最终目标是生成一定细化层次的系统描述。最后的模型必须满足不同的有效性约束才有意义。但是，正如许多创造性的过程一样，结果不必以线性方式产生，中间产品不必每一步都满足所有的有效性约束。实际上，一个工具不仅要处理语义上满足有效性约束的模型，还要处理在句法上有效的模型，这些模型满足一定的构造规则但可能会违背一些有效性约束。语义上无效的模型不能直接使用。相反，它们可以看作是通向最终目标的进展中的工作。

12.5.3 空值和未详细说明的值

一个完整的模型包含了它的所有元素的所有属性值。在许多情况下空值（无值）是一种可能出现的值，一个值是否可能为空是属性类型描述的一部分。例如，空值对集合大小的上限没有意义。有的集合有固定的大小，有的则没有固定的上限值，在这种情况下，集合的大小是无限的，所以具有空值与否实际上取决于一种数据类型的可能值的范围。

另一方面，在设计的早期，开发者也许没有注意到特殊特性的值。在特定的阶段，这个值可能没有意义，如建立领域模型时的可视性。或者，这个值有含义但建模者还未详细说明它，开发者应当记住它仍然需要细化。这种情况下，这个值是没有详尽说明的，这表示一个值最终是需要的但目前还没有被详细说明。它和空值不同，空值在最终模型里是合法值。许多情况下，特别是字符串中，空值是表示一个未被详尽说明的值的好方法，但并非全是这样。未被详细说明的值在结构完善的模型中是无意义的， UML定义不处理没有详细说明的值。这是支持UML的工具的职责,也是处于进展中的没有语义的工作模型的一部分。

China-pub.com
下载

第三部分

参考资料

China-pub.com
下载

第13章
术语大全

1. abstract(抽象)

抽象是不能被直接实例化的类、用例、信号、其他类元或其他泛化元素，它也可以用来描述没有实现的操作。反义词：concrete。

见abstract operation，generalizable element。

语义

抽象类是不可被实例化的类，即它可以没有直接实例，这既可能是因为它的描述是不完整的（如缺少一个或多个操作的方法），也可能是因为即使它的描述是完整的它也不想被实例化。抽象类是为了以后说明。抽象类必须有可能含有实例的后代才能使用，一个抽象的叶类是没用的（它可以作为叶在框架中出现，但是最终它必须被说明）。

具体类可以没有任何抽象操作（否则，它必为抽象的），但是抽象类可以有具体操作。具体操作是可以被实现一次并在所有子类中不变地使用的操作。在它们的实现中，具体操作可以只使用声明它们的类所知道的特征（属性和操作）。继承的目的之一即将这些操作在抽象的超类中分解以使得它们可以被所有的子类分享。一个具体操作可以是多态的，即后代类中的方法优先于它，但是它也可以不必是多态的，它可以是一个叶操作。一个所有操作都被实现的类可以是抽象的，但是这一点必须被明确声明。一个有着一个或多个未实现操作的类自然是抽象的。

同样的语义也适用于用例。抽象的用例定义了行为的片断，这个行为自己不能出现，但是它可以通过泛化、包括或扩展关系在具体用例的定义中出现。通过在抽象的用例中分解公用的行为，模型变得更小和易于理解。

类似的关系也存在于其他的类元和泛化元素中。

表示法

抽象类或抽象操作的名字用斜体表示。关键字 abstract可以放置在位于名称下面或后面的特性表中，如Account{abstract}。

China-pub.com 下载

见class name。

示例

图13-1表示一个抽象类 Account，它有一个抽象操作 computeinterest和一个具体操作 deposite。两个具体子类已经被声明了。因为子类是具体的，所以它们每一个必须实现操作 computeinterest。属性总是具体的。

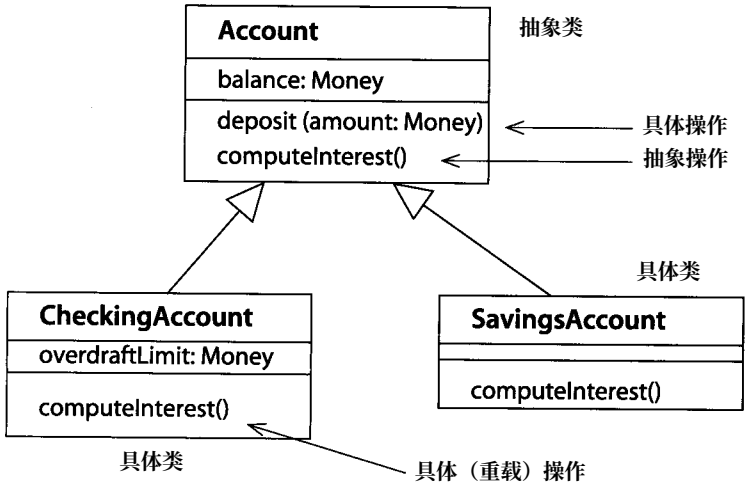


图13-1 抽象和具体的类

讨论

将一个类建模成抽象的或具体的，其间的差别并不像它第一次出现时那么清晰和基本。它更像有关模型的设计结果而不是继承特性。在设计的发展过程中，类的状态可能发生变化。如果将列举出所有可能性的子类加入到具体类中，那么这个具体类可以建模成抽象的。如果子类之间的差别被认为是不必要的且被删除，或者这些差别用属性值而不是用不同的子类表示，那么这个抽象类可以建模成具体的。

简化决定的方法之一是采纳以下设计准则：所有的非叶类必须是抽象的（除了某些为了以后说明的抽象叶类外，所有的叶类必须是具体的）。这并不是UML的规则，它既可以被采用也可以不被采用，设计这个“抽象超类”规则的原因是超类上可继承的方法和具体类上的方法经常有不同的需求，这种需求并不能被单个方法很好地实现。超类中的方法被迫做两种事：定义能被所有后代观察到的通用例子和为特定类实现通用例子。但是这两个目标经常发生冲突。相反，一个非抽象的超类能被机械地分离到一个抽象的超类和一个具体的叶子类中。抽象的超类包含被所有子类继承的方法；具体的子类包含所有特定的可实例化类要求的方法。在抽象的超类规则后也允许在保持特定具体类型的变量或参数与保持着超类的任何后代的变

China-pub.com
下载

量或参数之间存在完全的区别。

在图13-2中，考虑类Letter的声明，它并没有遵循抽象超类规则。该类有一个getNextSentence操作，它返回下一个还没有读的句子的明文，还有resetCursor操作，它将鼠标置回开始处。而子类EncryptedLetter表示已经被加密的字母。操作getNextSentence被重载因为明文在被返回前必须要解密。操作的实现完全不同。因为Letter是一个具体超类，所以要从普通Letter类或EncryptedLetter子类中分辨出普通Letter类(非重载)的参数是不可能的。

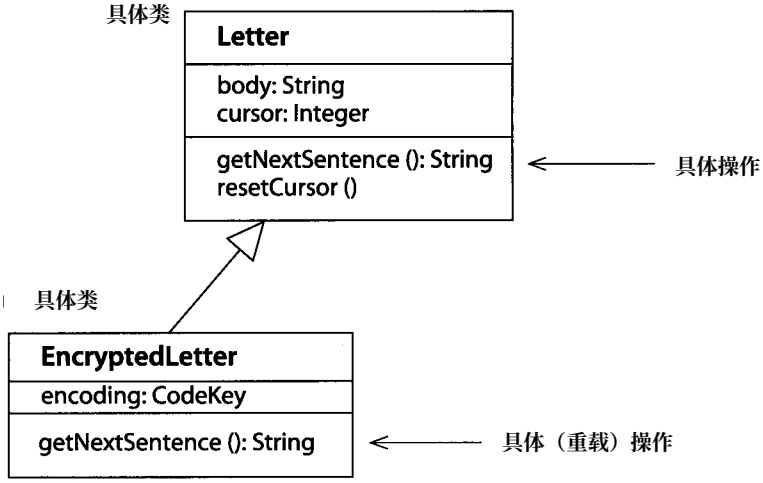


图13-2 具体的超类产生的模糊性

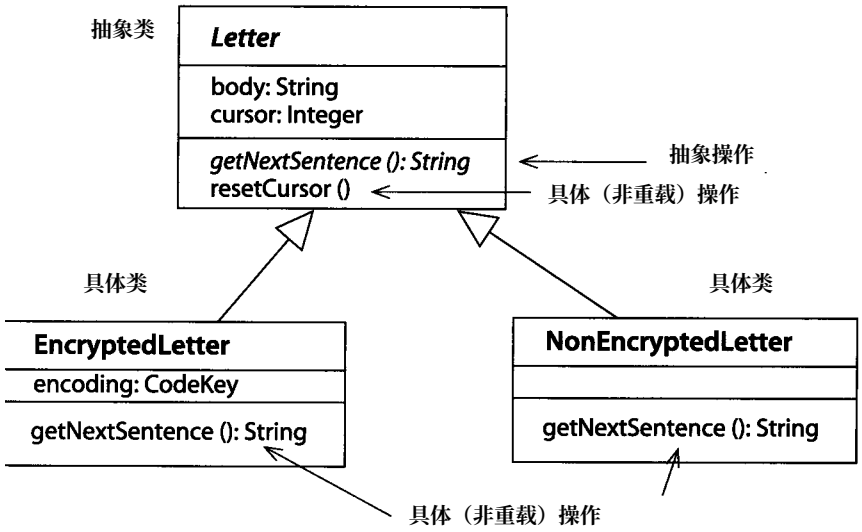


图13-3 抽象超类避免模糊性

China-pub.com 下载

抽象超类方法用于辨别抽象类 Letter（它既可能是加密的字母，也可能是未加密的字母）并且加入类 NonEncrypedLetter以表示具体例子，如图 13-3。在这个例子中，getNextSentence是一个被每个子类实现的抽象操作，resetCursor是一个在所有子类中相同的具体操作。这个模型是对称的。

如果遵循抽象类规则，那么抽象类声明能从类层次中自动确定，并且在图中表示它也是多余的。

声明一个抽象叶类一般是没用的，但有一个例外：当抽象类作为一组全局类作用域的属性和操作的通用命名空间时可以被声明。这种情况较少，大部分用于处理非面向对象语言的编程时，建议用户在大多数情况下不要用它。全局值通过引用全局依赖关系违反了面向对象设计的精神。单实例类可以以更扩展的方式提供同样的功能（可参见 [Gamma-95]）。

2. abstract class(抽象类)

抽象类是可能不会被实例化的类。

见abstract。

语义

抽象类可能没有直接实例，可能有间接实例（通过它的具体后代）。

见 abstract 的讨论。

3. abstract operation(抽象操作)

抽象操作缺少实现，即它只有说明而没有方法。实现必须被具体后代类补充。

见abstract，generalizable，inheritance，polymorphic。

语义

如果一个操作在类中被声明为抽象的，那么该操作缺少在类中的实现，且类本身也必须是抽象的。操作的实现必须由具体的后代来满足。如果类继承了一个操作的实现但是将操作声明是抽象的，那么抽象的声明就使类中被继承的方法无效。如果一个操作在类中被声明是具体的，那么类必须满足或继承从祖先那里得到的实现，这个实现可能是一个方法或调用事件。如果操作在类中根本没有被声明，那么类继承从它的祖先那里得到的操作声明和实现。

操作可以作为方法或由调用事件触发的状态机转化而实现。每个类可以声明它自己的方法、操作的调用事件或者继承祖先的定义。

China-pub.com
下载

表示法

抽象操作的名称用斜体字表示，如图 13-4，而关键字abstract可以放在操作特征标记后的特性表里。

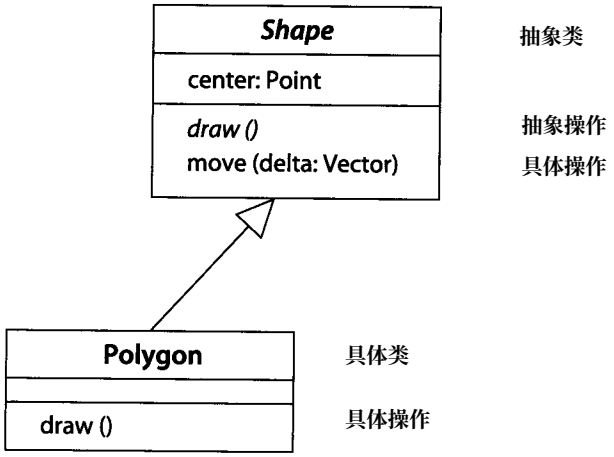


图13-4 抽象操作和类

讨论

继承概念的最大用途是支持能被每个具体后代类有区别地实现的抽象操作。抽象操作允许调用者在不知道哪个对象的类是目标的情况下使用操作（假设目标对象通过作为一个抽象类的间接实例支持这个操作，并且这个抽象类有抽象操作的声明）。这种多态操作的重要性在于，决定对象种类的职责从调用者转换到了继承机制。不仅调用者不会有编写例子声明的麻烦和代价，并且调用者也不必关心抽象类的哪个子类会存在，这意味着附加子类将与新的操作实现一起被加入。因此，抽象操作、多态和继承在不必改变调用普通行为代码的情况下，通过加入新的对象和行为促进系统的升级。这大大减少了系统升级的时间，更重要的是，它降低了偶然的不协调的可能性。

4. abstraction(抽象)

抽象是确认一件事物本质特征的行为，这种行为将这个事物与其他所有事物区分开来。抽象涉及到通过观察几组事物的本质公共特性来查找它们的共同点。抽象往往涉及到观察者的观点和目的；不同的目的导致同一事情的不同抽象。所用的建模过程都涉及到抽象，通常存在与不同目的的不同层次上。

抽象是一种将不同层次上的同一概念的两种元素联系起来的依赖关系。

China-pub.com 下载

见derivation，realization，refinement，trace。

语义

抽象依赖关系是不同抽象层上的两个元素之间的关系，比如在不同模型中、不同准确度上、不同具体性上或不同优化层中的描述。通常，两个描述不会被同时用到。正常情况下，一个元素比另一个更详细一些，客户元素比提供者元素更详细。如果不明确哪个元素更详细，那么两个元素都可以建模成客户。

抽象依赖关系的构造型是跟踪（关键字trace)、精化（关键字refine)、实现（关键字realize）和导出（关键字derive)。

表示法

抽象依赖关系表示成从客户元素指向提供者元素的箭头，并附有关键字《trace》、《refine》或《derive》。实现依赖关系有它自己特殊的表示符号，表示成指向提供者元素的有着封闭三角形的虚线箭头。

元素之间的映射可以作为约束附加在关系上。

标准元素

derive，refine，trace。

5. access(访问)

访问是一种许可依赖关系，允许一个包引用另一个包中的元素。

见friend，import，visibility。

语义

一个包（客户）如果要引用另一个包（提供者）内的元素，那么它必须引入这样一个包，该包包括客户包到提供者包的《access》或《import》依赖关系上的元素。一个包可以隐含地获得对由包含该包的任何包所引入的包的访问权（即，嵌套包可以看到包含包可以看到的一切)。

包中的元素可以访问包内所有可见的元素。可见性规则可以总结如下：

- 一个包中定义的元素在同一个包中是可见的。
- 如果一个元素在一个包中是可见的，那么它对所有嵌套在这个包中的所有包都是可见的。
- 如果一个包访问或引入另一个包，那么在要被引入或访问的包中定义为公共可见性的

China-pub.com
下载

元素对引入包都是可见的。

■ 如果一个包是另一个包的孩子，那么所有在父包中定义为公共的或受保护的可见性的元素对子包是可见的。

■ 访问或引入依赖关系是不能传递的，如果 A能看到B，且B能看到C，这并不意味着A能看到C。

结论：除非一个包能够访问它的嵌套包且嵌套包的内容是公共可见的，否则这个包不能看到它自己的嵌套包的内部。

下面是有关可见性的更深一步的规则：

■ 如果一个类元的内容，如它的属性和操作以及嵌套类，在类元中具有公共可见性，那么它们在包中是可见的。请注意一个子系统的未组织的内容是由上面提到的包规则指导的，但是任何子系统本身的属性或操作由这条规则指导。

■ 如果一个类元的内容具有公共的或受保护的可见性，那么它们对后代类元是可见的。

■ 一个类元的所有内容对类元内的元素都是可见的，包括类元的方法或状态机中的元素。

一般情况下都会涉及到对等包中的元素。在这种情况下，一个元素能看到它自己包内的所有元素和被它所在包引入的包的具有公共可见性的所有元素。一个类可以看到其他类中的公共特征。一个类也可以看到它的祖先的受保护的特征。

表示法

访问依赖关系用一个从客户包指向提供者包的虚箭头表示。箭头用关键字《 access》作为标号。

讨论

图13-5为一个两个包间的对等层访问的例子。包 P能够访问包Q，但是包Q不能访问包P。P包中的类K和L能看到包Q中的类M，但是它们看不到私有类 N。除了具有公共可见性的类 K之外，类M和N看不到包P中的任何类，因为包Q不能访问包P。要想一个类对对等包是可见的，这个类必须具有公共可见性，并且它的包必须被对等包访问或引入。

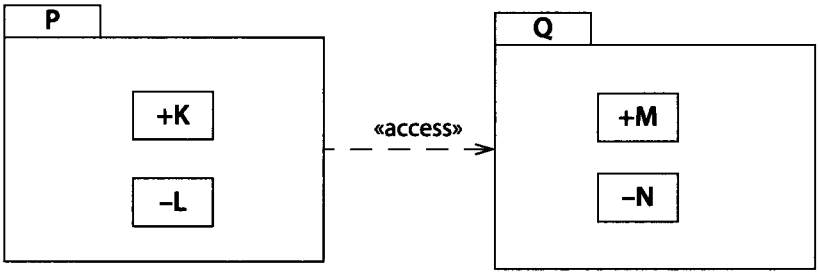


图13-5 对等访问

China-pub.com 下载

图13-6为一个有关可见性和访问声明的更复杂的例子。元素名字前的符号代表了它的可见性：+代表公共的，#代表是受保护的（只对后代可见），-代表是私有的（对外界是不可见的）。

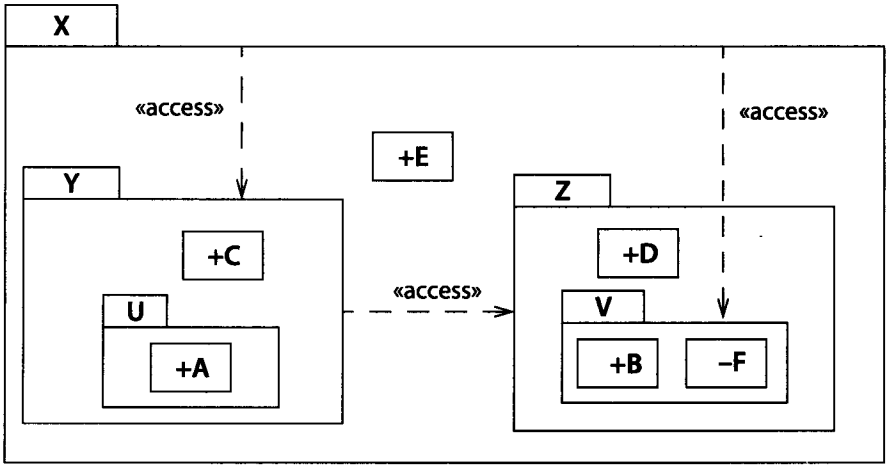


图13-6 访问规则

类A能看到C和E，因为它们包含在包Y和X内。

类C和A能够看到D，因为包Y引入了包Z。类A嵌套在包Y中并且能够看到Y能看到的一切。

类A、C和E能看到B，因为它们嵌套在包X中，而X引入了包含B的包V，但是，它们看不到F，因为F在它的包V中具有私有可见性。所以，类F在包V外是看不到的。

类E看不到D，因为D在包Z中，而Z并没有被包X访问。

类C和E都看不到A。类A在包U中，U没有被任何包访问。

类B和F能够看到类D和E，D和E建立在包含包中。它们也能看到C，C在包Y中，而包Y被包含包访问。虽然F是私有的，但这并不影响它看到其他的类，而其他的类看不到F。

类B和F能够互相看到，因为它们在同一个包中。类F对外面的包中的类是私有的，但对它自己包中的类不是这样。

6. action(动作)

动作是可执行的原子计算，它导致模型状态的改变和返回值。对比：activity。

见entry action，exit action，transition。

语义

动作是一个原子计算，即它不会从外界中断。动作可以附属于状态机中的转换（在两个

China-pub.com
下载

状态之间或在一个状态中）或交互中的某一步。通常，它是一个简单的或近似简单的系统状态的操作，常常位于一个独立对象的状态上。典型动作有分配属性值、访问属性或链值、创建新的对象或链、简单算法和向别的对象发送信号。动作是行为建立之外的步骤，动作的意思是指“快速”计算，以使得系统的反应时间不会被减少。系统可以同时执行几个动作，但是动作的执行应该是独立的。

动作也可以附属于转换，当转换被激发时动作被执行。它们也可以作为状态的入口动作和出口动作出现。这些动作由进入或离开状态的转换触发。所有动作都是原子的，即它们执行时完全不会被别的动作所干扰。

活动也是一种计算，但是它可以有内部结构并且会被外部事件的转换中断，所以活动只能附属于状态中，不能附属于转换。与动作不同，虽然活动自己也能中断，但是如果外界不中断它，它可以无限期地持续下去。而动作则不能从外部中断并且可以附属于转换或状态的入口或出口，而非状态本身。

结构

动作包括一个目标对象集合、一个对将要被发送的信号或将要被执行的动作的引用（即请求）、一张参量值表和一个可选的用于指明迭代的递归表达式。

对象集合。对象集合表达式产生一个对象的集合。在许多情况下，这个集合包含一个独立的固定的对象。有给定参量表的消息拷贝被同时发送到集合中的每个对象，即广播到每个对象。每个目标独立接收和处理消息的单独的实例。如果集合是空的，那么什么都不会发生。

请求。指明一个信号或声明一个操作。信号被发送到对象，操作被调用（对于具有返回值的操作，对象集合必须包含一个对象）。

参量表。参量列表。当赋值的时候，参量表中的值必须与信号或操作的参数相一致。参量被作为发送或调用的一部分。

循环。一个迭代表达式，说明需要执行多少次动作，并指定迭代变量（可选）。这个表达式也可以描述一个条件动作（即进行一次或不进行）。

几种动作

赋值动作。赋值动作将一个对象的属性值设置为给定值。该动作包含一个对目标对象的表达式、对象属性的名字和一个被分配到对象内属性槽的值的表达式。

调用动作。调用动作导致一个对象上操作的发生，即该对象上操作的调用。该动作包含一个消息名、一张参量表达式的表和一个目标对象集合表达式。目标可能是一个对象集合。在这种情况下，调用同时发生并且操作不会有返回值。如果操作有了返回值，那么它必须有一个对象作为目标。

调用动作是同步的。调用者在再次接收控制之前等待被调用操作的完成。如果操作被作为调用事件实现,那么调用者在再次接收控制之前一直等待，直到接收者执行被调用触发的转换。如果操作的执行返回值，那么调用者在它再次接收到控制时接受这些值。

China-pub.com 下载

创建动作。创建动作导致了对象的实例化和初始化（见 creation）。该动作包含一个对类的引用和一个可选的带有参量表的类作用域操作。动作的执行创建了一个类的新的实例，其属性值从计算它们的初始值表达式中得到。如果一个明确的创建动作被给定，那么它将被执行。操作常常会用创建动作的参量值覆盖属性值的初始值。

销毁动作。销毁动作导致目标对象的销毁，该动作有一个针对对象的表达式。销毁动作没有其他的参量。执行该动作的结果是销毁对象以及到它的所有链及所有组成部分（见 composition）。

返回动作。返回动作导致了一个到操作调用者的控制转换。该动作只允许在被调用使用的操作中存在。该动作包含一个可选的返回值表，当调用者接收到控制时该表对调用者是有效的。如果包含的操作被异步使用，那么调用者必须明确地选择返回消息（作为一个信号），否则它将会遗失。

发送动作。发送动作创建了一个信号实例并且用通过计算动作中的参量表达式得到的自变量初始化这个信号实例。信号被送到对象集合里的对象，这些对象通过计算动作中的目标表达式而得到。每一个对象接收它自己的信号拷贝。发送者保持它自己的控制线程和收益，且发送信号是异步的。该动作包含信号的名称、一张信号参量表达式表和对目标对象的对象集合表达式。

如果对象集合被遗漏，那么信号被发送到由信号和系统配置决定的一个或多个对象。例如，一个异常被发送到由系统策略决定的包含作用域。

终止动作。终止动作引起某种对象的销毁，这个对象拥有包含该动作的状态机，即该动作是一种“自杀”行为。其他对象会对对象的销毁事件做出反应。

无解释动作。无解释动作，一种控制构造或其他构造的动作。

表示法

UML没有一种固定的动作语言，它希望建模者使用一种实际的编程语言去编写动作。下面对OCL的改编是为了编写动作伪代码，但这并不是标准的一部分。

赋值动作

target：=*expression*

调用动作

object-set.operation-name(*argument*$_{list}$，)

创建动作

new *class-name*(*argument*$_{list}$，)

销毁动作

object.destroy()

返回动作

return *expression*$_{list}$,

发送动作

China-pub.com
下载

object-set.signal-name(*argument*$_{list}$,)

终止动作

terminate

无解释动作

if (*expression*) then (*action*) else (*action*)

如果需要明确区别调用与发送，关键字call和send可以作为表达式的前缀，它们是可选的。

讨论

UML 规格说明定义了一组动作，同时也说明在实际实现中可以用支持工具加入其他动作。这个决定是为了平衡精确性需求和开发者使用多种具有很广泛语义概念的目标语言的需要。编程语言的执行语义的变体要比数据结构和有用的控制构造集合中的变体多。不考虑理论上的可行性，微小的区别很难在语言中与实际的方式对应上。选择一种编程语言作为动作语言的基础会降低别的语言的作用，且动作的语义会留下一些不完整性和二义性，而这也是我们所不希望的。如果想使语义更准确些， UML必须与一种动作语言相结合。有些批评者说因为UML太自由了，所以它并不精确，但是它的不精确程度不会超过它所选择的动作语言的不精确程度。UML的真正缺陷是它没有实现动作和其他表达式的多语言表示，但是在今天的多语言世界上这几乎是不可能的。

7. action expression(动作表达式)

动作表达式是决定动作或动作序列的表达式。

讨论

UML并没有说明一个动作表达式的语法结构，由支持工具来负责这项工作。我们希望不同的使用者可以用编程语言、伪符号，甚至自然语言来表示动作。更精确的语义需要详细的设计，这也是许多用户会用到实际编程语言的地方。

8. action sequence(动作序列)

动作序列是被连续执行的一组动作，它是动作的一种。

语义

动作序列是一组动作，这些动作被连续地执行。整个序列被看作是一个原子单元，即它不能被中断。动作序列即一种动作，故也可以被附加到转换和交互步骤中。

China-pub.com 下载

表示法

动作序列用一个字符串表示，包含一系列由分号分隔开的动作。

如果动作用一种特定的编程语言表示，那么动作对声明序列的语法可以被替代使用。

示例

count：=0；reservations.clear（）；send kiosk.firstScreen()

9. action state(动作状态)

动作状态的用途是执行动作并转换到另一个状态。

见activity state，completion transition。

语义

动作状态的目的是执行一个入口动作，在这之后进行向另一个状态的完成转换。动作状态是原子的，即它不能被外部事件的转换中断。从理论上讲，它表示一个可以在忽略不计的时间内完成并不与同时发生的其他动作相互作用的计算。而实际上，它需要时间来执行，但是时间要比可能发生事件需要的反应时间要短。它不能有由事件触发的转换。动作状态没有子结构、内部转换或内部活动。它是一种哑状态，可用于把状态机组织成逻辑结构。它通常有一个输出的完成转换。如果有监护条件，则可能有多种输出的完成转换，所以也代表了一个分支。

表示法

动作状态并没有特殊的表示符号。它可以表示为具有入口动作的原始状态，也可以表示为一个活动状态。

10. activation(激活)

激活是操作的执行。激活（也叫做控制期）表示一个对象直接地或通过从属操作完成操作的过程。它对执行的持续时间和执行与其调用者之间的控制关系进行建模。在传统的计算机和语言上，激活对应栈帧的值。

见call，sequence diagram。

语义

激活是执行某个操作的实例，它包括这个操作调用其他从属操作的过程（见 call）。其内

China-pub.com
下载

容包括一个只能访问激活的局部变量的集合、一个方法内的当前位置（或其他行为描述）和一个表示调用内容的激活的引用（当当前激活终止时，它将恢复控制）。没有返回引用的激活必定是在一个主动类对象的状态机上的转换的结果，当转换完成时，状态机简单地等待下一个事件。

请注意这个定义将一个普通过程描述成可以在一个典型的冯·诺依曼机上实现。但是它以一种普通的方式表示，即它也适用于分布环境，该环境中没有共享内存并且在不同的内存空间中堆栈帧包含一张被链接的激活表。

表示法

激活在顺序图中用一个细长的矩形框表示，它的顶端与激活时间对齐而底端与完成时间对齐。被执行的操作根据不同风格表示成一个附在激活符号旁或在左边空白处的文字标号。进入消息的符号也可表示操作。在这种情况下，激活上的标号可以被忽略。如果控制流是过程性的，那么激活符号的顶部位于用来激发活动的进入消息箭头的头部，而符号的底部位于返回消息箭头的尾部。

如果有多个对象的并发活动，那么每个激活表示一个并发对象的执行。如果对象之间不进行通信，那么并发激活是独立的且它们的相对执行次数是不相关的。

在过程代码中，激活表示一段持续时间，在这段时间中过程或者被初始过程调用的从属

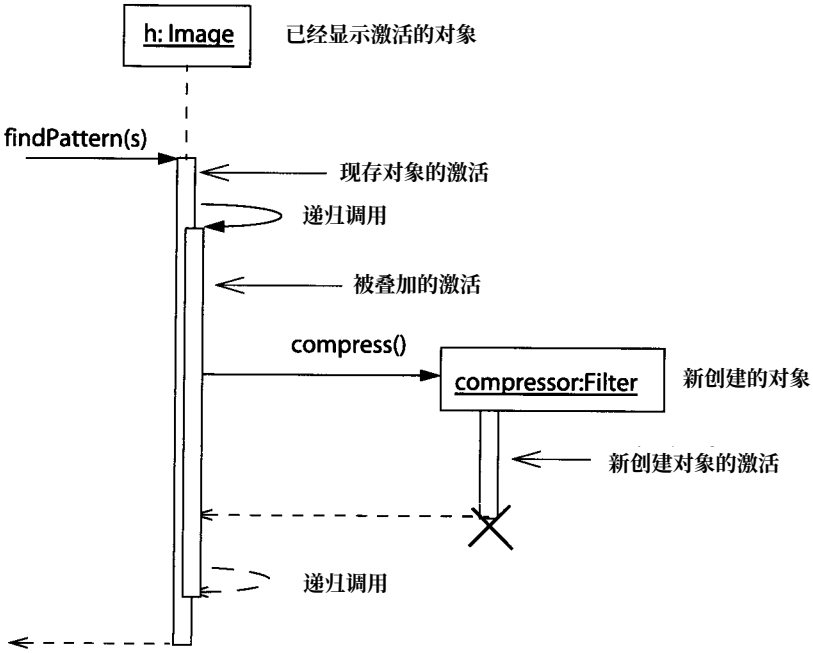


图13-7 激活

China-pub.com 下载

过程是活动的。换而言之，所有活动的嵌套过程激活被同时表示。这些同时发生的嵌套激活是传统计算机的计算栈帧。对于具有现存激活的对象的第二次调用而言，第二次激活的符号画在第一次激活符号的右边，这样它们好像叠加了起来。被叠加的调用可以嵌套于任意深度。调用可以针对同一个操作（即递归调用），也可以针对用一个对象的不同操作。

示例

图13-7表示由调用引起的激活，包括递归调用。

11. active(活动的/主动的)

活动状态是已经被进入且没有退出的状态，由一个对象保持。

见active class, active object。

语义

当进入一个状态的转换被激发后，这个状态变为活动的；当离开这个状态的转换激发时，这个活动的状态不再活动。如果一个对象有一个控制线程，那么至少有一个状态是活动的（在退化的例子中，类只有唯一的状态，在这种情况下，一个事件的反应总是相同的）。如果一个状态在一个对象的类的状态机中是活动的，那么这个对象被认为是拥有这个状态。

一个对象可以同时拥有多个状态，活动状态的集合叫做活动状态配置。如果一个嵌套的状态是活动的，那么包含它的所有状态都是活动的。如果对象允许并发性，那么多个并发的子状态可以是活动的。每个转换至多影响活动状态配置中的数个状态。转换中没有被影响的活动状态仍然是活动的。

一个组成状态可以是顺序的或是并发的。如果这个状态是顺序的和活动的，那么它的后继子状态的一个是活动的。如果这个状态是并发的和活动的，那么它的后继子状态的每一个都是活动的。即，一个组成状态扩展成一个活动子状态的与或树，在它的每一层，特定的状态是活动的。

在构造跨越组成状态边界的转换时必须保持这些并发性约束。一个顺序组成状态中的转换通常有一个源状态和一个目标状态。激发这样一个转换并不改变活动状态的数量。对并发组成状态的每个子区域，并发组成状态的转换通常有一个源状态和一个目标状态。这样一种转换叫做分叉。如果一个或多个作为目的地的区域被忽略，那么每个被忽略的区域的初始状态被隐含当作目的地；如果其中的一个区域缺少初始状态，那么模型是病态的。激发这种转换增加了活动状态的数量。从并发组成状态退出时，则刚好相反。

见state machine，其中包含有关并发状态和复杂转换语义的详细讨论。

示例

图13-8的上部为一个状态机的例子，它既有顺序的又有并发的组成状态。转换被忽略。

China-pub.com
下载

图的底部表示可以并发活动的状态的不同的配置。在这个例子中，有四种可能的活动状态配置。只有叶状态是具体的，而更高层次的状态是抽象的，即一个对象可以不属于这些状态也可以不属于它们的嵌套叶状态。举个例子，不属于 Q的子状态的对象可能也不属于状态 Q。因为Q是并发的，所以如果Q是活动的，C和D也必须是活动的。每个叶状态对应一个控制线程。在更大的例子中，可能的配置数量成指数增长，而且不可能将它们全部表示出来。

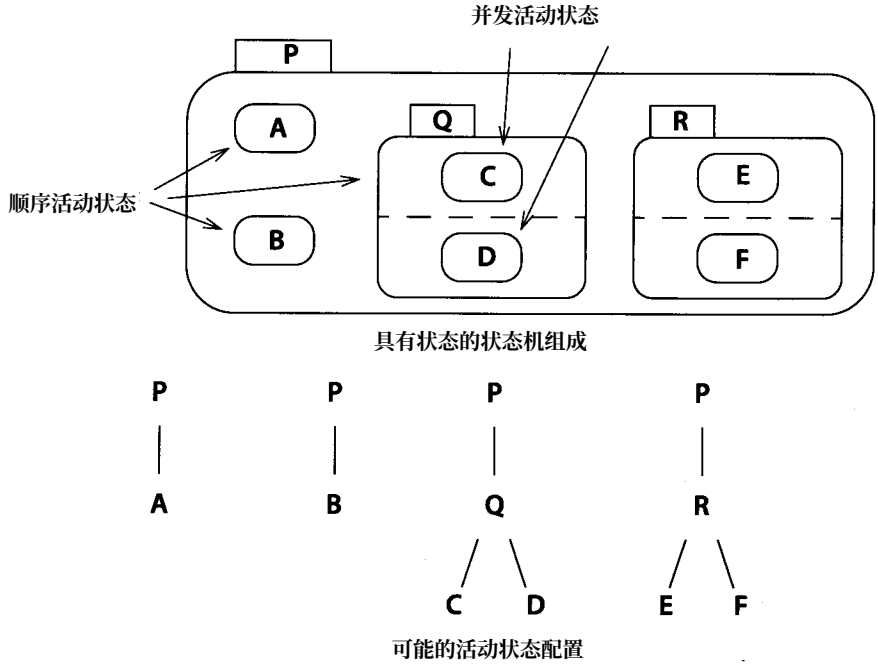


图13-8 并发活动状态

12. active class(主动类)

主动类的实例是主动对象。

见active object。

语义

主动类的实例是主动对象，主动类的构造型是进程和线程。

表示法

主动类用深色边线的框表示。

China-pub.com 下载

示例

图13-9表示一个主动类和其被动类的类图。图 13-10表示含有对应于这个模型的主动对象的协作图。

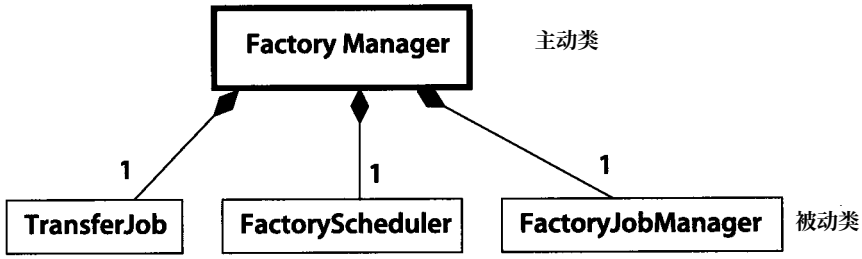


图13-9 主动类和被动类

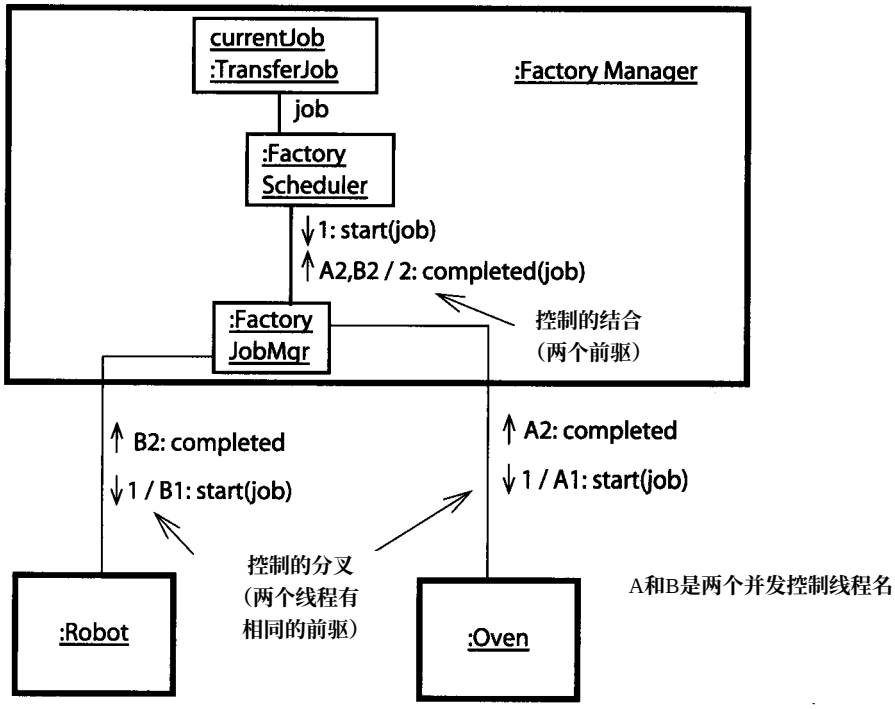


图13-10 主动对象和并发控制间的协作

13. active object(主动对象)

主动对象拥有一个控制线程并且能初始化控制活动，它是主动类的实例。

China-pub.com
下载

见passive object，process，thread。

语义

一个主动对象不在另一个线程、栈帧或状态机内运行。在整个系统的执行中，它具有独立的控制期。从某种定义来说，它是一个线程。每个主动对象是不同的执行集中点；主动对象不是再进入的，并且如果没有附加对象的创建递归执行是不可能的。

主动对象是传统计算机术语中执行栈帧的基础。一个主动对象的创建初始化了一个新的状态机实例。当状态机执行转换时，一个执行栈帧被建立并且一直存在直到转换动作完成，且对象等待外部输入。因此，一个主动对象不在另一个对象的作用域内运行。它能被另一个对象的动作创建，但是一旦被创建，它就独立存在。创造者可以是一个主动的或被动的对象。主动对象用事件驱动。别的对象对它的操作应该作为调用事件由主动对象实现。

被动对象可以作为动作的一部分由另一个对象创建。它有自己的地址空间。被动对象没有控制线程，它的操作在一个主动对象的栈帧中被调用。然而，它可以由状态机建模，来表示由于对它的操作引起的状态的改变。

一个传统的操作系统进程最好等同于一个主动对象。操作系统线程可以由或不由主动对象实现。

主动-被动的区分基本上是一个设计决定而且不限制对象的语义。主动的或被动的对象都可以有状态机，它们还能交换事件。

表示法

主动对象的协作角色在协作图中用具有重边线的矩形表示。通常，主动对象角色表示成与其内在部分的组合。

主动对象也可以用具有重边线的对象符号表示，名字下有下划线，但是主动对象仅在执行例子中出现，因此不那么普通了。

特性关键字{active}也可以用来表示主动对象。

示例

图13-10表示一个工厂自动化系统的三个主动对象：一个机器人、一个炉子和一个工厂管理者，其中管理者是一个控制对象。三个对象都同时存在和执行。工厂管理者在步骤1初始化一个控制线程，这个线程会分成两个分别由炉子和机器人执行的并发控制线程（A1和B1）。当每一个执行完毕，就合并到工厂管理者的步骤2。每个对象仍然存在且保持状态直到下一个事件到来。

14. active state configuration(活动状态配置)

活动状态配置是在状态机内同时活动的一个状态集合。一个转换的激发可以改变这个集合中的一些状态，而另一些不变。

China-pub.com 下载

见active，completion transition，state。

15. activity(活动)

活动是状态机内正在进行的非原子执行。对比：动作。

见completion transition，state。

语义

活动是状态机内子结构的执行，子结构允许中断点的存在。如果一个转换强迫从控制域退出，那么该转换放弃这个活动。活动并不由内部转换的激发终止，因为并没有状态的改变。内部转换的动作会明确地终止它。

活动可以由嵌套状态、子机引用或活动表达式建模。

示例

图13-11为一个警报系统，它说明了动作和活动的区别。当事件 detect intrusion发生时，系统激发一个转换。作为转换的一部分，动作 call police发生。它是一个动作，所以通常是原子的。当动作被执行时，没有事件会被接受。当动作被执行后，系统进入 Sounding状态。当系统处于这个状态时，它执行 sound alarm 活动。活动需要时间来完成，中断活动的事件可能会在此时出现。在这种情况下， sound alarm活动自己并不会中断，只要系统处于Sounding状态它就会持续下去。当 reset事件发生时，转换被激发并将系统返回到 Monitoring状态。当Sounding状态不再是活动的，它的活动sound alarm也被终止了。

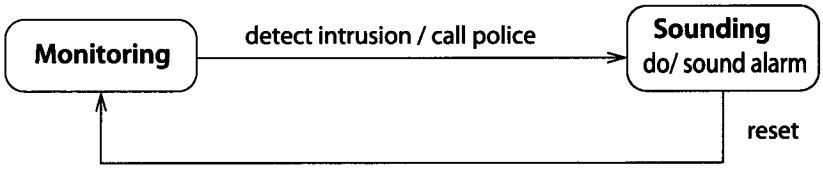


图13-11 动作和活动

16. activity diagram(活动图)

见activity graph。

17. activity expression（活动表达式）

活动表达式是对非原子计算、活动的文字表达式。这种表达式理论上讲可分为原子部分，

China-pub.com
下载

但是允许对整个事情进行文字表示会更方便一些。活动表达式的执行可以被使控制状态的无效转换终止。

语义

活动表达式是由某些语言(如编程语言或其他正式语言)表示的有用的过程或算法。它也可以用人类语言来表示。在这种情况下，工具不能执行它，它也不能被检查错误和其他属性。但是在工作的初期阶段，这已足够了。它也可以表示连续的真实世界操作。

表示法

活动表达式用由某种语言（下例为英语）解释的文字表示。

示例

do/invertMatrix	有限但费时
do/computeBestMove（time-limit）	计算直到时间用尽
do/sound siren	连续运行直到结束

18. activity graph(活动图)

活动图是状态机的一个特殊例子，在该状态机中所有的或大部分的状态都是活动状态或动作状态，所有或大部分的转换由源状态中活动的完成所触发。活动图表示一个程序或工作流。活动图是模型中的完整单元。

见state machine。

语义

活动图是强调计算过程中顺序的和并发步骤的状态机。工作流是被活动图所建模的过程的例子。活动图通常出现在设计的前期，即在所有实现决定前出现，特别是在对象被指定执行所有活动前。这种图是状态机的特例，在它当中状态代表活动的执行，就像一个计算或真实世界不间断的操作，而转换由操作的完成触发。活动图可以附属于操作和用例的实现。

在活动图中状态主要是活动状态或动作状态。活动状态是某种状态的速记，该状态有内部计算和至少一个输出完成转换，该完成转换由状态内活动的完成来激发。如果转换有监护条件，那么可以有多个输出转换。活动状态不应该有内部转换或基于明确事件的输出转换。对于这种条件要使用标准状态。动作状态是原子状态，即它们不会被转换中断。

通常，活动状态用于对一个过程的某个执行步骤建模。如果模型中的所有状态都是活动状态，那么计算的结果不会依赖于外部事件。如果并发活动不访问同一个对象且并发活动的相对完成时间不影响结果，那么计算过程是确定的。

活动图可以包含普通的等待状态，该状态的退出由事件触发。但是这种使用降低了集中

China-pub.com 下载

于活动的目的。如果有多个普通状态，则用普通状态模型。

动态并发性。具有动态并发性的活动状态表示并发执行多个独立的计算。活动与一个参量表集合同时被调用。集合中的每一个成员都是活动的并行调用的参量表。调用是互相独立的，当所有的调用完成时，活动结束并触发它的完成转换。

对象流。有时，查看一下操作和作为它的参量值或结果的对象之间的关系是有好处的。一个操作的输入和输出可以表示成一个对象流状态。它是一个状态的构造型，表示在计算过程中特定点的给定类的对象的存在。为了更精确，输入或输出对象可以在它的类中声明处在指定的状态。例如，“签署合同”操作的输出为“已签署”状态的合同类的对象流状态。该对象流状态可以是其他多个操作的输入。

泳道。活动图中的活动可以依照不同的准则划分为几组。每个组代表活动职责的一些有意义的部分，例如，商业组织负责给定工作流的某一步。根据它们的图形表示法特征，每个组被称作泳道。

表示法

活动图是状态机中的一种，但是几种速记表示法也适用于活动图，如：活动状态、分支、合并、泳道、对象流状态、状态类、信号发送和信号接收表示法和延迟事件。

请看一些可选符号的控制图标，它们可用于活动图中。

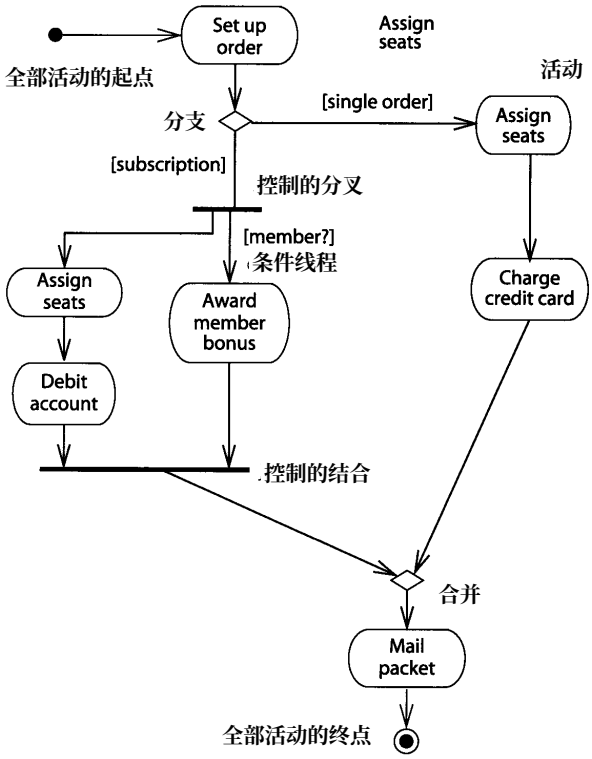


图13-12 活动图

China-pub.com
下载

示例

图13-12表示一个活动的工作流，它用于处理剧院售票处的订单。它包括一个分支和随后的合并，取决于订单是预订的还是个人票。分叉初始化逻辑上同时发生的并发活动。它们的实际执行可以重叠也可以不重叠。并发性由随后一个相对应的结合终止。如果只涉及到一个人，那么并发活动可以以任何次序执行（假定它们不能被同时执行，这是模型所允许的，但在实际中非常困难）。例如，售票处的人员可以先分配座位，再授予赠品，再借记账户；或者他们可以先授予赠品，再分配座位，再借记账户，但是他们只有在分配座位后才能借记账户。

分叉的一个输出部分有一个监护条件，检查预订者是不是会员。这是一个条件线程，它只有在监护条件被满足时才会激发。如果这个线程没有被激发，那么随后相对应的结合的输入部分被认为已完成。如果预订者不是会员，则只有一个线程被激发，它负责分配座位和借记账户，但是不会等待结合处的同步。

• 泳道。活动图中的活动可以分成为几个区域，每个区域在图中用虚线分开因此被叫做泳道。泳道是活动图的内容的组织单元。它没有内在的语义，但可以根据建模者的意愿使用。通常，每个泳道代表真实世界组织内的一个组织单元。

示例

图13-13中，活动被泳道分成三个部分，每个部分对应一个不同的资金保管者。虽然在这个例子中每部分都对应于对象，但是 UML并不要求这么做，会有明显的类对应于每个部分，而且这些类可能是执行操作以实现已完成模型的每个活动。

这张图也表示了对象流符号的使用。对象流对应于一个订单对象经过整个活动的不同状态。例如，符号 Order[placed]表示在计算中的位置，一个订单已经被提高到 Request Service活动的placed状态，但是还没有被Take Order活动使用。当Take Order活动完成后，订单进入entered状态，对象流符号将该状态表示在Take Order活动的输出上。该例中的所有对象流表示处于声明中不同时刻的同一对象。因为它们表示同一对象，故它们不能同时存在。如图所示，一条连续的控制路径可以从它们中画过。

对象流。由动作输入或输出的对象可以表示为对象符号。符号表示处于计算中某一点的对象，在该点对象适合作为输入或输出。虚线箭头表示从活动状态到作为活动输出之一的对象流的输出转换。虚线箭头也可以表示从对象流到用这个对象作为输入的活动状态的输入转换。通常，同一个对象可以作为一个活动的输出和一个或多个后继活动的输入。

当对象流箭头（虚线）满足冗余约束时，控制流箭头（实线）可以被忽略。换而言之，当动作产生的输出是后继动作的输入时，对象流关系包含一个控制约束。

状态类。通常，同一个对象被一些改变它的状态的后继活动所控制。为了更加准确，对象可以在图中出现多次，每次出现表示它生命中的不同的状态。为了区分同一个对象的多次出现，每个点的对象的状态可以放在方括弧内并附加在类的名字旁，如 PurchaseOrder [approved]。这个符号也可用于协作图。

China-pub.com 下载

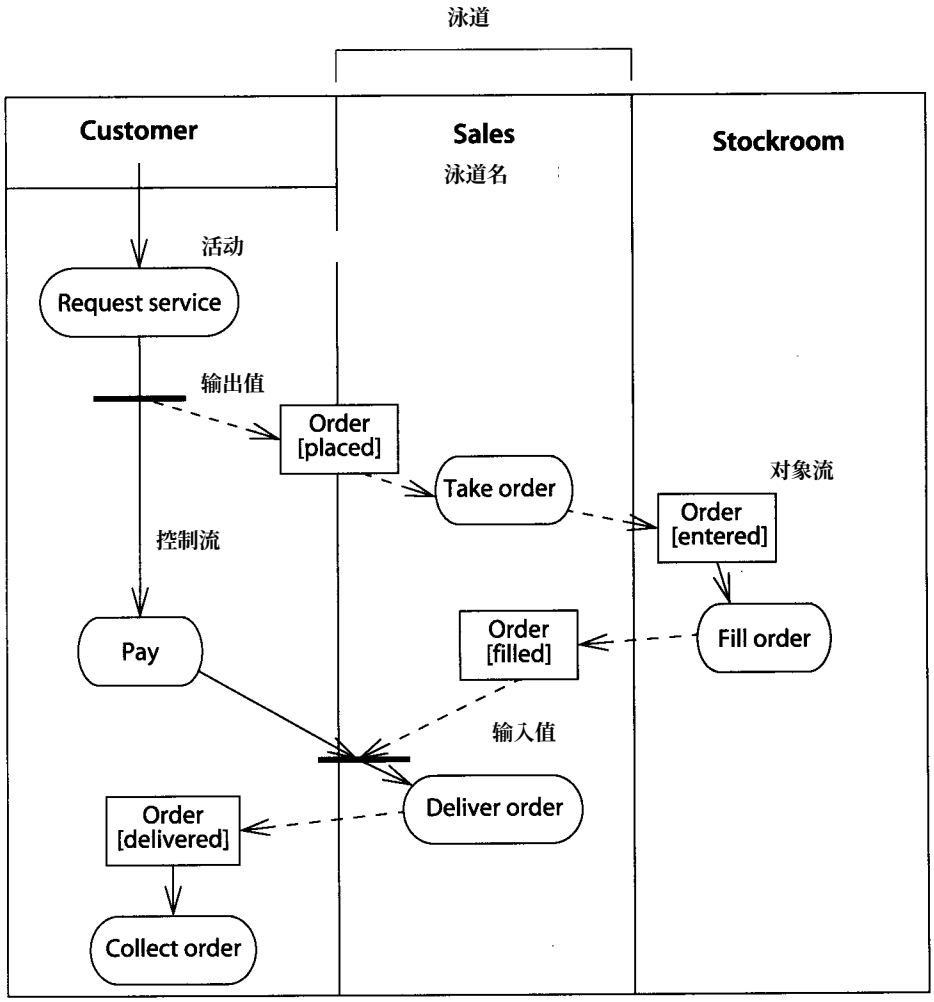


图13-13 带有泳道的活动图

见可用于活动图的其他符号的控制图标。

延迟事件。有时，当其他活动进行时，有一种事件必须为了晚一些使用而延迟（通常，没有立即处理的事件会被遗失）。延迟事件是放置在内部队列中，直到它被使用或被抛弃的事件。如果在状态或活动中发生延迟事件，则该状态或活动将对它们进行说明，其他事件必须被立即处理，否则将被遗失。当状态机进入一个新状态时，如果新状态不延迟这些事件，那么所有的延迟事件将会发生。如果新状态中的转换由先前状态中的延迟事件触发，那么转换立即激发。如果几个转换为隐含的，则它们中的哪一个会激发并不明确，且施行一条规则以选择一个要激发的转换是语义变更点。

如果一个事件在它所延迟的状态中发生，它可能触发一个转换，在这种情况下它不放在队列中。如果这个事件不触发转换，则它被放在队列中。如果活动状态改变，那么队列中的

China-pub.com
下载

事件可以触发新状态中的转换，但是如果它们在新状态中仍然是延迟的，那么它们仍然在队列中。如果一个事件在组成状态中必须是延迟的，但可以促使一个或多个子状态中的转换，那么使事件不再延迟以触发转换的能力是很有用的。否则，事件将不得不在每个子状态中延迟而不触发转换。请注意如果一个延迟事件与转换的触发器事件相一致，但是不满足监护条件，那么这个事件并不触发转换，也不会从队列中移去。

一个可延迟事件在状态内表示，后面有一个反斜杠和特殊操作 defer。如果事件发生且没有触发转换，则它被保存起来，且当对象转换到另一个状态时再次发生。当对象到达一个它不会被延迟的状态时，它必须被接受或被忽略。若没有 defer声明则可以取消先前的延迟。defer指示符可以放于一个组成状态上，这样事件就在整个组成状态内延迟。

动作状态是原子的，所以当这些状态活动时隐含地延迟任何此时发生的事件，无需将它们标记为延迟的。发生的事件被延迟直到动作完成，这时事件可以触发转换。

示例

图13-14表示煮咖啡的步骤。在这个例子中，外部对象（coffeepot）没有表示出来，表示出来的仅仅是由人直接执行的活动。打开壶的动作被建模成发送到壶的事件。活动 Get Cups 发生在打开咖啡壶之后。在得到杯子后，需要等待直到灯熄灭。然而，存在一个问题，如果 light goes out 事件在Get Cups活动完成之前发生，那么因为状态机没有准备好处理事件，

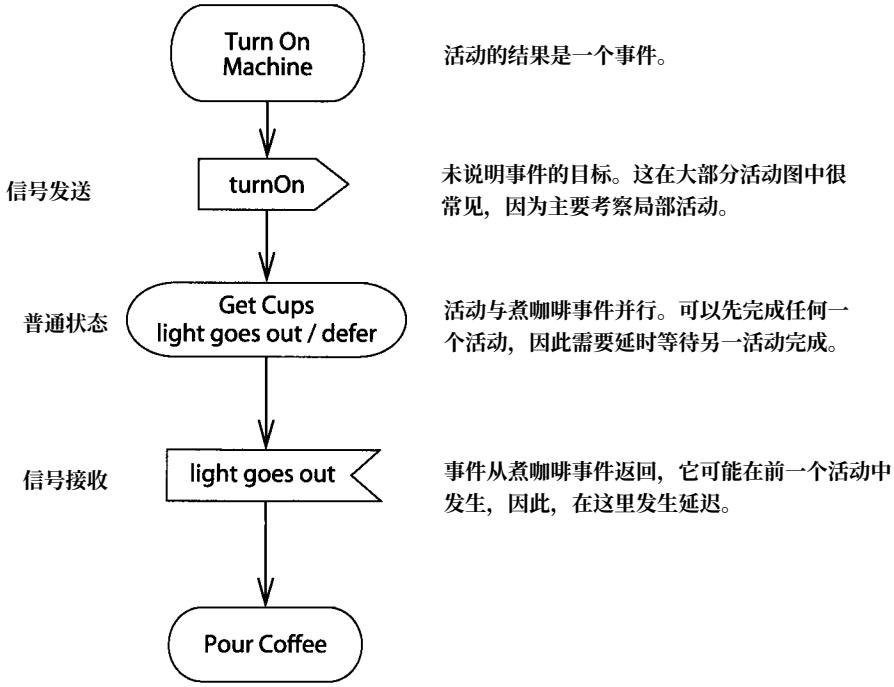


图13-14 延迟事件和控制图标

China-pub.com 下载

它将会丢失。为了避开丢失事件的危险，活动状态Get Cups被标识为延迟light goes out事件。如果事件在活动还在执行时发生，事件不再会丢失。相反地，它被保存在队列中直到状态机离开Get Cups状态，这时它被执行且触发了转换。

请注意light goes out 事件不是Get Cups状态的触发，因此当它出现时也不会终止活动的进行。这个事件是Get Cups活动完成后的一个接收状态的触发者。

动态并发性。有一个不确定值的动态并发性用活动符号右上角的复合串表示（如图13-15）。这预示着活动的多个拷贝同时出现。动态活动接收参量表的集合，细节必须用文字描述。如果并发性应用于多个活动，则它们必须包含于一个组成活动中，这个组成活动得到多重指示。

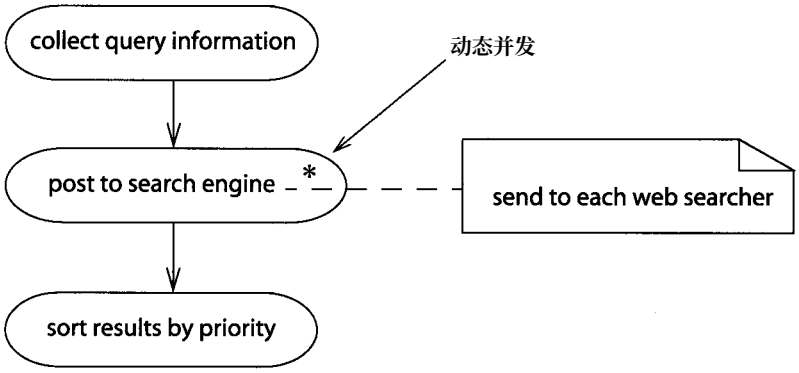


图13-15 动态并发性

图结构。状态机必须被良好地嵌套，即它被分解成并发的或顺序的子状态。对一个活动图，分支和分叉必须被很好地嵌套。每个分支必须有一个对应的合并，每个分叉必须有一个相对应的结合。这对活动图来说有时不方便。通常情况是采用部分有序图，其中没有直接的循环，但是分叉和合并不必相匹配。这种图不是良好嵌套的，但是它可以经过指定活动到线程和通过当转换跨越边界时引入同步状态两种方法来转变为良好嵌套图。分解不一定是唯一的，但是对部分有序图来说所有的分解将产生相同的可执行语义。所以，对一个简单的部分有序图没有必要表示一个明确的分解或同步状态。对涉及到条件和并发性的更复杂的图，可能需要更明确的分解。

19. activity state(活动状态)

活动状态表示一个具有子结构的纯粹计算的执行，通常为操作或位于其内的声明的调用或真实世界程序的执行。活动状态可以被使转换离开状态的事件从外部中断。活动状态不必自己中断。对于它可以活动多长时间是没有限制的。

见activity，completion transition。

China-pub.com
下载

语义

活动状态是具有内部计算和至少一个输出完成转换的状态，当状态中的活动完成时该转换激发（如果有监护条件则可以有几个这种转换）。活动状态不应该有内部转换或基于确切事件的输出转换。对于这种情况要求使用普通状态。活动状态通常用于对算法执行中的一步建模。如果模型中的所有状态都是活动状态且并发的活动不访问同样的值，那么计算是确定的，即使它涉及并发执行。

活动状态可以引用一个子机，通常为另一个活动图。这与将活动状态扩展成一个子机网络的拷贝是等价的。它是状态机的子例程。

活动状态是一个程序的执行过程的状态而不是一个普通对象的状态。

动作状态是原子的活动状态，即当它是活动时不会被转换所中断。它可以被建模成只有一个入口动作的活动状态。

活动状态可以被用于普通的状态机，但是它们更常用于活动图。

离开一个活动状态的转换通常不包括事件触发器。输出转换被状态中活动的完成隐含地触发。转换可以包括监护条件和动作，注意所有有关违反活动的转换的可能条件都要涉及到，否则控制被挂起。如果多个监护条件赋值为真，那么选择为不确定的。

对于其他条件使用普通状态。

表示法

活动状态用以下形状表示：上下边是直线，左右边是凸弧（如图 13-16）。活动表达式被放在符号内。在图中活动表达式不必是唯一的。

图13-16 活动

讨论

动作状态用于简短的簿记操作而活动状态用于任何有持续时间或复杂性的计算。这意味着动作可以锁定系统所以它必须是短暂的，但是活动可以被中断，所以如果一些紧急的事情发生不要求系统去完成它。UML语义并不阻止长动作，但是代码生成器可以合理地假定动作打算立即完成，同时活动对于其他动作是可中断的。

20. activity view(活动视图)

活动视图是系统的一个方面，它将行为的说明作为由控制流连接的活动进行处理。活动

China-pub.com 下载

视图包含活动图并且表示在活动图上。它松散地与其他行为视图组织在一起。

见activity graph。

21. actor(参与者)

参与者是直接与系统相互作用的系统、子系统或类的外部实体的抽象概念。参与者参与到用例或一组连贯的用例中以完成整个目标。

见use case。

语义

参与者刻划和抽象了一个外部用户或与系统和类元相互作用的一个相关的用户集合的特征。参与者是具有集中的目的和含义的理想化形式，不一定有确切对应的物理对象。一个物理对象可以兼有多个无关联的目的，所以可以由多个参与者建模。不同的物理对象可以包含同样的目的，故在这个方面可以由同一个参与者建模。用户对象可以是一个人、一个计算机系统、另一个子系统或另外一种对象。例如，一个计算网络系统的参与者可以包括 Operator、System Administrator、Database Administrator和普通的User，也可以有非人类参与者，如RemoteClient、MasterSequencer和NetworkPrinter。

每个参与者定义了一个角色集合，当系统的用户与系统相互作用时会采用它们。参与者的一个完整集合描述了外部用户与系统通信的所有途径。当一个系统被实现，参与者也被物理对象实现。一个物理对象如果可以满足多个参与者的角色，那么它可以实现多个参与者。例如，一个人可以既是一个商店的售货员又是顾客。这些参与者不是本质上相关的，但是它们可以由一个人来实现。当一个系统的设计被施行时，系统内的多个参与者被设计类实现（见realization）。

参与者与系统的不同的交互作用量化为用例，用例是涉及系统和它的参与者的连贯的功能块，用来完成对参与者有意义的事情。一个用例可以包括一个或多个参与者，一个参与者可以参与一个或多个用例。最终，参与者由用例和参与者在不同用例中所担任的角色决定。没有参加任何用例的参与者是无意义的。

用例模型刻划了一个实体（如系统、子系统或类）与外部实体相互作用时产生的行为的特征。外部实体是实体的参与者。对于一个系统，参与者可以既由人类用户又由其他系统实现。对于一个子系统或类，外部元素可以是整个系统的参与者，或者参与者可以是系统内的其他元素，如其他子系统或类。

参与者实例通过与用例实例传递消息实例（信号与调用）来与系统通信，在实现层，与实现这个用例的对象传递消息实例。这通过参与者和用例之间的关联来表达。

参与者可以列出它发送和接收的信号集合。参与者也可以有一组它所支持和需要的接口。参与者的接口必须与和它通信的每个用例的接口相容。换而言之，参与者必须接受用例发送

China-pub.com
下载

的所有信号，而且它不能发送给用例不能接收的信号。参与者的接口限制了参与者如何映射到类上。参与者也可以有一组表示它的状态的属性。

泛化

多个参与者可以有相同点，即它们可以以同样的方式与同一组用例交流。这个相同性用与另一个参与者的泛化关系来表示，它对参与者的共同方面进行建模。后代参与者继承了由祖先参与者拥有的角色和与用例的关系。一个后代参与者的实例常常在祖先希望被用到时使用（替代原理）。一个后代包括它的祖先的属性和操作。

表示法

参与者可以用具有构造型实体《actor》的类符号（矩形）表示。标准的参与者构造型图标是一个小人，参与者的名字在图的下面。参与者可以用间隔来表示它接收到的属性和事件，它也可以用依赖关系表示它发送的事件。这些是普通类元的能力（如图 13-17）。

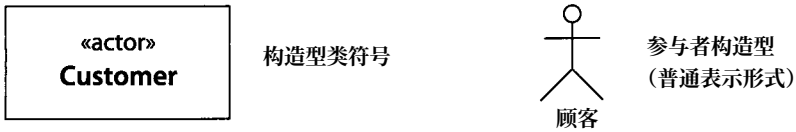


图13-17 参与者符号

22. actual parameter(实际参数)

见argument。

23. aggregate(聚集)

聚集是表示聚合关联（整体与部分）中整体的类。

24. aggregation(聚合)

聚合是一种关联形式，它指明一个聚集（整体）和组成部分之间的整体与部分的关系。见composition。

语义

一个二元关联可以声明为是一个聚合 —— 一种整体与部分的关系。关联的一端指出聚集

China-pub.com 下载

而另一端是无记号的。两端不能同时是聚集（或组成），但是两端可以同时是无记号的，这时它不是聚合。

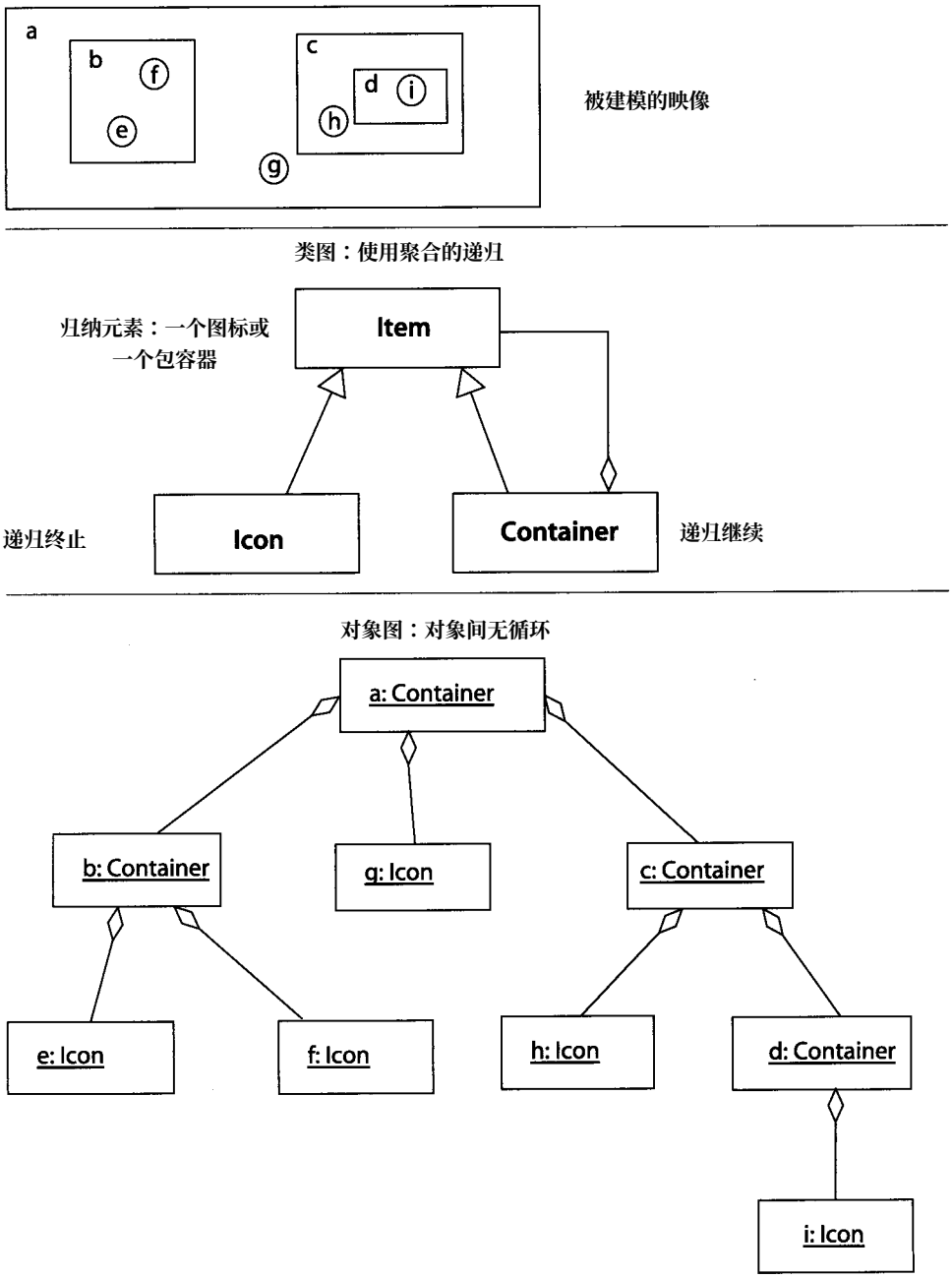


图13-18 对象的聚合是无循环的

China-pub.com
下载

从聚合关联实例化形成的链遵循若干规则。聚合关系是可转换的，非对称地跨越所有聚合链，甚至跨越那些来自不同聚合关联的链。可转换意味着如果从 B到A有聚合链的路径，那么说“ B是A的一部分 ”是有意义的。非对称性意味着在聚合链的路径中不存在循环。即一个对象不能直接地或间接地作为它自己的一部分。将两条规则合在一起，从所有聚合关联得到的聚合链组成的图形就形成了一个部分有序图，该图没有循环（树是一个具体而又常见的部分有序的情况）。图13-18给出了一个例子。

从对象B到对象A的链的有向路径表示存在从类 B到类A的聚合关联的有向路径，但是关联的路径会涉及到循环，在这当中同一个类可以出现多次。一个类到它自己的聚合关联的有向路径是一个递归。

有一种更强的聚合形式叫做组合。组成是有着附加约束的聚集，这些约束是，一个对象只能是一个组成的一部分且组成对象有安置它的每一部分的责任，即对它们的创建和销毁负责。

见composition。

在普通的聚合中，一个部分可以属于多个聚集，而且它可以独立于聚集存在。通常，聚集“ 需要 ”部分，即它可被认为是部分的积聚。但是部分可以独立存在，而不必认为仅仅是“ 部分 ”。例如，路径可以认为是段的集合。但是段可以独立存在而不管它是不是路径的一部分，并且同一段可以出现在不同路径中。

见association及association end以了解聚合的更多特性。

表示法

聚合用一端用空菱形表示的连接到聚集类的关联线表示（如图 13-19）。如果该聚合是一个组，那么菱形是实心的，见后文图 13-68。关联的两端不能同时有聚合标识。

聚集类可以有多个部分。聚集类和每个部类之间的关系是分离关联（如图 13-20）。

如果有两个或多个聚合关联指向同一个聚集类，那么可以通过将聚合端结合成一个独立的部分来将其画成一棵树（如图 13-21）。这要求聚合端的所有属性都是一致的，如，它们必须有相同的多重性。将聚合画成一棵树纯粹是表示方达的选择，并没有附加的语义。

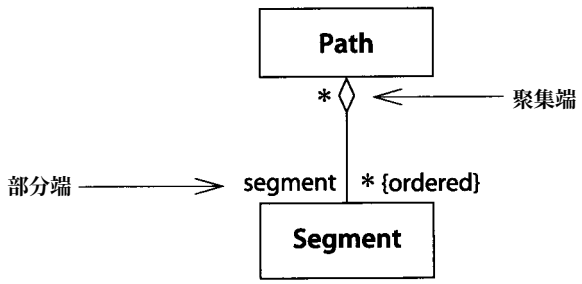


图13-19 聚合表示法

China-pub.com 下载

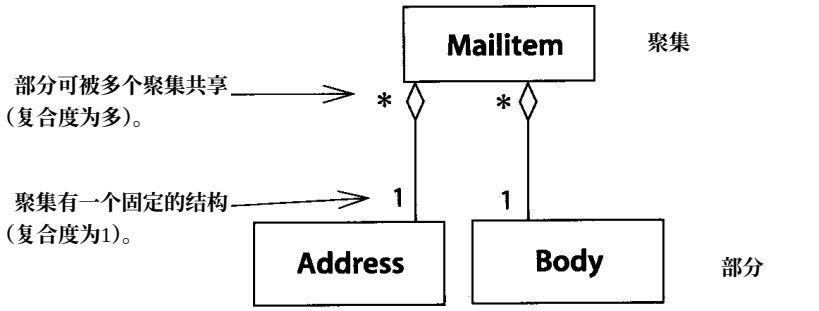


图13-20 由几部分组成的聚集

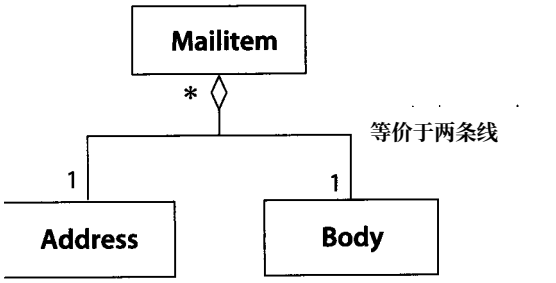


图13-21 同一个类的多重聚合的表示形式

讨论

聚合和关联之间的区别通常是个人喜好的问题而不是语义的区别，聚合即关联。聚合表示了这样一种思想：聚集是它的每一部分的总和。实际上，它加入到关联的实际语义是聚集链不能组成循环这一约束。其他约束，如存在依赖性，由多重性而不是聚合标识说明。除了附在聚合上的很少的一部分语义外，其必要性得到了大家的认同（因为各种不同的原因）。可把它作为建模的一帖安慰剂。

几种次要的特性与聚合有关，但是它们不够可靠故不能使它们成为聚合定义的一部分。这些特性包括从聚集到部分的操作的传播（如移动操作）和紧缩内存分配（以便聚集和它的递归部分能够在一次内存交换中高效加载）。一些作者对几种聚合进行了区别，但是区别相当细微且可能对通用的建模是多余的。

聚合是一个超越特殊关联的特性。可以跨越不同的类组成聚合。聚合对所有没有循环聚合链，包括来自不同关联的链的聚合关联（包括组合关联）的实例实施了一条约束。在某种意义上，聚合是关联的一种泛化，在聚合中，约束和一些操作应用于许多特定种类的关联。

组合有更多具体的语义，它们对应于物理包含和所有权的不同观点。当每个部分由一个对象所拥有并且每个部分没有独立于其拥有者的生命期，那么这时用组合比较合适。特别是在当拥有者被创造时，所有的部分也必须分配和初始化；当拥有者毁灭时每个部分也不会存活。

China-pub.com
下载

类的属性有这些特性，虽然它们没有被明确地建模成一种组合，但是仍然可以被认为如此。通过使用组合，可以避免存储器管理的负担、没有指向任何对象的指针的危险以及被遗弃的对象的危险。组合也适用于这样的情况，由于压缩和操作的原因，一组属性已经被分离到一个不同的类中，但是这些属性真正适用于主类。虽然用来实现关联的包容器类也通常是组成部分的候选者，但是它们必须由代码生成器生成且不会显式地被建模。请注意一个组成部分，如包容器类，可以包含对非组成部分的引用或指针，但是被引用的对象不会在引用对象毁灭时也被销毁。

25. analysis(分析)

分析是系统捕捉需求和问题的阶段。分析着重于做什么，设计着重于如何去做。在一个迭代过程中，各个阶段不必连续地执行。这个阶段的这种效果由分析层模型（特别是用例视图和静态视图）来表示。对比：analysis，design，implementation和deployment。

见stages of modeling，development process。

26. analysis time(分析时间)

分析时间是软件开发过程中分析活动执行时的时间。不要假设一个系统的所有分析都在同一时刻或先于其他活动发生，如设计和实现。对于任何独立元素，各种活动是连续的，但是对整个系统不同的活动可以混合在一起。

见design time，modeling time。

27. ancestor(祖先)

祖先是通过一个或多个父关系路径建立起的元素。

见generation，parent。

28. architecture(构架)

构架是一个系统的组织结构，包括系统分解成的各个部分、它们的连接性、交互机制和通知系统设计的向导规则。

见package。

语义

构架是有关软件系统组织的重要决定的集合，它包括结构元素和将这些元素连接起来的接口的选择、结构元素的大规模组织和它们的连接的拓扑结构、在元素的协作中说明的行为、

China-pub.com 下载

对整个系统都有效的重要的机制、指导它们的组织的构造风格。例如，决定建立一个两层的系统，每一层包含一定数量的子系统，这些子系统以一种特殊的方式通信，而该决定即构架决定。软件构架不仅仅涉及到结构和行为，也涉及到使用、功能、实施、弹性、重用、可理解性、经济和技术约束以及综合、美学的考虑。

讨论

有关系统分解的构架决定可以通过模型、子系统、包和构件说明。这些元素间的依赖关系是构架适应性和修改系统难易程度的主要指标。

构架的另一个主要部分是它准备建立在其上的机制。这些可以通过协作和模式获得。

非结构化的决定可以用标记值说明。

29. argument(参量)

参量是对应于参数的具体的值。

见binding，parameter，substituablility principle。

语义

消息的运行实例有一张参量值表，其中每一个的类型都必须与信号或操作声明中的对应参数的声明类型相匹配。如果一个值的类或数据类型与参数的声明类型相同或是它的后代，那么这个值就是相容的。通过替代原理，一个后代的值可用在任何声明祖先类型的地方。一个值的实现取决于模拟器或它出现的执行环境。

在协作或状态机中，表达式为动作而出现。在这些表达式中，调用和消息发送需要参数说明。这些参数说明也是表达式。当在运行时计算这些表达式时，它们必须计算与它们匹配的声明参数一致的值。

然而在模板绑定中，UML模型中的参量在建模时间出现。在这种情况下，参量表示成用某种语言（通常是约束语言或编程语言）表示的表达式。模板参量不仅仅可以包括普通的数据值和对象，也可以包括类元本身。在后一种情况下，对应的参数类型必须是类元或其他的元类型。模板参量的值必须在建模时确定，它不能用来表示运行时的参量。如果参数没有在建模时间被限定则不要使用模板。

30. artifact(制品)

制品是被软件开发过程所利用或通过软件开发过程所生产的一段信息，如外部文档或工作产物。制品可以是一个模型、描述或软件。

China-pub.com
下载

31. association(关联)

如果几个类元的实例之间有联系，那么这几个类元之间的语义关系即关联。

见association class，association end，association generation，binary association，n-ary association。

语义

关联是两个或多个特定类元之间的关系，它描述了这些类元的实例的联系。参与其中的类元在关联内的位置有序。在一个关联中同一个类可以出现在多个位置上。关联的每个实例（链）是引用对象的有序表，关联的外延即这种链的一个集合。在链集合中给定的对象可以出现多次，或者在关联的定义允许的情况下可以在同一个链中（在不同的位置）出现多次。关联将一个系统组织在一起，如果没有关联，那只有一个无连接类的集合。

结构

关联可以有一个名称，但是它的大部分描述建立在关联端点中，每个端点描述了关联中类对象的参与。关联端点只是关联描述的一部分，不是可区分的语义或可用符号表示的概念。

名称。关联可以有一个名称，在包含包的所有关联和类中它必须是唯一的（关联类既是一个关联又是一个类，所以关联和类分享同一个命名空间）。关联不是必须要有一个名称，它的端点的角色名称提供了在同一个类中辨别多个关联的另一种途径。按照习惯，名称以类在表中出现的顺序读出：Person为Company工作；Salesman卖Car给Customer。

关联端点。关联包含一张有多个关联端点的有序表（这些端点是可以被区分的并且是不可替换的）。每个关联端点定义了在关联中给定位置的一个类（角色）的参与。同一个类可以出现在多个位置上，而位置通常是不可交换的。每个关联端点指定了应用于对应对象的参与的特性，如在关联中一个独立的对象在链中会出现多少次（多重性）。某些特性，如导航性只应用于二元关联，但是多数可以应用于n元关联。

见association end。

实例化

链是关联的实例。它包含对于每个关联端点的槽，每个槽包含对一个对象的引用，该对象是作为对应关联端点的类的（直接的或间接的）实例。除对象表之外，链没有身份标识。关联外延中的链组成了一个集合，在这之中不会存在副本。一个对象在链集合中出现的次数必须与关联的每个端点的多重性相一致，例如，关联 SoldTickets将多张票与一场演出连接起来，那么每张票在一个链中只出现一次，但是每场演出可以出现多次，每次对应不同的票。

链在系统执行过程中可以被创造和销毁，服从每个关联端点可变性的限制。在某些情况下，链可以从关联端点一端的对象创建或改变而不能从另一端做同样的事。一个链从一组对

China-pub.com 下载

象引用中创建。链没有自己的身份标识，所以讨论改变它的值是没有意义的。然而，它可以被销毁，可创建一个新的链来代替它。一个关联类的链，除了定义它的身份的一组对象外还有一个或多个属性值，而且属性值在保持参与对象的引用的情况下可以由操作来改变。

表示法

二元关联用连接两个类边界的实线路径表示（如图 13-22），n元关联用菱形表示，菱形通过路径与每个参与到其中的类连接，见后文图 13-129 (在二元关联中菱形省略)。路径的多个端点可以连接到一个独立的类。

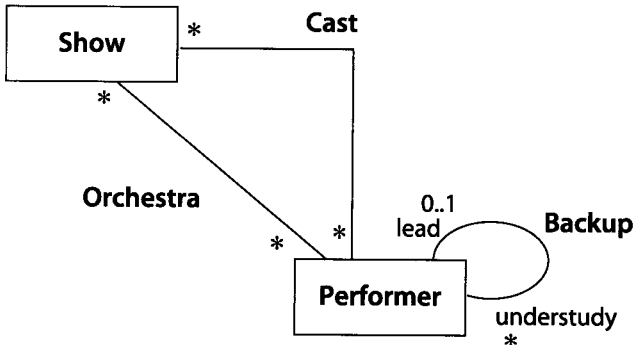


图13-22 关联

路径包含一个或多个相关的实线部分，通常是直线，但是也允许使用弧线和其他的曲线，尤其是在表示自关联（即一个类出现多次的关联）时。每个单独的部分没有语义含义。线的风格由用户选择。

见path。

路径的端点处有描述关联中类的参与情况的修饰符号。一些修饰符号位于路径的端点，线和类符号之间。如果有多个符号，那么它们被按照从线的端点到类符号的顺序摆放 —— 导航箭头、聚合/组合菱形、限定符（如图 13-23）。

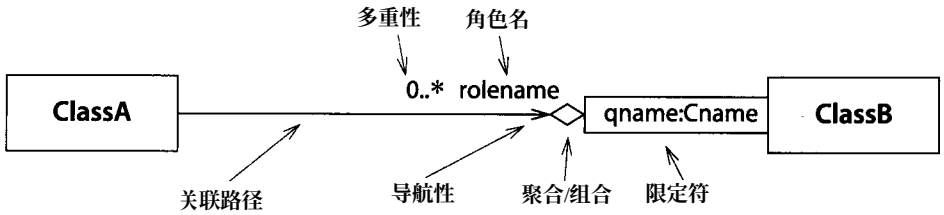


图13-23 关联端修饰符的顺序

其他修饰符，如名称标签，被放在靠近它们所确认的事物旁。角色名称被放置在靠近路径端点处。

见association end。

China-pub.com
下载

关联名称

关联的名称放置在路径的旁边，但远离关联的一端，这样就不会引起混淆（混淆的危险对人类而言纯粹是视觉上的，在一个图形工具内，相关的符号可以用清晰的内部超链相互连接）。关联名称可以在多段连接中从一个部分拖到另一个部分而没有语义冲突。关联名称可以用一个小的实心三角形表示表中类的顺序。直观地，名称箭头表示如何去读名字。在图 13-24 中，类Person和类Company之间的关联 Works For 有一个从Person指向Company的名称三角形，该关联可以读成“人为公司工作”。请注意名称旁的次序三角形纯粹是一个符号化的装置，它用来指明关联端点的顺序。在模型中，端点本质上是有序的，所以模型中的名字不需要次序特性。

图13-24 关联名称

关联上的构造型通过将构造型名字放在小双尖括号内表示，它放在关联名称前或替代关联名称。特性表可以放置在关联名称的后面或下面。

关联类

我们将类符号用虚线连接到关联路径上来表示关联类。对 n元关联来说，虚线连接到关联菱形上。关联中有关类的特性表示在类的符号中，而有关关联的特性则表示在路径上。然而，即使图形由两个图构成，基本的建模结构也是单个元素。

见association class。

异或约束

{xor}约束将两个或多个关联连接起来，这些关联在某一端连接到一个独立的类（基类）。基类的一个实例可以参与到通过约束连接的关联中的一个。必须注意到所选择关联的多重性。如果任何关联多重性包括基数 0，那么基类的一个实例可能不会有关于关联的连接，否则，它必定有一个。

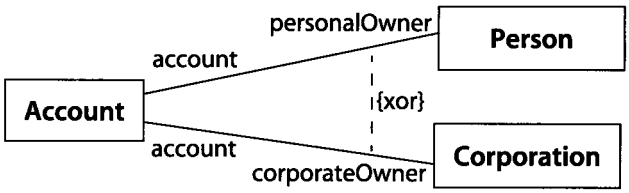


图13-25 异或关联

China-pub.com 下载

异或约束用连接两个或多个关联的虚线表示，所有关联必须有一个公共类，约束字符串{xor}标识在虚线旁（如图13-25）。远离公共类一端的角色名必须是不同的（这仅是一种通过使用标准约束重叠实现的约束表示法的预定义用法）。

讨论

关联不需要有名称。通常，角色名更方便一些，因为它们为导航和代码生成提供了名称，并且避免了如何读名称的问题。如果它有名称，那么名称在它的包内必须是唯一的。如果它没有名称且两个类之间有多个关联，那么必须用角色名来区分关联。如果两个类之间只有一个关联，那么类名对于确定关联来说就足够了。

当真实世界的概念有一个名称（如Marriage 或Job）时，关联名更有用一些。当关联名由读取的给定方向指定方向时，通常用角色名更好，这样读起来不会有歧义。

见transient link 中关于只在过程执行时存在的建模实例关系的讨论。

见composition中有关涉及两个关联的泛化的例子。

标准元素

implicit，persistence，xor。

32. association class(关联类)

关联类既是关联又是类，故它具有关联和类的特性。它的实例是具有属性值的链和对其他对象的引用。尽管它的表示法包括关联和类的符号，但是它的确是一个独立的模型元素。

见association，class。

语义

关联类具有关联和类的特性，它将多个类连接起来且它也具有属性和操作。当每个链必须有它自己的属性值、操作或对对象的引用时，可用到关联类。它可以被看作具有对每个关联端点的额外的类引用的类，对于实现它而言这是一条明显和通常的途径。每个关联类的实例都有对象引用和通过类部分说明的属性值。

连接类A和类B的关联类C不同于与A和B有二元关联关系的类D（见讨论部分）。与所有的链一样，像C这样的关联类的一个链从对象引用中得到它的身份标识。属性值没有涉及到提供身份上。所以即使它们的属性值有区别，两个链也不能有同样的（ a，b）对象对，因为它们不会有同样的身份。即，给定属性 E，不允许（a、b、e1）和（a、b、e2）都是C的实例，因为它们分享同一个身份（a、b）。然而，对象有固有的身份，所以两个对象可以有同样的属性值或对相同对象的链。即，关联（包括像 C这样的关联类）是一个有序表集合并且它的对象引

China-pub.com
下载

用中没有副本；然而，像D这样的隐含关系更像一个包，它可以有副本。见讨论。

关联类可以有操作，这些操作可以改变链的属性或者增加或删除链。因为关联类也是一个类，故它也可以参与到关联本身。

关联类可以不把它自己作为参与类中的一个（虽然有些人能够确切地找到这种递归结构的意义）。

表示法

关联类用类符号（矩形）表示，由一条虚线与关联路径连接（如图 13-26）。类符号的名称和附着在关联路径上的名称字符串是冗余的。关联路径可以有常用的关联端点符号。类符号可以有属性和操作，作为类它参与到自己的关联中。虚线之上没有符号，它不是一个关系而仅仅是整体的关联类符号的一部分。

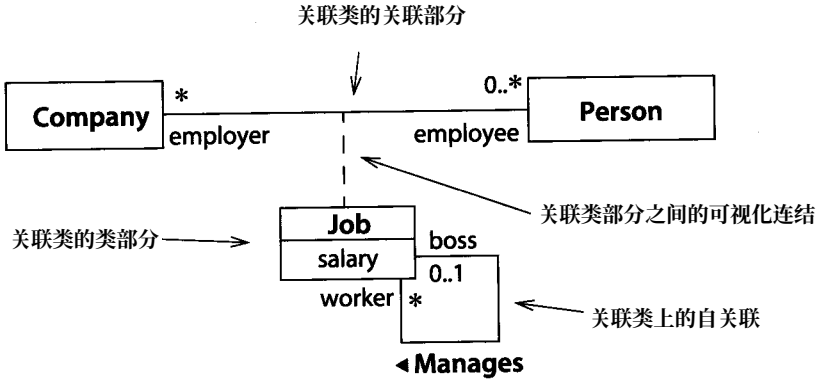


图13-26 关联类

风格指导

附属点不应该太靠近路径的两端，这样就不会使它看起来像附属在路径的端点或接到任何角色符号上。

关联路径和关联类是独立的模型元素，所以有着独立的名称。名称可以表示在路径上或类符号上或在两者上。如果关联类只有属性但是没有操作或其他的关联，那么名称可以表示在关联路径上，并且为了突出它的“关联自然性”而从关联类符号中省略。如果它有操作和其他的关联，那么名称可能将从路径上忽略并且为了突出“类自然性”而放在类矩形中。在这两种情况中，实际的语义都没有区别。

讨论

图13-26为表示雇用关系的关联类。公司与个人之间的雇用关系是多对多的。一个人可以

China-pub.com 下载

有多个工作，但只有一个工作是对指定的公司的。因为关联是多对多的，故薪水既不是公司又不是个人的属性，它必须是关系本身的属性。

老板与工人的关系不仅仅是两种人之间的关系，它是处于某种职业的人与处于另一种职业的人之间的关系，它是关联类和它自己之间的关联(Manages)。

下面的例子表示关联类和建模成类的具体化关系之间的区别。在图 13-27中，股票的所有者被建模成Person与Company之间的关联。关联类属性 quantity表示拥有股份的数量。这个关系建模成关联类是因为对任何 Person与Company对只有一个入口。

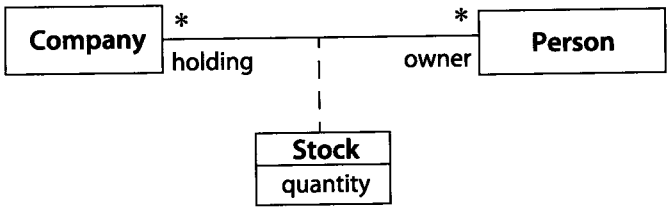


图13-27 具有属性的关联类

如图13-28，为了对购买股票进行建模，我们没有用关联类，因为对同一个人和公司来说可以有多次购买。然而，它们必须是可区别的，因为每次购买是不同的，有它自己的日期、价格以及数量。这个关系必须是具体化的，即形成具有自己的身份的可区别的对象。一个普通的类对是对这种情况建模的正确的方式，因为每次购买具有自己的身份，独立于与它联系的Person和Company类。

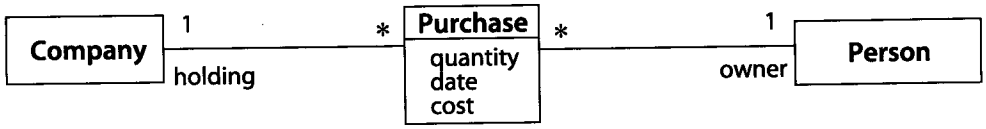


图13-28 具体化的关联

33. association end(关联端点)

关联端点是关联的一个结构部分，它定义了在关联中类的参与。在同一个关联中一个类可以连接到多个端点。关联中的关联端点有不同的位置而且有名称，并且通常是不可互换的。关联端点一旦脱离它的关联独立存在也不再有含义。

语义

结构

关联端点保持对目标类元的引用。它定义了在关联中类元的参与。关联的实例（链）必

China-pub.com
下载

须在指定位置包含给定类或它的一个后代的实例。类的后代继承关联的参与。

关联端点具有如下特性：

聚合	判定相关对象是聚集还是组成，有枚举值{none，aggregate, composite}。如果值不是none，那么关联被称作聚合或组合。缺省情况是none。只有二元关联才能是聚合或组合，并且只有一端可以是聚集或组成。
可变性	判定与对象有关的链集合是否可改变，有枚举值{changeable，frozen，addOnly}。缺省情况下是changeable。
接口说明者	相关对象及类元的说明类型的可选的约束（某些人称之为角色）。
多重性	与一个对象相关的对象的可能的数量，通常为整数值。
导航性	一个布尔值，表示是否可以将一个二元关联转化以得到有关一个类实例的对象或对象集合。缺省为true（可导航的）。
定序	判定一组不相关对象是否是有序的，枚举值有{unordered，ordered}。出于设计目的，sorted值也可以被用到。
限定符	寻找与关联相关的对象的选择器的属性表。
角色名	关联端点的名称，一个标识符字符串。该名称标识关联内对应类的特定角色。角色名在关联中以及在源类的直接和继承的伪属性（属性及类可见的其他角色名）中必须是唯一的。
目标作用域	判定链与对象或整个类是否相关，枚举值有{instance，classifier}。缺省为instance。
可见性	链是否可访问类而不能访问关联中相反的一端。可见性位于连接到目标类的一端。转换的每个方向都有它自己的可见性值。

表示法

关联路径的端点连接到对应类符号的矩形边缘上。关联端点特性用路径端点上或旁边的符号表示，该路径附属于一个类元符号（如图13-29）。下面是对每个特性的符号的简要的总结。

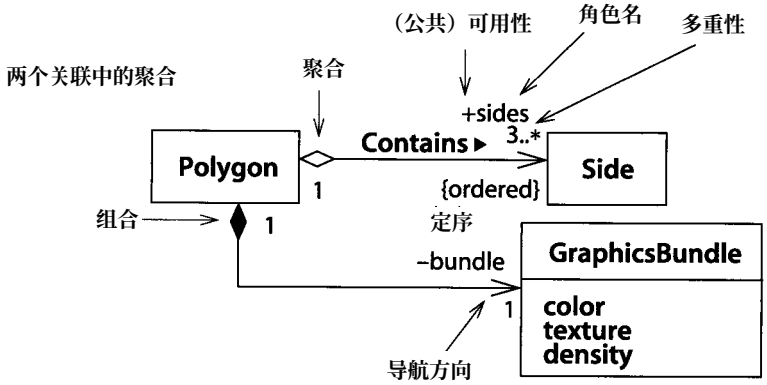


图13-29 关联端点上的不同装饰

China-pub.com 下载

聚合	位于聚集端点的一个空的菱形，组合为一个实菱形。
可变性	目标端点的文字属性{frozen}或{addOnly}，通常省略{changeable}。
接口说明者	角色名上的名称后缀，形式为：*typename*。
多重性	靠近路径端点的文本标记，形式为：*min..max*。
导航性	路径端点上的箭头表示了导航的方向。如果两个端点都没有箭头，那么即假设关联在两个方向上都是可导航的（因为在两个方向上关联都是不可导航的这种情况很少）。
定序	靠近目标端点的文本特性{ordered}，有目标类实例的有序表。
限定符	路径端点和源类之间的一个小矩形。矩形内包含了一个或多个关联的属性，即限定符。
角色名	靠近目标端的一个文本标记。
目标作用域	类作用域角色名是加下划线的，否则为实例作用域。
可见性	位于角色名前的可见性符号（+、#、-）。

如果在一个独立的角色上有多个符号，那么它们以如下顺序表示，从附属于类的路径的一端读起（如图13-23）：

限定符

聚合或组合符号

导航箭头

角色名和多重性应该放置在靠近路径的一端，这样它们不会与别的关联混淆。它们可以放置在线的任何一侧，不过把它们总是放在线的一侧更好一些，但有时当布局很密集时应以明确性为主。角色名和多重性可以放在同一个角色相对的两侧，或者也可以放在一起（如 *employee）。

标准元素

association，global，local，parameter，self。

34. association generalization(关联泛化)

关联泛化是两个关联之间的泛化关系。

见association，generation。

语义

关联之间的泛化是允许的，虽然这有点不太常见。与其他的泛化关系一样，后代元素必须加入到父的内容中（定义规则），并且作为父的外延（实例的集合）的子集。加入到内容中意味着加入附加的约束，一个子关联比它的父有更多的约束。例如，在图 13-30中，如果父关

China-pub.com
下载

联将类Subject和Symbol连接起来，那么子关联可以将类Order和OrderSymbol连接起来，其中Order是Subject的子，OrderSymbol是Symbol的子。作为外延的子集意味着子关联的每个链都是父关联的一个链，而反之却不能，该例遵循了这条规则，任何连接 Order和OrderSymbol的链也可以连接Subject和Symbol，但是不是所有连接Subject和Symbol的链都可以连接Order和OrderSymbol。

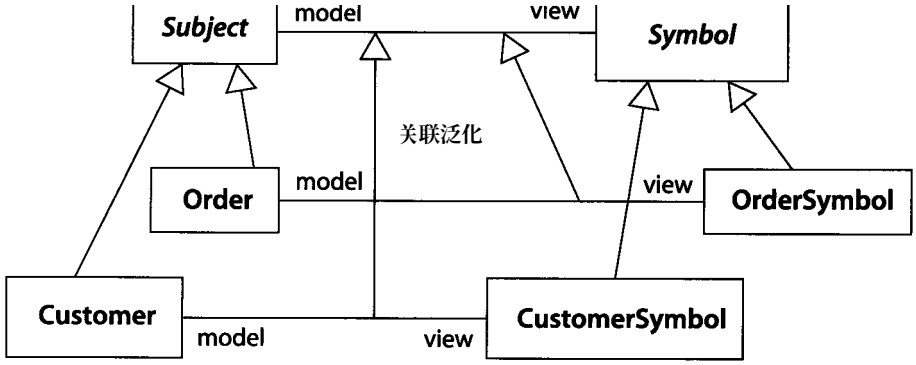


图13-30 关联泛化

表示法

泛化箭头符号（实线、空三角形箭头）将子关联和父关联连接起来，箭头在父关联一方。由于线段连接到其他的线段，关联泛化符号可能造成混淆，所以应该小心使用。

示例

图13-30表示Subject和Symbol之间普通model-view关联的两个特例：Order和Order Symbol之间的关联是一个特例，另一个是Customer和CustomerSymbol之间的关联。它们都将一个Subject类和一个Symbol类连接起来。Subject-Symbol关联可以看作是抽象的关联，而两个子关联是具体的。

这种由关联连接的成对类层次模式相当普遍。

标准元素

sdestroyed。

35. association role(关联角色)

关联角色是协作中两个类元角色的连接，它只适用于在由协作说明的特定情况下的两个

China-pub.com 下载

类元之间的关联。

见association，collaboration。

语义

关联角色是只在由协作所描述的情况下有意义和被定义的关联。它是协作的一部分，而不是其他情况下固有的关系。关联角色是协作的关键结构部分，允许有关上下文关系的描述。

在协作中，类元角色表示一个类元独立的出现，而区别于类元的其他出现和类元本身的声明。在基于协作的上下关系的情况下，它表示对基类元的使用的限制。同样，关联角色表示用于特殊关系中的关联，通常是对普通关联受限制的使用。关联角色将两个类元角色连接起来。当协作被实例化时，对象受限于类元角色而连接受限于关联角色。一个对象可以加入多个角色。

关联角色将两个或多个类元角色或是协作内的类元连接起来。它有对基关联的引用并且具有多重性，表示在一个协作的实例中可以有多少个链扮演角色。在某些情况下，协作中的一些连接可以被看作是对参与类之间通用关联的使用。协作表示了在协作内为了某种目的使用通用关联的某种方法。

在其他情况下，类元角色被协作外非法的关联连接起来。如果一个关联角色没有明确的基关联，那么它定义一个只在协作内合法的隐含关联。

表示法

关联角色与关联的表示法相同，即两个类元角色符号间的一条实线（如图 13-31）。

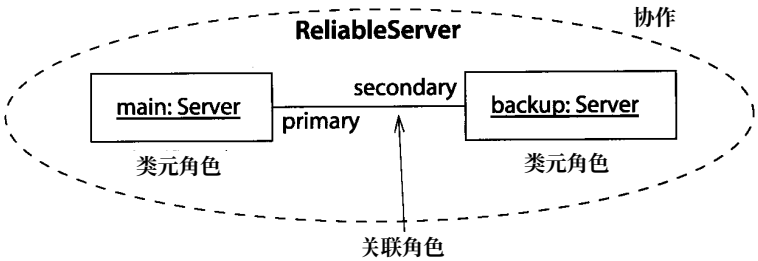


图13-31 关联角色

标准元素

new，transient。

36. asynchronous action(异步动作)

异步动作不要求发送对象暂停以等待结果，它是一个发送。

China-pub.com
下载

见send，synchronous action。

37. atomic(原子)

一个动作或操作，其执行必须作为一个单元被完成，原子也不能被部分地执行或被外部事件中断。通常，原子操作是小规模的且简单的操作，如赋值和简单的算术或字符串计算。一个原子计算出现在执行序列的确定点上。

见action，activity，run to completion。

语义

整个系统可以同时执行多个动作。如果我们说动作是原子的，这并不意味着整个系统是原子的。系统可以在几个动作之间处理硬件中断和分时操作。一个动作在它自己的控制线程内是原子的。被激发后它必须完成执行并且不能与其他同时存在的活动动作相互作用。系统可以处理中断和事件，但是不能影响原子动作。但是动作不能用作一个很长的处理机制，相对于对外部事件的反应时间来说，它们的持续时间应该是短暂的，否则系统不能实时地做出反应。

38. attribute（属性）

属性是类中特定类型的命名槽的描述，类的每个对象独立拥有类型的值。

语义

属性是类元中的命名槽，它描述了类元实例可能拥有的值。类元或其后代的每个实例拥有包含给定类型的值的槽。所有的槽是不同的，并且互相独立（类作用域属性除外，后文将对它进行描述）。如果属性是可修改的，那么在执行过程中，实例内槽所拥有的值可以被类型的不同值取代。

类元为它的属性形成了一个命名空间。包含在命名空间中的是伪属性，如离开类元的关联角色名和包含类元的泛化判别式。

结构

属性有下面的几个组成部分，详述见下：

可变性　　判定在初始化后槽中的值是否可以改变，是枚举值，缺省为changeable。可能的值有：

changeable　　对改变没有限制（缺省值）。

China-pub.com 下载

addOnly　附加值可以加到属性值的集合中。但是一旦被创建，值就不能被移动或改变（只在最大多重性大于1时有意义）。

frozen　在对象被初始化后值不能被改变。没有多余的值可以加入到值集合中。

初始值　初始值是说明值的表达式，在对象被初始化后被对象中的属性所拥有。表达式是一个文本字符串，包括用来计算表达式的语言的名称。当对象被实例化时表达式于语言的语境中计算。参见 expression，可得到更多详细资料。初始值是可选的。如果它是空的，那么静态模型不会说明由新的对象拥有的这个值，但是模型的其他部分会提供信息。一个明确的初始化过程，如构造函数，会替换初始值表达式。

类作用域属性的初始值在执行开始时初始化一次。UML没有说明不同类作用域属性初始化的相对顺序。

多重性　可以同时存在的属性值的可能数量。最常用的值“1”表示一个标量属性。值“0或1”表示一个有可选值的属性。属性类型域中缺少的值与其他值是有区别的（即值的缺少与值0是不同的，它是一个空集合）。其他多重性表示了潜在的多值属性。如果多重性不是一个单一的整数，那么由属性所拥有的值的数量可以改变。多重性“许多”表示一个无限制的值的集合。

名称　属性的名称，一个字符串，在类和它的祖先内必须是唯一的。它必须在类可到达的关联角色名中也是唯一的。

所有者作用域　由属性所描述的值槽在类的每个对象中可能是不同的或可由类的所有对象所分享。前者是实例作用域属性，后者是类作用域属性。大多数属性是实例作用域，它们持有一个特定对象的状态信息。类作用域属性持有一个完整类的信息，且有一个独立的槽对应整个类。然而，一个实例作用域属性是对一个值的描述，这个值直到对象被实例化后才存在。一个类作用域属性代表一个独立抽象值的声明，该值在系统整个生命周期存在。

目标作用域　由属性所持有的这个值可能是一个实例或是类本身。前者是实例作用域，它是缺省的。后者是类作用域，它不常见且通常涉及一些元建模。

类型　指定一个类或数据类型，槽中的值是它们的实例。值可能是给定类或数据类型的后代的实例。

可见性　判定属性是否能被其他类看见，枚举值有 public、private和protected。附加值可以被加入特定的模型编程语言中。

表示法

属性可以用文本字符串表示，它可以从语法上分析成不同的特性。缺省语法是：

China-pub.com
下载

《*stereotype*》$_{opt}$ *visibility*$_{opt}$ *name multiplicity*$_{opt}$：*type*$_{opt}$={*property-string*}$_{opt}$

可见性。可见性用一个标点符号表示。而在特性字符串中可见性可以用关键字表示。后一种形式必须用于用户定义或与语言相关的选择中。已定义的选择是：

+（公共的） 能够看见类的任何类可以看见属性。

#（受保护的） 类本身或它的任何后代可以看到属性。

-（私有的） 只有类本身可以看到属性。

名称。名称用标识符字符串表示。

类型。类型用指明类元的表达式字符串表示。类或数据类型的名称是合法的表达式字符串，表明属性的值必须是给定的类型。附加类型语法取决于表达式的语言。每种语言均具有从简单数据类型构造出新数据类型的语法。例如，C++有指针、数组和函数的语法，Ada有子范围的语法。表达式的语言是内部模型的一部分，但是它通常不在图中表示出来。假设它对于整个图是已知的或从它的语法上能明显看出。

类型字符串可以被隐藏，但是它仍然存在于模型中。

多重性。多重性用包含于方括弧中的多重性表达式表示，它位于属性名后。如果多重性值为“1”，那么表达式，包括方括弧，可以被省略。这表示每个对象只有一个保持给定类型值的存储槽。否则，多重性必须表示出来。见 multiplicity以了解其语法的详细表示。例如：

colors[3]：Saturation 具有3种颜色饱和度的数组

points[2..*]：point 具有2个或更多点的数组

注意0..1的多重性提供了空值（即缺少值，一个特殊的值的可能性。在多数数据类型域中空值不是一个值，它用域外的一个特殊值扩充了这个域。然而，对指针来说，空值通常是实现的一部分。下面的声明允许空值和空字符串之间的区别，C++和其他一些语言均支持该区别。

name[0..1]：String 若名称缺省则为空值

初始值。初始值用字符串表示。计算语言通常不明确表示出来，但是它存在于模型中。如果没有初始值，那么字符串和等价的符号都被省略。如果属性多重性包括值 0（可选的）并且没有给定明确的初始值，那么属性始于一个空值（零复制）。

可变性。可变性用关键字表示，如果没有选项给出，那么缺省值是 changeable。

标记值。零个或多个标记值可以附属于一个属性（或任何模型元素）。标记值的表示形式为*tag = value*，其中*tag*是标记的名称，*value*是一个文字值。标记值包含特性关键字，它是一个由括号括起的由逗号分隔的特性表。

作用域。类作用域属性用带下划线的名称和类型表达式字符串表示，否则该属性是实例作用域。这样表示的理由是类作用域属性是执行系统中的值，就像一个对象是一个实例值，所以都必须加下划线指明：

class-scope-attribute

图13-32给出了一些属性的声明。

China-pub.com 下载

+size: Area = (100,100)	公共的，初始值
#visibility: Boolean = invisible	受保护的，初始值
+default-size: Rectangle	公共的
maximum-size: Rectangle	类作用域
–xptr: XWindowPtr {requirement=4.3}	私有的，标记值

图13-32 属性

表示选项

编程语言语法。属性字符串的语法可以是一种编程语言的，如 C++或Smalltalk。特定的标记特性可以包含在字符串内。

风格指导

属性名用普通的类型界面表示。

讨论

类似的语法可用于说明限制符、模板参数、操作参数等等。

属性在语义上等同于组合关联。然而，内涵和用途通常是不同的。属性通常用于数据类型，即对没有身份的值使用属性。而关联用于类，即对有身份的值使用关联。其原因是对于有身份的对象，看清两个方向上的关系非常重要，而对于数据类型来说，数据类型通常附属于对象并且不了解它。

标准元素

persistence。

39. background information(背景信息)

同一个图或不同图中的类符号的每次出现可以有它不同的表示选择，例如，一个类的符号可以表示属性和操作而同一个类的另一个符号可以隐藏它们。工具可以提供附属于每个符号或整个图的风格表单。风格表单可以说明表示选择，而且它们适用于大多数的符号，而不仅仅是类。

并非所有的建模信息都可以用图形符号很好地表示，有些信息最好用文本或表格形式表示，例如，详细的编程信息最好用文本列表表示。 UML并不假定模型中的所有信息都可以表示成图，有一些信息只适用于表格。本文并不试图规定这种表格的格式或访问它们的形式，这是因为UML元模型中已经充分描述了其基本信息，而且表示表格信息也是工具的任务。不过，我们假定隐藏链可以在图形和表格中都存在。

China-pub.com
下载

40. become(变成)

变成是用于交互作用中的一种流依赖关系，在它当中目标对象代表源对象的一种新的形式，并且随后取代了源对象。

见class-in-state，copy，location。

语义

变成依赖关系是一种流依赖关系，它表示在交互中一个对象从另一个对象的导出。它表示了一个转换对象的动作。在一个变成流执行后，在计算中一个新对象替代了初始对象。如果仅仅表示一个对象值的改变，那么不必使用这个关系。相反地，这个关系用来表示对象中的质变，如状态的改变、类的改变或位置的改变。在这种情况下，模型包括对象的两个版本，但是变成关系表明它们的确是同一个对象在不同时间的两个版本，即它们有着同样的身份。

交互中的变成转换有一个顺序号，用来表明它相对于其他动作发生的时间。

表示法

变成流用虚箭头表示，由对象的早期版本指向较晚的版本，箭头持有构造型关键字《become》。箭头可以有一个交互中的顺列号，用来表示相对于其他动作发生的时间。变成转换可以出现于协作图、顺序图和活动图中。

在活动图中，变成转换可以用一个指向或背向对象流符号的虚箭头表示， become关键字可以被省略。

示例

图13-33表示一个用来打开桌面上一个关闭的目录图标的命令，其后是一个用来对现在打开的目录中的项目进行排序的命令。目录作为状态类对象被表示两次，并具有两个版本之间的变成转换。

后文的图13-132表示了在部署图中一个对象在两个节点间的迁移。

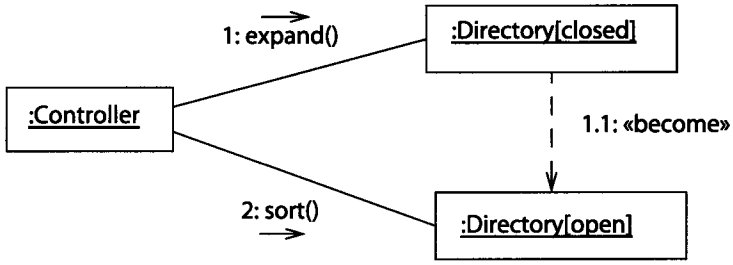


图13-33 变成流

China-pub.com 下载

41. behavior(行为)

一个操作或事件的可见的影响，包括它的结果。

42. behavioral feature(行为特征)

表示动态行为的模型元素，如操作或方法，它可以是类元的一部分。类元处理信号的声明也是行为特征。

标准元素

create，destroy，leaf。

43. behavioral view(行为视图)

行为视图是一个强调系统中实例行为的模型的视图，包括它们的方法、协作和状态历史。

44. binary association(二元关联)

两个类间的关联。

见association，n-ary association。

语义

二元关联是有着两个关联端点的关联，它是最常见的一种关联。因为二元关联的一个端点有一个唯一的另一端，所以二元关联对说明从对象到对象的导航路径特别有用。如果一个关联可以横越指定的方向，那么这个关联在指定方向上是可导航的。有一些特性针对 n元关联定义，如多重性，但是它们对二元关联更直观和有用。

表示法

二元关联用一个连接两个类符号的实线路径表示。修饰符号位于端点处，关联名可以放置在靠近实线处，离每一个端点有一定距离，这样不会误解成角色名。除了隐藏中心的菱形符号，二元关联的表示法与 n元关联相同。但二元关联有不适用于 n元关联的修饰符号，如导航性。

见association。

China-pub.com
下载

45. bind(绑定)

绑定依赖关系的表示法中的关键字。

见binding。

46. binding(绑定)

绑定即向参数赋值，并由一个参数化元素创建一个独立元素。绑定关系是一种依赖关系，它用来绑定模板以创建新的模型元素。

见bound element，template。

语义

一个参数化的定义，如操作、信号或模板，定义了一个元素的形式。然而，一个参数化的元素不能直接使用，因为它的参数没有确切的值。绑定是一种依赖关系，它表示向参数赋值以创建一个新的、可用的元素。绑定作用于操作并产生调用，或作用于信号并产生发送信号，或作用于模板并产生新的模型元素，其中前两个在执行时绑定以产生运行时间实体。除了作为例子或模拟结果它们通常不在模型中设置。参量值在执行系统内定义。

然而，模板在建模时绑定以创建用于模型中的新的模型元素。除了数据值之外，如字符串和整数，参量值可以是其他的模型元素，如类。绑定关系将值绑定到模版上，产生一个可以直接用于模型内的实际模型元素。

绑定关系有一个提供者元素（模板）、一个客户元素（新生成的绑定元素）和一张绑定到模板参数的数值表。绑定元素通过在模板体的拷贝中将每个参量值替换为对应的参数来定义。每个参量的类元必须与它的参数的已声明的类元或其后代相同。

绑定并不影响模板本身。每个模板可以绑定多次，每次产生一个新的绑定元素。

表示法

绑定用附属于虚箭头的关键字《 bind》表示，该虚箭头将生成元素（在箭头的尾部）和模板（在箭头的头部）连接起来。实际参量值用一个位于《 bind》关键字后，由圆括弧包含并由逗号分隔的文本表达式表示。

另一种更简洁的绑定表示法使用名称取代了箭头。绑定元素用模板的名称后的一个用尖括弧括起来的逗号分开的文本表达式表示（ $<argument_{list},>$）。

无论是哪一种情况下，每个参量都被表述为在模型建立时静态计算的文本字符串，它不像操作或信号参量一样动态计算。

在图13-34中，用箭头的显式形式声明了一个新的类 AddressList，它的名称可以用在模型或表达式中。隐含形式 Varray〈Point，3〉声明了一个没有名称的匿名类。它可以在使用隐含语法的表达式中使用。在这两种情况下都可以声明附加的属性或操作。如果需要扩展那么也可以声明子类。

China-pub.com 下载

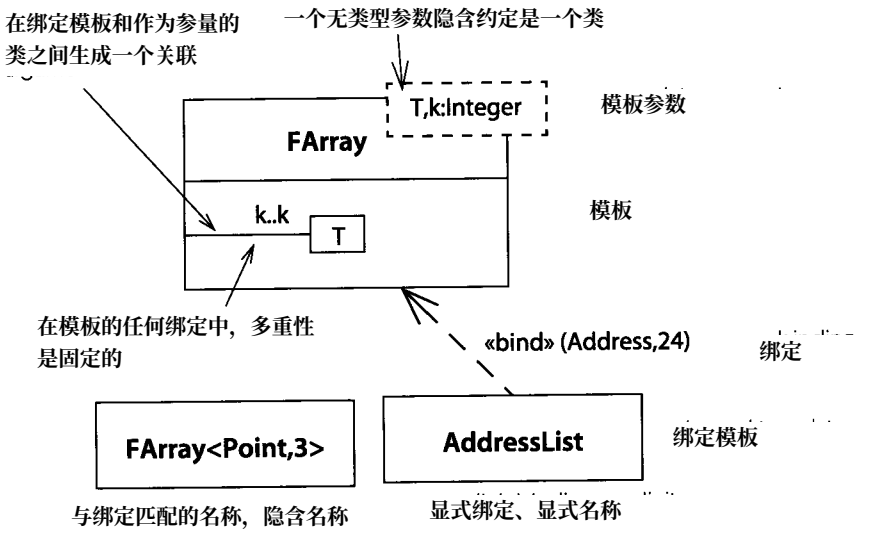


图13-34 模板声明和绑定

标准元素

bind。

47. Boolean(布尔型)

布尔型是值为true和false的枚举类型。

48. Boolean expression(布尔型表达式)

计算布尔值的表达式，用于监护条件中。

49. bound element(绑定元素)

通过将参量值绑定到模板参数而产生的模型元素。

见binding，template。

语义

模板是一组可能元素的参数化描述。为了得到实际元素，模板参量必须绑定于实际值。

China-pub.com
下载

每个参数的实际值是由绑定出现的作用域所提供的表达式。大多数参量是类或整数。

如果作用域本身是一个模板，那么外层模板的参数在绑定初始模板时可用作参量，得到重新参量化的效果。但是一个模板的参数名在它的主体外没有意义。两个模板中的参数名不能仅仅因为它们有相同的名称就假定它们等同，子例程参数仅可以依据它们的名称而对应。

绑定元素完全由它的模板说明，所以它的内容不能扩展。例如，不允许声明类的新属性和操作，但是一个绑定类可以被子类化而且子类以通常方式扩展。

示例

图13-35表示了一个模板的再绑定。PointArray是一个具有一个属性的模板——大小为n。我们想从现有的模板FArray中得到它，FArray有两个参数——元素类型T，大小k。为了得到它，FArray模板的参数k绑定于PointArray模板的参数n。FArray模板的参数T绑定于类Point。这样可以产生将一个参数从初始模板中移走的效果。为了用PointArray模板生成一个Triangle类，将大小属性n绑定于值3。为了生成一个Quadrilateral类，将大小属性n绑定于值4。

图13-35也表示将模板Polygon作为类Shape的孩子。这表示任何与模板Polygon绑定的都是Shape的子类，Triangle和Quadrilateral都是Shape的子类。

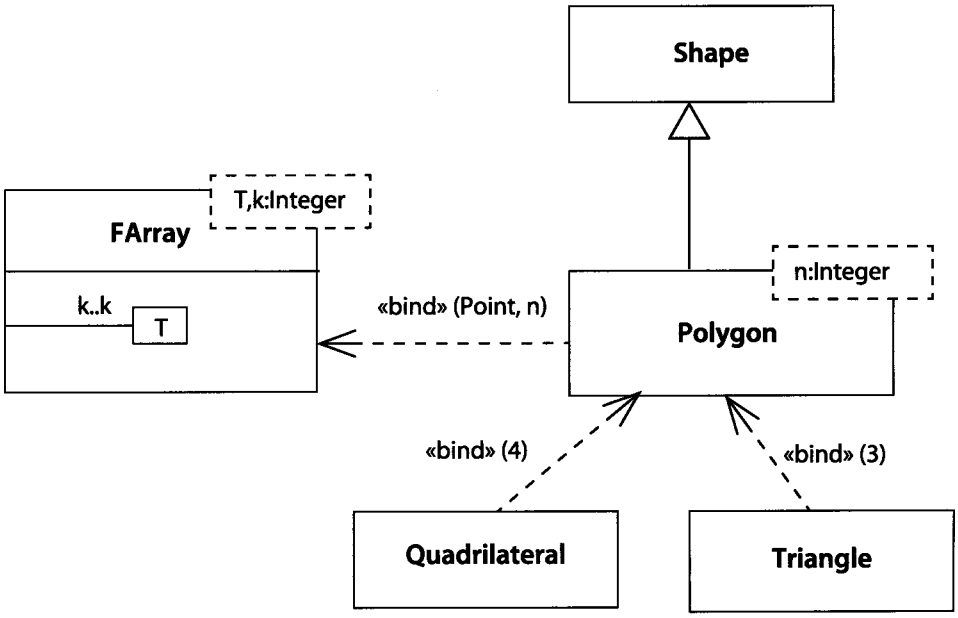


图13-35 模板再绑定

表示法

绑定元素可以用一个从模板指向绑定元素的虚箭头表示，箭头连有关键字《bind》。它也

China-pub.com 下载

可以用语法*TemplateName*<*argument*$_{list}$,>表示，用名称匹配确定模板。文本形式避免了表示模板或画出指向它的箭头。当绑定元素被用作一个属性或操作参数的类元时，这种形式特别有用。

见binding。图13-34给出了一个例子。

在一个绑定类中属性和操作分隔区通常被压缩，因为它们在绑定元素中不能改变。

绑定元素名（使用尖括号的隐含匿名形式或使用“绑定箭头”的显式形式）可以用在任何一个参数化模板的元素名可用的地方。例如，绑定类名称在类图中可以用作属性类型或一部分操作特征。图13-36给出了一个例子。

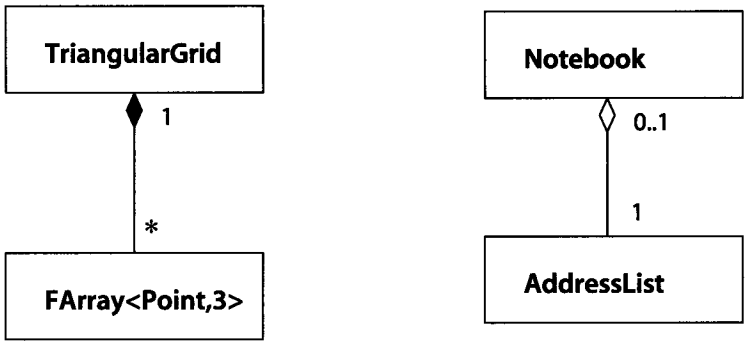


图13-36 关联中绑定模板的使用

讨论

类元是参数化的明显候选。它们的属性、操作或关联类元的类型是模板中的常用参数。参数化的协作是模式。从某种意义上说，操作本质上是参数化的。其他元素的参数化用途不那么明显，但是很可能进一步找到用途。

50. branch(分支)

分支是状态机中的一个元素，在它当中一个触发器可能导致多个可能结果，每个结果有它自己的监护条件。

见fork，join，junction state，merge。

语义

如果同一个事件依据不同的监护条件有不同的影响，那么它们可以建模成有着相同事件触发器的分离转换。然而，在实际情况中允许一个独立的触发器驱动多个转换是很方便的。在通常情况下常常是监护条件覆盖每种可能，这使得一个事件的发生能保证触发一个转换。

China-pub.com
下载

分支是转换的一部分，它将转换路径分成多个部分，每一部分有单独的监护条件。事件触发器放在转换前面的公共部分。分支的输出部分可以与另一个分支的输入部分连接而组成一棵树。树的每个路径代表一个不同的转换。转换中一个路径上所有条件的结合与概念上在转换激发之前被计算的一个独立条件相同。转换在一个独立步骤中激发，而不管它是否作为有分支的树出现。树仅仅是为了建模的方便。

在一个活动图内，离开一个活动状态的分支通常是完成转换 —— 即它们缺少显式的事件触发器，它们是在状态内活动完成时隐含触发的。如果有监护条件和分支，那么它们覆盖所有种可能是很重要的，因为这样可以保证一些转换被激发。否则，会因为输出转换不再重新激发而使活动图冻结。

表示法

分支可以用以不同的监护条件在多个转换弧上重复一个事件触发器来表示，也可以完成转换表示，如在活动图中。

然而，方便起见，转换箭头的头部可以连接到一个菱形符号，这个菱形符号表示了一个分支。转换箭头用触发器事件标记，但是其上不能有动作。任何动作都走到转换的最后部分。

一个菱形符号可以有一个或多个离开它的箭头。每个箭头用一个监护条件标记。保留字else可以用作一个监护条件。如果其他的（显式）监护条件为假，那么它的值就为真。每个箭头的头部可以连接到另一个分支或状态。连接到状态的箭头可以有一个相关的动作标记。

分支树的作用与将树扩展成对应于树的每个分支的分离转换弧的作用相同，它们分享同一个触发器事件，但是分别具有它们自己的监护条件、动作和目标状态的结合。图 13-37是表示分支的两种方法。

菱形符号也可以表示合并，与分支相反，这时两个不同的路径合并在一起，如图 13-38所示。合并的情况具有两个或更多的输入箭头和一个单独的输出箭头，不需要监护条件。

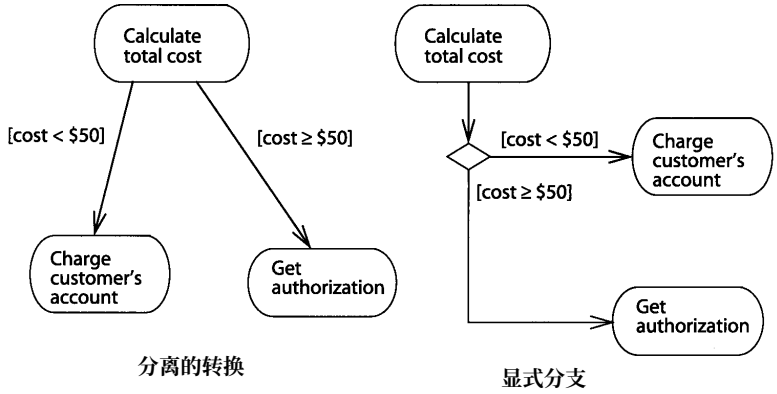


图13-37 表示分支的两种方法

China-pub.com 下载

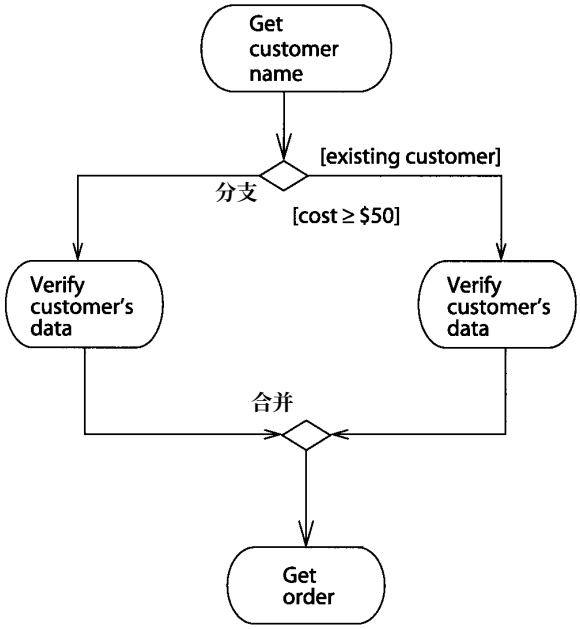


图13-38 分支和合并

51. call(调用)

用于激活一个操作。

见activation，call event，send。

语义

调用是在一个过程的执行点上激发一个操作，它将一个控制线程暂时从调用过程转换到被调用过程。调用过程的执行在调用时被阻断。在操作执行中调用者放弃控制并且在操作返回时重新获得控制。被调用过程从调用者那里得到一张参量表和对调用过程的一个隐含返回点，即调用点的下一条命令。被调用过程返回时，提供一张返回值表。

通常，调用在调用者的地址空间中执行，但是调用的语义并不局限于这一点。实际上，在一个分布式系统中这是不可能的，因为调用的接收者物理上是与调用者分开的。更重要的在于对调用过程位置和环境的隐含返回链的建立，它使得控制重新返回到调用者。调用过程位置在文本过程中建模成文本，在状态机中建模成状态。调用环境可以建模成激活。

调用的使用依赖关系对下列情况建模，在该情况中一个客户类的操作（或操作本身）调用提供者类的操作（或操作本身）。它用《call》构造型表示。

China-pub.com
下载

表示法

在顺序图或协作图上，调用用指向目标对象或类的文本消息表示。

调用依赖关系用从调用者指向被调用类或操作的含有构造型《 call》的虚线箭头表示。在编程语言中，大多数的调用都可以用文本过程的一部分表示。

52. call event(调用事件)

调用事件是接收一个操作的调用的事件，它由动作在状态机转换上实现。

见call，signal。

语义

调用事件是实现操作的一种方法，它是过程执行的另一种方式。如果一个类将一个操作的实现说明为一个调用事件，那么操作的调用将被看作一个触发类的状态机中的转换的事件。这允许比单一方法过程更加分散的操作的实现。

如果一个类用调用事件去实现一个操作，那么它的状态机必须具有由调用事件触发的转换。调用事件的特征与操作的相同：它们的名字和参数都相同。

当一个操作的调用发生时，目标对象的状态机被引用，并且如果调用事件符合一个活动转换(不同于当前活动状态)上的触发器事件，那么它触发该转换。如果转换被激发，则实现了实际效果。实际效果包括任何动作序列，包括 return（*value*）动作，其目的是将值返回给调用者。

当转换执行结束时，调用者重新获得控制并且可以继续执行。如果操作需要一个返回值而调用者没有接收到，或接收到不符合声明类型的返回值，说明模型有问题。如果操作不需要一个返回值，说明没有问题。请注意，如果没有转换被调用事件触发，则控制立即返回到调用者。如果操作需要一个返回值，说明模型有问题。否则，调用者仅仅立即继续执行。

如果接收者是一个主动对象，那么当接收者的状态机静止时，即从运行到完成的所有步骤都已实现时，调用事件被处理。

表示法

调用事件用状态图中与对应类上的操作匹配的事件触发器表示。转换上的动作序列可以有一个return声明，如果是这样，声明表示返回值。

示例

图13-39表示了一个可以被Locked 和Unlocked的账户。deposit操作总是向账户里加钱，而不管它的状态。如果账户是打开的，那么 withdraw操作取走所有的钱；如果账户是锁着的，

China-pub.com 下载

则取不走钱。withdraw操作被作为一个调用事件实现，它触发每个状态的内部转换。当调用发生时，根据活动状态，执行一个或其他动作序列。如果系统是锁住的，返回 0；如果系统是打开的，账户中的所有钱返回并且重新计数为 0。

调用过程文本

```
deposit (10);
...
amount := withdraw ( )
```

取出全部金额或未取出金额，取决于该帐户是否被加锁

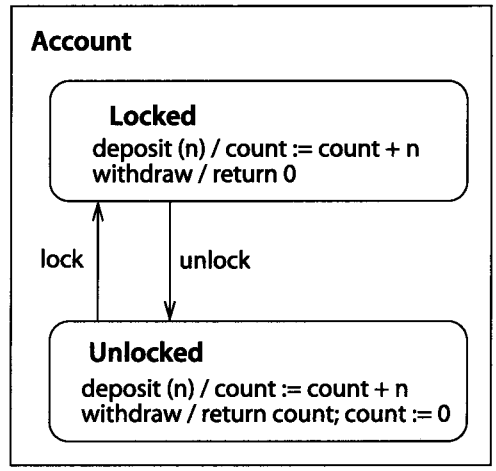


图13-39 调用事件

53. canonical notation(规范表示法)

UML定义了规范表示法，它用单色线和文本表示任何模型。这就是 UML模型标准的“出版格式”，可用于印刷图。

图形编辑工具可以扩展规范表示法并且提供交互能力。例如，一个工具可以提供突出显示屏幕中被选择元素的能力。其他交互能力包括模型中的导航和按照选择的特性过滤显示的模型。这种格式是暂时的，并且不受 UML制约。交互显示减少了模棱两可的弊端。所以， UML标准的焦点是每一种工具都支持的印刷的规范形式，一个交互工具可以而且应该提供交互扩展。

54. cardinality(基数)

基数是集合中元素的数量。它是一个具体的数值。基数不同于多重性，多重性是一个集合拥有的可能基数的范围。

讨论

“基数”这个术语被许多作者错用为我们所说的多重性，但是基数在数学上定义为一个数

China-pub.com
下载

字，而不是数的范围。这正是我们所用的定义。

55. change event(改变事件)

由于所引用的一个或多个值的改变而使布尔表达式得到满足的事件称为改变事件。

见guard condition。

语义

改变事件包含由一个布尔表达式指定的条件。事件没有参数。当条件所依赖的一个或多个变量的改变而使条件（从假）变成真时，事件就发生。

该类型的事件隐含一个对条件的连续的测试。而实际上，通过分析条件输入改变的时间点，开发者可以在良好定义的、抽象的时间上执行测试，这样就不需要连续的测试了。

当表达式的值从假改变到真（一个正值状态变化）时，事件就发生。当这种情况发生时事件发生一次，除非值再次先变成假，否则事件不会再发生。

请注意它与监护条件的区别。当转换上的事件发生时，监护条件只计算一次。如果条件是假的，那么转换不激发并且事件被遗失（除非这个事件又触发了其他的转换）。一个改变事件隐含连续计算，并且当值变成真时事件发生一次。在那个时刻，它可以触发一个转换或被忽略。如果它被忽略，因为条件还是真所以改变事件不触发处于后继状态的转换。改变事件已经发生，因此被抛弃。条件必须变成假，然后再变成真时，才能引发另一个改变事件。

布尔表达式的值必须是对象的属性，该对象拥有包含从它可到达的转换或值的状态机。

表示法

与信号不同，改变事件没有名称并且不被声明。它们仅仅被用作转换的触发器。改变事件用关键字when后跟圆括弧中的布尔表达式表示，例如：

when（self.waitingCustomers>6）

讨论

改变事件是对条件满足的测试。它实现起来十分昂贵，尽管可采用一些技巧来编译它使它不必连续地测试，但是改变事件本身隐含着高成本和一种特定的关系，即值的改变与由它触发的结果之间的直接因果关系。有时，这是令人满意的，因为它封装了结果，但应该小心使用改变事件。

改变事件表示对对象可见的值的测试。如果一个对象中属性改变意味着触发另一个对象（该对象不知道第一个对象的属性改变）的改变，那么这种情况应该建模成属性拥有者上的改变事件，它触发一个内部转换，向第二个对象发送信号。

China-pub.com 下载

改变事件并不显式地向任何地方发送。如果试图与另一个对象进行显式地通信，应该用信号。

改变事件可以通过多种方式实现，有些通过在合适的时间点在应用程序内部进行测试，而有些则通过利用基本的操作系统机制来完成。

56. changeability(可变性)

说明某属性值或者链是否可变的一种特性。

语义

本特性可以赋给关联端或者属性。此特性还可以赋给一个类，表明该类中的所有关联和属性都满足本特性（例如，特性值为“冻结”表示该类的对象的值在初始化后不可改变）。在特性表中表示可变性的关键字可取下列枚举值：

changeable	属性值可以任意变换，包括在多重性允许的范围内增加或者删除值。链可以随意变更，或者在多重性及其他约束允许的范围内被增加或者删除。如果没有指定其他值，则此值为缺省值。
frozen	属性值在初始化后不可变更，不可增加或者删除变量。当关联的另一端（即与带有冻结值的一端相对的另一端）的对象初始化后，则不可增加、删除或者修改链。但是，当另一端生成新的对象时，作为初始化的一部分，将生成指向冻结端的新链。
addOnly	如果多重性不是固定值，或者已经达到最大值，则可以增加新的属性值。一旦生成，只要类的对象仍然存在，则不能更改或删除属性值。可以增加新的链，但是不能在其生成后进行修改或删除。当一个参与对象被销毁时，其中包含的所有链，无论是否是 addOnly类型的，都将被删除。

57. child(子)

泛化关系中更具体的元素。对于类而言为子类。一个或者多个子关系的链称为后代。反义词：parent。

语义

子元素继承了其父元素的特征（同样间接继承了其祖先的特征），而且可以声明自己附加的特征。它还继承了涉及其祖先的关联和约束。子元素遵循替代原则 —— 即描述符的实例满足任何该描述符祖先声明的变量。子的实例是其父的间接实例。

China-pub.com
下载

58. class(类)

说明一系列拥有相同的属性、操作、方法、关系、行为的对象集。类表示被建模系统中的一个概念。根据模型种类的不同，此概念可能是现实世界的（对于分析模型而言），或可能含有算法和计算机实现的概念（对于设计模型而言）。类元是类的泛化，包含其他类似类的元素，如数据类型、参与者、构件。

语义

对对象集的数据结构及行为的描述称为类。类用于声明变量。作为变量值的对象必须属于一个类，该类应与声明该变量的类兼容，即该类必须是声明变量的类或其后代类。类还用来实例化对象。一个创建操作生成给定类的一个新实例。

类的实例对象是这个类的直接实例，同时是该类的父类的间接实例，对象保存每一个属性的值，接受其类的所有操作和信号，同时还可以出现在与类或类的父类相关的关联链中。

有些类可能不会被直接实例化，而只用于描述其后代类共享的结构，这种类被称为抽象类。可以实例化的类称为具体类。

类还可以被当作全局对象，该类的任何类作用域属性都是这个隐含对象的属性。这样的属性有全局性，在整个系统中拥有唯一的值。类作用域的操作是指适用于类本身，而不适用于对象的操作。最常见的类作用域操作是创建操作。

在UML中，类是一种类元，类元包括一系列类似于类的元素，但是它的完整表述仍在类中。

结构

类具有名称以及操作、方法和属性列表。类可能属于某种关联、泛化关系、依赖关系或者约束关系。类在其命名空间内声明，如在一个包或者其他的类中，类在其名称域内有不同的特性，如多重性或者可见性。类还拥有其他不同特性，如用于说明它是抽象类还是主动类。还有一个状态机可用来说明类的反应行为，即当接收到某事件时的反应。类可声明其所处理的事件集（包括异常处理）。类可为由零个或多个的接口提供实现，或为一个行为实现提供不同类型。接口列出了该接口支持的并由类实现的操作集。

类包含属性表和操作表，它们各自在类中建立一个命名空间。继承的属性和操作同样出现在相应的命名空间内。属性的命名空间中还包括伪属性，如用于遗留类和涉及该类或它的一个祖先的泛化关系的区分符的关联的角色名。每个名称在类及其祖先中仅能声明一次，否则将产生冲突，模型建立错误。如果所代表的操作相同，那么相应的操作名可重复，否则将产生冲突。

类也是一个命名空间，并为嵌套类元声明建立作用域。嵌套类元并不是类的实例的结构性部分。在类对象与嵌套类对象之间并没有数据联系。嵌套类是一个可能被外层类的方法所

China-pub.com 下载

使用的类的声明。类中声明的类是私有的，除非明确地设定为可见，否则在该类的外部是不可访问的。没有可见的表示法来表示嵌套类的声明。只有在工具中，才有可能用超链接对它们进行访问。嵌套的名称必须用路径名来引用。

表示法

类用实线边框的矩形来表示，矩形用两条水平线分为三栏。上面一栏包含类的名称以及其他适用于整个类的特性。中间一栏包含属性表。下面一栏包含操作表。中间和下面栏在类符号中可隐藏。

使用。类在类图中声明，并且在许多其他图中被使用。 UML为声明和使用类提供了一种图形表示法，同时为了在其他模型元素的描述中引用类提供了文本表示法。类图中对类的声明定义了类的内容：类的属性、操作以及其他特性。其他图中定义与类相关的其他关系和附件。

图13-40表示具有属性和操作的基类声明。

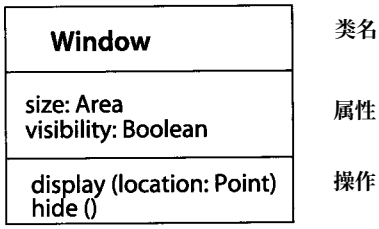


图13-40 基类声明

图13-41表示与上图同一类的声明，包括更详细的内容，主要是与实现有关的性质，如可见性、类层次作用域的创建操作以及依赖于实现的操作。

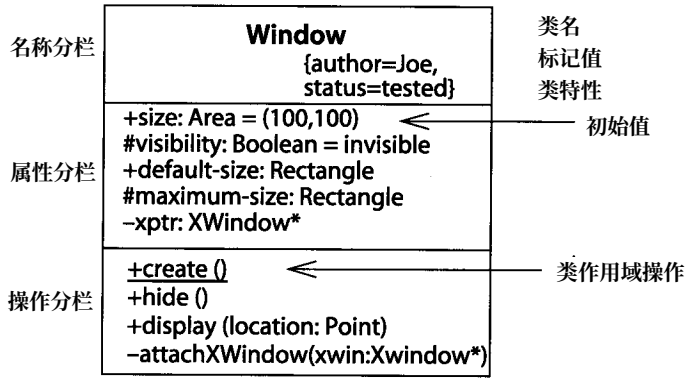


图13-41 带有可见性特征的详细的类声明

所有与类相关的内部信息在图 13-42中被隐藏了。这些信息仍出现内部模型中，通常会在

China-pub.com
下载

至少一幅图中表示。

图13-42 隐藏了所有详细内容的类符号

表示选项

隐藏分栏。属性和操作其中之一或全部可以被隐藏（图 13-43）。隐藏的分栏将不出现分隔线。如果某个分栏被隐藏，则无法推断其中的元素是否存在。请注意，空的分栏（即有分隔线，但是没有内容）说明相应的列表中没有元素。如果使用某种筛选程序，则说明没有符合其要求的元素。例如，只有公共操作是可见的，则相应的操作分栏为空说明没有公共操作。而对于私有操作不能得出任何结论。

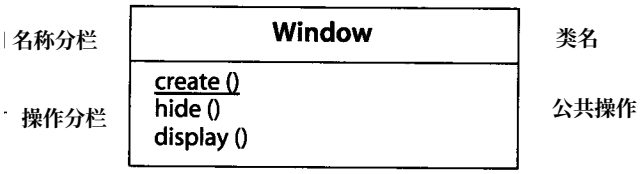


图13-43 隐藏了属性以及非公共操作的类声明

附加分栏。附加分栏用于表示其他预定义或者用户定义的模型特性，如事务规则、职责、变体、信号处理、异常处理等等。附加分栏的顶部写有分栏名称，并用特殊的字体与其内容区分开来（图 13-44）。标准分栏（属性、操作）则不需要分栏名称，不过在只有一个分栏可见时，可能会用分栏名以示强调或区别。多数的分栏中只是简单的字符串表，其中的每一个字符串代表一个特性。这里的“字符串”包括图标和嵌入的文档，如电子表格或者图形。还可能有更复杂的格式，但是 UML并没有为这些格式作特别说明，这是用户或工具的职责。如果分栏的性质可以从其中内容的格式判断出来，则分栏的名称可以省略。

见font usage，string。

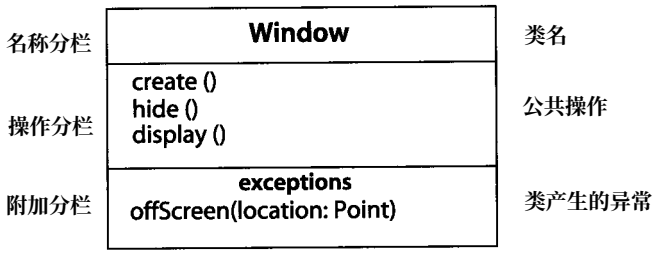
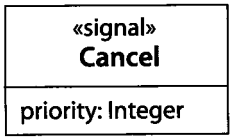


图13-44 带有额外命名分栏的类声明

China-pub.com 下载

构造型。构造型用位于类的名称上方的括在书名号（《》）内的文本字符串来表示（图 13-45）。还可以在名称分栏的右上角用图标来代替该文本字符串。带有构造型图标的类符号可以被“压缩”为构造型图标，并在图标内或者图标下方标明类的名称（见后文图 13-170）。类的其他内容被隐藏。

见stereotype。



完整声明

省略细节

图13-45 带有构造型的类

格式指导

- 用正常字体在类名称的上方居中书写构造型名称。
- 用黑体字居中或者向左对齐书写类名。
- 用正常字体向左对齐书写属性和操作。
- 用斜体字书写抽象类的名称，或者抽象操作的特征标记。
- 在需要处表示类的属性和操作分栏（在一套图中至少出现一次），在其他上下文或提及处可以隐藏它们。最好为一套图中的每个类定义一个“家”，并在此处对类进行详细描述。在其他位置则可以使用类的最小化表示形式。

讨论

类的概念既适用于逻辑建模中的一些用例，也适用于实现。它既包括类型的概念，又包括实现类的概念。在 UML中，在特定的语义变更点，一个对象可以有多个类，还可以在运行时修改类。在多数程序设计语言中常见的有更严格限制的类的表示法，可以被视为特殊的类。

标准元素

implemetationClass，type。

59. class diagram(类图)

类图是静态视图的图形表达方式，表示声明的（静态的）模型元素，如类、类型及其内容及相互关系。类图可以表示包的视图，包含嵌套包的符号。类图包含一些具体的行为元素，

China-pub.com
下载

如操作，但是它们的动态特征是在其他图中表示的，如状态图和协作图。

见classifier，object diagrams。

表示法

类图是用图形方式表示的静态视图。通常，为了表示一个完整的静态视图，需要几个类图。每个独立的类图不需要说明基础模型中的划分，即使某些逻辑划分，如包是构成该图的自然边界。

60. class-in-state(状态类)

一个类以及其对象可能拥有的状态。

见activity graph。

语义

带有状态机的类有许多状态，每个状态说明了处于这种状态的实例的行为、值以及约束。在某些情况下，某些属性、关联或者操作仅当对象处于特定的一种或几种状态时才有效。在其他情况中，操作的参量必须是参与状态下的对象。通常情况下，上述特殊情况只是行为模型的一个部分。但是有时，当在交互视图或者静态视图中直接建模时很有用。

状态类仍然是类，只是带有类的对象可能拥有的合法状态。如果类有并发的子状态，那么状态说明可能为一系列该类的对象可能同时拥有的子状态。状态类可以被当作一种类元。它的行为类似所描述类的一个子类。它可以被当作变量类或参数类使用。它可以参加那些只

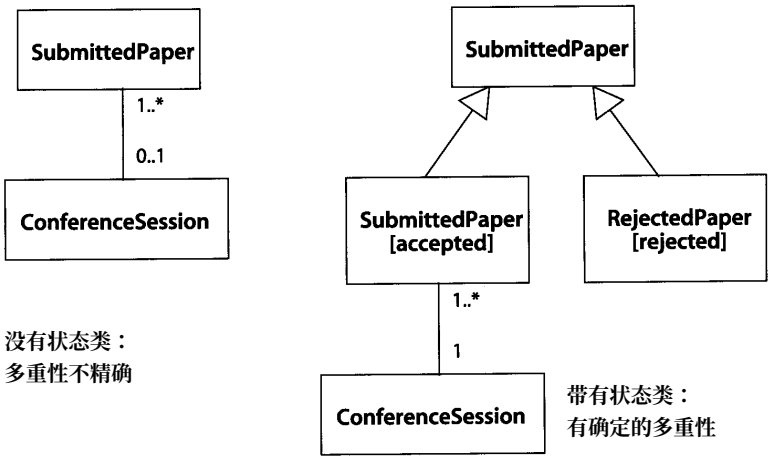


图13-46 状态类

China-pub.com 下载

适用于所给定状态的对象的关联。考虑图 13-46中SubmittedPaper与ConferenceSession之间的Assignment关联。这种关联仅在 SubmittedPaper处于accepted状态时才适用（目标多重性为1），而在rejected状态时则不存在。对于 SubmittedPaper而言，目标多重性为 0或1，因为类中同时含有 accepted和rejected的报告。然而在对处于 accepted状态下的Submitted Paper和ConferenceSession间的关联建模时，那么其目标多重性就是 1。

在表示活动图中操作的输入输出值时，状态类元素也是有用的。

表示法

状态类用类名后跟方括号内的状态名（ *Classname*[*statename*]）表示。方括号中也可以是用逗号分隔的几个并发状态名，表示对象可以是这些状态中的几种。

讨论

状态类和动态分类是为了实现允许对象在其生命周期内改变结构这一目的而采用的两种不同方法，具体采用哪种根据实现环境。

61. class name(类名)

每个类都必须有一个非空的类名，这在类的包容器（如包或者包容类）内对于类元而言是唯一的。名称的作用域是其包容包和可以看到该包容包的其他包。

见name，其中有关于命名以及唯一性规则的完整说明。

表示法

类名位于类矩形的顶部分栏中。名称分栏中可能还有关键字或者构造型名称和 /或构造型图标，以及用花括号括住的一系列标记值（如图 13-47）。

处于书名号内可选的构造型关键字可置于类名的上方，构造型图标可置于分栏的右上角。构造型名称不得与预定义的关键字重复，如 enumeration。

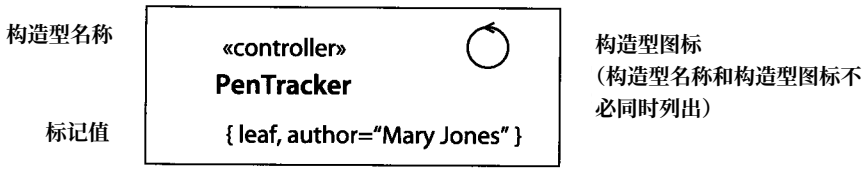


图13-47 名称分栏

China-pub.com
下载

接下来表示类的名称，类名用粗体字在分栏的水平正中标出。如果此类为抽象类，则类名用斜体表示。注意，任何显式的泛化状态的说明（如抽象或者具体），其字体应大于类名所用的字体。

在类名下，可以在花括号中列出标识特性的字符串表（元模型属性或者标记值）。此列表可以表示那些在UML中没有符号表示的类级属性，还可以表示标记值。可以用某些无值的关键字来表示某种特性与值的联合体。例如，叶类表示特性 {leaf}，与{isLeaf=true} 作用相同。

缺省情况下，假定表示在某个包内的类是在此包内定义的。要引用在其他包中定义的类，使用下列语法：

Package-name ∷ *Class-name*

作为名称分栏中的名称字符串（图 13-48）。用两个冒号（∷）将包的名称与类名连在一起，从而提供完整的路径名。如果路径名可以区分它们，则在不同包内的不同的类中可以使用相同的类名，但是这种情况容易造成错误，应该尽量避免。

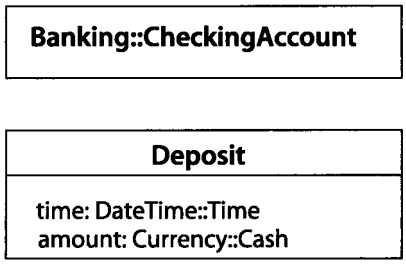


图13-48 在其他包中的类的路径名

在文本表达式中，也会引用类，尤其是在属性和变量的类型说明中。在文本表达式中，引用一个类只要使用它的名称，包括可能包的名称，这服从于表达式的语法规则。

62. classifier(类元)

类元是一种描述行为和结构特征的模型元素。类元的种类包括类、行为、构件、数据类型、接口、节点、信号、子系统以及用例。类是最常见的类元。虽然每种类元都有各自的元模型为代表，但是它们都可以按照类的概念来理解，只是在内容和使用上有某些特殊限制。类的大多数特性都适用于类元，通常只是为每种类元增加了某些特殊限制条件。

见generalizable element，static view。

标准元素

enumeration，location，metaclass，persistence，powertype，process，semantics，stereotype，thread，utility。

China-pub.com 下载

63. classifier role(类元角色)

协作的一个片断，用于描述各个参与者在协作中的角色。

见collaboration。

语义

协作说明了众多参与者之间的交互方式，它是类或者数据类型的实例。类元角色是对一个参与者的描述。每个角色处在自己的语境中，是类元的一个独特的使用。一个类元可能有多个角色，每个角色在一个协作中与其他角色有着不同的关系。一个角色不是独立的对象，而是某个可能参与协作实例的所有对象的描述。每当这个协作实例化时，就有不同的对象与链来扮演这些角色。

每个类元角色有其类元（其基础）的引用以及多重性。基础类元限制了可以充当角色的对象种类。对象所属的类可以与该基础类元的类相同，或是其子类。多重性说明在协作的一个实例中，同一时刻有多少个对象来充当这种角色。

类元角色可以有名称，也可以匿名。当需要多重多类时，类元角色可以有多个基础类元。

类元角色可以通过关联角色与其他类元角色相连接。

对象。一个协作代表了为了完成某个目标而共同工作的一组对象。角色表示一个对象(或一组对象)在完成目标的过程中所应起的那部分作用。对象是角色所属的基类直接或者间接的实例。在协作中，不需要基类所有的对象都出现，同一个基类的对象在一个协作中也可能充当多个角色。

在不同的协作中，同一个对象可以充当不同角色。协作代表了对象的一个方面。单独的一个物理对象可能包括多个方面，因此隐含了该对象与其起作用的协作间的联系。

表示法

类元角色用类元的符号（矩形）表示，符号中带有用冒号分格开的角色名和类元名称，即*rolename*：*BaseClass*。一个角色不是独立的对象，而是一个用于描述不同类元实例中出现的多个对象的类元。

角色名和类元的名称都可以省略，但是分号必须保留，从而与普通的类相区别。在一个协作中，由于所有的参与者都是角色，因而不易混淆。

类元角色可能会表示类元特征的一个子集，即用于给定的语境中的属性和操作。其余未被用到的特征将可以被隐藏。

图13-49表示了类元角色不同的表示法。

命名角色，多重性为1　　未命名角色，多重性为1　　命名角色，多重性大于1

图13-49　类元角色

China-pub.com
下载

标准元素

destroyed、new、transient。

64. client(客户)

客户是指一个需要其他元素提供服务的元素。它用于描述依赖关系中的角色。在表示法中，客户出现在依赖关系箭头的尾部。反义词：supplier。

见dependency。

65. collaboration(协作)

协作是对对象和链总体安排的一个描述，这些对象和链在语境中通过互操作完成一个行为，如用例或者操作。协作由静态部分和动态部分组成。静态部分描述在协作实例中对象和链可能承担的角色。动态部分包括一个或多个动态交互，表示在执行计算过程中不同时间里协作中的消息流。

见association role，classifier role，interaction，message。

语义

一组对象在给定的语境中，为了完成某个目标而交换消息，从而实现了一种行为。要理解设计机制，应该重点着眼于实现一个或一组相关目标时涉及的对象和消息，它们在更大的系统中也用于完成其他目标。使对象和链共同工作以实现某种目标而进行的安排称为协作，在协作中实现行为的消息序列称为交互。协作是对“对象社会”的一种描述。它是大型复杂模型中的一个片断，在模型中协作是为了实现一个目标。

例如，商业销售表示对于一些对象的一种安排，这些对象在事务处理中有着特定的相互关系。在事务处理范围之外，这种相互关系就没有意义了。销售的参与者包括销售者、购买者和代理人。为了实现某个特殊的交互，如出售房屋，参与者们交换特定消息序列，如报价或者签订合同。

在协作中流动的消息流可以选用状态机来进行描述，状态机将规定合法的行为顺序。状态机中的事件代表了协作中各角色间的消息交换。

协作由角色组成。角色是指在协作中起作用的类元或者关联的一部分。实例化时，角色可以带有类元或者关联的实例。同一个类元或关联可以承担不同的角色，每个角色有不同的对象和链。例如，在商业事务处理中，两个公司中一方可能是销售者，另一方为购买者。seller和buyer是协作关系Sale中Company类所承当的角色。角色只在协作中才有意义，在协作之外无意义。实际上，在其他协作关系中，角色可能会对调。某个对象在一个协作实例中

China-pub.com 下载

可能是seller，而在另一个中就是buyer。同一个对象可能在不同的协作中承担多种角色，这与角色在关联限定的作用域不同。无论对象是否参与了关联，关联所描述的关系对于一个类在所有的语境中有全局性的意义。协作所描述的关系局限于特定的语境中，在此范围之外，关系没有意义。

实现。协作实现了某个操作或者用例。它描述了该操作或者用例的执行实现所处的语境，即，开始执行时，对于存在的对象和链的安排，以及在执行过程中生成的和销毁的实例。交互中的行为序列的说明可以用顺序图或者协作图表示。

协作也可完成类的实现。类的协作是类操作的协作的联合。对于同一个类、系统、子系统可以设计不同的协作，每个协作表示与实体的一个视图相关的属性、操作和相关对象的一个子集，如某个特定操作的实现。

模式。参数化的协作表示一种可以在不同的设计中重用的设计结构。通常基本类元是参数。这种参数化的协作符合模式结构。

见template。

通过给基础类元参数提供实际类元，可以得到实例化的设计模式。每个实例产生模型中特定的类元集上的一个协作。一个模式可与模型中的不同类元集多次绑定，从而避免为每次出现定义新的协作。例如，一个模型视图模式定义了模型元素间的常规关系，它可与代表模型视图对的多个模型视图类对进行绑定。每个实际模型视图类对代表了对模式一次绑定。这些类对中的某对可能是房屋和房屋里的图像，另一对可能是股票和表示股票当前价格的图形。

模式还包括使用指导以及对与其优缺点的明确描述。这些内容可以作为注释或写在单独的文本文件中。

协作的层次。协作可以用不同的粒度水平表示。粗粒度的协作可以通过进一步细化成为粒度更细的协作。可以通过扩展一个或者多个操作来实现由高层次协作到低层次协作的细化。

协作还可以通过下级协作来实现。每个下级协作实现整个功能中的一个部分，并有其独立的角色集。整体协作中的每个角色可以与嵌套部分中的一个或者多个角色绑定。如果一个外层角色被绑定到多个内层角色，则其隐含了与低层协作的连接。这是用例之间交互作用的唯一方式。一个设计思路是从内层向外，首先构造内层，限定角色，而后将它们结合生成外层，拓宽角色使其拥有多重责任。

运行时绑定。在运行时，对象和链与协作中的角色绑定。通常在不同的协作中，一个对象可以绑定到一个或者多个角色上。如果一个对象绑定到多重角色，那么它表示角色之间的一种“偶然”交互关系，即，这种交互不是由角色本身固有的，而只是它们在更广的语境中使用时的一种副作用。通常一个对象在组成大型协作的多个协作中承担角色。这种协作的重叠提供了一种隐含的协作间的控制和信息流动。

结构

角色。一个协作包括一系列的角色，每个角色是对一个或者多个基础类元（类元角色）

China-pub.com
下载

或者基础关联（关联角色）的引用。角色是协作中一个位置，它描述了类元或关联在协作中的一次使用。角色本身也是一个类元，绑定于协作实例角色的对象是角色的暂时实例。在协作的一个实例中，每个类元角色绑定于一个对象，每个关联角色绑定于一个链。在同一个协作的实例中，一个对象可以绑定到多个类元角色上，但是这是不常见的，而且可以通过适当的约束加以避免。在同一个协作的实例中，同一个类的对象可以充当不同的角色。每个对象具有与其角色相适应的关系。每个类元角色拥有在协作中用到的类元的属性的一个子集。其余的属性与本协作无关，但是可能在其他协作中有用。如果同一个基础类元含有多重角色，每个角色应有各自的名称以示区别。如果在协作中某个基础类元只用到一次，则角色可以没有名称，因为类元不会混淆。角色定义了协作的结构。

如果支持多重分类，则角色可以有多个基础类元。绑定于类元角色的对象是每一个基础类元的一个实例。

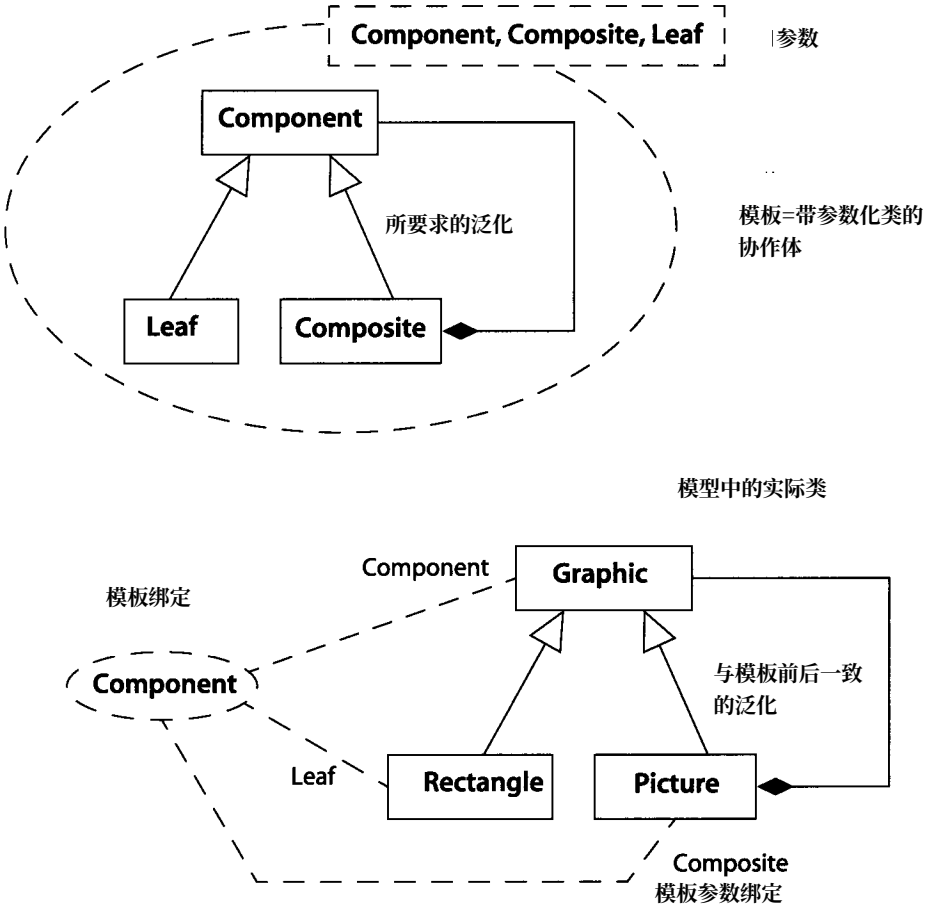


图13-50 模式：参数化的协作

China-pub.com 下载

泛化。协作还可以包含泛化关系和约束。这是对于参与协作的类元在其各自外层协作中可能拥有的关系的补充。只用当协作中的类元是参数形式时，泛化才是必要的。否则，它们的泛化结构只是作为其类元定义的一个部分出现，在协作中不可更改。在参数化的协作中（模式中）某些类元角色可能是参数。两个参数化类元之间的泛化关系说明任何绑定到角色的类元必须满足泛化关系（绑定到父角色的类元必须是绑定到子角色类元的祖先。它们之间不需要是父子关系。）

例如，摘自[Gamma-95]的Composite模式表示一个对象递归树，其中类 Component是树的一般元素，Composite是递归元素，Leaf是叶元素（如图 13-50）。Component是Leaf和Composite的父节点。Component是元素的聚集（递归）。Component，Composite，Leaf是模式中的参数。当模式被使用时，它们由具体的类替换。任何绑定到模式的具体类必须遵循Component与其子Composite，Leaf之间的祖先-后代关系。具体的例子可以是 Graphic，Picture，Rectangle；DirectoryEntry，Directory，File；或者任何递归类。如果一个绑定不满足泛化约束，则其形式错误。

约束。参数化的和非参数化的角色都可以有其约束。这些约束是对于绑定到角色的类元约束的补充。约束适用于任何协作实例。

消息。协作可以由一系列消息来描述其动态行为。带有消息的协作称为交互。可以有多重交互，每个交互说明同一个协作中的一个部分。例如，一个交互可以描述一个操作的实现，另一个交互说明另一个操作。消息带有其中的顺序信息，顺序信息的作用与定义由消息触发的状态机的作用相同。

资源。协作可能代表一个类、用例或者方法（协作是一个操作的实现，而不是操作的说明）。协作描述了源元素的行为。

表示法

在协作图中用类的符号（矩形）表示类元角色，关联路径（实线）表示关联角色，关联角色路径上带有消息符号。不带有消息的协作图表明交互作用发生的语境，而不表示交互。它可以用来表示单一操作的语境，甚至可以表示一个或一组类中所有操作的语境。如果关联路径上标有消息，则协作图表示一个交互。通常交互用来表示一个操作或者用例的实现。

协作图表示了协作中作为类元角色的对象所处的位置。类元角色与类元的区别在于它既有名称又有类名，语法为：*rolename:classname*。角色名和类名都可以省略，但是冒号必须保留。图中还表示了作为关联角色的对象之间的链，包括代表过程参数、局部变量以及自链接的暂时性链。多重关联角色如果与不同的类元角色连接，则可以有相同的关联名称。链上的箭头表明了箭头方向上的导航性（在对象之间的线上的箭头表明该链只有单方向导航性。邻接于线的箭头表明消息在链上按照所给方向流动。在单向链中，消息不能返回，所以消息流动必须与导航箭头相吻合）。

类元角色中的单个属性值通常不明确标出。如果消息必须传递给属性值，则应该将属性建模为使用关联的对象。

China-pub.com
下载

还可以使用其他图形标记来作为对关键字的补充或者替代。例如，每种生命期可以用不同的颜色表示。还可以用动画来表现元素的生成和销毁，以及不同时间内系统的状态。

操作的实现

表示操作实现的协作包括了目标对象角色的符号，以及目标对象实现操作时直接或者间接用到的其他对象的符号。关联角色上的消息表示一个交互中的控制流。每个消息表明了操作的方法中的一个步骤。

一个描述操作的协作包括了表示操作参数和执行中生成的局部变量的角色符号。在执行中生成的对象可以用{new}指明；在执行中销毁的对象用{destored}指明；在执行中生成，随后又被销毁的对象用{transient}指明。没有带关键字的对象是在操作开始前就存在，并且在操作结束后还存在的对象。

实现一种方法所用的内部消息被编号，编号从1开始。对于一个过程化的控制流，顺序的消息号码用点方法，按照嵌套层次排序。例如，第二层的步骤是消息2，它下一级的步骤就是2.1。对于并发对象间的异步消息交换，所有的消息编号都在同一层（即它们没有嵌套关系）。

见message，以全面了解包括顺序在内的消息语法。

完成协作图表示操作中所有对象和链的角色。如果某个对象没有表示，则假定它未被使用。但是假定协作图中的所有对象都被该操作使用是不安全的。

示例

在图13-51中，Controller对象调用redisplay操作。当操作被调用时，它已经与将要显示

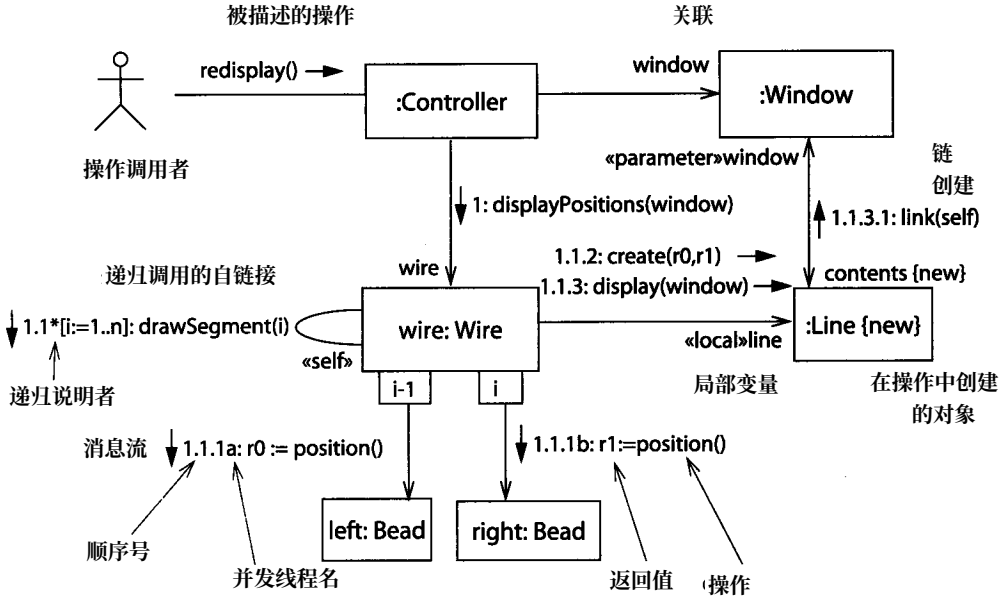


图13-51 带有消息流的协作图

China-pub.com 下载

图像的Window对象有了连接。它还与Wire对象有连接，该对象中的图形将被显示在窗口中。

redisplay操作的顶层实现只有一个步骤——在wire对象中调用diaplayPositions操作。这个操作的编号为1，因为它是最上层方法中的第一步。这个消息流被传递给后面将用到的Window对象的引用。

displayPositions操作调用同一个wire对象中的drawSegment操作，这个调用的编号为1.1，它通过self链传递。星形符号表示操作的重复调用，详细内容在方括号中说明。

每个drawSegment操作访问两个Bead对象，编号分别为i-1和i。虽然在此操作的语境中，Wire与Bead之间只有一个关联，但是需要两条指向两个 Wire对象的链。对象被标以left和right（这些是协作中的类元角色）。每个链上有一条消息传递。消息编号为1.1.1a和1.1.1b，这表明它们是操作1.1的步骤，最后的字母表示两条消息可以并行传递。在通常实现中，它们可能并不会并行执行，但是由于它们被声明为并行的，所以可以按照任何顺序执行。

当得到两个返回值（r0和r1）后，可以执行操作1.1之下的另一个步骤了。消息1.2是一个传递给Line对象的创建消息。实际上,消息到达Line类（至少在原理上是这样），该消息创建了一个与发送者相链接的新的Line对象。新的对象带有标签{new}，表明它是在操作的执行过程中被创建的，并且将在操作结束后继续存在。新建的链带有标签《ocal》，表明它不是一个关联，而是过程中的局部变量。局部变量是暂时性的，当过程执行完毕后将消失。因此不需要为它加上标签{transient}。

步骤1.3通过新创建的链向新创建的 Line对象发送一条display消息。指向window对象的指针作为参量一同传递，从而使得它可以被 Line对象作为《parameter》链访问。注意，与Line对象相链接的window对象就是与初始的Controller对象相关联的window对象，这一点对于操作很重要，并且在图中已标出。在最后一步 1.3.1中，要求window对象创建与Line对象的链，该链是一个关联，所以它有角色名contents，并且标以{new}。

想象擦去所有临时连接，就得到系统的最终状态。在 Controller和wire之间，以及wire和它的部分Bead之间，Controller与window之间，window与其contents之间都有链。但是，一旦操作完成，Line就不能访问包含它的window了。在这个方向上的链是暂时的，当操作完成后就消失了。同样，wire对象也不能再访问用于显示它的Line对象了。

实例级协作

与许多模型元素类似，协作可以用描述符或者实例表示。描述符级协作表示对象之间可能的关系。协作可以被多次实例化从而生成协作实例，协作实例表示特定对象之间的关系。

应该用哪种形式来为某个情景建模呢？如果图表示的是可能事件，则必须使用描述符级协作图。对象图没有偶然性。它们没有条件判断或者循环，这些是普遍性描述中的内容。实例没有取值范围，或者许多可能的控制路径，它只有一个值和唯一的历史。

如果图表示的是属性或者参量具体的值，如果图表示的是可变大小多重性的对象或者链的具体数目，或如果图表示的是对于执行中的分支或者循环的特定选择，这样的图必须是实例级协作图。

China-pub.com
下载

在许多情况下，可以使用任何一种视图，如当计算中没有分支时。任何执行都是其原型，描述符形式和实例形式间没有太多区别。

66. collaboration diagram(协作图)

协作图是表示角色间交互的视图，即，协作中的实例及其链。与顺序图不同，协作图明确地表示了角色之间的关系。另一方面，协作图也不将时间作为单独的维来表示，所以必须使用顺序号来判断消息的顺序以及并行线程。顺序图和协作图表达的是类似的信息，但使用不同的方法表示。

见collaboration，pattern，sequence diagram。

67. collaboration role(协作角色)

协作中的对象或者链，它说明协作实例中可能出现的对象和链的种类。

见association role，classifier role，collaboration。

语义

协作角色描述对象或者链。但是它不表示单独的对象或链，而是当协作实例化时，对象或链被替换的位置。协作可以是类元角色，也可以是关联角色。一个类元角色有一个或多个基础类元，可以实例化为对象。关联角色可有一个基础关联，且可实例化为链，该链可以是关联的实例，或是它的后代的实例。在许多情况下，关联只在协作中定义，即，它只出现在承担角色的对象上，离开协作后就没有意义了。在这种情况下，可以省略基础关联。关联在协作中被隐含地定义，在别的地方不可用。

表示法

协作角色可以是类元角色或者关联角色。

类元角色。类元角色是一个类元，用类的矩形符号表示。通常只表示名称分栏。名称分栏包含如下字符串：

classifierRoleName :*BaseClassifierName*$_{list}$

如果必要，基础类元名称可以包括外层包的详细路径名（在不会产生混淆的情况下，工具可以允许简化的路径名）。包名先于类名，并用双冒号隔开。例如：

display_window:WindiwingSystem::GraphicWindows::Window

类元角色的构造型可以用名称字符串上方的文本表示，也可以用右上角的图标表示。类元角色的构造型必须与其基础类元的构造型匹配。

China-pub.com 下载

代表一系列对象的类元角色包含位于类的矩形框右上角的多重性指示符（如 *），它说明在协作的一个实例中，可绑定到该角色的对象数目。如果该指示符被省略，则该值取 1。

类元角色的名称可以省略。在这种情况下，类名之中要保留分号，代表类的匿名对象。

如果支持多重类元，则角色可能有多个类元。对象就是它们之中每一个的实例。

类元角色的类可以被隐藏（包括分号在内）。

对于交互中创建的对象或者链，在它们的角色中有关键字 new作为约束。对于在交互中销毁的对象或者链，其角色有关键字 destroyed作为约束。即使对于某个没有名称的元素，也可以使用关键字。两个关键字可以同时使用，但是关键字 transient可以用于代替new destroyed。

关联角色。关联角色是一个关联，用两个类元角色符号间的路径表示。路径可以赋以以下形式的名称：

associationRoleName : BaseAssociationName

如果基础关联的名称被省略，则没有基础关联。基础关联的角色名及其他修饰成分将在路径上表示。

如果关联角色路径一端连接一个多重性大于 1的类角色，则在关联的末端加上表示多重性指示符以示强调。

图13-51 是一个关联角色的例子。

68. combination(组合)

将类元的两部分描述联系起来，从而组成为某元素的完整描述符，这两部分之间的相互关系称为组合。

见extend，include。

语义

面向对象的强大功能之一是可以以渐增的方式将模型元素的描述组合起来。继承机制将与泛化相关的类元组合起来，生成对于类的有效的完整描述。

组合描述符的其他两种方法是使用扩展和包含关系（泛化本来也可以列在该条目下，但是由于其特殊的重要性，所以将它作为独立的基本关系）。

表示法

组合用带有构造型关键字的虚线箭头表示。

见extend，include。

讨论

其他种类的组合也是可能的。某些程序设计语言，如 CLOS，实现了若干种不同的功能强

China-pub.com
下载

大的方法组成。

69. comment(注释)

附在单个元素或者元素集上的注释。注释没有直接的语义，但是它可以表示对于建模者或工具有意义的语义信息或者其他信息，如约束或方法体。

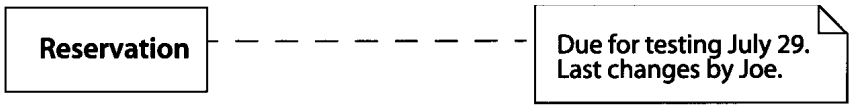
见note。

语义

注释包含文本字符串，如果建模工具允许，也可以带有嵌入的文件。注释可以附在模型元素、表示元素或者元素集上。它提供了关于任意信息的文本说明，但是没有语义作用。注释给建模者提供信息，可以用来搜索模型。

表示法

注释用注解的符号表示，即右上角向下折叠的矩形（“狗耳朵”），并且用虚线与被注释的一个或几个元素相连（如图 13-52）。建模工具可以使用其他形式来显示或者浏览注释信息，如弹出式备注、特殊字体等等。



注释的目标　　注释

图13-52　注释

标准元素

requirement，responsibility。

70. communication association(通信关联)

描述相互连接的元素实例之间通信关系的关联。在部署视图中，它是包含通信关系的节点之间的关联。在用例模型中，它是用例与通信关联的参与者之间的关联。

见actor，use case。

China-pub.com 下载

71. compartment(分栏)

封闭形状符号的图形化分，如将表示类的矩形水平地分成小矩形。每个分栏表示它所代表的元素的特性。有三种分栏：固定式、列表式、区域式。

见class、classifier。

表示法

固定式分栏用固定的图形形式以及文本部分表示固定的特性集。图形形式由元素种类决定，例如类名分栏，其中包含构造型符号和 /或名称、类名、表示类的不同的特性的字符串。根据元素的不同，其中某些内容可能被隐藏。

列表式分栏包含元素成分的编码表。一个例子是属性表。编码由成分的类型决定。元素的顺序可以是它们在模型中的自然顺序，或者按照一个或多个特性排序（在这种情况下，自然顺序将不可见）。例如，属性的列表可以先按照可见性，再按照名称排序。根据模型元素特性的不同，表中的条目可以被表示或者被隐藏。例如，属性分栏可能只表示公共属性。可以将构造型和关键字编入列表条目，从而将它们运用于每个独立成分。构造型和关键字可单独列入条目，从而适用于所有顺序成分。它们可以影响所有顺序成分，直至表结束或另一个这样的声明起作用。将《 constructor》在操作表中单独列为一行，可以将随后的操作构造为构造符，但随后的字符《 query》将撤销前一个声明，并用《 query》构造型代替它。

区域式分栏是一块包含表示元素子结构的子图的区域，通常隐含有递归，如嵌套的状态图。子图的性质是模型元素特有的。在一个符号内同时包含区域和文本分栏是合法的，但是将显得混乱。区域通常用于递归元素，而文本用于没有递归子结构的叶元素。

一个类有三个预先定义的分栏：名称，一个固定式分栏；属性，一个列表式分栏；操作，也是列表式分栏。模型设计者可以在矩形中增加其他的分栏，并在分栏的上方用不同的字体标出其名称。

图形语法取决于元素和分栏的种类。图 13-53表示了信号分栏，图13-54表示职责分栏。

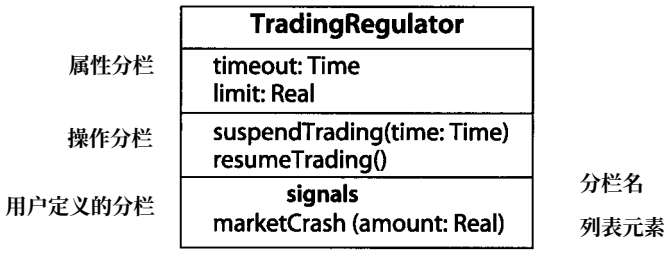


图13-53 类中命名的分栏

China-pub.com
下载

72. compile time(编译时间)

是指在软件模型编译时出现情况。

见modeling time，run time

73. completion transition(完成转换)

是指没有明确的触发器事件，而是由源状态中的活动完成来触发的转换。

见activity，transition，trigger。

语义

完成转换是指没有明确触发器事件的转换。转换的源状态完成了任何活动（包括嵌套状态）后，转换隐式触发。在组成状态中，活动的完成是由到达终态表示的。如果一个状态没有内部活动或者嵌套状态，则一旦入口和出口活动完成，完成转换将被立即触发。

如果一个状态有嵌套状态或活动，但是没有带有触发器事件的输出转换，则完成转换不一定能发生。当封装的状态还在等待完成其活动时，某个事件可能触发了内部状态的转换，这样，可能会跳过完成转换。

完成转换含有监护条件和一个活动。通常不希望出现只含有一个监护条件的完成转换，因为如果监护条件不满足，完成转换将永远不能发生（因为隐含的触发器只出现一次）。有时如果触发转换将对象带出死状态，描述某些错误也可能有用。更多情况下，一系列带有监护条件的完成转换带有涵盖所有可能的条件，使得当状态结束后，其中的某一个完成转换将立即激发。

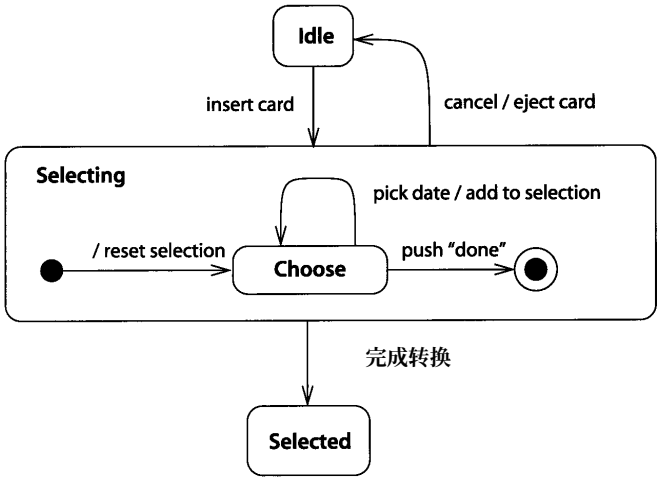


图13-54 完成转换

China-pub.com 下载

完成转换还用于将初始状态和历史状态与其后继状态相连接，因为这些伪状态在完成其活动后将不会保持活动性。

示例

图13-54表示的是订票程序的状态机模型片断。在顾客选择日期时， Selecting状态将一直处于活动状态，当顾客按下“ done ”键后，Selecting状态达到其最终状态。这触发了完成转换，转入Selected状态。

74. complex transition(复杂转换)

复杂转换是指有多于一个源状态和 /或多于一个目标状态的转换。它表示对引起并发数目变化的事件的响应。根据源状态和目标状态的数目，它可以是控制的同步、控制的分叉，或者两者兼而有之。

见branch，composite state，fork，join，merge。

语义

从上层看，系统经历了一系列状态。但是从整体的观点来看，对于具有分布性和并行性的大型系统而言，认为系统处于一种状态不足以说明问题。系统在同一时刻可以有多种状态。这些活动状态集称为活动状态配置。如果某个嵌套状态为活动的，所有包含此状态的状态均为活动的。如果对象允许并发，则可能有多个并发子状态为活动的。

在许多情况下，系统的活动可以被建模为相互独立或者交互不多的控制线程集。每个转换最多只能影响活动状态配置中的一小部分状态。当转换发生时，未受影响的活动状态继续保持为活动的。某一时刻的线程可以被视为活动状态配置中的一个子状态，每个线程对应一个子状态。每个子状态对事件有独立的反应。如果活动事件的数目是固定的，则该状态模型是一些相互交互的固定状态机的集合。但是，通常状态的数目（以及控制线程的数目）是随时间而变化的。一个状态可以转换到两个或者多个并发状态（控制分支），两个或者更多的并发状态可以转换到一个状态（控制的结合）。并发状态的数目以及变化由系统的状态机控制。

复杂转换是指源于或者指向一系列并发子状态的转换。复杂转换有多于一个的源状态和 /或目标状态。如果该转换有多个源状态，则它表示控制的结合；如果一个转换有多个目标状态，则它表示控制的分叉；如果它同时有多个源状态和目标状态，则它表示并行线程的同步。

如果一个复杂转换有多个源状态。只有当这些源状态都是活动的，转换才有可能被触发。而这些状态被激活的顺序是无关紧要的。如果所有源状态都是活动的，而事件发生，转换将被触发，如果监护条件得到满足，转换将激发。即使有多个源状态，每个转换也只被一个事件激发。UML不支持多事件，每个事件触发一个独立的转换，随后由控制结合点将它们与结果状态相连。

China-pub.com
下载

如果一个具有多个源状态的复杂转换没有触发事件（即为完成转换），则当所有源状态被激活后，转换将被触发，如果监护条件满足，转换将激发。

当复杂转换激发时，所有源状态以及同一个组成中的状态都变为不活动的，所有目标状态及同一个组成中的状态变为活动的。

在更复杂的情况下，可以进一步扩展监护条件，使得在某些子状态活动时才允许转换激发。

示例

图13-55是一个典型的带有复杂转换出入口的并发组成状态。图 13-56是该状态机的一个典型执行过程记录（活动状态用阴影表示）。记录表明不同时刻活动状态数目的不同。

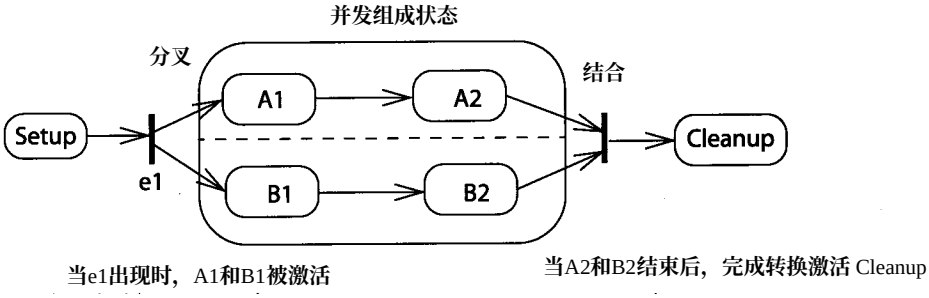


图13-55 分支与结合

并发状态

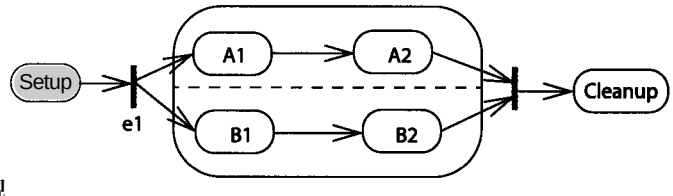
除非非常仔细地设计状态机结构，否则一系列复杂转换可能导致不一致性，包括死锁、状态多重占有等问题。Petri网理论充分地研究了这些问题，常见的解决办法是在状态机中引入良好结构的规则来避免不一致性，这就是状态机的“结构化编程”规则。有许多不同的实现办法，各有利弊。UML所采用的规则要求用与-或树将状态机精化。其优点在于：一个定义良好的嵌套结构容易建立、维护、理解；缺点是使一些有意义的配置被禁止了。总而言之，这类似于为得到结构化的程序而放弃使用 goto语句的做法。

一个复杂状态可被分解为几个相互排斥的子状态（“或”分解），或者被分解为几个并发子状态（“与”分解）。结构是递归的。通常“与”层可以替换“或”层。“与”层表示并发分解——所有子状态并发活动；“或”层表示顺序分解——每次只有一个子状态是活动的。可以从根节点开始递归扩展树的节点得到合法的并发状态序列，用所有的子状态代替“与”状态，用某个子状态代替“或”状态。这与状态图的嵌套结构相对应。

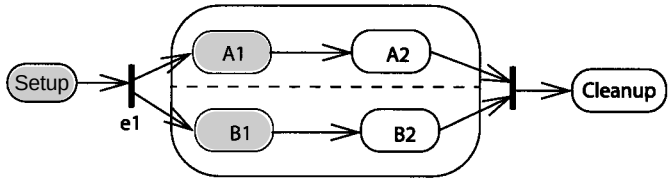
示例

图13-57表示与图13-55的状态机对应的状态与-或树。一组典型的并发活动状态用阴影表

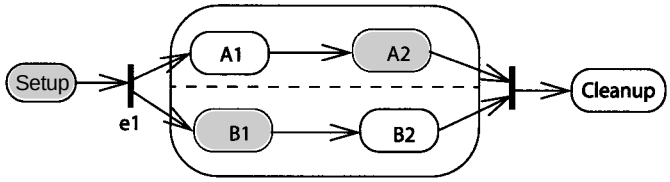
China-pub.com 下载



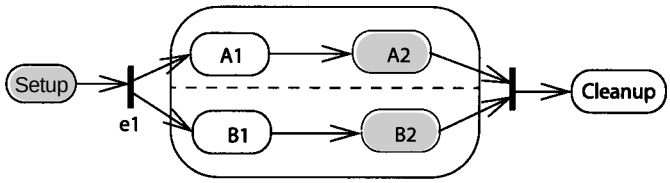
e1 出现



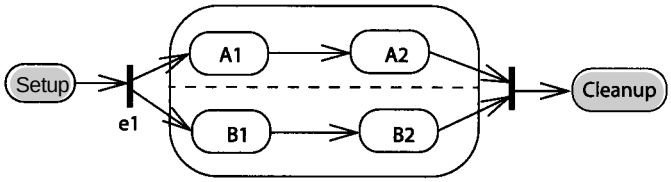
A1 完成



B1 完成



A2 和B2完成



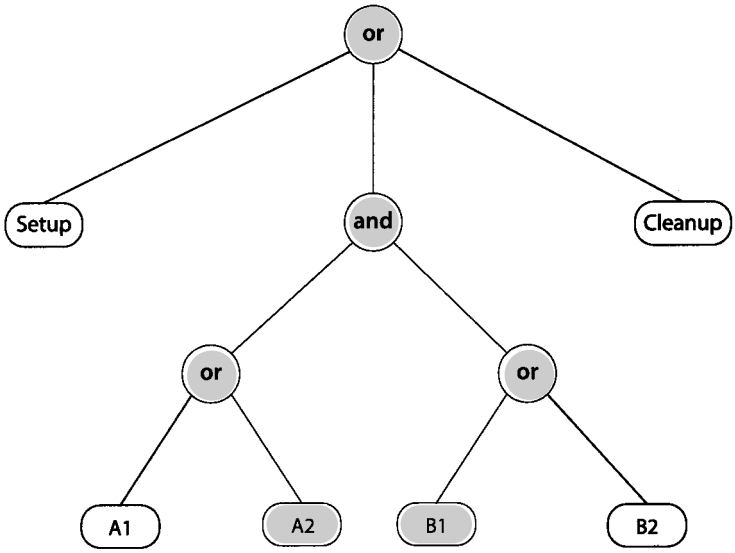
(活动状态用阴影表示)

图13-56 并发状态机的活动状态记录

示，这与图13-56 的步骤3对应。

如果转换进入一个并发区域，则它进入所有子状态。如果转换进入顺序区域，则它只进入了一个子状态。顺序区域中的活动状态可以变化，而在并发区域中，只要区域是活动的，则所有子状态都是活动的。但是，通常每个并发子状态又被进一步分解为顺序区域。

China-pub.com
下载



（典型活动状态集用阴影表示）

图13-57 嵌套状态的与－或树

因此，简单转换（具有一个源状态和一个目标状态的转换）必须连接两个在同一顺序区域内的状态，或者两个仅被“或”层分隔的状态。复杂转换必须将一个并发区域中的所有子状态与并发区域外的一个状态相连（我们忽略了更复杂的情况，但它们也必须遵循上述原则）。换而言之，进入并发区域的转换进入所有子状态，而离开并发区域的转换也离开所有子状态。

可以使用一种简化的表示法：如果复杂转换进入并发区域但省略了一个或几个子区域，则隐含表示有转向子区域的初始状态的转换。如果某个子区域没有初始状态，则模型结构有错误。如果有复杂转换离开并发区域，则隐含了从各个被省略的子区域发出的转换。一旦转换发生，则子区域中的所有活动将终止，即强行退出。转换可连接到封装的并发区域本身，它隐含表示指向各个并发子区域的初始状态的转换——一种常用的建模情况。同样，源自封装并发区域的转换暗示强行退出各子区域（如果有触发事件）或者等待子区域结束工作（如果无触发事件）。

复杂转换的规则保证了无意义的状态组合不会并发处于活动状态。一系列并发子状态是封装的组成状态的一部分，它们要么同时活动，要么都不活动。

条件线程

在活动图中，带分叉的一段可以有监护条件，被称为条件线程。当转换激发时，只有当监护条件满足后，线程才会被初始化。没有监护条件的线程则在转换激发后立即初始化。活动图中的并发必须是合理嵌套的——每个分支点之后都要有相应的结合点。如果一个条件线

China-pub.com 下载

程因为条件不满足而失败，则结合点之前相应的活动被视为已经完成，即，转换不再等待这个条件线程的控制流。如果分叉中的所有线程激发失败，则控制将立即从匹配的结合点重新开始。

条件线程等同于一个带分支和合并的图，该图处于活动图中的条件部分。

示例

图13-58 是带有两个条件线程的活动图。它表示一个航班的登机过程。在开始时，乘客必须先出示机票。随后有3个并发线程，其中两个为条件线程。座位安排是必须始终进行的，而只有当乘客有行李时才要进行行李检查，只有国际航班才有护照检查过程。当 3个线程全部完成后，控制结合，指向单一的线程，将机票和登机卡返还乘客。如果某个条件线程没能开始，则会被随后的控制结合点视为已经完成。

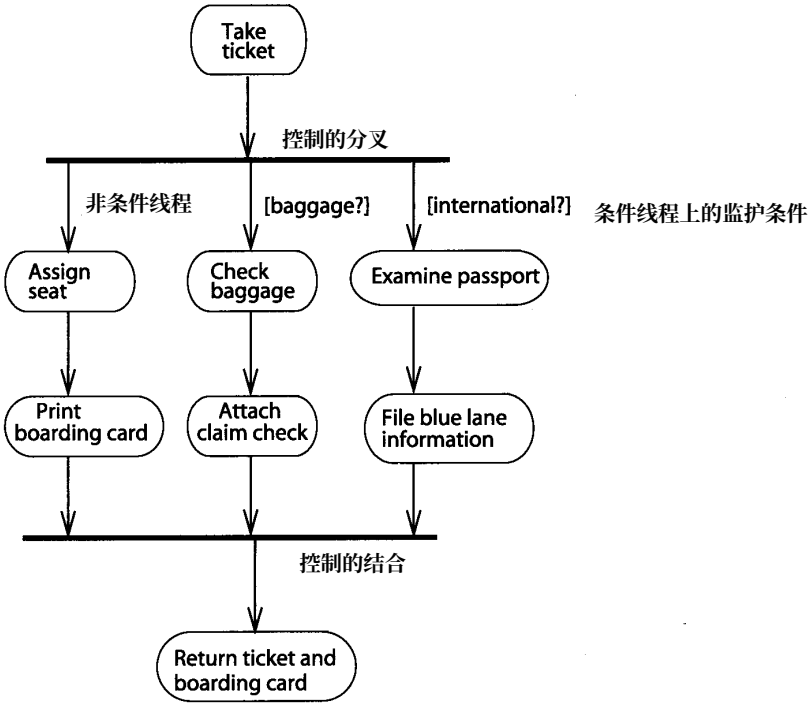


图13-58 条件线程

表示法

复杂转换用短粗线表示（表示同步和/或分支的同步条）。可能有一个或多个实线转换箭头从状态指向粗线（该状态为源状态）；可能有一个或多个箭头从粗线指向状态（该状态为目

China-pub.com
下载

标状态）。在粗线边上注明转换标签，用来描述触发器事件、监护条件、活动。箭头没有各自的转换标识串，它们仅是整个转换中的一部分。

示例

图13-59 比图13-55的状态机增加了退出转换。它也表示了从 Setup状态到各个子区域的初始状态的隐式分叉，以及从各子区域的终态到 Cleanup状态的隐式结合。

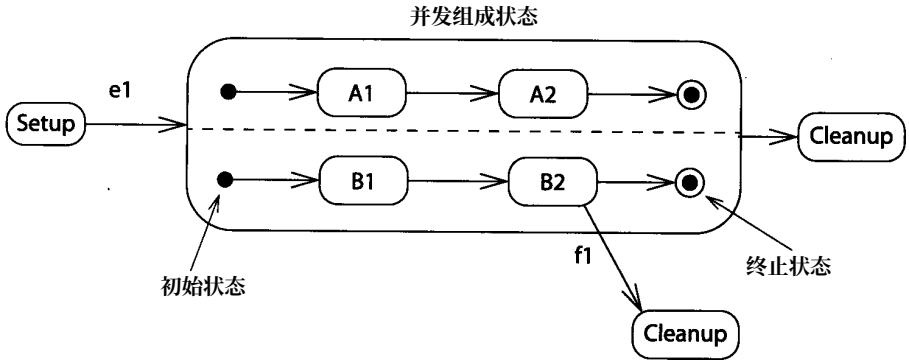


图13-59 复杂转换（分支，结合）

如果当B2状态活动时发生事件 f1，则指向Cleanup事件的转换将会发生。该转换是隐式的结合点，它既终结状态B2也终结状态A2。

75. component(构件)

构件是指系统中可替换的物理部分，该系统封装了实现并提供了一组接口的实现。

语义

构件是指系统中的一个物理实现片断，包括软件代码（源代码、二进制代码、可执行代码）或者相应成分，如脚本或命令文件。有些构件带有身份标识，并有物理实体，包括运行时的对象、文档、数据库等。构件只在实现域中存在 —— 它们是计算机的物理组成部分，可以与其他构件相连，由类似构件替换，移动，获得等等。模型可以表示构件间的依赖关系。例如，编译器和运行时的依赖关系，或者人类组织中的信息依赖关系。构件实例可用于表示运行时存在的实现单元，包括它们在节点实例中的定位。

构件有两个特征。它们封装了实现系统功能的代码和某些构成系统状态的对象实例。因为它们的实例含有身份和状态，我们称后者为有身份的构件。

代码特征。构件包含了实现类或者其他元素的代码（代码广泛采用和包括脚本、超文本结构以及其他可执行描述形式）。源代码构件是指实现类的源代码包。某些语言（如 C++）将

China-pub.com 下载

声明文件与方法文件分开，但它们都是构件。二进制代码构件是编译后代码的包，二进制代码库是一个构件。可执行构件含有可执行代码。每种构件都含有实现某种逻辑类和接口的实现类的代码。实现关系将构件与其实现类所实现的逻辑类和接口联系在一起。

构件的接口说明了它支持的功能。接口中的每个操作最终都必须映射到构件支持的实现元素上。

系统实现的静态可执行结构可由内部互连的构件集来表示。构件之间的依赖关系说明一个构件的实现元素需要另一个构件的实现元素提供服务。这种情况下，要求提供服务者在构件之间是公共可见的。构件可以有私有元素，但是这些元素不能直接为其他构件提供服务。

构件可以包含其他构件。一个被包含构件仅是其包含者的另一实现元素。

构件实例是节点实例上的构件的一个实例。源代码和二进制代码构件的实例可以驻留在某个特定节点的实例上，但最有用的还是可执行构件实例。如果一个可执行代码构件实例定位于一个节点实例，则构件支持的实现类的对象在定位到节点实例上后可执行操作。否则，位于节点实例上的对象不能执行操作，必须被移动或者复制到其他节点实例后才能执行操作。

如果构件没有身份，则它的所有实例都是相同的，其中哪个支持对象的操作都可以，它们的行为是相同的。因为没有身份，构件实例自身没有值或者状态。

身份特征。一个有身份的构件拥有身份和状态。它带有定位于其上的物理对象（所以，在节点实例上包含构件实例）。它可以有属性、与构件其他对象的组成关系、与其他构件的关联。从这一角度来看，它是一个类。但是，它的所有状态必须映射到它自身的实例上，这是构件与普通类的区别。通常用一个实现类来代表整个构件。这被称为主导类，并被看作与构件是等同的，但它们实际上是不同的东西。

当对象要求有身份的构件提供服务时，必须选择一个特定的构件实例，通常选用与该构件有关联的构件所拥有的对象之一。因为每个有身份构件实例都有状态，因此要求不同的实例将产生不同的结果。

示例

例如，一个拼写检查器是一个构件。如果它有固定的字典，则可被视为无身份标识的构件。它的所有实例产生相同的结果，而且对过去的要求没有记忆功能。如果字典可以被更新，则它必须被建模成有身份的构件。对应拼写检查器构件的不同实例，字典有不同的版本。要求服务时必须指明所选用的字典。通常，目标构件在特定语境中隐含指定，但这是设计中必须考虑到的一个选择。

结构

一个构件支持一系列实现元素，如实现类，即构件提供元素所需代码。一个实现元素可能被多个构件支持。

构件可以有操作和接口，这些都由其实现元素实现。

China-pub.com
下载

一个有身份的构件是物理实体的物理包容器，如运行时的对象和数据库。为了给其内部元素提供句柄，它可以有属性或者向外的关联，这些必须由其实现元素实现。一个有身份的构件可以指定一个支持所有公共属性和操作的主导类，但是这样的类仅是该构件的一个实现元素。

表示法

构件用矩形表示，其一侧有突出的两个小矩形。构件名称标在矩形中（如图 13-60）。

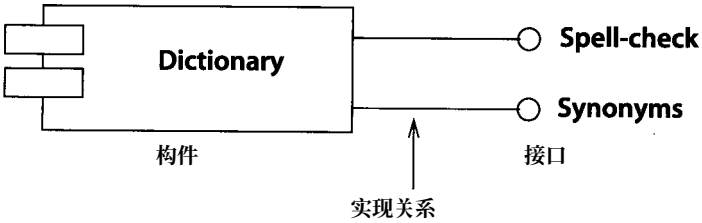


图13-60 构件

构件实例有各自的名称，用冒号与构件名称隔开。名称有下划线，以区别于构件类型名称（如图13-61）。构件实例的符号可以画在节点符号内，表示构件实例位于节点实例上（如图13-62）。如果构件没有身份，实例名称将被省略。属于有身份的构件实例的对象可以画在其内。带有属性或含有对象的构件自动成为有身份的构件。

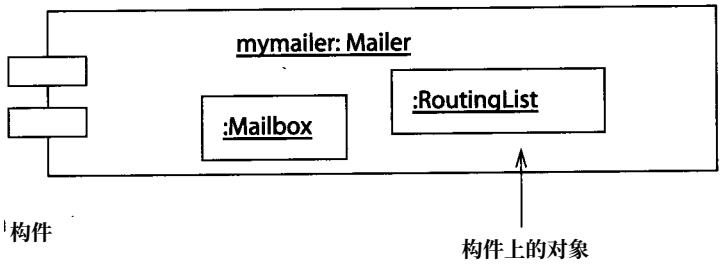


图13-61 带有对象的有身份的构件实例

主导类包含构件的接口。它可以用右上角有作为构造型图标的构件符号的类图形表示。在这种情况下，构件和它的主导类共享相同的接口，而且主导类可以通过组合链访问构件中的任何一个对象。图13-63 是一个例子。

外部对象可以使用的操作和接口可直接在类的符号中表示。这些是类似类的行为。子系统的内容在单独的图中表示。如果不需要表示子系统的实例化特征，可以使用普通包的表示法。

从一个构件到另一个构件或者模型元素的依赖关系用带有箭头的虚线表示，箭头指向提供服务的元素（如图13-64）。如果一个构件是某个接口的实现，可以用线将一个圆连到构件符号来简单地表示。实现一个接口意味着构件的实现元素支持接口的所有操作。如果一个构件使用

China-pub.com 下载

了其他元素的接口，依赖关系可以用带箭头的虚线表示，箭头指向接口符号。使用一个接口说明构件的实现元素只需要服务者提供接口所列出的操作。（但客户还可以依赖于其他接口）。

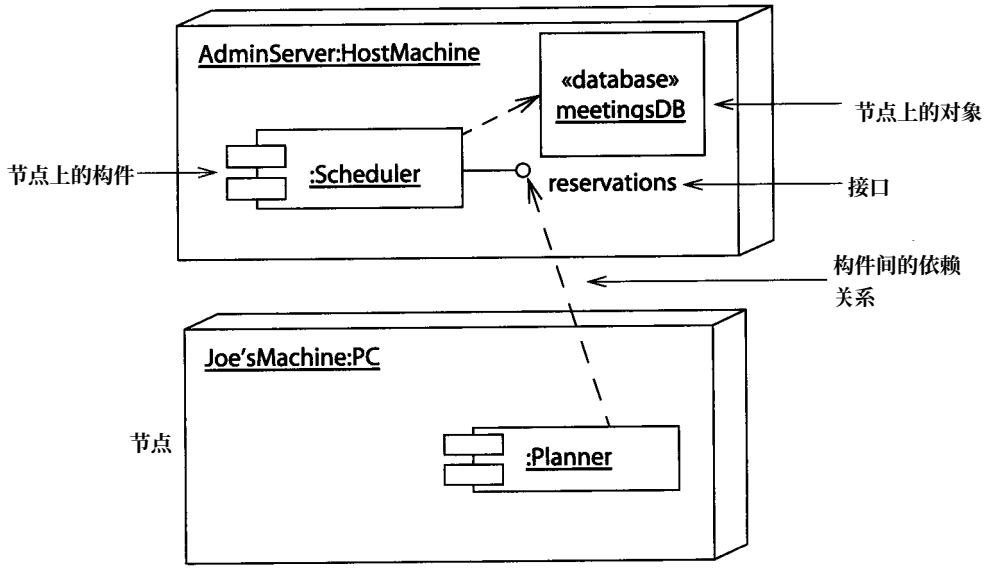


图13-62 节点上的构件实例

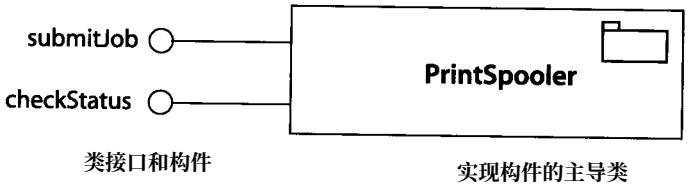


图13-63 构件的主导类

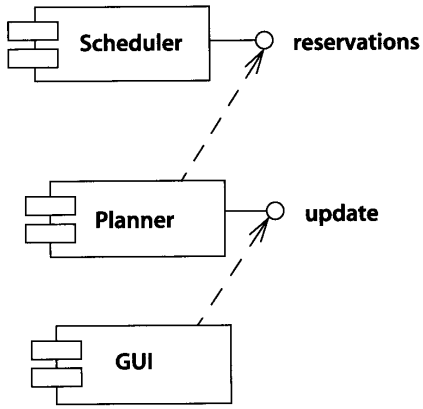


图13-64 构件的依赖关系

China-pub.com
下载

讨论

下列扩展定义说明了设立构件的意图，以及确定系统的一个部分是否被作为有意义构件的考虑：

■ 构件很重要，它在功能和概念上都比一个类或者一行代码强。构件通常拥有类的一个协作的结构和行为。

■ 一个构件基本独立于其他构件，但是很少独立存在。一个给定构件与其他构件协作完成某种功能，为了完成这项功能，构件假设了一个结构化的语境。

■ 构件是系统中可以替换的部分。它的可替换性使得可以用满足相同接口的其他构件进行替换。替换或者插入构件以形成一个运行系统的机制一般对构件的使用者是透明的，对象模型不需要多少转换就可使用或利用某些工具自动实现该机制。

■ 构件完成明确的功能，在逻辑上和物理上有粘聚性，因此它表示一个更大系统中一段有意义的结构和/或行为块。

■ 构件存在于具有良好定义结构的语境中。它是系统设计和组建的基石。这种定义是递归的，在某种抽象的层次上系统仅是更高抽象层次上的构件。

■ 构件不会单独存在，每一个构件都预示了它将处于的结构和 /或技术的语境。

■ 构件符合一系列接口。符合一个接口的构件满足接口指定的约定，且在接口适用的所有语境中都可被替换。

标准元素

document，executable，file，library，location，table。

76. component diagram(构件图)

表示构件类型的组织以及依赖关系的图。

语义

构件图表明了软件构件之间的依赖关系，包括源代码构件、二进制代码构件和可执行代码构件（如图13-64）。软件模块可以用一个构件来表示。有些构件存在于编译时，有些存在于链接时，有些存在于执行时，有些在多种场合存在。一个编译时构件只在编译时有意义。在这种情况下运行时构件是可执行的程序。

构件图只有描述符形式，没有实例形式。要表示构件实例，应使用部署图。

表示法

构件图表示了构件类元，以及其中定义的类和构件间的关系。构件类元还可以嵌套在其

China-pub.com 下载

他构件类元之中，从而表示定义关系。

构件中定义的类在构件中表示。虽然对各种大小的系统而言，提供构件中定义的类的列表可能比表示符号更方便。

可以用包含构件类元和节点类元的图来表示编译依赖关系，该关系用带箭头的虚线表示，箭头从用户构件指向它所依赖的服务构件。依赖关系的类型用语言说明，可作为依赖关系的构造型显示。

图还可以用于表示构件之间的接口和调用关系。虚线箭头从一个构件指向其他构件上的接口。

见component，可以得到构件图的实例。

77. composite aggregation(组成聚合)

见composition。

78. composite class(组成类)

通过组成关系与一个或者多个类相关联的类。

见composition。

79. composite object(组成对象)

组成对象表示一个由紧密结合的部分构成的高层次对象，它是组成类的一个实例，隐含了类与其成分之间的组成聚集。组成对象与协作类似（但是更简单，而且更严格）。然而，它由静态模型中的组成定义，而不是由协作中的与语境相关的关系定义。

见composition。

语义

组成对象与它所有的组成部分之间有组成关系。这意味着它负责这些部分的创建和销毁，且没有其他对象有类似的责任。换而言之，没有这些组成部分的垃圾收集点，组成对象可以，也必须在它死亡时销毁这些组成部分，或者把责任转交给其他对象。

组成关系通常由与组成对象自身（通常为一个记录）相同的数据结构内的物理限制实现。物理限制保证了组成部分的生命周期与组成对象的生命期匹配。

表示法

对象和链的网络可嵌套在对象符号的图形分栏中。图形分栏是位于属性分栏（可以省略）

China-pub.com
下载

之下的附加分栏。图形区域内包含的对象和链是组成对象的组成部分。但是，路径超出组成对象范围的链不是对象的组成部分，而是分离对象之间的链。

示例

图13-65是一个组成对象，名为桌面窗口，由不同部分组成。它包括 ScrollBar类的多个实例，每个实例在组成对象中有自己的名称和角色。例如， horizontalBar和verticalBar都是滚动条，但是它们的行为不同。在这一点上，它们类似于协作角色。

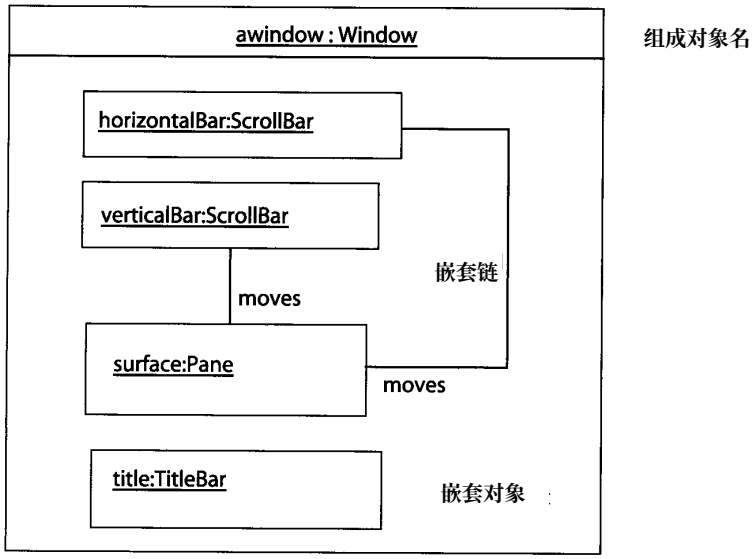


图13-65 组成对象

80. composite state(组成状态)

包含并行（正交）或顺序（互斥）子状态的状态。

见complex transition，simple state，state。

语义

组成状态可以使用“与”关系分解为并行子状态，或者通过“或”关系分解为互相排斥的互斥子状态。状态精化只能使用两种方法之一。其子状态还可以用两种方法之一进一步进行分解。如果一个顺序组成状态是活动的，则只有一个子状态是活动的。如果一个并发组成状态是活动的，则与它正交的所有子状态都是活动的。分解结果为与 -或树。每个状态机都有一个顶层状态，它是组成状态。

China-pub.com 下载

系统在同一时刻可以包含多个状态。活动状态集称为活动状态配置。如果一个嵌套状态是活动的，则所有包含它的组成状态都是活动的。如果对象允许并发，则可以有多个并发子状态同时为活动的。

见complex transition，可更详细地了解并发执行。图13-51显示了一棵与-或树。

新创建的对象始于初始状态，这是最外层的组成状态中必须包含的状态。创建对象的事件可以用来触发某个离开初始状态的转换。关于创建事件的讨论也适用于初始转换。

转入外层终态的对象将被销毁而停止存在。

结构

一个组成状态包括一系列子状态。组成状态可以是并发或者顺序的。

一个顺序组成状态最多可以有一个初始状态和一个终态，同时也最多可以有一个浅历史状态和一个深历史状态。

一个并发组成状态可能没有初始状态、终态或者历史状态。嵌套在它们里的任何顺序组成状态可包含这些伪状态。

表示法

组成状态是包含有从属细节的状态。它带有名称分栏、内部转换分栏和图形分栏。图形分栏中有用于表示细节的嵌套图。所有的分栏都是可选的。为了方便起见，文本分栏（名称分栏和内部转换分栏）可以缩略为图形分栏内的制表符，而无需水平延伸它。

将图形分栏用虚线分成子区域，表示将并发组成状态分为并发子状态。每个子区域代表一个并列子状态，它的名称是可选的，但必须包括带有互斥的子状态的嵌套状态图。用实线将整个状态的文本分栏与并发子状态分栏分开。

在图形分栏中，用嵌套的状态表图表示将状态扩展为互斥的子状态。

初始状态用小实心圆表示。在顶层状态机中，源自初始状态的转换上可能标有创建对象的事件。否则转换必须是不带标签的。如果没有标签，则它表示所有到封装状态的转换。初始转换可以有一个动作。初始状态是一个符号设备，对象可以不处于这种状态中，但必须转换到实际的状态中。

终态用外面套了圆环的实心圆表示。它表示封装状态中的活动完成，且触发标有隐含的完成事件活动的封装状态上转换（通常为无标签转换）。

示例

图13-66 表示包含两个互斥子状态的顺序组成状态、一个初始状态和一个终态。当组成状态为活动时，子状态Start（初始状态的目标状态）首先变为活动的。

图13-67表示带有三个正交状态的并发组成状态。每个并发子状态又进一步分为顺序子状态。当组成状态Incomplete成为活动状态时，初始状态的目标状态成为活动的。当三个子状态

China-pub.com
下载

都达到终态后，外部组成状态 Incomplete的完成转换被触发，Passed成为活动状态。如果当Incomplete为活动状态时发生fail事件，则所有的三个并发子状态结束，Failed成为活动状态。

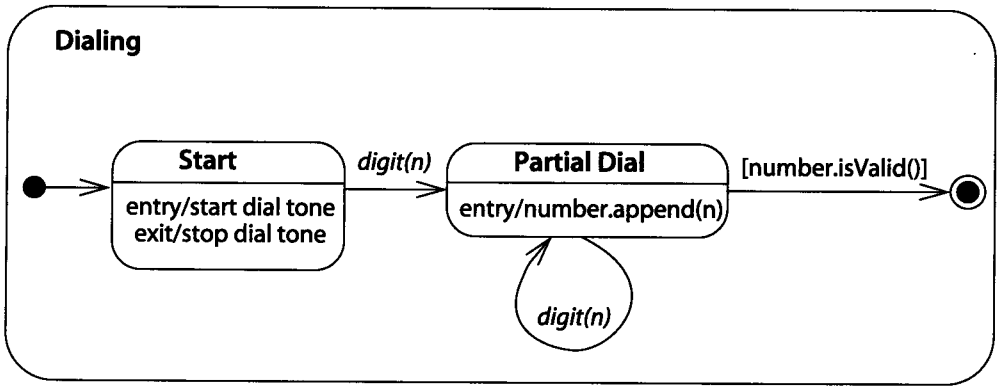


图13-66 顺序组成状态

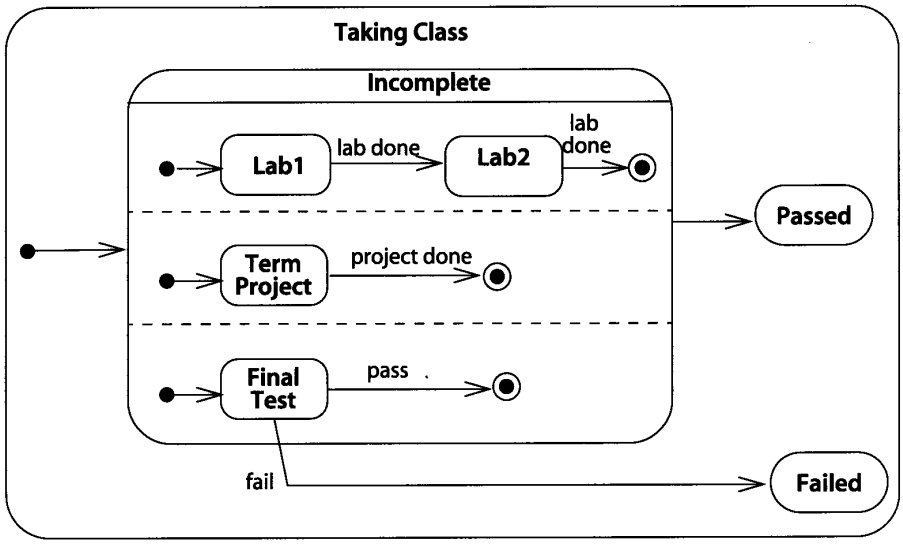


图13-67 并列组成状态

81. composition(组合)

组合是指一种带有很强所有关系且部分的生命期一致的聚合关联形式。一个部分可以仅属于一个组合。没有固定多重性的部分可以在组合创建后再被创建。但是一旦被创建，这些

China-pub.com 下载

部分将与组合同时存在并同时消亡（共享生存周期）。这些部分也可以在组合消失前被删除。组合可以是递归的。

见aggregation，association，composite object。

语义

很强的聚集关联形式被称为组合。组合是带有额外约束的聚集关联，一个对象在某一个时刻可以只属于一个组成，而且组成对象对其所有部分的安置负完全责任。作为第一个约束的结果，所有组成关系的序列（带有组成属性的所有关联）形成了一个对象和组合链的树林。一个组成部分不能同时被两个组成对象共享。这符合物理的组成概念 —— 一个部分不能同时成为两个对象的直接部分（虽然在不同粒度层次的树中它可以间接地成为多个对象的部分）。

组成对其组成部分负责，是指它要负责其部分的创建和销毁。在实现过程中，它负责部分的内存分配。在实例化过程中，组成必须保证其部分都生成了实例并与之正确地连接。它可以自行生成部分，或者对已存在的部分承担责任。但是在组成的生命期内，没有其他对象能对它的部分负责。这意味着设计组成类行为时应该了解，没有其他类可以销毁组成的部分或者对它们重新定位。在生命期内，组成如果对其部分负责，则可以增加额外的部分（如果多重性允许）。如果多重性允许且别的对象可以承担责任，则组成可以删除部分。一旦组成销毁，它必须销毁其部分，或将它们的安置权交给其他对象。

这个定义概括了组合的普遍逻辑和实现特征。例如一个包含属性值表的记录是一个对象及其属性的常见实现。当为记录分配空间时，其属性的存储空间也自动分配了，但属性值需要初始化。记录存在时，它的属性不能被删除。记录回收时，属性的存储空间也将被回收。其他对象不能影响记录中属性的空间分配。记录的物理特性符合组成的约束。

组合的定义便于无用单元的回收，如果组成自身被销毁，则指向其部分的唯一指针也被销毁，部分将不能访问，成为无用单元。收回不可访问部分对于无用单元回收来说是很简单的，而这正是区分组合与其他聚合的一个原因。

注意，部分不一定要被实现成为组成的存储块中的物理部分。如果一个部分独立于组成，则组成有责任根据需要为它分配/回收存储空间。例如，C++中组成的构造和析构功能的实现。

一个对象在同一时刻只能属于一个组成对象。这并不排除一个类在不同的时刻或者在不同的实例中成为多个类的部分。但同一时刻，一个对象只能存在一个组合链中。换而言之，一个部分可能属于“或”关系的组成中。并且，一个对象在其生命周期里可以是不同组成对象的部分，但是每个时刻只能属于一个组成对象。

结构

关联端的聚合特性可以为下列值：

none	所附类元不是聚集或组成。
aggregate	所附类元是聚集，另一端是其部分。
composite	所附类元是组成，另一端是其部分。

China-pub.com
下载

关联至少应该有一端的值为 none。

表示法

组合用关联路径上与组成元素相连一端的实心的菱形表示（如图 13-68）。多重性用常规方法表示，必须为 1或0..1。

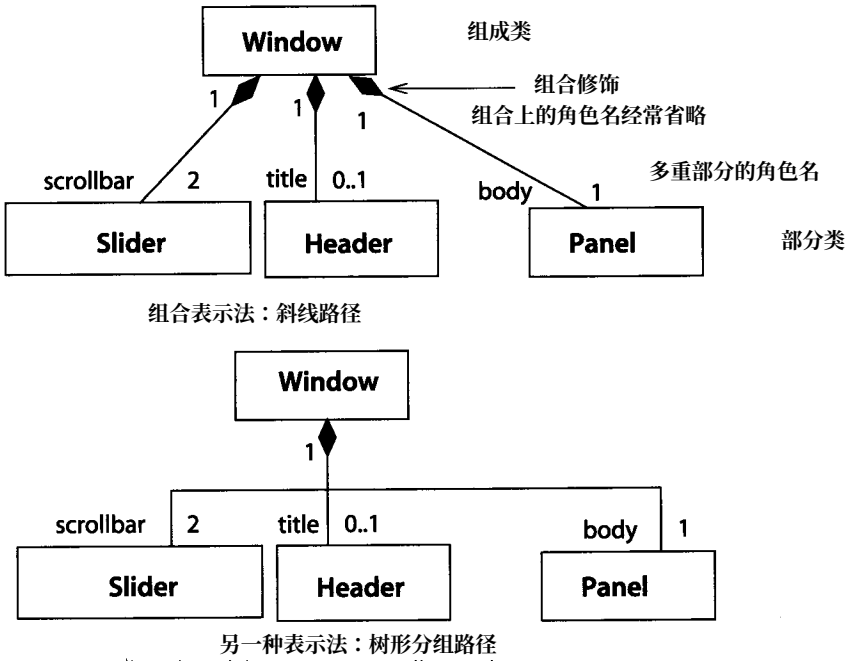


图13-68 组合关系表示法

另外，组合可以用组成符号中的嵌套部分符号表示（如图 13-69）。嵌套的类元可以在其组成元素中有多重性。多重性用字符串在部分符号的右上角标明。如果省略多重性标记，则其缺省值为“多”。组合中的嵌套元素可以有角色名。角色名在其类型前标出，语法为：

rolename:classname

其角色名是隐含的从组成到其部分的组合关联的角色名。

画在组成边界内的关联被认为是组合的部分。由关联的一条链连接的任何对象必须属于同一个组成。超出组成边界的关联不是组合的部分，由关联的一条链连接的对象可以属于同一个组成，也可以属于不同组成（如图 13-70）。

注意，属性实际上是类和其属性类之间的组合关系（如图 13-71）。但是通常属性被保留为简单数值（如数字、字符串、数据），而不是类的引用，因为在属性表示法中看不到部分类的其他关系。

注意，组合的表示法与协作的表示法类似。当协作的所有参与者是一个组成对象的部分

China-pub.com 下载

时，组合可以被看作一种协作。

图13-72是多层组合。

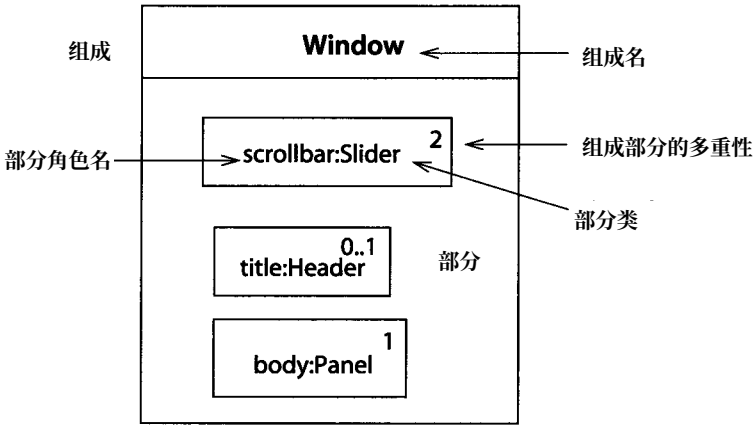


图13-69 图形嵌套的组合

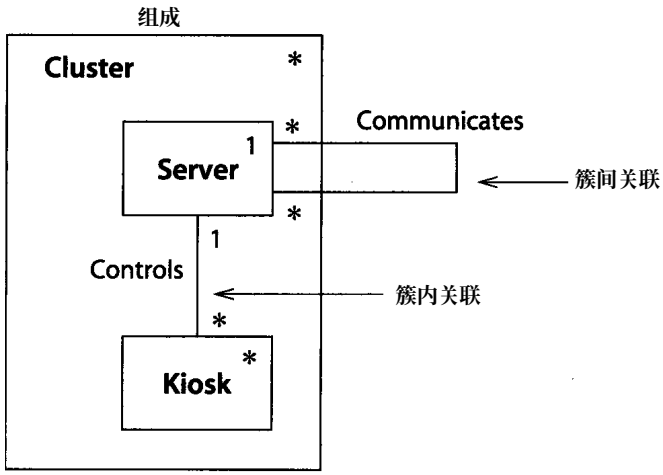
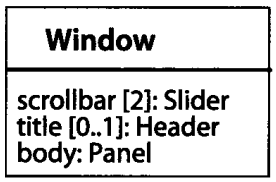


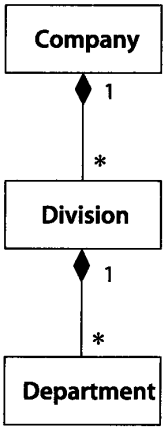
图13-70 组成之内和之外的关联



除非属性是非共享的和基于实现的，否则应尽量减少属性。

图13-71 属性是组合的一种形成

China-pub.com
下载



组合是可传递的：

部门是公司的间接组成部分

图13-72 多层组成

讨论

见aggregation的讨论，了解聚合、组合以及简单关联何时适用。

组合与聚合是元关系——它们超越了单个的关联，对整个系统施加约束。组合在组合关系中才具有意义。一个对象最多能有一个组合链（到组成），但是可能来自多个组合关联。即使链来至不同的关联，由组合和聚合链以及对象构成的图也必须是环状的。注意，这些约束适用于实例域——聚合关联自身常常为环状态，递归结构也总要求有关联环。

考虑图13-73中的模型。每个Authentication是一个Transaction的组成部分，Transaction可以是Purchase或Sale。但是并不是每一个Transaction都需要有一个Authentication。从图中可得出Authentication没有其他的组合关联。每个Authentication对象必须是一个Transaction对象（多重性为1）的组成部分，因为一个对象最多只能属于一个组成（根据定义），而Authentication已经是一个对象的组成部分（如图13-73所示），所以Authentication不能再成为其他组合关联的部分。可以由Authentication管理自身的存储区。虽然不是所有

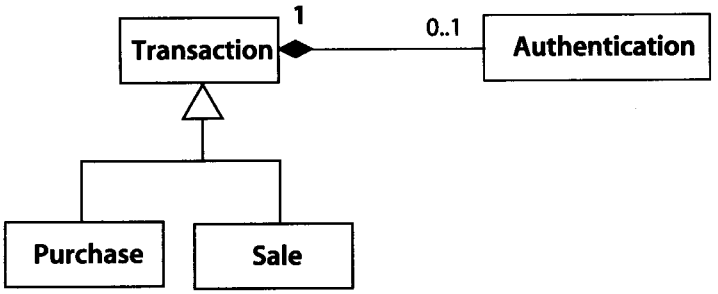


图13-73 到抽象组成类的组成

China-pub.com 下载

Transaction都有需要管理的Authentication，但是总可得到一个Transaction来承担管理责任（当然，如果设计者愿意，Authentication可管理自己）。

考虑图13-74，Autograph可以可选地成为Transaction或者Letter的部分。但是不能同时属于二者（由组合的规则决定）。该模型允许Autograph开始时是Letter的部分，随后再是Transaction的部分（此时，Autograph必须不是Letter的部分了）。实际上，Autograph不一定要是任何对象的部分。而且从本图中不能排除Autograph成为图中未标出的类或后来增加的类的部分的可能性。

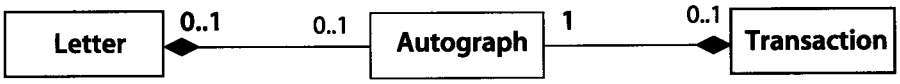


图13-74 共享部分的类

如果强调Autograph必须是Letter或者Transcation的部分呢？这需要重新设计模型。如图13-73，可以在其中增加一个Letter和Transaction上的新抽象超类（称为Document），且Autograph的组合关联从原来的类移到Document上。同时，Autograph到Document的多重性为1。

这种方法存在一个小问题：Document到Autograph的多重性必须被设为可选的，从而削弱了Transaction对Autograph的强制包含关系。可以用组合关联的泛化关系为这种情况建模，如图13-75所示。Autograph与Transaction间的组合关联被设定为Autograph与Document之间的组合关联的孩子。多重性为孩子声明（注意，它们仍然与祖先一致，因此可用孩子替代父母）。此外，通过在两个组合上增加约束条件，要求必须在两者中选一种，即可以使用原来的模型。

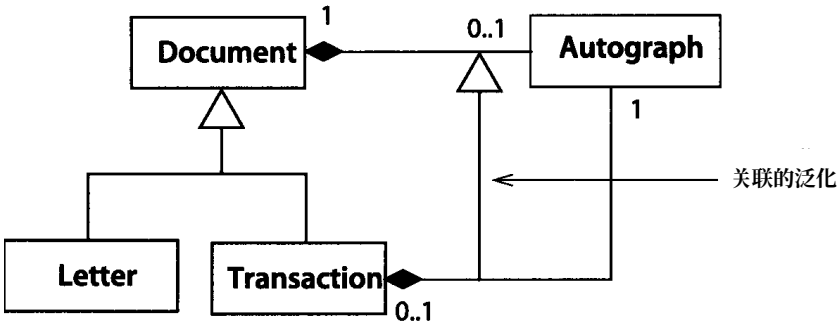


图13-75 组合关联的泛化

82. concrete(具体)

具体是指一个可以直接实例化的可泛化元素（如类）。它的实现必须经过充分的说明。对

China-pub.com
下载

类而言，它的所有操作必须被实现（由类或者其祖先）。反义词：abstract。

见direct class，instantiation。

语义

只有具体类元可以被实例化，因此泛化体系的叶节点必须是具体的。换而言之，所有抽象的操作和其他抽象属性都必须最终在某一个后代上实现（当然程序没有完成时，抽象类可能没有具体后代，如有待用户扩充的框架，但是在提供具体后代之前，这些类都不能在实现中使用）。

表示法

具体元素的名称用常规字体表示，抽象元素的名称用斜体。

83. concurrency(并发)

在同一时间间隔内，两个或两个以上的活动的执行。并不隐含地要求这些活动同步。通常，除了明确的同步点之外，它们的活动是相互独立的。可以通过插入或者同时执行多个线程来实现并发。

见complex transition，composite state，thread。

84. concurrency substate(并发子状态)

一个可以与同一个组成状态中的其他子状态同时存在的子状态。

见composite transition，disjoin substate。

85. conditional thread(条件线程)

活动图中由分叉的输入段到由相应的结合点的输入段结束的一块区域。

见composite state，complex transition。

86. conflict(冲突)

冲突是指从多个类继承相同名称的属性或操作时的情况，或一个事件触发多个转换，或者其他由正常规则可能产生矛盾结果的类似情况。根据不同模型元素的不同语义，用冲突化

China-pub.com 下载

解规则来解决冲突，冲突可能是合法的但产生了一个不确定结果，或者它可能表明了模型的错误。

讨论

冲突化解规则可以避免冲突出现。例如，如果多个类定义了相同的特征，那么使用最先的超类定义的该特征（这要求超类有序）。UML通常没有定义冲突化解规则，因为完全依赖它们是危险的。冲突易被忽视，常常成为模型中更深层问题的诱因。与其依赖这些微妙且易引起混乱的规则，不如要求一个精确的系统。在某种工具或者程序设计语言中，存在着这样的规则。最好在使用这些规则时，有工具给出警告，使建模者意识到冲突的出现。

87. constraint(约束)

约束是一个语义条件或者限制的表达式。 UML预定义了某些约束，其他可以由建模者自行定义。约束是UML的三个可扩展机制之一。

见expression，stereotype，tagged value。

见第14章，其中有预定义的约束列表。

语义

约束是用文本语言中的陈述句表达的语义条件或者限制。

通常，约束可以附属于任何一个或者一列模型元素上。它表示附属于模型元素上的语义信息。每个约束有约束体和解释语言。约束体是约束语言中关于条件的布尔表达式的字符串。约束应用于有序的一个或一列模型元素。应注意，这里的语言可以是形式化语言，也可以是自然语言。如果是自然语言，则约束是非形式化的，不能自动执行（但是形式化的约束也不一定都可以自动执行）。UML提供了约束语言OCL[Warmer-99]，也可以使用其他语言。

某些常用约束具有名称，从而避免每次使用时写出完整的语句。例如：两个共享同一个类的关联之间的xor约束表示共享类的一个对象在同一时刻只能属于关联的一方。

约束不是可以执行的机制，而是一种断言。它是表示必须由系统的正确设计来实施的限制。如何保证约束的实现是设计的任务。运行时约束作用于系统实例的“稳定”时刻，即，操作的执行和没有原子转换正在进行的时刻之间的时刻。在一个操作的执行过程中，可能在某些时刻暂时违反约束。

约束不能作用于自身。

即使后代定义了额外的约束，继承的约束 —— 在祖先模型元素或者构造型上定义的约束 —— 必须被遵守。继承约束不能忽略。如果有这样的需要，说明模型的结构不好，应该重新构造。但是，可以增加限制条件来加强继承约束。如果元素的继承约束有冲突，说明模型为非良性结构。

China-pub.com
下载

表示法

约束用大括号（{}）中的文本字符串表示。文本字符串是用约束语言写的代码体。

工具应该提供一种或几种形式化的约束语言。一种描述约束的预定义语言是 OCL。根据模型的不同，一些计算机语言（如 C++）也可以用来表示约束。此外，约束还可以用自然语言描述，这时，约束的翻译和执行由人工完成。每一种约束的语言是约束的一部分，但是在图中不一定标出（工具将保留其记录）。

对于分栏中用字符串表示的一列元素（如类的属性），约束字符串可能作为元素列的入口（如图13-76）。入口不代表一个模型元素，而是作用于其后列出的模型元素上的运行约束，其作用范围为直到出现另一个运行约束，或者元素列到头。运行约束可以被列表中稍后出现的新约束所替代。要消除运行约束，可以用空的约束来替代它。附属于某个列表元素的约束不能替代运行约束，但是可以为其增加额外的限制。

对于简单图形符号（如类或者关联路径），约束字符串可以标在图形符号旁边，如果图形符号有名称，则标在名称旁边。

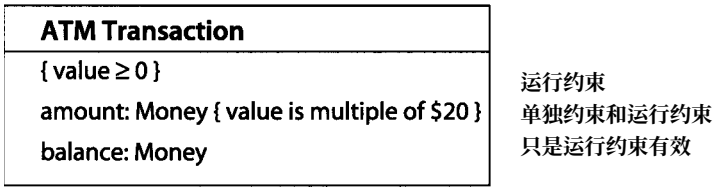


图13-76 带列表的约束

对于两个图形符号（如两个类或两个关联），约束用虚线箭头表示。箭头从一个元素指向另一个，并带有约束字符串（在大括号内）。箭头的方向与约束的信息相关。

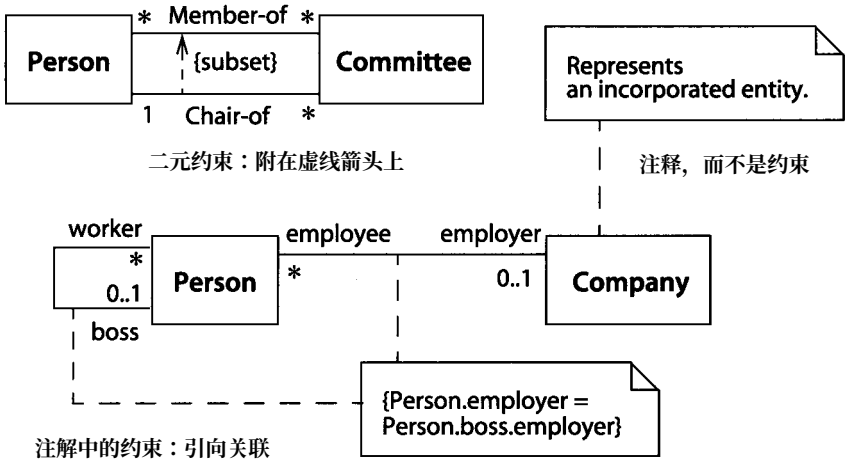


图13-77 约束表示法

China-pub.com 下载

对于三个或更多的图形符号，约束字符串位于注解符号中，并用虚线与各个图形符号相连（如图13-77）。这种表示法适用于其他情况。对三个或更多的同类路径（如泛化路径或者关联路径），约束标在穿过所有路径的虚线上。为避免混淆，不同的连线可以标号或加标签，从而建立它们与约束之间的对应关系。

讨论

约束是关于模型本身的语义表述，而注释是没有语义效用并可以附属于模型元素或表示元素上的文本声明语句。约束和注解都可以用注解符号表示。原则上，约束是可由工具执行的。在实践中，某些约束难以形式化地表述，要由人工强制执行。从广义上讲，模型中的许多元素都是约束，但是这里的约束是指那些不能用内置模型元素表达，必须单独用语言声明的约束语句。

约束可以使用多种语言，甚至使用人类语言来表述，但是工具不能验证用人类语言描述的约束。OCL语言[Warmer-99]专用于UML约束的表达，但是某些情况下，某些程序设计语言可能更适用。

因为约束用文本字符串表示，故普通的建模工具可以不必理解其含义就能读入并维护它。当然，用来验证或执行约束的工具或者插件必须能理解目标语言的语法和语义。

一些约束可以附加在构造型定义上，这说明所有属于此构造型的元素都处于这些约束下。

执行。当模型中有约束时，不一定要求给出约束被违反后的操作。模型只是对可能出现的情况的声明，使这些情况发生是实现的任务。一个程序可以包含断言和其他验证机制，违反约束应被视为程序失败。当然，如果模型能够帮助产生结构正确或可被验证的程序，则此模型达到了目标。

标准元素

invariant，postcondition，precondition。

88. construction(构造)

软件开发过程中的第三阶段，在此阶段进行详细设计，实现系统，进行软件、固件和硬件测试。在此阶段，分析视图和设计视图基本完成，并完成大部分实现视图和一些部署视图。

见development process。

89. constructor(构造函数)

创建并初始化类的实例的一种类作用域操作，可以作为操作的构造型使用。

China-pub.com
下载

见create、instantiation。

90. container(包容器)

包含其他对象的对象，它提供了访问或迭代其内容的操作，或者一个描述此类对象的类。例如数组、列表、集合。

见aggregation，composition。

讨论

通常不必明确地对包容器建模。它们通常是关联的“多”端的实现。在多数模型中，多重性大于1足以表明正确的语义。当使用设计模型生成代码时，用于实现关联的包容器类可以被指定为使用标记值的代码生成器。

91. context(语境)

与某个目标相关的建模元素集的视图，如执行一个操作或者构造一个模式。语境是模型的一部分，它为其中的元素提出约束或者提供环境。协作为其内容提供语境。

见collaboraion。

92. control flow(控制流)

在交互中，控制的后继轨迹之间的关系，如活动图或协作。

见action，activity graph，collaboration，completion transition，message，object flow state，transition。

语义

交互视图代表了计算过程中的控制流。交互中的简单元素是活动和对象。控制流表示动作与其参与者和后继动作之间以及动作和其输入和输出对象之间的关系。在简化的格式中，控制流表示一个对象到另一个对象，或者一个对象在不同时刻的不同版本之间的计算上的导出。（包含对象输入输出的控制流称为对象流。）

表示法

在协作图中，控制流用附加在连接类元角色（代表对象及其实例）的关联角色（代表链）

China-pub.com 下载

上的消息表示。在活动图中，控制流用活动符号之间的实心箭头表示。对象流用活动符号或者控制流箭头和对象流状态符号之间的虚线箭头表示。见相关章节，以做进一步了解。

93. control icons(控制图标)

简略地表示各种控制模式的可选符号。

见activity graph，collaboration，state machine。

表示法

下列符号为活动图而设计，但也可用于状态图。这些符号不允许那些不能用基本符号表示的内容，但对于常规控制模式而言，它们很方便实用。

分支。分支是由单一状态发出的一系列转换，必须总有一个转换上监护条件被满足。换而言之，如果发生触发器事件，只有一个转换能够激发。这些监护条件本质上表示了控制的分支。如果是完成转换，则分支是纯决策。为了方便起见，可以有一个分支的输出被标为else。如果没有选择其他路径，则走这一条。

分支由具有一个输入箭头和多个输出箭头的菱形表示。输入转换箭头上标明事件触发器（如果有），每个输出上标有监护条件（如图13-78）。

合并。合并是两个或两个以上的可选控制路径聚中的地方。它与分支相反。菱形同时是分支和结合的标志。如果有多个输入箭头，说明符号代表合并；如果有多个输出箭头，说明符号代表分支（如图13-78）。不一定必须有合并（多重转换进入单一状态称为合并），但它们对于表示与先前分支的匹配还是很有用的。

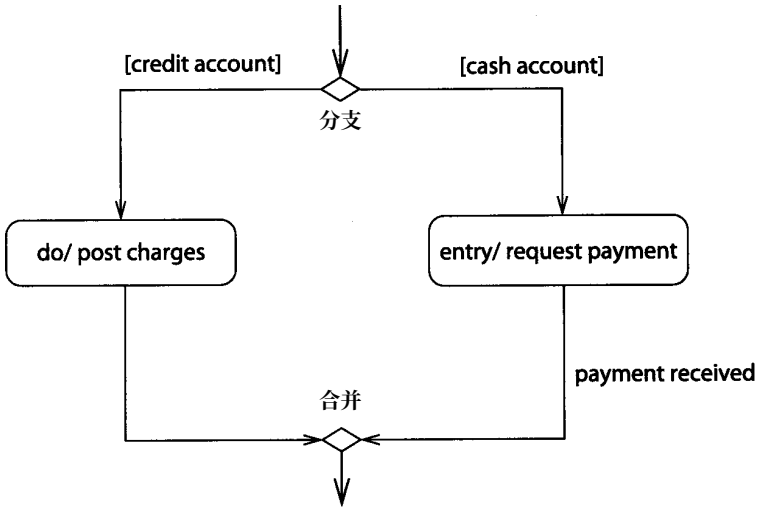


图13-78 分支与结合

China-pub.com
下载

信号接收。信号接收用凹五边形表示。信号的特征标记在符号内标出。由一个不带标签的转换箭头从先前动作状态指向五边形，另一个不带标签的转换从五边形指向其后继动作状态。该符号取代了转换上的事件标签，该转换在先前活动结束后，事件发生时被触发（如图13-79）。此外，可以用虚线箭头从对象符号指向五边形的缺口，表示信号的发送者。

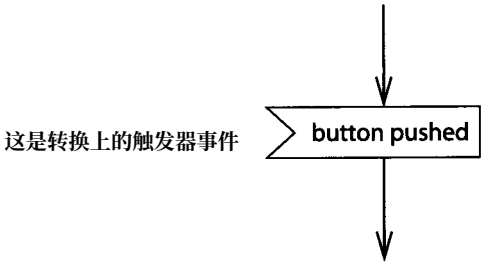


图13-79 信号接收

信号发送。信号发送用凸五边形表示。信号的特征标记在符号中标出。有一个不带标签的转换从先前动作状态指向五边形，另一个不带标签的转换由五边形指向其后继动作状态。这个符号取代了转换上的信号发送标签（如图 13-80）。此外，可以用虚线箭头从五边形的顶点指向对象符号，表示信号的接收者。

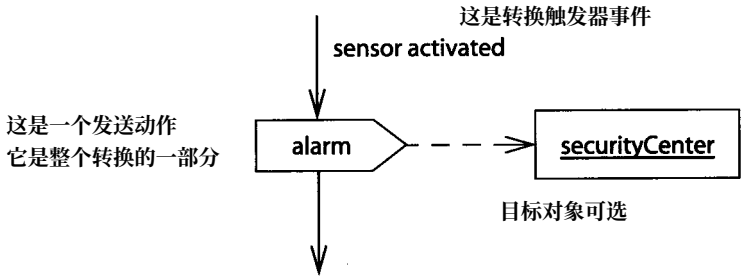


图13-80 信号发送

示例

如图13-81，EnterCreditCardData和ChargeCard是活动。当它们完成后，执行转入下一步。在EnterCreditCardData完成后，出现了根据金额区分的分支：如果所需金额大于25美元，则要求出示证明。一个 request信号被送往信用中心。在普通状态机中，它可以表示为从EnterCreditCardData转出的转换上的一个动作，二者意义相同。 AwaitAuthorrization是一个等待状态。它不是内部完成的活动，而必须等待来自信用中心的外部信号（ authorize）。当信号出现，触发了一个常规转换，系统进入 ChargeCard活动。触发器事件可以作为AwaitAuthorrization和ChargeCard之间的转换上的标签，二者是同一事物的不同表示法。

China-pub.com 下载

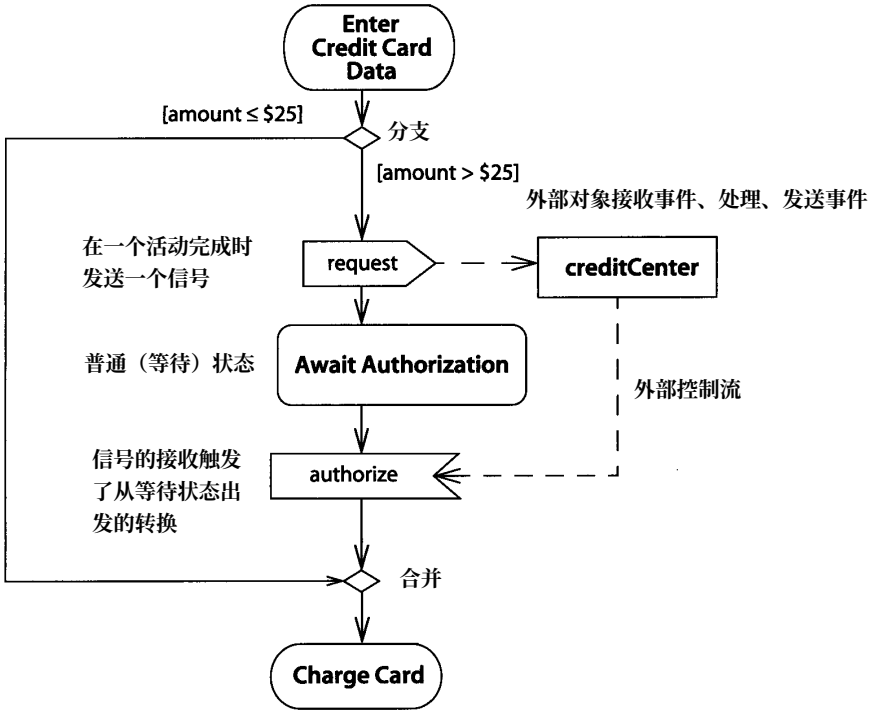


图13-81 表示发送和接收信号的活动图

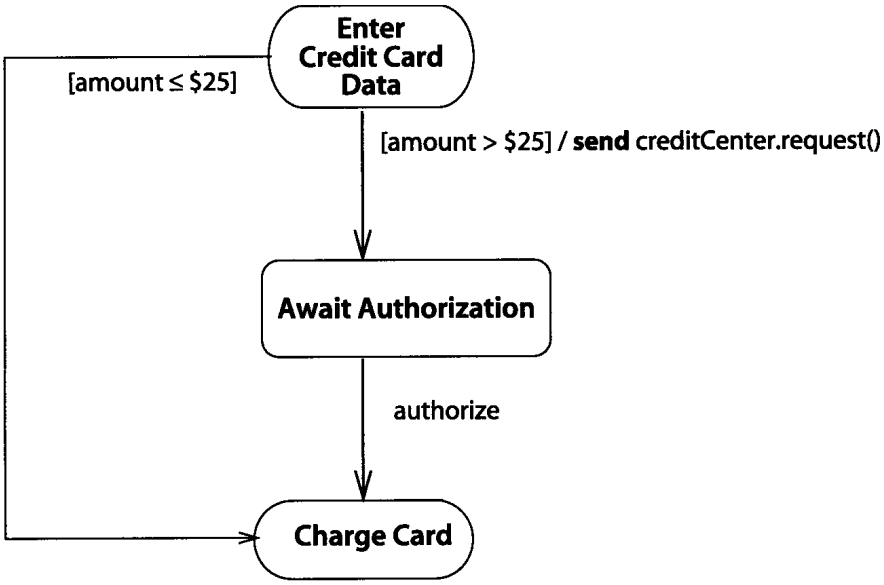


图13-82 没有特殊符号的活动图

China-pub.com
下载

图13-82是相同的例子，只是没有使用特殊的控制符号。

94.copy(复制)

交互中使用的一种流关系，目标对象成为源对象的一个副本，随后二者相互独立。

见become。

语义

复制关系是一种流关系，表示在交互中从一个对象导出另一个对象。它表示制作副本的动作。复制流执行过后，两个独立对象的值可以独立的变化。

交互中的复制转换可以具有顺序号，表示它相对于其他动作发生的时间。

表示法

复制流用从初始对象指向新生成对象的虚线箭头表示。箭头上标有构造型关键字《copy》，还可以有顺序号。复制转换可以出现于协作图、顺序图和活动图中。

示例

图13-83表示一个文件在另一个节点上的备份。首先进行复制（《copy》），随后将副本移动到下一个节点（《become》）位置。

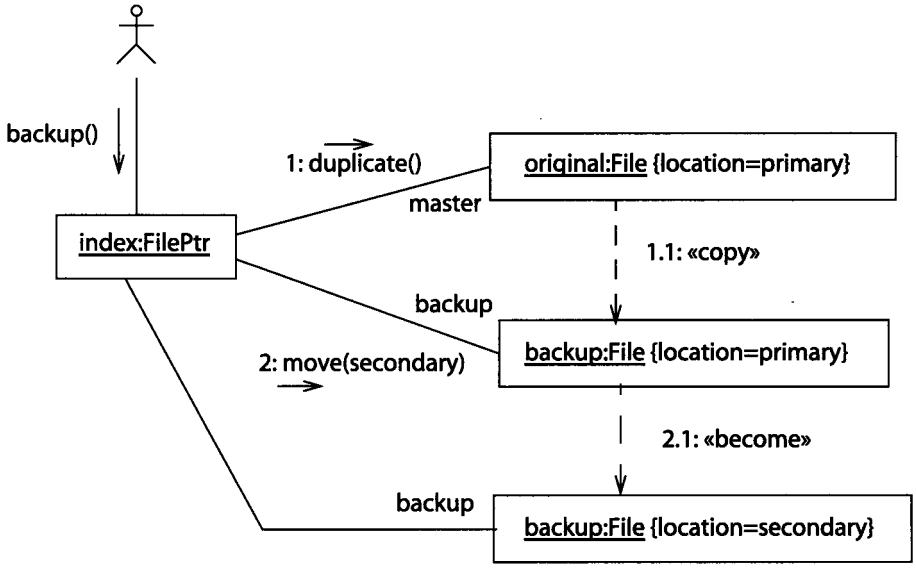


图13-83 复制和变成流

China-pub.com 下载

95. creation(创建)

对象或其他实例（如用例实例）的实现和初始化。反义词： destruction。

见instantiation。

语义

创建对象是实例化对象消息的结果。创建操作可以有参数，它们将用于新生实例的初始化。创建操作后，新的对象遵从其类的约束，并可以接收消息。

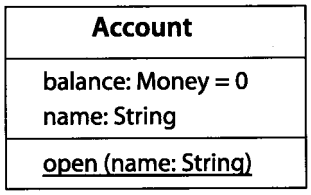
创建操作或者构造函数可以被声明为类作用域的操作。此类操作的目标（至少在概念上）是类本身。程序设计语言如 Smalltalk中，类被作为实际运行时对象实现，因此创建操作被作为对象的常规消息实现。在 C++中，没有实际的运行时对象，创建操作可以被视为运行时被优化的概念化消息。 C++的方法中排除了实例化类的计算，否则，每一个方法都可被建模为传递给类的消息。

类属性的初始值表达式（在概念上）是由创建操作计算的，其结果用于属性的初始化。创建操作的代码可以隐式取代这些值，因此初始值表达式是可重载的缺省项。

在状态机中，创建对象的构造操作的参数作为离开顶层初始状态的转换上的当前事件使用。

表示法

在类图中，创建操作（构造函数）做为一个操作包含在类的操作表中。它可以有参数表，返回值是类的实例，可以省略。作为类作用域的操作，其名称带有下划线（如图 13-84），表示《constructor》构造型。



初始值

创建操作

图13-84 创建操作

顺序图中的创建操作的执行用消息箭头表示，箭头指向对象符号（带有下划线对象名的矩形）。对象符号下方是对象的生命线（根据对象是否主动，可以为虚线或者双实线）。它持续到对象被销毁或者图结束（如图 13-85）。

协作图中创建操作的执行用带有特性 {new}的对象符号表示。传递给对象的第一条消息隐式为创建对象的消息。虽然消息实际指向类的本身，但通常被省略，且消息用于实例（新生但未初始化）的初始化，如图 13-86所示。

China-pub.com
下载

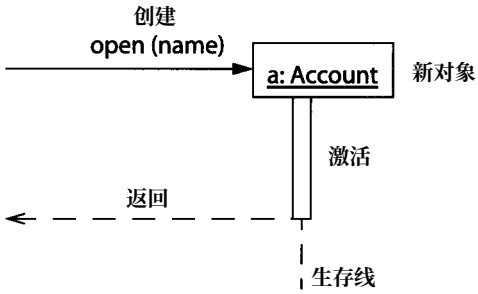


图13-85 创建顺序图

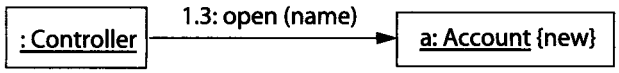


图13-86 协作图中的创建

见collaboration diagram，sequence diagram，其中有过程实现中创建的表示法。

96. current event(当前事件)

在状态机的执行中触发运行至完成步骤的事件。

见run to completion，state machine，transition。

语义

为了方便起见，状态机为响应一个事件可以遍历几个相连的转换片断。除最终转换外的所有转换都进入伪状态——为了构造状态机而存在的哑状态，但不等待外部事件。原则上，所有的转换可以汇集到一个转换上，但是，用伪状态划分多状态可以使多个转换共享普通后继事件。

转换链的执行是原子的，即它是不能被外部事件打断的运行至完成步骤的一部分。在这样的一列转换的执行中，附属的动作和监护条件可以隐含地访问通向触发的第一个转换的事件及其参数。这个事件就是转换中的当前事件。当前事件的类型可以被多态操作或者条件语句识别。一旦知道了确切的类型，就可以访问其参数了。

当前事件特别有利于帮助新创建对象的初始转换得到初始化参数。新对象创建后，创建事件成为当前事件，其参数可以被新对象状态机的初始转换所使用。

示例

图13-87 是从Idle状态到Purchase状态的转换，它由request事件触发。Purchase的入口动作调用setup操作，它使用当前事件。程序可以访问当前事件，从而得到 request事件及其参数product。如果有当前事件的多个绑定，则程序需要一个条件语句来得到正确的触发事件。语法由程序设计语言说明。

China-pub.com 下载

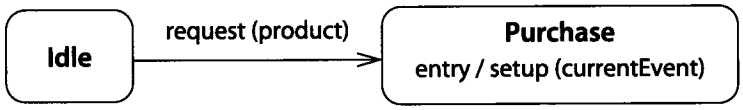


图13-87 当前事件的使用

表示法

当前事件的命名可以在带有关键字 currentEvent的表达式中完成。特定的表达语言提供了更详细的语法。

97. data type(数据类型)

没有身份标识的一组值的描述符（独立存在，可能有副作用）。数据类型包括预定义的简单类型和用户自定义的类型。简单类型有：数字、字符串、时间。用户定义的类型是枚举类型。程序语言中用于实现的匿名数据类型可以用语言类型定义。

见classifier，identity。

语义

数据类型是预定义的简单类型，是用户可定义类型的基础。它们的语义是在语言结构之外用数学定义的。数字是预定义的，包括实数和整数。字符串也是预定义的。这些类型是用户不可定义的。

枚举类型是用户自定义的一组命名元素，它们之间有预定义的顺序，此外没有其他特性。枚举类型有名称和一系列枚举值。枚举类型 Boolean 是预定义的，其枚举值为 false和true。

操作可以在数据类型上定义，操作的参数可以有数据类型。因为数据类型没有身份，只是简单值，故数据类型上的操作不对数据类型进行修改，而是直接返回值。因为没有身份，故创建新的数据类型是没有意义的。所有数据类型的值（概念上）都是预定义的。数据类型上的操作是对系统的一种查询，它不改变系统状态，仅返回一个值。

数据类型还可以用语言类型描述 —— 程序设计语言中的数据类型表达式。此类表达式在目标语言中指定了一个匿名的数据类型。例如，表达式 Person*（*）（String）是C++中的一个类型表达式，它不是简单的数据类型。

98. data value(数据值)

数据值是数据类型的实例，是无身份的值。

China-pub.com
下载

见data type，object。

语义

数据值是数学域的一个成员——一个纯数值。因为数据值没有身份，两个表示法相同的数据值是无法区分的。在程序设计语言中，数据值使用值传递。用引用传递数据值是没有意义的，修改数据值也是没有意义的，它的值是永远不变的。实际上，数据值就是值本身。通常所说的修改数据值，是指修改一个有数据值的变量，使其具有新的数据值，而数据值本身是不变的。

99. default value(缺省值)

作为某些程序设计语言或者工具的一部分而自动提供的值。元素特性的缺省值不是 UML 语义的组成部分，在模型中不出现。

见initial value, parameter, unspecified value。

100. deferred event(延迟事件)

当对象处于特定状态时，一个识别被延迟的事件。

见state machine，transition。

语义

一个状态可以将一系列事件指定为延迟的。当对象处于使某个事件延迟的状态时，如果该事件发生，则它将不会触发相应转换，即事件没有及时效应。直到对象转入一个不要求该事件延迟的状态之后，事件才有效用。若在该状态活动时出现其他事件，则按照常规处理。当对象进入新的状态之后，不再被延迟的保存事件一一出现，并在新的状态下触发转换（先前被延迟的事件的出现顺序是不定的，依赖事件的特定出现次序是危险的）。如果事件在新的状态下没有触发转换，则它将被忽略并丢失。

在普通的状态机中，应小心地使用延迟事件。它们可以更直接地被构造为并行状态相应的事件，在此期间，主运算部分可以进行其他工作。可将它们用于活动图中，它们可以保证运算顺序进行，并且不会丢失异步消息。

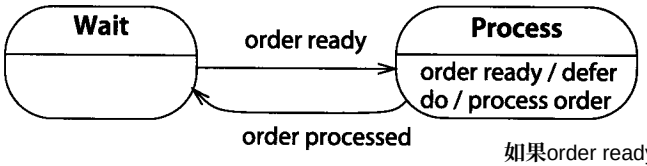
如果状态中有被延迟事件触发的转换，该转换将重载延迟且事件触发转换。

表示法

延迟事件用带有特殊保留动作 defer的内部转换表示。延迟用于状态及其嵌套子状态。

China-pub.com 下载

（如图13-88）。



如果order ready 发生，则它将被延迟至指向Wait 的转换发生为止

图13-88 延迟事件

101. delegation(委派)

一个对象对于另一个对象所发的消息的回应能力。委派可以用来替换继承。在某些语言（如self）中，它由语言的继承机制支持。在大多数其他语言中（如 C++和Smalltalk），它可以通过与另一个对象的聚合或者关联实现。第一个对象的操作要求第二个对象的操作继续完成它的工作。对比：inheritance。

见association。

102. dependency(依赖)

两个元素之间的一种关系，其中一个元素（提供者）的变化将影响另一个元素（客户），或向它提供所需信息。它是一种将数种不同的建模关系组织到一起的简便方法。

语义

依赖关系是表示一个或几个模型中两个元素间关系的语句。它可以将几种不同的元素结合起来，好像生物学中用“无脊椎动物”结合所有不是“脊椎动物”的不同门的生物。

在表示不对称的知识系统时，独立的元素称为提供者，不独立的元素称为客户。

为了说明依赖关系在模型中的角色，它可以带有名称。但是依赖关系自身的表示通常足以说明问题，不需另加名称。依赖关系可以有构造类型，用来建立依赖关系的精确属性。依赖关系还可以有文本说明，用来对它进行非正式但很详细的描述。

两个包之间有依赖关系，说明两个包中至少一对元素之间有依赖关系（除直接与包相关的访问与引入之外）。例如，两个类之间的使用依赖关系可以表示为两个包含它们的包之间的依赖关系。两个包之间有依赖关系不能说明两个包中所有的元素有依赖关系——事实上，这种情况非常少见。

见package。

China-pub.com
下载

依赖关系中可以包含下属依赖关系的引用。例如，两个包之间的依赖关系可以引用类之间基本的依赖关系。

依赖关系不必传递。

注意，关联和泛化关系满足依赖关系的一般定义，但是它们有各自的模型表示法和含义，通常不被视为依赖关系。实现关系有独立的表示法，但是被视为依赖关系。

依赖关系有几种表示不同关系种类的变体：抽象、绑定、组合、许可、使用。

抽象。抽象依赖关系表示概念的抽象层次。两个元素用不同的方法表示同一个概念。通常一个元素更抽象，另一个更具体，也可能两个元素是同一抽象层次的不同表示法。根据关系的具体程度，抽象包括构造型踪迹、精化（关键字 refine）、实现（有自己特殊的表示法）、导出（关键字derive）。

绑定。绑定依赖关系将由模板约束的元素与模板相连。模板参数的参量作为列表附属于绑定依赖关系。

许可。许可依赖（通常作为特定的构造型）将包或者类与另一个允许它使用某些内容的包或者类相连。许可依赖关系的构造型有访问、友元、引入。

使用。使用依赖关系（关键字《use》）将客户元素与提供者元素相连。提供者的变化将导致客户的变化。使用通常表示一种实现依赖关系，其中一个元素依靠另一个元素的服务来实现自身的操作。使用的构造型包括调用、实例（关键字 instantiate）、参数、发送。这是一个开放的列表，不同的程序设计语言中有其他的使用依赖关系。

表示法

依赖关系用两个模型元素之间的虚线箭头表示。箭尾处的模型元素（客户）依赖于箭头处的模型元素（提供者）。箭头上可带有表示依赖关系种类的关键字，还可以有名称（如图13-89）。

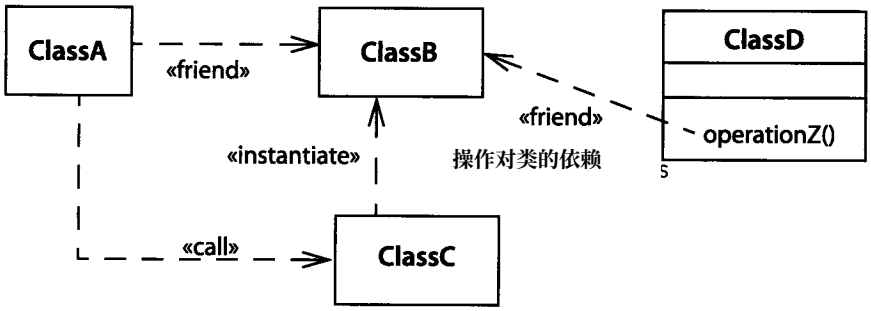


图13-89 类之间的一些依赖关系

有几种其他的关系也使用有关键字的虚线箭头表示，但它们不是依赖关系。这些元素有流（变成和复制）、组合（扩展和包含）、附属于所描述模型元素的约束和注解。如果注解或者约束是元素之一，可以省略箭头，因为约束或者注解总是在箭尾处。

China-pub.com 下载

标准元素

become，bind，call，copy，create，derive，extend，include，import，friend，instanceOf，instantiate，powertype，send，trace，use。

103. deployment(部署)

描述现实世界环境中运行系统的配置的开发步骤。在这一步骤中，必须决定配置参数、实现、资源配置、分布性和并行性。这一步骤的结果体现于部署图和部署文件中。

见development process，stages of modeling。

104. deployment diagram(部署图)

表示运行时过程节点、构件实例及其对象的配置的视图。构件表示代码单元在运行时的表现。运行时不存在的构件不出现在部署图中，而是在构件图中表示（因为它们已经被编译）。部署图表现实例，构件图表现构件类型的定义。

见component，interface，node。

语义

部署图含有用通信链相连的节点实例。节点实例包括运行时的实例，如构件实例和对象。构件实例和对象还可以包含对象。模型可以表示实例及其接口之间的依赖关系，还可以表现节点或者其他包容器之间实体的移动。

部署图有描述符形式和实例形式。实例形式（前文已经介绍过）表现了作为系统配置的一部分的具体节点实例上的具体构件实例的位置，这是部署图的常见形式。描述符形式说明哪种构件可以存在于哪种节点上，哪些节点可以被连接，类似于类图。

表示法

部署图是节点符号与表示通讯关联的路径构成的网状图（如图 13-90）。节点符号可以包含构件实例，说明构件存在或运行于该节点上。构件符号可以包含对象，说明对象是构件的一部分。构件之间用虚线箭头相连（可能穿过接口），说明一个构件使用了另一个构件的服务。必要时可以用构造型说明依赖关系。

部署图类似于对象图，通常用于表示系统中的各个节点的实例。很少用部署图定义节点的种类和节点之间的关系。

构件在节点之间的移动和对象在构件之间的移动可以用带有关键字《become》的虚线箭头表示。在这种情况下，构件或对象只在一部分时间内处于相应节点或者构件上。后文图 13-

China-pub.com
下载

133的部署图中的对象在节点之间移动。

见become。

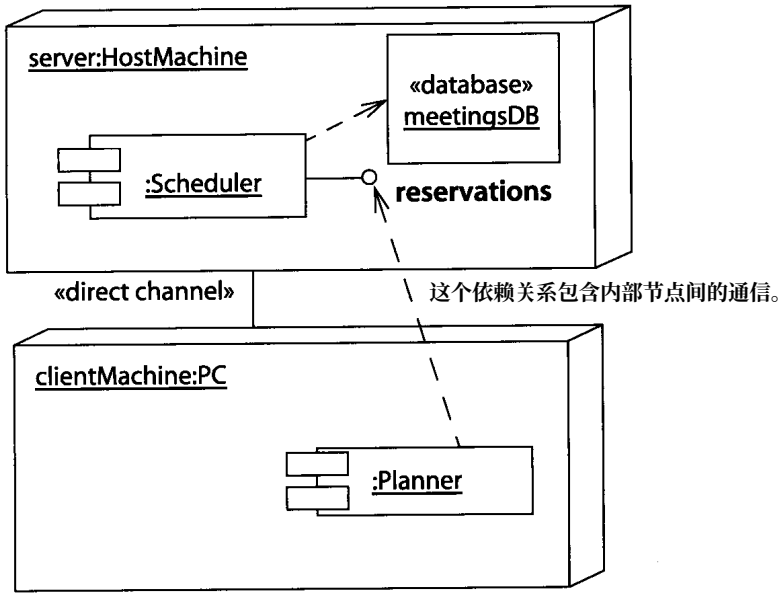


图13-90 客户-服务器系统的部署图

105. deployment view(部署视图)

表示分布式系统中的节点、各个节点上的构件以及节点上的构件中的对象的视图。

见deployment，deployment diagram。

106. derivation(导出)

两个元素间的一种关系，可从一个元素计算出另一个元素。导出可建模成一个带有关键字derive的抽象依赖关系的构造型。

见derived element。

107. derived element(导出元素)

可以由其他元素计算得出的元素，它不增加语义信息，只是为系统设计的方便而存在。

见constraint，dependency。

China-pub.com 下载

语义

模型中的导出元素在逻辑上讲是多余的，因为它可以由一个或者几个其他元素计算得出。计算导出元素的公式可以作为约束给出。

模型中使用导出元素有几种原因。从分析层次上看，导出元素不是必须的。但是它可以作为一种宏来定义或者命名一个有用的概念。必须记住，导出元素没有为模型增加语义信息。

在设计层次的模型中，导出元素表示一种优化 —— 在模型中，一个元素可由另一个元素计算得出但物理上并不存在，这避免了重复计算的开销。导出元素的例子有计算的中间结果或者对一列值的索引。若导出元素所依赖的元素变化，则导出元素随之刷新。

表示法

导出元素用元素名称之前的斜线（/）表示，如属性、角色名或者关联名（如图 13-91）。

计算导出元素的具体方法可以用带有构造型《derive》的依赖关系说明。通常，将依赖关系的箭头从元素的约束中去掉，并在导出元素旁加一个约束字符串。

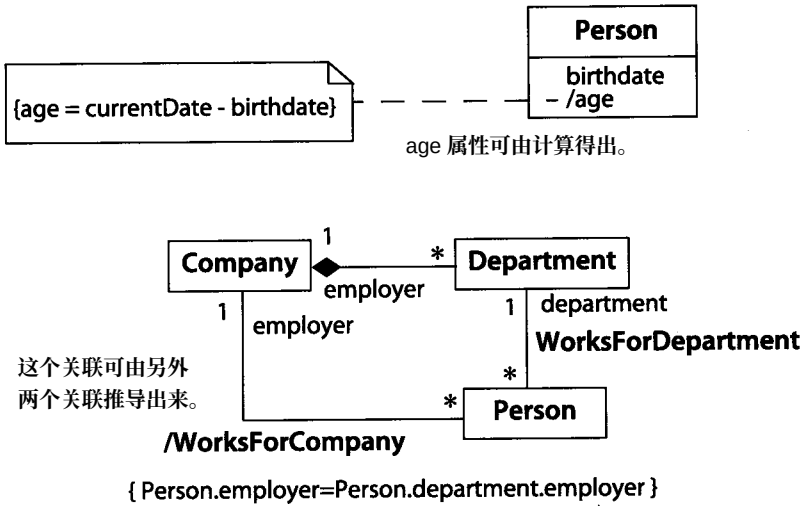


图13-91 导出属性和导出关联

讨论

导出关联可能是最常见的导出元素。它提供了一种可以由两个或者两个以上的基础关联计算出来的虚拟关联，如图 13-91。导出的关联为 WorksForCompany，可以由 WorksForDepartment和employer组合计算得出。在实例中使用 WorksForCompany可以避免重复进行计算，但是这一关联不提供任何额外信息。

关联泛化（如图13-30）略有不同。它提供了针对同一个关联的两个不同层次的细节内容。

China-pub.com
下载

通常不会同时实现两个层次，而是只实现子关联。有时也会实现父关联，并将子关联作为约束。

108. descendant(后代)

后代是指一个子或者含有一个子关系的链的元素，是特定关系的传递闭包。反义词：ancestor。

109. descriptor(描述符)

用来描述一系列实例的共有特性的模型元素，特性包括结构、关系、行为、约束、目的等等。对比：instance。

语义

描述符是描述一列实例的模型元素。模型中的大多数元素都是描述符。这个词几乎包含了模型的所有内容——类、关联、状态、用例、协作等。有时类型也表示相同的含义。但是通常类型的适用范围比较窄，仅仅指类似于类的元素。而描述符包括所有描述性的元素。它有内涵和外延，内涵是结构描述和常规规则，外延是描述符所解说的实例。外延在运行时不一定是物理上可以访问的。模型中的一对重要关系即描述符-实例。

表示法

描述符及其实例之间的关系使用相同的集合符号表示，不同的是实例的名称下有下划线。描述符有名称，而实例既有自身的名称又有描述符名称，二者用冒号隔开，名称带有下划线。

110. design(设计)

系统的一个阶段，它从实际代码之上的逻辑层次说明系统将如何实现。在设计中，从战略和战术上确定如何满足功能需求和质量要求。这一步的成果体现为设计层模型，特别是静态视图、状态机视图和交互视图。对比：analysis design, implementation, deployment。

见stages of modeling，development process。

111. design time(设计时间)

在软件开发过程的设计活动中出现的（情况）。对比：analysis time。

见modeling time，stages of modeling。

China-pub.com 下载

112. destroy(销毁)

消灭一个对象并收回其资源。它通常是一个明确的动作，也可以是其他动作、约束或废物回收的结果。

见destruction。

113. destruction(销毁)

消灭一个对象并收回其资源。消除一个组成对象将导致消除其组成部分。消除一个对象，不会消除与它通过一般关联或者聚合关系相关的对象，但是包含该对象的链接将被消除。

见composition，final state，instantiation。

表示法

过程实现中销毁的表示法见协作图和顺序图（如图13-162）。在顺序图中，在对象的生命线上画大×表示对象被销毁（如图13-92）。它位于导致对象被销毁的消息上，或者位于对象自我终结的地方。销毁对象的消息可以带有构造型《destroy》。在协作图中，用对象上的约束{destroyed}表示对象在交互中被销毁。如果对象在交互中生成并被销毁，可以使用约束{transient}。

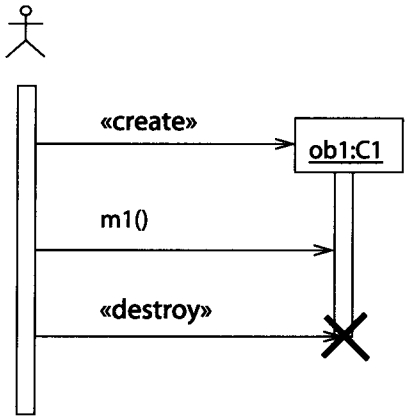


图13-92 创建和销毁

114. development process(开发过程)

为了以一种受控制、可重用的方式生产软件而进行的一系列部分有序活动以及相关指导。软件开发过程的目的是为了保证完成系统的功能和可靠性。

见stages of modeling。

China-pub.com
下载

讨论

UML是一种建模语言，而不是过程。它的目的是描述模型，而该模型可以用不同的开发过程实现，为了标准化，描述开发工作的结果比解释开发过程更为重要。因为有很多很好的建模方法，而且模型的最终使用者也不必了解它的构造过程。因此 UML支持多种过程。

关于迭代、增量、用例驱动、基于构架的开发过程的详细信息，见 [Jacobson-99]。

建模阶段与开发阶段的关系

建模阶段适于迭代的开发过程，包括初始、细化、构造和转换四个阶段。在一个应用程序的开发中，这些阶段顺序进行，但每个阶段包含一次或者多次迭代。在一次迭代中，每个模型元素按照从分析到部署的步骤行进。每个元素有相应的步速。虽然开发阶段和建模阶段不同步，但是二者之间有关联。在早期的开发阶段和每个阶段的早期迭代中，更强调早期建模阶段。

图13-93表示在后继阶段和迭代过程中的着重程度的对比。在初始阶段，主要着重于分

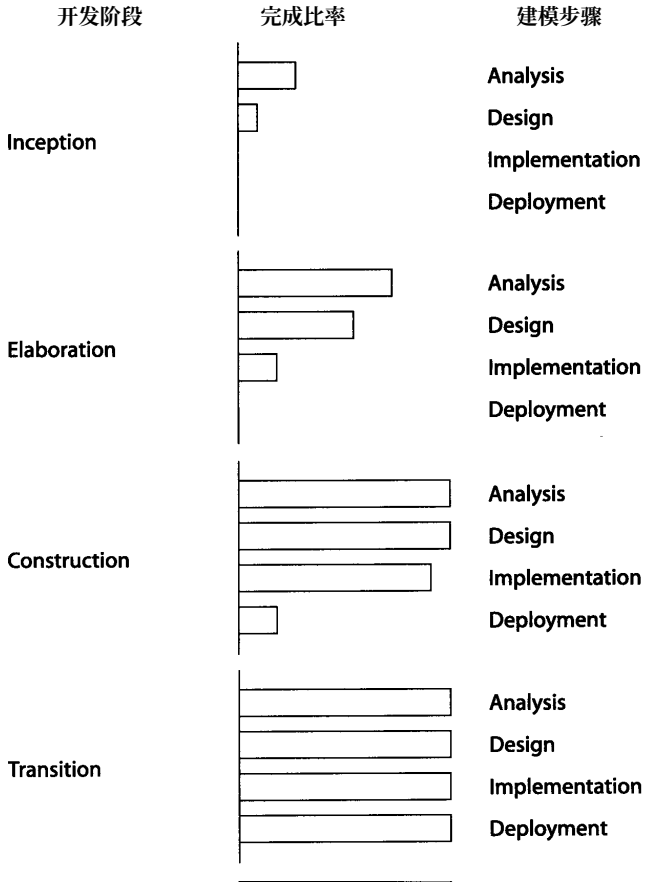


图13-93 开发阶段后的进展

China-pub.com 下载

析；在细化阶段建立面向设计和实现的元素过程模型；在构造和转变阶段完成所有元素。

115. diagram(图)

模型元素集的图形表示，通常为弧（关系）和顶点（其他模型元素）相互连接构成的。UML支持类图、对象图、用例图、顺序图、协作图、状态图、活动图、构件图和部署图。

见background information，font usage，hyperlink，keyword，label，package，path，presentation element，property list。

语义

图不是语义元素，而是表示有语义的模型元素。元素的意义不受表示法的影响。

图包含在包中。

表示法

多数UML的图和某些复杂符号由路径连接的图形构成。信息用拓扑结构表示，而不依赖于符号的大小、位置（有一些例外，如带有时间轴的顺序图）。有三种重要的可视关系：连接（通常是二维图形）、包容（包含二维符号的封闭图形）、可视的附件（图中“靠近”另一个符号的符号）。这些几何关系映射为表示法的分析形式中图中的节点的连接。

UML表示法画在二维表面上。有些图形是三维图形的二维投影，如管道。但是它们仍然被视为二维图标。在不久的将来三维布图可能会运用在桌面电脑上，但是目前仍然没有普及。

UML表示法中有4种图形结构：图标、二维符号、路径、字符串。

图标是大小、形状都固定的图形。它不能扩展也不含有内容。图标作为路径的端点或者独立符号可以出现在其他符号中。例如，聚合（菱形）、导航性（箭头）、终态（牛眼）、对象销毁（大X）符号都是图标。

二维符号有不同的大小，可以扩展并含有其他内容（如字符串或者其他符号）。其中大多数带有大小相同或者不同的分栏。结束于符号边界的路径表示与二维符号相连的路径。拖动或删除二维符号将影响其内容及与之相连的路径。例如，类（矩形）、状态（圆角矩形）、注解（带折角的矩形）的符号都是二维符号。

路径是一列线，或者终点相连的弯曲符号。在概念上，路径是一个单一的拓扑实体，但是它可以有多个片断。片断不能脱离路径而存在。路径的两端始终有其他图形符号（不能有悬空端）。路径可以有终点——在路径末端顺序排列的、对路径起限制作用的图标。例如，关联（实线）、泛化（带有三角形图标的实线）、依赖（虚线）的符号都是路径。

字符串以一种“未经分析”的形式表示不同的信息。 UML假定字符串的每个使用都可以通过相应的语法分析成为基本的模型信息。例如，属性、操作和转换都有各自的语法，这些语法可以被工具扩展。字符串可以是分栏的内容、列表中的元素（列表中的位置传达信息时）、

China-pub.com
下载

附属于符号或者路径的标签或者图中的独立元素。例如，类名、转换标签、多重性的表示和约束都是字符串。

116. direct class(直接类)

解说一个对象的最完整的类。

见class，generalzation，inheritance，multiple classification，multiple inheritance。

语义

一个对象可以是多个类的实例——如果它是某个类的实例，那么它同时是该类的祖先的实例。直接类是对对象最完整最详细的解说。对象是直接类的直接实例，是直接类的祖先的间接实例。对象不能是其直接类的后代的实例（根据定义）。

如果系统允许多重分类，则不存在完整地描述对象的单个直接类。对象可以是多个类的联合直接实例。只要这些类的后代不描述同一个对象，对象就是所有包含其描述的类的直接实例。换而言之对象的直接类之间不能有祖先关系。

如果类的实例中生成了对象，则该对象是此类的直接实例。

117. direct instance(直接实例)

直接实例是一个由给定的类进行最详细描述的实例（如对象）。表述为“对象O是类C的直接实例”，这里C是O的直接类。

见direct class。

118. discriminator(判别式)

从处于泛化关系中的一组子中选出一个子的伪属性。与其他使其父特化的隐含性质相比，所有的子表示了使其父特化的一个给定性质。它表示特化的维度。

见generalization，powertype，pseudoattribute。

语义

有时，模型元素可以基于不同的性质来特化。每个性质表示了一个特化的独立正交维度。例如，交通工具可以按照动力进行特化（汽油发动机、火箭引擎、风力、畜力、人力），也可以按照适用范围进行特化（陆地、水中、空中、外太空）。判别式是特化维度的名称。一个对象可以从多个维度特化，但是它们必须表现为具体实例。

泛化关系可以有一个判别式，它是一个表示用于划分某个父的子的维度的字符串。用同

China-pub.com 下载

一个判别式名对一个父进行特化而得到的所有特化关系构成了一个组，每个组是一个独立的特化维度。判别式名的完全集表示了特化父的维度完全集。实例必须同时是来自于每个特化组的子类的实例。例如，交通工具必须既有动力又有适用范围。

每个判别式表示了父的一个抽象性质，该性质通过具有与父有特化关系的子来特化。有多个判别式的父有多个维度，所有的维度都必须特化为具体元素。因此处于一个判别式组中的子是抽象的。它们中的每一个只是父的部分描述，着重于某个性质而忽视其他。例如，着重于动力的交通工具的子类忽略适用范围。具体元素要求同时对所有的维进行特化。这可通过对来自于每个维的子的具体元素的多重继承来实现，或通过对来自每个维的子的实例的多重分类来实现。在结合所有判别式之前，描述仍是抽象的。

例如，实际的交通工具必须同时有动力和适用范围。以风为动力的水上交通工具是帆船；畜力的空中交通工具没有名称，但是这种实例存在于童话中。

判别式无标签表示“空”判别式，“空”判别式仍被认为是合法的（“缺省”判别式）。这一约定使得常见的无判别式的情况也可以同样处理。如果所有的泛化路径都没有判别式，则所有的子都处于同一判别式。换而言之，所有特化都属于一个判别式，这与没有判别式的情况产生相同的语义。

结构

每个特化（泛化）弧都带有判别式字符串，它可以为空串。

判别式是父的伪属性，它在父的属性和关联角色中必须是唯一的。伪属性的域是子类集。在不同的子和父中允许多次出现同一个判别式名，表示这些子属于同一个部分。

表示法

判别式可用泛化箭头上的文本标签表示（如图 13-94）。如果带有同一判别式的两个泛化

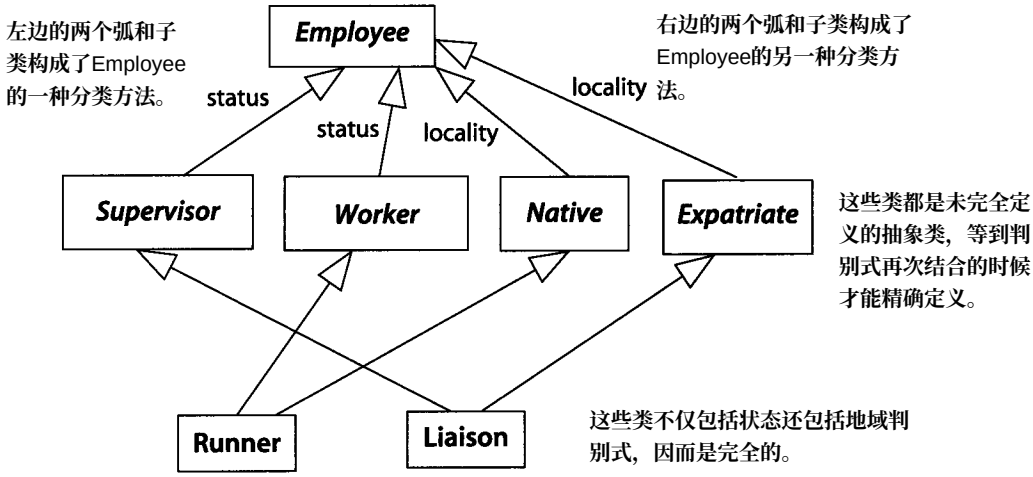


图13-94 判别式

China-pub.com
下载

弧共用一个箭头，则判别式可以标在箭头上。

示例

图13-94表示了在两个维度上对 Employee的一次特化：雇员的 status和locality。每个维度都有由子类表示的值的范围。但产生的可实例化子类应该在两个维度上都满足。例如，Liaison类既是Supervisor又是Expatriate。

在多重继承的子类中，在两个维度再次结合之前，单一维度的后代是抽象的。

119. disjoint substate(互斥子状态)

一个不能和同一组成状态中的其他子状态同时存在的子状态。对比： concurrent substate。见complex transition，composite state。

120. distribution unit(分布单元)

定位于一个操作系统进程或者处理器中成组存在的一些对象或构件，可以表现为运行时的组成或者聚集。它是部署视图中的一个设计概念。

121. dynamic classification(动态分类)

一种泛化的语义变更，其中的对象可以变换类型或者角色。对比： static classification。见multiple classification。

语义

在许多程序设计语言中，对象实例化以后不能改变生成它的类。这种静态分类限制简化了编译器的实现，但是在逻辑上却不是必要的。例如，在静态分类下作为圆的实现的对象只能是圆，而不能在x轴上拉伸。在动态分类下，它可以在一个轴上拉伸成为椭圆。

UML模型中可以使用两种分类。语义变更点即是一个例子。尽管区别对于执行很重要，不同的选择对模型的影响很小。同一个类必须以两种方式定义，但其支持的操作在两种情况下可以不同。

122. dynamic concurrency(动态并发性)

一种代表多个活动的并发执行的活动状态。

China-pub.com 下载

见activity graph。

123. dynamic view(动态视图)

模型中处理行为在不同时刻的说明与实现的一个方面，与静态视图中的静态结构有区别。动态视图是一个成组术语，包括用例视图、状态机视图、活动视图以及交互视图。

124. elaboration(细化)

软件开发过程中的第二阶段，在该阶段中，开始系统设计并开发并测试构架。在此期间，完成分析视图以及设计视图的架构部分。如果构造了可执行原型，则也完成部分实现视图。

见development process。

125. element(元素)

组成模型的原子。本书所说的是可以用于 UML模型的元素，即表达语义信息的模型元素以及用图形表示模型元素的表达元素。

见diagram，model element。

语义

元素的意义相当广泛，没有什么具体的语义。

所有元素可以带有下列特征：

标记值　　任何元素可以有 0个或者多个标记值。标记是说明数值意义的名称。 UML中的标记不是固定的，它可以携带任何对建模者或工具有意义的信息。

标准元素

documentation。

126. entry action(入口动作)

进入一个状态时执行的动作。

见exit action，run to completion，state machine，transition。

China-pub.com
下载

语义

状态可以选择带有入口动作。无论怎样进入该状态，在外部状态或者转换的动作之后，内部状态的动作进行之前，都会执行入口动作。入口动作不能以任何方式越过。无论是独立状态还是嵌套状态活动时，都要先执行入口动作。

■ 执行顺序。在两个具有入口动作和出口动作的状态之间的转换上，执行顺序为：执行源状态及其外部状态（不包括源状态和目标状态的外部状态）的出口动作，随后执行转换上的动作，最后从目标状态的外部状态开始（外层优先）一一执行入口动作，直到执行完目标状态的入口动作为止。图13-117的转换带有这样的动作。

表示法

入口动作按照内部转换的语法编码，带有哑事件名 entry（entry是保留字，不能用作实际事件的名称）。语法如下：

entry/*action-sequence*

每个状态有一个入口动作，该动作可以是一个序列，这样就不会失去一般性。

讨论

入口和出口动作在语义上不是必须的（入口动作可以从属于所有进入转换），但它们实现了状态的封装，从而使外部使用与内部结构分开。它们允许定义初始化和终态动作，而不必担心会被略过。对于异常尤其有用，因为它定义的动作即使出现异常时也会被执行。

入口动作有助于完成进入一个状态时必须完成的初始化工作。一种使用是对具有状态积累信息的变量进行初始化。例如，支持用小键盘输入账号或者电话号码的用户界面，在进入之前要清除上次的号码。重新设置错误数（如口令错误次数）也是一个例子。入口动作还用来给状态分派临时存储空间。

通常，出口动作与入口动作一起使用。入口动作分配资源，出口动作释放它们。即使出现外部转换，资源也会被释放。这是处理例外和用户错误的好办法。用户级的错误将触发高级转换，并跳过嵌套状态，但嵌套状态在失去控制之前还可以释放空间。

127. enumeration(枚举)

枚举是一种数据结构，它的实例构成了有名称的字面值。通常，同时声明枚举名和其字面值的名称。

见classifier，data type。

语义

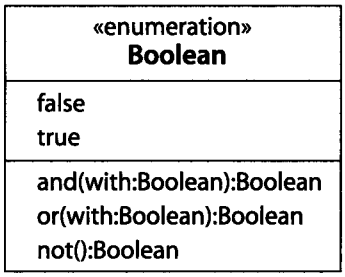
枚举是用户定义的数据类型，它包括名称和枚举字面名称的有序序列。每个字面名称是

China-pub.com 下载

枚举范围内的一个值 —— 是数据类型的一个预定义的实例。例如 RGB Color={red,green, blue}。Boolean数据类型是预定义的枚举类型，其字面值为 true和false。

表示法

枚举用一个矩形表示，在上方的分栏中，枚举名称上标有关键字《 enum》(如图13-95)。第二个分栏中是枚举字面值的列表。第三个分栏（如果有）是此类型上的操作。它们必须是可以查询的，因此不必再确切地声明。



枚举名

枚举值，按序

枚举操作

一元操作，返回另一个枚举值

图13-95 枚举的声明

128. event(事件)

占用一定时间和空间，值得注意的出现情况的说明。

见state machine，transition、trigger。

语义

在状态机中，一个事件的出现可以触发状态的转换。事件由参数表（可以为空）从事件的创造者向接收者传递信息。事件发生的时间是每个事件的隐含参数。其他参数在事件中定义。

事件的出现（实例）带有与各个事件参数对应的参量（实际值）。这些参量的值可以被附属于由事件触发的转换的动作引用。

有四种事件类型：

■ 调用事件。接收到一个请求，从而激活某个操作。希望的结果是事件的接收者触发一个转换，从而执行相应操作。事件的参数是操作的引用、操作的参数和返回引用。转换完成后，调用者收回控制权（若没有激发转换，则立即收回）。

■ 改变事件。由事件中某个表达式说明的布尔条件的满足，没有参数。这种事件隐含了对于条件的连续测试。当条件从假变为真时，事件将发生。然后，实际上条件被满足的时间

China-pub.com
下载

常常与其他事件的发生有关，所以通常不需要定时询问。

■ 信号事件。信号的接收，信号是对象之间进行通信用的特定的有名称的实体。信号有明确的参数列表。它由一个对象明确地发送给另一个或者一些对象。广播可以视为发送给所有对象的信号，但是实际上二者的效果可能不同。发送者在发送信号时明确了信号的参量，发送给对象的信号可能触发它们的零个或者一个转换。

信号是对象之间异步通信的手段。进行同步通信需要使用两个异步信号，每个通信方向上一个。

信号可以泛化，从父信号生成的子信号继承父信号的参数，还可以增加自己的参数。子信号满足其祖先要求的触发。

■ 时间事件。满足一个时间表达式，如到达绝对时间或对象进入某状态后经过一定的时间。注意，无论是时间段还是绝对时间，都可以用现实的时钟或者虚拟的内部时钟（不同的对象有不同的内部时钟）来定义。

表示法

见不同事件的表示法。

标准元素

create，destroy。

129. exception(异常)

由于行为失败引起的由基本执行机构产生的信号。

见composite state，signal。

语义

异常通常由于执行失败引起由基本实现机制隐式产生。它可以被视为发送给产生执行的主动对象或者过程激活的信号。对象的状态机可以对异常采取相应的动作。这包括忽略当前进程、跳转到执行中的某处、执行某个操作而不修改状态或者忽略事件。向高层状态转换的功能使异常处理灵活和强劲。

因为异常是信号，所以它带有参数表。参数值由发现错误的执行机构（如操作系统）设定。处理异常的操作可以读到这些参数。在大多数语言中，处理异常的操作可以控制它或将它转发。

表示法

构造型《exception》可用于区别声明和异常。构造型无需使用状态机中的事件名。

China-pub.com 下载

130. exit action(出口动作)

退出一个状态时执行的动作。

见entry action，run to completion，state machine，transition。

语义

状态可以选择带有出口动作。无论以什么方式退出该状态，在附属于内部状态或者转换的动作完成之后，在附属于外部状态的动作开始之前，都要执行出口动作。出口动作不能以任何方式跳过。在状态失去控制前一定要被执行。

入口和出口动作在语义上不是必须的（入口动作可以附属于所有输出转换），但它们实现了状态的封装，从而使外部使用与内部结构分开。它们允许定义初始化和终态动作，而不必担心会被略过。它们对于异常尤其有用，因为它们定义的动作既使出现异常时也会被执行。

表示法

出口动作按照内部转换的语法编码，带有哑事件名 exit（exit是保留字，不能用作实际事件的名称）。语法如下：

exit/*action-sequence*

每个状态只有一个出口动作，该动作可以是一个序列，这样就不会失去一般性。

讨论

出口动作可以在退出状态时完成必要的清理活动。它的最大作用是释放状态执行过程中分配的临时存储区和其他资源。

通常，出口动作与入口动作一起使用。入口动作分配资源，出口动作释放它们。即使异常发生，资源仍被释放。

131. export(示出)

在包的语境中，通过调整元素的可视性可使得元素在其命名空间之外可被访问。与访问和引入相反，示出是使外部元素可以在包内被访问。

见access，import，visibility。

语义

包通过调整元素的可视性水平，允许其他包看到该元素，从而实现示出（对于引入该元素的包是公共的，保护它自身的子）。

China-pub.com
下载

讨论

一个元素（如类）要看到另一个同一等级的元素必须满足两个条件。首先包含该元素的包应该将它设定为公共可见的，此外，引用目标元素的包必须能访问或者引入包含目标元素的包。两者都是必须的。

132. expression(表达式)

对可以用给定语言解释的语句进行编码的字符串。许多表达式在解释后产生某个值，其他执行特定动作。例如，表达式“（7+5*3）”计算出一个整数。

语义

表达式定义了一个语句，它可以用于实例集（可能为空）、值或者语境中执行的某个动作的性能。表达式不修改它进行计算的环境。表达式包含字符串和计算所用语言的名称。

表达式元素包含解释语言的名称和表达式的字符串（按照设计语言的语法编码）。假定该语言可以被解释。提供解释器是建模工具的任务。所用语言可以是约束 -说明型语言，如OCL；可以是程序设计语言，如C++，Smalltalk；或者是人类语言。当然，自然语言书写的表达式不能被工具自动计算，而必须由人来处理。

表达式的不同子类产生不同类型的值，有布尔表达式、对象集表达式、时间表达式、过程表达式。

UML模型中的表达式可以作为动作、约束、监护条件等等。

表示法

表达式是用指定语言表述的字符串。表达式的语法由工具或者语法分析器识别。假定分析器可以在运行时识别这些字符串并得到相应的值，或者识别语义结构从而了解表达式的意思。例如，类型表达式可以计算出类元引用，布尔表达式可以计算出 true或false值。建模工具应该知道所使用的语言，图中一般不会标明，因为表达式本身通常已经表明了自身的意义。

示例

```
self.cost<authorization.maxCost
forall(k in tagrets) {k.update()}
```

133. extend(扩展)

扩展是指扩展用例与基用例之间的关系，说明如何将扩展用例定义的行为插入基用例定

China-pub.com 下载

义的行为序列。扩展用例以模块化的方法递增地修改基用例。

见extension point，include，use case，use case generation。

语义

扩展关系将基用例和扩展用例连接起来。在此关系中，扩展用例不必为独立的可实例化的类元，它只是包含一个或几个片断，用来描述递增地修改基用例所需的行为序列。扩展用例中的每一个片断都可插入基用例的某个位置。扩展关系有一系列扩展点名称，其个数与扩展用例中的片断数相等。每个扩展点必须在基用例中定义，当用例实例执行到扩展点所指的基用例的位置，且满足所有扩展用例的控制条件时，实例就转入与扩展用例的片断相对应的行为序列中。当扩展片断执行完毕后，控制从转换点返回初始用例。

一个基用例可以有多个扩展用例，用例实例的生命期中可以经过多次扩展。如果多个扩展用例从基用例的同一个扩展点处展开，则它们的行为顺序是不可知的。一对扩展和基用例之间可以有多个扩展关系，只要扩展是从基用例的不同位置插入即可。扩展可以嵌套。

扩展关系中的扩展用例可以访问并修改基用例中定义的属性。但是，基用例看不到扩展用例，不能访问它的属性和操作。基用例定义了可添加扩展的模型框架，但是基用例看不到扩展用例。扩展可以显式地改动基用例的行为。请注意它与用例泛化的区别。在扩展中，扩展用例在基用例的一个实例中增加新的行为。在泛化中，子用例的实例增加父用例中没有的新行为，父用例不受子用例的影响。

一个扩展用例可以扩展多个基用例，一个基用例可以被多个扩展用例扩展。这并不表示这些基用例之间有特殊关系。

在扩展、包含或者泛化关系中，扩展用例可以作为自身的基用例。

结构（扩展用例）

扩展用例包括一列插入片断，每个片断是一个行为序列。

结构（基用例）

基用例定义了一些扩展点，每个点是基用例中一个或者一列可以插入行为的位置。

结构（扩展关系）

扩展关系有一系列扩展点名称，它们必须在基用例中出现。名称的数目必须与扩展用例中的片断数目相一致。

扩展关系可以有条件、基用例属性表达式或者事件的发生，如接收到信号。当用例实例的执行到达第一个扩展点引用的位置时，条件决定是否执行扩展用例。如果没有条件，则认为它始终为真。如果满足用例的条件，那么将执行扩展用例。如果扩展点引用了基用例中的多个位置，则可能在其中任何一个上执行扩展用例。

China-pub.com
下载

如果条件始终为真，扩展可以多次执行。扩展用例的每个片断执行相同的次数。如果必须限制执行次数，那可以使用相应的条件。

执行语义

当执行基用例的用例实例到达基用例中扩展关系引用的位置时，将计算扩展关系的条件。如果条件为真或者为空，则执行扩展用例。在许多情况下，条件包括某事件的发生或扩展用例片断所需的值的可用性，如参与者发出的开始扩展的信号。条件可能依赖于用例实例的状态，包括基用例的属性值。如果事件没有发生或者条件为假，则不开始用例的扩展片断。扩展片断执行完毕后，用例实例返回基用例，从刚才离开处继续向下执行。

如果满足条件，则立即执行扩展用例的其他插入。如果扩展点引用基用例的多个位置，则其中任何一个的条件都可能被满足。在序列中的任何位置，条件都可以为真。

如果一个扩展用例中有多个插入序列，那么只要条件在第一个扩展点被满足，所有扩展片断都将执行。不再为后续片断重新判断扩展条件，其中后续片断在用例实例到达基用例相应位置时插入。用例实例在两个扩展点之间继续执行基用例。一旦开始，就必须完成所有片断。

注意，通常用例是不确定的状态机（语法上），而不是可执行的过程。因为条件可能需要外部事件的出现。将用例实现为类的协作，需要转换到明确的控制机构中，正如语法的实现需要转换为一种更有效但更难理解的可执行形式。

注意，基和扩展相关的，一个扩展可以是更进一步扩展的基础。这并没有任何困难，上述规则仍然可以使用 —— 插入是嵌套的。例如，假设用例 B在执行扩展点 X处扩展了用例 A，用例C又在Y点扩展了用例B（如图13-96）。A的实例执行到扩展点 X后，开始执行用例 B。该实例到达 Y点后，开始执行用例 C。C执行完成后，返回用例 B，扩展B执行完后，再执行返回用例A。这与嵌套的过程调用或者其他嵌套结构类似。

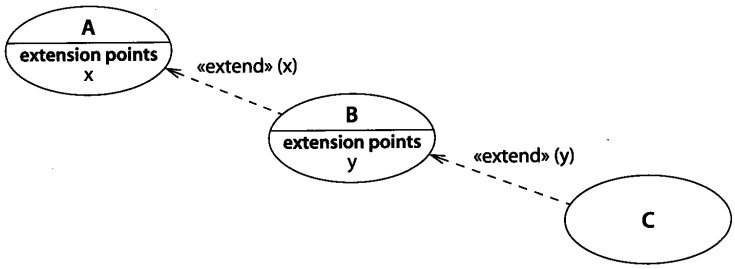


图13-96 嵌套扩展

表示法

从扩展用例向基用例引一个虚线箭头，指向基的箭头为树枝形，并带有关键字《 extend》。在关键字之后，可以在括号中列出扩展点名称。

图13-97是具有扩展关系的用例，图 13-98是用例的行为序列。

China-pub.com 下载

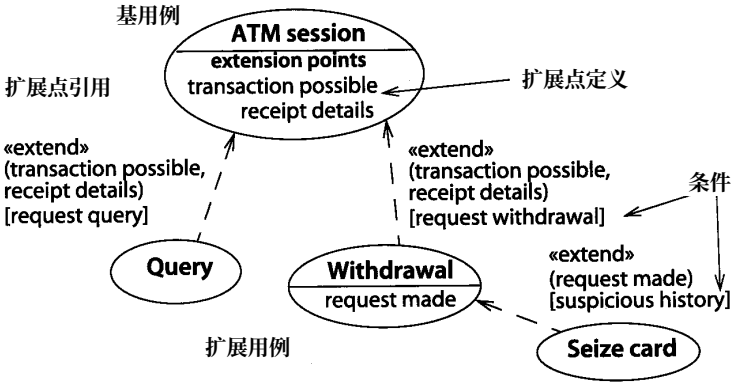


图13-97 嵌套关系

ATM会话的基用例：

show advertisement of the day
include (identify customer) 包含
include (validate account) 包含
（此处为扩展点引用） <--------- <transaction possible>
print receipt header
（另一个扩展点目标） <--------- <receipt details>
log out

查询扩展用例：

segment 第一个片断
receive request query
display query information

segment 第二个片断
print withdrawal information

提款扩展用例：

segment 第一个片断
receive request withdrawal
specify amount
（另一个扩展点目标） <--------- <request made>
segment 第二个片断
disburse cash

取卡扩展用例：

segment 唯一的片断
swallow the card
end the session

图13-98 用例的行为序列

讨论

扩展、包含、泛化关系都是向初始用例中增加行为。它们有许多共同点，但实际上区分

China-pub.com
下载

它们很方便。表13-1对于三种关系进行了比较。

表13-1 用例关系的比较

特 性	扩 展	包 含	泛 化
基行为	基用例	基用例	父用例
附加行为	扩展用例	包含用例	子用例
引用方向	扩展用例引用基用例	基用例引用包含用例	子用例引用父用例
基行为是否被附加行为修改	扩展用例隐式地修改基用例的行为。除非基用例没有扩展，否则扩展用例一旦存在，基用例的实例将会执行扩展用例	包含用例显式地修改基用例的效果。基用例可以有或者没有包含用例，但是基用例的实例执行包含用例	父用例的执行效果不受子用例的影响。要得到附加行为的效果，子用例，而不是父用例，必须实例化
附加行为可否实例化	扩展用例无需是可实例化的，可以是框架	包含用例无需是可实例化的，可以是框架	子用例无需是可执行的，可以是抽象的
附加行为可否访问基的属性	扩展用例可以访问并修改基用例的状态	包含用例可以访问基用例的状态。基用例必须为包含用例提供相应的属性	子用例可以访问并修改基用例的状态（通过常规继承机制）
基行为可否看到增加行为	基用例不能看到扩展用例，在没有扩展用例时必须是结构完整的	基用例可以看到包含用例，可以依赖于它的结果，但是不能访问它的属性	父用例不能看到子用例，在没有子用例时必须是结构完整的
重复	取决于条件	只重复一次	子控制自己的执行

134. extension point(扩展点)

引用用例行为序列中一个或一些位置的命名标记，在这些位置可以插入附加行为。扩展点的声明开启了用例扩展的可能性。插入片断是扩展用例（与基用例有扩展关联的用例）的行为序列。扩展关系包括扩展点名称列表，它们说明扩展关系的用例片断从何处插入其行为。

见extend，use case。

语义

扩展点有名称，而且引用了用例行为序列中的一个或者几个位置。扩展点可以引用用例的两个行为步骤之间的一个位置。另外，它可以引用一组离散位置。还可以引用行为序列中的一个区域（不超过序列两步之间的所有位置集）。

位置是用例的状态机描述中的一个状态，或者其他描述中的相应成分 —— 在语句列表中的两条语句之间或在交互的两条消息之间。

扩展关系包含可选的条件和扩展点列表，列表中扩展点的数目与扩展片断数目相等。用例实例执行到基用例的扩展点时，如果条件满足则执行相应的插入片断。

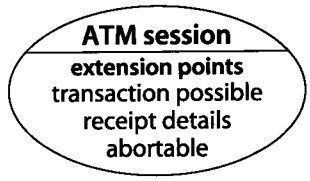
China-pub.com 下载

不用改变扩展点的身份标识就可以变换它的位置。使用命名扩展点，可以将扩展行为序列的说明和基用例的内部细节分开。基用例的修改或重排不会影响扩展用例。而且扩展点在用例中的移动也不影响关系和扩展用例。对于各类模块，这种独立性需要仔细选择扩展点，而且这种独立性不能在所有情况下都得到保证。

表示法

用例的扩展点可以列在名为 extension points的分栏内。

扩展点必须也引用用例行为序列中的一个或者几个位置。扩展点名称可用作行为序列中状态、语句、区域的标签。但是这并不意味着它存在于初始行为序列文本中。扩展点是允许将位置与执行插入的选择分开的名称。编辑工具可以将扩展点放到行为序列的覆盖层上，而不用修改初始文本；或者使用映射到语句上的扩展点名称的独立列表（使用语句标号、内部标签或者直接图形）。无论如何，最终效果是将扩展点定位在行为序列中的一步和多步之间。在图13-99的伪代码例中，尖括号中的扩展点名称引用了行为序列中的一个位置。根据说明序列的语言不同，可以用多种语法为扩展点定位。本例不是唯一的方法。



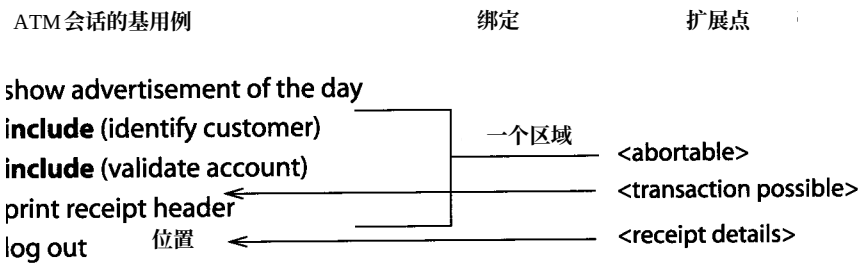


图13-99 扩展点声明

135. extent(外延)

由一个描述符描述的一组实例。有时也称为扩展。反义词： intent。

语义

一个有描述内容（内涵）和一些被描述的实例（外延）的描述符（如类或者关联）。内涵

China-pub.com
下载

说明构成外延的实例的特性。外延不一定有物理表示，运行时也不一定能得到。例如，建模者不应当假定可以得到作为类的实例的对象集。

136. feature(特征)

在类元（如接口、类或数据类型）中，作为表的一部分封装起来的特性，如操作和属性。

137. final state(终止状态)

组成状态中的一个特殊状态，当它处于活动时，说明组成状态已经执行完成。后组成状态进入终态后，将激活离开组成状态的转换，如果满足条件，转换将发生。

见activity，completion transition，destruction。

语义

为了实现封装，最好将组成状态的外部视图和内部结构尽可能分开。从外部看，组成状态是不透明的实体，隐藏了内部结构。从外部视图看，转换的起点和终点都是状态自身。从内部看，转换连接着状态中的子状态。初始状态和终止状态是支持状态封装的机制。

终止状态是一个特殊状态，它表明组成状态的活动已经完成，离开组成状态的完成转换可以开始了。终止状态不是伪状态。终止状态可能在一段时间内保持为活动的，这与初始状态不同。当等待组成状态中的其他并行子状态完成时，控制保留在终止状态中。

不允许有从终止状态出发的事件触发转换（否则这就是一般状态）。源于组成状态内的任意多个转换可以进入终止状态，但不能有源于外部的转换。进入终止状态的转换是一般转换，可以有触发器事件、监护条件和动作。

如果对象到达顶级终止状态，状态机将终止，对象将销毁。

表示法

终止状态用牛眼形图标 —— 小实心圆外套一个圆环表示。该符号位于组成状态内（如图

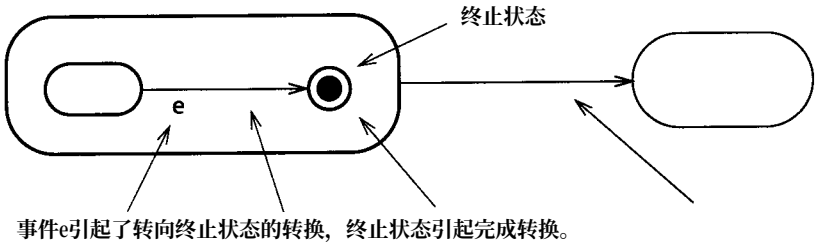


图13-100 终止状态

China-pub.com 下载

13-100)。一个组成状态只能有一个（直接）终止状态。然而嵌套组成状态中，可以有多个附加的终止状态。为了方便表示，状态中的终止状态符号可以重复出现，表示相同的终止状态。

138. fire(激发)

执行一个转换。

见run to, completion, trigger。

语义

当转换要求的事件发生时，如果满足监护条件，转换将执行其动作，且活动状态改变。

当对象接收到事件后，如果状态机处于运行至完成这一步骤，则保存该事件。当步骤完成后，由状态机处理这一事件。当对象处于包含转换的状态或该状态的嵌套子状态中时，若其事件被处理，则相应转换被触发。触发器事件是发生事件类型的祖先。如果是有多个源状态的复杂转换，则转换进行前所有源状态必须是活动的。源状态活动完成后，可以执行完成转换。如果源状态为组成状态，则所有直接子状态完成或者达到终止状态后可以执行完成转换。

处理事件后，计算监护条件（如果有）。如果条件的布尔表达式为真，则转换将激发。转换上的动作被执行，转换的目标状态成为活动的（但内部转换不改变状态）。在状态变化中，执行对象的初始状态和转换的目标状态最短路径上的任何出口动作和入口动作。注意，初始状态可能是转换源状态的嵌套子状态。

如果转换不满足监护条件，则不会激发，其他监护条件满足的转换可能激发。

如果有多个转换可以激发，则其中只有一个能真正激发。嵌套状态内的转换比外部转换优先。否则，转换的选择是不定的，这符合现实世界中的常见情况。

实际上，实现可以确定转换激发的顺序。这并不改变语义，合理设置监护条件以保证彼此互斥，可以达到相同的效果。更简单的方法是采用“如果没有进行其他转换，则使用本转换”。

• 延迟事件。如果某个事件或其祖先在某个状态中被定为延迟的，则它将不触发转换，直到对象进入一个不要求当前事件延迟的状态位置。对象进入新的状态后，原来延迟而现在不延迟的事件成为“有待解决的”，并以不确定的顺序出现。如果第一个待解决事件没有触发转换，则它将被忽略，下一个待解决事件出现。如果事件发生引起了状态的转换，则根据新状态的延迟情况，重新确定尚处于待解决的延迟事件，建立新的待解决事件集。

实现可以更严格地规定延迟事件的处理顺序，或者用某种操作来限制顺序。

139. flow(流)

在连续时间点上同一个对象两个版本之间的关系。

见become，copy。

China-pub.com
下载

语义

流关系将同一对象在不同连续时间点的两个版本联系在一起。它可以连接实例级交互中一个对象的两个值，或者描述符级交互中同一对象的两个类元角色。它表示对象从一个状态到另一个状态的转换，可以表示值、控制状态、位置的改变。

流依赖关系的构造型有成为和复制，用户可以增加其他构造型。

表示法

流关系用带有构造型的虚线箭头表示，构造型不能省略。

标准元素

become, copy。

140. focus of control(控制期)

顺序图中的一个符号，表示对象直接或者间接（通过子程序）执行一个活动的一段时间。

见activation。

141. font usuage(字体使用)

用不同字体或者其他图形标志来区分正文。

见graphic marker。

讨论

斜体字表示抽象类、属性、操作。其他字体用于强调或者区别表示语法中的不同部分。建议对于类元名称、关系名称用黑体字，对于辅助元素，如属性、操作、角色名等用正常字体。分栏的名称用特殊字体，但选择权在于编辑工具。工具可以选择字体来强调元素、区分保留字和关键字、对元素的特性编码，也可以允许用户选择字体。对于颜色有类似的考虑，有色盲的人可以不用颜色区分。上述所有内容是对于本书所述表示法的扩展。

142. fork(分叉)

在复杂转换中，一个源状态可以转入多个目标状态，使活动状态的数目增加。反义词：jojn。

China-pub.com 下载

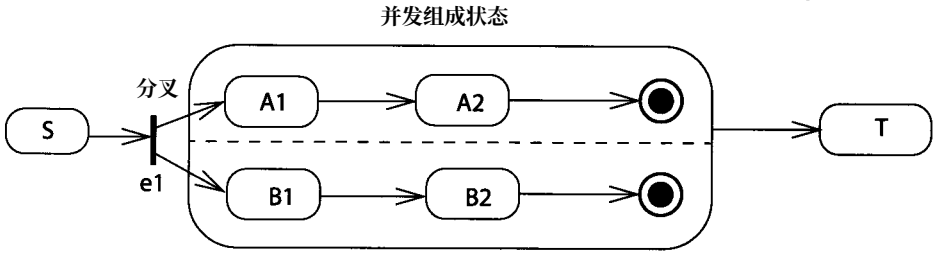
见complex transition，composite state，join。

语义

分叉是指有一个源状态和多个目标状态的转换。如果在源状态活动时出现触发器事件，则所有目标状态都将成为活动的。目标状态必须是并行组成状态中的不同区域。

表示法

分叉用具有一个转入箭头和多个转出箭头的粗线表示，它可以带有转换标签（监护条件、触发器事件和动作）。图13-101是进入并行组成状态的明显分叉。



当e1发生时A1和B1被激活。

图13-101 分叉

143. formal argument(形式参量)

见parameter。

144. framework (框架)

为某个域中的应用程序提供可扩展模板的泛化结构。

见package。

145. friend (友元)

一种使用依赖关系，允许客户访问提供者，即使客户没有足够的访问提供者的可见性。

见acess，import，visibility。

China-pub.com
下载

语义

友元依赖关系用于保证一个操作或者一个类对使用类的内容的许可，虽然从另外角度看来许可不够充分。它通常是规则中的异常，这种功能要小心使用。

表示法

从得到许可的类或者操作向提供内容的类引一条虚线表示友元依赖关系；构造型关键字《friend》标在箭头上。

146. full descriptor(完全描述符)

一个直接实例的完全隐含描述。完全描述符通过继承其所有祖先组装而成。

见direct class, inheritance, multiple class ification。

语义

实际上，对于类或者其他元素的声明只是其实例的部分描述，称为类片断。通常，类的对象含有比它的直接类的类片断所描述的更多的结构，其余的结构由祖先类继承而来。对实例所有的属性、操作和关联的描述称为完全描述符，它通常不由模型或程序表示。继承规则的目的是提供一种自动将类片断组成为完全描述符的方法。理论上有不同的实现方式，称为元对象协议。UML为继承定义的一系列规则，涵盖了大多数程序设计语言，也可用于建立概念模型。注意，可能存在其他可能性，如CLOS语言。

147. functional view(功能视图)

这种视图将系统分解为功能或者提供功能的操作。通常认为功能视图不是面向对象的，可能导致不易维护的结构。在传统开发方法中，数据流图是功能视图的核心。 UML不直接支持这种视图，但是活动图中有一些功能特征。

148. generalizable element(可泛化元素)

可以参与泛化关系的模型元素。

见 generalization，inheritance。

语义

可泛化元素可以有父和子。带有元素的变量可以带有该元素后代的实例。

China-pub.com 下载

可泛化元素包括类、用例、其他类元、关联、状态、事件和协作，它们继承其祖先的特征。每种可泛化元素的哪个部分是继承来的，取决于元素的种类。例如，类、继承属性、方法、操作、关联中的地位和约束。关联继承了参与类（本身可被特化）和关联端的特性；用例继承属性、操作、与参与者的关联、与其他用例的扩展和包含关系、行为序列；状态继承转换。

见generalization，association generalization，use case generalization。

结构

可泛化元素的特性表明它可以在泛化关系中的何处出现。

抽象 说明可泛化元素是描述直接实例还是抽象元素。true表示元素是抽象的（不能有直接实例），false表示它是具体的（可以有直接实例）。抽象元素必须具有具体后代才能使用。带有无方法操作的类是抽象的。

叶 说明可泛化元素是否可以特化。true表示元素不能有后代（叶），false表示可以有后代（无论当前是否有后代）。作为叶的抽象类只能起组织全局属性和操作的作用。

根 说明元素是否必须为无祖先的根。true表示元素必须为根，false表示不必为根，可以有祖先（无论当前是否有祖先）。

注意，声明叶和根类不影响语义，但这种声明可以给设计者提示。如果能避免对全局变量的分析或者对多态操作的全局保护，那么可以有更高的编译效率。

标准元素

leaf。

149. generalization(泛化)

一个较广泛元素和一个较特殊元素之间的类元关系。较特殊元素完整地包含了较广泛元素，并含有更多信息。较特殊元素的实例可以用于任何使用较广泛元素的地方。

见association generalization，inheritance，substitutability principle，use case generalization。

语义

泛化是两个同类的可泛化元素（如类、包或其他元素）之间的直接关系，其中一个元素被称为父，另一个为子。对类而言，父称为超类，子称为子类。父所说明的直接实例带有所有子的共同特性，子所说明的实例是上述实例的子集，不仅有父的特性，还有独有的特性。

China-pub.com
下载

泛化是一种可传递的反对称关系，一端为父，另一端为子。在父方向上经过一个或几个泛化关系的元素称为祖先，在子方向上经过一个或几个泛化关系的元素称为后代。不允许出现泛化环，一个类不能既是自己的祖先又是自己的后代。

在最简单的情况下，类（或者其他可泛化元素）有单一的父。在复杂情况下，子有多个父。子继承了父的所有结构、行为和约束，称为多重继承（或者多重泛化）。父元素对子元素有可见性。

关联、类元、状态、事件和协作都可以泛化。

关于关联的泛化的应用，见 association generaliazation。

关于用例的泛化的应用，见 use case generalization。

节点和构件类似于类，它们的泛化与类类似。

约束

约束可以应用于泛化关系集以及有同一个父的子。可以规定下列特性：

disjoint	一个祖先不能有两个子（多重继承时），实例不能同时成为两个子的间接实例（多重分类语义）。
overlapping	一个祖先可以有两个或者更多的子，实例可以属于两个或者更多的子。
complete	列出了所有可能的子，不能再增加。
incomplete	没有列出所有可能的子，有些已知子没有声明，还可以增加新的子。

表示法

类元之间的泛化关系用子元素（如子类）到父元素之间的实线路经表示，路径上指向较广泛元素的一端带有空心的三角形（如图13-102）。指向父的线可以被结合为一棵树（如图13-103）。

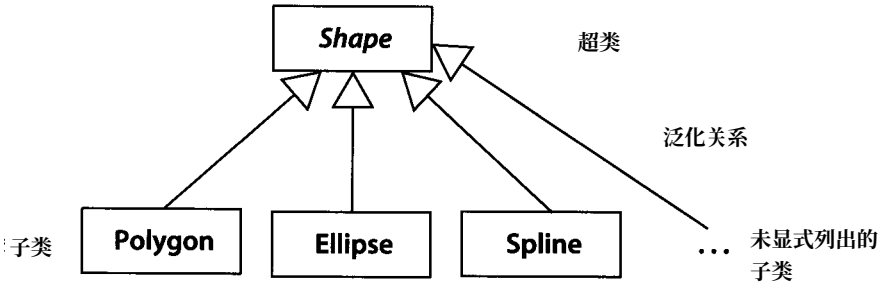


图13-102 泛化关系

泛化还可以应用于关联，但是太多的线条可能使图很乱。为了标明泛化箭头，可以将关联表示为关联类。

如果图中没有标出模型中存在的类，应该在相应位置上用省略号（...）代替（这不表示将来要加入新的类，而是说明当前已经有类存在。这种方法表示忽略的信息，而不是语义元素）。

China-pub.com 下载

一个类的子类表示为省略号说明当前至少有一个子类没有在视图中标处。省略号可以有描述符。这种表示法是由编辑工具自动维护的，不用手工输入。

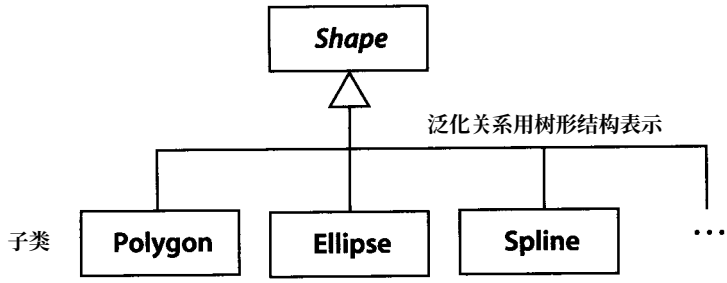


图13-103 树形泛化关系

表示选项

到一个给定超类的一组泛化路径可以用具有一段公用路经（包括三角形）的树表示，树枝指向子类。这只是一种表示法，不代表 n维关系。在基础模型中，每对超类-子类之间有一个泛化关系。弧是否分开来画没有语义上的区别。

如果在几个子类公用的泛化三角形上带有文本标签，则标签适用于所有路径。换而言之，所有子类共享给定的特性。

示例

图13-104是泛化关系上声明的约束。图中表示了“树形”泛化（路径为平行线，有公用

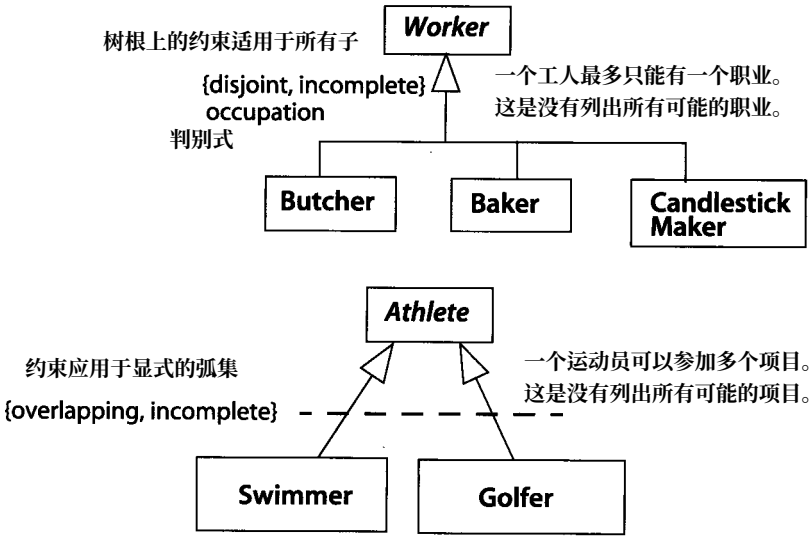


图13-104 泛化约束

China-pub.com
下载

的箭头），同时表示了“二元形”泛化（每一对父-子关系有独立的箭头）。

讨论

定义泛化关系中的父元素时不用了解子元素，但是子元素必须知道父元素的结构。但是，通常父元素是为了供子元素扩展而设计的，因此设计时应了解预期的子元素。泛化关系的优点之一在于经常可以出现设计父元素时没有想到的子元素，这带来了更强的功能。

实现关系与泛化关系类似，不过仅仅只有行为说明被继承，结果和实现并未被继承。如果说明的元素没有属性、关联，而只有抽象操作，则泛化和实现是相同的，仅有行为说明被继承。实现关系实际上不生成客户，因此操作必须在客户中，或者从其他元素处继承来。

标准元素

complete, disjoint, implementation, incomplete, overlapping。

150. graphic marker(图形标记)

一种标记元素，如几何图形、纹理、填充模式、字体、颜色等。

见font usage。

表示法

表示法的符号是由不同的图形标记构成的。图形标记本身没有语义作用，但是表示法的目的是尽可能多地以一致、正交的方式使用图形标记。

有些图形标记用于构造预定义的 UML符号，而其他一些不用于规范表示法。例如，颜色没有被赋予含义，因为有些打印机不支持彩色，而且有些人不能分辨色彩。这些未定义的图形标记可以按照建模者或者工具的需求供编辑工具使用。

UML表示法只允许有限的图形扩展。图标或图形标记（如纹理、颜色）可以与构造型联系起来。UML没有定义图形说明的格式，但是有很多位图和格式可供图形编辑器使用（虽然其可移植性很难满足）。

图标说明及其替代表示的一般形式易于理解，我们也允许工具开发者溶入自己的想象——注意，扩展性的过度使用可能会造成工具可移植性的减弱。

151. guard condition(监护条件)

进行相关转换之前必须满足的一个条件。

见branch，conditional thread，junction state，transition。

China-pub.com 下载

语义

监护条件是一个布尔表达式，它是转换说明的一部分。收到转换的触发器事件之后，系统保存该事件直到状态机完成当前的运行至完成步骤，随后进行事件处理，计算监护条件。如果条件为真，则可以进行转换（如果有多个转换可以进行，只能进行一个）。事件处理时进行测试。如果此时条件为假，除非触发器事件再次出现，否则不再重新计算监护条件。

监护条件必须是一个询问——它不能修改系统的值或者状态的值，又不能有副作用。

完成转换也可以有监护条件，此时，它选择一个分支执行。

表示法

监护条件是转换字符串的一部分，它是用方括号括住的布尔表达式：

[*boolean-expression*]

表达式中用到的名称必须是转换内部可见的，可以是触发器事件的参数或者当前对象的属性。

152. guillemets(书名号)

书名号（《》）在法语、意大利语和西班牙语中表示引用。在 UML表示法中表示关键字和构造型，如《bind》、《instanceOf》许多字体中都有该符号，必要时可以用两个尖括号（ << >>）代替。

见font usage。

153. history state(历史状态)

一种伪状态，说明内部组成状态在退出之后仍然记得它之前的活动子状态。

见composite state, pseudostate, state machine, transition。

语义

历史状态允许顺序组成状态记住从组成状态发出的转换之前组成状态的最后一个活动子状态。转向历史状态的一个转换将使前一个活动子状态再次成为活动的，并执行相应的入口动作和出口动作。历史状态可以有来自外部组成状态或者初始状态的转换。可以有一个没有标签的输出转换，该转换表示最初保存的历史状态。当没有保存的状态时，如果转换转向历史状态则可用到它。历史状态不能有来自组成状态内部的进入转换，因为它已经是活动的。

历史状态可以记忆浅历史和深历史。浅历史状态保存并激活与历史状态在同一个嵌套层次上的状态。如果一个转换从嵌套子状态直接退出组成状态，则将激活组成状态中顶级的子

China-pub.com
下载

状态。深历史状态记忆组成状态中更深的嵌套层次的状态。要记忆深状态，转换必须直接从深状态中转出。如果转换从深状态转换到一个浅状态，并由此转出组成状态，将被记忆的将是浅状态的转换。一个深历史状态的转换将恢复任何层次上之前活动的状态。在这个过程中，如果入口动作出现在包含所记住状态的内层状态上，则执行该入口动作。组成状态可能同时有深浅两种历史状态，进入的转换必须连到二者之一。

如果组成状态进入终态，则它将丢弃所有保存的历史状态，好像从没进入过此状态一样。

表示法

浅历史状态用带有H的小圆圈表示，如图13-105。深历史状态用带有H*的圆圈表示。

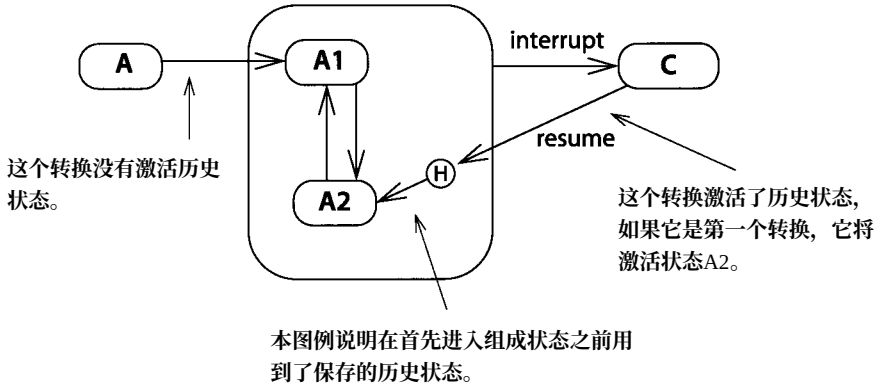


图13-105 历史状态

154. hyperlink(超级链接)

两个表示法元素之间可以通过某些命令穿越的不可见连接。

见diagram。

表示法

纸面上的表示法不包含隐藏信息。但是计算机屏幕上的表示法可以带有附加的不可见的超级链接，它们在静态视图中不出现，但是在动态视图上可以被激活，从而访问图形视图中或者文字表格中的信息。这种动态链接如同可见信息一样，是动态表示法的一部分。本文没有限制它们的使用，只是我们没把它们当成工具的必要功能。本文试图为 UML定义一种静态表示法，但是有些有用信息难以在这些视图中表示。另一方面，我们对动态工具的了解不够，又不想对创新的动态表示法只字不提。在不久的将来，动态表示法将变得成熟并成为标准，但是目前为时尚早。

China-pub.com 下载

155. identity(身份)

一个对象的继承特性，用于将它区别于其他对象。

见data value，object。

语义

对象彼此是不同的。对象的身份是它的概念句柄，是一个供其他对象引用或确定的继承特征。概念上，对象不需要用主键或其他机制来标识自身，且这种机制不应该存在于模型中。在实现中，身份由地址或者主键实现，但它们是基本实现结构的一部分，不用显示地列为大多数模型的属性。

156. ill formed(非良性结构)

结构设计不正确的模型，其中有一个或多个预定义的或模型说明的规则或约束被违反。反义词：well formed。

见conflict、constraint。

语义

违反良好结构规则和约束的模型是不能使用的，因为其语义是矛盾的。使用这种模型将产生无意义的结果。建模工具有责任找出非良性结构并加以隐藏，防止出现更大的问题。因为有时会使用UML内置语义之外的扩展结构，所以并不总能保证自动识别。此外，自动检查不能保证操作的一致性。因此，在实践中应该结合自动识别与人工识别。

完成的模型必须是良好结构的，但是模型建造的某个中间步骤生成的模型版本可能是非良性结构的，因为那只是未完成的框架。在将一个可用的模型修改为新模型的过程中，也可能生成非良性结构的模型。这类似于编写程序 —— 供编译的程序必须是可用的，但是文本编辑器的中间版本可以是不能用的。因此，支持工具必须可以编辑和保存非良性结构的模型。

157. implementation(实现)

1) 定义某事物是如何构造、计算的。例如，类是类型的实现，方法是操作的实现。对比：specification。实现和说明之间是实现关系。

见realization。

2) 用可执行的媒体（如程序设计语言、数据库、数字化硬件）描述系统功能的一个步骤。对实现而言，必须产生下层的决策以使设计适合具体的实现媒体，并与环境相适应（每种语言有各

China-pub.com
下载

自的限制)。如果设计得好，实现的决策将是局域性的，且任何决策不会影响系统的全局。这一步由实现层模型捕捉，特别是静态视图和代码。对比：analysis, design, implementation 和 deployment。

见development process，stages of modeling。

158. implementation class(实现类)

提供物理实现的类的构造型，包括属性、与其他类的关联和操作的方法。实现类用于具有静态单分类的传统面向对象语言。这样，系统中的一个对象必须有且只有一个实现类作为它的直接类。与类型不同，类的构造型允许多重分类。在某些语言中（如 Java)，对象可以有一个实现类和多个类型，实现类要与类型一致。

见type，其中对比了类型和实现类。

159. implementation inheritance(实现继承)

继承父元素实现——继承结构（如属性和操作）与代码（如方法)。相反，接口继承包括了对接口说明的继承（操作)，但是不继承方法和数据结构（属性和关联)。

UML中泛化的意义包括接口和实现的继承。如果仅仅继承实现（私有继承)，可以在泛化关系上使用关键字《implementation》。如果只要继承接口，可以使用到接口的实现关系。

见generalization，inheritance，interface inheritance，private inheritance。

160. implementation view(实现视图)

模型的一种视图，包含系统中构件的静态声明、依赖关系和可能由构件实现的类。

见component diagram。

161. import(引入)

许可依赖关系的一种构造型，其中提供者包中的元素名称被加入客户包的命名空间。

见access，package，visibility。

语义

提供者包的命名空间中的名称被加入客户包的命名空间，访问指定的可见性规则不变。如果引入的名称和原有命名空间的名称冲突，说明模型为非良性结构。

China-pub.com 下载

见access，其中有访问和引入的可见性规则。

表示法

用虚线箭头从得到访问权限的包指向提供者所在的包，箭头上带有构造型《 import》。

162. inactive(不活动的)

不活动的状态，不能由任何对象处理。

163. inception(初始)

软件开发过程的第一步，这期间构思并计算系统的初始方案，开发某些分析视图和少量其他视图。

见development process。

164. include(包含)

基用例与包含用例之间的关系。说明如何将包含用例中定义的行为插入基用例定义的行为中。基用例可以看到包含用例，并依赖于包含用例的执行结果。但是二者不能访问对方的属性。

见extend，use case，use case generalization。

语义

包含关系将基用例和包含用例连接在一起。关系中的包含用例不是能够独立实现的类元，而是显式地描述了插入执行基用例的用例实例中的附加行为序列。一个基用例可以有多重包含关系。同一个包含用例可以被包含于多个基用例中，这些基用例之间不必有任何关系。只要每个插入在基用例的不同位置，同一对基用例和包含用例之间可以有多个包含关系。

包含用例可以访问基用例的属性或者操作。包含用例提供了可被多个基用例重用的压缩行为。基用例可以看到为它设置属性值的包含用例，但不能访问包含用例的属性，因为当基用例重新得到控制后，包含用例已经结束了。

注意：（所有种类的）附加内容可以嵌套。一个包含可以是各种进一步包含、扩展或泛化关系的基。

结构

包含关系有下列特性：

China-pub.com
下载

位置　　基用例的行为序列体中的位置，在此位置处可以插入包含。当执行基用例实例达到该位置时，用例实例执行包含用例，随后继续执行基用例。包含是基用例行为序列中的显式语句，因此，位置是隐含的。这与扩展关系不同。

包含只执行一次，通过引用包含的基用例行为序列中的循环可以达到多次包含的效果。

表示法

包含用从基用例指向包含用例的虚线箭头表示，箭头上带有关键字《 include》(如图13-106)。位置可以作为特性标在箭头旁，但通常作为基用例的文本部分引用，在图中不表示。图13-107是用例行为序列。

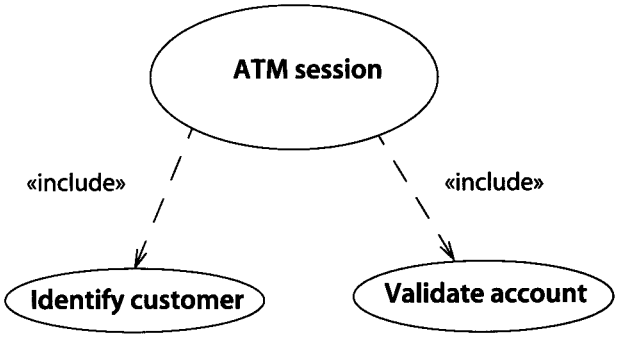


图13-106　包含关系

ATM 会话基用例:

show advertisement of the day	行为步骤
include identify customer	包含
include validate account	包含
print receipt header	行为步骤
log out	行为步骤

Identify Customer 包含用例:

get customer name
include verify identity
if verification failed **then** abort the session
obtain account numbers for the customer

Validate Account 包含用例:

establish connection with account database
obtain account status and limits

图13-107　用例的行为序列

China-pub.com 下载

165. incremental development(增量式开发)

系统模型和其他制品的开发形成一系列的版本，每个版本完成一定程度上的功能和细化，每个版本都比上一个版本增加了细节内容。这种方法的优点在于每个新的版本都是在上一个版本的基础上进行少量改动，不易出错。这种方式与迭代开发概念密切相关。

见development process。

166. indirect instance(间接实例)

一个实体，该实体是一个元素（如类）的实例，同时又是该元素子的实例。即它是一个实例，但又不是直接实例。

167. inheritance(继承)

一种使得一个较具体元素遵从一个较广泛元素定义的结构和行为的机制。

见full descriptor，generalization。

语义

继承使得可泛化元素的完整描述能通过组装泛化层次的声明段自动构造出来。泛化继承是一棵模型元素（如类）的声明构成的树（实际上是偏序树）。但每个声明都不是针对完整可用的元素的，而只说明该元素比它的祖先增加的内容。继承就是将这些递增的声明结合为描述实际实例的完全描述符的（隐含）过程。

可以认为每个可泛化元素都有两个描述符：一个片断描述符和一个完全描述符。片断描述符是模型中元素所声明的递增特征表，如类声明的属性和操作。元素和其父的片断描述符是不同的。模型中不会明确地出现完全描述符。它是元素实例的完全描述，如类的对象的所有属性和操作。完全描述符是元素和其父的片断描述符的联合体。

继承是对元素的递增定义，其他细节，如方法查找算法只是特定语言的实现机制，而不是核心定义的一部分。虽然这种解释看来有点奇怪，但它不需要多数其他定义中的实现过程，同时又与这些定义兼容。

冲突

如果继承的片断集中多次出现同一特征，则可能会出现冲突。继承集中的属性只能声明一遍。如果出现多次声明，则将导致冲突，且说明模型为非良性结构（这一限制不是基于逻辑推理，而是为了防止出现必须用路径名区分属性时可能产生的异义）。

操作可以被多次声明，只要声明相同（方法可以不同），或者子声明强化了某个继承声明（如将子声明为队列或增加它的并行状态数）。子声明的方法取代（重载）祖先的方法声明，

China-pub.com
下载

不会有冲突。如果从没有祖先关系的两个元素中继承到了不同的方法，则会出现冲突，且说明模型为非良性结构。

讨论

泛化是元素之间的类元关系，它解释了一个元素是什么，继承是组合共享递增声明，从而形成元素的完全描述的机制，这两者是不同的，但有着密切的关系。在泛化关系上应用继承机制，可以实现描述的分解和共享以及多态行为。这是 UML和多数面向对象语言使用的方式，程序设计语言还可能采用其他方法。

168. initial state(初始状态)

转换的目标状态是组成状态的边界时，用来指明其缺省起始位置的伪状态。

见composite state，creation，entry action，initialization，junction state。

语义

为了实现封装，应该尽可能地将组成状态的外部视图与内部细节分开。从外部看，组成状态是隐藏了内部结构的不透明实体，从外部视图看，转换的起止点都是状态自身。从内部视图看，它们连接到状态内的子状态。

初始状态是一个伪状态，它代表了进入转换与组成状态边界的连接点。它不是真实的状态，不能保留控制，它只是一种说明控制应该转向哪里的语法方式。初始状态必须要有一个输出的无触发转换（没有事件触发器的转换，因此遇到初始状态可自动有效）。完成转换连接到组成状态中的一个真实状态上。完成转换上可以有动作，该动作在状态的入口动作（如果有）之后遇到状态时执行。除了入口动作（在所有入口处执行，否则缺省）之外，它还允许动作与缺省入口关联。这个动作可以访问隐含的当前事件 —— 即触发最终使得转换转向初始状态的转换中的第一片断的事件。

初始状态不能有带事件触发器的输出转换，进入转换与进入其组成状态的转换是一样的，也应该避免，应该将这种转换与组成状态相连。

通常，初始状态上的转换是无监护条件的，因而必须是来自初始状态的唯一转换。多个输出转换可能带有监护条件，但这些条件必须涵盖所有的可能性（或者把某个监护条件设为else）。关键是控制必须马上离开初始状态。它不是真实的状态，必须激活某个转换。

类的顶级状态中的初始状态，代表了类的新的实例的创建。当执行它的输出转换时，隐含的当前事件是创建对象的事件，它具有由构造函数操作传来的参量值。这些值可供输出转换上的动作使用。

对象创建

一个类的大多数组成状态的初始状态都很类似，它可能包含具有构造型《 create》的触

China-pub.com 下载

发器，以及带参数的命名的事件触发器。可以有多个具有不同触发器的此类转换。每个触发器的特征标记必须与类的创建操作匹配。当类的新对象实例化时，触发与其创建操作相应的转换，该转换从调用创建操作处得到参量值。

表示法

初始状态用组成状态符号中的小实心圆，其上可以有输出转换箭头。一个组成状态中只能有一个初始状态（直接）。但是嵌套的组成状态中可以有多个组成状态。

示例

在图13-108中，我们从状态X开始，出现事件e之后，激活转换并执行动作a，转换指向状态Y，执行入口动作b之后，初始状态成为动作状态。随后立即进行输出转换，执行动作c并进入状态Z。

如果系统处于状态X时发生事件f，则激活另一个转换执行动作d，转换直接到达状态Z，这一过程中不包括初始状态，因为控制转交给Y，所以要执行动作b，但不执行动作c。

图13-109中的初始状态带有分支。假设系统开始于X状态，当事件e出现时，执行动作a，系统则进入状态Y并完成入口动作b，控制转给初始状态，测试当前对象的size属性。如果值为0，控制转入状态Z；如果值不为0，控制转入状态W。

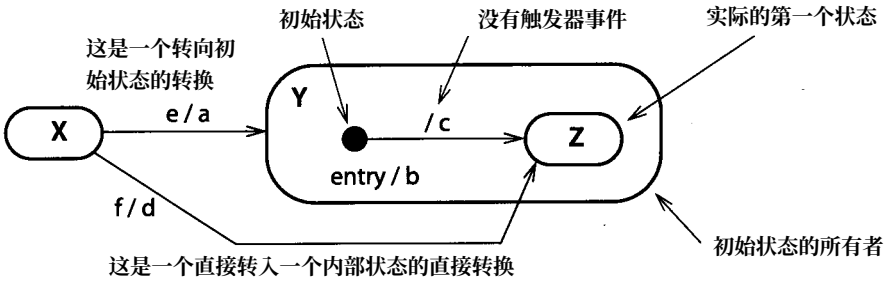


图13-108 初始状态

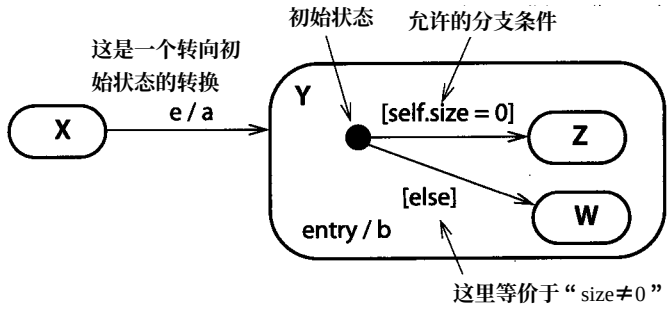


图13-109 带有分支的初始状态

China-pub.com
下载

169. initial value(初始值)

说明对象初始化之后某一属性值的表达式。

见default value，initialization。

语义

初始值是附属于属性的表达式。表达式是一个文本字符串，可以用某种语言进行解释。当对带有此属性的对象进行初始化时，用给定的语言和系统当前值计算该表达式。计算结果用于对新对象中的这一属性进行初始化。

初始值是可选的，如果没有，则属性声明没有指出新对象所带的值（模型的其他某个部分将提供这一值）。

注意，对象明确的初始化过程（如构造函数）可以忽略初始值表达式或者覆盖属性值。

类作用域属性的初始值只在执行开始时使用一次， UML没有规定不同类作用域属性初始化的相对顺序。

170. initialization(初始化)

为新创建的对象赋值 —— 即其属性、所属关系的链和控制状态。

语义

概念上，新对象的创建是一步完成的，但将其理解为两步会更容易理解：创建和初始化。首先给出空壳对象，它有适当的结构和属性位置，并赋予了身份，身份可以用不同的方法实现，如包含对象的存储块地址或整数计数器。总之，身份在系统中是唯一的，可以作为查找和访问对象的句柄。到此为止，对象仍然不是合法的 —— 它违反其值和关系上的约束。下一步是初始化，计算给出的初始值表达式，将结果加入相应的属性位置。创建方法可以显式计算属性值，从而重载缺省的初始值，结果值必须满足类的约束。创建方法还可以生成包含新对象的链，它们必须满足与类有关的关联多重性的要求。初始化完成后，对象成为合法的，并应符合其类的所有约束。初始化完成后，属性或其可变性特性为 frozen或add Only的关联不能再变化，直到对象被销毁为止。这个初始化的操作是原子的，不能被打断，也不能中途离开。

171. instance(实例)

具有身份和值的独立实体。描述符说明具有相似特性的实例集的形式和行为。实例的身份和值符合描述符的说明，模型中出现的实例主要是符合描述符模型的实例。

China-pub.com 下载

见descriptor，identity，link，object。

语义

实例具有身份。换而言之，在系统执行的不同时刻，即使实例的值改变了，该实例也可由不同时间点的同一实例来标识。任何时候，实例的值可以是数据值或对其他实例的引用。数据值是退化形式，其身份与其值相同，从另一个角度看也可以认为它没有身份。

除了身份和值以外，每个实例带有描述符，它用来约束实例可能取的值。描述符是描述实例的模型元素，这即是描述符-实例二元性，大多数 UML建模概念都有这样的双重特征，多数元素的主要内容就是各种描述符。模型的目的即用系统的实例和实例的值来说明系统可能的不同取值。

每种描述符描述一种实例。对象是类的实例，链是关联的实例；用例说明可能的用例实例，参数说明可能的变元值，如此类推。某些实例没有家族名，除非正式设置，往往被忽略，但这并不排除它们的存在。例如，状态描述在一条执行轨迹中状态的可能出现。

模型说明系统可能的取值以及执行过程中从一个值变到另一个值时系统的行为。系统的值是系统内所有实例以及它们各自值的集合。若系统中每个实例都是模型中的某个描述符的实例，并且实例集满足模型中所有显式和隐式约束，则系统值合法。

模型中的行为元素说明系统及其中的实例如何从一个值转变到另一个值。实例身份的概念对这种说明十分重要。每个行为步骤以其先前值说明了少数几个实例值的变化。系统中其余的实例值不变。例如，对象上的局部操作，可以用对象每个属性的新值的表达式来表示，同时系统其余部分不变。非局部性的函数可以被分解为几个对象上的局部函数。

注意，执行系统中的实例不是模型元素，通常，它们甚至不算是模型的组成部分。模型中的实例作为典型结构和行为的解释或例子出现，是系统取值或其历史的执行踪迹的简短捕捉，这对于人的理解是很有用的，但是它们通常不定义任何东西，只是指示许多可能的取值。

直接实例。每个对象都是某个类的直接实例，也是这个类的祖先的间接实例。对于其他可泛化元素的实例也是一样，如果一个类说明某个对象且此类的后代不再说明这一对象，该对象就是此类的直接实例。在多重分类中，一个实例可以是多个类元的直接实例，这些类元之间没有互为祖先关系。在某些执行语义下，由一个类元指明实现类，其他指明角色或类型。完整描述符描述了实例的所有属性、操作、关联和其他特性，它们可以从直接类元处得到，也可以从祖先处继承得到。在多重分类中，完整描述符是每个直接类元的特性的结合。

创建。见instantiation，其中描述了如何创建实例。

表示法

虽然描述符与实例是不同的，但是它们有许多共同的特点，包括相同的格式（因为描述符必须说明实例的格式）。因此，为描述符 —— 实例对选择表示法时应尽可能直接地体现二者的对应关系。方法不多，每种各有优缺点。在 UML中，将二者用同样的几何符号表示，并给

China-pub.com
下载

实例元素的名称字符串加下划线。

虽然图13-110表示的是对象，但下划线表示法一样适用于其他实例，如用例实例、构件实例和节点实例。

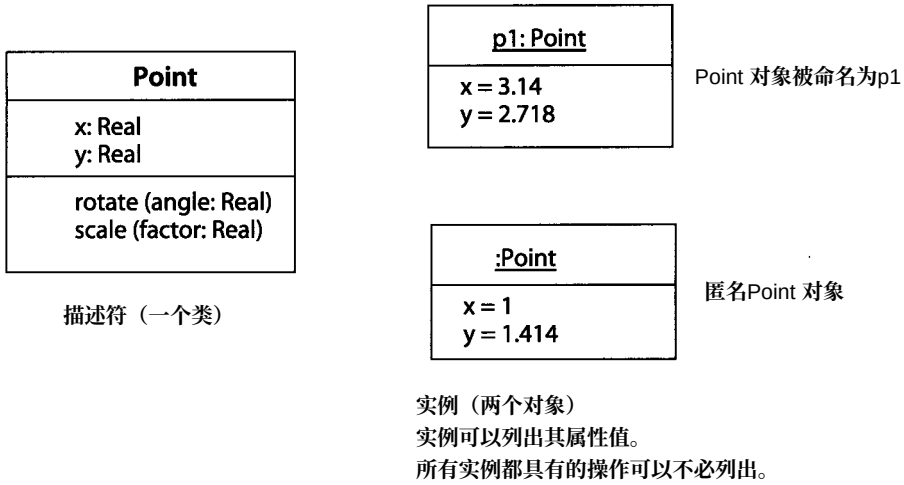


图13-110 描述符与实例

因为模型中的实例是作为例子出现的，通常仅包括与具体例子有关的细节。例如，不必列出完整的属性值表，如果关注的是其他东西（如对象间的消息流），甚至可以忽略整个属性表。

172. instance of(的实例)

实例及其描述符之间的关系。

见instance。

173. instantiable(可实例化的)

可以有实例，同义词：concrete。

见abstract，direct instance，generalizable element 。

语义

可泛化元素可以被声明为抽象的或是可实例化的。如果它们是可实例化的，则可以创建直接实例。

China-pub.com 下载

174. instantiate(实例化)

创建描述符的实例。

见instantiation。

175. instantiation(实例化)

模型元素新实例的创建。

见initialization。

语义

实例是在运行时由简单创建动作或创建操作创建的。首先为新的实例生成身份，接着按其描述符的说明生成数据结构，最后由描述符和创建操作者对它进行特性值的初始化。

实例化使用依赖关系表示了创建实例的类或包含此操作的类与实例化对象所属类之间的关系。

对象。当新对象被实例化（创建）时，被赋予身份和存储区，并被初始化。对象的初始化定义了其属性值、关联和控制状态。

通常，一个具体类有一个或多个类作用域的构造操作，其目的是创建该类的新对象，所有构造操作是一个隐含的简单操作，它创建一个新的未加工的实例，该实例随后由构造操作初始化。未加工的实例生成之后，即拥有其描述符所规定的格式，但其值未经初始化。因此，实例初始化之前不能被系统所使用。

链。同样，链由创建动作或操作创建，通常由附属于参与类的实例作用域操作创建，而不是关联元素自身的构造操作（虽然某些情况下这也是一种可能的实现技术）。同样存在一个创建对象之间新链的基本的隐含的简单操作。如果一组对象间已存在同类关联，则操作无效（因为关联的外延是一个不能包含相同值的集合）。对普通关联无须进一步操作，但是关联类的链还须对其属性进行初始化。

用例实例。用例实例化意味着创建用例实例，而且用例实例在用例控制的开始处开始执行。用例实例可能先暂时执行有扩展关系或包括关系的其他用例，而后再返回初始用例。当执行到相应用例末端后，用例实例将结束。

其他实例。其他描述符的实例也可以通过类似的两个步骤生成：首先创建未加工的实例，建立身份并分配数据结构；接着对新实例的值初始化，使之满足相应约束。例如，激活作为调用操作的直接结果被隐含地创建。

创建实例的确切机制是运行时环境的职责。

表示法

实例化依赖关系用虚线箭头表示，从执行实例化的类或操作指向被实例化的类，箭头上

China-pub.com
下载

标明构造型《instantiate》。

讨论

有时用实例化来表示将模板绑定，从而生成绑定元素，但是绑定关系更适合于这种情况。

176. intent(内涵)

描述符结构和行为特性的正式描述。对比：extent。

见descriptor。

语义

描述符（如类和关系）既有描述（内涵），又有其描述的实例集合（外延）。内涵的目的在于用可以执行的方式对实例的结构和行为进行说明。

177. interaction(交互)

说明对象或其他实例是如何传递消息的，交互在协作的语境中定义。

见interaction diagram。

语义

协作中的对象或其他实例为了完成某一任务（如执行一个操作）而通过消息交换来进行通信。消息可以包括信号、调用以及通过条件和时间事件进行的更隐含的交互。为完成一个特定目的而进行的消息交换模式称为交互。

结构

交互是一组对象之间为完成某一任务（如实现一个操作）而进行的一系列消息交换的行为说明。交互是协作加上协作中加于链上的消息流。为了说明交互，必须先说明协作——即定义交互的对象及其关系。而后说明可能的交互序列。可以用带有条件（分支或条件信号）的单描述符或者多重描述（每个描述符说明可能的执行路径中的一条），协作行为的完整描述可以用状态机实现，其状态就是操作或其他过程的执行状态。

表示法

交互用顺序图或协作图表示，两种图都能表示协作的执行。顺序图明确地表示了协作的

China-pub.com 下载

行为，包括消息的时间顺序以及方法激活的显式表示。但顺序图只表示参与的对象而不表示它们与其他对象的关系和属性。因此，它不能全面表示协作的语境。协作图表示交互的完整语境，包括对象及其关系，因此对设计更为适用。

178. interaction diagram(交互图)

应用于多种视图的着重于对象交互的表现方式。包括协作图和顺序图，与活动图密切相关。

见collaboration，interaction。

表示法

交互图中表示对象之间交互的模式。交互图在同一个信息基础上发展为不同的形式，各自有不同的侧重点。它们是：顺序图、协作图和活动图。

顺序图表示按时间排序的交互，着重表现参与交互对象的生命线和它们交换的信息。顺序图不表示对象之间的链。根据目的不同，顺序图有不同的形式。

顺序图有一般形式（说明所有可能的序列），还有实例形式（说明与一般形式一致的一个执行序列）。在没有分支与循环的情况下，两种形式是同构的，描述符是其实例的原型。

协作图表示执行操作的对象间的交互。它类似于对象图，表示了实现高层操作所需的对象和它们之间的链。

消息的时间顺序用消息流箭头上的序号表示。同步和异步信号都可以用相应的语法表示。顺序图中用箭头的几何顺序表示时间顺序，因此不用序号。有时在顺序图中使用序号是为了方便，或为了切换到协作图。

顺序图和协作图用不同的方式表示了类似的信息。顺序图表示消息的确切次序，更适合于实时说明和复杂的情形。协作图表示对象之间的关系，更适合于理解对象的全部作用和过程设计。

讨论

活动图表示执行高层操作中的过程步骤。它不是交互图，表示的是过程步骤之间的控制流而不是对象之间的控制流，侧重点在于过程中的步骤。它不表示到目标类的操作的任务分配。活动图构造了过程中执行状态的状态机。活动图中的许多特殊图标与 UML基本结构等效，只是受限于附加的约束，但它们的使用给过程带来了方便。

179. interaction view(交互视图)

表示对象之间为完成某任务而进行消息交换的视图，包括协作和交互，用协作图和交互

China-pub.com
下载

图表现。

180. interface(接口)

说明元素特有行为的命名操作集。

见classifier，realization。

语义

接口是类、构件或没有内部结构说明的其他实体（包括汇总单元，如包）的外部可见操作的描述符。每个接口仅描述实际类的行为的有限部分。一个类可以支持多个接口，每个接口效果上或互斥或重叠。接口没有实现，缺少属性、状态和关联，它只有接收的信号和操作，接口可以有泛化关系，子接口有其祖先的全部操作和信号，但可以有附加操作。接口与没有属性、方法只有抽象操作的抽象类等价。接口中的所有操作都是公共可见的（否则，不可能引用它们，因为接口没有所谓的“内部”来使用它们）。

下列扩展定义说明了接口的用途：

■ 接口是用于说明类或构件的某种服务的操作集合。

■ 接口用于为一组操作命名，并说明其特征标记和效用，接口着眼于服务的效果，而不是结构。接口不为其中的操作提供实现，接口的操作列表中还可以包括类准备处理的信号。

■ 接口用于说明服务者为其他模型元素提供的服务。接口为一组操作命名，这组操作共同实现系统或部分系统的部分行为。

■ 接口定义了类或构件提供的服务，它定义的服务由类或构件实现。因此接口跨过了系统的逻辑和物理的界限。一个或多个类（可能是某个构件子系统的部分）可以提供一个接口的逻辑实现。一个或多个构件可以提供符合同一接口的物理包。

■ 如果类实现一个接口，则它必须声明或继承接口的所有操作，它也可以有另外的操作（见realization）。如果一个类实现多个接口，则它必须包含所有接口的操作。多个接口中可以有相同的操作。如果特征标记匹配，那么一定是同一操作，否则会发生冲突，且模型为非良性结构（实现可以采用语言特定规则来匹配特征标记，如 C++中忽略参数名和返回类型）。接口不声明类的属性或关联，它们是类的实现的一部分。

■ 接口是可泛化元素，子接口继承祖先的全部操作并可以有新的操作。实现被视为行为继承。类继承其他类元的操作，而不是结构。一个类可以实现另一个类。起说明作用的类就像是接口，只有操作部分影响关系。

■ 接口参与关联。接口不能作为关联的出发点，但可以是关联的目标，只要该关联只对接口有导航性。

China-pub.com 下载

表示法

接口是一种类元，可以用带关键字《interface》的矩形表示。接口支持的操作列在操作分栏中。操作分栏中还可以有信号（带构造型《signal》）。信号也可列在独立分栏中。属性分栏可以省略，因为它总是为空的。

接口还可以表现为小圆圈，接口名称位于圆圈下方。圆圈符号用实线与支持接口的类或其他元素相连，它还可以连向高层的包容器，如包含类的包。这说明类支持接口类型的全部操作（可能更多）。圆圈表示法不表示接口支持的操作。操作列表用完整矩形表示法表示。虚线箭头将接口和使用其操作的类连接起来，箭头指向圆圈。虚线箭头表示类使用接口中声明的操作，但客户类并不需要接口中所有的操作。服务通常用效果测试来说明，如果提供者提供一列接口中的操作，则满足客户的需求。

实现关系用带实心三角形箭头的虚线表示（虚线泛化关系），箭头从类指向它支持的接口。这与通过实现类表示类型实现的表示法相同。事实上，这种符号可用于任意两个类元符号之间，表示客户（箭尾）支持提供者（箭头）定义的所有操作，但不必支持服务者的数据结构（属性和关联）

示例

图13-111是处理证券价格的商业构件的简化视图。FinancialPlanner是记录个人花费与投资的个人财政应用程序。它要有刷新证券价格的能力。MutualFundAnalyzer仔细检查公共基金，需要有刷新基金和基本证券价格的能力。刷新价格的能力用接口 UpdatePrices表示。有两个构件实现该接口，用连接到接口符号的实线表示。构件 ManualPriceEntry允许用户人工输入所选证券的价格。QuoteQuery通过调制解调器或因特网从报价服务器上得到证券的价格。

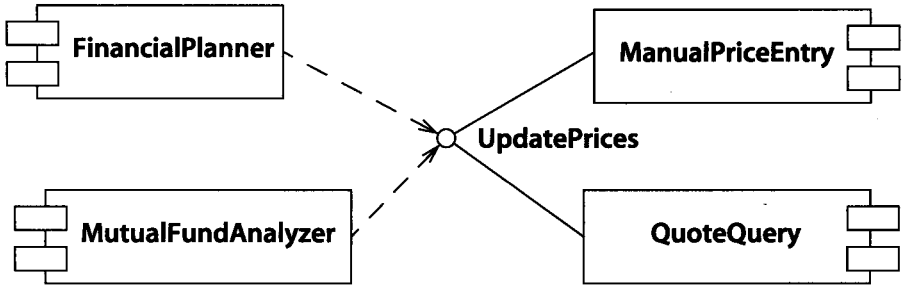


图13-111 接口的提供者和客户

图13-112表示作为类符号的关键字的接口的完全表示法。该接口有两个操作——询问证券的价格并获得值，提交证券价格表并接受改动的价格表。在此例中，QuoteQuery 构件使用实现箭头与接口相连，这与上图中的关系相同，只是更确切而已。

图中还有一个新的接口 PeriodicUpdatePrices，它是初始接口的子。它继承了两个原始操作并增加了一个操作，请求定期自动刷新价格。该接口由构件 QuoteServer实现。它也实

China-pub.com
下载

现了QuoteQuery中的两个操作，但方法不同。它不共享 QuoteQuery的实现（本例中），因此不继承它的现实。

图13-112表示了接口继承和完全继承的区别。后者包含前者，但子接口可以用与父接口不同的方法实现。 QuoteServer支持QuoteQuery实现的接口（即 UpdatePrices），但不能继承QuoteQuery的实现（通常，同时继承实现和接口更为便利，所以两个层次常常是等同的）。

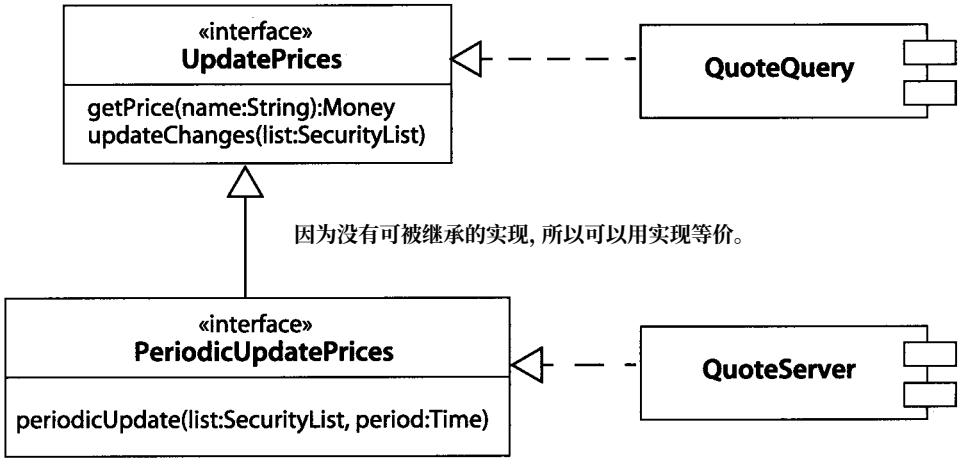


图13-112 完全接口表示法

接口可以包括它能处理的信号列表（如图 13-113）。

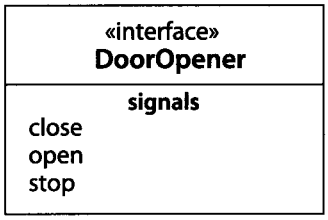


图13-113 带有信号的接口

接口用于定义类、构件的行为，而不限制其实现。这允许将接口继承与实现继承分开，如Java语言中的声明。

181. interface inheritance(接口继承)

继承父元素的接口而非其实现或数据结构。支持接口而不继承实现的设想通过用实现关系来建模实现。

注意，在UML中，泛化关系隐含了接口和实现继承。只有用私有继承才能保证只继承实

China-pub.com 下载

例而不继承接口。

见implementation inheritance，inheritance，private inheritance，realization。

182. interface specifier(接口说明符)

对关联类要求的行为的说明，这种行为用于满足关联的内涵。它包括对说明所需行为的接口、类或者其他类元的引用。

见association role，rolename，type。

语义

通常，关联类的功能不仅支持一个关联。例如，类可能参与其他关联，其行为是所有关联所要求的行为之和。应该更明确地说明一个类支持一个关联的功能。接口说明符是附属于关联端的一个类元，用于说明支持此关联所需的功能，而不考虑目标类中供其他关联使用的功能。

接口说明符不是必须的，多数情况下，一个类只参与一个关联，没有多余功能。如果省略接口说明符，则关联拥有对关联类全部功能的访问权。

说明符可以是类元集合，每个类元都说明目标类必须支持的状态行为。目标类必须满足所有要求，有可能比说明符要求的还多。

表示法

接口说明符语法如下：

rolename：$iname_{list}$。

*iname*是代替角色名的接口或其他类元的名称。如果有多个说明符，则它们的名称用逗号隔开。

示例

在图13-114中，Server类将请求存入Array类中，为此，它仅需要Queue类的功能，如不

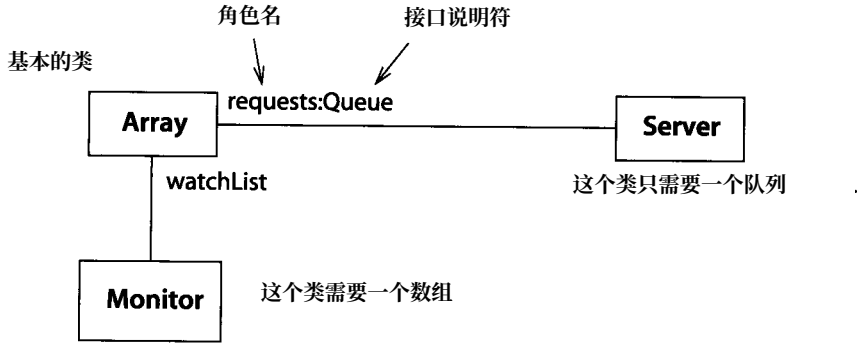


图13-114 接口说明

China-pub.com
下载

随机访问信息。实际的 Array类满足接口说明符 Queue（一个包括队列功能的数组）的需求。Monitor用Array来表示请求的状态，它使用 Array的全部功能。

讨论

使用角色名和接口说明符等于创建一个小型协作，其中仅包含一个关联角色和两个类元角色，其结构由初始关联的角色名和角色类元定义，因此协作使用关联和类。初始类必须符合接口说明符（可以是接口和类型）

183. internal transition(内部转换)

附属于状态的有动作的转换，但它不会引起状态改变。

见state machine。

语义

内部转换允许引起动作的事件同时不改变状态，它有源状态而无目标状态，一旦被激活，转换上的动作将被执行，但状态不会改变。因此，不执行入口或出口动作。从这方面来看，它与自转换不同，自转换会导致退出嵌套状态，并引发出口或入口动作。

表示法

内部转换用文本条目列在状态的内部转换分栏中，条目与外部转换的文本标签语义相同。因为没有目标状态，故不需要使用箭头。其语法如下：

event-name / action-expression

entry、exit、do和include是保留字，不能作为事件名，它们用来声明入口动作、出口动作、活动执行或子机器的执行。这些特殊动作使用内部转换语法是为了更明显。它们不是内部转换，这些保留字也不是事件名。

图13-115给出了表示法。

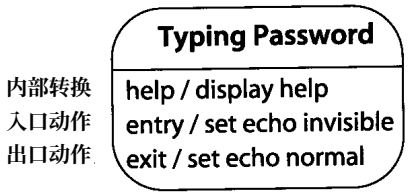


图13-115 内部转换语法

China-pub.com 下载

讨论

内部转换可被视为一种“中断”，它引发一个动作但不改变当前状态，因此不激活出口或入口动作。最好的办法是通过将一个内部转换连接到一个组成状态来对动作建模，该动作必须在一些状态中出现，但不得改变状态的活动状态——例如，表示帮助消息或计算事件发生次数。不宜于对夭折或异常建模，它们应该用转入新状态的转换来建模，因为它们的出现与当前状态不符。

184. invariant(不变量)

一个必须永远为真的约束（或者至少在所有操作完成时为真）。

语义

不变量是一个必须在无活动操作的任何时刻都为真的布尔表达式。它是一种断言而不是可执行的语句。根据表达式形式的不同，它可以或不可以被事先确定。

见precondition，postcondition。

结构

不变量被作为约束建模，带有构造型《invariant》。

表示法

后继条件可以表示为带《invariant》关键字的注解。注解附在类元、属性或其他元素上。

185. iteration expression(迭代表达式)

产生迭代用例集的表达式。每个迭代用例指定迭代中一个动作的执行。迭代用例可以包含对迭代变量的赋值。每个迭代用例只执行一次动作。

见message。

语义

迭代表达式表示条件的或迭代的执行。根据包括的条件，表示零个或多个消息。选择有：

*[*iteration-clause*]　迭代

[*condition-clause*]　分支

迭代表示消息序列，*iteration-clause*表示迭代变量和测试细节，但可以省略（在未确定迭

China-pub.com
下载

代条件的情况下）。*iteration-clause*用伪代码或实际的程序设计语言表达。UML没有说明其格式。可以是如下形式：

*[i:=1..n]

条件表示的消息的执行与否取决于*condition-clause*是否为真。*condition-clause*用伪代码或程序设计语言表达。UML没有说明其格式，可以是如下形式：

[x > y]

注意，分支与不带星号的迭代表示法相同，可以将它视为只发生一次的迭代。

迭代表示法假设迭代中的消息按顺序执行，它们也可能并行执行，其表示法为星号加双竖线。例如：

*[i:=1..n]|| q[i].calculateScore()

注意在嵌套控制结构中，迭代表达式在顺序号的内部层次中不迭代，每层在其语境中定义自己的迭代。

186. iterative development(迭代开发)

包括一系列步骤的系统开发，或称为迭代。每个迭代都比前一个更接近所希望的系统。每一步的结果必须是可执行的系统，以供执行、测试和调试。迭代开发类似于增量式开发。在迭代增量式开发中，每轮迭代在上一轮的基础上增加了新的功能。功能增加的顺序要考虑到迭代大小的平衡和尽早发现潜在的风险。

见development process。

187. join(结合)

状态机、活动图或顺序图中的一个位置，在此处有两个或更多并发线程或状态归结为一个线程或状态。反义词：fork。

见complex transition，composite state。

语义

结合是有多个源状态和一个目标状态的转换。如果所有源状态都处于活动且发生触发器事件，则执行转换，目标状态为活动的。源状态必须在并发组成状态的不同区域中。

表示法

结合用具有多个进入箭头和一个输出箭头的粗线表示。它可带有转换标签（监护条件、触发器事件和动作），图13-116是并发组成状态中的多个状态的一个显式结合。

China-pub.com 下载

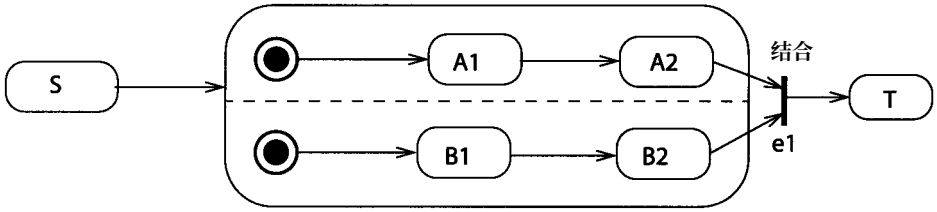


图13-116 结合

讨论

见merge。

188. junction state(结合状态)

状态机中作为一个综合转换一部分的伪状态。它在转换执行中不打断运行至完成步骤。

见branch，merge。

语义

状态机中的转换可以从源状态跨过几个组成状态的边界到达目标状态。执行这种转换时会激活一个或多个出口或入口动作。某些情况下，有必要将转换上的一个或多个动作与附属于嵌套状态的出口或入口动作交汇。对只有单一动作的简单转换而言，这是不可能的。

使多个触发器有单一输出或允许一个触发器有多个带监护条件的输出将会十分便利。

结合状态作为一种伪状态，使得从多个转换段中建立一个综合转换成为可能。结合状态可以有一个或多个进入段和一个或多个输出段。它不能有内部活动、子机器，或者具有触发器事件的输出转换。它是构建转换用的哑状态，因而不能处于“活动”的状态。

结合状态可以构建源自多个片断的转换。结合状态链中只有第一个段可以有触发器事件，但每个段都可以有监护条件。其后续段必须无触发。有效的监护条件是所有单个监护条件的结合体。除非所有条件被满足，否则转换不会开始。换而言之，状态机不会停留在结合状态。

如果多个转换进入一个结合状态，则它们各自可以有不同的触发，也可无触发。通过结合状态集的每条路径代表了一个不同的转换。

输出转换也可以有监护条件。如果有多个输出转换，则每个必须有独自的监护条件，这就是分支。

输出转换可以附带动作（结合状态可以有一个内部动作，但它等价于将一个动作连到输出转换）。一旦所有监护条件满足，动作将被执行。转换不能部分激发而停在结合状态，必须到达某个常规状态。

China-pub.com
下载

当进入转换激活后，输出转换会立即被激活，随即执行附带的动作。进入和输出的转换是一个原子步骤（运行至结束步骤）的部分 —— 它不能被任何动作或事件中断。

表示法

状态机中用小圆圈表示结合状态。它没有名称，可以具有进入和输出转换箭头。

示例

图13-117是状态S到T的两个完整转换 —— 由事件f触发的单片断转换和事件 e触发的多片断转换，事件e中使用了两个结合状态。图中还表示了转换动作与入口或出口动作的交汇。

注意，转换线上的动作标签的位置没有意义。如果动作 d位于状态X中，那么它将在X退出Y进入之前执行。因此，d应位于转换最外面的位置。

后文图13-179和图13-184给出了其他例子。

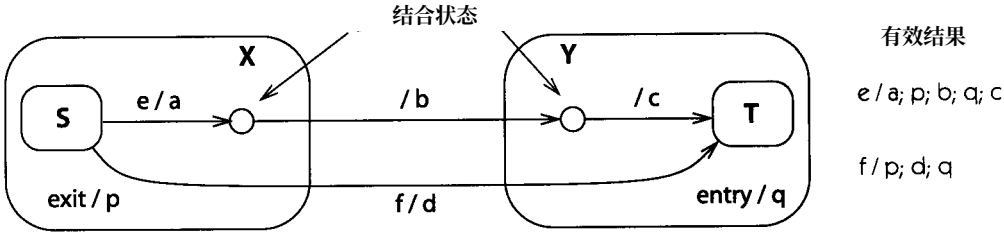


图13-117 结合状态

见control icon, 其中有状态图和活动图中可能出现的其他符号

189. keyword(关键字)

关键字是为没有独立语法的模型元素分类用的文本符号。

见graphic marker，stereotype。

表示法

关键字用于没有单独表示法的内置模型元素和用户定义的构造型。通常将关键字写在书名号内。

《*keyword*》

如果关键字是某个平面符号（如类矩形）的一部分，则关键字位于该符号边界内。

本文中说明了某些预定义关键字，并用保留字的方式表示。其他可供用户作为构造型使用。不能使用与预定义关键字相同的构造型。

China-pub.com 下载

讨论

容易区别的符号是有限的，因此UML表示法用文本关键字来区别同一主题下的不同变体，包括基类的元模型子类、元模型基类的构造型，以及表元素组。对用户而言，元模型子类和构造型间的元模型区别并不重要，但它对工具制造者和元模型实现者而言是重要的。

190. label(标签)

在图中使用字符串的一种方法，仅仅是一个用于表示法的术语。

见diagram。

表示法

标签是图中在逻辑上附属于另一个符号的图形字符串。通常将字符串标在符号旁或封闭区域内来表示这种附属关系。对某些符号而言，标签位置是确定的（如在线下）。而对多数符号而言，标签必须靠近线或目标。编程工具可以维持标签和图形符号之间的内部联系，这样即使标签和图形符号分开表示，二者在逻辑上也是相连的。在最终视图中必须都表示清楚，以保证符号和标签连接关系不会产生混淆。虽然图中并不一定能完全清楚地表示附属关系，但这种关系在图的结构层次上是清晰无异议的。工具可以用其他方法表示标签与符号之间的附属关系（如着色线或匹配元素闪烁）。

191. language type(语言类型)

用编程语言的语法定义的匿名数据类型。

见data type。

语义

语言类型是一种由编程语言解释成数据类型的表达式。它可以为属性、变量、参数的类型。它没有名称，不能用于声明新的数据类型。

例如，C++数据类型“ Person*(*)(Contract*,int) ”可被定义为C++语言类型。

语言类型的目的在于编程语言中的实现。更多的逻辑关系应该用关联实现。

192. layer(层)

将模型中同一抽象层次的包分组的一种构架模式，每个层在某个实现层次上表示一个虚

China-pub.com
下载

拟世界。

193. leaf(叶)

在泛化层次结构中，没有子的可泛化元素，它必须是供使用的具体（完全实现的）实体。

见abstract，concrete。

语义

叶特性声明一个元素必须为叶。如果声明了叶的子，则说明模型为非良性结构。其目的是保证类不被修改，因为类的行为必须被可靠地建立。叶声明还允许将系统的各个部分分开编译，这通过确保方法不被重载以及促进方法代码的嵌入来实现。特性为假的元素可能实际上是叶，但它可以在将来修改模型时生成子。视为叶或者被限定为叶不是基本语义特性。

194. lifeline(生命线)

顺序图中的虚线，用来表示对象在一段时间内存在，此线与时间轴平行。

见sequence diagram。

语义

生命线表现了对象存在的时段。对象在拥有控制线程时激活 —— 即当它作为线程的根时。被动对象在控制线程通过时暂时存在 —— 即当存在过程调用时。后者称为激活，它包括过程调用下层过程的时间。

表示法

对象或类元角色在顺序图中用垂直虚线表示，称为生命线。生命线表示对象在特定时间段内存在。

生命线间的箭头代表对象之间的消息。箭头表示对象接收信息，由一个操作完成；箭尾表示对象发送信息，由一个操作激活。生命线之间箭头排列的几何顺序，代表了消息的时间顺序。

如果对象在顺序图表示的时间段内创建或销毁，则其生命线就在相应时间点开始或终止。否则，生命线贯穿图的始终。对象符号位于生命线上。如果对象在图中被创建，则对象符号位于创建它的消息上，否则，对象符号位于任何消息箭头上。如果对象在图中被销毁，则用大X表示销毁，X或者标在导致销毁的消息的箭头上，或者标在被销毁对象的最终返回消息上（在自毁的情况下）。转换开始时存在的对象位于图的顶部（第一个箭头上方）。转换结束后仍

China-pub.com 下载

存在的对象，生命线越过最后一个箭头。

生命线可分为两条或是更多，以表示条件控制。每条踪迹对应消息流的一个条件分支。生命线也可在某一点结合，见后文图 13-162。这种表示法易引起混淆，应小心使用。

对象永久或暂时活动的时期用一条实线表示生命线。第二条双线表示迭代。详见 activation。因为主动对象总是活动的，有时省略双线。

生命线被状态符号中断表示状态转换，这与合作图中的成为转换相应。指向状态的箭头表示引起状态变化的消息。后文图 13-163给出了一个例子。

195. link(链)

对象引用元组，是关联或关联角色的实例。

语义

链是两个或多个对象之间的独立连接，它是对象引用元组（有序表），是关联的实例。对象必须是关联中相应位置处类的直接或间接实例。关联不能有来自同一关联的重复链，即两个相同的对象引用元组。

作为关联类实例的链除了对象引用元组外还可以有属性值表。不允许具有相同元组对象引用的重复链，无论其上的属性是否不同。链的身份来自对象引用元组，它必须是唯一的。

链可以用于导航。换而言之，链一端的对象可以得到另一端的对象，然后可以发送消息（称通过联系发送消息）。如果关联对目标方向有导航性，则这一过程就是有效的。如果关联是不可导航的，访问可能有效或无效，但消息发送通常是无效的，相反方向的导航性另外定义。

在协作中，关联角色是与语境有关的暂时的类元之间的关系，关系角色的实例也是链，其寿命受限于协作的长短。

表示法

二元链用对象间的路径 —— 一个或多个相连的线或弧。在自反关联中，路径是两端指向同一对象的回路。

见association。

角色名位于链的端点，关联名位于路径旁。表示实例的名称有下划线。链没有实例名，它的身份来自相关的对象，因为实例没有多重性，链也没有多重性。多重性是说明可以存在的实例数目的描述符特性。链角色上可以表示其他关联符号（聚合、组合、导航）。

链上可以表示限定符，限定符的值可以表示在其下的方格中，图 13-118有普通和限定词链。

链还可以表示关联的特性，如导航方向、聚合、组合、实现构造型和可见性。

China-pub.com
下载

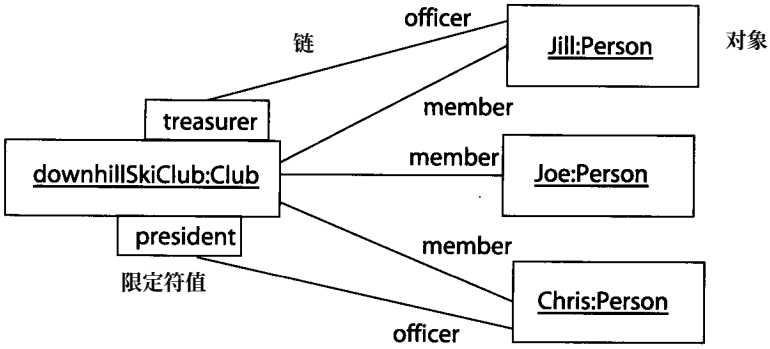


图13-118 链

n 元链。n 元链用菱形表示，引出到每个参与对象的路径，其他表示与二元链相同。

讨论

对象图如何表示依赖关系？通常，依赖关系表示类而不是对象之间的关系，并在类图，而不在对象图中表示。过程参量、局部变量和操作调用应作为实际的数据类型出现而不仅仅是依赖关系。因此，它们可以表示为链，过程的调用者要求对目标对象的引用——这就是链。有些链是协作中关联角色的实例，如大多数参数和局部变量。记住：依赖与类相关，而不是与各个对象相关。

196. link end(链端点)

关联端的实例。

197. list(列表)

由另一个模型元素拥有且嵌套其中的长度可变的有序模型元素序列。

见classifier，state。

语义

类元包含多个附属元素列表，包括属性、操作和方法。状态包含内部转换列表。其他元素包含其他元素列表。每种列表独自说明。除了属性和操作列表外，可用其他列表说明预定义或用户定义的值，如职责、规则或修改历史。 UML没有定义这些可选列表，由用户定义的列表操作取决于工具。

China-pub.com 下载

嵌入列表和其中的元素要么属于包容类，要么属于其他包容元素。多个包容器不能共享所有权。其他类可以访问列表中的元素 —— 例如通过继承或关联，但只有直接包容器拥有被包容列表的所有权。这些列表元素连同包容器一同存储、复制和销毁。

列表元素的顺序由建模者决定。这种顺序会很有用 —— 例如，可用于生成编程语言声明序列的代码生成器。如果建模者不关心顺序，可能因为模型处于分析阶段或因为语言与顺序无关，这种顺序虽仍在存在但可以忽略。

表示法

嵌入列表用某分栏中的字符串列表表示，每个元素对应一行字符串。每个字符串是特征的编码表示，如属性、操作、内部转换等等。编码种类由每种元素说明。

定序。字符串的顺序和列表元素的顺序一致，但内部存储顺序可能并非如此，字符串存储可能取决于某种内部特性，如名称、可见性或构造型。注意：每项仍维持在基本模型中的初始顺序。定序信息仅仅在视图中忽略。

省略号。省略号（…）作为列表末尾的元素表示模型中还有符合条件的元素，但未在列表中列出。在其他视图中，可能列出这些元素。

构造型。列表元素可以有构造型。构造型关键字位于元素字符串前面。

特性字符串。特性字符串说明元素的特性。在元素之后的大括号内，用逗号分隔列出特性和约束。

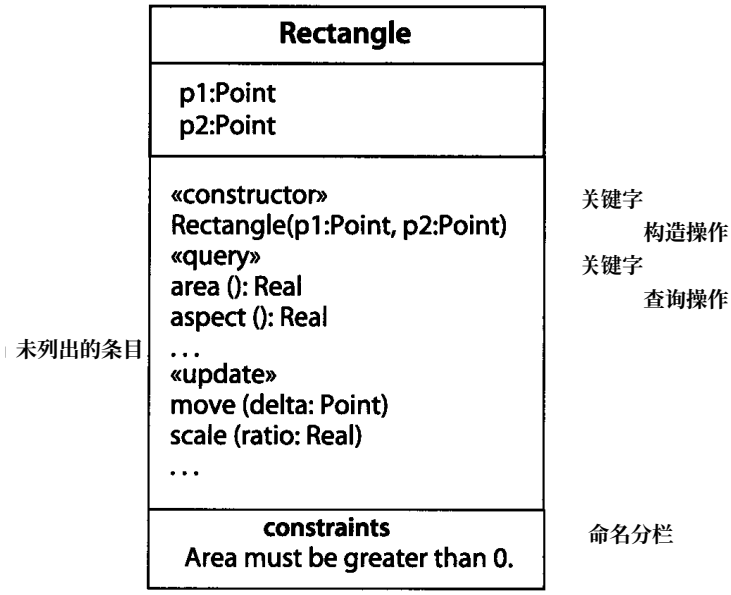


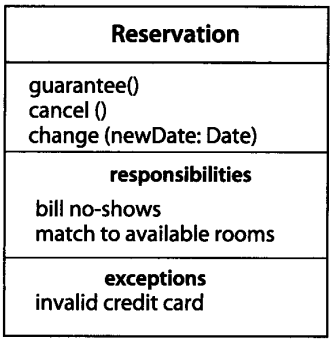
图13-119 应用于一组列表元素的构造型关键字

China-pub.com
下载

组特性。一组列表元素可以有特性或构造型。如果单独的一行中出现构造型、关键字、特性字符串或约束，则此行不表示一个列表元素，而是此后所有后续元素共同的特性，如同它们直接作用于每一行一样。缺省效果持续到出现表中另一组特性为止。插入带有一个空关键字（《》）的一行可以撤销所有组属性，但通常将不受限于组特性的所有项置于表头，这样会更明了些。图13-119表示了作用于多个元素的构造型。

注意，组特性只是一种方便的表示法，每个模型元素仍带有各自的特性值。

分栏名称。分栏可以有名称，用特殊字体表示，标在分栏顶部。如果有省略或新增加的用户自定义的分栏，这种标志十分有用。对类而言，预定义的分栏名为 attributes和operations。用户定义的分栏可能是 requirements。类的名称分栏必须有，因此不需要也不允许分栏名称。图13-119和图13-120表示命名分栏。



预定义的操作分栏

用户定义的分栏

分栏名称

图13-120 有名称的分栏

表示选项

定序。工具可能按存储顺序列出表元素。此时，看不到元素的继承顺序。顺序基于某些内部特性，不表明额外的建模信息。典型的排序规则包括字母顺序、构造型顺序（构造函数、析构函数以及一般方法）、可见性排序（公共、受保护及私有），等等。

过滤。列表元素可以按照某些规则进行过滤。选择规则的说明是工具的职责。如果过滤列表为空，说明没有符合要求的元素，但初始列表可能或不可能包含不满足规则从而不可见的其他元素。工具应负责是否说明以及如何说明局部和全部过滤，还可以用单独的图来对此说明。

如果省略一个分栏，则不能确定其中的元素的存在与否。空的分栏说明没有符合过滤条件的元素。

注意，属性也可以通过组合表示（如图 13-71）。

198. location (位置)

一个运行时实体在某环境中的物理放置，如分布式环境中的对象或分栏。在 UML中，位

China-pub.com 下载

置是分散的，位置单位是节点。

见component，node。

语义

位置的概念需要实体存在空间的概念。UML没有建立三维空间，而是提供了由通信路径连接的拓扑模型空间。节点是运行时实体可以存在的计算资源。节点之间用通信路径相连，实体的定位即分配节点指针。在节点上，有些实体存在于其他嵌入实体内。例如，对象存在于构件或其他对象中，这些实体的位置即其包容实体的位置。

对象或构件实例可以移动到新的位置上。这可以通过成为关系表示，成为意味着在某一时刻第一个实体被第二个实体代替，而第二个实体原来在另一个位置上。

表示法

实体（包括对象、构件实例和节点实例）的位置在另一个实体之内可以用物理嵌套表示，如图13-121所示。包容关系也用组合箭头表示，或者实例可以有一个特性标记 location，其值为包容元素的名称。

如果对象在交互中移动，它可以表示为多个版本，版本间用 become转换相连。见图13-121。become箭头可以有顺序号，说明对象移动的时间。每个对象符号表示某时间段对象的一个版本。消息必须与正确的对象版本相连（如图 13-117）。

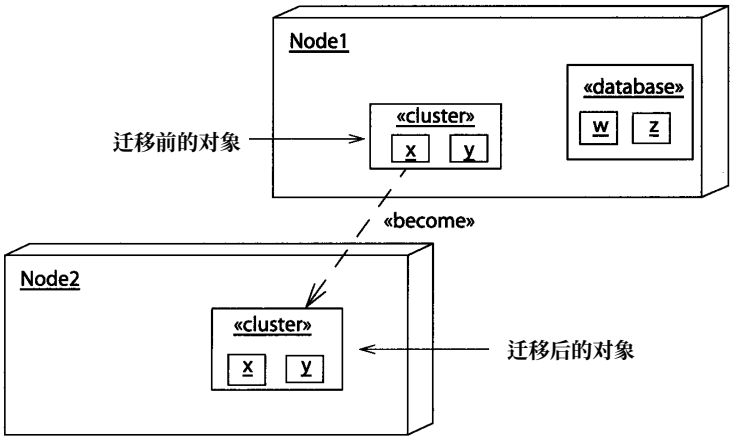


图13-121 节点和对象的迁移

199. many(多)

多重性0 ..* 的缩写 —— 零个或无限多个。换而言之，大小不限。

见multiplicity。

China-pub.com
下载

200. member(成员)

类元的已命名结构化继承部分的名称，可以是属性、操作或方法。每个类元可以有零个或多个各种成员。给定种类的成员列表以字符串列表的形式表示在类元符号的分栏中。

见list。

201. merge(合并)

状态机、活动图或顺序图中的一个位置，两个或多个可选的控制路径在此汇合或“无分支”。反义词：branch。

见junction state。

语义

合并是指两个或多个控制路径汇合的情况。在状态机中，一个状态有多个输入转换。没有提供合并的特定模型构造。如果它是单一运行至完成步骤的一部分，则可以由结合状态表示。

表示法

在状态图、顺序图或活动图中，合并用带有多个输入箭头和一个输出箭头的菱形表示。

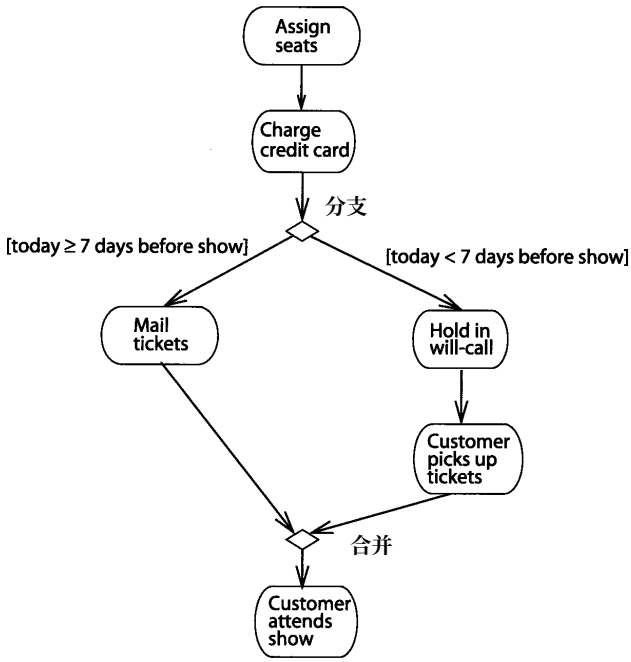


图13-122 合并

China-pub.com 下载

不需要条件，见图13-122。

菱形也用于分支（与合并相反），但不会混淆。因为分支有一个输入箭头和多个输出箭头，并且都有监护条件。

可以结合分支与合并，但用处不大。可以有多个输入转换和多个带标签的输出转换。

注意，合并只是一种便利的表示法，省略它不会丢失信息。合并和分支常常成对使用。

讨论

请区分合并和结合，合并汇合了两个以上的控制路径。在任何执行中，每次只走一条，无需同步。

结合汇合了两条或多条并行控制路径。在任何执行中，所有路径都要走过，当它们都到达结合的源状态时，结合才被激活。

202. message(消息)

从一个对象（或其他实例）到另一个对象的信息的传递，活动可以保证这一点。消息可以是信号或调用操作，收到消息实例被认为是事件的实例。

见call，collaboration，interaction，operation，send，signal。

语义

消息是从一个对象（发送者）向另一个或几个其他对象（接收者）发送信号，或由一个对象（发送者或调用者）调用另一个对象（接收者）的操作。消息的实现有不同方式，如过程调用、活动线程间的内部通信、事件的发生等。在逻辑层上，发送信号和调用操作是类似的，都激活发送者和接收者之间的通信，接收者收到一个值，并由此判断应该干什么。调用可以视为一种信号，激活一个带有一个返回指针参量的发送，随后该参量用于发送一个返回信号给调用者。因此，调用可以被建模为两个消息；一个调用消息和一个返回消息。在实现层上，信号和调用各有不同的特性和行为，因此是不同的UML元素。

信号的接收可能触发接收者状态机的转换。接收者有两种不同的调用处理方式可以选用（由接收者的模型决定方式）。操作作为过程体（方法）实现，当信号到来时它将被激活。过程执行完后，调用者收回控制，并可以收回返回值。另一种方式是主动对象，操作调用可能导致调用事件，它触发一个状态机转换。这种情况下，没有方法体，但转换可以有动作，转换也可为调用者提供一个返回值。转换完成后或调用事件未触发转换时，调用者收回控制。

消息中包括目标对象集的表达式。消息发往集合中的所有对象。除非另行说明（由约束），消息将并行地送往所有对象。这说明执行的顺序是不定的，可以是并列的。如果必须按照一定顺序发消息，它们应该循环发送。对于调用，所有调用完成后，调用者收回控制。

发送消息的时间可以用消息名称上的表达式表示。

China-pub.com
下载

见timing mark。

结构

消息有发送者、接收者和动作。

在交互中，发送者是发送消息的类元角色。接收者是接收消息的类元角色。动作为调用、信号、发送者的局部操作或简单动作，如创建或销毁。动作带有参量表、接收者表达式以及对激活的操作或信号的引用。它还包括消息执行的迭代和条件说明。

在交互中，消息之间有前驱-后继关系和调用-被调用的关系，后者适用于过程方法。每次调用增加一个嵌套层次。在调用中，消息是有序的，也可以并行。

前驱-后继（顺序）关系将线程上的消息组织为一列。一个消息可以有多个前驱或者后继。如果两个消息之间没有顺序关系并且没有共同的前驱，则它们可以并行。如果一个消息有多个前驱，只有在所有前驱完成后，该消息才能执行。这种消息是一个同步点。

调用-被调用（激活）关系定义了嵌套的过程结构。调用过程（使用调用动作）的消息是被调用过程体内所有消息的激活者。其中，被调用消息也有前驱 -后继关系，由此确定相对顺序（允许并行）。

如果消息为调用，在调用过程完成并返回之前，调用者将锁定。如果接收者将消息处理为调用事件，初始转换完成后就出现返回值，此后调用者收回控制，被调用过程继续执行。

顺序和激活关系只涉及同一个交互中的消息。

表示法

顺序图和协作图的表示法不同。

顺序图

在顺序图中，消息用从一个对象（发送者）的生命线指向另一个对象（目标）的生命线的实线箭头表示。与外部消息相比，如果箭头和生命线垂直，说明消息发送是即刻的，至少是很快的。如果箭头倾斜，则说明消息的传送有一定的时间延迟，其间可以传送其他消息。对象发给自身的消息，箭头的起点和终点都是同一条生命线。消息按时间顺序从顶到底垂直排列。如果多条消息并行，它们之间的顺序不重要。消息可以有顺序号，但因为顺序是用相对关系表示的，通常省略顺序号。

传递延迟。消息箭头通常是水平的，说明消息的传递是原子的 —— 即与交互相比它的时间很短，且在此期间不会“发生”其他事件。对多数计算机而言，这是正确的假设。如果消息的传送需要一定时间，在此期间可以出现其他事件（如来自对方的消息到达），则消息箭头可以画为向下倾斜的。

分支。从同一点发出多个箭头表示分支，每个箭头有监护条件。根据条件是否互斥，有条件和并行两种结构。

China-pub.com 下载

迭代。相连的一系列消息可以被封装并标记为迭代，迭代标志说明这组消息可以多次出现。对过程而言，连续执行的条件列在迭代的下方。如果存在并行，则可能有某些消息是迭代的一部分，有些只执行一次。

协作图

在协作图中，消息用带有标签的箭头表示，附在连接发送者和接收者的路径上。路径用于访问目标对象，箭头沿路径指向接收者。如果消息发送给对象本身，箭头指向对象自己，并标有关键字《self》。链上可以有多个消息，沿相同或不同的路径传递。消息的相关顺序由消息标签中的顺序号表明。

双视图

消息箭头上标有消息的名称（操作或信号名）以及参量值。消息上还可以有顺序号，表示它们在交互中的顺序。在顺序图中，箭头的物理位置表明了相应的顺序，可以省略顺序号。协作图中必须有顺序号。顺序号可以表明并行线程，这对两种图都有用。消息还可以有监护条件。

控制流类型。下列不同的箭头形状表示不同的消息控制流。

实心实线箭头 ⟶

过程调用或其他嵌套控制流。在外层控制流再次开始之前，应该完成所有控制流。可以用于常规过程调用。也可用于并行活动状态。表示其中一个状态发送了信号，并在等待嵌套控制流完成。

棍状箭头 ⟶

平面控制流，每个箭头表示下一个步骤。对于嵌套过程，它对应于动作树叶节点的向底扫描活动。

半个棍状箭头 ⟶

异步控制流。与棍状箭头不同，用于表示两个对象之间按顺序执行的异步消息。

虚线棍状箭头 - - ⟶

从过程调用返回，返回箭头可以忽略。因为它在激活结束时是隐含的。

其他变体

可以表示其他控制种类，如超时、中止等。这些是UML核心之外的扩展。

消息标签。语法如下：

$perdecessor_{opt}\ guard\text{-}condition_{opt}\ sequence\text{-}expression_{opt}\ return\text{-}value\text{-}list{:=}_{opt}\ message\text{-}name\ (argument_{list,})$

标签说明发送的消息、参量和返回值，以及大型交互中消息的顺序，包括调用嵌套、迭代、分支、并行和同步等。

China-pub.com
下载

前驱。在协作中，前驱是用逗号隔开的顺序号列表，并后跟反斜线（/）。

sequence-number$_{list,}$/

如果列表为空，则省略该条目。

每个*sequence-number*是一个无任何循环项的*sequence-expression*，它必须与其他消息的*sequence-number*相匹配。

这表示在列表中的所有顺序号代表的消息流发送之后本消息流才能发送（线程超越所要求的消息流，监护条件仍然满足）。因此，监护条件表示线程同步。

注意，顺序号在某个消息之前的消息是它的隐含前驱，不用特别列出。同一前缀的消息顺序号构成序列。数字前驱是指该顺序号最后一位少 1，即消息3.1.4.5是 3.1.4.6的前驱。

在顺序图中，可见的顺序决定序列，并用对象发送自身的任何消息之前出现发给自身的多个消息来表示同步。

顺序表达式。顺序表达式是顺序项列表，之间用冒号（:）隔开。每一项代表交互中一个嵌套层次。如果所有控制并行，则无嵌套。语法如下：

label recurrence$_{opt}$

其中*label*是*integer*或者*name*

整数表示过程调用的相邻高层中消息的顺序。同一整数项中的不同消息在嵌套层是顺序相关的。例如，在激活3.1中，消息3.1.4位于消息3.1.3之后。

名称表示并发控制线程。例如，在激活 3.1 中，消息3.1a和3.1b并行。所有并行控制线程在同一嵌套层次上深度相同。

循环表示条件或迭代执行。表示根据条件执行零个或多个消息。选择有：

* [*iteration-clause*] 迭代

* [*condition-clause*] 分支

迭代代表了给定深度上的消息序列。[*iteration-clause*]可以省略（在未指定迭代条件的情况下）。[*condition-clause*]用伪代码或程序设计语言实现。UML没有说明其形式，可以为*[i:=1..n]。

条件表示一条消息的执行取决于*condition-clause*是否成立。*condition-clause*用伪码或程序设计语言实现。UML没有说明其形式，可以为[x>y]。

注意：分支与迭代的表示法一样，只是没有 *号。可以视为执行一次的迭代。

迭代表示法假设其中的消息将顺序执行，也可能并行执行，用双竖线表示（*||）。

注意，在嵌套控制结构中，循环在内层并不重复。结构的每一层均在语境中指明它自身的迭代。

特征标记。特征标记是说明名称、参量和操作、消息或信号返回值的字符串。具有下列特性：

return-value-list 说明交互中消息返回值的名称，名称之间用逗号隔开。它可以作为后续消息的参量。如果消息不返回值，则省略返回值和赋值操作。

China-pub.com 下载

message-name 目标对象中引发的事件名称（通常是要求执行操作的事件）。可以用不同的方式实现，其中之一是操作调用。若实现为过程调用，则它是操作名且操作必须在接收者的类中定义或继承。在其他情况中，它可以是接收者对象引发的事件名称。通常用消息名称和参量列表类型来确定一个操作。

argument-list 括号中用逗号隔开的参量列表。空表也可以使用括号。每个参量是使用伪代码或适当编程语言写出的表达式（UML中未说明）。表达式可以使用先前消息（在同一作用域中）的返回值和起源于同一对象的导航表达式（即其属性、发出的链和可达路径）。

示例

下面为控制消息标签语法：

2:display(x,y)	单个消息
1.3.1:p:=find(specs)	带返回值的嵌套调用
[x<0] 4:invert(x,color)	条件消息
3.1*:update()	迭代
A3,B4/C2:copy(a,b)	与其他线程同步

表示选项

可以用消息旁的数据标记代替参量和返回值的文本表达式（如图 13-123）。标记是标有参量表达式或者返回值名称的小圆圈，其上的小箭头指向消息的方向（参量）或反方向（返回值）。标记表示返回值和参数。文本和标记两种表示法都可以使用，但是文本方式更简练，建议使用。

消息语法由编程语言的语法说明，如 C++或Smalltalk语言。但视图中的所有表达式应使用同一种语法。

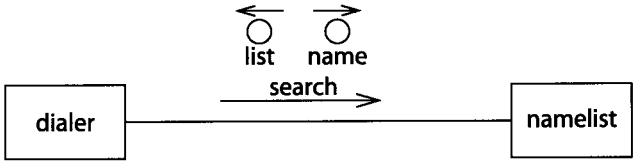


图13-123 数据流标记

203. metaclass(元类)

实例为类的类，通常用于构造元模型。元类可以用关键字《metaclass》建模为一个类的

China-pub.com
下载

构造型。

见powertype。

204. meta-metamodel(元-元模型)

定义了表达元模型的语言的模型。元-元模型与元模型之间的关系类似于元模型与模型之间的关系。这一层次通常仅仅与工具建立者、数据库建立者有关。 UML就是一种被称为元对象机制（MOF)的元-元模型定义的。

205. metamodel(元模型)

定义表达模型的语言的模型，是元-元模型的实例。UML元模型定义了UML模型的结构。

206. metaobject(元对象)

元模型语言中所有实体的统称，如，元类、元类型、元属性、元关联。

207. metarelationship(元关系)

描述符及其实例关系的统称，包括实例关系和强类型关系。

208. method(方法)

操作的实现，说明生成操作结果的算法或过程。

见concrete，operation，realization。

语义

方法是操作的实现。如果操作不是抽象的，则它必须有方法或调用事件，或定义在具有操作的类上，或从组类继承。方法用过程表达式说明，是用描述算法的特定语言（如 C++、Smalltalk或自然语言）书写的字符串。当然，语言应与目的匹配。例如，人类语言适于早期分析，不适用于代码生成。

除非操作声明为抽象的，否则操作声明隐含方法的存在。在泛化继承中，每个重复的操作声明暗示了重载同一操作中的任何继承方法的新方法。

China-pub.com 下载

注意，方法是可执行的过程 —— 一种算法 —— 而不是结果说明。例如，事前 -事后说明不是方法。方法受制于算法、计算复杂性和封装的实现和地址分配问题。

在某些方面，方法的特性会比其操作更严格。尽管操作没有定义为查询，但方法可以声明为查询。但是如果操作定义为查询，则方法必须是查询。同样，方法可以强化并行特性。顺序操作可以被实现为监护或者并行的方法。此时，方法符合操作的声明，只是加强了其中的约束。

表示法

方法用操作声明表示，只是没有抽象特性（如图 13-124）。如果操作是继承的，则方法可以通过以标准文本形式重复操作声明以表示一个具体操作来表示。方法体的文本用附属于，操作列表实体的注解表示，但图中通常不表示方法体，它们对文本编辑器是隐藏的，可用命令行方式表示。

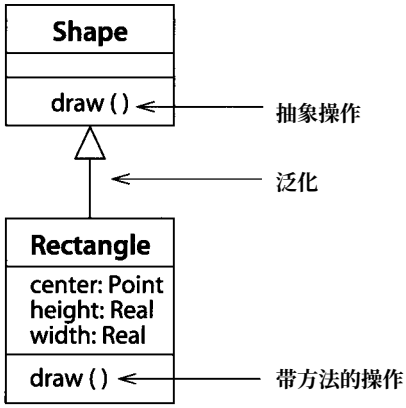


图13-124 非抽象操作的方法

209. model(模型)

系统语义上的完整抽象。

见package，subsystem。

语义

模型是从一个特定视点对系统进行的或多或少的完整抽象。说它是完整的，因为它从给定视图全面地描述了系统或实体。提供独立视图的模型可以独立维护。

模型可以分解为包的包容层次结构，最顶层的包对应于整个系统。模型的内容是从顶层

China-pub.com
下载

包到模型元素的包容（所有）关系的闭包。

模型还可能包含一部分系统环境，由参与者和界面表示。特别可以建立环境与系统元素的对应关系。系统及其环境构成了更高层次上的大型系统。因此，将不同层上的元素以平滑的方式联系在一起是可能的。

不同模型中的元素彼此没有直接影响。但它们可能代表了不同层次或开发步骤中的同一概念。因此，它们之间的关系，如跟踪或精化，对开发过程十分重要，往往会影响重要的设计决定。

表示法

模型可以用带有构造型《model》的包表示。但表示法中没有什么细节来表示模型。工具可以表示模型列表，但是其间很少有关系。最有用的是将模型名称变为其顶层包或者其中所有内容的映射。

讨论

没有一种系统视图或系统本身可以称为完整的，因为系统总能与外界有某种模型中没有表示的联系。因此，封装模型是一个相对的概念，必须指定实际工作的需求。

UML模型是一种包，它着重于某个视图。每个模型有自己的层次，可以与系统其他视图的层次相同或不同。

210. model element(模型元素)

从建模的系统中抽象出来的元素。它与表达元素不同，表达元素（通常可见）是一个或多个人类交互的模型元素。

语义

所有有语义的元素都是模型元素，包括现实世界的概念和计算机系统实现概念。以表现模型为目的的图形元素是表达元素，不是模型元素，因为其中不含有语义信息。

模型元素可以有名称，不同类型元素的名称的使用和约束不同，下面将分别说明。每个模型元素有与其类型相符的命名空间。所有模型元素可以有下列特性：

标记值	任何模型元素或表达元素可以带有零个或多个标记 -值对。标记是说明值的含义的名称。标记在 UML中不是固定的，但可以扩展，从而表示各种建模者或编程工具有用的信息。
约束	模型元素可以有零个或多个约束。约束是用约束语言字符串表达的限制条件。

China-pub.com 下载

构造型　　模型元素可以与零个或多个构造型名称，只要构造型应用于基模型元素。构造型不改变基类结构，但是可以为有该构造型的元素增加标记值。

此外模型元素可以参与依赖关系。

211. model management view(模型管理视图)

模型的这一特征将自身组织为结构化的部分 —— 包、子系统和模型。这种视图有时作为静态视图的一部分，通常与静态视图结合使用。

212. modeling time(建模时间)

于软件开发过程的建模活动中所发生的事情，包括分析和设计。使用注意：当讨论对象系统时，区分建模时间与运行时间是非常重要的。

见development process、stages of modeling。

213. module(模块)

存储和操作用的软件单元，包括源代码模块、二元代码模块和可执行代码模块。它不与单个UML结构相对应，而是包含几个结构。

见component, package, subsystem。

214. multiobject(多对象)

指明多个对象集合而非单对象的类元角色。

见classifier role，collaboration，message。

语义

多对象是代表多个对象的类元角色，多个对象通常在关联的“多”端。多对象用在协作中，表示以一对象集而非单对象为单元的操作。例如，在对象集中查找一个对象的操作作用于整个集合而非个别对象。这种分组不影响系统基本的静态模型。

表示法

多对象用两个矩形表示，上面的矩形偏向斜下方，表示有许多矩形（如图 13-125）。指向

China-pub.com
下载

多对象符号的消息箭头表示送往这些对象的消息，如查找一个对象的选择操作。

要对关联对象集中每个对象执行操作，需要两个消息：一个指向多对象的迭代消息将链分解为单个对象，另一个消息通过这些（暂时）链发送消息。图中可简化将消息合并为一，包括一个迭代和对每个对象的应用。目标角色名采用“多”指示符（*），表示隐含有多个链。虽然可将它写为单个消息，但它在基本模型中（及在任何实际代码中），需要两层结构（迭代查找链，消息使用链）。

来自集合的对象用常规的对象符号表示，并用组成链附属于多对象，表示它属于集合。指向单个对象符号的箭头表示发送给单个对象的消息。

典型地，发送给多对象的选择消息返回对某一对象的引用，随后原发送者再向它发送消息。

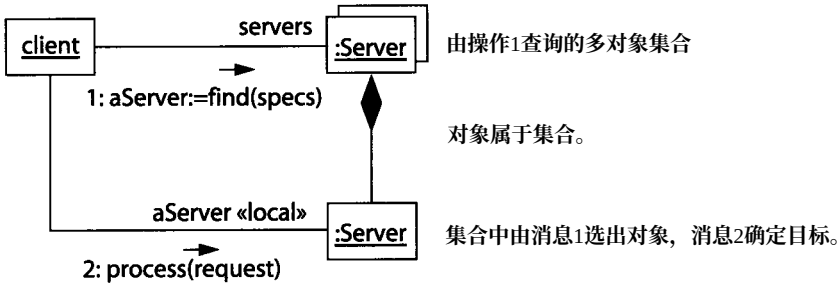


图13-125 多对象

215. multiple classification(多重分类)

泛化的语义变更，其中一个对象可以直接属于多个类。

语义

它是一种语义变更点，其中对象可以是多个类的直接实例。与动态分类共同使用时，对象可以在运行时获得和失去类。它允许类代表对象承担的临时角色。

虽然多重分类符合常见的逻辑关系，但却给编程实现造成了困难，常用的语言不支持多重分类。

216. multiple inheritance(多重继承)

泛化的一种语义变更点，其中元素可以有多个父。这是 U/NL中的缺省设置，在多数情况下是必须的，建模者也可以对元素的使用进行某种设置。对比： single inheritance。

China-pub.com 下载

217. multiplicity(多重性)

说明允许的侯选值范围，如集合可以设定的大小。多重性说明可用于关联端、组成类中的部分、消息的迭代次数和其他目的。本质上讲，多重性是一个非负整数子集（可能无限）。对比：cardinality。

见multiplicity of association，multiplicity of class。

语义

多重性是对集合大小的约束。概念上，它是非负整数的子集。实际上，它是一个有限的整数集合，是最大值和最小值之间的区间。任何集合必须有限，但上界可以是有限值或者无界（无界的多重性称为“多”）。上界必须大于0，否则多重性为0。0多重性的用处不大，因为它仅允许空集。多重性用字符串表示。

通常，多重性是一个整数范围 —— 有最大值和最小值。但它通常又是不连续的非负整数的集合。整数集合可能是无限的 —— 即上界可能无限（但集合中的每个特定基数是有限的）。

在实际中，这一整数集合可以声明为几个互斥、不相连的整数区间。区间是指最大值和最小值之间的连续的整数。有些无限集合不能这样表示，如偶数集合，但这种简化表示不会丢失太多信息。在设计中，通常只用有上下限的一个整数区间来描述多重性，因为多重性只用于定义可能需要的最大存储空间。

见multiplicity of association和multiplicity of class，以进一步了解这些元素多重性的使用细节。

表示法

多重性用用逗号隔开的整数区间列表的文本表达式表示，格式为：

mininum..maximum

其中*mininum*和*maximum*是整数，或者*maximum*为“*”，表示无上界。表达式2..*读做“两个或更多个”。

区间的格式还可以为：

number

*number*是表示单个大小区域的整数。

包含单个星的多重性表达式*与0..*等价，表示不限制数目（“0个或多个，无限制”）。这种常见的多重性被称为“多”。

示例

0..1
1
0..*

China-pub.com
下载

*

1..*

1..6

1..3，7..10，15，19..*

格式指导

- 建议使用递增顺序的区间。如：1..3，7，10比7，10，1..3好。
- 应将两个相连的区间连成一个，如0..1比0，1好。

讨论

多重性表达式可以含有变量，但模型完成后必须替换为整数——即必须是参数或常数。多重性与动态数组边界不同，在运行时不能改变。它用于说明集合中可能出现的值的范围（最坏情况），应用程序的数据结构和操作应与之相符合。它是建模时的常数。如果运行时边界可变，应选的多重性为“多”（0..*）。

图中可能省略多重性，但它隐含存在于模型中。在完成的模型里，未说明的多重性没有意义。不知道多重性和将它声明为“多”是一样的，因为此时的待选值可以有任意多个，这与“多”的定义一样。

见unspecified value。

218. multiplicity of association(关联的多重性)

关联端声明的多重性。

见multiplicity。

语义

定义在关联端的多重性，说明该位置上可以有多少个对象。

对于二元关联，目标端的多重性限制了源端的一个对象可以与目标类的多少个对象之间有关联。多重性作为一个整数区间给出（见multiplicity）。常见的多重性有1、0或1、0或无限多、1或多。“0或无限多”被称为“多”。

在n元关联中，多重性在每个n-1维端上定义。例如，给定类(A，B，C)之间的三元关联，C端的多重性分别说明C可以有多少个对象参与A和B的关联。如果多重性为多，多，1)，则对每个(A，B)对，只有一个C值；对每个(B，C)对可以有许多个A值，但A、B、C的许多值可以参与关联。

见n-ary association中对n维多重性的讨论。

表示法

多重性标注在使用它的路径旁（如图13-126），数字范围形式为n1..n2。

China-pub.com 下载

见multiplicity，以进一步了解语法和规范说明（可能比多数实际应用所需的内容更广泛）。

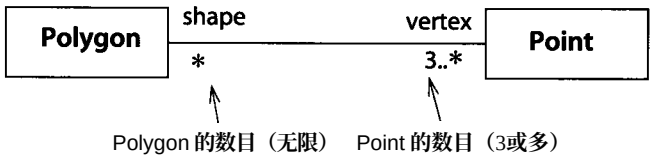


图13-126 关联的多重性

219. multiplicity of attribute(属性的多重性)

每个对象可能有的属性值。

语义

属性的多重性说明有该属性的对象可以有多少个值。

常见的多重性为（1..1），即每个对象的属性有1个值，其他还有0或1（可选或空值）、0或无限多（值的集合）、1或无限多（非空集合）。“0或无限多”称为“多”。

表示法

多重性标在分栏中的属性名称之后的括号中，后面有冒号（如图 13-127）。如果没有标出，则多重性为1（缺省值）。

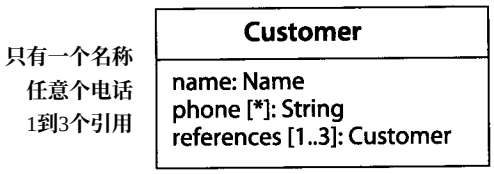


图13-127 属性的多重性

220. multiplicity of class(类的多重性)

类的实例可能的数目范围——同时可以存在多少个实例。

语义

对于类，多重性说明类可以有多少个实例，缺省值是无限多个，但有时确定数目也很有用。特别是在声明单个类——只有1个实例的类时，需要用多重性建立整个系统的语境和参数。

China-pub.com
下载

协作中也用到类的多重性，它附属于类元角色，说明在一个协作实例中可以有多少实例被绑定于角色。

表示法

类或类元的多重性为标在矩形右上角的字符串（如图 13-128）。多重性为多（无限）时，字符串可以省略。

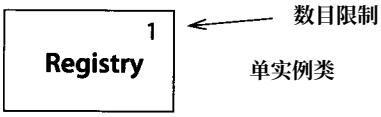


图13-128 类的多重性

221. n-ary association(n元关联)

3个或更多类之间的关联。对比：binary association。

语义

关联的每个实例都是来自于相应类的 n元组值。一个类可以在关联的多个位置中出现。二元关联是一种特例，有简化的表示法和特有的特性（如导航性），这些特性对 n元关联没有意义（至少是相当复杂）。

n元关联也可以声明多重性，但不像二元关联那么容易理解。关联端的多重性说明了当另外n-1端的值固定时本端值的数目。这种定义也适用于二元关联。

聚合（包括组合）只对二元关联有意义，n元关联的角色不能有组合或聚合标记。

二元关联和只有两端的 n元关联没有什么语义区别。有两端的关联被认为是二元关联，有两个以上端的关联被视为 n元关联。

表示法

n元关联表示为一个大菱形（大是相对路径端点而言），以及由菱形引出的通向各个参与类的路径。关联的名称（如果有）标在菱形旁。修饰符号与二元关联一样位于每条路径的端点上。可以有多重性，但不能有限定符和聚合。

可以用虚线将关联类符号与菱形符号相连，以此表示有属性、操作和 /或关联的n元关联。

示例

图13-129表示一支球队每个赛季的最高记录保持者。记录保持者在赛季期间可以转会，

China-pub.com 下载

在其他队中留下记录。在记录本上，每个链是独立的。

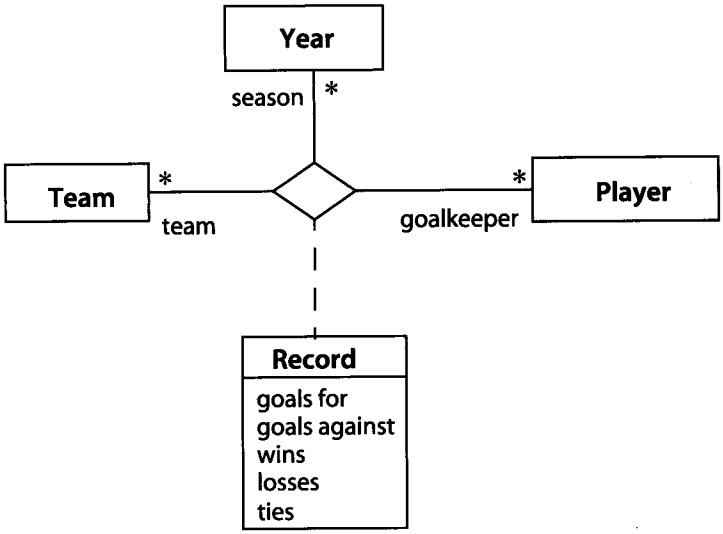


图13-129 三维关联也是关联类

格式指导

通常，线从菱形顶点或某边的中点开始绘制。

讨论

在n元关联中，多重性的定义是按其他 n-1维进行的。例如，给定类（A，B，C）之间的三元关联，C端的多重性说明 C与每个A、B对之间的关联中可以有多少个C对象。如果多重性为（多，多，1），则对每个可能的（A，B）对只有一个C值。对于给定的一个（B，C）对，可以有多个A值，且A、B、C的多个值独自参与关联。在二元关联中，该规则简化为一个关联端对另一端的多重性。

只对一端定义多重性没有意义（正如某些作者所提议）。因为对任何有意义的n元关联端，多重性都是多。否则，n元关联可以化简为一个类和一个包含其他类的关联类之间的二元关联，这样更容易实现。因为二元关联更易实现，且宜导航，所以最好避免 n元关联。只有当需要所有值来确定一条链时， n元关联才有用。通常， n元关联实现为一个类，类的属性中含有指向参与对象的指针。将它建模为关联是因为关联中没有重复的链。

考虑学生在学期之内选修某位教授的课程的例子（如图 13-130）。学生不会多次选修一位教授的同一门课程，但是可以从同一位教授处选修几门不同的课程。一位教授可以教多门课程。图中表示了多重性。Professor的多重性是（0..1），其他多重性是多（0..*）。

China-pub.com
下载

每个多重性值是相对于其他端的对象对而言的。对于（课程，学生）对，可以有 0个或1个教授；对于（学生，教授）对，可以有多门课程；对于（课程，教授）对，可以有多个学生。

注意：如果关联化为类，则可能有多个相同的（学生，课程，教授）组成，这是不希望出现的情况。

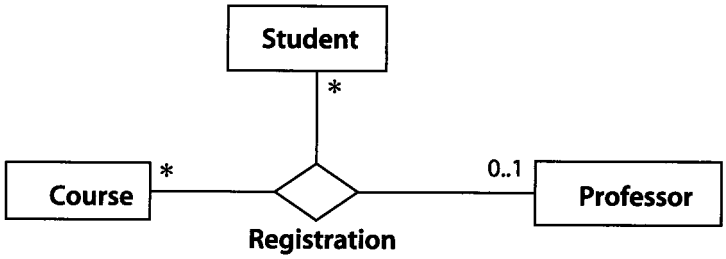


图13-130 n 维关联的多重性

222. name(名称)

用于标识模型元素的字符串。

见namespace。

语义

名称是一种标识符 —— 用某种语言预先定义的有限的字符序列。实现可以规定名称的格式，如不得包含某种符号（如标点符号），还可以规定初始字符等等。特别是，认为名称可用作选择符在不同数据集合中查找主键。例如，罗马字体通常有大写和小写、数字、分隔符 (如下划线连字符)，而其他标点符号取决于具体实现。

工具和语言可以规定名称的长度和使用的字符范围，这些限制比对于任意的字符串（如注释）的限制更严格。

名称是在命名空间，如包或者类内定义的。在命名空间中，名称在其所在的语义组内必须是唯一的，如类元、状态、属性等，但不同组内的名称可以重复（但是也应该避免）。每个命名空间属于更大的命名空间。嵌套的名称及其外部名称以至最外层元素的名称可以用一个名称路径名字符串表示。

表示法

名称用字符串表示。名称通常单独列为一行并只包括非打印字符。规范的名称格式包括英文字符、数字、下划线。如果某个实现允许其他字符，则某些字符可能要编码显示以防止

China-pub.com 下载

混淆。这是工具的实现职责。

用双斜线分隔的不同层次命名空间的名称可以构成路径名。

223. namespace(命名空间)

模型的一部分，名称在此处定义并使用。在命名空间内，每个名称有唯一的意义。

语义

所有有名称的元素都在命名空间内声明，它们名称的作用域也是该命名空间。顶级的命名空间是包（包含子系统）或者包容器，包容器的目的主要是将元素组织为易于人类理解并访问的组，并在开发中将模型组成为易于计算机存储、维护的组。基本的模型元素，包括类、关联、状态机、协作都是它们各自内容（如属性、关联端、状态机、协作角色）的命名空间。每个模型元素的作用域在各自的描述中说明。每种模型元素有各自的命名空间。

命名空间内定义的名称必须唯一（这正是使用名称的目的）。给出命名空间和名称，可以找到具体的模型元素（如果它有名称 —— 有些元素是匿名的，必须通过有名称的元素与它的关联来查找）。命名空间可以嵌套，给出嵌套的命名空间名称，就可以向内查找。

包可以访问或者引入另一个包，从而访问它的命名空间。

系统自身定义了最外层的命名空间，它是所有名称的基础。它是一个包，通常还带有几层嵌套的包，直到得到最终基本元素的名称为止。

表示法

路径名（穿过几个嵌套命名空间的路径）由用双冒号（::）隔开的命名空间（如包或类）的名称连接而成。如下例：

UserInterface::HelpFacility::HelpScreen

224. navigability(导航性)

说明是否可以穿过类的表达式中的二元关联得到与类的实例相关联的对象或对象集。此概念不适用于n元关联（见text）。导航性的特性是枚举类型，值可以是 true(可导航)，false(不可导航)。

见navigation effciency。

语义

导航性说明角色名可否用于表达式，以通过一个对象的关联，到达附属于关联端的对象

China-pub.com
下载

或对象集。如果有导航性，则关联定义了该角色名的另一个关联端的类的伪属性——即角色名可以像属性一样用于表达式，并得到值。角色名还可用于表达约束。

没有导航性说明与角色名相对的类“看不到”关联，因此不能用它构成表达式。没有导航性的关联不会创建源类到目标类之间的依赖关系，但是可能有其他子句创建这样的依赖关系。

没有导航性不是说没有遍历关联的方法。可以从其他方向遍历关联，可以在其他类的实例中进行搜索，找出指向对象的类。这种方法仅仅在小范围内适用。

n元关联不能定义导航性，因为这需要对类的集合一一定义导航性。这是可以实现的，但是作为一种基本特性就太复杂了。这不是说 n元关联不能遍历，而是这样的规则过于复杂，难以用布尔值定义。

导航性通常与导航效率相关，但是 UML规则中没有严格的要求。

表示法

可导航的关联用附属于目标类的关联路径上的箭头表示。箭头说明遍历的方向（如图 13-131）。导航性的符号可以省略（对于图中所有的关联）。箭头可以画在0、1或者2个关联端上。

为了方便，对于双向导航的关联可以将省略箭头。理论上，这可能会与两个方向都不能导航的关联混淆，但是实际上很少有这种关联，因此出现时可以特别表明。

没有必要标注“未确定”的导航性。如果没有确定导航性，也可以归入常见的状态。关于导航性的讨论只是对它作或者不作限制。

不可导航的：
产品未存储订单列表

可导航的：
每一个订单都有对应的产品列表。

可以找到某产品的订单，但必须搜索产品。

图13-131 导航性

225. navigable(可导航的)

一种可以用于表达式遍历的关联或者链。它的可导航特性为 true。这种链通常用指针或者指针集实现。

见navigability、navigation efficiency。

226. navigation(导航)

在图中遍历连接，特别是对象模型中的二元链和属性，从而得到对象的映射值。在后一

China-pub.com 下载

种情况下，导航路径可以表示为属性名或者角色名的序列。

见navigability。

227. navigation efficiency(导航效率)

表明是否可能有效地从一个对象开始遍历二元关联从而与之关联的得到对象或对象集。这个概念不适用于n元关联。导航效率与导航性相关，但不是它定义的特性。

见navigability。

语义

可以用常规方法定义导航效率，以便符合抽象设计和各种程序设计语言的要求。如果得到关联对象集的花费正比于集合中对象的数目（不是多重性的上界，那可能是无穷）加上某个固定常数，则认为二元关联可以有效地进行导航。计算复杂性时，花费是 O(n)。如果多重性是1或者0、1，则访问的花费必须为常数，这阻碍了变长列表的搜索。一种较松导航效率定义允许的最小花费为log(n)。

虽然通常用嵌入到包含对象属性的块中的指针来实现多重性为 1的可导航关联，也可以用散列表来达到外部实现，表中带有平均访问花费。因此，关联可以被实现为参与类之外的外部对象的查表操作，且仍被认为是可导航的。（某些实时情况下，应该规定最环情况下的花费，而不是平均花费。这不必改变基本的定义，只要换上最坏情况下的花费，但是不能再使用散列表等算法）。

如果关联在给定方向上不能进行导航，并不是说完全无法遍历，而是遍历的花费将很大——例如要在更大的表中查找。如果某个方向上的访问不经常出现，则查找是合理的选择。导航效率是设计概念，它允许设计者用计算的复杂性花费设计对象访问路径。通常，导航隐含导航效率。

可能（如果很少）有在两个方向上都不能高效导航的关联。这种关联可能实现为链的集合，在进行任何方向的导航时必须对该集合进行搜索。它可以遍历，但是效率很低。总之，这种关联的使用范围很小。

讨论

导航效率说明得到与给定对象相关的对象集合的效率。多重性为 0..1或者1时，实现显然是源对象的指针。当多重性为多时，通常实现为包含许多指针的包容器类。包容器类自身可能驻留在类的对象的数据记录上，这由它是否可以通过固定花费（指针访问的常规情况）得到来决定。包容器必须是可以高效导航的。例如，对于关联而言，所有链的列表不是高效的，因为对象的链和许多无用的链混在了一起，需要查找。按对象存储的链的列表是高效的，因为避免了不必要的查找。

China-pub.com
下载

在限定关联中，背离限定符方向的导航性表示可以通过源对象和限定符值高效地得到对象或对象集合。这和用散列表或按限定符值索引的二叉树查找的实现是一致的，这是将限定符作为建模概念的目的。

228. node(节点)

节点是运行时的物理对象，代表一个计算资源，通常至少有存储空间和执行能力。运行时对象和运行时构件实例可以驻留在节点上。

见location。

语义

节点包括计算设备（至少商业模型中的）和人力资源或者机械处理资源。节点可以用描述符或实例表示。节点定义了运行时可计算实例驻留的位置，可计算实例包括对象和构件实例。

物理节点有更多的特性，如能力、吞吐量、可靠性。 UML没有预定义这些特性，但它们可以在UML模型中用构造型或标记值建立。

节点可用关联连接表示通信路径。这些关联可以有构造型，以区分不同的通信和通信的不同实现。

节点是实现视图中的继承部分，不属于分析视图。部署模型中表示节点实例而不是节点类型。虽然节点类型有重要意义，但是各个节点的类型通常是匿名的。

节点是一种类元，可以有属性。通常，节点实例在部署图中表示。节点的描述符用处很小。

表示法

节点的表示看起来像立方体的偏方向投影。

节点描述符的语法是：

node-type

其中*node-type*是类元名称。

节点实例有名称和类型名称。节点可能具有带下划线的名称字符串。名称字符串的语法如下：

name：*node-type*

其中*name*是各个节点的名称（如果有）。*node-type*说明它是何种类型的节点。两个元素都是可选的。

依赖箭头（指向构件的虚线箭头）用于表示节点类型支持构件类型的能力。可以使用构造型来声明依赖关系。

构件实例和对象可以表示在节点实例符号中。这说明该元素驻留在节点实例上。还可以

China-pub.com 下载

用聚合和组合关联路径表示包容关系。

节点之间用关联符号相连。两个节点之间的关联说明它们之间的通信路径。关联可以有构造型，说明通信路径的特性（如信道或者网络的种类）。

示例

图13-132表示了包含一个对象（簇）的两个节点，该对象从一个节点的构件向另一个节点迁移。

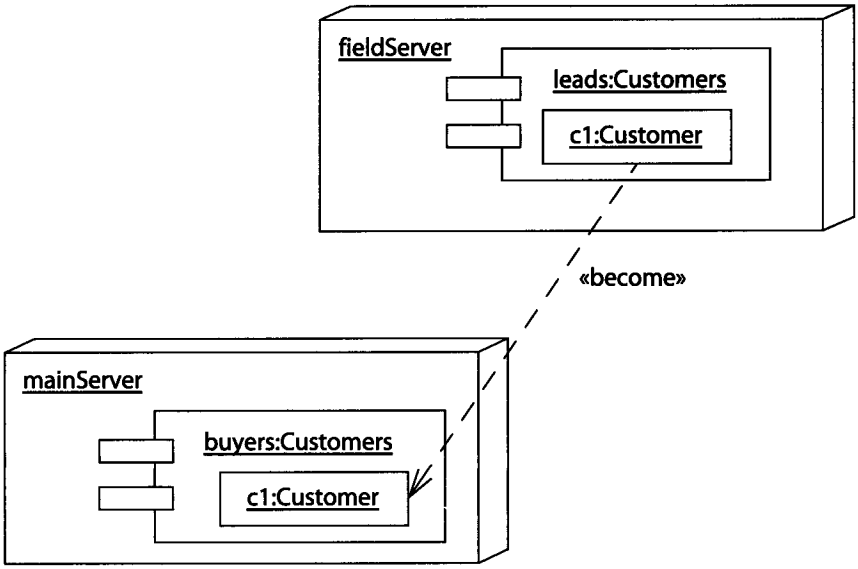


图13-132 节点之间的迁移

229. note(注解)

表示注释或者其他文本信息的符号，如方法体或者约束。

表示法

注解用右上角向下折的矩形表示。其中有不能由 UML解释的文本或者文本的扩展（如嵌入文件）。注解可以为不同的模型元素提供信息，如注解、约束、方法等。注解通常不会明确说明其解说的元素的类型，但是可以从格式和使用中看出来。注解可以用虚线与元素相连。如果注解解释多个元素，虚线指向每个元素。

注解可以用关键字来说明其意义，如关键字《constraint》说明这是约束。

China-pub.com
下载

示例

图13-133表示了各种注解，包括操作的约束、类的约束和注释。

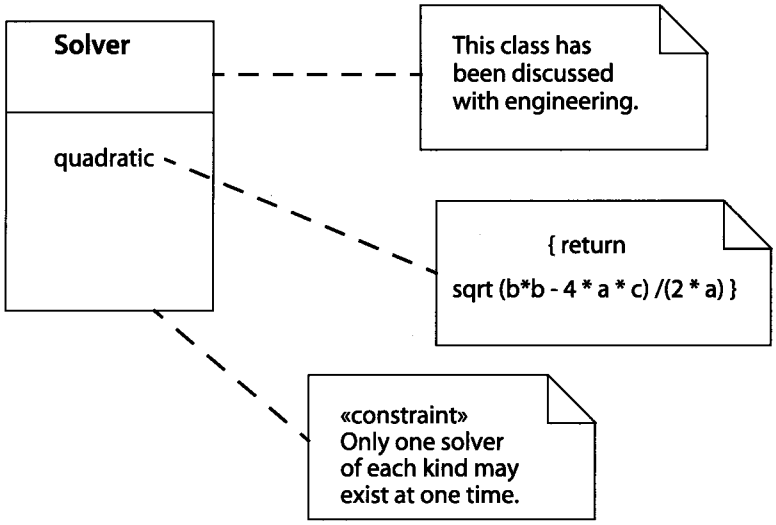


图13-133 注解

230. object(对象)

封装了状态和行为的具有良好定义界面和身份的离散实体；即对象实例。

见class，identity，instance。

语义

对象是类的实例，类描述了存在的可能对象集。一个对象可从两个相关视点来观察：作为一个实体，它在某个时刻有明确的值；作为一个身份持有者，它在不同时刻有不同的值。前一个视图相当于快照，快照即一个系统在某个时刻某一点的存在。快照中的对象具有一个位置（在分布系统中），且每一个属性都有值。对象通过链集和其他对象相连接。

每一个对象拥有唯一的身份且可通过唯一的句柄引用，句柄可标识对象和提供对对象的访问。把对象作为一个身份是与协作实例相对应的，在协作实例中对象通过交换消息与其他对象在运行时进行联系。

在对象的完全描述符中，每一个属性都有一个属性槽——即，每一个属性在它的直接类和每一个祖先类中都进行了声明。当对象的实例化和初始化完成后，每个槽都有了一个值，它是被作为属性类型声明的类元的实例。当系统执行时，属性槽中的值可以改变，除非属性

China-pub.com 下载

的可变性被隐藏。在操作执行的任何时刻，对象的值必须满足模型所施加的所有隐式和显式的约束。在操作执行的过程中，约束可以暂时忽略。

如果执行环境允许多重分类，则一个对象可以是多个类的直接实例。在对象的直接类中和任何祖先中声明的每一个属性在对象中都有一个属性槽。相同属性不可以多次出现，但如果两个直接类是同一祖先的后代，则不论通过何种路径到达该属性，该祖先的每个属性只有一个备份被继承。

如果执行环境允许动态分类，则在执行期间对象可以改变它的直接类。如果可在过程中获得属性，则它们的值必须通过更改直接类操作指明。

如果执行环境允许多重分类和动态分类，则在执行过程中可以获得和失去直接类。然而，直接类的数目不能少于 1（类必须有结构，即使它是暂时的）。

可以调用对象去执行任何直接类的完全描述符中的任何操作 —— 即对象拥有直接和继承操作。

对象可以作为变量和参数的值，变量和参数的类型被声明为与对象相同的类或该对象直接类的一个祖先。换而言之，类的任何后代的实例可以是变量的值，该变量的类型被声明为类。这即替代原则，该原则不具有逻辑必要性，但可以简化编程语言的实现。

表示法

对象是类的实例。实例表示法的总原则是用相同的几何符号作为描述符，但用带有下划线的实例名将它作为个体区分开来。所有的值都在实例中显示，但所有实例共享的特性仅在描述符中标注。

对象的规范表示法为包含两个分栏的矩形。顶部的分栏包含对象名和类名，底部的分栏包含属性名和值的列表（如图 13-134）。无需显示操作，因为对于同一类的对象来说它们都是相同的。

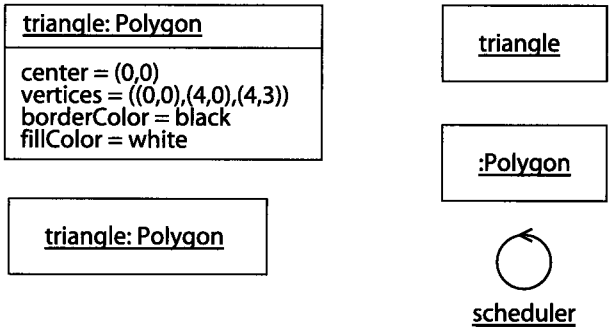


图13-134 对象表示法

顶部分栏显示对象名和类名，都标以下划线，语法如下：

China-pub.com
下载

objectname：*classname*

如果必要的话，类名可包括封装包的完全路径名。包名先于类名且用双冒号隔开。例如：

displayWindow:WindowingSystem::GraphicWindows::Window

类的构造型可以用文本描述（位于名称字符串上的书名号中）或作为右上角的图标。对象的构造型必须符合它的类的构造型。

在显示对象为实例的多类时，用逗号分隔类名列表。在协作中某些类可能是对象扮演的暂时的角色。例如：

aPerson:Professor,Skier

显示处于特定状态的对象，使用如下语法：

objectname ：*classname[statename-list]*

列表必须是一个用逗号隔开的可以并行的合法状态名列表。

若要显示类的变化（动态分类），对象必须显示两次，每个类一次。两个符号通过成为关系连接起来，表明它们代表同一对象。

第二个分栏用列表显示对象的属性及值。每个值行的语法如下：

attributename：*type=value*

在类的属性声明中类型是冗余的，可以省略。值用代表值的字符串指明。属性名不用下划线。

对象的名称可以忽略。在这种情况下，类名后的冒号应该保留。这表示一个匿名类对象，通过它的关系给予身份。每个包含一个匿名对象的符号表示一个独特的对象，它通过与其他对象之间的关系将自己区分开来。

对象类可以被省略（连冒号一起），但当可能避免混淆时应该显示。

属性值分栏作为一个整体可以省略。

值没有意义的属性可以省略。

为了显示在计算过程中属性值的变化，可用成为关系来表明这是同一对象的两个版本。

231. object diagram(对象图)

对象图显示某时刻对象和对象之间的关系。对象图可看成类图的特殊用例，类图显示实例和类。对象图也和协作图相关，协作图显示处于语境中的对象原型（类元角色）。

见diagram。

表示法

对于对象图无需提供单独的形式。类图中就包含了对象，所以只有对象而无类的类图就是一个“对象图”。然而，“对象图”这一术语在刻划特定使用时非常有用。

China-pub.com 下载

讨论

对象图用对象集和链表示系统某时刻的状态。它包含带有值的对象，而非描述符，当然，在许多情况下对象可以是原型。用协作图可显示一个可多次实例化的对象及其关系的通用模式，协作图包含对象和链的描述符（类元角色和关联角色）。如果将协作图实例化，则产生对象图。

对象图不显示系统的演化过程。出于此目的，可用带消息的协作图或顺序图表示一次交互。

232. object flow(对象流)

各种控制流表示了对象与对象、操作或产生它（作为输出）或使用它（作为输入）的转换间的关系。

见control flow、object flow state。

语义

对象流是将对象流状态作为输入或输出的控制流。

表示法

用从源实体到目的实体的虚线来表示对象流。一个或两个实体可以是用对象符号表示的对象流状态。箭头上可用一个关键字来表明它是何种对象流（变成或复制）。如果箭头上没有标签，则为变成关系。

见object flow state，become，copy。

233. object flow state(对象流状态)

代表计算过程中某时刻某个类的对象的存在的状态，如交互视图或活动图。

见control flow，class-in-state。

语义

无论活动图还是交互视图都表示了通过消息激发的目的对象的操作间的控制流，但是消息并没有显示做为操作参量的对象流。这类信息流可用使用对象流状态的行为模型表示。

对象流状态表示计算过程中某时刻某个类的对象的存在，如交互视图或活动图。对象可以是一个活动的输出和其他活动的输入。在活动图中，对象可以是一个转换的目的（通常用分叉表示，另外的分支是主控路径），以及一个活动的完成转换的源。当前一个转换激发时，

China-pub.com
下载

对象流状态变成活动的。这表示了类对象的创建。为了显示对象变为某个状态的过程，而非新对象的创建，可将对象流状态声明为状态类。

对象流状态必须与它所表示的结果和参数的类型匹配。如果它是一个操作的输出，则必须与结果的类型匹配。如果它是一个操作的输入，则必须与参数的类型匹配。

如果对象流状态后跟一个活动的完成转换，那么一旦对象值有效，活动便执行。无需附加的控制输入。换而言之，正确形式的数据创建是活动执行的触发器。

控制路径的前一个动作和值的对象流状态能导向一个复杂转换，这表明一个活动的发生同时需要控制路径和值的出现。当所有的输入转换准备好后，活动执行。多路径到达一个转换表示同步。

对象流状态通常对于人们理解输入输出关系很有用处，而不是为了让计算更简洁。用对象流状态表示的信息是已经可得的。

活动图中的活动产生的事件可建模为一个对象流状态，其类元是一个信号。可使用《signal》构造型。对象流状态是活动的输出。如果活动产生多个事件，则对象流状态是分叉的目标。

表示法

状态类的对象在活动图中用一个矩形表示，其中包含带下划线的类名，后跟用中括号括住的状态名：

Classname[*Statename*]

例如：

Order[Placed]

对象流状态的符号表示对象做为过程中的一个状态存在，而不仅是对象自身。对象流状态的符号（代表一个状态）可作为一个转换箭头的目的和多个转换箭头的源。为了在活动图中把它们与普通转换区分开，它们用虚线而非实线表示。它们代表对象流。

示例

图13-135表示了活动图中的对象流状态。一个对象流状态由一个操作的完成创建。例如，Order[Placed]由Request Service 的完成创建。因为这个活动后跟着另一个活动，所以对象流状态Order[Placed]是分叉符号的一个输出。另一方面，状态 Order[Entered]是活动Take Order完成后的结果，它没有其他后继活动。

图13-136显示了有关建房的活动图的一个部分。当框架建好后，木匠空闲下来可去建屋顶，房子可以装水管了。这些事件可以建模为信号的对象流状态 —— Carpenter free 和 Frame ready。建屋顶和安装水管作为这些事件的结果。因此对象流状态作为活动的输入显示。在模型中，一个活动的完成产生一个事件，事件触发隐含在活动的连接中的下一个活动。明显的事件要求可省略。因此，对象流状态信号的出现是为了提供信息，而非结构的实现。

China-pub.com 下载

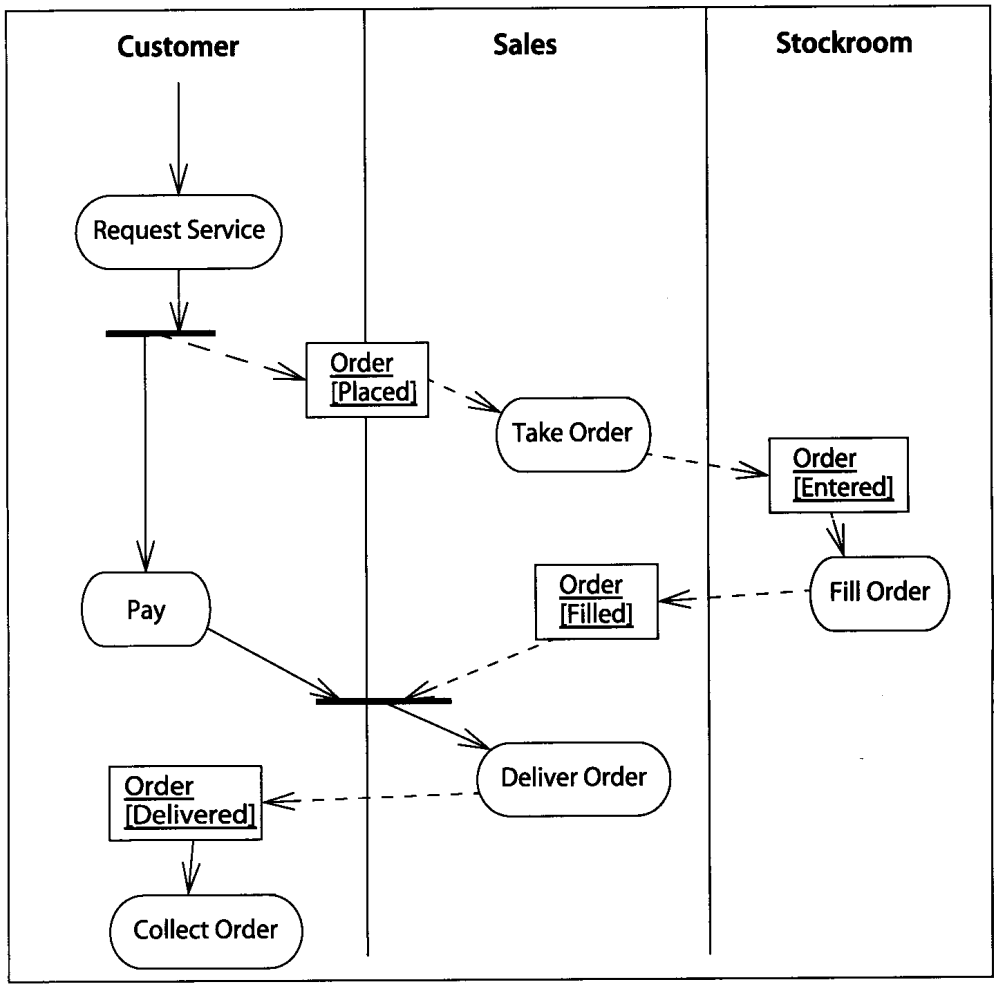


图13-135 活动图中的对象流状态

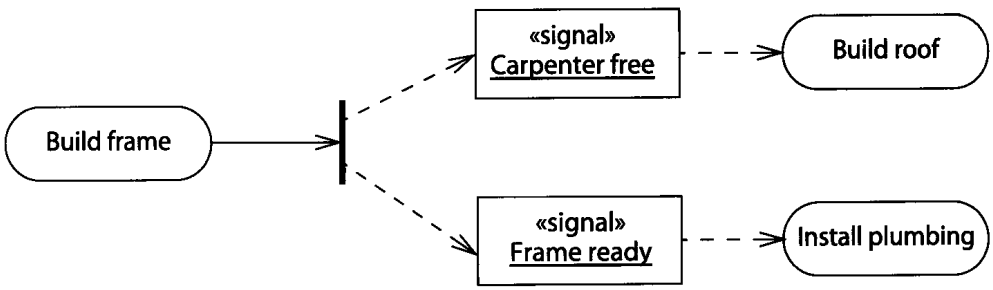


图13-136 活动图中的信号产生

China-pub.com
下载

讨论

对象流状态表示一个计算的数据流视图。然而，和传统的数据流不同的是，它只存在于控制流模型（状态机或活动图）的确定点，而不是处于数据流模型中。这使它处于面向对象的框架结构中。面向对象将数据结构、控制流和数据流三个视点统一到同一个模型中。

234. object lifeline(对象生命线)

顺序图中的线，表示对象的存在期。

235. object set expression(对象集表达式)

当在运行时计算时产生对象集的表达式。

语义

发送动作的目标是对象集表达式。当这样一个表达式在运行时计算时，产生一个对象集，一个被标注的信号被平行发送到对象上。对象集表达式可能产生一个元素，在这种情况下发送动作是一个普通的顺序动作。它甚至不产生元素（即空集），在这种情况下没有发送动作发生。

236. OCL

对象约束语言，一种指定约束和查询的文本语言。 OCL不是用于编写动作或可执行代码的。详细资料见[Warmer-99]。

语义

对象约束语言（OCL）是一种文本语言，用于编写导航表达式、布尔表达式和其他查询语句。它可用于构建约束、监护条件、动作、前置条件和后置条件、断言和其他 UML表达式。有关OCL完整的语法和语义描述见[Warmer-99]。后面选择的概要包含了有关创建导航表达式和布尔表达式最有用的OCL语法。完整的语言包括了大量有关汇集和简单类型的预定义操作符。

表示法

下面列出一些普通导航表达式的语法。这些形式可以链接在一起。最左边的元素必须是对象或对象集表达式。表达式意味着运行在一个值集上。详细资料和语法请看 OCL描述。

China-pub.com 下载

item.selector	*selector*是项目的一个属性名或项目链接的目标末端的角色名。结果是属性值或相关对象。结果是值或是值集取决于项目的多重性和关联。
item.selector(*arguments*$_{list}$)	*selector*是项目的一个操作名。结果是作用于项目上的操作返回值。
item.selector[*qualifier-value*]	*selector*指明了限定项目的限定关联。qualifier-*value*是限定符属性的值。结果是通过限定符选择的相关对象。注意这种语法可以限定形式作为数组下标。
set→*set-property*	*property*是内嵌的OCL集合函数名。结果是集合的特性。如果set-*property*不是预定义的OCL函数则不合法。下面列出了若干特性。
set→select（*boolean-expression*）	*boolean-expression*写在集合中对象的项目中。结果是集合中满足布尔表达式的对象子集。
set→size	集合中元素的数目。
self	表示当前对象（如果语境清晰则可省略）。
operator	常用的数学和布尔操作符：= < > <= >= <> + - * / not。

示例

flight.polit.training_hours>=flight.plane.minimum_hours

拥有足够训练时间的飞行员集

company.employees->select(title="boss" and self.reports->size>10)

做过超过10次报告的老板的数目

237. operation(操作)

操作是可以调用对象执行的转换或查询的规格说明。它有一个名称和参数列表。方法是执行操作的过程。它有运算法则和过程描述。主动类的操作也可通过调用事件执行。

见call，call event，method。

语义

操作指明了目标对象状态（以及从目标对象可达的系统其余对象的状态）的转换或返回给操作调用者值的查询。操作可以作为方法或调用事件来实现，其中事件导致了主动对象的状态机的转变。操作由调用激发，调用者被挂起直到操作执行完成，此后调用者继续调用点后的控制，如果操作提供了返回值则接收。

操作在类中声明。声明被类的后代继承。如果另外的声明有相同的“匹配特征标记”，则

China-pub.com
下载

为相同操作。实现可指定一条关于匹配特征标记的规则来测试冲突，它包括操作名和参数类（而不是名称或方向），但不包括返回参数。相同的操作可出现在后代类中，此时，它被认为是继承声明的重复并被忽略。目的是利用名称匹配来允许一个操作在不同包的类中多次声明。作为所有其他声明的共同祖先的操作声明被称为起源，它表示被其他操作继承的操作统治声明。

如果两个操作声明有相同的名称和参数类型的有序列表（不包括返回参数），但其他特性不同（例如，一个参数在操作中是输入参数，而在别的操作中是输出参数），那么声明冲突，且模型为非良性结构。

方法是一个操作的执行（也可以通过调用事件执行）。如果类中声明的操作无抽象特性，则类中有一方法定义。否则，操作可能是抽象的（且无方法），或是具体的，带有继承方法。

结构

操作具有以下组成要素：

并发 并发的语义是调用同一个被动实例，它是一个枚举值，可取值有：

- sequential：调用者必须协作以使每一时刻只有一个对对象的调用可执行（在任何顺序操作上）。如果并发调用发生，则语义和系统的完整性不能保证。
- guarded：并发线程对一个对象的多次调用可能同时发生（在任何监护操作上），但同一时刻只允许一个调用发生。其他调用被阻塞直到第一个操作执行完成。确保不会由于同时发生的块而产生死锁是建模者的责任。在同时发生的顺序操作的情况下监护操作必须正确执行（或阻塞自己），否则，不能声明监护语义。
- concurrent：并发线程对一个对象的多次调用可能同时发生（在并发操作上）。所有调用可按正确语义并发执行。设计并发操作时必须保证在同一对象的并发、顺序或监护操作的情况下都能正确执行。否则，不能声明并发语义。

多态 操作（方法或调用事件）的实现是否可以被后代类重载。如果可以，实现能够被后代类重载，提供了方法的一种新定义或一个不同的状态机转换。实现呈现不同形式——即多态。如果不可以，当前的实现不做任何改变地被后代继承。它只有一种形式。

查询 操作的执行是否改变系统的状态——即是否是一个查询。如果是，操作返回值，但无副作用。如果不是，它可能更改系统状态，但不能保证改变。

名称 操作的名称，字符串。名称和参数类型列表（不包括参数名称或返回类型）被称作操作的匹配特征标记。匹配的特征标记在类和它的祖先中必须是唯一。如果有重复，则认为是操作的重复声明，它必须完全

China-pub.com 下载

匹配。如果匹配，则除了在最高层祖先的操作声明外，其他全被忽略。如果不匹配，则模型为非良性结构。

参数列表　操作参数声明列表。见parameter list。

返回类型　调用的操作如果有返回值，则为返回值类型列表。如果没有返回值，则此特性为空值。注意许多语言不支持多返回值，但它仍是一个有效的建模概念，可通过许多方法实现，如把一个或多个参数作为输出值。

作用域　操作是否实施在单个对象或类本身（所有者作用域）上。可取值有：

- instance　操作可以实施在单个对象上。
- Class　操作可以实施在类本身上——例如，创建类实例的操作。

说明　描述操作执行产生的效果的表达式——例如，之前或之后条件。UML没有规定说明的格式，可采用各种格式。

可见性　对于其他类而非定义操作的类的操作的可见性。见visibility。

方法具有与操作相同的组成元素。此外，它还有其他的组成元素：

行为　描述方法实现的可选状态机。

体　描述方法过程的表达式。可用字符串或可能的分析格式来表示。尽管信息说明可用自然语言表示，但体通常用编程语言表达。一般说来，如果使用状态机，则不使用体的值。

协作　协作集把方法的实现描述为角色间的有序消息集（交互）。

调用事件的组成元素和操作相同。操作的实现必须指定一个或多个转换，转换可由调用事件触发。

表示法

操作用由操作的特性组成的字符串表示。缺省语法是：

$$《stereotype》_{opt}《visibility》_{opt}\ name（parameter\text{-}list）：return\text{-}type_{opt}\{property\text{-}string\}_{opt}$$

构造型、可见性、返回类型表达式、特性字符串（以及它们的定界符）可省略。参数列表可为空。图13-137显示了一些典型操作。

```
+display (): Location
+hide ()
«constructor» +create ()
-attachXWindow(xwin:Xwindow*)
```

图13-137　包含多种操作的操作列表

名称。操作（不保括参数）的名称字符串。

参数列表。用逗号隔开的参数声明列表，由方向、名称和类型组成。整个列表包含在圆括号中（包括空表）。见*parameter list*和*parameter*。

返回类型。用逗号隔开的类元（类、数据类型和界面）名称列表字符串。操作参数列表

China-pub.com
下载

后跟一个冒号（:），冒号后跟类型字符串。如果不返回任何值（如 C++ 的void）,则冒号和返回类型字符串可省略。一些但非全部编程语言支持多返回值。

可见性。可见性用符号“ + ”、“ # ”或“ - ”标识，它们分别表示 public，protected和private。可见性也可用特性字符串中的关键字表示（如 {visiblity=private}），这种格式用于用户定义或与语言相关的选择。

方法。操作和方法用相同的语法声明。处于泛化层次最顶端的操作特征标记是操作声明。后代类中的同等特征标记是操作的冗余声明，当各个类分别发展时，这些特征标记对于声明方法或声明操作可能有用。如果操作声明具有抽象特性（操作名为斜体或用关键字 abstract标注），则声明没有对应的方法。否则，声明表示了操作声明和实现方法。

利用操作名和有序的参数类型表，但不包括返回参数进行操作和方法匹配。如果剩余特性不一致（例如，输入参数与输出参数不匹配），则存在冲突，模型为非良性结构。

如果两个相同的操作声明没有共同的祖先，然而却是从一个公共类继承来的，则模型为非良性结构。在这种情况下，在继承了这两个操作的类中声明产生了冲突。

方法体。方法体可用附属于操作声明上的注解字符串表示。如果它是用某种语言（一种语义约束）写的正式的说明，则说明的文本应用花括号括起来。否则，如果它只是行为的自然语言描述（注释），则应为标准文本。方法声明和它的状态机或协作的连接没有明显的表达方式，通常用编辑工具中的超链接表示。

说明。描述操作执行效果的表达式。这可用多种方法表达，包括文本、之前之后条件和不变量。无论用何种方法表达，应该根据操作对于系统状态可观察的效果来表述说明，而不是根据执行算法。算法属于方法的范围。

说明用附属于操作入口的注解字符串约束表示。

查询。用isQuery=true 或isQuery=false格式的特性字符串显示选择项。选择 true也可用关键字query表示。缺省为false —— 即操作可改变系统状态（但并不保证能改变）。

多态。用isPolymorphic=true（重载） 或isPolymorphic=false（不重载）格式的特性字符串显示选择项。缺省为true —— 即可重载。

作用域。实例作用域操作用不带下划线的操作字符串指明。类作用域操作用不带下划线的名称字符串指明。

并发。用concurrency=*value*格式的特性串显示选择项，*value*可取sequential、guarded或concurrency其中之一。

信号。关键字《signal》置于操作列表中的某个操作前，这表示类接收信号。参数即信号的参数。声明可以没有返回值。对象接收信号后的反应由状态机表示。在其他应用中，这个符号能表示类对象对于可建模为信号的错误条件和异常的反应。

表示选项

参量表和返回值类型可省略（全部而非分别）。

China-pub.com 下载

可见性的表示可用不同的方法，如使用特殊图标或对元素分组。

操作特征标记字符串的语法可以是某个编程语言的语法，如 C++或Smalltalk。特殊标记的特性包含在字符串中。

格式指导

- 操作名通常以小写字母开始。
- 操作名用正常体表示。
- 抽象操作用斜体表示。

标准元素

semantics。

238. ordering(定序)

用于表述集合是否是有序或无序的值集的特性，如通过关联与一个对象相联系的对象集。

见association，association end，multiplicity。

语义

如果关联端点的多重性上界大于 1，则与二元关联另一端上的每一个对象关联的是一个对象集。定序特性声明集合是有序还是无序。如无序，则集合中的对象无明显顺序，它们仅构成一个普通集合。如有序，则集合中的元素有明显的顺序。元素的顺序是关联表达的信息的一部分——即这是除元素自身信息外的附加信息。通过顺序可获得元素。当新链添加到关联中时，它在顺序中的位置必须由添加链的操作指明。位置可以是操作的一个参量或可能是隐含的。例如，某个操作可以将新链添加到已知链表的末尾，但新链的位置必须用某种方式指明。

注意有序集合不同于由元素的一个或多个属性排序的集合。排序完全由集合中的对象的值决定。因此，尽管它可用于访问，但它不添加任何信息。换而言之，有序关联的信息是元素自身信息外的附加信息。

定序特性可运用到任何有多重性元素上，如多重性大于 1的属性。

定序关系可通过多种方式实现，但通常把实现作为指定语言的代码生成特性。对于普通的说明ordered，实现的扩展可能将代替数据结构。

排序的集合需要对排序规则单独说明，最好把它作为约束。

表示法

定序用花括号里的关键词表示，关键词置于它所作用的路径末端（如图 13-138）。无关键

China-pub.com
下载

词表示无序。关键词{ordered}表示它是一个有序集。出于设计目的，关键词 {sorted}可用以表示一个通过内部值排序的集合。

对于多重性大于1的属性，定序关键词可放在属性字符串的后面，置于花括号内，作为特性字符串的一部分。

如果定序关键词忽略，集合无序。

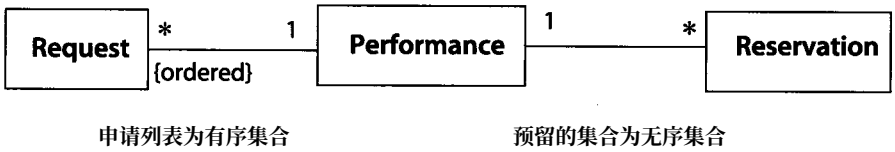


图13-138 有序和无序集合

讨论

有序集合有关于顺序的信息，信息附加在集合中实体本身上。因此，它不可导出，而必须在添加实体时指明。换而言之，任何添加实体的操作必须指明实体在实体表中的位置。当然，将新实体插入到一个隐含位置（如表头或表尾）的操作也能实现。然而仅因为一个集合有序，并不表示可允许实体的任何顺序。这些必须由建模者确定。一般而言，新实体在表中位置是新建操作的参数。

注意二元关联的定序必须对每一个方向单独指定。除非一个方向上的多重性大于1，否则定序无任何意义。一个关联可能完全无序，也可一个方向有序，另一方向无序，或两个方向都有序。

假定类A和类B存在一个关联，在B方向上有序。通常，对象A的操作添加一个新链，指明B对象和新链在现有B对象列表上的位置。更常见的情况是，A对象的操作新建一个B对象，同时也新建一个A和B之间的新链。列表被添加到由A维持的链表中。可以新建一始于B端的新链，但通常新链被插入到A到B列表的缺省位置，因为始于B端的表中的位置意义不大。当然，程序员可根据需要实现更复杂的情况。

两个方向都有序的关联不太常见，因为要指明两个方向上的插入点有些为难。但这也可能，特别是新链被添加到每一方向上的缺省位置时。

注意，排序集除实体集合信息外，不包含任何额外信息。排序在计算上节约时间，却不增加信息。它可认为是一种设计优化，无需包括在设计模型中。它可作为定序特性的一个值，但它无须为添加实体到集合的操作指定位置。通过检查排序列表中属性的方法自动决定新实体的位置。

239. orthogonal substate(正交子状态)

状态集中的一个，可将复合状态划分为子状态，所有子状态都是活动的。

见 composite state，concurrent substate。

China-pub.com 下载

240. owner scope(所有者作用域)

表明特征是作用到单个对象上还是由整个类共享。

见 scope，target scope。

语义

所有者作用域表示对于每一个类的实例有一个不同的属性槽，还是整个类只有一个属性槽。对于操作，所有者作用域表示操作作用于一个实例，还是作用于类自身（如新建操作）。有时简称为作用域。可取值有：

instance	每一个类元实例有自己的单独的属性槽的拷贝。槽中的值独立于其他槽中的值。这是一般的情况。 对于操作而言，操作作用于单个对象上。
class	类元自身有属性槽的一个拷贝。类元的所有实例共享此槽。如果语言允许类作为真实对象，则类的属性作为对象。 对于操作而言，操作作用于整个类上，如新建操作或返回整个实例集的统计数字的操作。

表示法

类作用域的属性或操作带下划线（如图 13-139）。实例作用域的属性或操作不带下划线。

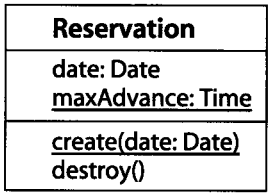


图13-139 类作用域的属性和操作

讨论

对于关联而言，表明链的源位置上是实例还是类元。但这条信息可由另一方向上的目标作用域指明，所以所有者作用域没有必要，因此它不用于关联。

241. package(包)

包提供了将设计元素分组的通用的组织机制。包可以嵌套在其它的包中。系统可能对应一

China-pub.com
下载

个单独的高层包，模型中的所有元素都直接或间接地包含在这个包中。在一个包里既可以出现模型元素也可以出现图。

见access, dependency, import model, namespace, subsystem。

语义

包是一组模型元素和图。每个模型元素，如果不是其它模型元素的一部分，那么它必须在唯一的命名空间内声明。包含一个元素声明的命名空间被称为拥有这个元素。包是一个可以拥有任何种类的模型元素的通用的命名空间。每个图必须被一个唯一确定的包所有，但这个包可能嵌套在（从而包含在）另一个包之内。一个包可能包含附属包和普通模型元素。一些包还可能是子系统或整个模型。可以将整个系统描述为一个高层的包，它直接或间接地包含了所有的模型元素。所有的UML模型元素都能用包来进行组织。

包拥有模型元素、模型子集和图。包是进行配制控制、存储和访问控制的基础。每个模型元素或者为一个包所有，或者自己作为一个独立的包。所以模型元素的所有关系组成了一个具有等级关系的树状结构。然而，模型元素（包括包）可以引用其它包中的元素，所以包的使用关系组成了网状结构。

模型、子系统和系统是特殊类型的包。一个系统可表示为一个子系统，该子系统是包层次结构的根。它是唯一不被其它模型元素所有的模型元素。这个根包间接地包括了模型中的所有组成部分。UML中有好几种预定义的模型和子系统的构造型。

见第14章。

一个包可能与另一个包之间存在依赖关系。在大多数情况下，这种依赖关系囊括了包的内容之间的所有依赖关系。两个包之间的使用依赖关系的含义是：这两个包内至少各有一个元素之间具有使用关系（而不是每一对分属于两个包的元素之间都有使用依赖关系）。

访问依赖关系是包之间特有的关系而不是包内元素之间的关系。它说明客户包中的元素得到了访问许可，可以与提供者包中的元素发生关系。这些关系取决于可见性声明。访问的意思不是目标包中元素的名称占据了源包的命名空间 —— 命名空间是不同的并且元素可以用路径名（包含嵌套的包）唯一标识。引入关系是访问依赖关系的一个变体，它类似于 Ada 语言中的 uses 语句。它从提供者命名空间向客户命名空间添加名称（名称之间不能发生冲突）。但是访问依赖关系并不改变客户命名空间。访问依赖关系主要是大型软件项目的开发中要求采用的机制，而不是UML基本语义的要求。

被嵌套的包可以直接访问嵌套它的外部包（任意嵌套层次）的任何元素，不需要指明引入关系也不需要指明访问的可见性。而一个包要看到它所嵌套的内部包的内容，必须要用引入关系指明它所嵌套包含的包。被嵌套包含的包一般来说是一个封装的边界。

包用关键字 private，protected或public来定义它所包含的内部元素的可见性。私有元素对包外部元素完全不可见；被保护元素只对那些与包含这些元素的包有泛化关系的包可见；公共元素对所有引入包以及它们的后代都可见。

China-pub.com 下载

见access，以详细获取各种包内元素的访问可见性规则。

表示法

包用一个大矩形框来表示，矩形框上带一个短小突出的部分（一个小矩形）连在大矩形框的一个角上（通常是左上角）。它是一个折叠式图标。

如果包的内容没有表示出来，那么在大矩形框中写上包的名称。如果包的内容写在了大矩形框内，则包名放在突出的小矩形中。

关键字字符串放在包名之上。关键字可以包括：子系统、系统和模型。用户自定义的构造型也标记为关键字，但是不能与UML预定义的关键字冲突。

属性列表用花括号括起，放在包名之下。例如：{abstract}。

包中的内容可以列在大矩形框之中。

包元素对外的可见性可以通过在该元素的名称前面添加可见性标志来加以说明（“+”表示公共，“-”表示私有，“#”表示被保护）。

可以画出包符号之间的关系以显示包中的一些元素之间的关系。特别的，包之间的依赖关系（不同于许可依赖关系，如访问和引入）表明元素之间存在一种或多种的依赖关系。

表示选项

工具可以通过选择性的显示某种可见性级别的元素，如所有的公共元素，来说明可见性。

工具也可以通过图形标记，如颜色或字体来说明可见性。

格式指导

人们希望内容庞大的包通过带有名称的图标来显示，而这些具体内容则可以通过“缩放成”一个更详实的视图来动态地访问。

示例

图13-140显示了一个订单处理子系统的包结构。这个子系统本身为一个带有构造型的包。它包含了几个普通的包。而包之间的依靠关系通过虚线来表示。这个图还显示了该子系统所依赖的几个外部包。这些也许是书架外的构件或图书馆元素。

包之间的泛化表明了一个通用包的变体。例如，Externa|Store包可以被实现为Random Store或者“StreamStore”。

讨论

包主要用于访问及配置控制机制，以便允许开发人员，尤其是大型工作组，在互不妨碍的情况下组织大的模型并实现它们。从根本上而言，它们正是开发人员希望的样子。更为特殊的是，要想能够起作用，包必须遵循一定的语义规则。因为它们是作为配置控制单元，所

China-pub.com
下载

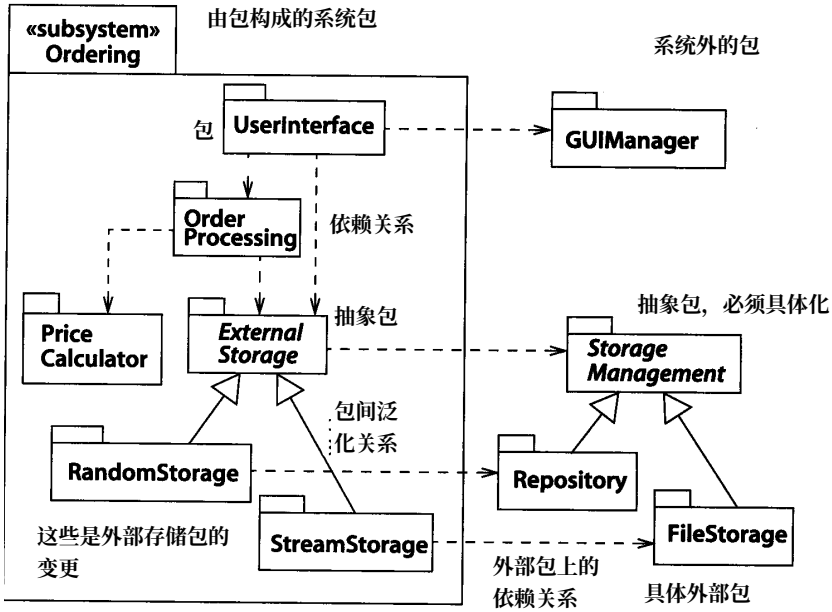


图13-140 包和包之间的关系

以它们应该包含那些可能一起发展的元素。包也必须把必须一起编译的元素分组。如果对一个元素的改变会导致其他元素的重新编译，那么这些元素也应该放到相同的包内。

每个模型元素必须包含在一个包或别的模型元素里。否则，模型的维护、修改和配置控制将变成不可能。拥有模型元素的包控制它的定义。它可以在别的包里被引用和使用，但是对这个包的改变将要求访问许可并对拥有该包的包进行更新。

标准元素

access，extend，facade，framework，stub，system。

242. parameter(参数)

参数即可以被改变、传递或返回的变量的声明。参数可能包括名称、类型和方向。参数被用于操作、消息、事件和模板。对比：argument。

参数使用依赖关系将含有参数的操作或含有这种操作的类同参数的类联系起来。

见 argument，binding。

语义

当使用封装元素时，参数即与之绑定的参量的存放地。它限定了该参量的可取值范围。

China-pub.com 下载

它包括以下的部分：

缺省值　　参数没有参量时所使用的一个值表达式。在参数列表绑定于参量时计算该表达式。

方向　　参数信息流的方向，是一个枚举值。取值如下：

- in　　输入参数。该参数的改变对调用者不可用。
- out　　输出参数。没有输入值。最终值对调用者是可用的。
- inout　　可以修改的输入参数。最终值对调用者是可用的。
- return　　调用的返回值。该值对调用者是可用的。从语义上来说，它与输出参数没有区别，但是它的结果可用于内嵌表达式。

前面所列的选项并不是所有的编程语言都支持，但是各项所含的概念在大多数的语言中都有意义，而且可以被映射成一个有意义的表达式。

名称　　参数名称。必须在所在的参数表内保证唯一。

类型　　对一个类元（在大多数的过程中，是类、数据类型或者接口）的引用。绑定于该参数的参量必须是类元或其后代的一个实例。

表示法

参数用文本字符串表示，它可以分解为参数的不同特性。缺省的语法如下：

direction name: type=default-value

方向。方向是操作名称前的一个关键字。如果该关键字不存在，则方向为 in。方向的可选值有 in, out, inout 和return。返回参数通常位于操作特征标记的别的位置，这时返回参数不必标记方向。

名称。名称为一个字符串。

类型。类型为一个字符串，该字符串是类、接口或者数据类型的名称。

缺省值。该值为一个表达式。该表达式的语言应为特定的工具所能理解，但并不是规范格式。

作用域。如果作用域是类，则操作字符串带有下划线；如果作用域是实例，则操作字符串不带有下划线。

参数依赖关系。参数依赖关系用虚线箭头表示，箭头从含有该参数的操作或含有该操作的类到该参数的类。箭头上标有《parameter》构造型。

示例

Matrix ::transform(**in** diatance:Vector, **in** angle:Real=0): **return** Matrix

以上所有的方向标签都可以省略。

243. parameter list(参数表)

对操作或模板所接受的值的说明。参数表是参数说明的一个有序列表。该列表可以为空，

China-pub.com
下载

在这种情况下，操作不调用任何参数。

见parameter。

表示法

参数表是参数声明的一个序列，由圆括弧括起，并用逗号分隔。

(*parameter* $_{list,}$)

即使参数表为空，圆括弧还是应该显示：

()

244. parameterized element(参数化元素)

见 template。

245. parent(父)

在泛化关系中更具一般性的元素。对于类而言为父类。一个或多个父关系的链（即传递闭包）称为祖先。反义词：child。

见generalization。

246. participates(参与)

参与即模型元素与关系或具体化的关系之间的连接。例如，关联中的类参与，协作中的类元角色参与。

247. passive object(被动对象)

自身不具备线程控制的对象。它的操作在主动对象内的线程控制下执行。

语义

主动对象是拥有控制线程的对象，它们可以激发控制行为。而被动对象是具有值但是不能激发控制的对象。但是当处理已有的线程所接受的请求时，被动对象的操作可以发送消息。

China-pub.com 下载

表示法

被动对象用个类的矩形表示，其中的对象名称带有下划线。被动类用类矩形表示，其中的类名没有下划线。这些矩形具有正常的边界（没有加粗）。而主动对象在显示时则使用加粗的边界。

248. path(路径)

连接各个符号的图形片断的一个连接序列，通常用来显示关系。

表示法

路径是图中各个符号之间的图形化的连接。路径被用来说明各种各样的关系，如关联、泛化和依赖。两个连接在一起的片断的结束端点重合。尽管有的工具只支持直线段和圆弧，但片断可以是直线段、圆弧或其他形状（如图 13-141）。理论上，线可以画成任意角度，尽管有的建模者希望把线限制成直角，而且还可能为了外观和摆放上更为容易，强制性地把线放到一个规则的网格内。通常，虽然路径的路线并不重要，但是，路径应该避免穿过封闭区域，因为穿过图形区域的边界可能具有语义上的含义（例如，同一个协作内的两个类的关联应该画在协作区域内，表明这是同一个协作实例内的对象的关联，而穿过某个区域的路径表明不同协作实例的对象之间的关联）。更为精确地说，路径是一个拓扑结构。准确的路线并没有语义，但是与其他符号的连接和交叉有重要意义。明确的路径布局对可理解性和美观有很大影响，而且还暗示了关系的重要性及其他事情。但是，这种考虑是基于人而非基于计算机。人们希望工具能使路径的路由和重路由变得容易。

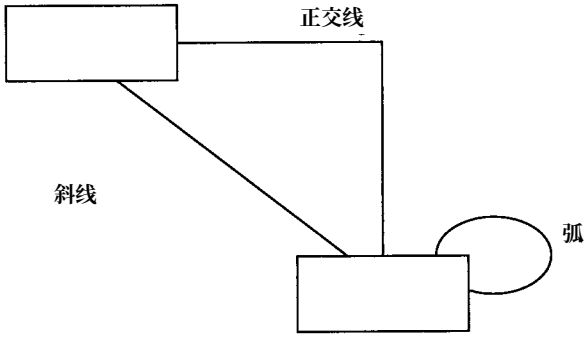
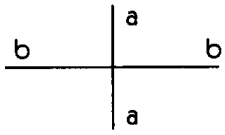


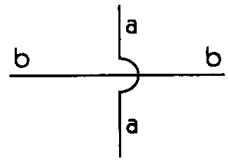
图13-141 路径

在大多数的图表中，线之间的交叉没有重要意义。为了避免交叉线的二义性，可以在交叉点上画一个小的半圆或者缺口（如图 13-142）。更为普遍的做法是，建模者将一个交叉当作两条独立的线，避免同两个直角在它们的拐角处接触的情况发生混淆。

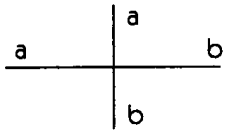
China-pub.com
下载



交叉线



明确的交叉线

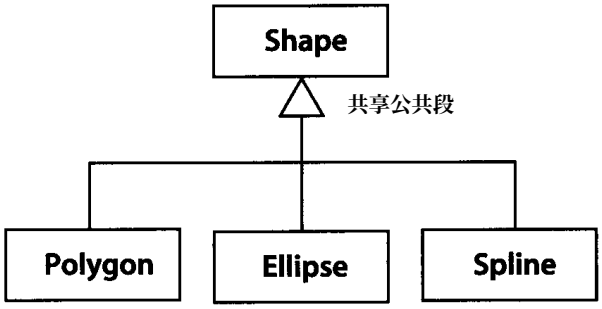


连接拐角不好
为避免交叉引起的混淆应重新安排路线。

重叠拐角

图13-142 路径交叉

在一些关系中（如聚合和泛化），相同类型的几条路径可能连接到同一个符号上。如果不同模型元素的特性匹配，则这些连接到该符号的线段可以合并成一条线段，这样从该符号引出的路径分支成为不同的路径而形成树（如图 13-143）。这完全是图形表示上的选择。概念上，单个路径都是有区别的。当不同段的建模信息不完全相同时，这种表示法可能不会被使用。



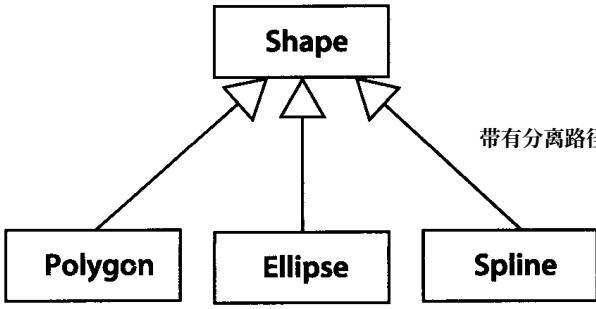


图13-143 有共享段的路径

China-pub.com 下载

249. pathname(路径名)

由嵌套的命名空间的名称连接而成的字符串。该字符串从包含整个系统的不明确的未命名的命名空间开始，以元素本身的名称结束。

语义

在一个系统内，路径名唯一地表明了一个模型元素，如属性或者状态。在表达式中它可以被用来引用一个元素。并不是每种元素都有名称。

表示法

路径名用嵌套的命名空间和元素名称的列表表示，之间由双冒号分隔。命名空间即带有嵌套声明的包或元素。

Accounting::Personnel::Employee::address

属性address 存在于包 Employee 存在于包 Personel 存在于包 Accounting

路径名称是对包中由路径前缀所命名的元素的引用。

250. pattern(模式)

模式即参数化的协作，该协作表示参数化的类元、关系和行为的集合。通过将模型（通常是类）中的元素绑定到模式的角色上去，这些类元、关系和行为可以应用到多种情况下。模式即协作模板。

语义

模式表示可以在一个或多个系统中多次使用的参数化协作。要成为一个模式，协作必须能够在多种情况下都能够应用，以便可以给它命名。在多种情况下都能工作是个待解决的问题，而模式就是对该问题的一个解答。当然，模式并不是该问题的唯一解答，但它是已经被证明非常有效的方案。大多数模式都有缺点和优点，这主要取决于更为广阔的系统的各个方面。建模者在决定使用某种模式之前应该充分考虑这些优点和缺点。

UML参数化协作表示了某种类型模式的结构和行为方面的视图。模式也包括不被 UML直接修改的其他方面，如缺点和优点的列表。这些方面大多可以用语言表达。见 [Gamma-95]可以获得对模式全方位的理解，及一些设计模式的目录。

根据模式生成协作。协作可以用来说明设计结构的实现。通过参数化它的各个要素，相同类型的协作可以应用多次。模式即参数化的协作。通常，在协作中角色的类是参数。通过将值（通常是类）绑定于它的参数，模式可被实例化为协作。对于参数化的角色，通过为每

China-pub.com
下载

一个角色声明一个类即可绑定模板。一般而言，模式中的关联角色不被参数化。当模板被绑定后，它们表示绑定到该协作上的类之间的关联 —— 即，为生成协作而绑定模板会产生额外的关联。

表示法

为了生成协作而绑定模式用虚线椭圆表示，椭圆中包含了模式的名称（如图 13-144）。从模式绑定符号到每个参与协作的类（或者别的模型元素）之间画一条虚线。每一条线都标有参数的名称。在大多数的情况下，协作中角色的名字就可以被用作参数的名称。所以，模式绑定符号可以表示出设计模式的应用，同时也可以显示出模式使用过程中出现的实际类。模式绑定通常并不能把由该绑定产生的协作的内部结构表示出来。这一点由绑定符号隐含。

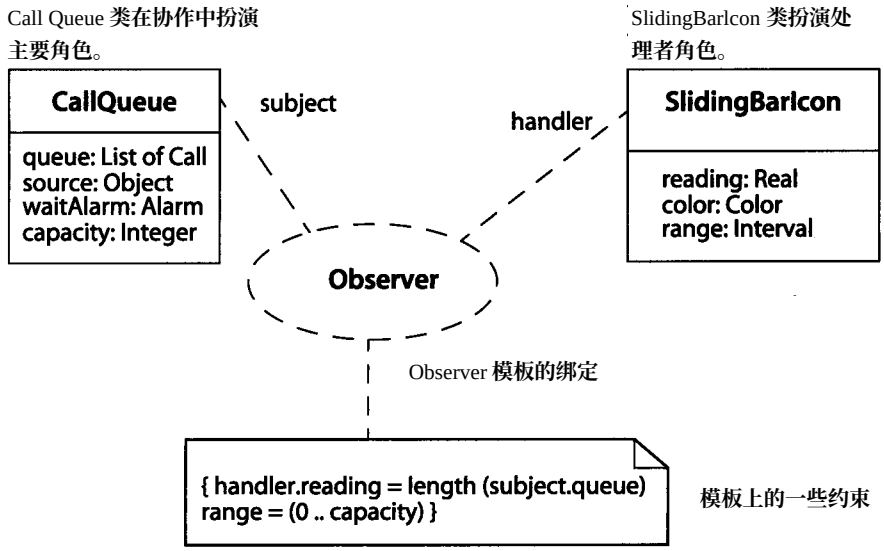


图13-144 为生成协作而绑定模式

251. permission(许可)

许可是一种依赖关系，它授权客户元素使用提供者元素的内容（受到内容元素的可见性声明的影响）。

语义

许可依赖关系的构造型是访问、友元和引入。没有构造型的单纯的许可依赖关系不存在的。访问和引入依赖关系用于包，而友元依赖关系则被用于类或者操作。

China-pub.com 下载

表示法

许可依赖关系用虚线箭头表示。箭头从客户（获得许可的元素）指向提供者（给予许可的元素），箭头上标有关键字。

标准元素

friend，import。

252. persistent object(永久对象)

永久对象即在产生它的线程停止存在后继续存在的对象。

253. polymorphic(多态)

多态是指一种操作的实现（方法或者被调用事件所触发的状态机）可能由它的的后代类来提供。非多态性的操作是一个叶操作。

见 abstract operation，generalization，inheritance，method。

语义

如果操作是多态的，那在它的后代类可能会为它提供一种方法（不论在初始类是否已经提供了方法）。否则，必须在声明该操作的类中为该操作实现一个方法，并且该方法不能在后代类里被重载。一个方法如果已经由类声明或者从祖先继承，则它可以使用。抽象操作必须是多态的 （因为它没有直接的实现）。如果一种操作被声明为叶操作，则它是非多态的。

如果一种操作在类中被声明为多态的 —— 即，没有被声明为叶操作 —— 那么它可能在后代类中被声明为叶操作。这可以阻止该操作在更远的后代类中被重载。叶操作不可以在它的后代类中被声明为多态的，它也不可以被重载。

如果方法在类中被声明并且在它的后代类中进行了重载， UML并没有指定方法结合的规则（见下面的讨论）。可以通过使用标记值以语言说明的方式来处理机制，如方法之前、之后或环绕声明。在任何情况下，动作（如显式调用继承来的方法）当然取决于动作语言。

表示法

非多态的操作通过关键字{leaf}声明。否则，操作被认为是多态的。

讨论

抽象操作必须是多态的 。否则，它根本不能被实现。 Bertrand Meyer 把这称为延迟操作，

China-pub.com
下载

因为它的说明在类中定义，而它的实现在子类中定义。这是继承在建模和编程方面一个本质，可能是最本质的应用。使用继承，操作可以应用于不同类的对象。调用者不需要知道各个对象所属的类。唯一的要求是所有的这些对象都继承自定义该操作的祖先。而祖先类不需要实现该操作，只需要定义它们的特征标记。调用者甚至不需要知道可能子类的列表。这意味着新的子类可以在以后加进来而不影响多态性操作。当新类加入时，调用操作的源代码不需要修改。在编写完初始代码之后再加入新类的能力是面向对象技术的重要基础之一。

多态的一个更有争议的使用在子类中定义的一个不同的方法来替换在类中已经定义的方法。这经常被当作一种共享形式引用，但这是非常危险的。重载不是增量式的，所以初始方法里的一切都必须复制到子方法中，即使只是一个很小的变化。这种重复有可能出错。尤其是，如果初始方法在以后有了改动，并不能保证子方法也被改动。有时子类使用一个完全不同的操作实现，但是很多专家不鼓励这种重载，因为它有潜在的危险。通常，方法应该或者没有重载地完全继承或者延迟，在后一种情况下，超类中没有实现，所以不存在冗余或者不一致的危险。

为了使子类能够扩展操作的实现而不失去继承方法，大多数编程语言提供某种形式的方法合并，使用继承方法但同时也允许加入另外的代码。在 C++中，继承方法必须显式地通过类名和操作名来调用，它把类的继承机制严格地建立在代码之上，所以并不是完全的面向对象的方法。在Smalltalk中，方法可以用 super 来调用操作，使继承方法处理这个操作。如果类的层次关系发生了变化，则继承仍然有效，只是可能这时使用的是另一个类的方法。但是，重载方法必须显式地提供对 super 的调用。错误可能会发生，而且确实在发生，因为编程人员会忘记在改动的时候插入调用。最后， CLOS提供了非常通用和复杂的自动方法合并规则，在一个操作的执行过程中可能会调用几个不同的方法。整个操作由几个段共同实现而不是被强制成为单个方法。这非常通用但是对使用者来说更难于管理。

UML没有强制只使用一种实现方法合并的办法。方法合并是一个语义变更点。任何实现办法都可以使用。如果编程语言在方法合并方面比较差，那么建模工具可以在生成合适的编程语言代码方面提供帮助，也可以在使用了重载方法而没有发现的情况下发出警告。

254. postcondition (后置条件)

后置条件即在操作完成时必须为真的约束。

语义

后置条件即在操作执行完成时必须为真的布尔表达式。它是一个断言，不是可执行语句。有时可以提前自动检验，这取决于表达式的确切形式。可以在操作完成之后对后置条件进行检测，这是调试程序的本性。条件为真，任何其他情况都是编程错误。后置条件即作用在操

China-pub.com 下载

作实现上的约束。如果它不被满足，那么操作的实现是错误的。

见invariant，precondition。

结构

后置条件被模型化为一个带有构造型《postcondition》的约束，附属于操作。

表示法

后置条件可以用带有关键字《postcondition》的注解表示。该注解附属于受影响的操作。

示例

图13-145显示了作用在数组排序操作上的后置条件。数组（a'）的新值与初始值（a）有关联。这个例子以结构化的自然语言表示。以更为正式的语言来声明也是可能的。

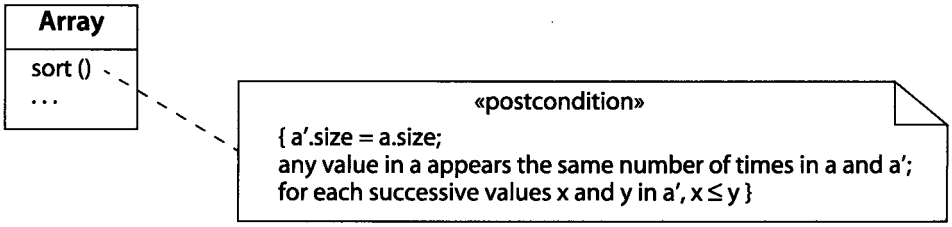


图13-145 后置条件

255. powertype(强类型)

实例是给定类的子类的元类。

见metaclass。

语义

给定类的子类本身可以被看作元类的实例。这样的元类就被称为强类型。例如，类 Tree可能有子类 Oak, Elm和Willow。作为对象来看，这些子类是元类 TreeSpecies的实例。TreeSpecies即Tree范围里的强类型。

表示法

强类型用带有构造型《powertype》的类表示。通过标有构造型《powertype》的虚线箭头，它被连接到一个泛化路径集合（如图 13-146）。

China-pub.com
下载

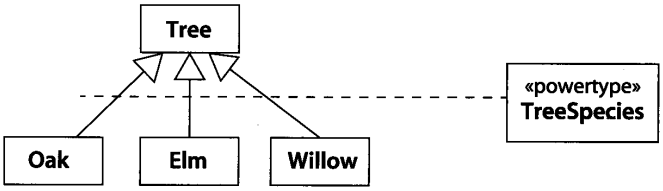


图13-146 强类型

256. precondition(前置条件)

前置条件即在操作被调用时必须为真的约束。

语义

前置条件即在操作被调用完成时必须为真的布尔表达式。满足这个表达式是调用者的职责。接收者不用去检查它。前置条件不是一个防御性的条件，它必须为真，它不是执行操作的方式。为保证可靠性，可以在操作开始时检查前置条件，这是调试程序的本性。条件为真，任何其他情况都是编程错误。如果这个条件不被满足，那么关于操作或系统的完整性不能做任何评价。它可以可靠地发现错误。实际上，接收者显式地检查前置条件可以发现许多错误。

见invariant，postcondition。

结构

后置条件被模型化为一个带有构造型《postcondition》的约束，附属于操作。

表示法

后置条件可以用带有关键字《postcondition》的注解表示。该注解附属于受影响的操作。

示例

图13-147显示了作用在矩阵乘法操作上的前置条件。

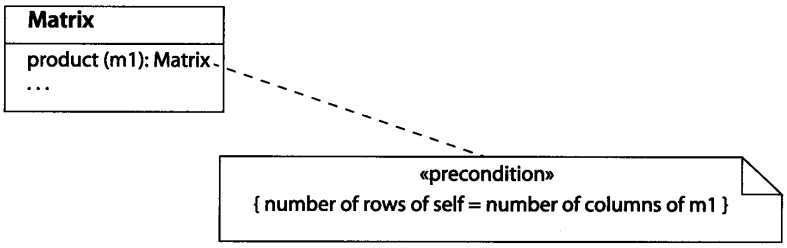


图13-147 前置条件

China-pub.com 下载

257. presentation element(表示元素)

一个或多个建模元素的文本或图形投影。

见diagram。

语义

表示元素（有时也被称为视图元素，尽管它们也包含非图形方式的表示）表示了一个模型中为人类感知的信息。它们是表示法。它们显示了模型元素的语义信息部分或者全部的。它们也有可能加入对人类有益的美学信息，例如，把概念上相互联系的元素归为一组，但是所加入的信息没有语义内容。人们希望表示元素应该做到在低层的模型元素发生变动时能够保持自身的正确，这样，模型元素不用为保证操作正确而担心表示元素。

本书UML 表示法的描述定义了从模型元素到屏幕上的图形表示的映射。表示元素作为对象的实现是工具实现的职责。

258. primitive type(简单类型)

预定义的基本数据类型，如整数或者字符串。

见enumeration。

语义

简单类型的实例没有身份。如果两个实例具有相同的表示，那么它们是无法区分的，且可以通过值来传递而不会丢失任何信息。

简单类型包括数字和字符串，也可能是别的系统所依赖的数据类型，如日期和货币，它们的语义在UML以外已经事先定义好。

人们希望简单类型能够和编程语言中的类型紧密对应。

见enumeration，它是用户自定义的数据类型而不是预定义的简单类型。

259. private(私有的)

私有是一个可见性值，它表明给定元素在它的命名空间以外是不可见的，即使是对该命名空间的后代也是不可见的。

260. private inheritance(私有继承)

私有继承是一种结构继承，这种继承没有继承行为说明。

见implementation inheritance，interface inheritance，substitutability principle。

China-pub.com
下载

语义

泛化可能具有构造型《implementation》。这就表明客户元素（通常是一个类）继承了提供者元素的结构（属性、关联和操作），但是没有必要使它自己的客户也能使用该结构。因为这样一个类（或者别的元素）的祖先对别的类是不可见的，故该类的实例不能作为变量使用，也不能作为提供者的类的参数使用。换而言之，类不能替代它私有继承的提供者。私有继承不遵守替代原理。

表示法

私有继承用泛化箭头上的关键字《 implementation》来表示，箭头从继承元素（客户）指向提供被继承结构的元素（提供者）。

讨论

私有继承只是一种实现机制，它不应该被认为是对泛化的一种应用。泛化要求可替代性。在不包含实现结构的分析模型中，私有继承不是非常有意义。即使对于实现，因为私有继承包含了对继承的非语义性质的应用，所以，对私有继承的使用也应该多加小心。通常，更为明智的选择是使用与提供者类之间的关联。许多作者（包括笔者）认为私有继承根本不能使用，因为它以非语义方式使用继承，而这种方式在模型发生变动时是非常危险的。

261. procedure expression(过程表达式)

过程表达式即其计算代表一个过程执行表达式，而这个过程会影响到正在运行的系统的状态。

语义

过程表达式是一个可执行算法的编码。它的执行可能（实际上经常是）影响系统的状态——即，它的执行具有副作用。通常，过程表达式并不返回值。它的执行目的即改变系统的状态。

262. process(过程)

1) 操作系统中一个重量级的并发和执行单元。见 thread，其中包括重量级进程和轻量级进程。如果需要，可以通过使用构造型产生实现上的区别。

2) 软件开发过程——开发一个系统的步骤和指导。

China-pub.com 下载

3) 执行一个算法或者是动态处理一些事情。

263. product(产品)

产品即开发的制品，如模型、代码、文档、工作计划。

264. projection(投影)

从一个集合到该集合的子集的映射。大多数模型和图都是从可能得到的所有信息的集合上产生的投影。

265. property(特性)

表示传递有关模型元素信息的值的一般性术语。特性具有语义效果。 UML中预定义了一些特性，其他特性是用户定义的。

见attribute，relation，tagged value。

语义

特性不但包括标记值（用户定义）和附属于元素的关系（用户定义），而且包括内嵌属性(UML定义)。从一个用户的观点来看，特性是内嵌的还是作为标记值由用户实现的，往往并不重要。

讨论

应该注意到我们是在非常普遍的意义上使用特性的，它表示附属于元素的任何可能值，包括属性、关联和标记值。在这种意义上，特性可能包括从一个给定元素开始找到的间接得到的值。

266. property list(特性列表)

特性列表是一个文本语法，用于显示特性和附属于元素的特性，尤其是标记值，还包括模型元素的内嵌属性。

表示法

花括弧内一个或者多个由逗号分隔的特性说明。每个特性声明具有下面的形式：

China-pub.com
下载

property-name = value

或：

property-literal

这里的*property-literal***是一个唯一的枚举值，它的出现代表了一个唯一的特性名。**

示例

{abstract , author = Joe , visibility = private }

表示选项

如果特性说明经过合适的标注后能够和其他信息区分开来，则工具可以在单独的行中表示特性说明，带不带括弧均可。例如，类的特性可以以一种不同的显示方式，如斜体或者不同的字体，列在类的名称下面。这是工具的问题。

注意，特性字符串可以用于表示内嵌属性和标记值，但这种使用在规范形式简单时应避免。

267. protected(受保护的)

受保护是一个可见性值，它表示给定元素在它的命名空间外只对它的命名空间的后代可见。

268. pseudoattribute(伪属性)

与行为类似属性的类相联系的值，即它的每个实例都有唯一的值。

见discriminator、rolename。

语义

伪属性包含关联角色名和泛化鉴别器。关联角色名是类中关联的另一端的伪属性。泛化鉴别器是父元素中的伪属性。在每个子元素里，鉴别器的值即子元素的名称。

在表达式中，伪属性可以被作为一个名称以便从对象中取得一个值。因为属性名和伪属性名都可以用在表达式里，所以它们处在相同的命名空间且在这个命名空间中唯一。对于继承的属性和伪属性名，它们的名称也必须保证唯一。

269. pseudostate(伪状态)

伪状态是状态机中具有状态的形式而其行为却不同于完全状态的顶点。

China-pub.com 下载

见history state，initial state，junction state，stub state。

语义

当一个伪状态处于活动的时候，状态机还没有完成它的运行到完成步骤，也不会处理事件。伪状态包括初始状态、结合状态、桩状态和历史状态。伪状态用来链接转换段，且到一个伪状态的转换意味着会有一个到另一个状态的自动转换而不需要事件触发。

终止状态和同步状态不是伪状态。它们是特殊的状态，当状态机完成运行到完成步骤时仍然保持活动，但是从它们出发的转换存在限制。

270. public(公共的)

公共是表示给定元素在它的命名空间以外仍然可见的可见性值。

271. qualifier(限定符)

二元关联上的属性或者属性列表的槽，在此关联中，属性的值从整个对象集合里选择一个唯一的关联对象或者关联对象集。限定符是关联的遍历索引。

见association class，association end。

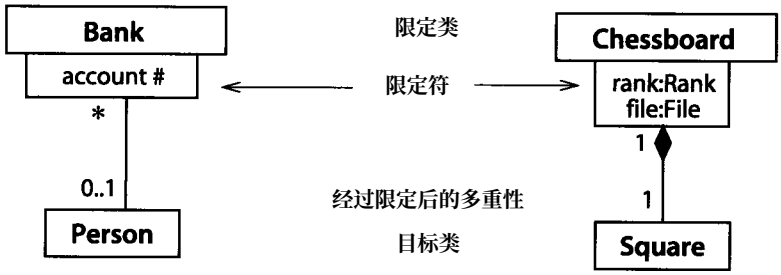
语义

二元关联把一个对象映射到相关对象的集合上。有时通过提供一个能够区分集合中其他对象的值，可以把一个对象从集合里选取出来。这个值可以是目标类的一个属性。但是，这个值通常可能是关联本身的一部分，即其值是在将新的链添加到关联类时创建者提供的关联属性。这样一个二元关联上的属性即限定符。一个对象连同一个限定值决定了一个唯一的相关对象或者相关对象子集。该值限定了关联。在一个实现语境中，这样的属性被称作索引值。

限定符被用来从与一个对象（被称作被限定对象）相关的对象集合中选择一个或者多个对象（如图13-148）。被限定符值选中的对象称为目标对象。限定符总是作用于在目标方向上多重性为多的关联。在最简单的情况下，每个限定符只从目标对象集中选择一个对象。换而言之，一个被限定对象和一个限定符值产生一个唯一的相关目标对象。给定一个限定对象，每个限定符值映射到一个唯一的目标对象。

很多种类的名称都是限定符。在一定的语境中这样一个名称与一个唯一的值映射。限定对象提供语境，限定符是名称，而目标对象是结果。任何 ID或者别的唯一代码都是限定符，目的是为了唯一地选定一个值。数组可以被设计成一个限定关联。数组是限定对象，数组下标是限定符，而数组元素是目标对象。对于一个数组，限定符的类型是整数范围。

China-pub.com
下载



(Bank, account#)→0或1个Person

Person→多个(Bank, account#)

(Chessboard ,rank, file)→1个Square

Square→1个(Chessboard ,rank, file)

图13-148 限定关联

限定符可以被用在一个导航表达式里以选择通过关联与对象相关的一个对象子集——即那些具有限定符属性值或者值列表的关联。限定符在通过关联相关的对象集中充当选择器。在大多数情况下，限定符的目的是为了从相关对象集中选择出一个唯一的对象，以使限定关联的行为像一个查询表。

结构

限定符。限定符属性是二元关联端的一个可选部分。限定符限定了附属于关联的类。类的对象和限定符值从二元关联的另一端的类中选择一个对象或者对象集。二元关联的两端都有限定符是可能的，但是这种情况很少见。

限定符是一个关联属性或者属性列表。每个属性具有一个名称和一个类型但是没有初始值，因为限定符不是独立对象，当向关联中增加链时，每个限定符值都必须说明。

限定符不能用在n元关联中。

多重性。限定关系的多重性被放在二元关联限定符的相对的一端（应该记住限定类和限定符一同形成一个与目标类相关的组成值）。换而言之，限定符附属于关联的“近端”，而多重性和角色名附属于“远端”。

附属于目标关联端的多重性表示通过一个（源对象，限定符值）对可以选择多少个对象。普通的多重性值包括0..1（可以选择一个唯一的值，但是每个可能的限定符值不一定选择一个值）、1（每个可能的限定符值选择一个唯一的目标对象，所以限定符值的取值域必须是有限的）和*（限定符值是把目标对象划分为子集的索引）。

在大多数情况下，多重性是0或1。这种选择意味着一个对象和一个限定符最多可能产生一个相关对象。多重性为1意味着每个可能的限定符值恰好产生一个对象。这显然要求限定符的取值域必须是有限的（至少在计算机实现范围内）。多重性在映射有限的枚举类型方面是有用的——例如，由PrimaryColor (取值为red, green和blue)限定的 Pixel 可以为图像中的每个点产生由 red-green- blue 值组成的三位字节。

China-pub.com 下载

未限定关联的多重性没有显示说明。但一般假定它们的多重性为多，至少是多于一个。否则，不需要限定符。

限定关联上多重性为多没有很重要的语义影响，因为限定符没有减少目标集合的多重性。这样的多重性代表了一个设计要求：必须提供一个索引以遍历这个关联。此时，限定符把目标对象集合分成各个子集。在语义上，除了有一个关联属性外没有增加任何东西，这个关联属性也划分链（隐式的）。在一个设计模型中，限定符的设计必须作到遍历是高效的 —— 即，不能要求在所有目标值中做线性搜索。通常通过某种类型的查询表来实现。数据库或者数据结构中的索引可作为限定符。

在限定关联的相反方向（即，从目标类到限定对象），多重性表示与目标对象相关的（限定对象，限定符）对的数目，而不是限定对象的数目。换而言之，如果几个（限定对象，限定符）对映射到相同的目标对象，那么反向多重性为多。值为 1的从目标到限定符的反向多重性意味着只有一个限定对象和限定符值对与目标对象相关。

表示法

限定符用附属于关联路径端的一个小矩形表示，其中路径介于最终路径段和限定类的符号之间。限定符的矩形是路径的一部分，而不是类的一部分。限定符附属于它所限定的类 —— 即，限定类的一个对象连同一个限定符值在关联的另一端唯一地选择一个目标类对象的集合。

限定符属性列在限定符的框中。列表中可能有一个或者多个属性。除了初始值表达式没有意义之外，限定符属性和类属性具有相同的表示法。

表示选项

限定符不可能被隐藏（它提供了重要的细节，忽略这些细节会改变关系的内在性质）。

工具可以使用更细的线来表示限定符的矩形，以便清楚地和类的矩形区分。

限定符的矩形最好比它所附属的类的矩形要小，虽然实际情况并不总是这样。

讨论

限定关联上的多重性处理起来就像限定对象和限定符是一个单一的实体，一个组成键。在正方向中，目标端的多重性表示与组成值（限定对象 +限定符值）相关的对象的数目。在反方向中，多重性刻画了与目标对象相关的组成值的数目，而不是与各个目标对象相关的限定对象的数目。这就是为什么限定符被放在关联路径中靠近类符号的那一端的原因 —— 你可以认为关联路径把组成值连接到目标类上。

不能预先给未限定关系声明多重性。但是实际上，在正方向上，它的多重性往往是多。除非很多目标对象被联系到一个限定对象上，否则限定一个关联毫无意义。对于逻辑建模，限定符的目标是通过添加限定符把多重性减为 1，这样可以保证查询时，只返回一个单一的值，而不是一个值的集合。限定符值的唯一性通常是一个重要的语义条件。几乎所有的应用都有

China-pub.com
下载

很多限定关联。很多名称是限定符。如果一个名称在某些语境中是唯一的，那么它是一个限定符，而且这个语境应该被正确地识别和建模。并不是所有的名称都是限定符。例如，人名就不是限定符。因为人名是含义模糊的，所以大多数数据处理应用程序使用某种识别数码，如顾客号码、社会保险号码或者是雇员号码。如果应用程序要求查询信息或者在查询关键字的基础上遍历数据，则模型应该使用限制关联。任何语境，如果其中的名称和识别代码被定义为从集合中做出选择，则它们通常应该定义为限定关联。

应该注意到，限定符值是一个链而不是目标对象的特性。考虑一下 Unix 文件系统，每个目录都是一系列入口，这些入口在这个目录中是唯一的，尽管相同的名称可以在别的目录中使用。每个入口指向一个文件，这个文件可以是数据文件或者是另一个目录。可以有多个入口指向相同的文件，此时，这个文件有多个别名。 Unix系统被设计成多对一的关联，在这个关联中，文件名所限定的目录产生一个文件。应该注意到文件名不是文件的一部分，它是目录和文件之间关系的一部分。文件并不是只有一个名称。可能在很多目录中有很多名称（甚至是在相同的目录中有几个名称）。名称不是文件的属性。

限定关联的主要动机是满足设计具有自然而且重要的实现数据结构的重要语义状态的需要。在正方向中，限定关联是一个查询表 —— 对于一个限定对象，每个限定符值产生一个目标对象（如果值集中没有限定符值则为空值）。查询表通过数据结构来实现，如散列表、B-树、比必须线性查询的无序列表有效得多的有序列表。在几乎所有情况中，使用链接表或者其他无序数据结构来查询名称或者代码，这种设计不好，但不幸的是很多程序员这样做。使用限定关联来为合适的状态建模及使用高效的数据结构来实现它们，对于好的编程很重要。

对于一个逻辑模型，在正方向上使用取值为多的多重性没有太多意义，因为实际上限定符并不能增加任何关联属性无法显示的语义信息。但是在设计算法和数据结构的模型中，限定符带有另外的含义 —— 即使选择更为有效。换而言之，限定关联表示了为查询在限定符值上做了优化的带有索引的数据结构。在这种情况下，如果存在一些必须通过一个普通的索引值访问而不需要搜索其他值的值，则取值为多的多重性可以表示这些值集合。

通常限定符属性不应该包含在目标类的属性中，因为它在关联中的出现是多余 的。但是在索引值的情况下，可能有必要选择一个本身就是目标类的属性的值，把它作为冗余的限定符。索引值固有地就是冗余的。

约束

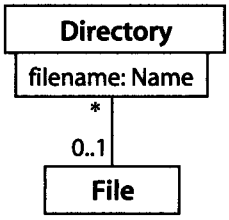
有些复杂的状态并不能用非冗余的关系集来直接建模。最好的方法是使用限定关联加上显式说明的附加约束来捕捉基本的访问路径。因为这种状态并不普遍，所以我们认为试图找到一个能够直接捕捉所有多重性约束的表示法弊大于利。

例如，考虑一个文件名唯一地确定一个文件的目录。一个文件可能对应多个目录 -文件名对。这就是前面所述的基本模型。图 13-149 显示了这个模型。

现在，我们想加入附加的约束。假定一个文件必须仅在一个目录里，但是在该目录中，

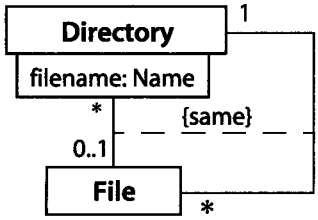
China-pub.com 下载

它可以有很多个名称——即，有多种方法来命名相同的文件。可以在 File 和 Directory 之间建立一个冗余的关联，Directory 上的多重性为 1（如图 13-150）。这两个关联的冗余通过约束{ same }表示，约束{same} 表示这两个元素是相同的但是在不同的细节级别。因为这些关联是冗余的，所以只有限定关联才能实现，而其他关联则被作为作用在其内容上的运行时约束。



（目录，文件名）→0或1个文件
目录→多个文件
文件→多个（目录，文件名）
文件→多个目录
文件→多个文件名

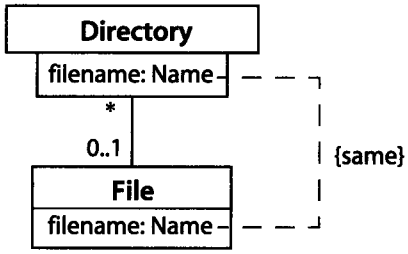
图13-149 简单限定符



（目录，文件名）→0或1个文件
目录→多个文件
文件→多个（目录，文件名）
文件→一个目录
文件→多个文件名

图13-150 在一个目录下具有多个名称的文件

一个为人所熟悉的约束是每个文件可能出现在多个目录里，但是不论它在何处出现，都必须具有相同的名称。其他文件可能具有相同的名称，但是它们必须出现在不同的目录中。这可以通过把 Filename 作为 File 的一个属性而同时又把类属性和限定符约束为相同来建模（如图 13-151）。这种模式经常作为一个搜索索引发生，尽管在一个普通的索引中，限定目标的多重性为多。所以，这种情况比索引具有更多的语义含义，因为索引只是一个实现工具。



（目录，文件名）→0或1个文件
目录→多个文件
文件→多个（目录，文件名）
文件→多个目录
文件→1个文件名

图13-151 在所有目录中具有相同名称的文件

China-pub.com
下载

第三种情况是允许一个文件以不同的名称出现在多个目录里，但是这个文件在单个目录里只能出现一次。对于这种情况，可以使用冗余的限定关联和共享相同属性 filename的关联类来建模（如图13-152）。

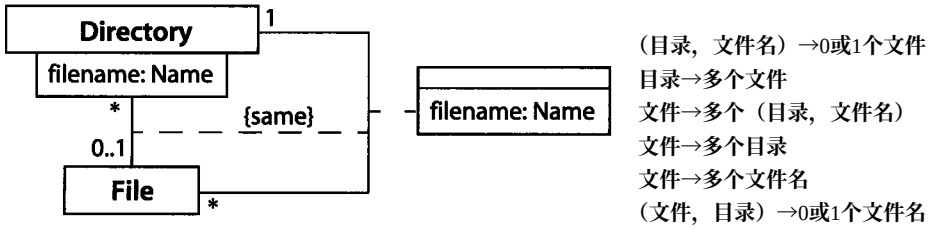


图13-152 在任何目录中最多只有一个名称的文件

这些例子连同冗余关系说明了约束的本质。实际上，较为令人满意的做法是以文本的方式声明约束，而以图形的方式声明限定关联。

272. query(查询)

查询是返回一个值但是不会改变系统状态的一种操作，是没有副作用的操作。

273. realization(实现)

实现是规格说明和其实现之间的关系。它表示不继承结构而只继承行为。

见interface。

语义

规格说明描述了某种事物的行为和结构，但是不决定这些行为如何实现。而实现则提供了如何以高效的可计算的方式来实现这些行为的细节。规格说明行为的元素和实现行为的元素之间的关系叫做实现。通常有很多方式来实现一个规格说明。一个元素可以实现多个规格说明。所以实现是元素之间多对多的联系。

实现的含义是客户元素必须提供者服务元素的所有行为，但是没有必要和它的结构或者实现相匹配。例如，一个客户类元必须支持提供者类元的所有操作，同时它也必须支持所有声明提供者类元的外部行为的状态机。但是任何提供者类元的说明实现的属性、关联、方法或者状态机都不和客户类元发生联系。应该注意到客户实际上并不从提供者那里继承操作。客户必须自己声明或者从祖先那里继承这些操作以覆盖提供者的所有操作。换而言之，实现中的提供者表明客户必须有哪些操作，但是由客户提供它们的操作。

China-pub.com 下载

某些种类的元素，如接口和用例，是用来说明行为的，它们不包含任何的实现信息。别的元素，如类，是用来实现行为的。它们包含实现信息，但是也可以以一种更为抽象的方式把它们作为说明符来使用。通常，实现把规格说明元素，如用例或者接口，连接到实现元素上，如协作或者类。也可能使用实现元素，如类，来进行规格说明，可以把它放在实现中规格说明的一边。在这种情况下，只有提供者的规格说明部分影响客户。实现部分和实现联系是不相关的。更为精确的说，实现就是两个元素之间的关系，而在这个关系中，其中一个元素的外部行为规格说明部分影响另外一个元素的实现部分。这也可以认为是只继承行为规格说明而不继承结构或者实现（由客户声明操作的需要）。

如果规格说明元素是一个抽象类，没有属性，没有关联，只有抽象操作，那么这个抽象类的任何规格说明实现这个抽象类，因为除了规格说明外没有其他可以继承。实现元素必须支持规格说明元素的所有行为。例如，一个类必须包含它所实现的接口的所有操作和语义，这些语义与接口要求的所有规格说明相符。类可以实现额外的操作，而操作的实现也可以做额外的事情，只要没有违反接口操作的规格说明。

表示法

实现关系用一个虚线路径表示，其中在靠近提供规格说明的元素的一端带有封闭三角形箭头,路径的末端在提供实现的元素一边（如图 13-153）。

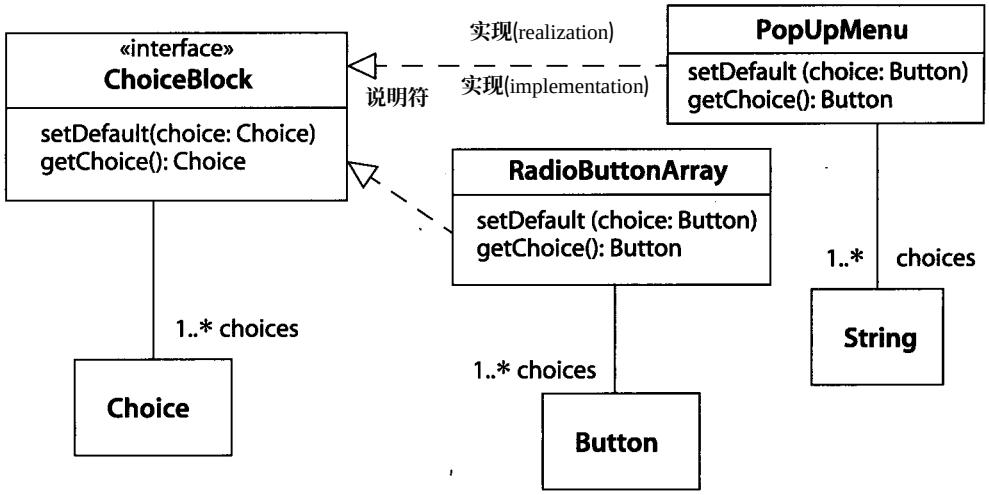


图13-153 实现关系

讨论

另外一种重要情况是通过协作实现用例（如图 13-154）。一个用例声明了外部可见的功能

China-pub.com
下载

和行为顺序，但是它没有提供实现。协作描述实现用例行为的对象以及这些对象为了实现这些行为而相互作用的方式。通常，一个协作实现一个用例，但是协作可以用附属的协作来实现，每个附属协作完成一部分工作。实现一个协作的对象和类通常也出现在别的协作中。协作中的每个类向被实现的用例贡献一部分功能。所以，一个用例最终是由几个类通过片来实现的。

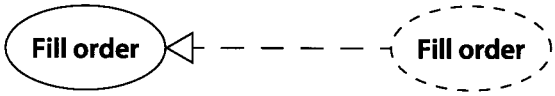


图13-154 通过协作实现用例

274. realize(实现)

实现就是为声明元素提供实现。

见realization。

275. receive (接收)

接收即处理从发送者对象传送过来的消息实例。

见sender，receiver。

276. receiver(接收者)

接收者即处理从发送者对象传送来的消息实例的对象。

277. reception(接收)

类元准备对信号接收做出反应的声明。它是类元的一个成员。

语义

接收即类元准备接收信号实例的声明。接收类似于一个操作。接收声明了类元支持的消息的特征标记，并且说明了它的含义。

结构

接收具有下面的属性：

China-pub.com 下载

多态　　类元对信号的响应是否总是相同的。由特性 IsPolymorphic 使用下面的值来编码：

true　　响应是多态的：响应取决于状态，并且可以被后代重载。

false　　不论处于何种状态，响应必须是相同的，而且不能被后代重载。净效果是在处理这个事件的状态机上必须存在一个转换。

信号　　指定类元准备响应的信号。

说明　　说明信号的接收所带来的效果的表达式。

表示法

接收可以显示在类或者接口的操作列表里，使用操作的语法，在信号的名称前面加上关键字《signal》。

还有一种方法，信号特征标记的列表可能被放在它自己的分栏中，分栏具有名称 Signals。这两种方式都显示在图13-155中。

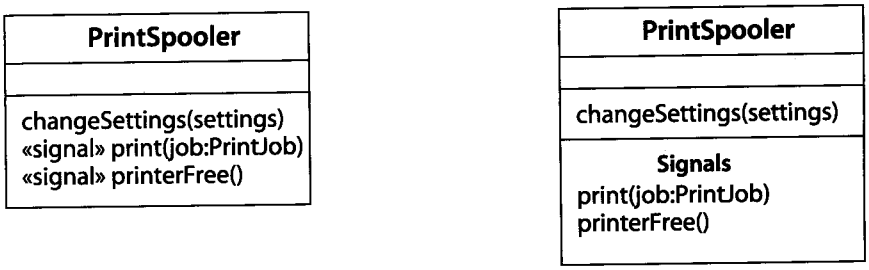


图13-155　表示信号接收的两种方法

278. reference (引用)

引用是模型元素的一种表示，通常被称为指针。

语义

模型元素通过两种元关系相连：拥有和引用。拥有关系是元素和其组成部分之间的关系。拥有关系形成一棵严格的树。被包容的元素附属于包容元素。拥有关系、配置控制和模型的存储都是建立在包容层次上。

引用是相同细节层次上的元素之间的关系，或者是处于不同包容器中的元素之间的关系。例如，引用是一个关联和其参与类之间的关系，是一个属性和它的类或者数据类型之间的关系，是一个范围模板和它的参量值之间的关系。为了使引用成为可能，执行引用的元素必须对被引用元素是可见的。通常这意味着包容引用源的包必须对包容引用目标的包具有可见性。

China-pub.com
下载

这就要求在包之间应该存在合适的访问或者引入关系。还要求被引用元素必须具有一个可见性设置，以允许它在它自己的包之外是可见的，除非引用源在同一个包中。

注意引用是一个内部元模型关系，不是用户可见的关系；它被用来创建别的关系。

279. refine(精化)

表示法中精化依赖关系的关键字。

280. refinement (精化)

精化表示对已经在一定细节水平或者在一个不同的语义水平上做了说明的事物做更为全面的说明的关系。

见abstraction。

语义

精化即在具有映射关系（不必要是完整的）的两个元素之间的一个历史或者可计算的连接。通常，这两个元素处在不同的模型中。例如，一个设计类可能是一个分析类的精化，它可能具有相同的逻辑属性，但是它们的类可能来自一个特定的类库。但是，一个元素也可能精化一个相同模型中的元素。例如，一个经过优化的类是简单但不是非常有效的类的精化。精化关系可能包含映射关系的描述，这种映射可能使用一种正规语言编写的（如 OCL或者一种编程语言或者逻辑语言），也可能是非正式的文本（显然，它排除了任何自动计算但是可能对初期的开发有用）。精化可以用来为阶段性开发、优化、转变和框架细化建模。

结构

精化是一种依赖关系。它把一个客户（更为发展的元素）连接到一个提供者（作为精化基础的元素）。

表示法

精化用带有关键字《refine》的依赖关系箭头（一个头部在一般性元素，尾部在特殊元素上的虚线箭头）表示。映射关系可以通过一条与注解相连的虚线附加到依赖性路径上。已经提出了各种各样的精化，可以用更进一步的构造型来表示。在很多情况下，精化连接处在不同模型中的元素，所以在图形方式下是不可见的。通常它是隐含的。

示例

优化是一种典型的精化。图 13-156 显示了一个棋盘，它在分析模型里具有简单的表示，

China-pub.com 下载

但是在设计模型里有更为精细和模糊的表示。设计类不是分析类的特化，因为设计类具有一个完全不同的形式。分析模型里的类和设计模型里的类具有相同的名称，因为它们代表不同语义水平上的同一个概念。

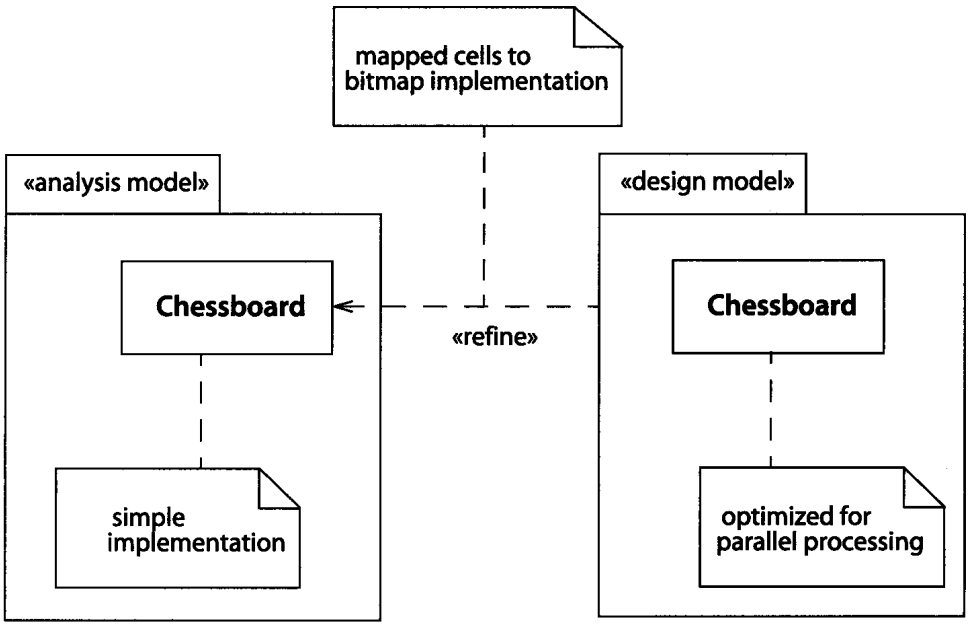


图13-156 精化

281. reification(具体化)

具体化某事物的动作。

见 reify。

282. reify (具体化)

把通常不被当作对象的事物当作对象处理。

讨论

具体化具有长久的哲学和文学含义。 它把抽象概念的特征描述为神话和诗歌里的事物和人。例如，神 Thor 这种称呼是对雷的具体化。柏拉图的理想主义理论使人们普遍流行的观念

China-pub.com
下载

得到了改变。他把纯粹的概念，如美、善、勇气等，认为是真正永恒的现实，而把物理实体作为不够完美的复制品——具体化达到了它的最终极限。

具体化是面向对象中最有用的观点之一，它几乎在建模的每个方面都起着基础的作用。建立一个模型首先要求把对象放到连续的世界中去。人类很自然地以他们说话的顺序做这件事——名词是一个事物的具体化，而动词是对一个动作的具体化。在模型和开始没有作为对象处理的程序，如动态行为中，具体化尤其有用。大多数人把操作当作一个对象，但是操作的执行（这个词本身就是一个具体化）呢？通常人们把执行当作是一个过程，将其具体化并给它一个名称——激活——你可以立刻赋特性给它，形成同别的对象的关系，操纵它，或者存储它。行为的具体化把动态的过程转化为可以存储和操纵的数据结构。这对于建模和编程是非常有力的概念。

283. relationship(关系)

模型元素之间具体化的语义连接。关系的种类包括关联、泛化、元关系，流以及几种依赖关系。

语义

表13-2 显示了各种UML的关系。第一列（类型）显示了它们在元模型中的分组；第二列（变体）显示了不同的关系种类；第三列（表示法）显示了关系的基本表示法：关联是实心路径，依赖是虚线箭头，泛化是带有三角形箭头的实心路径；第四列（关键字）显示了关键字和附加的语法。

表13-2 UML关系

类型	变体	表示法	关键字或符号
抽象	导出	依赖	《derive》
	实现	实现	- - - - ▷
	精化	依赖	《refine》
	跟踪	依赖	《trace》
关联		关联	————
绑定		依赖	《bind》(*parameter*$_{list}$)
扩展		依赖	《extend》(*extension point*$_{list}$，)
流	成为	依赖	*sequence-number*：《become》
	复制	依赖	*sequence-number*：《copy》
泛化		泛化	———▷
包含		依赖	《include》
元关系	实例	依赖	《instanceOf》
	强类型	依赖	《powertype》
许可	访问	依赖	《access》
	友员	依赖	《friend》

China-pub.com 下载

(续)

类 型	变 体	表 示 法	关键字或符号
许可	引入	依赖	《import》
使用	调用	依赖	《call》
	实例化	依赖	《instantiate》
	参数	依赖	《parameter》
	发送	依赖	《send》

284. repository(仓库)

模型、接口和实现的存储地，是操纵开发制品的环境的一个组成部分。

285. request(请求)

发送给实例的激励的声明。它可以是一个操作的调用或者是一个信号的发送。

286. requirement(需求)

系统期望的特征、特性或者行为。

语义

文本需求可以建模成带有构造型《requirement》的注释。

讨论

术语“需求”是一个自然语言词汇，它与意图说明系统的期望特征的各种 UML结构对应。最常见的是，与用户可见的事务对应的需求被作为用例捕获。非函数化的需求，如执行和质量规格，可能被作为文本声明捕获，它们最后跟踪到最终设计的元素。 UML注释和约束可以被用来代表非函数化的要求。

287. responsibility(职责)

类或者别的元素的契约或义务。

语义

职责可以用注释上的构造型表示。注释附属于具有职责的类或者别的元素上。职责表达

China-pub.com
下载

成一个文本字符串。

表示法

职责可以表示在类元符号矩形里的命名分栏中（如图 13-157）。

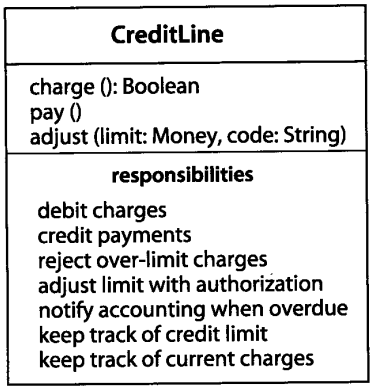


图13-157 职责分栏

288. reuse (重用)

预存在制品的使用。

289. role(角色)

位于对象结构中的命名槽，表示处在特定语境中的元素的行为。角色可以是静态的（如关联端点），也可以是动态的（如协作角色）。协作角色包括类元角色和关联角色。

见collaboration。

290. rolename(角色名)

角色名即关联中特定关联端点的名称。

见pseudoattribute。

语义

角色名提供了一个名称，不但用于在关联的对象之间导航，而且还被用来命名区分关联

China-pub.com 下载

端点。因为角色名可以在这两种互补的方式下使用，所以角色名必须在两个命名空间里同步地保持唯一。

关联中所有的角色名必须是不同的。在一个自关联 (多次包含相同类的关联)中，角色名必须消除附属于相同类的端点的歧义。否则，角色名是可选的，因为类名可以用来消除端点的歧义。

角色名也可用来从一个对象导航到相邻的相关对象。每个类都能看到附属于它的关联，而且可以用它们来找到与其实例相关的对象。习惯上，附属于相邻类的关联端点之上的角色名可用来形成一个导航表达式，以访问此关联上的对象或对象集。在图 13-158里，类B通过一个单对多的关联与类 A关联，通过一个单对单的关联与类 C关联。给定类B的一个对象bb，表达式bb.theA产生类A的一个对象集，而表达式 bb.theC产生类C的一个对象。实际上，位于关联远端的角色名像类的一个伪属性 —— 即，它可以用在访问表达式中遍历整个关联。

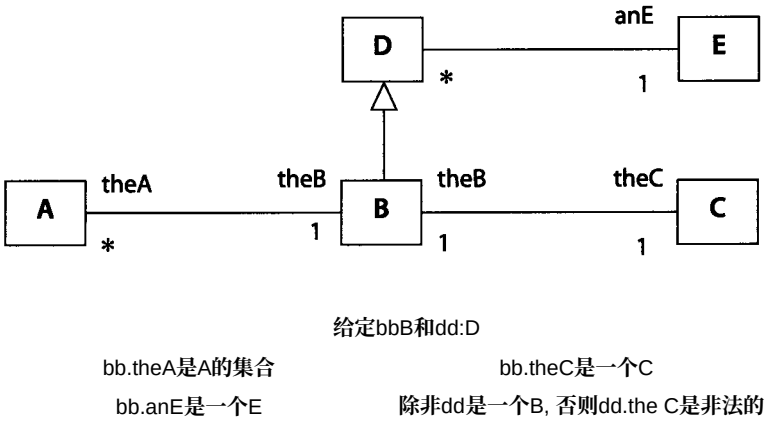


图13-158 关联之上的导航

因为角色名可以像属性名一样被用来提取值，所以角色名进入处在关联远端的类的命名空间。它和属性名具有相同的命名空间。在命名空间里，角色名和属性名都必须是唯一的。属性和关联角色名被继承，属性名和伪属性名在被继承的名称里必须保持唯一。附属于祖先类的角色名可以被用来在后代里导航。在图 13-158里，表达式bb.anE是合法的，因为类B 从类D里继承了角色名anE。

如果每个关联可以被唯一地确定，则角色名和关联名是可选的。关联名或者其端点上的角色名可以确定一个关联。没有必要同时具有这二者，尽管这样也是可以的。如果它是两个类之间的唯一的关联，则关联名和角色名都可以省略。原则上，导航表达式要求一个角色名。实际上，工具可以从关联类的名称为生成隐含的角色名提供一个缺省规则。

表示法

角色名通过放置在与一个类框相交的关联路径端点的图形字符串来表示。如果角色名存

China-pub.com
下载

在，则它不能被省略。

角色名可能带有一个可见性标记 —— 一个箭头 —— 表明在关联远端的元素是否可以看到附属于角色名的元素。

291. run time(运行时间)

计算机程序执行的时间段。比较：modeling time。

292. run to completion(运行到完成)

在其整体中必须完成的转换或者动作序列。

见action，atomic，state machine，transition。

语义

在一个状态机里，一定的动作或者动作序列是原子的 —— 即，它们不能被其他动作终止、放弃或者中断。当一个转换激发时，所有附属的动作和被它激发的动作必须作为一个组被完成，包括在它进入和离开的状态上的进入动作和退出动作。转换的执行被称为运行到完成，因为它不会等待接收其他事件。

运行到接收的语义可以和普通状态的等待语义进行比较。当一个状态为活动状态时，事件可能会引发一个到其他状态的转换。在这个状态里的任何活动都被这个转换所放弃。

一个转换可能由多个段组成，这些段构成一个链并且由伪状态分隔。几个链可能会合并，也可能会分开，所以整个模型可能包含由伪状态分隔的段的图。链中只有第一个段可以有一个触发器事件。当这个触发器事件被状态机处理的时候，转换被触发。如果所有段上的监护条件都被满足，则转换发生，并且如果没有其他转换激发，则它激发。连续段上的动作被执行。一旦执行开始，链中所有段上的动作都必须在运行到完成步骤完成之前完成。

在一个运行到完成转换的执行过程中，引发转换的触发器事件作为当前事件对动作是可用的。入口和出口动作可以包含触发器事件的参量。各种各样的事件可以引起入口或者出口动作的执行，但是动作可以在一个 case 语句里区别出当前事件的类型。

鉴于动作的运行到完成语义，它们应该被用于对分配、检验标志、简单的算术以及其他种类的操作建模。长时间的计算应该建模成可以中断的活动。

293. scenario (脚本)

脚本即说明行为的一系列的动作。脚本可以用来说明交互或者用例实例的执行。

China-pub.com 下载

294. scope (作用域)

作用域类元成员，如属性、操作或者角色名的外延。它是在每个实例中代表一个值还是代表这个类元的所有实例的一个共享值。当不带有限定而单独使用时，表示所有者作用域。

见creation，owner scope，target scope。

语义

任何作用域都是所有者作用域或者目标作用域。

所有者作用域。表示如果在整个类里有一个给定名称的槽，是否每个类的实例都有一个不同的属性。

instance　每个类元实例有它自己属性槽的不同副本或者它自己的关联对象的集合。在一个槽里的值独立于其他槽里的值。这是缺省情况。

对于一个操作而言，这个操作作用于单独的对象（一个普通的操作）。

class　类元本身有属性槽的一个副本或者关联对象的一个集合。类元的所有实例共享对一个槽的访问。

对于一个操作而言，这个操作作用于整个类，如一个创建操作或者返回关于整个实例集合的数据的操作。这样的操作不能作用于单个的实例。

目标作用域。表示一个属性的值或者关联里的目标值是否是实例（缺省）或者类元。

instance　每个属性槽或者每个关联的链都包含对目标类元的实例的引用。链的数目由多重性限制。这是缺省情况。

class　每个属性槽或每个关联链都包含对目标类的一个引用。关联中的信息在建模时间确定并在运行时间不改变，并且它不需要存储在每个对象里。实际上，链包含类自身，而不是实例。这可能对一些实现信息有用，但是对于大多数建模目的，这种能力可以被忽略。

讨论

类作用域属性或者关联为整个类提供全局的值，应该小心使用或干脆不使用，尽管大多数面向对象的语言都提供这一功能。问题在于它们包含全局信息，违反了面向对象的设计原则。另外，全局信息在分布式系统里易出问题，因为它强制集中式访问而类对象可能分布在许多机器上。最好是引入显式的对象以保存需要的共享信息，而不是把类作为带有状态的对象使用。模型和费用都是明显的。

构造函数（创建操作、工厂操作）必然具有类级源作用域，因为没有它们可以操作的实例。这是必要而正确的类作用域的使用方法。其他类级源作用域操作具有和属性一样的困难——即，它们暗含关于类实例的集中全局信息，这在分布式系统中是不实际的。

目标作用域的可用性具有限制，只能被用在特殊的情况下——通常，只用于细节性的编

China-pub.com
下载

程目的。

295. self-transition (自转换)

源状态和目标状态相同的转换。它被认为是一个状态改变。当它激发时，源状态退出然后重新进入，所以入口动作和出口动作被激发。自转换不同于内部转换，因为在内部转换里没有状态的改变发生。

296. semantic variation point(语义变更点)

元模型语义发生变更的点。它为元模型语义的解释提供了一定的自由。

讨论

相同的执行语义并不是对所有应用都适合。不同的编程语言以及不同的目的请求语义的变更，有的较微妙，有的较大。语义变更点是个问题，因为不同的建模者和执行环境对特定语义的理解不能达成一致。通过识别和命名语义变更点，可以避免一个系统的“正确”语义的争论。

例如，是否允许多重多类或动态多类的选择即语义变更点。每种选择都是一个语义变更。其他语义变更点的例子包括一个调用是否可以返回多个值以及类在运行时间是否作为实际对象存在。

297. semantics(语义)

某事物涵义及行为的正式说明。

298. send(发送)

发送者对象生成一个信号实例并把它传送到接收者对象以传送信息。

发送使用依赖关系把发送信号的操作或方法或者包含这样的操作或方法的类与接收这个信号的类联系起来。

见signal。

语义

发送是对象能够执行的特殊操作。它说明一个待发送的信号、信号的参量列表，以及接

China-pub.com 下载

收信号的目标对象的集合。

对象发送一个信号到一个对象集合 —— 通常是包含单一对象的集合。广播可以认为是发送一个信息到所有对象的集合，尽管实际上为了提高效率广播可能以特殊的方式实现。如果目标对象的集合包含多个对象，则信号的副本会同时发送给集合中的每个对象。如果集合是空的，则没有信号被发送。这不是个错误。

创建一个新对象可以被认为是发送一个消息到工厂对象，如类，这个对象创建新的实例并把消息发送给它作为它的“出生事件”。这为创建者同它的创建物之间的通信提供了一种机制 —— 出生事件可以认为是从创建者到新对象，同时带有实例化新对象的副作用。图 13-159 显示了使用文本语义和图形语义的对象创建。

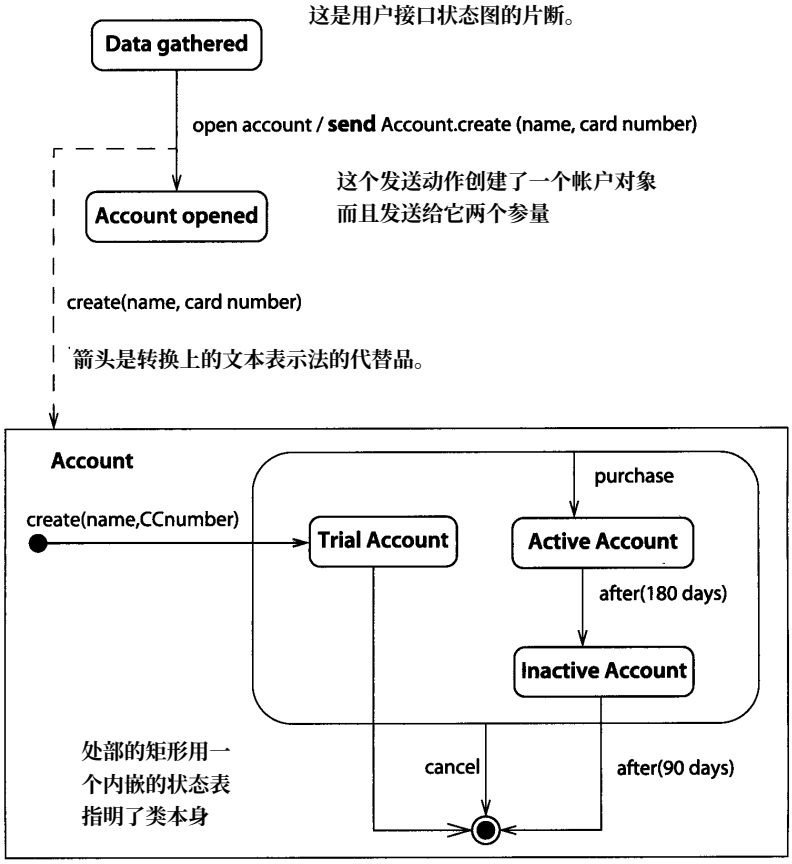


图13-159 通过发送消息创建一个新对象

如果目标语言，如C++，不支持类作为运行时间对象，那么这种方法可以使用。在这种情况下，创建动作被编译（在其普通性上强加了一些限制 —— 例如，类名必须是一个字面值），但是基本内涵是相同的。

China-pub.com
下载

发送依赖关系是从这个信号的发送者到信号的接收者的使用依赖的构造型。

文本表示法

在一个转换中，尽管信号发送只是一种特殊形式的动作，但它有自己的语法。在动作序列中，发送表达式语法为：

send *destination-expression.destination-message-name*($argument_{list,}$)

关键字send 是可选的。调用和发送可以通过声明消息名加以区分。但有时显式的区分很有帮助。

*destination-expression*的计算值必须是对象集合。只有一个对象的集合是合法的。消息发送到集合中的每个对象。

*destination-message-name*是被目标对象接收的信号或者操作的名称。参量是计算值的表达式，必须与事件或操作的声明参数相兼容。信号和操作的区别在于信号在包中声明而操作在目标类中声明。在内部模型里没有二义性。

示例

内部转换用光标位置在窗口中选择一个对象，然后发送一个 highlight 信号给它。

right-mouse-down (location) [location **in** window]

/object := pick-object (location); **send** object.highlight ()

图形表示法

消息的发放也可以用图形符号来表示。

状态机之间的消息发送可以用从发送者到接收者的一个虚线箭头来表示。箭头上标有信号名和消息的参量表达式。状态图必须包含在表示系统里的对象或类的矩形框里。从图形上来说，状态图可以在对象符号内嵌套，或者是隐含的并在别的地方显示。状态图代表了协作对象的控制。交互者可能是协作的角色，或者是表明类对象之间相互通信的普通方式的类。图13-160包含显示在三个对象之间发送信号的状态图。

应该注意这种表示法也可以用在别的图中以显示类或者对象之间的发送事件。

发送者符号（在箭头的尾部）可能是一个：

类　　消息由这个类的对象在它生命周期的某点发送，但是具体细节没有说明。

转换　　消息作为激发转换的动作的一部分发送（如图 13-159和图13-160）。这是消息发送的文本语法的另一种表示法。

接收者符号（在箭头的头部）可能是一个：

类　　消息被对象接收并且可能在这个对象内触发一个转换。类符号可能包含一个状态图（如图13-160）。接收者对象可能具有使用相同事件作为触发器的多个转换。当目标对象是动态计算出的时候，这种表示法是不可能的。这种情况

China-pub.com 下载

下，必须使用文本表达式。

元类　这种表示法被用来为类作用域操作（如新实例的创建）的激发建模。这样一个消息的接收会引起在它的缺省初始状态下实例化一个新对象。接收者所看到的事件可以被用来从它的缺省初始状态触发一个转换，所以也就代表了从创建者向新对象传递信息的一种方法。

转换　转换必须是使用事件类中唯一的转换，或者至少是唯一能被这个特定消息发送所触发的转换（如图 13-159）。当被触发的转换依赖于接收对象的状态时，这种表示法是不行的。在这种情况下，箭头必须指向类。

发送依赖关系。发送依赖关系用一个从发送信号的操作或类到接收信号的类的虚线箭头表示。构造型《send》附加在箭头上。

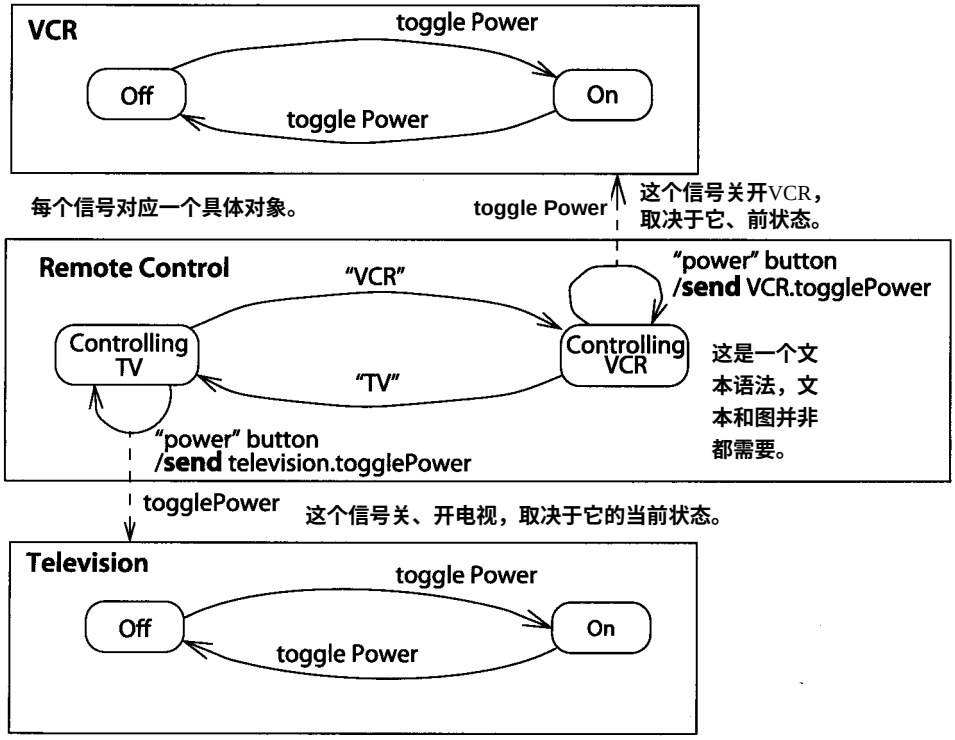


图13-160　在对象之间发送信号

299. sender(发送者)

给接收者对象传送消息实例的对象。

见call，send。

China-pub.com
下载

300. sequence diagram(顺序图)

以时间顺序显示对象交互的图。实际上，它显示了参与交互的对象和所交换消息的顺序。见activation，collaboration，lifeline，message。

语义

顺序图表示交互——以时间为次序的对象之间的通信的集合。不同于协作图，顺序图包括时间顺序但是不包括对象关系。它可以以描述符形式存在（描述所有可能的脚本），也可以以实例形式存在（描述一个实际的脚本）。顺序图和协作图表达了相似的信息，但是它们以不同的方式显示。

表示法

顺序图具有两个方向：垂直方向代表时间，水平方向代表参与交互的对象（如图13-161和图13-162）。通常，时间向下延伸（如果需要，坐标轴也可以反转）。通常只有消息的顺序是重要的，但是在实时应用中，时间轴可以是一个实际的测量尺度。对象的水平次序没有重要意义。

每个对象显示在单独的列中。对象符号（对象名带有下划线的矩形框）放置在代表生成这个对象的消息的箭头的端点，其垂直位置表示这个对象第一次创建的时间。如果对象在图的第一个操作之前就存在，则对象符号在任何消息之前处在图的顶部。从对象符号画一条虚线到对象销毁的那一点（如果销毁发生在图表示的时间内）。这条线称为生命线。一个大的X放在对象停止存在的那一点，或者放在表示销毁对象的消息的箭头的头部，或者放在对象自己销毁的那一点。对于对象活动的任何阶段，生命线加粗一倍。这包括主动对象的整个生命

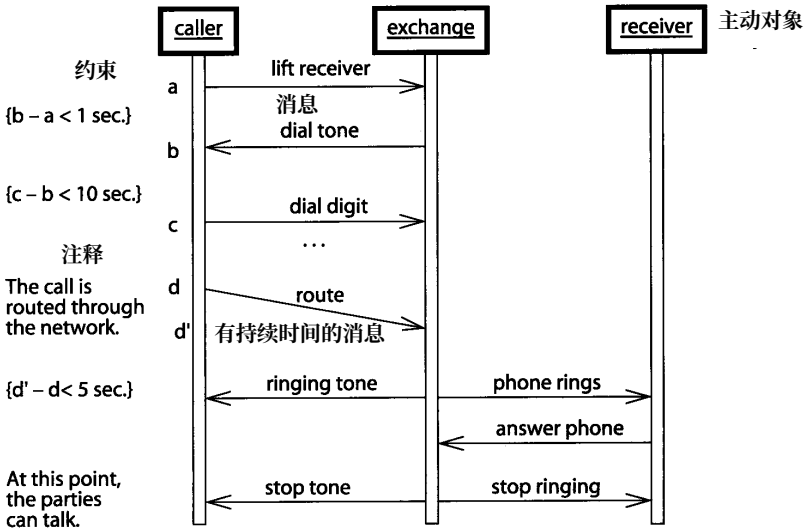


图13-161 具有异步控制的顺序图

China-pub.com 下载

或被动对象的激活——对象的某个操作执行的阶段，包括这个操作等待它所调用的操作返回的时间。如果这个操作直接或间接地递归调用它自己，则另一条两倍实心线覆盖在它上面以表示双重激活（可以是多于两个）。对象的相对次序没有重要意义，尽管合理的安排它们以使消息箭头所覆盖的距离最小是有帮助的。激活的注释可以放在附近的空白处。

每个消息显示为一个从发送消息的对象的生命线到接收消息的对象的生命线的水平箭头。在箭头相对的空白处放置一个标号以表示消息被发送的时间。在许多模型中，消息被认为是瞬时的，至少是原子的。如果一条消息需要一定的时间才能到达，消息箭头就应该向下倾斜表示接收时间晚于发送时间。两端都可以有标号来表示消息接收或者发送的时间。

对于主动对象之间的异步控制流，以两倍实心线表示对象，用箭头表示消息。两个消息可以同时发送，但是两个消息不能同时接收，因为无法保证同步接收。图 13-161显示了异步顺序图。

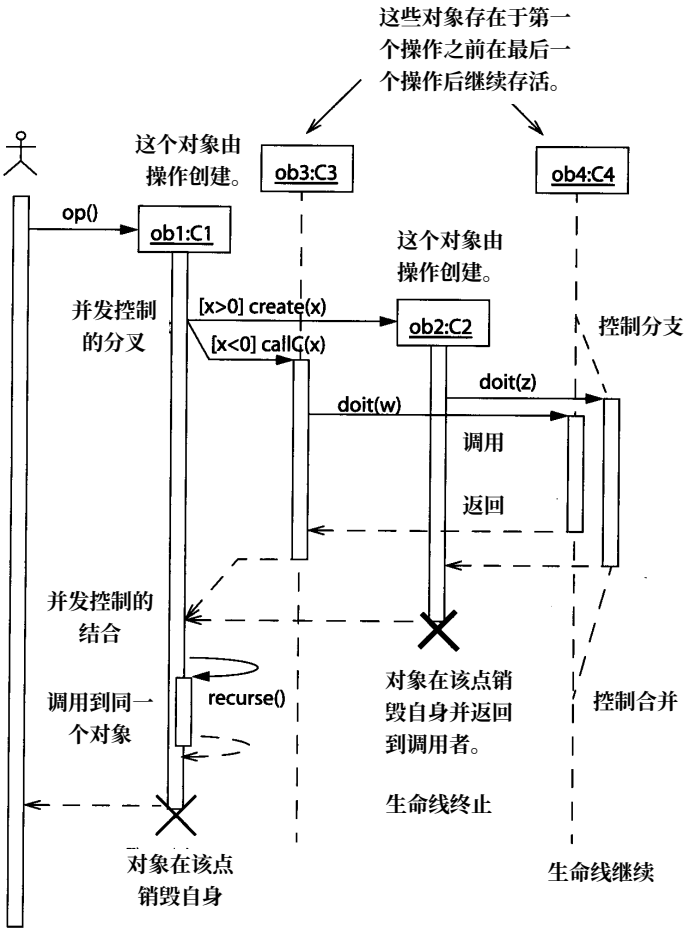


图13-162 具有过程控制流的顺序图

China-pub.com
下载

图13-162 显示了顺序图上的过程控制流。当为过程控制流建模时，对象产生对一个调用的控制直到返回。调用以实心箭头表示。调用箭头的头部可能启动一个激发或者是一个新的对象。返回以虚线显示。返回箭头的尾部可能结束这个激活或者对象。

分支用对象劈开的生命线表示。每一个分支都可以发送和接收消息。通常，每个分支发送不同的消息。对象的生命线最终应合并在一起。

图13-163显示了一张戏票的生命周期中的状态。生命线可能被状态符号中断以表示状态的改变。这对应于协作图中的变成转换。可以画一个指向状态符号的箭头以表示引起状态变化的消息。

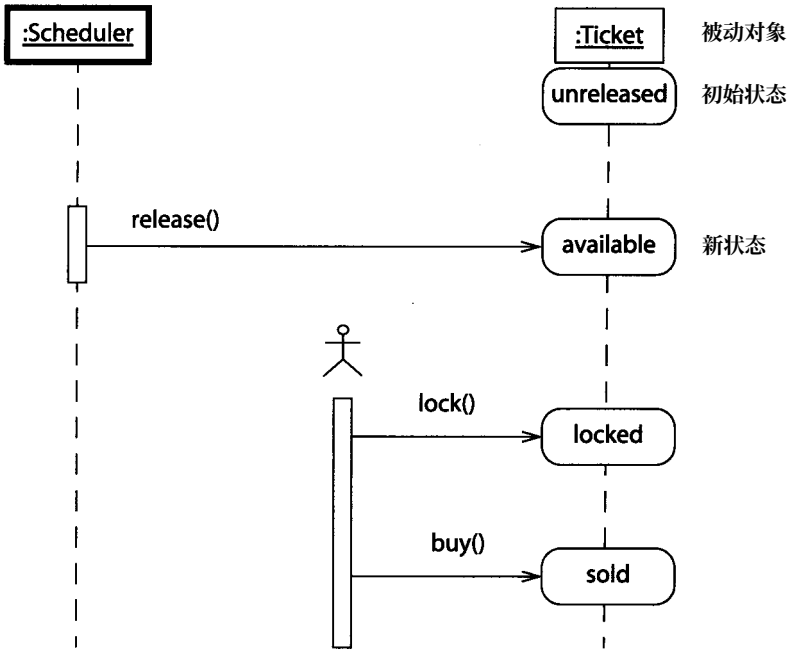


图13-163 顺序图中的对象状态

注意这种表示法的大部分都是直接从 Bushmann, Meunier, Rohnert, Sommerlad和Stal [Buschmann-96]的对象消息顺序图表示法得来的，而对象消息顺序图表示法本身又是从消息顺序图表示法继承修改而来的。

顺序图也可以以描述符形式显示，在这种方法里组成部分是角色而不是对象。这样的图显示普通的情况，不是单个的执行。描述符级的图没有下划线，因为这里的符号代表角色不是单个的对象。

301. sequence number(顺序号)

在一个表示交互中消息的相对执行次序的协作图中，消息标签的文本部分即顺序号。顺

China-pub.com 下载

序号可以表示嵌套调用中消息的位置、控制线程的名称以及条件和递归执行的规格说明。

见collaboration，message。

302. signal(信号)

对象之间异步通信的规格说明。信号可以带有表示为属性的参数。

见event，message，send。

语义

信号是以对象之间显式通信为目的的显式的命名类元。它具有一个显式的参数列表，表示为它的属性。它被显式地从一个对象发送到另外一个对象或者对象的集合。普通的信号广播可以被认为是发送一个信号到所有的对象 —— 尽管实际上为了效率上的原因，广播通常以别的方式实现。发送者在发送的时候声明信号的参数。发送一个信号等同于初始化一个信号对象然后把它传送到目标对象的集合。信号的接收是一个为了在接收者的状态机中激发转换的事件。发送到对象集合的信号可能在每个对象中独立地激发零个或者一个转换。信号是对象之间异步通信的显式方式。要进行同步通信，必须使用两个信号，每个方向用一个。

信号是可泛化的元素，子信号从父信号继承来。它继承父亲的属性，也可以增加它自己的属性。子信号可以激发声明为使用它的祖先信号的转换。

信号声明在它被声明的包中具有作用域。它没有被限制在单个类里。

类和接口都可以声明它准备处理的信号。这种声明是一个接收，它可能包含对信号被接收是期望的结果的规格说明。这个声明具有说明它是否是多态的特性。如果它是多态的，则它的后代可以处理这个信号，这可能会使这个信号无法到达当前类。如果它不是多态的，则它的后代都不能截获这个信号的处理。

信号是一种类元，可以具有访问和修改属性的操作。所有的信号共享隐含的操作 send。

send(*targetSet*)。

信号被发送到目标集合中的每个对象。

表示法

构造型关键字《signal》放在具有信号名的操作声明前以表示类或者接口接收这个信号。信号的参数包含在声明中。声明不一定要有返回类型。

信号的声明可以表示为一个类符号的构造型。关键字《signal》出现在信号名上面的矩形框里。信号的参数作为属性出现在属性部分。操作分栏可能包含访问操作。

图13-164显示了使用泛化表示法把一个子信号联系到它的父信号上。子信号继承它的祖先的参数同时也可以增加自己的参数。例如，信号 MouseButtonDown 具有属性 time，device，location。

China-pub.com
下载

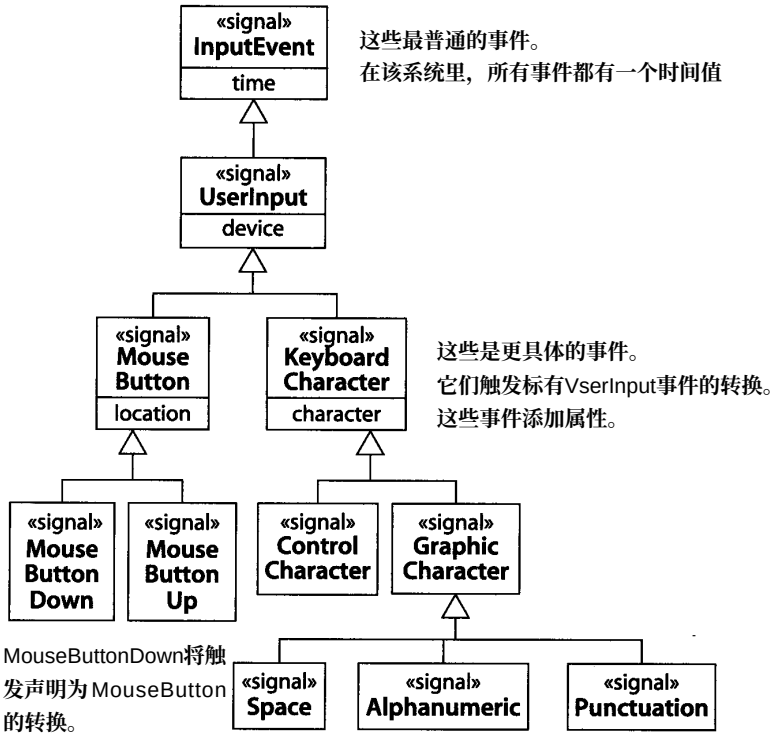


图13-164 信号声明

要使用信号作为一个转换的触发，使用下面的语法：

event-name（*parameter*）

而参数列表具有下面的语法：

parameter-name：*type-expression*

信号参数被声明为可以带有初始值的属性，这个初始值可以在初始化或者发送时被重载。如果一个信号实例被创建、初始化，然后发送到一个对象，则使用初始值。如果信号使用操作调用语法发送，则初始值就是信号参数的缺省值。

讨论

信号是对象之间最基础的通信，比过程调用具有更为简单和清楚的语义。信号固有地是从一个对象到另一对象的单向异步通信，所有的信息通过值来传递。它适合于为分布式并发系统建模。

为建立同步通信，使用一对信号，每个方向使用一个信号。一个调用可以认为是带有隐含返回指针参数的信号。

China-pub.com 下载

303. signal event(信号事件)

一个对象对发送给它的信号的接收事件，它可能会在状态机内触发转换。

304. signature(特征标记)

行为特征的名称和参数特性，如操作或者信号。特征标记包括可选的返回类型（操作有，而信号没有）。

语义

操作的特征标记是它的声明的一部分。一些（不是全部）特征标记被用来匹配操作和方法以检查抵触或者重载。为达到这个目的，使用操作的名称和参数类型的顺序列表，但是不包括参数名称和方向，返回参数也被排除在外。如果两个特征标记匹配，但是其他的特性不一致（例如，一个in 参数对应一个out 参数），则声明冲突，且模型为非良性结构。

305. simple state(简单状态)

其中没有嵌套状态的状态。嵌套状态的集合形成一颗树而简单状态是叶。简单状态没有子结构。它可能具有内部转换、入口动作和出口动作。对比：*composite state*。

306. simple transition(简单转换)

具有一个源状态和一个目标状态的转换。它表示对独占状态区域内的状态改变事件的响应。它的执行结果不会改变并发的数目。

307. single classification(单分类)

单分类即每个对象都恰好有一个直接类的执行体制。它是大多数面向对象编程语言的执行模型。是否允许单分类还是多重分类是一个语义变更点。

308. single inheritance(单继承)

每个对象只允许有一个父的泛化的语义变更。是否允许单继承或者多重继承是一个语义

China-pub.com
下载

变更点。

309. singleton(单实例类)

单实例类即只有一个实例的类（通过声明）。单实例类是在面向对象的框架内在应用中代表全局信息的一种方式。

语义

每个应用必须至少有一个单实例类（通常是隐含的）以为应用建立语境。通常，这个单实例类等同于应用本身，并通过计算机中的控制栈和地址空间来实现。

表示法

单实例类以右上角标有一个小“1”的类符号来表示（如图13-165）。该值代表了系统中类的多重性。

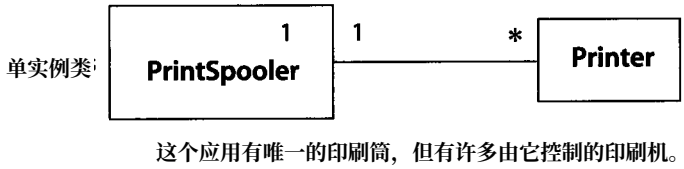


图13-165 单实例类

310. snapshot(快照)

快照即在执行过程中的某时刻，形成系统配置的对象、链和值的集合。

311. source scope(源作用域)

槽被实例拥有还是被类拥有的指示器。

见scope。

312. source state(源状态)

状态机中从转换出发的状态。转换应用于源状态。如果对象在这个状态或者其嵌套状态

China-pub.com 下载

中，则转换有可能被激发。

313. specialization(特化)

特化即通过增加子对象产生对一个模型元素的更为特殊的描述。相反的关系即泛化，泛化也被用来作为特殊元素和一般元素之间关系的名称，因为没有更为合适的术语。子元素是对父元素的特化。相反，父元素是对子元素的泛化。

见genaralization。

314. specification(规格说明)

某事物是什末或者做什末的断言式的描述。例如，用例或者接口就是一个声明。对比：实现

315. stages of modeling(建模阶段)

在设计和建立一个系统的过程中，一个元素或者模型所经历的开发状态。

见development process。

讨论

综合的开发效果可以划分为不同的活动。这些活动不是按顺序执行，相反，它们是在开发过程阶段中递归地执行。分析阶段捕获需求和理解系统的需要。设计阶段在数据结构、算法以及现有系统的约束下，产生一个实际可行的解决方法。实现阶段在可执行的语言或者介质上（如数据库或者数字硬件）构造解决方案。部署阶段把解决方案在一特定的物理环境中付诸实施。这种划分方法可能有点武断，且并不总是很清楚，但是它具有指导意义。

但是，这种开发观点不能等同于开发过程的顺序阶段。在传统的瀑布过程中，它们确实是不同的阶段。在更为现代化的递归开发过程中，它们不能完全区分。在某时间点上，开发活动可能存在于不同的级别，所以它们最好被理解为需要在系统的每个元素上执行的不同的任务，但不是同时执行。

想象一下一组建筑物，它们具有各自的地基、墙壁和房顶。必须为所有建筑物完成所有这些东西，但不是所有的都同时完成。通常，每个建筑物的各个部分是按顺序完成的。但是，有时房顶可以在所有的墙壁完成之前开始。墙壁和房顶的区别偶尔也会消失 —— 考虑一下建在地面上的圆顶。

某些UML元素是为开发的所有阶段准备的。其他元素是以设计或者实现为目的的，它们

China-pub.com
下载

只会在模型足够完善时才出现。例如，属性和操作的可见性规格说明倾向于在设计阶段出现。在分析阶段主要包括公共成员。

讨论

瀑布开发过程被分为各个阶段，每个阶段在整个系统上执行一次。传统的阶段包括分析、设计、实现和部署。但在递归过程中，整个系统的开发不是以互锁的步骤进行的。元素可能以不同的进度开发，每个元素在开发过程中经过相同的阶段，但是不同的元素以不同的速度进行，所以系统作为一个整体不是处在任何一个阶段。

每个建模阶段处理必须理解和建模的领域。早期的阶段捕获更为逻辑化和抽象的特性。后期的阶段更注重于实现和执行。分析阶段捕捉系统的需求和专业词汇。设计阶段捕捉理想条件下抽象实现的算法和数据结构。它也可能涉及到效率、计算复杂性以及为建立一个支持系统的软件工程方面的考虑。实现阶段在真实的编程语言的真实条件下为系统产生一个操作描述，包括真实语言的缺陷和把系统分解为可以独立开发和存储的部分。运行时间处理资源的并发、计算环境以及大规模运行。

UML包含各种适合于各个不同开发阶段的结构。一些结构（如关联和状态）对所有的阶段都有意义，一些结构（如导航性和可见性）在设计阶段有意义但在分析阶段只展示一些不必要的实现细节。这并不排除在工作的较早阶段对它们进行定义。一些结构（如特定的编程语言语法）只在实现阶段有意义，如果过早引入会妨碍开发过程。

模型在开发过程中会改变。一个 UML模型在开发的不同阶段呈现不同的形式，着重于不同的UML结构。建模时首先应该理解不是所有的结构都在所有阶段有意义。

见development process，可以找到关于建模阶段和开发阶段之间联系的讨论。

316. state (状态)

在对象的生命期中满足一定条件，执行某些活动或者等待某事件的条件或者情况。

见activity，activity graph，composite state，entry action，exit action，final state，internal transition，pseudostate，state machine，submachine，synch state，transition。

语义

一个对象在它的生命期中有一系列的状态。对象在一段有限的时间中位于某个状态中。为了方便可以引入假状态，它们执行琐屑的动作然后退出。但是这不是状态的主要目的，而且假状态原则上可以被去掉，尽管它们对于避免重复是有用的。

状态包含在状态机里，状态机描述对象响应事件而发展的历史情况。每个状态机描述某个类的对象的行为。每个类可以有一个状态机。转换描述了处于某状态的对象对事件的接收做出的响应：对象执行一个附属于转换的可选动作并改变到一个新的状态。每个状态有自己

China-pub.com 下载

的转换集。

动作是原子的，不能被中断。动作附属于转换 —— 状态的改变，而转换本身也是原子的不能中断。进行中的活动可能和状态关联。这样的活动被声明为嵌套的状态机或者是一个 do 表达式。进行中的活动也可以以一对动作表示，一个在进入状态时启动激发的入口动作和一个在从状态退出时终止激发的出口动作。

状态可以分组为组成状态。组成状态上的任何转换会影响其中的所有状态，所以以相同的形式影响很多子状态的事件可以通过一个转换处理。组成状态可以是顺序的或者并行的。在顺序组成状态中只有一个状态可以是活动的。在并发组成状态中，所有的状态同时处于活动。

为促进封装，组成状态可以具有初始状态和终止状态。它们是伪状态，目的是为了优化状态机的结构。到组成状态的转换代表初始状态的转换，这等同于直接到它的初始状态的转换，但是这个初始状态可以在外部使用而不用知道它的内部结构。

到组成状态的终止状态的转换代表了在这个封闭状态里活动的完成。封闭状态中活动的完成激发这个封闭状态上的活动事件的完成，并引发封闭状态上的完成转换。完成转换是没有显式触发器事件的转换（或者，更精确地说，尽管没有显式建模，它以这个完成事件作为隐式触发器）。对象最外面的状态的完成对应它的死亡。

如果状态是并发组成状态，那么它的并发子区域必须在组成状态上的完成事件发生前完成。换而言之，从组成并发状态触发的完成转换代表它的所有并发子线程的控制结合。它等待所有的子线程完成才进行。

结构

状态具有下列部分：

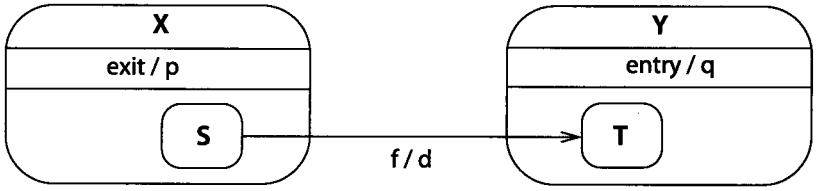
名称。即状态的名称，它必须在封闭状态中保持唯一。名称可以省略，以生成一个匿名状态。任何数目的不同匿名状态可以共存。嵌套状态可以通过它的路径名确定（如果所有的封闭状态都有名称）。

子状态。如果一个状态机具有嵌套的子结构，则它被称为组成状态。组成状态或者是互斥的子状态构成的网络（即依次活动的子状态），或者是并发子状态的集合（即并发活动的子状态）。没有子结构的状态（除去可能的内部动作）是一个简单状态。

入口和出口动作。状态可能具有入口和出口动作。这些动作的目的是封装这个状态，这样就可以不必知道状态的内部状态而在外部使用它。当进入状态时，入口动作被执行，它在任何附属于输入转换的动作之后而在任何状态的内部活动之前执行。状态退出时执行出口动作，它在任何内部活动完成之后而在任何附属于输出转换的动作之前执行。在穿过几个状态边界的转换上，入口和出口动作可能以嵌套的方式执行。首先，出口动作被执行，从最内层的状态开始向最外层的状态进行，然后，转换上的动作被执行，之后入口动作被执行，从最外层的状态开始以最内层的状态结束。图 13-166显示了激发一个穿过几个状态边界的转换的

China-pub.com
下载

结果。入口和出口动作在任何情况下都不能避免即使有异常发生。它们为状态机行为的规格说明提供了一种封装机制，并且能够保证必要的动作在任何情况下都能执行。



有效结果：f/p:d:q

图13-166 穿过状态边界的转换，带有入口和出口动作

内部活动。状态可以包含描述为表达式的内部活动。当状态进入时，在入口动作完成之后活动开始。如果活动结束，则状态完成。然后一个从这个状态出发的完成转换被触发。否则，状态等待触发转换以引起状态的改变。如果在活动正在执行时转换触发，则活动结束并且出口动作被执行。

内部转换。状态可能包含一系列的内部转换，它们除了没有目标状态不能引起状态转换外和普通转换相像。如果对象的事件在对象正处在拥有转换的状态时发生，则内部转换上的动作被执行，但是没有状态的改变也没有入口和出口动作发生，即使内部转换在封闭状态中声明（因为状态没有改变）。这一点与自转换不同，在后者中，一个从一个状态到同一个状态的外部转换发生，导致所有嵌套在具有自转换的状态中的状态的出口动作、它自身的出口动作、它的入口动作的执行。在转向当前状态的自转换上，动作被执行，退出然后重新进入。如果当前状态的封闭状态的自转换激发，那么终止状态是封闭状态自身，而不是当前状态。换而言之，自转换可以强制从嵌套状态退出，但是内部转换不能。

子状态机。状态体可以代表通过名称引用的分离的状态机的副本。被引用的状态机被称为子状态机，因为它被嵌套在更大的状态机里，产生引用的状态被称为子状态机状态。子状态机可以附属于为动作提供语境的类，如可以读写的属性。子状态机的目的是在很多状态机里重用以避免重复相同的状态机片断。子状态机是一种状态机的子例程。

在子状态机的引用状态里，子状态机通过带有可能的参数列表的名称引用。这个名称必须是一个具有初始和终止状态的状态机的名称。如果子状态机在它的初始转换上有参数，那么参数列表必须具有匹配的参数。当进入子状态机时，首先执行入口动作，然后从初始状态开始执行子状态机。当子状态机到达它的终止状态时，其出口动作被执行。子状态机被认为执行完毕，可能会引起以隐式的完成活动为基础的转换。

到子状态机引用状态的转换激活目标子状态机的初始状态。但是有时希望转向子状态机中其他状态的转换。桩状态是一个放在子状态机引用状态中的伪状态。转换可以从主状态机中的其他状态连向桩状态。如果转向桩状态的转换激发，则子状态机副本中的引用状态变成活动的。

China-pub.com 下载

子状态机代表状态中嵌套的可中断的活动。它等同于以唯一的子状态机的副本代替子状态机状态。不用提供状态机，而可以在子状态机上附加过程表达式（这时一种活动）。活动可以认为是一系列状态的定义，每个简单表达式定义一个状态，在任何两步之间都可以中断。它不同于动作，动作是原子的和不可中断的。

动态并发。活动状态或子状态机状态可能具有多重性和并发性表达式。多重性说明了状态的多少个副本可以并发执行。普通情况是仅为 1的多重性，意味着这个状态表示普通的控制线程。如果多重性的值没有固定，则意味着执行的数目在运行时间动态地确定。例如，值 1..5 意味着从活动的1~5份副本并发执行。如果存在并发表达式（如果并发性不是 1，则这是必须的），那么在运行时间它必须在参量列表的集合上计算。集合的基数表示状态的并发活动的数目。每个状态接收不同的参量值列表作为它隐含的当前事件的值。活动的动作可以访问当前事件的值。当所有的执行都已经完成，则动态并发状态被认为完成，执行转到下一个状态。这种能力针对活动图而言。

延迟事件。状态中延迟发生的列表，如果它们不触发转换，那么直到它们触发转换或者系统产生一个到其中事件不被延迟的状态的转换 ，它们才被重新执行。这种延迟事件的实现包含事件的内部队列。

表示法

状态用圆角矩形表示。它可能有一个或者多个分栏。这些分栏是可选的，具体如下：

名称分栏。表示状态的（可选）名称，其形式为字符串。没有名称的状态是匿名的，而且互不相同。但是不能在同一个图里重复相同的命名状态符号，因为这容易使人混淆。

嵌套状态。显示组成状态本身的由附属嵌套的状态构成的状态图。状态图位于外部状态的范围里。转换可能直接连接到嵌套状态，还可以连接到外部状态的边界。在互斥区域中，子状态直接在组成状态里画出。在并发区域中，并发状态符号用虚线分为子区域（即平铺）。

见composite state。

内部转换分栏。表示内部活动或者动作列表。这些活动或者动作是在对象处于状态中时，接收到事件而做出响应执行的，结果不改变状态。内部转换具有下面的形式：

$event\text{-}name_{opt}(argument_{list})_{opt}[guard\text{-}condition]_{opt}/action\text{-}expression_{opt}$

动作表达式可以使用拥有对象的属性和链以及输入转换的参数（如果它们出现在所有的输入转换里）。

如果没有参数，则参量列表（包括圆括弧）可以省略。监护条件（包括方括弧）和动作表达式（包括斜杠）是可选的。

入口和出口动作具有相同的形式，除了不能用作事件名的保留字： entery和exit：

entery/*action-expression*

exit/*action-expression*

入口和出口动作不能有参量或者监护条件（因为它们是隐含而不是显式调用）。为了在入口

China-pub.com
下载

动作上获得参数，当前事件可以通过动作访问。这一点在获得新对象的创建参数时特别有用。

保留动作名defer 表示在状态或者子状态里可以延迟的事件。内部转换不能有监护条件或者动作：

event-name/defer

保留字do 表示非原子活动的表达式：

do/*activity-expression*

子状态机引用状态。嵌套子状态机的激发用状态符号体中的字符串表示，形式如下：

include *machine-name*($\overline{argument}_{list}$)$_{opt}$

do-活动和子状态机都描述了非原子计算，这些计算通常运行到完成，但是可以被激发转换的事件中断。

示例

图13-167 显示了带有内部转换的状态。图 13-168显示了子状态机的声明和使用。

• 动态并发。值不为 1的动态并发以状态符号右上角的多重性字符串表示。对于普通的顺序执行，这个字符串不应该被包含。这种表示法主要用于活动图而在状态图里应该尽量避免。

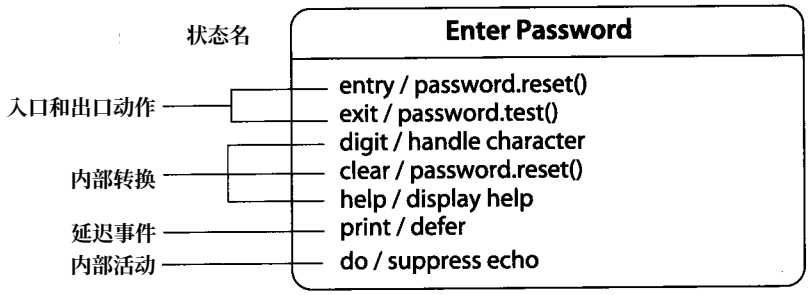


图13-167 具有入口和出口动作以及延迟事件的内部转换

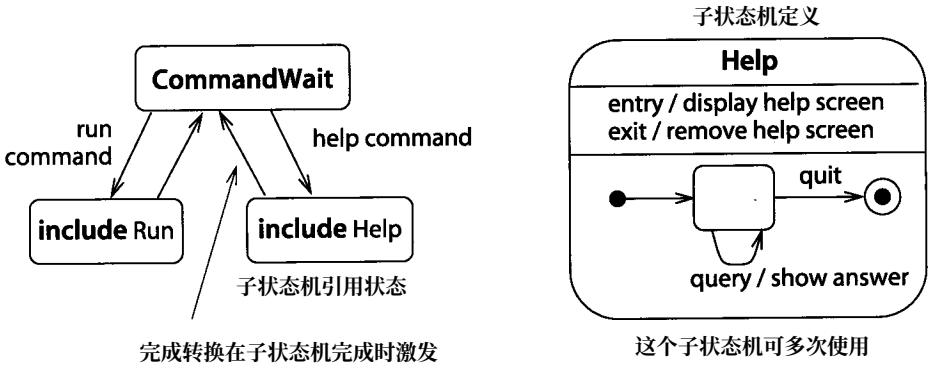


图13-168 子状态机

China-pub.com 下载

317. state machine (状态机)

对象或者交互在其生命期中为响应事件而经历的状态顺序的规格说明，连同它的响应动作。状态机附属于源类，协作或者方法，声明源元素实例的行为。

见activity graph，composite state，event，pseudostate，state，transition。

语义

状态机是一个状态和转换的图，描述了类元实例对事件接收的响应。状态机可以附属于类元，如类和用例，还可以附属于协作和方法。状态机所附的元素被称为状态机的主机。

完整的状态机是一个可递归分解为子状态的组成状态。最里面的简单状态没有子状态。

状态机执行语义

状态机执行语义在本书的其他部分进行讨论。注意下面的部分描述了状态机执行的语义效果，且不应该被认为是一种实现方法。有很多方式实现这些语义，多数语义在别处描述，但是为了方便我们把它们收集在这里。

在对象或者其他实例的状态机的活动状态配置里，在任何时候都存在一个或者多个活动的状态。如果一个状态是活动的，那么离开这个状态的转换可能会激发，引起一个动作的执行，并使得另一个状态或位于初始状态位置的状态激活。多个活动的叶状态表示内部并发性。状态机的结构和的转换对可以并发活动的状态施加了约束。简单的说，如果一个顺序组成状态处于活动，则只有一个直接的互斥子状态必须处于活动。如果一个并发组成状态处于活动，则每个直接子状态都必须处于活动。

转换激发和动作

首先作一个基本假设：状态机在某一时刻处理某个事件而在处理另外的事件之前必须完成这个事件的结果。换而言之，在事件处理过程中，事件之间不能交互。这就是“运行到完成”处理方法。这并不意味着所有的计算都是不能中断的。一个普通的扩展计算可以分为一系列原子步骤，在任何步骤之间外部事件可以中断这个计算。这与计算机里的物理情况非常相似，中断可以发生在分离的小步骤之间。

一个推论是事件是异步的。两个事件不能在完全相同的时间发生 —— 更为精确地说，如果两个事件恰在相同的时间发生，这只是巧合，可以用任何次序处理它们而不会丢失任何一般性。不同的执行次序所产生的结果是不同的 —— 竞争条件是并发系统的一个重要特性 —— 但是不能在分布式系统里假定同时性。任何做这个假定的计算在逻辑和物理上都存在缺陷。并发执行要求分布式系统里的独立性。

概念上来讲，动作是瞬间的，没有两个事件是同时的。在一个实现中，动作的执行需要

China-pub.com
下载

一定的时间，但是重要的是动作是原子而不可中断的（概念上）。如果一个对象在执行动作的期间接收到一个事件，则这个事件被放到一个队列中直到动作执行完成。只有在没有动作被执行时才会处理事件。如果动作发送一个信号到另外一个对象，那么这个信号的接收不是同步的。它在动作和转换完成后像另外一个事件一样被处理。对一个操作的调用会挂起调用者直到操作被执行。它可以被实现为触发接收者的状态机的方法或者调用事件，这可由接收者选择。为避免长时间不能处理事件，动作应该简单一点。动作不应该为受保护区域或者长时间可中断计算建模，它们应该设计为子状态机或者是嵌套活动状态。这允许事件处理，也允许嵌套计算被中断。如果在一个真实的系统里包含一个长时间的动作，那么事件有可能无法及时处理。这就是糟糕的模型的后果。动作与可能发生的事件所要求的响应时间相比必须简短。

当对象没有执行动作时，它应该立即处理它所接收的事件。概念上来说，动作是瞬间的，但是实际上它需要一定的时间，所以新的事件必须放在对象的一个队列里。如果队列里没有事件，对象会等待直到有事件出现然后处理它。概念上来说，对象每次只处理一个事件。这不是限制因为动作被认为是原子的，十分简短。在实际的实现中，事件可能以确定的次序存放在队列里。但是，UML语义并没有说明处理并发事件的次序，建模者也不应该假定。如果事件必须以某一特定的次序处理，那么状态机必须合理建造以强制实施这种次序。物理实现应该选择一些简单的定序规则。

在对象处理事件时，它的活动状态配置可能包含一个或者多个并发状态。每个状态接收一份独立的事件副本并独立地响应。并发状态的转换独立激发。一个子状态可以改变而不影响其他的状态，有一种例外的情况是复杂转换，如分叉或者结合（在后面描述）。

对于对象的每个活动状态，状态的输出转换作为激发的候选者。如果类型与候选转换相同的事件被处理或者事件是转换的触发器事件的后代，那么候选转换被触发。转换不会被祖先事件所触发。当事件被处理并激发一个转换时，监护条件被计算。如果监护条件的值为真，则转换被允许。监护条件的布尔表达式中可能包含触发器事件的参量，以及对象的属性。应该注意监护表达式不可以产生副作用，即它们不能改变对象的状态或者系统的其余部分。所以，它们的计算次序与结果无关。只有处理事件时才计算监护条件。如果它的计算值为假，那么若其中的变量在以后改变了值，也不会重新计算。

为构造复杂条件，可以用多个片断给转换建模。第一个片断有触发器事件，紧跟其后的是具有监护条件的分叉树。树的中间节点是伪状态，是为构造转换而存在的假状态，但是它们在运行到完成步骤的末尾不能保持活动。每个经过片断树的可能路径被认为是独立的转换，可独立地执行。每个片断不能单独激发。一系列片断上的监护条件必须为真，否则转换（包括它的所有片断）根本不会激发。实际上，分支点上的监护条件经常分割可能的结果。所以，实现每次可以处理多段转换，但不是总是如此。

如果没有转换被启动，则事件被忽略。这不是错误。如果恰有一个转换被启动，则它被激发。如果来自单个状态的多个转换被启动，则它们中的一个被激发。如果没有说明任何的约束，那么选择是不确定的。不能假定选择是公平、可预测或者随机的。实际的实现可能会

China-pub.com 下载

提供解决冲突的规则，但是建模者应该给出明确的意图，而不应该单纯依赖这些规则。不论是否有转换激发，事件都消灭。

离开一个活动状态的转换有资格被激发。另外，包含活动状态的组成状态是触发的候选者。这可以认为是通过嵌套状态而进行的转换继承，类似于通过子类进行的操作继承。只有内部状态上没有转换激发时，外部状态上的转换才有资格激发。否则，外部转换会被内部转换掩盖。

当转换激发时，它的任何附属动作都会执行。动作表达式可以使用触发器事件的参量以及可以到达的拥有对象的属性和值。动作是原子的，在任何附加事件被处理之前完成。如果一个转换有多个片断，则触发器事件的参数作为隐含的当前事件是可得的。

如果对象具有并发状态，那么它们不能通过共享存储器交互。并发状态应该是独立的，应该作用在不同的值集上。任何交互都应该通过发送信号显式进行。如果两个并发状态必须同时访问一个共享资源，则它们必须显式地向资源发送信号，由资源做出仲裁。实现里可能会去掉这样显式的通信，但是必须小心防止出现无意义或者危险的冲突。若并发动作确实访问了共享值，那么结果是不确定的。

如果穿过组成状态边界的转换被激发，则入口或者出口动作可能会被执行。当转换的源状态和目标状态处在不同的组成状态时会发生边界穿过。当激发的转换是从一个外部复合状态继承来的时，也会发生边界穿过，因此强制对象从一个或多个内部状态退出。注意内部转换不会引起状态的改变所以从来不会引发入口或者出口动作。

为确定执行的入口和出口动作，找出对象的当前活动状态（可能嵌套在具有该转换的组成状态里）和转换的目标状态。然后找到封装了当前状态和目标状态的最内层组成状态。它被称作公共祖先。当前状态和直到公共祖先的任何封闭状态 (但不包括公共祖先)的出口动作被执行,最内层的首先执行，然后转换之上的动作执行。在这之后，目标状态和直到公共祖先的任何封闭状态（但不包含公共祖先）的入口动作被执行，外层的首先执行。换而言之，每次从一个状态退出直到到达公共祖先，再依次进入各个状态直到到达目的状态。公共祖先上的入口和出口动作不被执行，因为它没有被改变。这个过程保证每个状态都被很好地封装。

转换上的动作在任何出口动作执行完毕后而在任何入口动作执行之前执行。

注意自转换（一个到其自身的转换）的激发会引起可能处于活动（该转换可能从一个封装的组成状态继承而来）的源状态的任何嵌套状态的退出。它还会引起源状态的出口动作的执行，然后是它的入口动作。换而言之，状态退出然后重新进入。如果这不是期望的结果，那么应该在这个状态里使用内部转换。这不会引起状态的改变，即使活动状态嵌套在具有这个转换的状态里。

在运行到完成步骤的执行过程中，所有的动作都可以访问一个隐含的当前事件，该事件是引发运行到完成序列里的第一个转换的事件。因为可能存在多个可以导致一个动作的执行的事件，动作可以在当前事件的类型上作出区分（像在 Ada 或者多态操作）以执行可选的代码分支。

China-pub.com
下载

在所有的动作执行完毕后，初始当前状态不在活动（除非它是目标状态），转换的目标状态处于活动，额外的事件可以被处理。

可以把转换构造成有几个其中间节点是结合状态的片断。每个片断具有自己的动作。这些动作可能和整个转换的入口和出口动作产生交错。对于入口和出口动作，如果片断是一个完整的转换，则转换片断上的每个动作在其应该发生的位置发生。图 13-117即提供了一个示例。

内部转换

内部转换具有源状态但没有目标状态。它的激发确实不会引起状态的改变，即使激活的转换是从一个封装状态继承而来。因为状态没有改变，故没有入口和出口动作被执行。内部转换的唯一结果是执行它的动作。激发一个内部转换的条件和激发外部转换的条件是相同的。

注意内部转换的激发可能会掩盖使用相同的事件的外部转换。所以，有时定义一个没有动作的内部转换是有目的的。如上所述，在顺序区域里每个事件只激发一个转换，而内部转换具有比外部转换更高的优先权。

内部转换对于处理没有状态改变的事件很有用。

初始状态和终止状态

为封装状态经常可以使状态的内部和外部分离。也可以把一个转换连接到一个组成状态上而不必知道状态的内部结构。这可以通过使用组成状态中的初始状态和终止状态来完成。

一个状态可能具有初始状态和终止状态。初始状态是一个伪状态（一个和普通状态有连接的假状态）对象不可能保持在初始状态。对象可以保持在终止状态，但是终止状态不能有任何显式的触发转换，它的目的是为激发封装状态上的完成转换。初始状态必须有一个输出完成转换。如果存在多个输出转换，那么它们必须都缺少触发，而监护条件必须分离可能的值。换而言之，当初始状态被激发时，有且仅有一个输出转换被激发。对象从不会保持在初始状态，而会立刻转换到一个普通状态。

如果一个组成状态有初始状态，那么转换可以直接连接到组成状态作为目标。任何到组成状态的转换都隐式地是一个到组成状态的初始状态的转换。如果组成状态没有初始状态，则转换不能把组成状态作为目标。它们必须直接连接到子状态。一个具有初始状态的状态可能有直接连接到内部状态的转换，也可以有直接连接到组成状态的转换。

如果组成状态具有结束状态，那么它可能是一个或者多个输出完成转换的源，即缺乏显式事件触发器的转换。完成转换是由状态中活动的完成隐含地启动的转换。一个到终止状态的转换意味着组成状态执行完成。当一个对象转换到终止状态时，离开封装的组成状态的完成转换在其监护条件被满足的情况下允许激发。

组成状态也可以拥有输出转换，即具有显式的事件触发器的转换。如果一个引发这种转换的事件发生，则状态中任何正在进行的活动（在任何的嵌套深度）都会被终结，被终结的

China-pub.com 下载

嵌套状态的出口动作被执行，转换被处理。这样的转换通常用来为异常和错误条件建模。

复杂转换

进入并发组成状态的转换意味着一个进入它的所有并发子状态的转换。这可以以两种方式发生。

一个转换可以有多个目标状态，每个并发子状态一个。注意这样的分叉转换仍然只有一个触发器事件、监护条件和动作。这是直接声明了每个目标状态的显式的到组成状态的转换。这代表了一个进入并发子线程的显式控制分叉。

转换可以省略一个或者多个并发子状态的目标，或者它把组成状态本身作为目标。在这种情况下，每个被省略的并发子状态必须有一个初始状态以指明它的缺省启动状态。否则，状态机为非良性结构。如果复杂转换激发，则显式的目标并发子状态将处于活动，其他的并发子状态的初始状态也处于活动。简单地说，任何到并发子状态的转换暗示着进入一切没有显式说明的其他对等并发子状态的初始状态的转换。到组成状态本身的转换暗示着进入它的每个并发区域的初始状态的转换。如果一个并发组成状态处于活动，则它的每个子区域都处于活动。

同样，从任何并发子状态输出的转换暗示着从所有子状态输出的转换。如果事件的发生引起这样的转换激发，则其余子状态的活动被终结，它们执行出口动作，转换自己的动作被执行，目标状态变成活动的，因此减少了活动并发状态的数目。

到一个并发子状态的终止状态的转换不会强制终结别的并发子状态（这不是子状态外的转换）。当所有的并发子状态到达它们的终止状态时，封装组成状态应该已经完成它的活动，任何离开组成状态的完成转换都启动激发。

复杂转换可以具有多个源状态和多个目标状态。在这种情况下，它的行为是上面所述的分叉和结合的综合。

历史状态

组成状态可能包含历史状态，历史状态本身是个伪状态。如果一个被继承的转换引起从组成状态的自动退出，那么状态会记住当强制性退出发生的时候处于活动的状态。到组成状态中的历史伪状态的转换表示被记住的子状态要重新建立。到别的状态或者封装状态本身的显式转换不能启动历史机制故采用通常的转换规则。但是，组成状态的的初始状态可以连接到历史状态。在这种情况下，到组成状态的转换确实（非直接）调用历史机制。历史状态可以有一个没有监护条件的输出完成转换，转换的目标是缺省的历史状态。如果状态区域从来没有进入或者已经退出，那么到历史状态的转换会到达缺省的历史状态。

存在两种历史状态：浅历史状态和深历史状态。浅历史状态保存被直接包含（深度为 1）在相同的组成状态里的状态作为历史状态。深历史状态保存在最后一个引起封装组成状态退出的显式转换之前处于活动的所有状态。它可能包含嵌套在组成状态里的任何深度的状态。

China-pub.com
下载

一个组成状态最多只有一种历史状态。每个状态可能有它自己的缺省历史状态。

如果可以更为直接地建模，则应该避免历史机制，因为它过于复杂，而且不是一种好的实现机制。深历史状态尤其容易出问题，应该尽量避免用它支持更显式（也更易于实现）的机制。

表示法

状态图表示状态机或者状态机的嵌套部分。状态用状态符号表示，转换用连接状态符号的箭头表示。状态可能通过物理包含和平铺包含子图。图 13-169 显示了这样一个例子。

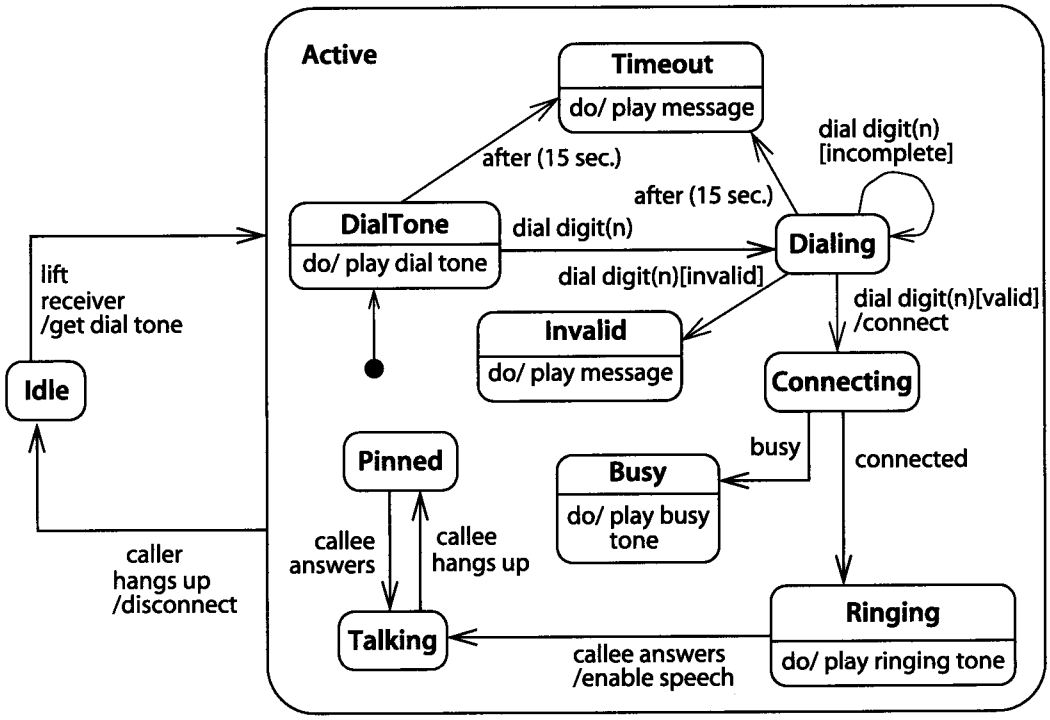


图13-169 状态图

状态图表示法是David Harel 发明的，结合了Moore机(有关入口动作)和Mealy机(有关转换动作)的各个方面，而且增加了嵌套状态和并发状态的概念。

要得到更详细的细节，见state, submachine, pseudostate, entry action, exit action, transition, internal transition和activity。要探究活动流的合适表示法的另一个变体，见 activity graph。也可见control icons, 其中有用于活动图且而在状态图里也可以使用的可选符号。

讨论

状态机可以有两种使用方式。所以，它们的含义可以以任何一种来理解。在一种情况下，

China-pub.com 下载

状态机可以声明它的主元素（典型的是一个类）的可执行行为。此时，状态机描述了它的主元素从整体中的其他部分接收到事件时的响应。响应用转换来描述，每个转换描述主元素在给定状态下接收到一个事件的响应。结果描述为一个动作和一个状态改变。动作可以包括向别的对象发送信号从而触发它们的状态机。状态机提供了对系统行为的简化规格说明。

在第二种情况下，状态机可以被用来作为一个协议规格说明，显示在类或者接口上操作激发的合法次序。在这样一个状态机里，转换被调用事件触发而它们的动作调用期望的操作。这意味着在这一点，调用者被允许调用操作。协议状态机不包括说明操作本身行为的动作。它显示了在特定的次序下，哪个动作可以被调用。这样的状态机说明了有效的操作顺序。这是状态机在语言（从计算机科学的语言理论）里作为顺序生成器的一个应用。这样的状态机被用来作为系统设计的约束。它不能直接执行，也没有说明如果发生非法的顺序会出现何种情况，因为不应该发生非法的顺序。系统设计者有责任保证只有合法的顺序发生。第二种应用比第一种更为抽象，因为第一种情况在可执行的方式下说明了所有情况的响应。但第二种情况通常比较方便，尤其在高层次和过程编码中。

318. state machine view(状态机视图)

处理各个元素在其生命期中的行为规格说明的系统的视图。这个视图包含状态机。在动态视图里它和别的行为视图松散地结合在一组里。

319. statechart diagram(状态图)

显示一个状态机（包括简单状态、转换、嵌套组成状态）的图。它最初是由 David Harel 发明的。

见state machine。

320. static classification(静态分类)

泛化的一个语义变更，其中对象不能改变类型，也不能改变角色。选择静态分类或者是动态分类是一个语义变更点。

321. static view(静态视图)

对系统中的事物及它们之间的静态关系特征化的总体模型的视图。它包含类元和它们相互之间的关系：关联、泛化、依赖和实现。有时被称为类视图。

China-pub.com
下载

语义

静态视图显示了系统的静态结构，特别是存在事物的种类（如类和类型）、它们的内部结构、相互之间的关系。尽管静态视图可能包含具有或者描述暂时性行为的事物的具体发生，但静态视图不显示暂时性的信息。

组成静态视图的最上层包括类元（类、接口、数据类型）、关系（关联、泛化、依赖、实现）、约束和注释。它也可以包含包和子系统作为有组织的单元。别的组成元素隶属于并包含在最高层元素里。

与静态视图相关并且在图里经常与之合并的是实现视图、部署视图以及模型管理视图。

静态视图可以与动态视图进行比较，动态视图补充静态视图并且建立在静态视图之上。

322. stereotype (构造型)

在一个基于已经存在的模型元素的类型的模型中定义的新型的模型元素。构造型可以扩展已存在元模型类的语义，但是不能扩展它的结构。

见constraint、tagged value。

见第14章，其中有预定义构造型列表。

语义

构造型代表了具有相同形式（如属性和关系）不同目的的已存在模型元素的一个变更。通常，构造型代表一个使用区分。除了有不同的外表外，构造型元素可能基元素的约束之外，还有附加的约束。人们希望代码生成器或者别的工具能够特别地对待构造型元素，如通过生成不同的代码。构造型的目的在于普通的建模工具，如模型编辑器或者仓库，在大多数情况下把构造型元素作为带有附加文本信息的普通元素看待，而在某些特定地语义操作里，如形式检查、代码生成和报表编写，才区分不同的元素。构造型代表了 UML的内在可扩充机制之一。

构造型从基本模型元素类导出。每个带有构造型的元素都具有基本模型元素类的特性。

构造型也可以从别的构造型特化而来。构造型声明是可泛化元素。子构造型有父构造型的特性。最后，每个构造型都建立在某个模型元素类的基础上。

构造型可以有请求标记列表，其中一些具有缺省值，当没有显示声明的标记值时就使用缺省值。可以声明每个标记的允许取值范围。每个带有构造型的元素必须有与标记列表对应的标记值。如果在一个构造型元素里没有显式说明，具有缺省值的标记会自动使用缺省值。

构造型可能在基本元素所具有约束之外，还会有附加的约束列表以增加新的条件。每个约束应用于每个带有构造型的模型元素。每个模型元素也受到应用于基本元素的约束的影响。

构造型是一种虚拟元模型类（即它在元模型里不明显），它在模型中而不是通过修改 UML的预定义元模型来增加。基于这个原因，新构造型的名称必须不同于已存在的 UML元类名称

China-pub.com 下载

或者其他构造型或者关键字。

任何模型元素最多只能有一个构造型。这条规则在逻辑上不是很重要，但是它简化了构造型的语义和表示法而在功能上没有任何实质的损失，因为构造型本身的多重继承是允许的。构造型可以是别的构造型的孩子。如果一个元素具有多个构造型，这种情况下，可以重新设计，使它只具有一个是其他构造型孩子的构造型。这偶尔会迫使建模者额外生成一个的假构造型以合并其他的构造型，但是我们认为一般情况下的简单性弥补了这种偶尔的不方便。

UML中预定义了一些构造型，其他的可能是用户定义的。构造性是 UML中的三个扩充机制之一。

见第14章，其中包括预定义构造型列表。

见constraint，tagged value。

表示法

构造型使用的普通表示法是用符号来表示基本元素，并在元素名（如果有）上面放一个关键字字符串。关键字字符串是构造型的名称，位于书名号中，书名号是法语或其他语言中所用的引用标记符号，如《 foo》（书名号看上去像两个尖括号，但它在大多数扩充字体集中是一个字符。大多数计算机都有这些特殊字符。印刷中可用两个尖括号作为替代）。关键字字符串通常放在模型元素名的上面或者前面。关键字字符串也可以用做列表里的元素。在这种情况下，它应用于随后的列表元素直到别的字符串代替它，或者一个空的构造型字符串清空它。注意构造型名不能和作用于相同元素类型的预定义关键字相同（为避免混淆，预定义的关键字名应该避免用在任何构造型上，即使它作用在不同的元素之上且在原则上可以区分）。

为允许UML表示法有限的图形扩充，可以为构造型关联一个图标或者图形标记（如纹理或者颜色）。UML没有说明图形规格说明的形式，但是存在很多位图和格式，可以在图形编辑器里使用（尽管它们的可移植性是个问题）。图标可以用在两个方面。一种情况下，它可以在构造型所基于的基本模型元素的符号里代替或者补充构造型关键字字符串。例如，在一个类矩形里，它被放在名称部分的右上角。在这种形式下，条目的普通内容可以在它的符号里看到。另一种方式，整个模型元素符号压缩为一个图标，图标里有元素的名称，或者把元素的名称放在图标的上面或下面。包含在基本模型元素符号里的其他信息被省略。

图13-170 显示了构造类的各种表示方法。

UML避免使用图形标记，如颜色，因为它们给某些人（色盲）和某些重要的设备（如打印机、复印机、传真机）带来了挑战。任何 UML符号都不要求使用这样的图形标记。用户可以为了他们自己的目的（如在某个工具里高亮显示）自由使用图形标记，但是应该知道这在交流上的限制，在必要的时候应该使用规范表示法。

构造型声明。构造型的分类层次可以在类图中显示。但是，这是一个元模型图，必须和普通的模型图区分开来（通过用户和工具）。在这样的图里，每个构造型显示为带有关键字《stereotype》的类符号（矩形）。泛化关系可以显示扩展的元模型层次（如图 13-171）。鉴于扩展内部元模型层次的危险性，具体的工具可以，但不是必须，把这种能力在类图里显示出来。

China-pub.com
下载

对构造型名的声明不是普通建模者所必须要求的，但是它可以作为一种支持。

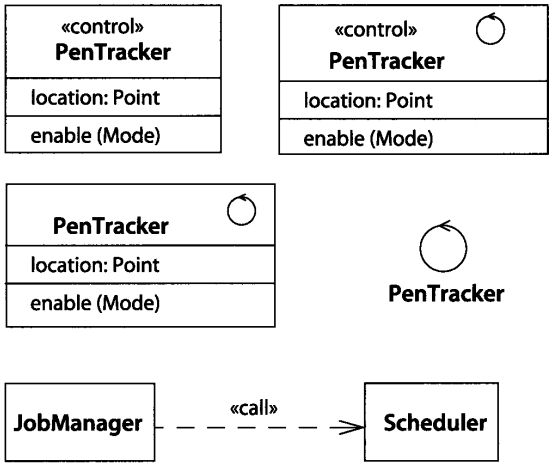


图13-170 各种构造型表示法

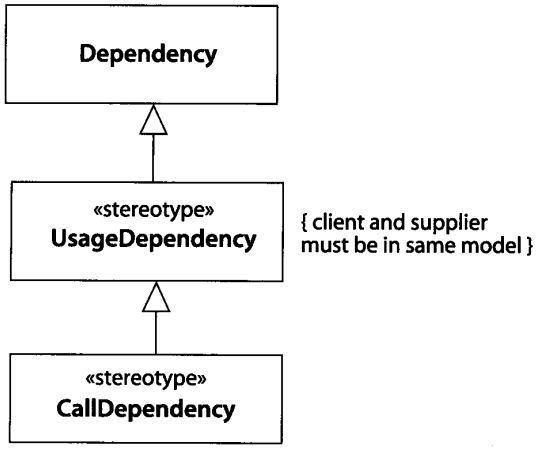


图13-171 构造型声明

323. string(字符串)

文本字符的一个序列。字符串表示的细节取决于实现，可能会包含支持国际字符和图形的字符集。

语义

很多语义特性，尤其是名称，把字符串作为它们的值。字符串是在某适合于显示模型的

China-pub.com 下载

信息的字符集合里的字符序列。字符集合可能包含非罗马字母和字符。 UML没有说明字符串的编码，但它假定编码足够通用以允许任何合理的使用。原则上，字符串的长度应该是没有限制的，任何实际的限制必须大到不受约束。字符串也应该包含各种自然语言的字符的可能性。身份（名称）应该全部由一个有限字符集里的字符构成。注释和其他类似的没有直接语义内容的描述性字符串可以包含别的种类的媒体元素，如图、图片或者视频剪辑，以及别的种类的嵌入文档。

表示法

图形字符串是具有一定实现灵活性的简单表示元素。在某些语言里，它被认为是线性顺序的字符串，可能包含各种类型的嵌入文档。人们希望它可以支持多种人类语言的使用，但是细节留给编辑工具去实现。图形字符串可以附属于直线或别的符号。

字符串被用来显示具有字符串值的语义特性，也用来对其他语义特性的值编码以便显示。从语义字符串到表示字符串的映射是直接的。从别的语义特性到表示字符串的映射由语法控制。例如，属性的显示表示法对名称、类型、初始值、可见性以及作用域进行编码而形成一个显示字符串。

对编码的非规范扩展是可能的 —— 例如，属性可以用C++的表示法显示。但在这些编码中，有的会丢失一些模型信息，所以工具应该以用户可选的方式支持它们且同时要保持对规范UML表示法的支持。

字体和字体大小是独立于字符串的图形标记。它们可以为各种模型特性建立代码，这些模型特性有的由文档提出，有的留给工具或者用户。例如，斜体字显示抽象的类和抽象的操作，下划线显示类作用域特征。

工具可以以各种方式处理长字符串，如截取一个固定的长度、自动换行以及插入滚动条。我们希望如果需要，可以找到办法保留全部的字符串。

324. structural feature(结构特征)

模型元素的静态特征，如属性或者操作。

325. structural view(结构视图)

一个总体模型的视图，该模型着重于系统中对象的结构，包括它们的类型、类、关系、属性和操作。

326. stub state(桩状态)

子状态机引用状态中的一个伪状态，子状态机引用状态确定被引用子状态机中的状态。

China-pub.com
下载

见stubbed transition，submachine，submachine reference state。

语义

到子状态机引用状态的转换激活目标子状态机的初始状态。但是有时候，希望一个子状态机中到别的状态的转换。桩状态位于子状态机引用状态中并确定子状态机中的状态。转换可以在桩状态和包容状态机的状态之间连接。如果到桩状态的转换激发，则子状态机中被确定状态被激活。如果子状态机中的某个状态处于活动，那么从确定这个状态的桩状态输出的转换即触发的候选者。不允许相同子状态机引用状态中的桩状态之间的连接。

表示法

从桩状态输出或者到桩状态的转换用到子状态机引用状态或者从子状态机引用状态输出的桩转换 —— 即一个箭头 —— 表示箭头开始或者结束于代表子状态机引用状态的状态符号里的一个短条。该条标有名称，该名称必须和被引用子状态机里的状态名称匹配。

后文的图13-173显示了子例程引用状态里的一个桩，图13-174显示了对应的子状态机的定义。

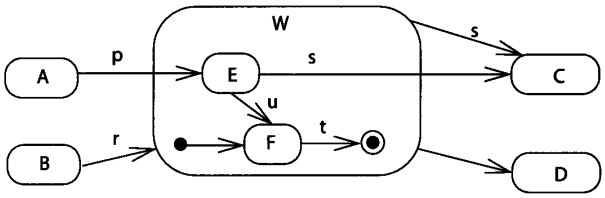
327. stubbed transition(桩转换)

到组成状态的转换的一种表示法，但是细节被省略。

见stub state。

表示法

桩用位于封闭状态边界内的一条小的垂直线表示（如图 13-172）。桩可以标有状态的名称，



可被抽象表示成

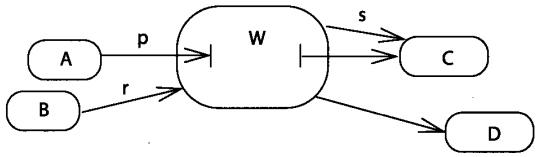


图13-172 桩转换

China-pub.com 下载

但是当细节被隐藏时，名称通常被省略。它表示转换连接到一个隐藏的内部状态。到初始状态的转换或者离开终止状态的转换不能使用桩。桩表明额外的子状态存在于模型里但是在这个图里被省略。

子状态机引用状态中的桩转换（如后文图 13-173）引用对应子状态机定义中的一个状态（如后文图13-174）。这不是隐藏细节的情形，故必须包括桩名称。

328. subclass(子类)

泛化关系中另一个类的子，即更为具体的描述。父类被称为超类。

见generalization，inheritance。

语义

子类继承它的超类的结构、关系和行为，而且可以对它进行增加。

329. submachine(子状态机)

可以作为别的状态机的一部分被调用的状态机。它不附属于类而是一种状态机子例程。它的语义效果像是复制它的内容而后插入引用它的状态。

见state，state machine，submachine reference state。

330. submachine reference state(子状态机引用状态)

引用子状态机的一个状态，其一个副本取代子状态机引用状态而成为封闭状态机的一部分。它可以包含桩状态，桩状态用于确定子状态机中的状态。

见state，state machine，stub state。

语义

子状态机引用状态等同于插入子状态机的一个副本以代替引用状态。

表示法

子状态机引用状态用带有如下形式标签的状态符号表示：

include *submachine-name*

转换用指向子状态机引用状态中桩状态的箭头表示。箭头被画为桩转换 —— 即在一个交叉条上结束的箭头。交叉条标有被引用子状态机中状态的名称。

China-pub.com
下载

示例

图13-173显示了包含子状态机引用状态的状态机的一部分。该状态机向有帐户的顾客售票。它必须确认顾客的身份，这是它的工作的一部分。确认顾客是别的状态机的需求，所以它被作成一个独立的状态机。图13-174 显示了状态机 Identify的定义，它被用做别的状态机的子状态机。子状态机的普通入口将读顾客的卡，但还存在一种显式的入口状态，由售票员手工输入顾客的名字。如果确认过程成功，则子状态机在它的终止状态终结。否则，它转到状态 Failure。

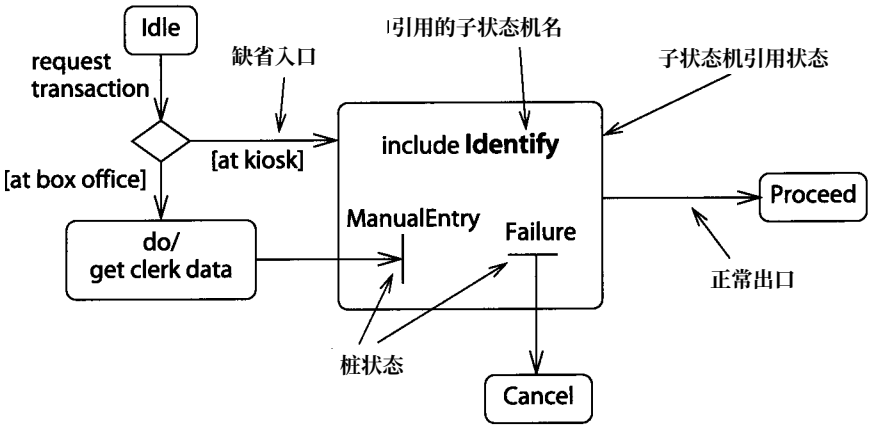


图13-173 子状态机引用状态

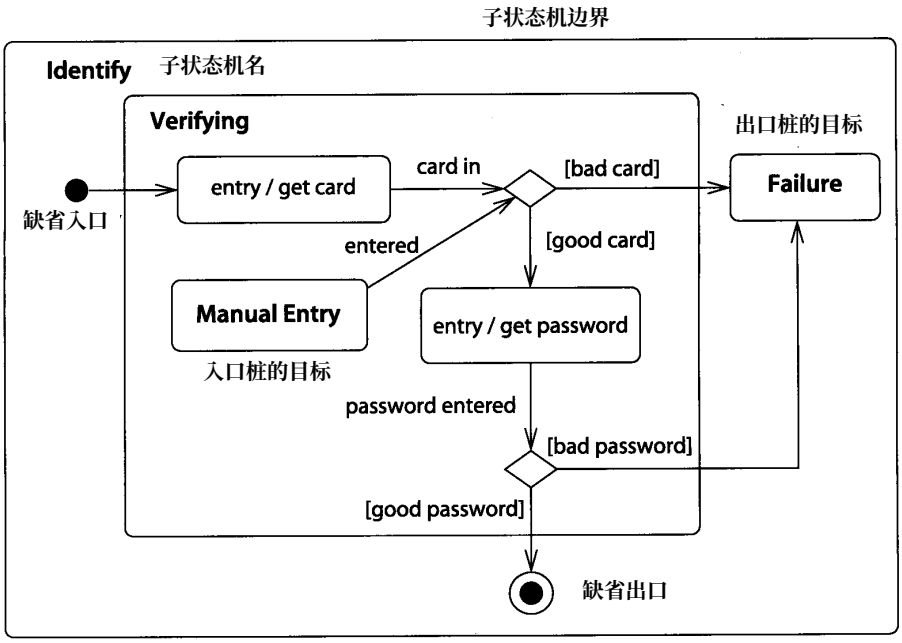


图13-174 子状态机定义

China-pub.com 下载

在图13-173里，子状态机引用用带有关键字 include 和子状态机名的状态图标表示。子状态机的普通入口用指向其边界的箭头表示。转换激活子状态机的初始状态。普通出口用从其边界输出的完成转换表示。如果子状态机正常结束，则转换激发。

显式状态ManualEntry 的入口通过到子状态机引用符号中的桩的转换来表示。该桩标有子状态机中目标状态的名称。同样，显式状态 Failure的出口通过从一个桩输出的完成转换来表示。到桩的转换可以被触发或者是无触发的。

331. substate(子状态)

作为组成状态一部分的一个状态。

见composite state，concurrent substate，disjoint substate。

332. substitutability principle(替代原理)

给定其类型声明为 X的变量或者参数的一个定义，任何 X的后代的实例都可以用做实际值而不会违反声明和使用的语义。换而言之，后代元素的实例可以替代祖先元素的实例（这是由 Barbara Liskov 提出的）。

见generalization，implementation inheritance，inheritance，interface inheritance，polymorphic，private inheritance。

讨论

替代原理的目的是使多态操作可以正常工作。这不是一个逻辑原理，而是提供一定程度封装的实用编程规则。泛化关系支持可替代性。

替代原理要求子不能去掉或者放弃它的父的特性。否则，子不能用于其父被声明的情形。

333. subsystem(子系统)

作为一个单元处理的元素的包，包括把整个包的内容作为一个单元处理时对其行为的规格说明。子系统被作为包和类建模。子系统有一个接口的集合，这些接口描述了它与系统其他部分的关系以及使用环境。

见 interface，package，realization。

语义

子系统是系统内可以被作为一个抽象独立单元的部分。它表示系统某一部分的突发行为。

China-pub.com
下载

作为一个整体，它有它自己的行为规格说明和实现部分。行为规格说明定义了它作为一个整体与其他子系统交互的突发行为。行为规格说明以用例和其他行为元素的形式给出。实现部分以构成其内容的附属元素的形式描述了行为的实现，并且作为被包容元素之间的协作集合给出。

系统本身构成最顶层的子系统。一个子系统的实现可以写成低级子系统的协作。在这种方式下，整个系统可以扩展为子系统的层次树，直到最底层的子系统以普通类的形式定义给出。

子系统可能包含结构元素和规格说明元素，如由子系统示出的用例和操作。子系统规格说明通过结构元素实现。子系统的行为实际上是它里面的元素的行为。

结构

对一个子系统的规格说明包含指定为规格说明元素的元素，以及作为一个整体定义在子系统上的操作和接口。规格说明元素包括用例、约束、用例之间的关系等等。这些元素和操作定义了子系统作为突发实体所执行的行为，即它的各个部分协同工作的净结果。用例说明了子系统和外部参与者的完整的交互顺序。接口说明了子系统或其用例必须提供的操作。规格说明没有说明各个部分是如何交互以完成必要的行为的。

子系统里的其他元素实现它的行为。这可能包含各种类元和它们之间的关系。子系统实现元素之间的协作集合实现这个规格说明。通常，由一个或者多个协作实现每个用例。每个协作描述实现元素的实例如何协作执行由用例或操作说明的行为。在规格说明级别子系统所有的消息都必须被映射到它的实现元素和其他子系统的实现元素之间的消息上。

子系统是一个包，具有包的特性。尤其是子系统的引入像对包的描述那样工作，且子系统之间的泛化具有和它的内容相同的可见性。

表示法

子系统用在子系统名上包含关键字《subsystem》的包符号（带有小标签的矩形）表示（如图13-175）。

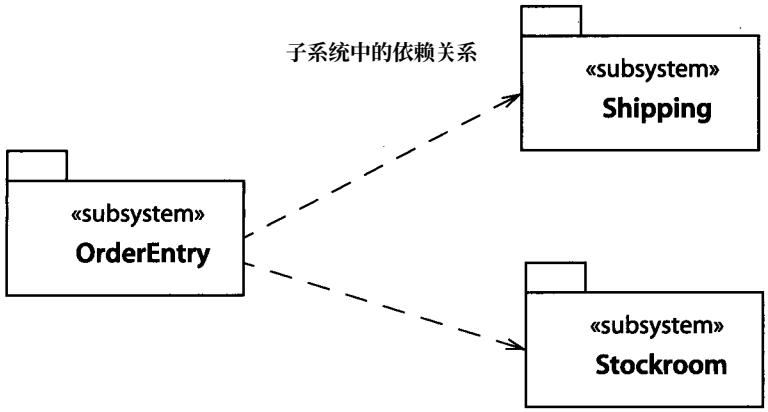


图13-175 子系统

China-pub.com 下载

讨论

考虑一个子系统，它处在设计元素的突发分组里，如逻辑类。构件是实现元素的突发分组，如实现级类。在很多情况下，子系统作为构件实现。这简化了从设计到实现的映射。所以，这是一种常用的构架方法。而且，很多构件被实现为直接实现构件接口的支配类。在这种情况下，子系统、构件、类可以有完全相同的接口。

334. subtype(子类型)

其他类型的子的类型。更中性的术语“子”可以用于任何可泛化元素。

见generalization。

335. summarization(概要)

将元素集合的特性过滤、合并和抽象到它们的包容器上，以给出系统的一个更高级别、更抽象的视图。

见package。

语义

包容器，如包和类，可以具有导出特性和关系这些特性和关系是对其内容的特性和关系的。这使得建模者能在一个更高级别而更少细节，因而也更为容易理解的水平上来更好地理解系统。例如，两个包之间的依赖表明在两个包中的至少一对元素之间存在依赖关系。概要比初始信息有更少的细节。可能存在一对或者多对依赖关系，这些依赖关系由包级别的依赖关系所表示。在任何情况下，建模者知道对一个包的修改可能会影响其他的包。如果需要更多的细节，则建模者一旦注意到高层的概要就可以仔细检查其内容。

同样，两个类之间的使用依赖关系通常表示它们的操作之间的依赖关系，如调用一个操作（不是方法！）的类之上的方法。很多类一级的依赖关系是从操作和属性之间的依赖关系导出。

一般而言，在包容器上总结的关系表示至少存在一个内容之间的关系。通常它们并不表示所有被包含的元素都参与关系。

336. superclass(超类)

泛化关系中其他类的父，即更加普通的元素规格说明。父类被称为超类。

China-pub.com
下载

见generalization。

语义

子类继承它的超类的结构、关系和行为，并且可以进行增加。

337. supertype(超类型)

超类的同义名。更加中立的术语“父”可以用于任何可泛化的元素上。

见generalization。

338. supplier(提供者)

提供可以被其他元素调用的服务的元素。对比：client。在表示法中，提供者位于虚线依赖关系箭头的头部。

见dependency。

339. swimlane (泳道)

为了对活动的职责进行组织而在活动图上进行的分块。泳道并没有一个固定的含义，但是它们经常对应于业务模型中的组织单元。

见activity graph。

语义

活动图中的活动状态可以基于它们的表示法而被组织成各个部分，称为泳道。泳道是为组织活动图而由状态组成的分组。每个泳道代表特定含义的状态职责的部分，例如，负责工作流步骤的业务组织。建模者可以在任何适合的条件下使用它们。如果使用泳道，那么它们将划分状态图的各个状态。

每个泳道有一个与其他泳道不同的名称。在 UML 里它没有额外的语义，但是它可以带有一些现实世界的含义。

表示法

活动图通常可以被划分为泳道，每个泳道通过垂直实心线与它的邻居泳道相分离（如图13-176）。每个泳道代表整个活动的部分高级职责，而整个活动可能在最后由一个或者多个对

China-pub.com 下载

象实现。泳道的相对次序在语义上没有重要意义，但是可能会表示现实世界里的某种关系。每个活动状态被赋到一个泳道上，并且在泳道里被取代。转换可能会穿过泳道，对转换路径的路由没有意义。

因为泳道是到任意种类的划分，所以如果几何上的区域安排不可行，那么还可以通过别的方式实现。可以使用颜色或者仅使用标记值来表明各个划分区域。

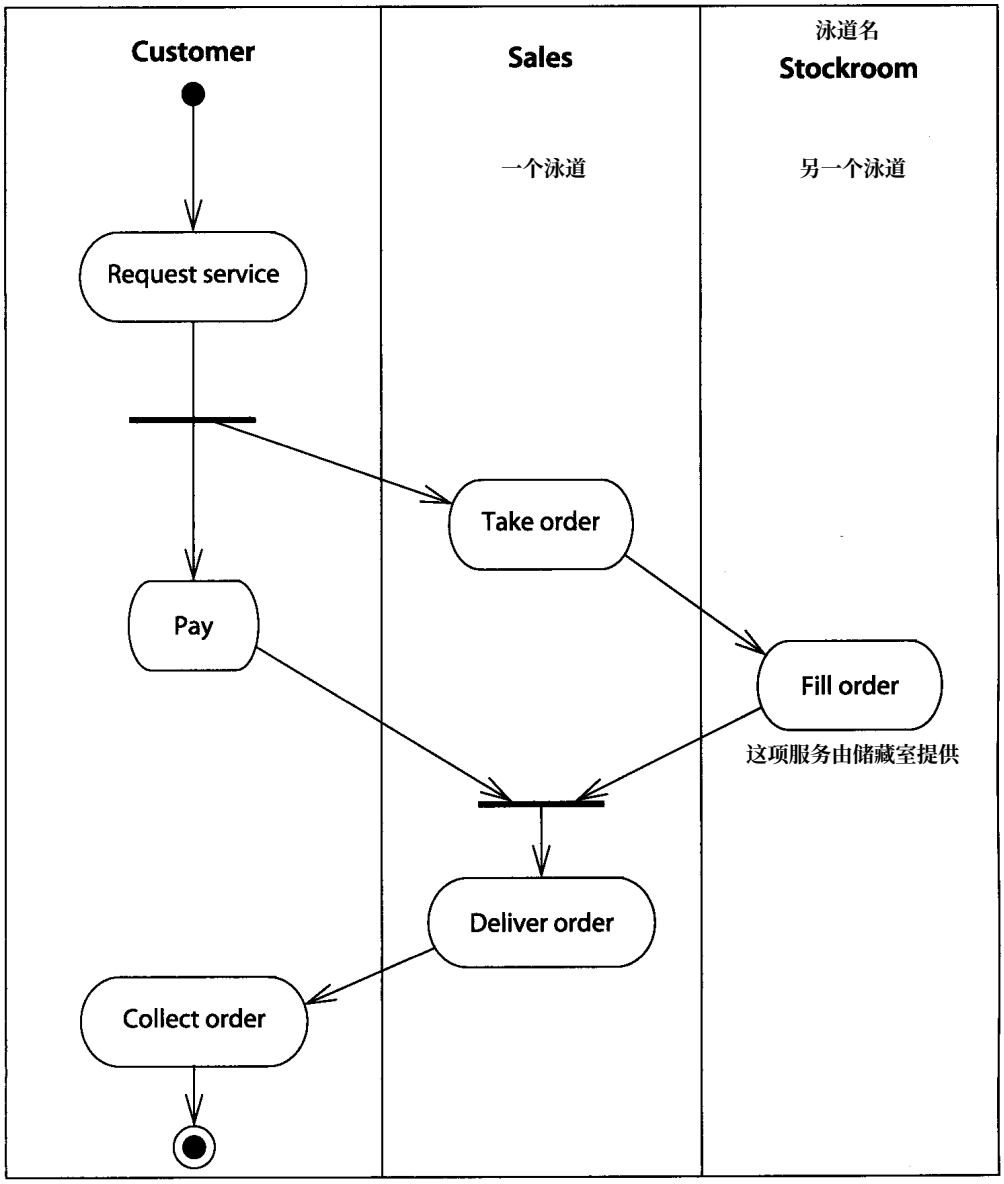


图13-176 活动图上的泳道

China-pub.com
下载

340. synch state(同步状态)

一个特殊的状态，它可以启动状态机中两个并发区域之间的控制同步。

见complex transition，composite state，fork，join，state machine，transition。

语义

组成状态可能由几个并发区域组成，每个区域有自己的状态集。当进入一个并发组成状态时，每个并发区域都变成活动的。在每个并发区域里，都存在一个控制线程，并且每个区域独立于其他的区域（并发的含义）。但是，偶尔也会要求并发区域之间的控制同步。一种方法是一个区域里的转换带有一个依赖于别的区域里的某个状态的监护条件。这种方法对于相互执行，包括共享资源是有用的，但是它不能捕获如下情况：一个区域里的活动在另一个区域中有影响后果。为了捕获这种情况，可以使用同步状态。

同步状态是一个连接两个并发区域的特殊状态。区域可以是对等的 —— 即两个属于相同的组成状态的并发区域 —— 或者它们可嵌套在对等区域中的任何深度。但这两个区域不能顺序相连。

一个转换把一个区域里的分叉的输出连接到同步状态的输入上，另外一个转换把同步状态的输出连接到另外一个区域的结合的输入上。换而言之，同步状态是一个缓冲区，间接地把一个区域里的分叉连接到另一个区域里的结合上。分叉和结合都必须在各自的区域里有一个输入状态和输出状态。

第一个区域中转换的激发被同步状态记住，直到第二个区域中结合转换激发。如果结合转换上的条件在同步状态变为活动之前被满足，那么结合必须等到第一个区域中的转换被激发。换而言之，它代表第二个区域中的一个转换，该转换直到第一个区域中的转换激发后才能被激发。注意因为每个分叉和结合在自己的区域里必须有一个输入和输出，所以同步状态不会改变每个并发区域中基本的顺序行为，也不会改变形成组成状态的嵌套规则（除了同步状态和它的弧不属于任何并发区域而属于它们的组成父状态之外）。

生产者-消费者问题是同步状态的一个典型实例。生产者激发一个激活同步状态的转换，消费者具有一个在激发前必须要求同步状态的转换。

如果到同步状态的输入转换是循环的一部分，那么第一个区域可能超过第二个区域。换而言之，同步状态可能有多个环（使用 Petri 网的术语）。因此，不同于普通状态，同步状态代表了一个计数器或者队列（如果信息在区域之间流动则为后者 —— 例如，同步状态是一个对象流状态）。缺省情况下，同步状态可以有无限多个环，但是建模者可以为同步状态可以带有的环的数目定义一个上界。如果同步状态的能力被超出，则为一个运行时错误。大多数情况下，这个上界是无限或者 1，后者表示一个简单的锁存器。为 1的上界只有在可以保证不会发生超限的情况下被使用。这是建模者的责任。

如果当封闭的组成状态退出时，在同步状态上存在环，那么这些环将被销毁。当进入组

China-pub.com 下载

成状态时，同步状态为空。

可能存在多个进入同步状态的输入弧，但是它们必须来自相同顺序区域里的分叉。同样，可以存在多个离开同步状态的输出弧而在顺序区域结合。因为每个区域都是顺序的，故不会存在多条弧的冲突危险。

同步状态可以是一个对象流状态。在这种情况下，它代表从一个区域传到另外一个区域的值构成的队列。

表示法

同步状态用一个小圆圈来表示，圆圈中上界，上界可以是一个整数或者是一个表示无限的星号(*)。从同步条(一根粗线)符号到同步状态有一个转换箭头，还有一个从同步状态到另外一个区域里的同步条的转换箭头（如图 13-177）。

同步状态最好画在两个区域的边界之间，但并不总是可以做到这样（两个区域并不相邻），在任何情况下，连接的拓扑结构都没有二义性。

在活动图中，每个转换弧代表一个状态。因此，可以从分叉的输出画一个箭头到结合的输入，而不必显式地显示同步状态（但同步状态需要显示它的边界）。

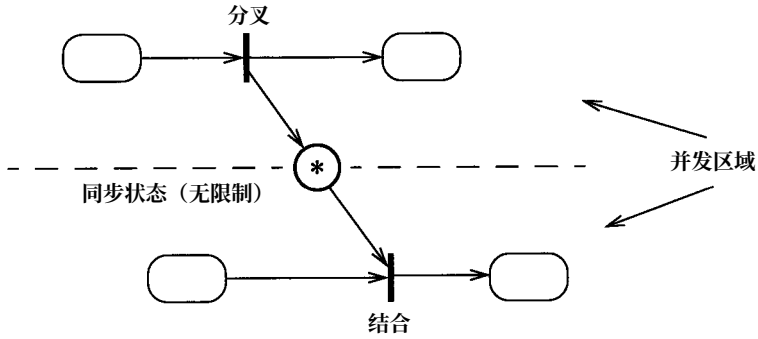


图13-177 同步状态配置

示例

图13-178显示了购票情况的状态图。除了必须在选座之后才能计算款项并寄出外，购票和收钱并发进行。这种同步通过在 Pick Seats 和 Post Charges 之间插入一个同步状态来显示。在Pick seats 之后存在一个分叉，因为它后面有 Print tickets 和同步状态。Print tickets 不必等待同步。在Post charges 前有一个结合，因为它必须等待Validate account 和同步状态。当Print tickets 和 Post charges 都结束后，该组成状态结束，Mial tickets 被执行。

该同步状态具有一个为 1 的上界。不需要更大的上界，因为该组成状态的每次执行只存在一个同步。

China-pub.com
下载

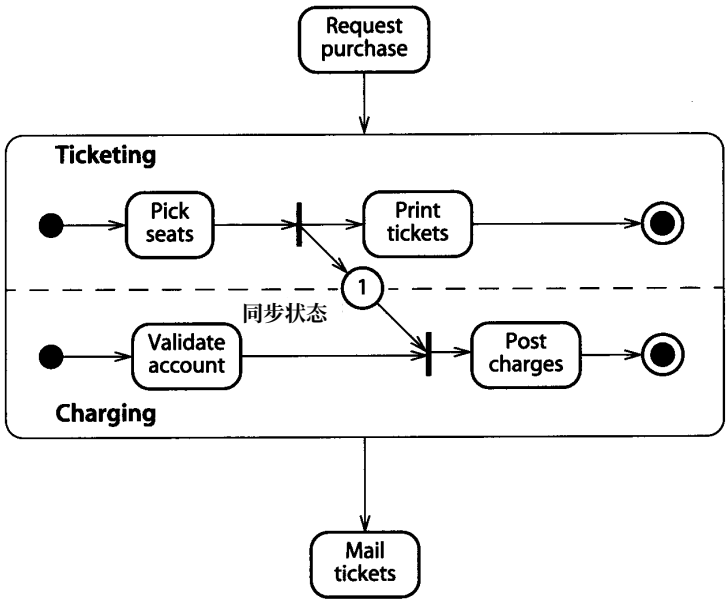


图13-178 单个订单的同步状态

图13-179显示了订单填写处理过程的批处理形式。在这种变体中，很多订单离线填写。订单由一个服务器填写，收钱由另外一个服务器处理。在订单填写之前不能收钱，但是订单可以超过收钱而使同步状态有一个无限的上界。这是经典的生产者-消费者情况。

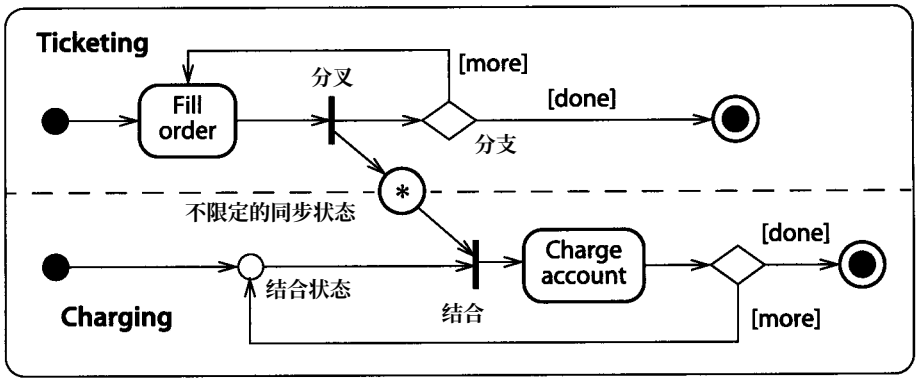


图13-179 带有无限同步状态的生产者-消费者情况

讨论

同步状态为生产者-消费者问题提供了具有最少花费的建模的能力，并且比普通的并发结

China-pub.com 下载

构更具安全性，因为每个并发区域始终保持一个控制线程。因为并发父状态退出的时候会清空同步状态，所以不存在超限的危险（如果上界无限）。但如果在包含分支的循环里使用同步状态，则存在挂起的危险：一个区域已经结束而另外的区域还在等待一个永远不会到达的同步环。如果每个状态都处在等待从对方发出的环的分支里，则存在发生死锁的可能。在允许决定的并发系统里，没有办法完全避免这样的情况。即使没有同步状态，由于不定问题的存在，也不能保证终结。

341. synchronous action(同步动作)

发送对象停下来等待响应的一个请求，一个调用。对比： asynchronous action。

342. system(系统)

组织起来以完成一定目的的连接单元的集合。一个系统可以用一个或者多个视点不同的模型描述。系统是“完整模型”。

语义

系统由一个高级子系统建模，该子系统间接包含共同完成现实世界目的的模型元素的集合。

表示法

系统可以用一个带有构造型《system》的包表示。但是很少需要作为一个单元来显示一个系统。

343. tag(标记)

标记值对中的选择值。它代表在建模时定义的特性的名称。

344. tagged value(标记值)

附在一个元素上,具有一定信息的标记值对。

见constraint，stereotype。

见第14章，其中给出了预定义标记列表。

China-pub.com
下载

语义

标记值是可以附在任何元素上（包括模型元素和表示元素）以负载各种信息的选择器值对，通常对于元素的语义是次要的，但可能对建模企业很重要。选择器被称为标记，它是一个字符串值。每个标记代表一个可以用于一种或者多种元素的特性。在模型中的任何元素上，标记名只能最多出现一次。标记的值可能是各种类型的，但是被编码为字符串。对值的解释是建模者和建模工具之间的约定。可以把标记值实现为由标记进行索引的查询表以提高访问效率。

标记值代表任何以文本方式表达的信息，通常被用来存储项目管理信息，如元素的作者、测试状态或某个元素对最终系统的重要性（标记可能是 author, status 和 importance）。

标记值代表了对UML元类的元属性的适度扩展。这不是一种完全适用的扩展机制但是为了后端工具的利益而被用来向已经存在的元类中增加信息，如代码生成器、报表编写器和激励。为避免混淆，标记应该与它们所应用的模型元素的已经存在的元属性区别开。可以使用一种工具简化这种检查。

UML中预定义了一些标记，其他的可以是用户定义的。标记值是允许把任何信息附加到模型上的扩展机制。

表示法

标记值用如下方式表示：

tag = value

其中，*tag* 是标记的名称而 *value* 是一个字面值。标记值可以和其他的特性关键字一起被包含在一个由括弧包含，逗号分割的特性列表中。

可以声明一个关键字代表一个带有特定值的标记。在这种情况下，关键字可以单独使用。如果不存在标记，则当作标记的一个其他的合法值：

tag

示例

{author=Joe,status=tested,requirement=3.563.2a,suppress}

讨论

大多数模型编辑程序提供基本的功能以把标记值作为字符串进行定义、显示和查找，但并不用它们来扩充UML的语义。但是，后端工具，如代码生成器、报表编写器，可以读取标记值并灵活地改变它们的语义。注意标记值列表是一种老思想——例如，Lisp 语言里的特性列表。

标记值是一种向模型附加非语义性的项目管理和跟踪信息的方法。例如，标记 author 可能带有一个元素的作者，标记 status 可以带有开发状态，如 incomplete, tested , buggy和

China-pub.com 下载

complete。

标记值还可以用来把依赖于实现语言的控制附加到 UML 模型上而不用把语言的细节建立到UML中去。代码生成标志、提示、附注都可以编码为标记值而不会影响低层的模型。对于各种语言来说，相同模型上的多个标记集合是可能的。模型编辑者和语义分析者都不必理解标记 —— 它们可以作为字符串来处理。后端工具，如代码生成器，可以理解并处理标记值。例如，标记值可以命名用来重载多重性为多的关联的缺省实现的包容器类。

标记值满足了很多种类的信息必须附加在模型上的需要，但它不是一种完整的元模型扩展机制。标记形成一个平的命名空间，它必须采用某些约定以避免冲突。它们没有提供声明值的类型的方法。它们的目的也不是对建模语言本身进行语义扩展。标记有点像元模型属性，但是它们不是元模型属性，还没有像元模型元素一样被规范化。

标记的使用，正像编程语言库中函数的使用一样，可能需要一定时期的发展，在这个过程中，在不同的开发者之间可能会发生冲突。经过一定的时间，某些标准使用可能会出现。UML不包含标记的“注册”，也没有提供这样的期望：早期的标记使用者可能会保留它们以防止将来用在别的地方。

345. target scope(目标作用域)

对一个值是实例还是类元的声明。

见scope。

讨论

目标作用域主要用来存储作为属性值或者关联目标的类。它的用途是受限的。单独的词汇“作用域”表示所有者作用域。

346. target state(目标状态)

转换的激发形成的状态机的状态。在对象处理完一个可以激发转换的事件后，对象就处在转换的目标状态（或者是目标状态集，如果是一个有多个目标状态的组成状态）。它不适用于内部转换，因为内部转换不会引起状态的改变。

见transition。

347. template(模板)

一个参数化的模型元素。要使用模板，这些参数必须绑定（在建模时）实际值。同义词：

China-pub.com
下载

parameterized element。

见binding，bound element。

语义

模板是对一个带有一个或者多个未绑定的形式参数的元素的描述符。因此它定义了一系列的潜在元素，每个元素都通过把参数绑定到实际值来声明。一般而言，参数是代表属性类型的类元，但是它们也可以代表整数甚至操作。模板里的附属元素用形式参数来定义，所以当模板被绑定的时候，它们也会被绑定。

模板类是对一个参数化类的描述符。模板体可能包含代表模板本身的缺省元素以及形式参数。通过把参数绑定到实际值可以生成一个实际类。模板类中的属性和操作可以用形式参数来定义。模板也可能与其参数之间存在关系，如关联或者泛化。当模板被绑定的时候，其结果是被绑定的模板类和被绑定到关系参数上的类之间的关系。

模板类不是一个直接可用的类（因为它有未绑定的参数）。必须把它的参数绑定到实际值以生成实际类。只有实际类才可以作为关联的父亲或者目标（但是允许从模板到另一个类的单向关联）。模板类可能是一个普通类的子类，这意味着所有通过绑定模板而形成的类都是给定类的子类。它也可以是某个模板参数的子，这意味着被绑定的模板类是被当作参量传递的类的子。

参数化可以被用在其他模型元素上，如协作甚至整个包。这里给出的对类的描述以明显的方式应用于别的建模元素上。

模型的内容不直接受模型的良性结构规则的影响。这是因为它们具有直到被绑定才具有完整语义的参数。模板是一种二级的模板元素 —— 不是直接为系统建模的元素，而是为别的元素建模的元素。所以模板的内容在系统的语义之外。绑定模板的结果是受良性结构规则影响的普通模型元素和目标系统中的标准元素。模板一定程度的良性结构规则可以从对于其绑定结果必须为良性结构的考虑中导出，但是我们不会试图列出它们。在一定意义上，它的内容被复制，参数被实际值所代替。结果成为有效模型的一部分，好像它已经被直接包含。

其他种类的类元，如用例或者信号，也可以被参数化。协作也可以被参数化，然后成为模式。

表示法

模板用位于类矩形或者别的模型元素右上角的小虚线矩形表示。虚线矩形包含类的形式参数的列表。每个参数有一个名称和一个类元。列表不能为空（否则不存在模板），尽管在表示中它可能被省略。参数化类的名称、属性和操作出现在类矩形中，但是也可以包含形式参数。其他种类的参数化元素相似处理。形式参数可以出现在模板体内以显示一个由某个参数确定的相关类。

参数的语法如下：

China-pub.com 下载

name:type

其中 *name* 是参数的身份，具有模板作用域。*type*是一个为参数指定类型表达式的字符串。

如果省略类型名称，则认为是一个类元的类型表达式，如类名或者数据类型。其他参数类型（如Interger）必须显式地显示出来，其结果必须是一个有效类型表达式。

图13-180显示了带有一个整数参数和一个类参数的模板。该模板和它的每个参数有一个关联。

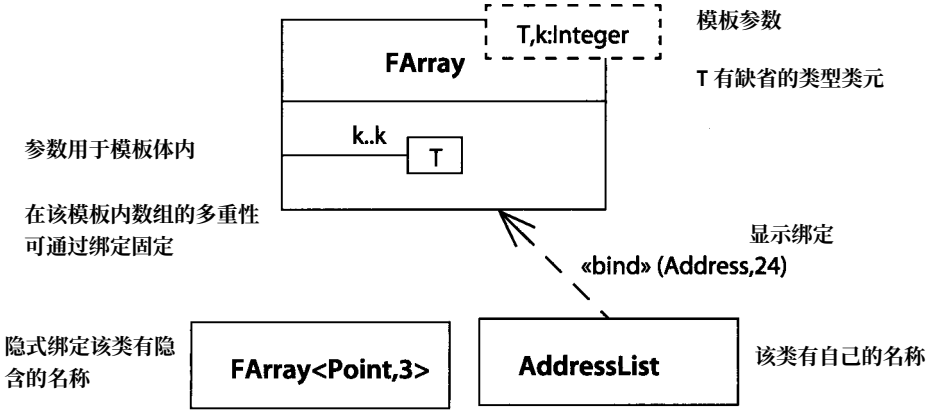


图13-180 把参数作为引用使用的模板表示法

讨论

有效模型是绑定所有模板而形成的隐含模型。在有效模型中模板参数没有意义，因为那时它们已经被绑定。它们只可以在模板体作用域里使用。这足以处理包含在参数化元素中的组成元素，如参数化类中的属性或者操作。

参数化元素外部的元素有更多的困难。例如，一个类可能与别的类有关联或者泛化关系。如果那些类是模板的参数，则它们不能是有效模型的一部分，也不能是普通类的一部分。因此，参数化元素包含一个代表一个模型段的体。模型段不是有效模型的一部分。它是模板本身的一部分，它可能包含模板参数，如表示某个类的参数。当模板被绑定时，体被隐含地复制，参数被实际值代替，而副本成为有效模型的一部分。每个模板的实例都会对有效模型进行增加。图13-181显示了一个例子。

模板体隐含地包含了一个表示实例化的模板元素本身的元素——例如，通过绑定模板而生成的类。这个隐含元素可以用来构造关系（如关联或者泛化）以模板化参数。在这种表示法中，参数画在模板边界里，到模板边界内部的连接表示一个到隐含实例化模型元素的关系。当模板被实例化后，这些则变成有效模型中被绑定元素（刚刚生成）和元素（已经存在且为模板参数）之间的关系。

模板可以是其他元素的子。这意味着由它生成的绑定元素都是给定元素的子。例如，在

China-pub.com
下载

图13-181中，每个变量数组（VArray）是一个数组（Array）。所以，Varray<Point>是Array的一个子，Varray<Address>是Array的一个子，以此类推。

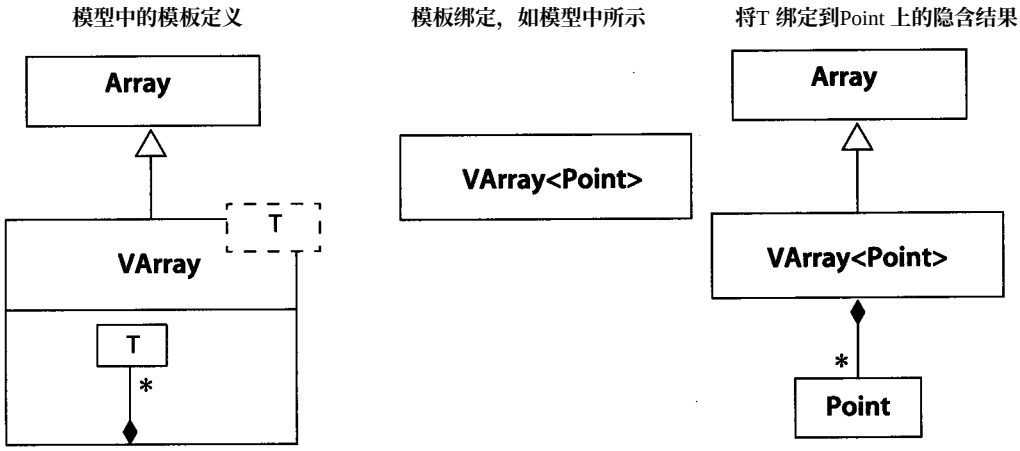


图13-181 与其参数存在关系的模板

模板通常不能是其他元素的父。这意味着通过绑定模板而生成的每个元素都是其他元素的父。尽管会有人为这种情况赋予一定的含义，但这是不真实的。

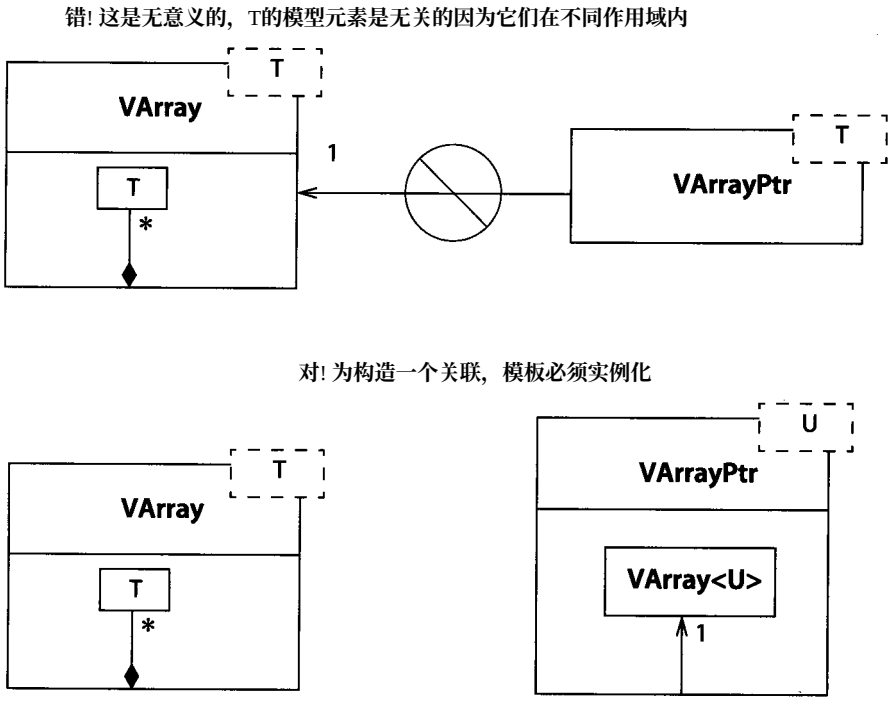


图13-182 模板之间的关联

China-pub.com 下载

两个模板不能只因为共享相同的参数名而有关联（这样做意味着第一个模板的实例化都和第二个模板的每个实例化存在关系，这不是通常所期望的。这一点经常被过去的作者所误解）。参数只在它自己的模板里有效。在两个模板中为一个参数使用相同的名称不会使它们成为相同的参数。通常，如果两个模板有必须相连的参数化元素，则一个模板必须在另一个模板体中实例化（回忆一下，模板隐含地在它自己的体中实例化。所以，两个模板都有效地在体内实例化，关系存在于实例化地元素之间）。图13-182显示了定义这种关系的一个错误的和一个正确的尝试 —— 在这种情况下，有一个指向相同类型的参数化数组的参数化指针类。

一种类似的方法是声明一个是另一个绑定于相同参数的模板类的子的参数化类。另一种方法是在只有一个参数的第三方模板中实例化这两个模板。该参数用来绑定对方模板的一个副本。然后可以在模板的实例化副本之间建造关联。在大多数情况下，并不需要这样做，因为关系可以在任何一个模板里声明。

348. thread(线程)

经过一个程序、动态模型或者控制流的其他表示执行的单个路径。也是把一个主动对象实现为轻量进程的构造型。

见active object，complex transition，composite state，state machine，synch state。

349. time(时间)

表示一个绝对或者相对时刻的值。

见time expression。

350. time event(时间事件)

表示某个时间表达式被满足的事件，如一个绝对时间的发生或者对象进入一个状态后的一段给定时间。

语义

时间事件是一个依赖于时间段因而取决于时钟的存在的事件。在现实世界中，时钟是隐含的。在计算机中，它是一个物理实体，在不同的计算机中可能存在不同的时钟。时间事件是从时钟到系统的一条消息。注意对于现实世界的时钟或虚拟内部时钟（在后一种情况下，对于不同的对象可能存在差异）可以定义绝对时间或者流逝时间。

China-pub.com
下载

时间事件可能建立在绝对时间（一天的时间或者系统中的时钟设定）或者相对时间（进入某个状态或者某个时间发生后的流逝时间）的基础上。

表示法

时间事件不像信号那样声明为一个命名事件。相反，时间表达式仅用做转换的触发。

讨论

在任何实际实现中，时间事件不是来自整体 —— 它们来自系统内或者系统外的某个时钟对象。因此，它们与信号几乎无法区分，尤其是在实时和分布式系统里。在这样的系统里，使用哪个时钟的问题必须解决 —— 不存在像“实时”这样的东西（它在真实世界里也不存在 —— 问一下爱因斯坦）。

351. time expression(时间表达式)

计算结果为一个绝对或者相对时间值的表达式。用于定义时间事件。

语义

大多数时间表达式或者是进入某个状态后的流逝时间或者是特定绝对时间的发生。别的时间表达式必须以一种特别的方式定义。

表示法

流逝时间。表示进入包含转换的状态后的一定时间段的事件，用关键字 after后跟一计算(在建模时)一定长度时间的表达式表示，如：

after (10 seconds)

after (10 seconds since exit from state A)

如果没有说明起始点，那么从进入包含转换的状态开始计算。

绝对时间。表示绝对时间的发生的事件可以用关键字 when后跟一由括弧括起的包含时间的布尔表达式表示。

when (date =Jan.1,2000)

352. timing mark(时标)

事件或者消息发生时间的一种表示法。时标可用在约束中。

China-pub.com 下载

语义

时标根据消息的名称而形成一个表达式。在交互中，消息被给予一个名称，时标表达式可以根据它来形成。在下面的表达式中 *message*是消息的名称：

message.sendTime()　　消息发送的时间

message.recieveTime()　　消息接收的时间

表示法

时标表达式用文本表示。

示例

下面的约束限制了产生一个拨号音所要求的时间。

{dialtone.sendTime() - offhook.sendTime() <1 second}

353. trace(跟踪/踪迹)

表示历史开发过程或者两个元素之间的特殊模型关系的依赖关系，这两个元素表达了相同的概念，但没有说明从一个导出另外一个的具体规则。这是最不特殊的依赖关系，并且它具有最少的语义。它主要用于在开发过程中对人的思考过程的提醒。

见dependency，model。

语义

跟踪是依赖的一种，它表明在不同的含义级别上表示相同概念的两个元素之间的连接。它不表示模型中的语义。它表示具有不同语义的元素之间的连接 —— 既不同含义层次的不同模型上的元素之间的连接。在元素之间没有显式的映射。通常它表示在不同开发阶段捕捉一个概念的两种方式之间的连接。例如，属于相同主题的不同变体的两个元素可以由跟踪连接。跟踪不能表示运行时的实例之间的关系。相反，它是模型元素本身之间的一个依赖关系。

跟踪主要应用于在系统的开发过程中跟踪已经变化的需求。跟踪依赖关系可以联系两种模型（如用例模型和设计模型）中的元素或者相同模型中不同形式的元素。

表示法

跟踪由带有关键字《trace》的依赖关系箭头（虚箭头，其尾部在新元素上而头部在老元素上）表示。但是通常元素处在不同步显示的不同模型中，所以，实际上关系在工具中经常实现为一个超级链接。

China-pub.com
下载

354. transient link(暂时链)

只存在一段有限时间的链，如一个操作的执行。

见association，collaboration，usuage。

语义

在执行过程中，一些链存在一段有限的时间。当然，如果时间跨度足够大，几乎所有的对象和链的生命期都是有限的。但是有些链只存在于一定的限制语境中，如一个方法的执行过程中。过程参量和局部变量可以由暂时链表示。把所有这样的链建模成关联是可能的，但是关联上的条件必须非常粗略地进行声明，失去了在限制对象合并中的精度。这样的情况可以使用协作来建模，协作是对对象和存在于特定语境的链的配置。

来自协作的关联角色可以认为是只在行为实体（如一个过程）的执行过程中存在的一个暂时链。它在类模型中作为使用依赖关系出现。要得到完整的细节，需要参考行为模型。

表示法

暂时链用带有一个附属于链接角色以表示各种不同实现的构造型的关联表示。可以使用下列构造型：

《parameter》	过程参数。
《local》	过程的局部变量。
《global》	全局变量（在整个模型或包里可见的）；如果可能应该尽量避免，因为它违反了面向对象的精神。
《self》	自链接（对象给自己发送消息的能力，在对象中是隐含的，只仅用在具有消息流的动态情况下显示）。
《association》	关联（缺省，不必说明除非为了强调）；它不是暂时链，列出来只是为了完整性。

355. transient object(暂时对象)

只在生成它的线程的执行过程中存在的对象。

356. transition(转换)

用于表示一个状态机中两个状态之间的一种关系，即一个在某初始状态的对象通过执行指定的动作而进入第二种状态，当然，这种转换需要某种指定的事件发生和指定的监护条件

China-pub.com 下载

得到满足。在这个状态的变化中，转换被称作激发。简单转换只有一个源状态和一个目标状态。复杂转换有多个源状态和（或）多个目标状态。它表示在一系列并发的活动状态中或在一个分叉型或结合型的控制下所发生的转化。内部转换有一个源状态但没有目标状态。它表示对没有状态转化的事件的反应。状态和转换是状态机的顶点和节点。

见state machine。

语义

转换表示在一个对象的生命历史中所有的状态之间可能有的路径以及在状态变化时发生的动作。转换显示某状态的对象对于一个事件的发生进行反应的路径。状态和转换是状态机中的顶点和弧，用来描述一个类元中所有实例的可能的生命历史。

结构

转换包括源状态、事件触发器、监护条件、动作和目标状态，有的可以省略。

源状态。源状态是被转换所影响的状态。如果某对象处于源状态，而且此对象接受了事件的触发或监护条件得到了满足，则此状态的输出转换将激发。

目标状态。目标状态是转换结束后的主动状态，它是主对象要转向的状态。目标状态不会用于不存在状态变化的内部转换。

事件触发器。事件触发器是一个事件，它被处于源状态的对象所接受后，转换只要得到监护条件的满足即可激发。如果此事件有参数，则这些参数可以被转换所用，也可以被监护条件和动作的表达式所用。触发转换的事件则成为当前事件，而且可以被随后的动作访问，这些动作都是由事件初始化的运行到完成步骤的一部分。

没有明确的触发器事件的转换称作完成转换（或无触发器转换），在结束时被状态中的任一内部活动隐含地触发。组成状态通过到达它的终止状态而表明其完成。如果一个状态没有内部活动或嵌套状态，而状态在任意入口动作执行之后被输入，那么完成转换被立即触发。要注意到，完成转换必须满足监护条件才能激发。若完成发生时监护条件不成立，则隐含的完成事件会被消耗，而且以后即使监护条件再成立，转换也不会激发了（这种行为用一个改变事件来建模）。

注意，状态机中某个事件的所有表现必须有同样的特征标记。

监护条件。监护条件是一个布尔表达式，它于一个事件的操作触发了转换时计算，事件包括完成转换上隐含的完成事件。如果一个事件发生时状态机正在执行运行到完成步骤，则此事件被保存，直到步骤完成和状态机静止。否则，此事件被立即处理。如果表达式的值为真，转换可以激发；如果表达式的值为假，转换不能激发；如果没有转换适合激发，事件被忽略。这种情况并非错误。有着不同监护条件的多重转换可以被同样的事件触发。若此事件发生，则所有监护条件都被测试。若有多个监护条件为真，也只有一个转换会激发。如果没有给定优先权，则选择哪个转换来激发是不确定的。

China-pub.com
下载

注意，监护条件的值只在事件被处理时计算一次。如果其值开始为假，以后又为真，转换不会被激发，除非有另一个事件发生，且此时的监护条件为真。注意，监护条件不是持续监控一个值的正确方法。这种情况应该使用改变事件。

如果转换没有监护条件，则监护条件被认为是真，而且一旦触发器事件发生，转换被启动。若有几个转换同时启动，只有一个可以激发，其选择是不确定的。

为了方便，一个监护条件能够被拆解成一系列简单的监护条件。实际上，这一系列监护条件可以是某个触发器事件或监护条件的分支，每一分支都是一个单独的转换，每一转换被有着不同的但却是有效的监护条件的（单独的）触发器事件所触发，这是对触发器事件的路径上所有监护条件的与。这条路径上所有的表达式都会在转换激发前计算。转换不能部分地激发。事实上，一套独立的转换可以部分地共享它们的描述。图 13-183是一个示例。

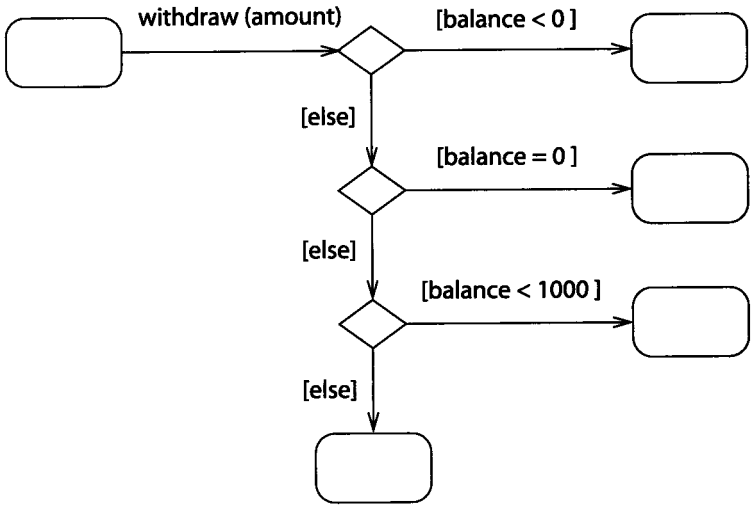


图13-183 监护条件树

注意，监护条件树和将转换排序的能力仅仅是为了方便，因为同样的效果还能通过一套独立的转换实现，只要每个转换有它自己的互斥监护条件。

动作。转换可含有描述一个动作的动作表达式。这个表达式是为了过程计算，过程计算可以对拥有状态机的对象产生影响（而且，间接地，其他对象也会被影响）。动作表达式可以使用触发器事件的参数，以及所拥有的对象的属性和关联。在整个运行到完成步骤中，若当前事件被包括以后的非触发器段、出口动作和入口动作在内的事件激发，则触发器事件可用。

动作可以是一个动作序列。单个动作是原子的——即它不能在外部被终止，而必须在任何其他动作或事件的处理之前被完整地执行。动作应该持续最少时间以免状态机被中止。任何一个在动作执行期间接收到的事件都会被保存，直到动作结束，此时接受到的事件被计算。

分支。为了方便，几个共享同一触发器事件却有着不同监护条件的转换能够在模型和表示法中被分在同一组中，以避免触发器或者监护条件的相同部分被重复。这仅仅是一种表示

China-pub.com 下载

上的方便，不会影响转换的语义。

见branch，查看其表示法的详细说明。

表示法

转换用从一个状态（源状态）指向另一个状态（目标状态）的实线箭头表示，其上标有转换字符串。图13-184显示了两个状态间的一个转换以及一个分成多段的转换。

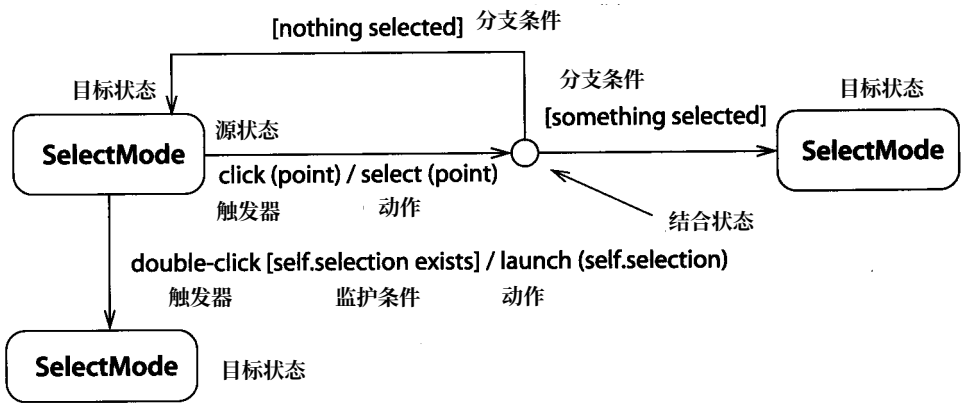


图13-184 转换

内部转换用状态符号中的转换字符串表示。转换字符串格式如下：

$\overline{name:}_{opt}\ event\text{-}name\ _{opt}\ \overline{(parameter\text{-}list)}\ _{opt}\ \overline{[guard\text{-}condition]}\ _{opt}\ \overline{/action\text{-}list}\ _{opt}$

*name*可用于引用转换，尤其是用于形成时标。其后有一个冒号。

*event-name*命名一个事件，其后跟有参数列表。如无参数，参数列表可以省略。事件名和参数清单对于完成转换是省略的。参数清单格式如下：

$\overline{name:type}\ _{list}$

*guard-condition*是一个布尔表达式，由触发器事件的参数、属性和用状态机描述的对象的链等条目组成。监护条件还可以包括状态机中并发状态的测试或某些可达对象的显式指定的状态——例如，[in Statel]和 [not in State 2]。状态名可以完全由包含它们的嵌套状态来限定，产生如下形式的路径名：State 1::State 2::State 3。在整个状态机中，如果在不同的组成状态区域有同样的状态名出现，则可用此形式。

*action-list*是一个过程表达式，它在转换激发时得到计算。它可由操作、属性、拥有对象的链、触发器事件的参数等条目组成。动作包括调用、发送及其他种类的动作。动作列表可包含多个被分号隔开的动作子句：

$action\ _{listi}$

分支。一个转换可包含一个触发器事件，后跟结合状态树，用小圆圈表示。这相当于一套独立的转换，每一个相当于通过树的每一条路径，其监护条件是此路径上所有条件的与。

China-pub.com
下载

只有每条路径的最后一段可有一个动作。

结合状态如果表示分叉或合并，也可用菱形表义，在意义上没有不同。

讨论

转换表示从一状态到另一状态的原子变化，也可能伴有原子动作。转换是不可中断的。转换上的动作应当短暂，通常是琐碎的处理，如赋值声明和简单的计算。

357. transition phase(转换阶段)

软件开发过程的第四个阶段，在该阶段中，配置实现系统准备在现实世界语境中执行。在这个阶段，完成部署视图以及前面各个阶段尚未完成的视图。

见development process。

358. transition time(转换时间)

见timing mark。

359. trigger(触发器)

一个事件，其发生使一个转换能够激发。其名称可以是个名词（事件本身），也可以是个动词（事件的发生）。

见completion transtion,transition。

语义

每个转换（除了完成转换是因内部活动的完成而激发）都会将事件作为结构的一部分而引用它。如果某个事件在对象处于一个包含输出转换的状态中时发生，而此输出转换的触发器即事件或是事件的祖先，那么，转换的监护条件会被测试。如果条件满足，则转换能够激发；如果缺少监护条件，则它必须得到满足。如果有多个转换满足条件，实际上也只有一个可以激发，而选择哪一个可能是不确定的（如果对象有多个并发状态，从任何一个状态开始的转换都可能激发，但最多只有一个转换可以激发）。

注意，监护条件只在触发器事件（包括隐含的完成事件）发生的时候被测试一次。若某事件发生时没有转换能够激发，此事件仅仅会被忽略。这不是错误。

在一个监护条件或者附属于转换或目标状态的入口动作的动作中，触发器事件的参数都能被使用。

China-pub.com 下载

在转换之后的运行到完成步骤执行当中，触发器事件对转换的分步骤中的动作作为当前事件保持可用。在一个入口动作或多段转换的一个后续段中，这个事件的准确类型可能是不可知的，因此，此事件的类型会在使用多态操作或情况说明的某个动作中得到辨别。一旦得到了事件的确定类型，则其参数能够被使用。

表示法

触发器事件的名称和特征标记是转换标签的组成部分。

见transtion。

触发器事件可以在表达式中由保留字 current Event访问。这个关键字引用多段转换的首段的触发器事件。

360. triggerless transition(无触发器转换)

没有明确的事件触发器的转换。当它离开一个普通状态时，表示一个完成转换，即该转换是被活动的完成而非某个明确的事件所触发的。当它离开一个伪状态时，表示一个转换段，若其先前的段已经完成动作，此转换段可被自动遍历。无触发器转换用于将初始状态和历史状态与它们的目标状态相连接。

361. tuple(元组)

值的有序列表。通常情况下，此术语的含义是一系列相似格式的列表（这是一个标准的数学术语）。

362. type(类型)

作为形容词：属性、参数或变量的值必须保留而声明的类元。其实际值必须是类型的一个实例，或是其后代的实例。

作为名词：用于说明实例集（如对象）以及适用于对象的操作的类的构造型。类型可以不包含任何方法。对比：interface,implementation class。

类型和实现类

类能被构造成类型或实现类（虽然对它们也可以无差别地进行支配）。类型被用来说明一个对象的域以及适用于这些类的操作，但没有定义这些对象或操作的物理实现。类型可以不包含任何方法，但它可以提供操作的行为规格说明。它也可以包括属性和关联，这是为了说

China-pub.com
下载

明操作的行为。类型的属性和关联不决定它的对象的实现。

实现类定义了物理数据结构和对象的方法，与传统语言中实现的一样，如C++和Smalltalk。如果实现类包括所有与类型所具有的行为一样的操作，则实现类被认为实现了一个类型。实现类可以实现多个类型，而多个实现类可以实现同样的类型。实现类的属性和关联不必与类型中的。实现类可以在其物理属性和关联中为其操作提供方法，而且可以声明其他类型中所没有的附加操作。

对象必须是只有一个实现类的实例，此实例指定了对象的物理实现。然而，一个对象是有多个类型的实例。如果对象是只有一个实现类的实例，那么此实现类必须实现此对象作为一个实例的全部类型。如果动态分类可用，则在生命期内，对象可以获得和失去类型。通过这种方式使用的类型刻画了一个可变的角色，对象可以接受，以后再放弃。

尽管类型和实现类的在使用上并不相同，但其内部结构是一致的，所以它们被建模为类构造型。它们支持泛化、替代原理以及属性、关联和操作的继承。类型可只特化类型，而实现类也可只特化实现类。类型只能通过实现与实现类相连。

表示法

说明一个无明显特征的类时是没有关键字的。类型用关键字《 type》说明，而实现类用关键字《implementation class》说明。图13-185显示了一个例子。

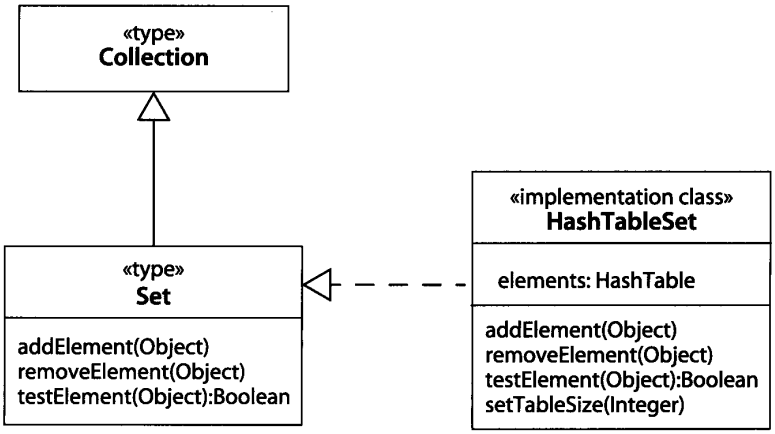


图13-185 类型和实现类的表示法

依靠实现类来实现的类型通过使用实现关系来建模，用带有实三角箭头的虚线表示（一个“虚线泛化箭头”或一个“有实箭头的依赖关系”）。这种符号表示了在操作上的继承关系，但没有表示结构的继承（属性或关联）。实现可以在任何两个类元间使用，其中一个类元支持另一个类元的操作，但依靠实现类实现的类型是一种常见的用法。

讨论

针对如C++和Smalltalk这样的传统程序设计语言，类型和实现类在某种程度上受到一些限

China-pub.com 下载

制。在UML中，类的概念能够直接地以更加全面的方式得到应用。 UML对多重分类和动态分类都提供支持，去除了对类型和实现类之间存在区别的需要。然而，在用传统的语言编制代码时，这些区别的确有用。

363. type expression(类型表达式)

一个表达式，其值是指向一种或多种数据类型的引用。例如，在典型的程序语言里，属性类型声明即类型表达式。

示例

在C++中，类型表达式如下：

Person*

Order[24]

Boolean(*)(String,int)

364. uninterpreted(无解释的)

一个占位符，提供给一个或多个其实现不是由 UML定义的类型。每个无解释的值都有一个相应的字符串来表示。在任何物理实现中，如模型编辑，实现必须完成，所以其中不能有无解释值。

365. unspecified value(未指定值)

一个还没有被模型指定的值。

讨论

一个未指定值在特定的含义下完全不是一个值，而且除了其值不是必需的或其值是不相关的那些特性以外，非指定值不能在一个完成模型中出现。例如，多重性不能是未知的，它必须有值。无值相当多重性为多，因此 UML的语义不允许或不处理缺少值或未指定值的情况。

在另外一个含义中“未指定”是未完成模型的一个有用部分。它有这样的意思：“我仍然没有考虑过这个值，而我已经对它做了记录，以便我可以在后来赋给它一个值。”未指定显示说明了模型是未完成的。这是个有用的且被工具支持的功能。正是由于其本来的性质，这样的一个值不能在已完成的模型里出现，而且它并不关注对它的语义做出定义。在需要值的时候，如在代码生成的时候，工具能自动地为未指定值提供一个缺省值，但这只是一个方便的

China-pub.com
下载

措施，而不是语义的一部分。未指定值是UML语义之外的概念。

同样，为特性提供的缺省值没有语义定义。模型中的特性只不过有一个值。对于新创建元素的特性，工具可以自动地为其提供值。需要再次说明，这只是一个由工具带来的操作上的简便，不是UML语义的一部分。语义完全的UML模型没有缺省值或未指定值。

366. usuage(使用)

一个依赖关系。在此关系当中，一个元素（客户）为了其功能或实现，需要另一个元素（提供者）的存在——通常是因为实现的原因。

见collaboration，dependency，transient link。

语义

使用依赖关系是这样一种状况：某个元素（客户）为了其实现或功能，需要另一个元素（提供者）的存在。所有的元素必须存在于同等的意义级别上——即，在抽象或实现级上不涉及位移（如在一个分析级的类和一个实现级的类之间的映射）。使用依赖关系经常涉及实现级元素，如C++的包含文件，为此它隐含了编译器结果。使用可以被进一步构造，以表明依赖关系的精确性质，如调用属于另一个类的某个操作或实例化属于另一个类的某个对象。

表示法

使用由一个带有关键字《use》的虚箭头（依赖关系）来表示。箭头由客户（依赖）元素指向提供者（独立）元素。

讨论

使用通常与暂时链相符合——即类的实例之间的一种连接，这个连接不是始终都有意义或存在，只存在与某些语境中，如一个子过程的执行。在这种情况下，依赖关系构造不对所有的信息建模，除非它的确存在。协作构造提供了对这种关系在所有细节上建模的能力。

标准元素

call, create, instantiate, send。

367. use(使用)

使用依赖关系的表示法中的关键字。

China-pub.com 下载

368. use case(用例)

动作的顺序的规格说明，包括变更顺序和错误顺序，使系统、子系统或类能够通过与外部参与者间的交互而执行。

见actor，classifier。

语义

用例是一个连贯的功能性单元，由一个用消息顺序表示的类元（系统、子系统或类）提供，这些消息与被系统执行的动作在系统和一个或多个外部用户（表现为参与者）间交换。

用例的目标是要定义类元（包括一个子系统或整个系统）的一个行为，但并不显示类元的内部结构。每个用例说明类元提供给它的用户的一种服务，也即一种对外部可见的使用类元的特定方式。它描述由某用户以用户和类元间交互的观点来初始化的完整的顺序，以及由类元执行的响应。交互只包括系统与参与者之间的通信。内部行为或实现被隐藏。类元或系统用例的整个集合分割和覆盖它的行为，每个用例代表一部分量化的、有意义的并对用户可用的功能。注意用户包括人、计算机和其他对象。参与者是一个用户意图的理想化事物，并不是物理用户的代表。一个物理用户能映射到许多参与者，而一个参与者能代表多个物理用户的同一个方面。

见actor。

用例包括对用户请求做出响应的常规的主线行为，以及常规顺序的可能变体，如间隔顺序、异常行为和错误处理。其目的是描述在其所有变更中的一个连贯的功能，包括所有的错误条件。一个类元的用例集说明了使用此类元的所有不同的方式。为了方便，用例可以被分组打包。

用例是一个描述符，它描述潜在的行为。用例的执行是用例实例。用例的行为能够用附加的状态机或用文本代码（与状态机相同）来说明，也可以用一种非正式的文本描述。行为能用脚本集解释，但并不形式化。在开发的早期阶段，这可能已足够。

用例实例是一个用例的执行，由来自参与者的一个实例的一条消息初始化。作为对此消息的响应，用例实例执行一系列被用例指定的动作，如给参与者实例发送消息，不必只是给初始参与者。参与者实例可以向用户实例发送消息，而此种交互会持续下去，直到实例已经对所有的输入做出响应。当它不再得到输入时结束。

作为在与外部参与者的交互中的一个整体，用例是对系统（或其他类元）的行为的规格说明。在实现行为的系统中，协作实现了用例，内部对象之间的内部交互也由它来描述。

结构

用例可以有类元特征和关系：

特征。用例是类元，因此有属性和操作。属性用于表示用例的状态——即执行用例的过

China-pub.com
下载

程。操作表示用例能执行的一件工作，它不是直接从外部随时支取，但可以被用来描述用例对系统的影响。操作的执行可以与来自参与者的消息相关联。操作作用于用例的属性，而且间接作用于用例所附属的系统或类。

到参与者的关联。参与者和用例间的关联表明参与者实例与系统实例或类元实例之间的通信，而对一些对于参与者很重要的结果产生影响。参与者对类元的外部用户建模，因此，如果此类元是一个系统，则它的参与者是系统的外部用户。较低级子系统的参与者可能是整个系统中的其他的类。

一个参与者可以与多个用例通信 —— 即参与者可以请求系统的多个不同服务 —— 而且，一个用例可以与一个或多个提供服务的参与者通信。注意，两个被同一系统指定的用例不能相互通信，这是因为其中任何一个用例都为系统单独描述了一个完整的使用。它们可以通过共享的参与者间接地进行交互。

参与者和用例之间的交互能够用接口来定义。接口定义了参与者或用例可以支持或使用的操作。由同一用例给出的不同接口不需要互斥。

用例通过泛化、扩展和包含关系与其他用例相联系。

泛化。泛化关系将特化用例和更一般的用例联系起来。子用例继承父用例的属性、操作和行为顺序，而且可以增加属于自己的另外的属性和操作。子用例通过在任意点向父用例插入额外的动作序列而向父用例增加增量行为，它也可以修改某些继承操作和顺序，但这必须像任何一个重载一样地完成，以使父用例的目的得以保留下来。子用例之上的任何扩展或包含关系也可有效地修改继承自父用例的行为。

扩展。扩展关系是一种依赖关系。客户用例通过向基顺序插入额外的动作序列而向基用例加入增量行为。客户用例包含一个或多个分开的行为顺序段。扩展关系包含来自基用例的一列扩展点名称，在数量上与客户用例的段的数量一样多。一个扩展点代表基用例中的一个或一组位置，扩展可以在这位置上被插入。在扩展点上，扩展关系也可以有一个条件，以此来使用父用例的属性。当父用例的实例到达被扩展关系中的扩展点所引用的位置时，计算条件。若条件为真，则与之相符的子用例中的行为段被执行。若没有条件约束，则被认为是永真。如果扩展关系具有多个扩展点，那么条件只有在首段中的优先执行的第一个扩展点上计算。

扩展关系不会创建一个新的可实例化的用例。相反，它隐含地向初始的基用例增加行为。基用例隐含地包括了扩展行为。未扩展的初始基用例在其旧有的形式下是不可用的。换而言之，如果扩展了一个用例，则不能显式地在没有扩展的可能性时将基用例实例化。用例可以有多个扩展，这些扩展都适用于同一基用例，而且如果它们的条件分别得到满足，还能够被插入一个用例实例中。另一方面，一个扩展用例可以扩展多个基用例（或不同的扩展点的相同基用例），每一个都在其正确的扩展点上（或是扩展点的列表）。如果在同一个扩展点上有多个扩展，则它们执行的相对顺序是不确定的。

注意，扩展用例是不会被实例化的，而基用例必须被实例化以得到联合的基 -扩展行为。扩展用例可以或不可以被实例化，但不论是哪种情况，它都不包括基用例行为。

China-pub.com 下载

包含。包含关系表示的是：在一个客户用例的控制之下，对提供者用例在一个客户用例的交互顺序中的行为顺序的包含，而此客户用例正处于由客户的描述所指定的位置上。这是一个依赖关系，不是泛化，因为提供者用例不能在客户用例出现的位置上被替代。客户可以使用基用例的属性来获得值和通信结果。用例实例执行的是客户用例，当它到达包含点时，开始执行提供者用例，直到结束。然后，它重新开始执行客户用例，超过包含位置。在不同的执行之间，提供者用例的属性没有值是坚持不变的。

用例可以是抽象的，这意味着它不能直接在一次系统执行中被实例化。它定义一个行为的片断，而此行为是被具体用例特化或包含的，或者它是一个基用例的一次执行。如果它能被自己实例化，则它也可以是具体的。

行为。用例的行为顺序能够用状态机、活动图或某种可执行的文本代码来描述。状态机的动作或代码的声明可以调用用例的内部操作来说明执行的效果。动作也可以表示向参与者发送的消息。

用例可以用脚本或纯文本进行非形式化的描述，但这种描述是不严密的，而且只能用于人类解释。

用例的动作可以在调用由用例描述的类元的操作期间被说明。一个操作可以被多个用例调用。

实现。用例的实现可以用协作来说明。协作依靠用例所描述的类元中的对象而对用例的实现进行描述。每个协作描述系统各部分之间的语境，在系统中有一个或多个交互顺序发生。协作和它们的交互定义了系统中的对象怎样交互来完成指定的用例外部行为。

系统能够用各种抽象级别的用例说明。例如，说明系统的用例可以被精化为附属用例集，其中每个说明子系统的一个服务。由超级（较高级别）用例说明的功能完全可跟踪到由附属（较低级别）用例说明的功能。一个超级用例和一组附属用例在不同的抽象级别上说明同样的行为。附属用例协同提供超级用例的行为，这些附属用例的协同是由超级用例的协作来说明的，而且可以在协作图中表示。超级用例的参与者作为附属用例的参与者出现，且附属用例互为参与者。在一组实现了整个系统的嵌套的用例和协作中，这些关系将实现的结果层次化。

表示法

用例用一个包括用例名的椭圆形来表示。如果用例的属性和操作必须被显示出来，可以将用例绘制成一个带有关键字《use case》的矩形类元。图13-186表示的是一个用例图。

扩展点是一个用例内部的命名实体，描述了来自其他用例的动作序列可以被插入的位置。扩展点提供扩展和行为顺序文本之间的一个间接级别。扩展点引用了用例行为顺序中的一个或一组位置。对于使用此扩展点的扩展关系，这种引用能够被独立地改变。每个扩展点必须在用例中有一个独有的名称。扩展点可以在一个用例的区域里被列举出来，标题为 extension points（如图13-187）。

China-pub.com
下载

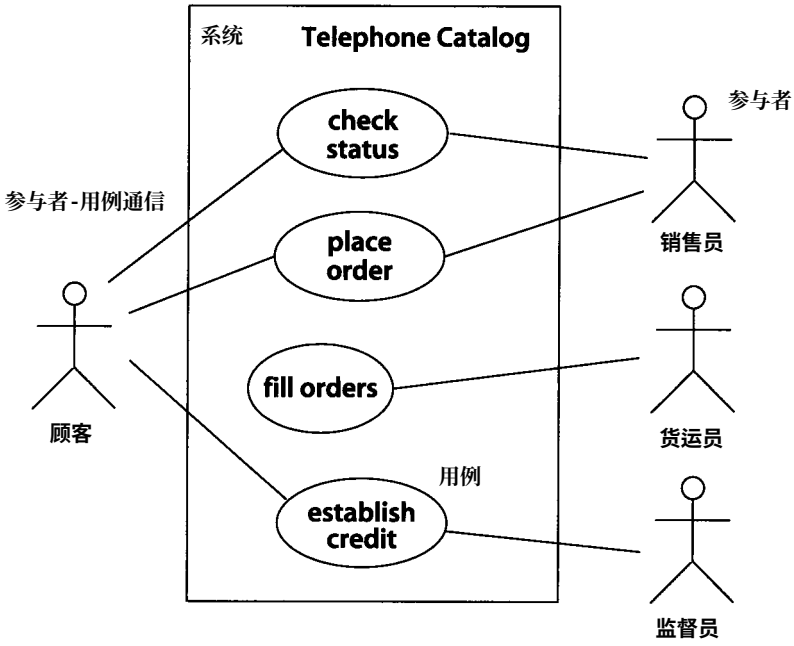


图13-186 用例和参与者

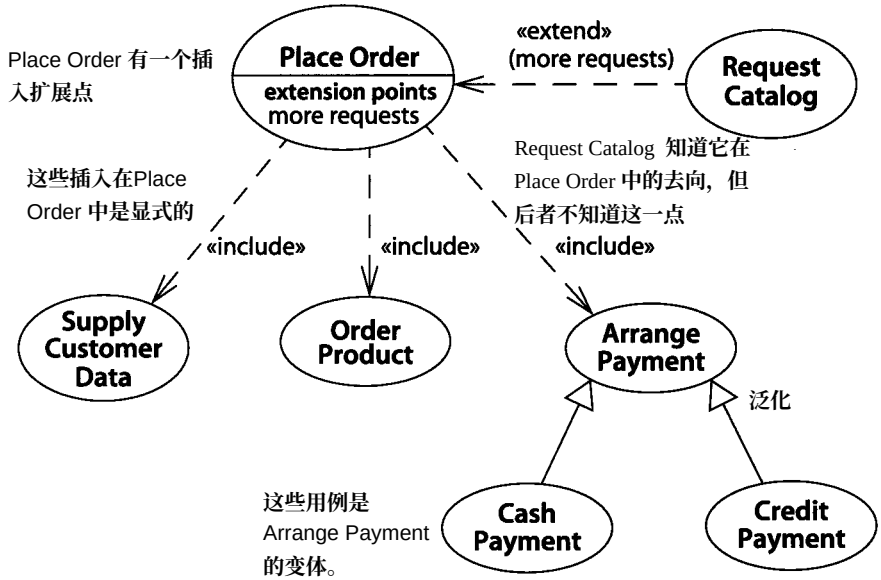


图13-187 用例关系

用例和参与者之间的通信关系用一个关联符号表示 —— 连接用例和参与者符号的一条实

China-pub.com 下载

线。因为通信是参与者和用例之间唯一的一种关联，所以关键字《 communication》通常可以被省略。通常情况下，由于参与者和用例唯一地定义了这个关系，因而在这条实线上不标注名称或角色名。

泛化关系用一个泛化箭头表示 —— 从子用例到父用例的一条实线，一个实三角箭头指向父用例。

扩展关系或包含关系用一个带有关键字《 extend》或《include》的依赖关系箭头表示 —— 带有条状箭头的一条虚线，指向客户用例。在扩展关系上也有一个扩展点名称列表（在图上可以隐藏）。

图13-187表示了各种用例关系。

行为规格说明。用例和它的外部交互顺序之间的关系通常用一个指向顺序图的超链来代表。此超链不可见，但可以在一个编辑器中移动。行为也可以用一个状态机或用附属于用例的程序设计语言文本来说明。自然语言文本可以当作一种非正式的规格说明。

见extend，其中提供了某些行为顺序的一个例子。

用例和其实现之间的关系可以表示成一个实现关系，此实现来自一个到协作的用例。但因为这些关系经常是在不同的模型中，所以通常用一个不可见的超链来代表。期望的情况是有一种工具可以支持“放大”用例以查看其作为一个协作的脚本和 /或实现的能力。

369. use case diagram(用例图)

表示处于同一系统中的参与者和用例之间的关系的图。

见actor，use case。

表示法

用例图是包括参与者、由系统边界（一个矩形）封闭的一组用例、参与者和用例之间的关联、用例间的关系以及参与者的泛化的图。用例图表示了来自用例模型（用例，参与者）的元素。

370. use case generalization(用例泛化)

一个用例（子用例）和另一个用例（父用例）之间的类元关系，描述了子用例与其他具有相同父用例的用例共享的特性，这是适用于用例的泛化。

语义

父用例可以被特化成一个或多个子用例，这些用例可以代表父用例的更多明确的形式

China-pub.com
下载

（如图13-188）。因为用例是一个类元，所以子用例要继承父用例的所有属性、操作和关系。这个继承操作的实现可以被实现子用例的协作进行重载。

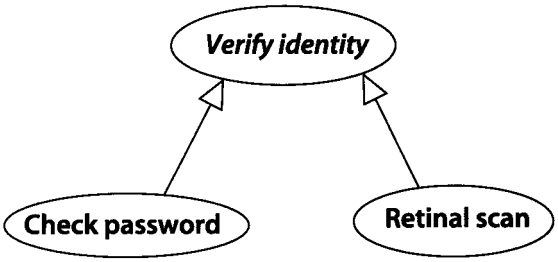


图13-188 用例泛化父用例Verify Identity的用例行为：

父用例是抽象的，无行为顺序。

Obtain password from master database
Ask use for password
User supplies password
Check password against user entry

一个具体的后代必须提供如下所示的行为。

Obtain retinal signature from master database
Scan user's retina and obtain signature
Compare master signature against scanned signature

图13-189 父用例和子用例的行为顺序

子用例继承父用例的行为顺序，而且可以插入额外的行为（如图 13-189）。父用例和子用例可潜在地实例化（如果它们不是抽象的），而同一父用例的不同的特化是独立的，这不同于扩展关系，在其中多重扩展都隐式地修改同一基用例。行为可以通过在继承自父用例的行为顺序中增加步骤而被加入到子用例中，也可以通过对子用例声明扩展和包含关系的方式来完成。如果父用例是抽象的，则其行为顺序可以有分段，这些分段在父用例中是显式不完整的，而且必须是由子用例提供的。子用例可以修改继承自父用例的步骤，但由于方法上的重载，这种能力必须小心使用，因为要保证父用例的意图必须保留下来。

泛化关系将子用例与父用例连接起来。子用例可以访问和修改由父用例定义的属性。

用例的替代意味着子用例的行为顺序必须包含其父用例的行为顺序。父用例顺序中的步骤不需要相互邻接，子用例可以在继承自父用例的行为顺序中插入额外的步骤。

对用例的多重继承的使用需要一个显式的规格说明，说明如何在父用例的行为顺序中插入步骤来生成子用例的顺序。

用例泛化可以使用私有继承来共享基用例的实现，而不完全替代，但这个功能应该保守

China-pub.com 下载

使用。

表示法

使用普通的泛化符号——从子用例到父用例的一条实线，空三角箭头指向父用例。

示例

图13-188显示了抽象用例Verify identity和其由两个具体用例的特化，用例的行为见图 13-189中。

371. use case instance(用例实例)

被一个用例指定的序列顺序的执行。

见use case。

372. use case model(用例模型)

一个模型，它描述的是系统或其他类元的功能需求。

见actor，use case。

语义

用例模型表示系统或其他类元的功能，表现为与系统的外部交互。用例图中显示了用例模型。

373. use case view(用例视图)

对于用例而言，系统中关系到指定行为的那个方面。用例模型是集中于此视图的模型。用例视图是松散地成组的建模概念中的一部分，就像动态视图。

374. utility(效用)

类的构造型，它以类声明的形式将全局变量和过程进行分组。效用的属性和操作分别成为全局变量和全局过程。效用不是一个基本建模结构，却方便了编程。它没有实例。

China-pub.com
下载

语义

效用的属性是全局变量，而且效用的操作是全局操作。由于作为类作用域成员，全局属性和操作能够被更好地模型化，所以对于面向对象的程序，效用不是必需的。此结构是为了与非面向对象的语言相兼容而提供的，如 C。

表示法

效用用类符号表示，在类名字符串之上有构造型关键字《 utility》。属性和操作表示全局成员。符号中没有声明类作用域成员。

375. value(值)

见data value。

376. vertex(顶点)

在状态机中，转换的一个源或目标。顶点可以是状态或伪状态。

377. view(视图)

模型的一个投影，它可从一个透视图或优势位置看到，而且它省略了与这个透视图无关的实体。这个词用在这里不是表示一个表示元素。相反，它包括语义模型和可见表示法中的投影。

378. visibility(可见性)

一个枚举类型，其值（public, protected 或private）说明了它所涉及的模型元素是否在其命名空间的外部可见。

见access，其中有应用于包间引用的可见性规则的讨论。

语义

可见性声明了建模元素引用一个元素的能力，此元素与引用元素处于不同的命名空间。可见性是一个元素和它的包容器之间的关系的一部分，包容器可以是包、类或某个其他的命名空间。有三个预定义的可见性：

public　　任一元素，若能够看见包容器，则也能够看见包容器指示的元素。

China-pub.com 下载

protected　　只有包容器中的元素或包容器的后代能够看见指示的元素，其他元素不能引用或使用它。

private　　只有包容器中的元素能够访问指示的元素。其他元素，包括包含容的后代的元素，都不能引用或使用它。

附加的可见性的种类可以为某些编程语言定义，如C++的implementation可见性（实际上，所有非公共可见性的形式都是由语言决定的）。使用附加的选择必须依靠用户与建模工具和代码生成器之间的约定。

表示法

可见性用放在模型元素名称前的一个特性关键字或一个标记符来表示：

public　　+

protected　　#

private　　-

可见性标记可以取消。缺少可见性标记说明可见性没有显示，而不是说未定义或是公共的。即使可见性没有显示，也有工具给新元素的可见性赋值。可见性标记是对一个可见性特性说明字符串完全形式的速记。

可见性也可以用关键字（public, protected, private）来说明。当一个内部列表元素应用于一整快属性或其他列表元素时，常用这种形式。

任何语言专用的或用户定义的可见性的选择必须用一个特性字符串或用一个工具专有的约定来说明。

类。在一个类中，可见性标记标在列表元素上，如属性和操作。这表明其他类能否访问这些元素。

关联。在一个关联中，可见性标记标在目标类（通过可见性的设置，能够被访问的关联端）的角色名上。这表明在远端的类能否通过关联到达带有可见性标记的一端。

包。在一个包中，可见性标记标在直接为包所包含的元素上，如类、关联和嵌套包。这表明另一个访问或引入第一个包的包能否看见这些元素。

379. well formed(良性构成)

指出一个模型是被正确构造的，模型满足所有预定义的和模型专有的规则与约束。这样的模型具有的语义是有意义的。没有良性构成的模型称为非良性构成。

China-pub.com
下载

第14章
标准元素

标准元素是为约束、构造型和标记而预定义的关键字。它们代表了通用效用的概念，这些通用效用没有足够的重要性或者与核心概念存在足够的差异使它们包含在 UML核心概念中。它们和UML核心概念的关系就如同内建的子例程库和编程语言的关系。它们不是核心语言的一部分，但它们是用户在使用这种语言时可以依赖的环境的一部分。列表中也包括了表示法关键字 —— 出现在别的模型元素的符号上但代表的是内建模型元素而不是构造型的关键字。为关键字列出了表示法符号。

本章交叉引用了第13章中的术语。

1. access(访问)

(stereotype of Permission dependency)

两个包之间的构造型依赖，表示目标包的公共内容对于源包的命名空间是可访问的。

见access。

2. association(关联)

(stereotype of AssociationEnd)

应用于关联端点（包括链端点和关联角色端点）上的一个约束，声明对应的实例通过一个实际的关联可见，而不是通过暂时链，如参数或局部变量。

见association，association end，association role。

3. become(变成)

(stereotype of Flow relationship)

其源和目标代表不同时间点的相同实例的构造型依赖关系，但源和目标有潜在的不同的

China-pub.com 下载

值、状态实例和角色。从A到B的一个变成依赖关系意味着在时间/空间上的一个不同的时刻实例A变成B，可能具有了新的值、状态实例和角色。变成的表示法是一个从源到目标的虚箭头，带有关键字《become》。

见become。

4. bind(绑定)

(keyword on Dependency symbol)

代表绑定关系的依赖关系上的关键字。它后面跟着由括弧括起逗号分隔的参量列表。

见binding，bound element，template。

5. call(调用)

(stereotype of Usage dependency)

源是一个操作而目标也是一个操作的构造型依赖关系。调用依赖关系声明源操作激发目标操作。调用依赖关系可以把源操作连接到其作用域内的任何目标操作，包括封闭类元和其他可见类元的操作，但不限制在这些操作上。

见call，usuage。

6. complete(完整)

(constraint on Generalization)

应用于一个泛化集合的约束，声明所有的子已经被声明（尽管有的可能被省略），且附加的子不应该在后面声明。

见generalization。

7. copy(复制)

(stereotype of Flow relationship)

源和目标是不同的实例，但有相同的值、状态实例和角色的构造型流关系。从 A到B的复制依赖关系意味着 B是A的一个精确复本。 A中的特征改变不必反映在 B中。复制表示法是一个从源到目标的箭头，带有关键字《copy》。

见access，copy。

China-pub.com
下载

8. create(创建)

(stereotype of BehavioralFeature)

表示指定的特征创建该特征所附属的类元的实例的构造型行为特征。

(stereotype of Event)

表示创建一个实例的构造型事件，该实例封装了事件类型所作用的状态机。创建只能作用于该状态机顶层的初始转换。实际上,这是唯一可以作用于初始转换的触发。

(stereotype of usage Dependency)

表示客户类元创建了提供者类元的实例的构造型依赖关系。

见creation，usuage。

9. derive(导出)

(stereotype of Abstraction dependency)

源和目标通常是，但不一定总是相同类型的构造型依赖关系。导出依赖关系说明源可以从目标计算得到。源可以为设计原因（如效率）而实现，尽管逻辑上它是冗余的。

见derivation，derived element。

10. destroy(销毁)

(stereotype of BehavioralFeature)

表明指定的特征销毁了该特征所附属的类元的实例的构造型行为特征。

(stereotype of Event)

表示封装了事件类型所作用的状态机的实例被销毁的构造型事件。

见destruction。

11. destroyed(被销毁的)

(constraint on ClassiferRole and AssociationRole)

表示角色实例在封闭的交互执行开始时存在而在执行结束之前被销毁。

见association role，classifier role，collaboration，destruction。

12. disjoint(互斥)

(constraint on Generalization)

China-pub.com 下载

作用于泛化集合的约束，说明对象不能是泛化集合中多个子的实例。这种情况只会出现在多重继承中。

见generation。

13. document(文档)

(stereotype of Component)

表示一个文档的构造型组件。

见component。

14. documentation(文档编制)

(tag on Element)

对所附属的元素进行的注释、描述或解释。

见comment，string。

15. enumeration(枚举)

(keyword on Classifier symbol)

枚举数据类型的关键字，其细节说明了由一个身份集合构成的域，那些身份是该数据类型的实例的可能取值。

见enumeration。

16. executable(可执行的)

(stereotype of Component)

表示一个可以在节点上运行的程序的构造型组件。

见component。

17. extend(扩展)

(keyword on Dependency symbol)

用例之间的扩展关系的依赖符号上的关键字。

China-pub.com
下载

见extend。

18. facade(虚包)

（stereotype of Package）

只包含对其他包所拥有的元素进行引用的构造型包。它被用来提供一个包的某些内容的公共视图。虚包不包含任何它自己的模型元素。

见package。

19. file(文件)

（stereotype of Component）

表示包含源代码或者数据的文档的构造型组件。

见component。

20. framework(框架)

（stereotype of Package）

包含模式的构造型包。

见package。

21. friend(友元)

（stereotype of Permission dependency）

构造型依赖关系，它的源是一个模型元素，如操作、类或包，而目标是一个不同的包模型元素，如类或包。友员关系授权源访问目标，忽视所声明的可见性。它扩展了源的可见性，使目标可以看到源的内部。

见access，friend，visibility。

22. global(全局)

（stereotype of AssociationEnd）

应用于关联端点（包括链端点和关联角色端点）的约束，说明与链的另一端点的对象相

China-pub.com 下载

比，所附属的对象由于在全局作用域内故而是可见的。

见assciation，association end，collaboration。

23. implementation(实现)

(stereotype of Generalization)

构造型泛化，表示客户继承了提供者的实现（属性、操作和方法），但没有把提供者的接口公共化，也不保证支持这些接口，因此违反可替代性。这是私有继承。

见generalization，private inhitance。

24. implementationClass(实现类)

(stereotype of Class)

构造型类，不是一个类型，它代表了某种编程语言的类的实现。一个对象可以是一个（最多一个）实现类的实例。相反，一个对象可以同时是多个普通类的实例，随时间得到或者丢失类。实现类的实例也可以是零个或者多个类型的实例。

见implementation class，type。

25. implicit(隐含)

(stereotype of Association)

关联的构造型，说明该关联没有实现（只是概念上）。

见association。

26. import(引入)

(stereotype of Permission dependency)

两个包之间的构造型依赖，表示目标包的公共内容被添加到源包的命名空间中。

见access，import。

27. include(包含)

(keyword on Dependency symbol)

China-pub.com
下载

用例之间的包含关系的依赖符号上的关键字。

见include。

28. incomplete(不完整)

(constraint on Generalization)

应用于泛化集合的约束，说明不是所有的子都已经被声明，后面还可以增加额外的子。

见generalization。

29. instanceOf(的实例)

(keyword on Dependency symbol)

客户是一个实例而提供者是一个类元的源关系。从 A到B的这种依赖意味着A是B的一个实例。它的表示法是一个带有关键字《instanceOf》的虚箭头。

见descriptor，instance，instance of。

30. instantiate(实例化)

(stereotype of Usage dependency)

类元之间的构造型依赖，这些类元表明客户的操作创建提供者的实例。

见instantiation，usuage。

31. invariant(不变量)

(stereotype of Constraint)

必须附在类元或关系集合上的构造型约束。它表明必须为类元或关系以及它们的实例维持约束的条件。

见invariant。

32. leaf(叶)

(keyword on Generalizable Element and BehavioralFeature)

不能有后代也不能被重载的元素，即不是多态的元素。

China-pub.com 下载

见leaf，polymorphic。

33. library(库)

(stereotype of Component)

表示静态或者动态库的构造型组件。

见component。

34. local(局部)

(stereotype of AssociationEnd)

关联端点、链端点或者关联角色端点的构造型，它声明所附属的对象在另一端对象的局部作用域中。

见association，association end，collaboration，transient link。

35. location(位置)

(tag on Classifier symbol)

支持类元的构件。

(keyword on Component instance symbol)

构件实例所在的节点实例。

见component，location，node。

36. metaclass(元类)

(stereotype of Classifier)

构造型类元，表明这个类是某个其他类的元类。

见metaclass。

37. new(新)

(constraint of ClassifierRole and AssociationRole)

表示角色的实例在封闭交互的执行过程中被创建，在执行结束后依然存在。

China-pub.com
下载

见association role，classifier role，collaboration，creation。

38. overlapping(重叠)

（constraint on Generalization）

应用于泛化集合的约束，说明一个对象可以是泛化集合中多个子的实例。这种情况只在多重继承或多重分类中出现。

见generalization。

39. parameter(参数)

（stereotype of AssociationEnd）

关联端点（包括链端点和关联角色端点）的构造型，说明所属的对象是对另一端对象操作的调用的一个参数。

见association role，classifier role，collaboration，parameter，transient link。

40. persistence(永久)

（tag on Classifier, Association, and Attribute）

表示一个实例值是否比生成它的过程存在的更久。值为 persistent或者transient。如果被用在属性上，则可以更好地确定是否应该在类元中保存属性值。

见persistent object。

41. postcondition(后置条件)

（stereotype of Constraint）

必须附属于一个操作上的构造型约束。它表示在调用该操作之后必须保持该条件。

见postcondition。

42. powertype(强类型)

（stereotype of Classifier）

构造型类元，表示该类元是一个元类，该元类的实例是别的类的子类。

China-pub.com 下载

(keyword on Dependency symbol)
客户是一个泛化集合而提供者是一个强类型的关系。提供者是客户的强类型。
见powertype。

43. precondition(前置条件)

(stereotype of Constraint)
必须附属于一个操作上的构造型约束，在调用该操作时该条件必须被保持。
见precondition。

44. process(过程)

(stereotype of Classifier)
构造型类元，一个代表重量过程的主动类。
见active class，process，thread。

45. refine(精化)

(stereotype on Abstraction dependency)
表示精化关系的依赖上的构造型。
见refinement。

46. requirement(需求)

(stereotype of Comment)
声明职责或义务的构造型注释。
见requirement，responsibility。

47. responsibility(职责)

(stereotype on Comment)
类元的协议或者义务。它表示为一个文本字符串。
见responsibility。

China-pub.com
下载

48. self(自身)

（stereotype of AssociationEnd）

关联端点（包括链端点和关联角色端点）的构造型，说明一个从对象到其自身的伪链，目的是为了在交互中调用作用在相同对象上的操作。它没有暗含实际的数据结构。

见association role，classifier role，collaboration，parameter，transient link。

49. semantics(语义)

（tag on Classifier）

对类元含义的规格说明。

（tag on Operation）

对操作含义的规格说明。

见semantics。

50. send(发送)

（stereotype of Usage dependency）

客户是一个操作或类元，提供者是一个信号的构造型依赖，说明客户发送信号到某个未说明的目标。

见send，signal。

51. stereotype(构造型)

（keyword on Classifier symbol）

用来定义构造型的关键字。其名称可能用作其他模型元素上的构造型名称。

见stereotype。

52. stub(桩)

（stereotype of Package）

构造型包，表示一个只提供另外一个包的公共部分的包。

注意该词汇也被UML用来描述桩转换。

见package。

China-pub.com 下载

53. system(系统)

(stereotype of Package)

包含系统模型集合的构造型包，它从不同的视点描述系统，不一定互斥 —— 在系统声明中的最高层构成物。它也包含了不同模型的模型元素之间的关系和约束。这些关系和约束没有增加模型的语义信息。相反它们描述了模型本身的关系，如需求跟踪和开发历史。一个系统可以通过一个附属子系统集合来实现，每个子系统由独立的系统包中的模型集合来描述。一个系统包只能包含在一个系统包里。

见package，model，system。

54. table(表)

(stereotype of Component)

表示一个数据库表的构造型组件。

见component。

55. thread(线程)

(stereotype of Classifier)

作为一个主动类的构造型类元，它表示一个轻量控制流。

注意在本书中该词汇具有更广的含义，可以表示任何独立并发的执行。

见active class，thread。

56. trace(跟踪)

(keyword on Abstraction dependency)

表示跟踪关系的依赖符号上的关键字。

见trace。

57. transient(暂时)

(constraint on ClassifierRole and AssociationRole)

声明角色的一个实例在封闭交互的执行过程中被创建，但在执行结束之前被销毁。

见association，classifier role，collaboration，creation，destruction，transient link。

China-pub.com
下载

58. type(类型)

(stereotype of Class)

用于实例（对象）域和应用于对象的操作的规格说明的构造型类。类型不可能包含任何方法，但是可以有属性和关联。

见implementation class，type。

59. use(使用)

(keyword on Dependency symbol)

表示使用关系的依赖符号上的关键字。

见usage。

60. utility(效用)

(stereotype of Classifier)

没有实例的构造型类元。它描述了一个由类作用域的非成员属性和操作构成的集合。

见utility。

61. xor(异或)

(constraint on Association)

作用在连接到同一个类上的关联集合上的约束，它声明任何被共享的类的对象只能有一个来自这些关联的链。它是异或（不是或）约束。

见association。

China-pub.com
下载

第四部分

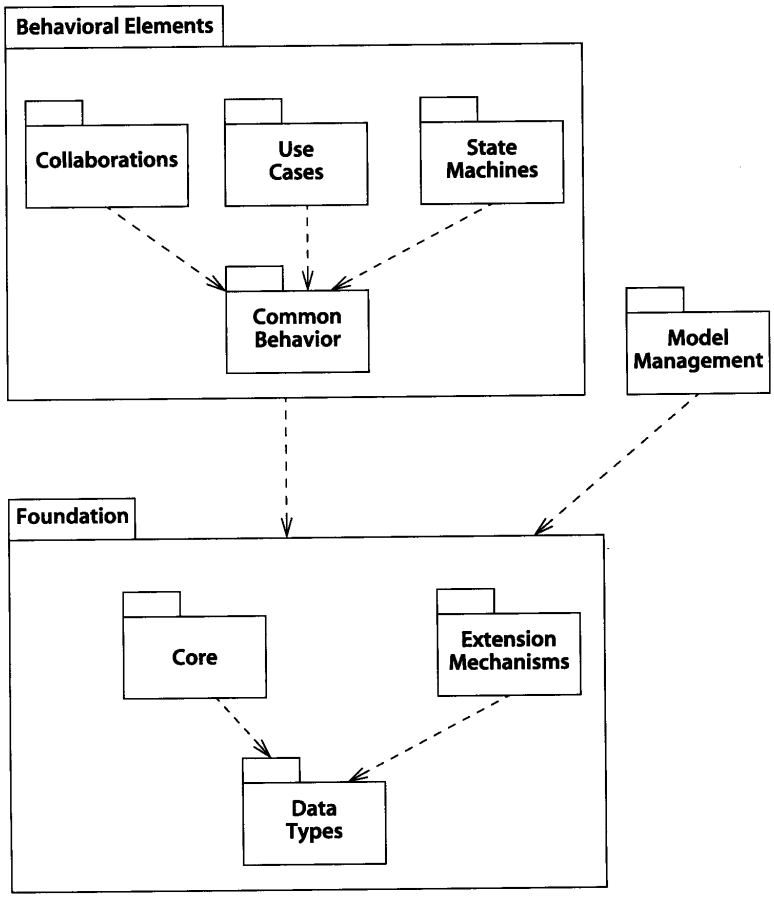
附　　录

China-pub.com
下载

附录A
UML 元模型

1. UML定义文档

UML 由对象管理小组[UML-98]出版的文档集定义。这些文档包含在本书所附的 CD中。



图A-1 UML 元模型的包结构

China-pub.com 下载

本章阐述了这些文档所描述的UML语义模型的结构。

UML使用元模型正式定义——即，UML中的构造函数的模型。元模型本身在UML中表达。这是元循环解释的一个例子——即用其自身来定义的一种语言。事物不是完全循环的。只有UML的一个小子集被用来定义元模型。原则上，定义的固定点可以从一个更基本的定义引导而来。实际上，不必如此费劲。

语义文档的每个部分包含一个显示部分元模型的类图、定义在各个部分的元模型类的文本描述以及它们的属性和关系、作用在元素上由自然语言和 OCL语言表达的约束、定义在各部分的UML构造的动态语义的文本描述。所以动态语义是非正式的，但是一个完全正式的描述是不可行而且也是不可读的。

表示法在一个独立的章里进行描述，该章引用了语义章并把符号映射到元模型类上。

2. 元模型结构

元模型分为三个基本的包（如图A-1）：

- 定义UML静态结构的基本包。
- 定义UML 动态结构的行为元素包。
- 定义UML 模型的组织结构的模型管理包。

3. 基本包

基本包包含四个子包。

核心

核心包描述了UML的主要静态构造，包括类元、它们的内容以及它们的关系。内容包括属性、操作、方法和参数。关系包括泛化、关联和依赖。也定义了几个抽象的元类，如可泛化元素、命名空间和模型元素。这个包也定义了模板和各种依赖子类，还有构件、节点和注释。

数据类型

数据类型包定义了用于元模型的数据类型类。

扩展机制

扩展机制包描述了约束、构造型和标记值机制。

China-pub.com
下载

4. 行为元素包

行为包给三个主要视图共享的行为结构提供了一个子包，也为每个主要视图提供了一个子包。

公共行为

公共行为包描述了信号、操作和动作。它也描述了对应于各种描述的实例类。

协作

协作包描述了协作、交互、消息、类元角色和关联。

用例

用例描述了参与者和用例。

状态机

状态机包描述了状态机结构，包括状态和各种伪状态、事件、信号、转换和监护条件。它也描述了活动模型的额外的结构，如动作状态、活动状态和对象流状态。

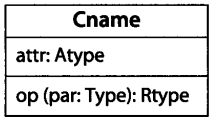
5. 模型管理包

模型管理包描述了包、模型和子系统。它也描述了所有关系和命名空间及包的可见性特性。它没有子包。

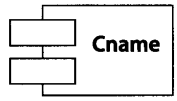
China-pub.com
下载

附录B
表示法小结

本附录包含了对表示法的一个精练的可视化总结。包括了主要的表示元素，但没有给出

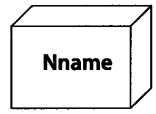


类

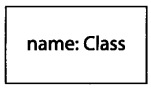


构件

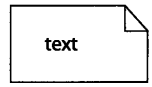
活动类



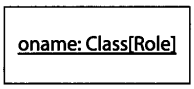
节点



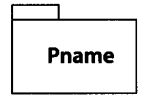
角色



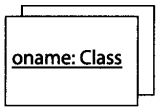
注释



对象



包

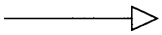


多对象

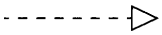
接口

协作

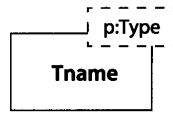
关联



泛化



实现



模板参数

模板

依赖关系

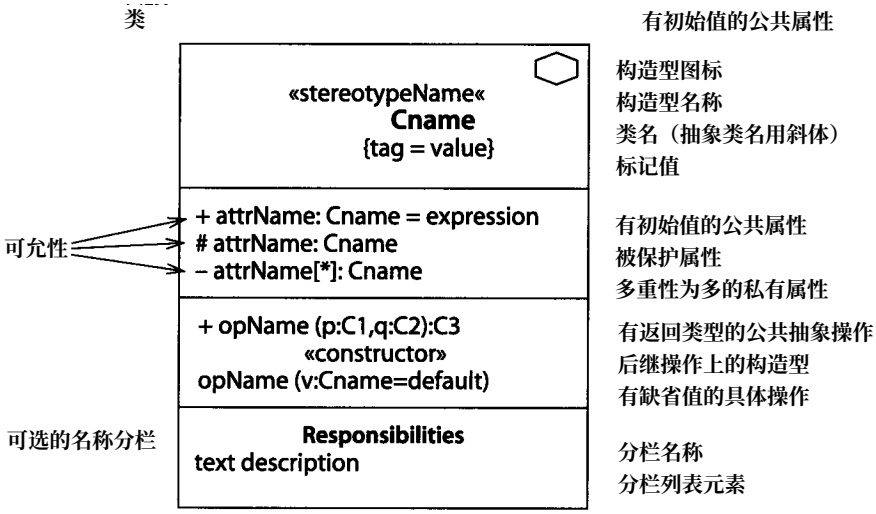
{ expression }

约束

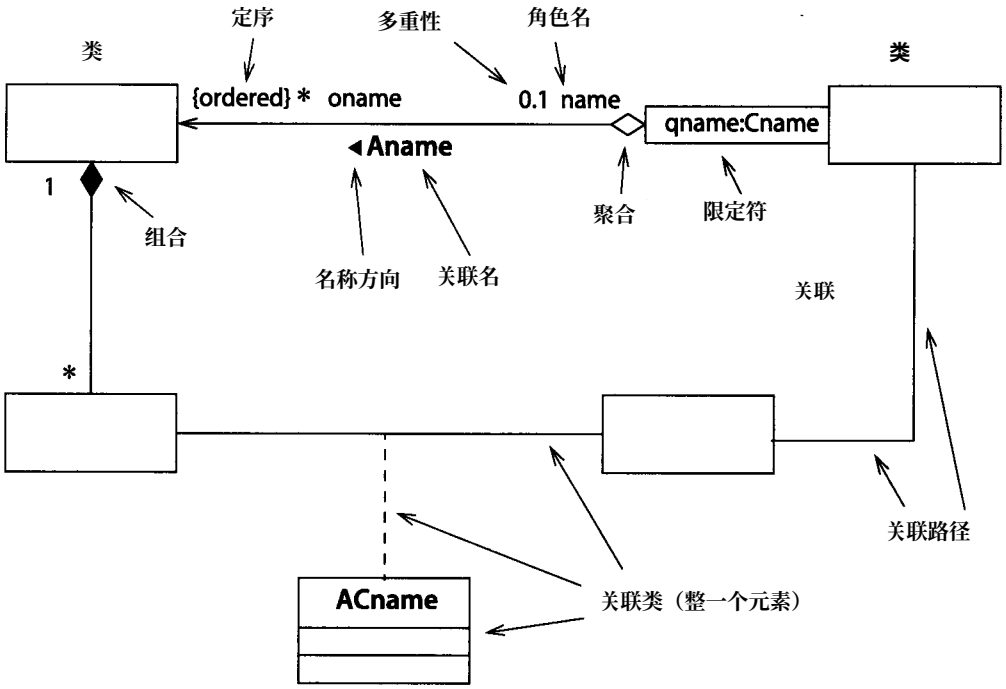
图B-1　类、构件、部署和协作图上的图标

China-pub.com
下载

所有的变体和选项。要得到完整的细节，可参见每个元素的解释。

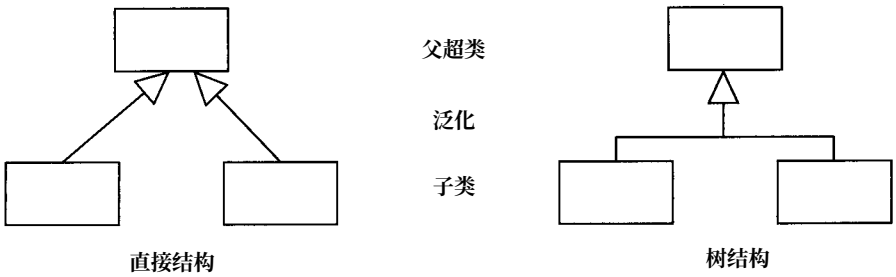


图B-2 类内容

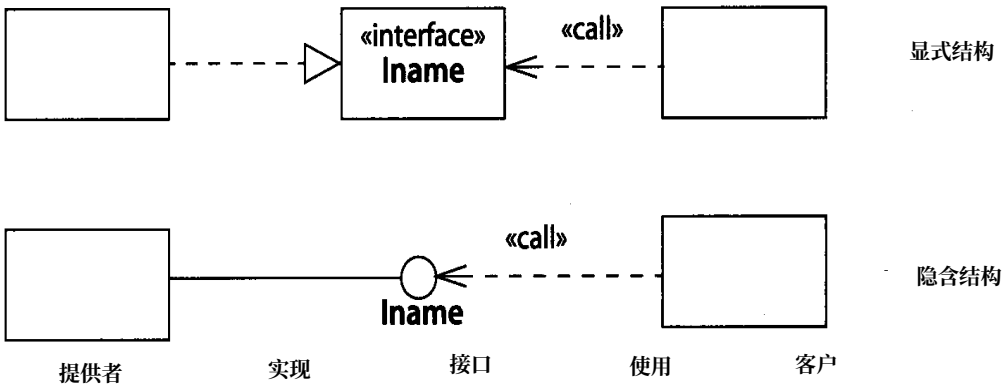


图B-3 类图中的关联符号

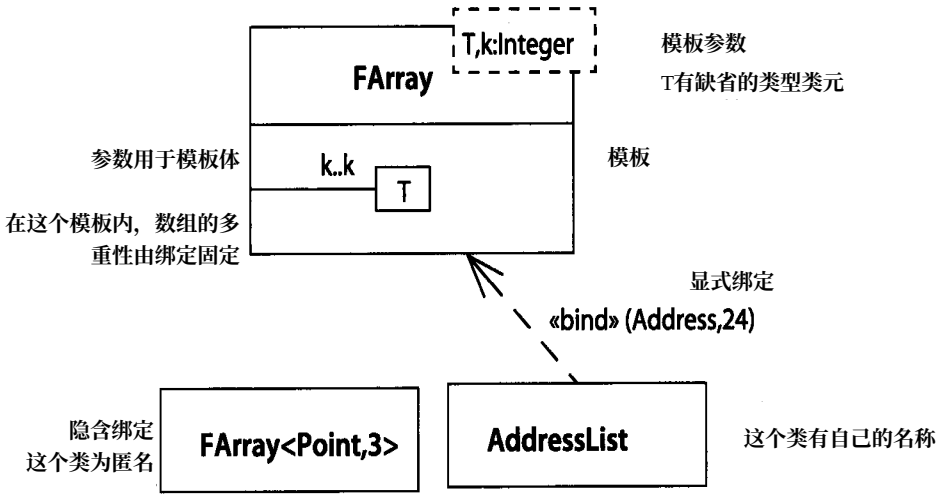
China-pub.com 下载



图B-4 泛化

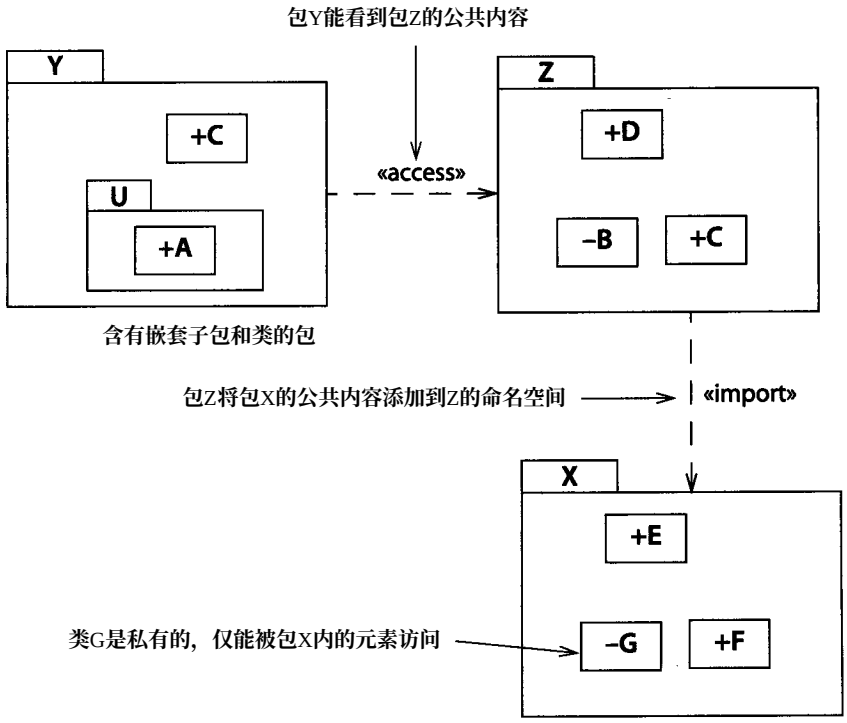


图B-5 接口的实现

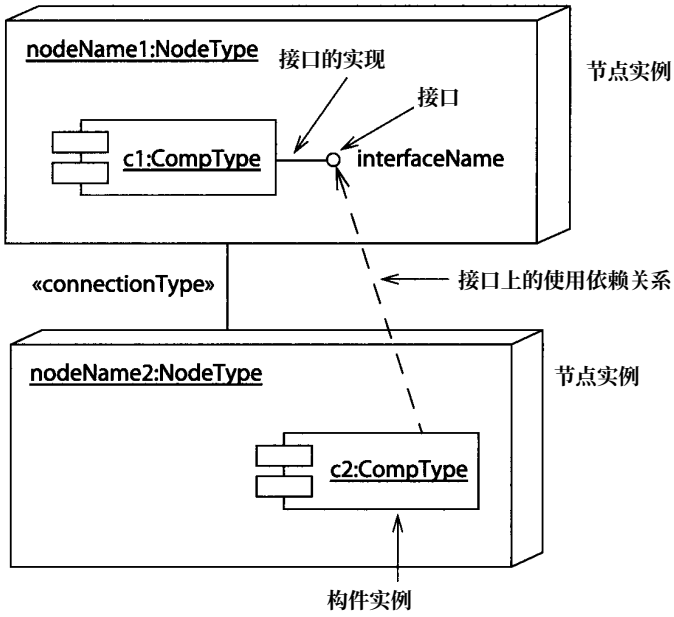


图B-6 模板

China-pub.com
下载

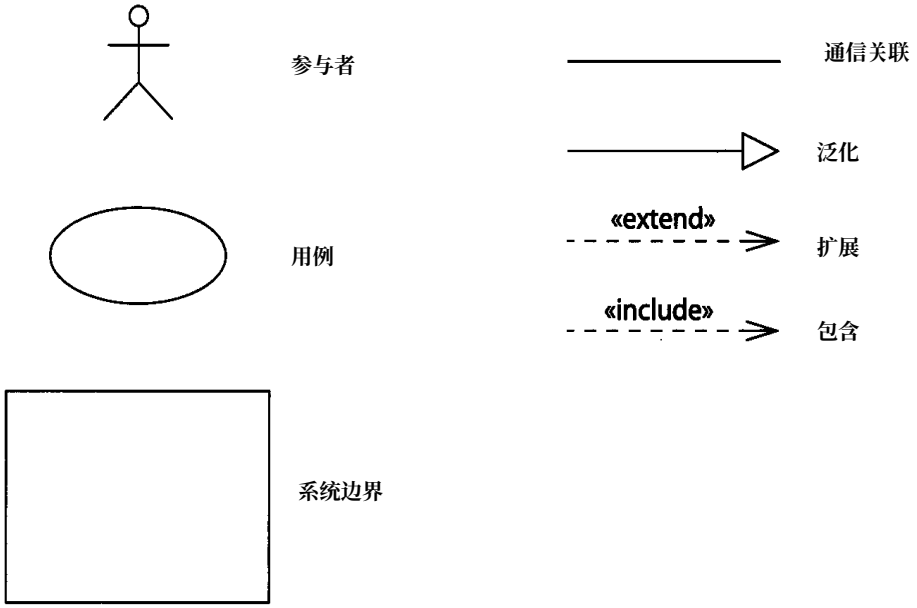


图B-7 包表示法

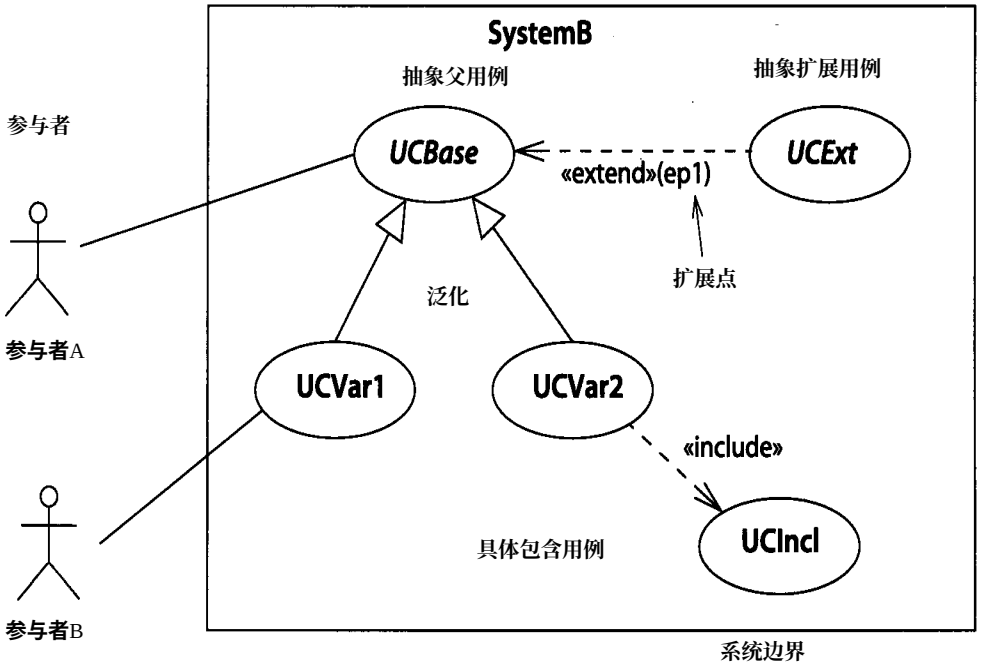


图B-8 构件和节点表示法

China-pub.com 下载

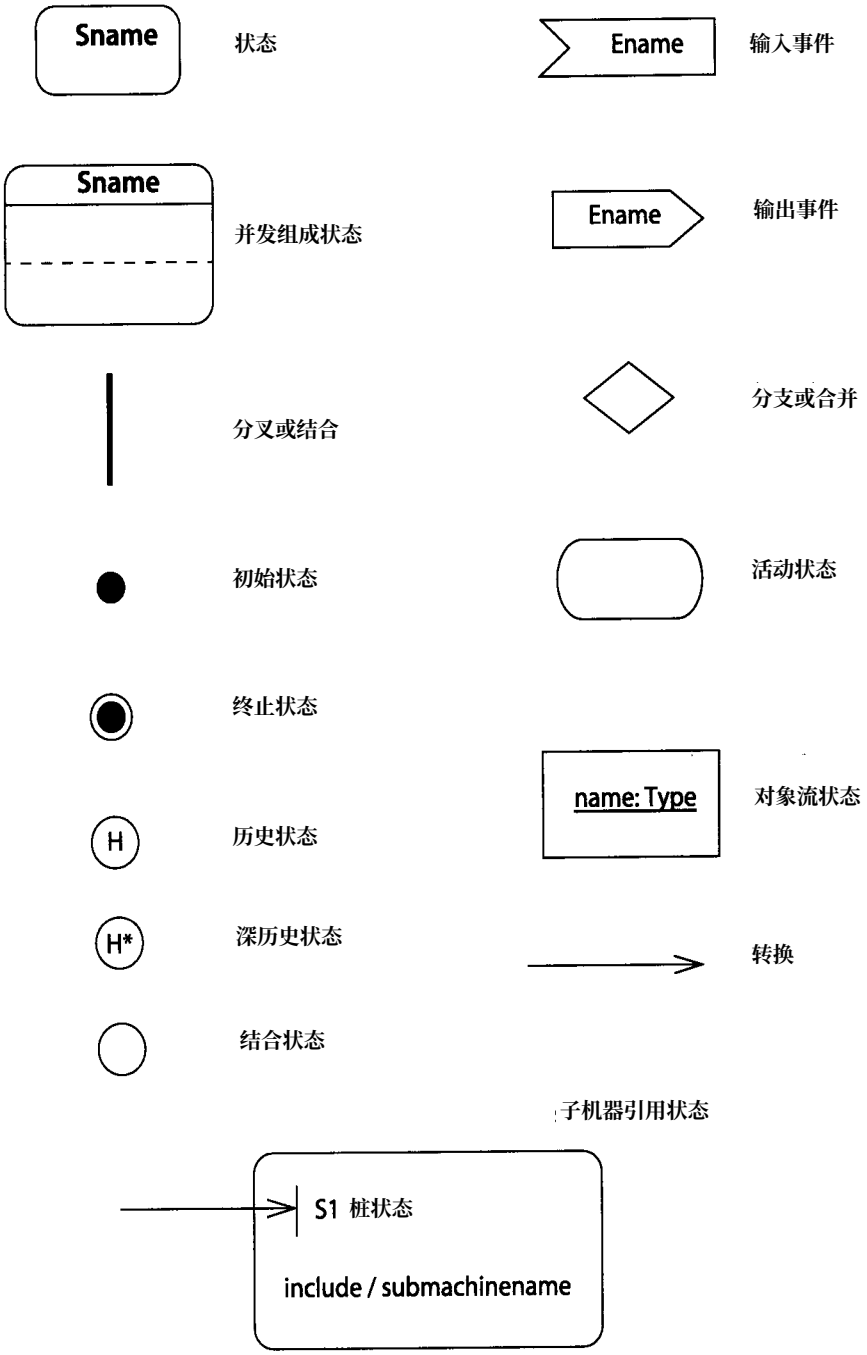


图B-9 用例图中的图标



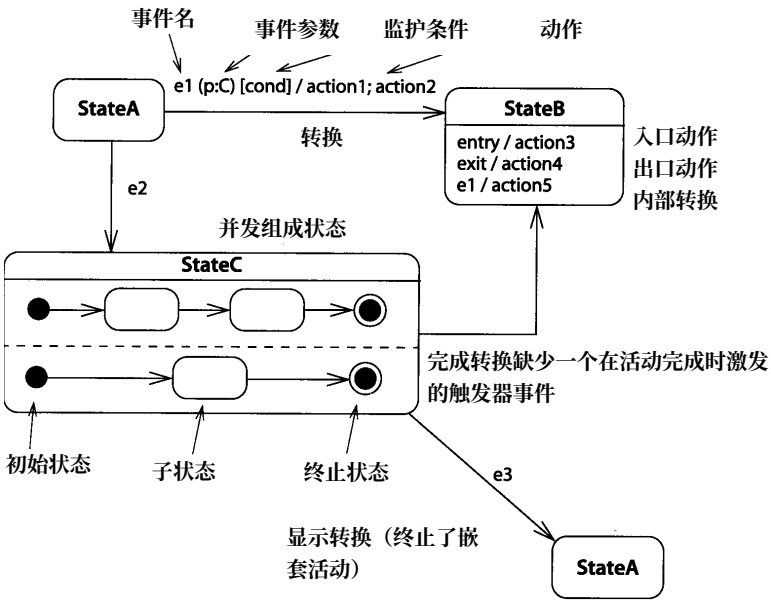
图B-10 用例图表示法

China-pub.com
下载

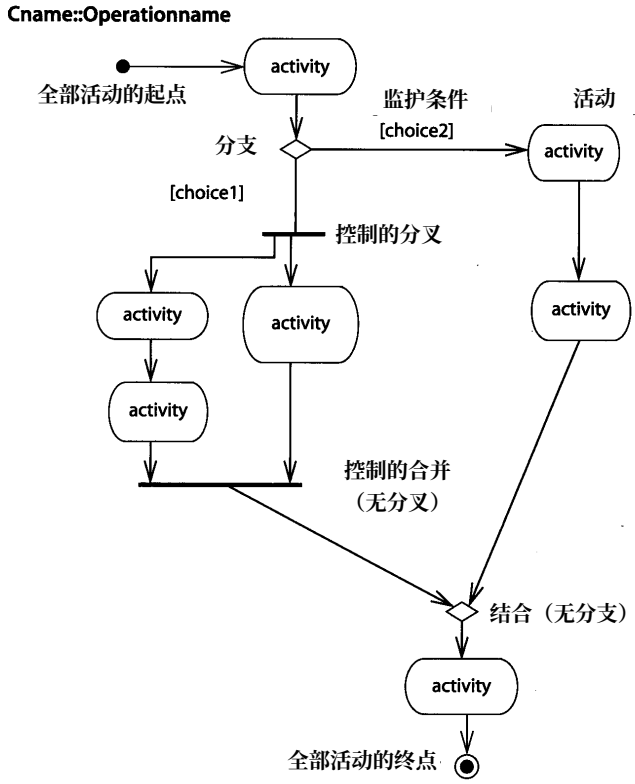


图B-11 状态图和活动图上的图标

China-pub.com 下载

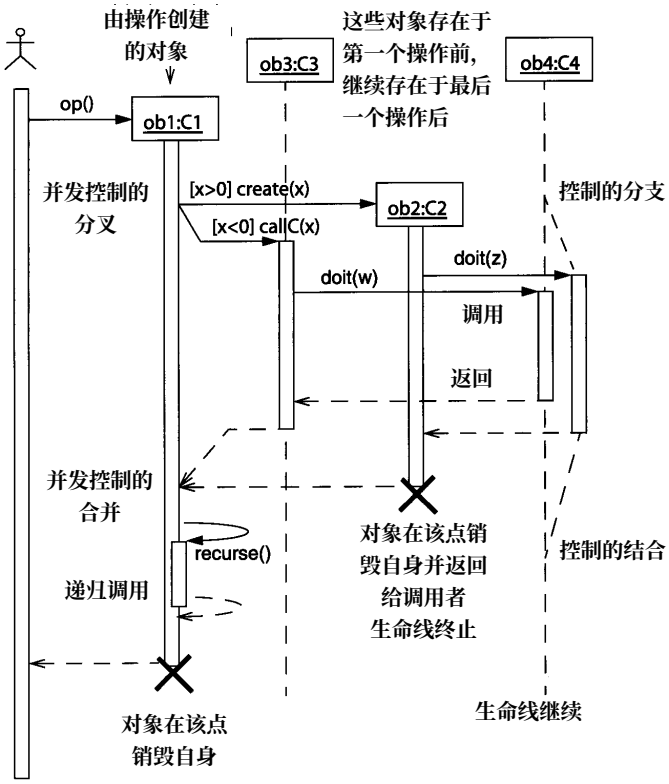


图B-12 状态图表示法

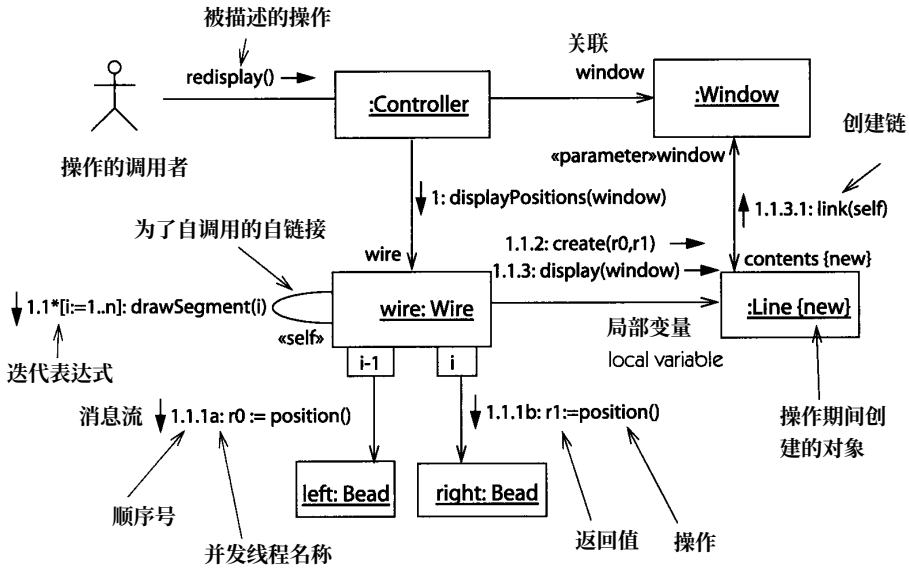


图B-13 活动图表示法

China-pub.com
下载

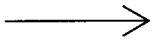


图B-14 顺序图表示法

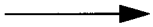


图B-15 协作图表示法

China-pub.com 下载


顺序消息

异步消息


调用

图B-16 消息表示法

China-pub.com
下载

附录C
过 程 扩 展

1. 剪裁UML

UML是为多用途而设计的。尽管 UML 是一种可以表示大量基本的建模特性的普遍语言，但是有时候需要剪裁语言以适应特定的领域。在自然语言中，黑话、俗语和专用词汇经常可以方便特定行业中的交流，但是外行人无法理解。 UML 可以以相似的方法剪裁，使用各种机制，如命名约定、风格规则、预定义类和到软件的隐含映射。尤其是， UML扩展可以用预定义构造型、约束和标记值来定义。

UML有很多表达性的结构，所以除非绝对必要，应该避免特定的扩展。通过定义，扩展服务于有限的区域，可能给外部造成误解和混淆。但是如果能充分着重目标区域，扩展的表达能力有时会超过一致性上的损失。经过一定的时间，有的扩展可能因足够有用而被加入到标准UML中，而有的可能被放弃。

本章描述了和UML文档一起打包的两个扩展。对于它们的使用不是强制的，甚至也没有推荐，但是它们展示了如何为开发过程定义一个扩展。一些相似的扩展已经被书面定义。

2. 软件开发过程扩展

这些扩展基于统一过程前身的 Objectory开发过程。它们用于软件开发过程的四个阶段：用例捕获、分析、设计和实现。

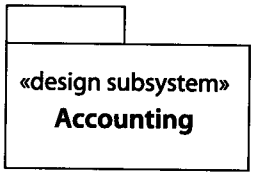
这些扩展包括包单元、类、关联上的构造型以及连接元素上的约束。没有新的标记。

组织构造型

表C-1显示了组织构造型 —— 即模型、包和子系统上的构造型。存在应用于各个开发阶段的各种构造型。在对象过程中，每个阶段有不同的模型，在不同模型的元素之间存在跟踪关系。从顶部到底部，术语系统、子系统和服务包描述了打包的层次。

组织构造型没有特别的图标。它们用折叠式图标表示，并在书名号中（《》）标有构造型的名称，如图C-1。

China-pub.com 下载



构造型名称
子系统名称

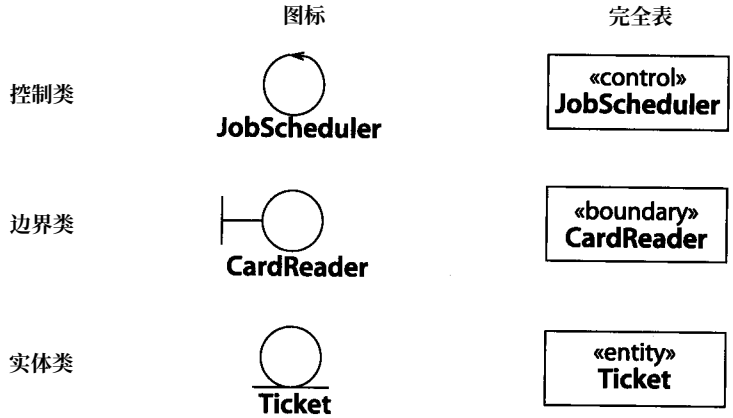
图C-1 软件开发过程的组织构造型表示法

表C-1 软件开发过程的组织构造型

基 类	过程阶段构造型			
	用例	分析	设计	实现
模型	用例模型	分析模型	设计模型	实现模型
包	用例系统 用例包			实现系统 实现子系统
子系统		分析系统 分析子系统 分析服务 包	设计系统 设计子系统 设计服务包	

类构造模型

类上定义了三个构造型：控制、边界、实体。如图 C-2。



图C-2 软件开发过程的构造型

控制类描述的是管理交互的对象，如事物管理者、设备控制者、操作系统监护者。对每个用例都有行为说明。控制类不会比它所支持类的寿命长，它用附有箭头的圆圈表示。

边界类描述的是调节系统和外部参与者间关系的对象，如有序入口表或灵敏元素。它用 T

China-pub.com
下载

形条的圆圈表示。边界对象与系统的寿命一样长。

实体类描述的是被动对象。它们没有对交互初始化。实体对象参加许多用例，比单个交互寿命长。它用下面标有横线的圆圈表示。

关联构造型

关联上有两个构造型：通信和预订。用带有构造型名称的关联路径表示。

通信关联连接参与者和用例并由此进行通信。这是参与者和用例间的唯一关联，所以关键词可省略。

预订关联连接客户类（预订者）和提供者类（出版者）。预订说明了出版者产生的事件集合。当其中一个事件发生时预订者被通知。

3. 业务建模扩展

这些扩展是为了给真实世界的业务组织建模和理解真实世界的情况，并非为实现软件。这些构造型不是业务建模必须的，但它们覆盖了一些共同情况。

这些扩展包括包单元、类、关联上的构造型，还包括连接元素上的约束。没有新标记。

组织构造型

表C-2表示了业务过程建模的组织构造型。

表C-2 业务建模的组织构造型

基　类	用例捕捉	对象模型
模型	用例模型	对象模型
包	用例系统 用例包	
		对象系统 组织单元 工作单元

用例模型是用例视图的模型。用例系统和用例包是组织其内容的两个层次。

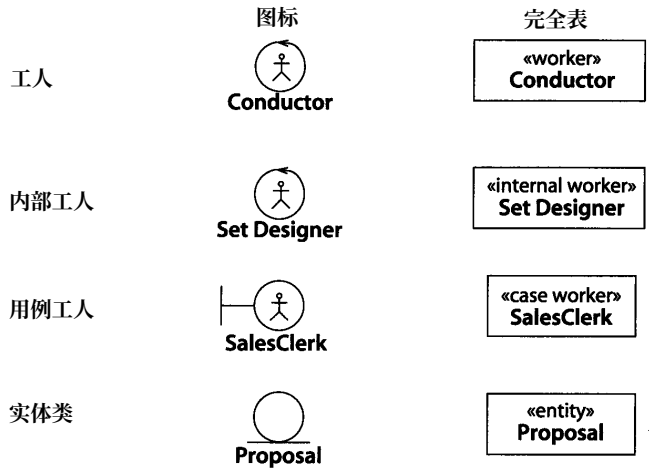
对象模型是业务系统内部结构的模型。对象系统是高层次的子系统，包括组织单元和低层的工作单元。组织单元对应实际业务的组织单元，工作单元则对应更小但有意义的小组。

业务建模组织单元没有特殊的图标，它们用带有构造型的折叠式符号表示。

类构造型

事务对象除了参与者外（定义在标准UML中），还有几个类上的构造型——工人、用例工人、内部工人、实体，如图C-3。

China-pub.com 下载



图C-3 事务模型的类构造型

工人代表在系统中活动的人。案例工人是直接和外部参与者交互的工人。内部工人是与系统中的工人和实体交互的工人。

实体类描述被动对象。它们不能引发交互。它们可以参与多个用例，通常比单个交互存在更久。在工作情况下，实体通常代表工作产品。

关联构造型

关联有两个构造型：通信和预订。和软件开发过程相同。用带有构造型关键字的关联路径表示。

通信关联把参与者连接到与之通信的用例上。这是参与者和用例之间的唯一关联，所以可以省略关键字。

预订关联把客户类（预订者）连接到提供者类（出版者）上。预订声明了出版者可能产生的事件集合。当其中某个事件发生时，预订者被通知。

China-pub.com
下载

参考文献

[**Blaha-98**] Michael Blaha, William Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Prentice Hall, Upper Saddle River, N.J., 1998.

[**Booch-91**] Grady Booch. *Object-Oriented Analysis and Design with Applications, 1st ed.* Benjamin/Cummings, Redwood City, Calif., 1991.

[**Booch-94**] Grady Booch. *Object-Oriented Analysis and Design with Applications, 2nd ed.* Benjamin/Cummings, Redwood City, Calif., 1994.

[**Booch-96a**] Grady Booch. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, Menlo Park, Calif., 1996.

[**Booch-96b**] Grady Booch. *Best of Booch: Designing Strategies for Object Technology*. SIGS Books, New York, N.Y., 1996.

[**Booch-99**] Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Mass., 1999.

[**Buschmann-96**] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, Chichester, U.K., 1996.

[**Coad-91**] Peter Coad, Edward Yourdon. *Object-Oriented Analysis, 2nd ed.* Yourdon Press, Englewood Cliffs, N.J., 1991.

[**Coleman-94**] Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes, Paul Jeremaes. *Object-Oriented Development: The Fusion Method*. Prentice Hall, Englewood Cliffs, N.J., 1994.

[**Cox-86**] Brad J. Cox. *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, Reading, Mass., 1986.

[**Embley-92**] Brian W. Embley, Barry D. Kurtz, Scott N. Woodfield. *Object-Oriented Systems Analysis: A Model-Driven Approach*. Yourdon Press, Englewood Cliffs, N.J., 1992.

[**Gamma-95**] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass., 1995.

[**Goldberg-83**] Adele Goldberg, David Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, Reading, Mass., 1983.

[**Harel-98**] David Harel, Michal Politi. *Modeling Reactive Systems With Statecharts: The STATEMATE Approach*. McGraw-Hill, New York, N.Y., 1998.

[**Jacobson-92**] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Övergaard. *Object-*

China-pub.com 下载

Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Wokingham, England, 1992.

[Jacobson-95] Ivar Jacobson, Maria Ericsson, Agneta Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology*. Addison-Wesley, Wokingham, England, 1995.

[Jacobson-97] Ivar Jacobson, Martin Griss, Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley, Harlow, England, 1997.

[Jacobson-99] Ivar Jacobson, Grady Booch, James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, Reading, Mass., 1999.

[Martin-92] James Martin, James Odell. *Object-Oriented Analysis and Design*. Prentice Hall, Englewood Cliffs, N.J., 1992.

[Meyer-88] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, New York, N.Y., 1988.

[Rumbaugh-91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, N.J., 1991.

[Rumbaugh-96] James Rumbaugh. *OMT Insights: Perspectives on Modeling from the Journal of Object-Oriented Technology*. SIGS Books, New York, N.Y., 1996.

[Rumbaugh-99] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Mass., 1999.

[Selic-94] Bran Selic, Garth Gullekson, Paul T. Ward. *Real-Time Object-Oriented Modeling*. Wiley, New York, N.Y., 1994.

[Shlaer-88] Sally Shlaer, Stephen J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Yourdon Press, Englewood Cliffs, N.J., 1988.

[Shlaer-92] Sally Shlaer, Stephen J. Mellor. *Object Lifecycles: Modeling the World in States*. Yourdon Press, Englewood Cliffs, N.J., 1992.

[UML-98] *Unified Modeling Language Specification*. Object Management Group, Framingham, Mass., 1998. Internet: www.omg.org.

[Ward-85] Paul Ward, Stephen J. Mellor. *Structured Development for Real-Time Systems: Introduction and Tools*. Yourdon Press, Englewood Cliffs, N.J., 1985.

[Warmer-99] Jos B. Warmer, Anneke G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, Reading, Mass., 1999.

[Wirfs-Brock-90] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener. *Designing Object-Oriented Software*. Prentice Hall, Englewood Cliffs, N.J., 1990.

[Yourdon-79] Edward Yourdon, Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press, Englewood Cliffs, N.J., 1979.

China-pub.com
下载

索 引

AddOnly（关键字） 144
Booch 方法，Booch method 5,7
C++ 4,131,169,178,185,192,194,196,213,243,264~265,291~292,305,330,356,383~385,394
  数据类型语法，data type syntax 252
CLOS 4,160,223,305
Cobol 4
CRC 5
Fortran 4
Frozen(关键字) 144
Fusion（方法名） 5
Guarded(关键字) 289
In（关键字） 298
Inout（关键字） 298
Local（关键字） 377
Mealy机，Mealy machine 351
MOF 参见“元对象机制”
Moore机，Moore machine 351
n-元关联，n-ary association 273
Objective C 4
Objectory 4,7,423
OCL 287
OMG 参见“对象管理小组”
OMT 5,7
Out（关键字） 298
overlapping（约束） 225
Petri网环，Petri net token 365
Rational 软件公司 5
Simula-67 4
Smalltalk 4,132,192,196,213,264~265,292,305,383

B

绑定，bind 135
绑定，binding 135
绑定元素，bound element 136
包，package 294
  依赖关系，dependency 73,196
包含，include 232
包容器，container 187
背景信息，background information 132
被动对象，passive object 299
编译时间，compile time 163
变成，become 133
标记，tag 368
  预定义标记，predefined 395
标记值，tagged value 368
标签，label 252
表，table
  动作表，action 56
  关系表，relationship 35,323
    视图间关系表，across view 31
  开发过程构造型表，development process stereotype 424
  类元表，classifier 33
  流表，flow 68

China-pub.com 下载

事件表，event 52
视图和图，view and diagram 20
业务建模构造型表，business modeling stereotype 425
依赖关系表，dependency 43,323
用例关系表，use case relationship 49,217
转换表，transition 54
状态表，state 58
表达式，expression 213
表的过滤，filtering of lists 257
表示法，notation
表示法考虑，consideration 81
表示法小结，summary 414
规范表示法，canonical 82
表示元素，presentation element 308
并发，concurrency
并发子状态，concurrent substate 164
动态并发性，dynamic 106
特性，property 289,291
消息发送，message send 260~261
转换规则，rules on transition 100
并发，concurrent
关键字，keyword 289
线程消息语法，thread message syntax 263
子状态，substate 183
独立子状态，independent of others 348
规则，rules for 165
执行语义，execution semantics 350
组成状态，composite state 60
不变量，invariant 248
不定问题，halting problem 368
不完全，incomplete
模型，model 230
约束，constraint 225
不一致模型，inconsistent model 83
布尔型，boolean 136
表达式，expression 136
部分，组成，part,composite 178
部署，deployment 198
视图，view 70,199
图，diagram 198

C

参量，argument 118
参数，parameter 297
表，list 298
关键字，keyword 377
参数化元素，parameterized element 参见“模板”
参数方向，direction of parameter 298
参与，participate 299
参与者，actor 112
仓库，repository 324
操作，operation 288
协作实现，realization by collaboration 157
层，layer 252
查询，query 289,317
属性特征，attribute property 291
查询表，lookup table 313,315
产品，product 310
超级链接，hyperlink 229
超类，superclass 362
超类型，supertype 363
成员，member 259
程序设计语言考虑，programming language consideration 82
冲突，conflict 183,234
重用，reuse 325
重载操作，overriding operation 305
抽象，abstract 87,224
操作，operation 90

China-pub.com
下载

类，class　90
　表示法，notation　151
抽象，abstraction　91
抽象超类规则，abstract superclasses rule　88
出口动作，exit action　212
　执行语义，execution semantics　348
出生事件，birth event　330
初始的，initial
　值，value　237
　　求值，evaluation　192
　状态，state　235
　　执行语义，execution semantics　349
初始化，initialization　237
初始阶段，inception phase　232
触发器，trigger　378,381
　事件，event　55
传递延迟，transmission delay　261
创建，create
　动作，action　96
创建，creation　192
　事件，event　235,329~330

D

代码，code　231
　代码生成，generation　82
　构件中的代码，in component　169~171
单的，single
　分类，classification　338
　继承，inheritance　338
单实例类，singleton　339
当前事件，current event　193
　执行语义，execution semantics　348
导出，derivation　199
导出元素，derived element　199
导航，navigation　277
　表达式，expression　121~122,264,276,287~288,313,325~326
　使用链，using link　254
　效率，efficiency　278
导航性，navigability　276
到时消息，time-out message　262
的实例，instance of　239
典型精化的优化，optimization as refinement　321
调用，call　140
　动作，action　95
　事件，event　141
　双信号调用，as two signals　336
迭代，iteration
　表达式，expression　248
　消息发送，message send　262
迭代开发，iterative development　249
　过程，process　341
顶点，vertex　393
定序，ordering　292
　表的定序，of lists　257
动态，dynamic
　并发，concurrency　207
　分类，classification　207
　视图，view　208
动作，action　94
　表，table of　56
　表达式，expression　97
　序列，sequence　97
　隐含，implicit　54
　语言，language　96~97
　转换，on transition　379
　状态，state　98
断言，assertion　248,305
对象，object　281
　扮演多重角色，playing multiple roles　154

China-pub.com 下载

扮演角色，playing role 152
创建对象，creation 240
集合，set 95
集表达式，set expression 287
流，flow 284
状态，state 284
生命线，lifeline 287
图，diagram 283
对象管理小组，object management group 5,411,428
对象图，object diagram 46
对象移动，migration of objects 198
对象约束语言，Object Constraint Language 参见“OCL”
多，many 258
多对象，multiobject 269
多态，polymorphic 304
属性特性，attribute property 291
多态特性，polymorphism property 289,320
多重，multiple
分类，classification 269
继承，inheritance 269
多重性，multiplicity 270
n-元关联多重性，n-ary association 273
关联多重性，association 271
类多重性，class 272
属性多重性，attribute 272
限定关联多重性，qualified association 313~314

E

二元关联，binary association 134

F

发送，send 329
表示法，notation 189
动作，action 96
依赖关系，dependency 332
发送者，sender 332
返回，return
动作，action 96
关键字，keyword 298
类型，type 290
消息，message 262
泛化，generalizaion 224
的维数，dimension of 205~206
判别式，discriminator 205
用例，use case 390
与实现的比较，compared to realization 42
方法，method 265
组合/合并，combination 160,305
访问，access 92
非活动的，inactive 232
非良性结构模型，ill-formed model 230
分布单元，distribution unit 207
分叉，fork 221
分栏，compartment 162
附加，additional 147
名称，named 257,325
隐藏分栏，suppressing 147
分类，classification
单和多重，single and multiple 41
静态和动态，static and dynamic 41
分配，allocation
分配职责，responsibility for 178
内存分配，of memory 201~202
分析，analysis 117
时间，time 117
分支，branch 138
顺序图表示法，sequence diagram notation 261
状态图表示法，statechart notation 188,379~380
风险，risk

China-pub.com
下载

最小化，minimizing　249
父，parent　299
复杂转换，complex transition　164
执行语义，execution semantics　350
复制，copy　191
赋值，assignment　95

G

改变事件，change event　143
概要，summarization　362
根，root　224
跟踪/踪迹，trace　376
公共的，public　312
功能视图，functional view　223
工作进展过程，work in progress　83
构架，architecture　117
构件，component　169
图，diagram　173
构造函数，constructor　186
组成构造函数，for composite　178
构造阶段，construction phase　186
构造型，stereotype　353
列表表示法，notation in list　256
图标，icon　148
预定义，predefined　395
关键字，keyword　251
预定义关键字，predefined　395
注解中的关键字，in note　280
关联，association　119
n元，n-ary　273
导航效率，navigation efficiency　278
端点，end　124
泛化，generalization　126
关键字，keyword　377
角色，role　127
类，class　157
限定，qualified　312
状态类，to class in state　149
关系，relationship　323
表，table of　35,323
规范表示法，canonical notation　142
规格说明，specification　340
操作说明，of operation　291
过程，process　309
扩展，extension　423
过程表达式，procedure expression　309
过程的，procedural
控制，control　335
消息，message　262

H

合并，merge　259
表示法，notation　188
合作对象社会，society of cooperating objects　64
后代，descendant　201
后置条件，postcondition　305
互斥，disjoint
泛化的互斥约束，constraint on generalization　225
互斥子状态，substate　207
环，token　365
环境，environment　80
活动，activity　104
表达式，expression　104
视图，view　61,111
图，diagram　104
图，graph　105
状态，state　110
活动的/主动的，active　100
对象，object　102

China-pub.com 下载

类，class 101
活动状态配置，active state configuration 103

J

基本包，foundation package 412
基数，cardinality 142
基用例，base use case 213~214,232
激发，fire 220
激活，activation 98
继承，inheritance 234
多态操作，polymorphic operation 305
私有，private 308
间接实例，indirect instance 234
监护条件，guard condition 227
简单，simple
转换，transition 338
状态，state 338
简单类型，primitive type 308
建模 ，modeling
工具，tool 82
时间，time 268
综述，overview 10
建模工具，tool for modeling 82
建模阶段，stages of modeling 340
交叉线表示法，crossing lines notation 300~301
交互，interaction 241
视图，view 242
图，diagram 242
脚本，scenario 327
接口，interface 243
继承，inheritance 245
说明符，specifier 246
接收，reception 319
信号接收，of signal 336
接收者，receiver 319
节点，node 279
结构，structural
视图，view 356
特征，feature 356
结构化分析，Structured Analysis 4
结构化设计方法，Structured Design 4
结合，join 249
结合状态，junction state 250
精化，refinement 321
静态，static
分类，classification 352
视图，view 32,352
具体，concrete 182
具体化，reification 322
关联的具体化，of association 124
聚合，aggregation 113
组合，composition 177
聚集，aggregate 113
绝对时间，absolute time 375
角色，role 325
角色名，rolename 325

K

开发，development
方法，method
传统开发方法，traditional 4
面向对象开发方法，object-oriented 4
过程，process 202
构造型，表，stereotype，table of 424
过程扩展，extension 423
可变性，changeability 144
可导航的，navigable 277
可泛化元素，generalizable element 223
可见性，visibility 393
规则，rule 75,92

China-pub.com
下载

可实例化的，instantiable　239
客户，client　153
空值，null value　83
控制，control
　流，flow　187
　图标，icon　188
控制期，focus of control　221
快照，snapshot　339
框架，framework　222,399
扩展，extend　213
　条件，condition　215
扩展点，extension point　217
扩展机制，extensibility mechanism　76

L

类，class　145
　和类型，and type　382
　名，name　150
　图，diagram　148
　由构件支持的类，supported by component　170
　状态类，in state　149
类级作用域，class-level scope　328
类型，type　382
　表达式，expression　384
　和实现类，and implementation class　382
类元，classifier　33,151
　表，table of　33
　角色，role　152
历史状态，history state　228
　执行语义，execution semantics　350
链，link　254
　创建，creation　240
　端点，end　255
　暂时链，transient　377
良性构成，well formed　394
列表，list
　分栏，compartment　162
　元素，element　255
列表中的组特性，group property in list　257
菱形符号，diamond symbol
　n元关联，n-ary association　273
　分支，branch　139
　聚合，aggregation　115
　组合，composition　179
流，flow　220
　表，table of　68
流逝时间，elapsed time　375
路径，path　300
路径名，pathname　302
　表示法，notation　151
轮廓，profile　79

M

枚举，enumeration　209
面向对象历史，object-oriented，history of　4
描述符，descriptor　201
　完全描述符，full　223
名称，name　275
　限定符名，as qualifier　314~315
命名空间，namespace　276
模板，template　370
　实例化模板，instantiated　135
模块，module　268
模式，pattern　302
　参数化协作，parameterized collaboration　154
模型，model　266
　不一致，inconsistent　81,83
　层次，levels of　12
　定义，definition　10
　管理，management

China-pub.com 下载

包，package 413
视图，view 73,268
含义，meaning 15
目的，purpose 10
内容，content 13
元素，element 267
目标，target
对象集，object set 260~261
状态，state 370
作用域，scope 370

N

内部转换，internal transition 247
语法，syntax 380
执行语义，execution semantics 249
内存分配，memory allocation 201~202
组成内存分配，by composite 178
内涵，intent 241
逆向工程，reverse engineering 82

P

判别式，discriminator 205
配置控制，configuration control 296
瀑布开发过程，waterfall development process 340~341

Q

前置条件，precondition 307
嵌入文档，embedded document 356
强类型，powertype 306
请求，request 324
全局，global
变量，variable 393
关键字，keyword 377
缺省值，default value 195

R

入口动作，entry action 208
执行语义，execution semantics 348
软件开发过程扩展，software development process extensions 423

S

散列表，hash table 315
设计，design 201
模式，pattern 302
时间，time 201
身份，identity 230
构件身份，of component 169~170
生产者-顾客，producer-consumer 365
生成器状态机，generator state machine 352
生命线，lifeline 253
省略号，ellipsis 256
时标，timing mark 375
时间，time 374
事件，event 374
时钟，clock 374~375
实参，actual parameter 参见“参量”
实例，instance 237
直接，direct 205
实例化，instantiate 240
实例化，instantiation 240
模型的实例化，of model 45
实例级作用域，instance-level scope 328
实时结构化设计，Real_Time Structured Design 4
实现，implementation 230
继承，inheritance 231
类，class 231
视图，view 70,231
实现，realization 317
操作实现，of operation 154

China-pub.com
下载

接口类实现，of class by interface　243
用例实现，of use case　154,318~319,388
与泛化的比较，compared to generalization　42
使用，usage　385
示出，export　212
事件，event　210
表，table of　52
处理，handling　346~347
视图，view　393
表，table of　20
间的关系，connections among　31
小结，summary　19
受保护的，protected　311
属性，attribute　129
作为组成部分，as composite part　178~180
树表示法，tree notation
泛化树表示法，for generalization　226
聚合树表示法，aggregation　115
组合树表示法，composition　179
数据，data
标记表示法，token notation　264
类型，type　194
值，value　194
书名号，guillemet　228
双向性，bidirectionality　38
顺序，sequence
号，number　335
图，diagram　22,65
和协作图，and collaboration diagram　68
顺序，sequential
关键字，keyword　289
消息，message　262
顺序图，sequence diagram　333
私有的，private　308
继承，inheritance　308
死锁，deadlock　368
所有关系，ownership　295
所有者作用域，owner scope　294
索引值，index value　312

T

特化，specialization　340
的维数，dimension of　205~206
特性，property　310
类特性表示法，notation for on class　150~151
列表，list　310
列表中的特性字符串，string in list　256
特征，feature　219
特征标记，signature　338
匹配，matching　288
提供者，supplier　363
替代原理，substitutability principle　360
条件线程，conditional thread　183
通信关联，communication association　161
同步，synch
条，bar　168
状态，state　365
同步，synchronization　164
条，bar　168
同步，synchronous
动作，action　368
控制，control　333~337
消息，message　262
统一，定义，unified, definition of　6
统一建模语言，UML
标准化，standardization　5
表示法小结，notation summary　414
裁制，tailoring　79
视图，view　19
表，table of　20

China-pub.com 下载

定义，definition 3
概念域，concept area 7
规格说明文档，specification document 411
过程扩展，process extension 423
环境，environment 80
历史，history 4
目标，goal 7
元模型，metamodel 411
投影，projection 310
图，diagram 204
图形标记，graphic marker 227

W

外部转换，external transition 54
外延，extent 218
完成转换，completion transition 163
完全描述符，full descriptor 223
完整，complete
约束，constraint 225
伪属性，pseudoattribute 311
伪状态，pseudostate 311
委派，delegation 196
未指定值，unspecified value 384
位置，location 257
文件构造型，file stereotype 399
无触发器转换，triggerless transition 382
无分叉，unfork 参见“结合”
无分支，unbranch 参见“合并”
无解释的，uninterpreted 384
动作，action 96
无用单元回收，garbage collection 178
物理视图，physical view 70

X

系统，system 368
构造型，stereotype 405
细化阶段，elaboration phase 208
线程，thread 374
限定符，qualifier 312
限定关联，qualified association 312
导航效率，navigational efficiency 278~279
消息，message 260
销毁，destroy 202
动作，action 96
销毁，destruction 202
效用，utility 392
协作，collaboration 153
角色，role 159
图，diagram 159
和顺序图，and sequence diagram 68
用例实现，realization of use case 48
信号，signal 336
表示法，notation 189
声明，declaration 52
事件，event 52,338
信号广播，broadcast of signal 336
行为，behavior 134
顺序，sequence 388
规格说明，specification 390
行为的，behavioral
视图，view 134
特征，feature 134
元素包，elements package 413
形参，formal argument 参见“参数”
虚包，facade 399
需求，requirement 324
跟踪，tracking 376
许可，permission 303
循环，recurrence 95

China-pub.com
下载

Y

延迟的，deferred
　操作，operation　304
　事件，event　195
业务建模，business modeling
　构造型，表，stereotype,table of　425
　扩展，extension　425
叶，leaf　253
　操作，operation　304
依赖，dependency　196
　包间的依赖关系，among package　73
　关系表，table of　43,323~324
异步，asynchronous
　动作，action　128
　控制，control　334
　事件，event　346
　消息，message　262
异常，exception　211
异或，xor　121
引入，import　231
引用，reference　320
　表示法，notation for　151
永久对象，persistent object　304
泳道，swimlane　363
泳道式业务组织，business organization as swimlane　363~364
用例，use case　386
　创建，creation　240
　泛化，generalization　390
　关系，relationship　49
　　表，table of　49,217
　模型，model　392
　实例，instance　392
　视图，view　392
　图，diagram　390
友元，friend　222
有效模型，effective model　372
有效系统实例，valid system instance　46
语境，context　187
语言，language
　类型，type　252
　面向对象语言，object-oriented　4
语义，semantic
　变更点，variation point　329
　分层，level　34~35
语义，semantics　329
元对象，metaobject　265
元对象机制，Meta-Object Facility　265
元关系，metarelationship　265
元类，metaclass　264
元模型，metamodel　265
　UML，411
元素表，element list　255
元-元模型，meta-metamodel　265
元组，tuple　382
原子，atomic　129
源，source
　状态，state　339
　作用域，scope　339
约束，constraint　184
　语言，language　45,184~186,213,287
　预定义约束，predefined　395
运行到完成，run to completion　327
　当前事件，current event　193
运行时间，run time　327

Z

暂时，transient
　对象，object　377

China-pub.com 下载

链，link 377
增量式开发，incremental development 234
正交子状态 ，orthogonal substate 293
执行语义，execution semantics 346
直接，direct
类，class 205
实例，instance 205
值，value 参见“数据值”
值0，不同于值的缺少， zero, different from absence of value 130
职责，responsibility 324
制品，artifact 118
中止消息，balking message 262
终止动作，terminate action 96
终止状态，final state 219
执行语义，execution semantics 349
主导类，dominant class 170~172
注解，note 280
注释，comment 261
转换，transition 377
表，table of 54
并发规则，concurrency rule 100
复杂转换，complex 164
激发规则，firing rule 346
阶段，phase 381
内部转换，internal 343
片断，segment 347~348
时间，time 381
桩转换，stubbed transition 357
桩状态，stub state 356
状态，state 341
表，table of 58
状态机，state machine 346
嵌套规则，nesting rule 110
视图，view 352
执行语义，execution semantics 346
指定类行为，specifying behavior of class 351~352
指定执行次序，specifying execution order 351~352
状态图，statechart diagram 352
子，child 144
子类，subclass 358
子类型，subtype 362
子系统，subsystem 360
子状态，substate 360
子状态机，submachine 358
引用状态，reference state 358
字符串，string 355
字体使用，font usage 221
自(身)，self
关键字，keyword 377
语言，language 196
转换，transition 329
执行语义，execution semantics 346~349
组成，composite
对象，object 174
关键字，key 313
聚合，aggregation 174
类，class 174
状态，state 175
历史状态，history state in 228
组成记录，record as composite 178
组合/合并，combination 160
组合，composition 177
组合嵌套，nesting for composition 179
祖先，ancestor 117
作用域，scope 328
操作的作用域，of operation 291