第一部分　UML和UP介绍

第1章　什么是UML

1.1　章节指示图

本节提供了对UML的历史和高级结构的简要回顾。我们在此提到的许多课题将会在以后章节中展开论述。

初学者应该从了解UML的历史和原则起步，如果你有使用UML的经验，或者你确信对UML的历史已经有了充分了解，你可以跳过这部分，直接阅读1.7节，该节讨论UML的结构。这种讨论有三条主线，读者可以以任一顺序进行阅读。要了解UML构造块，请参见1.8节；要了解UML公共机制，请参见1.9节；要了解构架和UML，请参见1.10节。

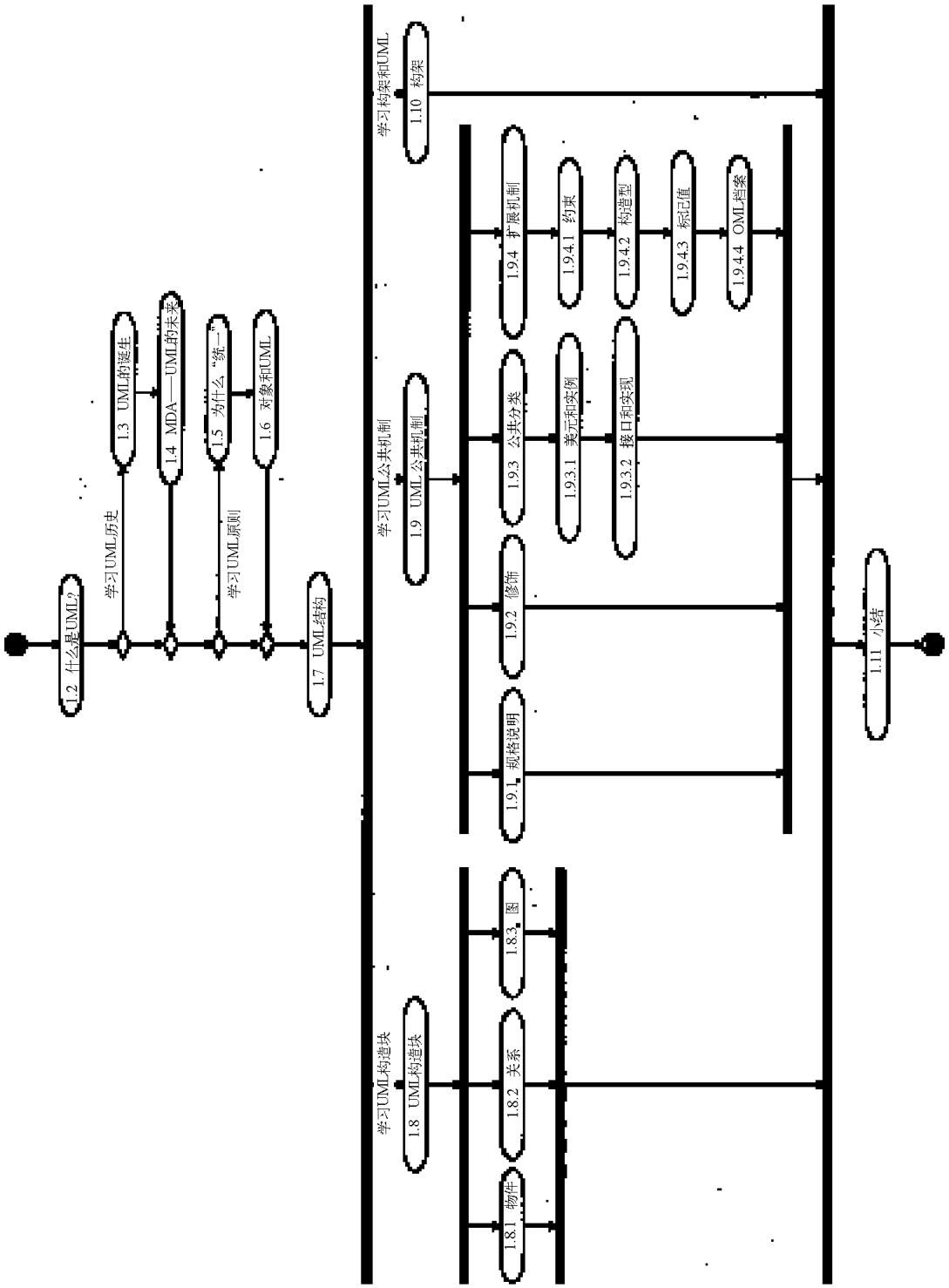

1.2 什么是UML?
学习UML历史
1.3 UML的诞生
1.4 MDA——UML的未来
学习UML原则
1.5 为什么“统一”
1.6 对象和UML
1.7 UML结构
学习UML构造块
1.8 UML构造块
1.8.1 物件
1.8.2 关系
1.8.3 图
学习UML公共机制
1.9 UML公共机制
1.9.1 规格说明
1.9.2 修饰
1.9.3 公共分类
1.9.3.1 差示和实例
1.9.3.2 接口和实现
1.9.4 扩展机制
1.9.4.1 约束
1.9.4.2 构造型
1.9.4.3 标记值
1.9.4.4 OML档案
学习构架和UML
1.10 构架
1.11 小结

图 1-1

1.2 什么是UML

统一建模语言（Unified Modeling Language，UML）是通用可视化建模语言。尽管UML常常与建模OO软件系统相关联，但是由于它内建的扩展机制，它具有更加广阔的应用范围。

UML被设计用来整合建模技术和软件工程领域中当前最好的实践。同样，它被明确设计成可由UML建模工具所实现。这是基于对现实的一种认识：现代的、现代的软件系统通常需要工具的支持。UML图是人类可读的，而且它也易于由计算机表现。

要认识到UML不提供给我们任何一种建模方法论。当然，方法论的某些方面可以由包含在UML模型的元素所暗含，但是UML本身仅仅提供可以用于创建模型的一种可视化语法。

统一过程（Unified Process，UP）是一种方法论——它告诉我们为了建模一个软件系统所需要利用、执行或者创建的工作者（worker）、活动（activity）和制品（artifact）。

UML没有束缚于任何特定方法论或者生命期，它却真正能够与所有现有的方法论一起使用。UP使用UML作为它的底层可视化建模语法，因此你可以认为UP是UML的首选方法，因为它是适应于UML最好的方法，但是UML本身能够（并且的确）为其他方法提供了可视化建模支持。请参见www.open.org.au网站上的OPEN（Object-Oriented Process，Environment and Notation）方法。

UML和UP的目标一直是支持和封装过去10年内软件工程中最好的实践。为此，UML 和 UP 统一了可视化建模语言和软件工程过程中先前的尝试，使之成为最好的解决方案。

1.3 UML的诞生

1994年以前，OO方法领域有点混乱。存在几种互相竞争的可视化建模语言和方法学，它们各有优缺点，各有支持者和批评者。在可视化建模语言方面（概况见图1-2），突出的领导者是 Booch（Booch方法）和Rumbaugh（对象建模技术，或简称为OMT），它们占据一半以上的市场。在方法学方面，到目前为止，Jacobson方法是最强有力的方法，尽管很多作者也宣称他们发明了“方法”，但实际上它们中的大多数只是一种可视化建模语法以及一组或多或少有用的语言和指南。

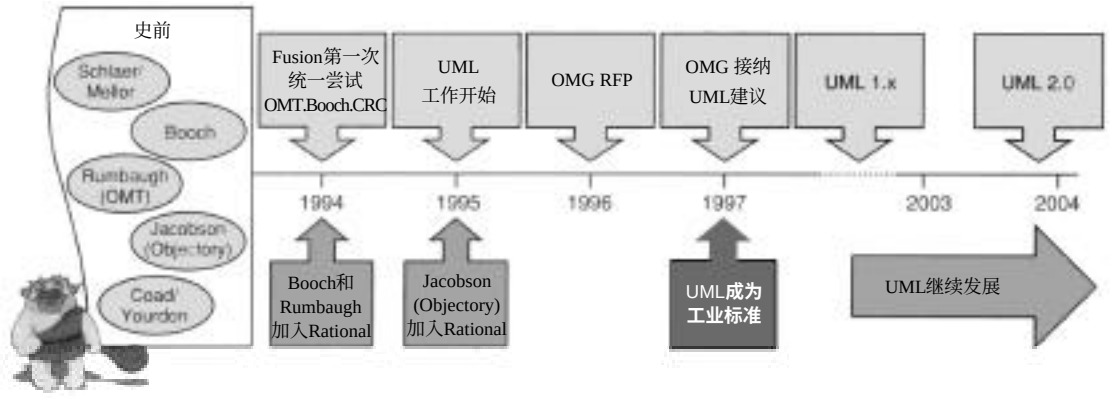


图 1-2

1994年与Coleman的Fusion方法的统一工作是非常早的一次尝试。尽管值得称道，但是这次尝试没有包含主要方法的原创作者（Booch、Jacobson和Rumbaugh），而且说明该方法的书很晚才面

市。Fusion很快就被另一种事件（UML）所赶超，同样是在1994年，Booch和Rumbaugh加入Rational公司，致力于创建UML语言。当时，这种交替曾使我们为之忧心忡忡，因为它使Rational公司赢得了大半个方法论市场。然而，事实证明这些担心完全是多余的，UML从此已经成为一个开放的工业标准。

1996年，UML被OMG（Object Management Group）提议为OO可视化建模语言的推荐标准，UML被提交。1997年，OMG采纳了UML，第一个开放的OO可视化建模语言工业标准诞生了。至此，所有的竞争方法渐渐淡去，UML无可争议地成为OO建模语言的工业标准。

2000年，UML 1.4将一个重大扩展引入UML，添加了动作语义。这些动作语义描述了一组（可以由特定动作语言实现的）原始动作的行为。动作语义加上动作语言允许在UML模型中直接书写UML行为元素（例如类的操作）的详细规格说明。这是一个意义重大的发展，它使得UML规格说明在计算上更加完整，并且使UML模型可执行变得可能。例如，要看遵循动作语义的动作语言的UML实现，请参见Kennedy Carter（www.kc.com）的xUML。

2005年，在我们更新本书第2版的时候，UML 2.0 规范已经完成。UML现在是一门非常成熟的建模语言。从UML规范开始发布到现在已经走过了7个春秋，全球成千上万的软件开发项目的实践证明了UML的价值。

UML 2.0引入了很多新的可视化语法。其中一些语法取代（并阐明了）1.x的语法，另一些语法是全新的，添加到UML中表示新的语义。对于特定元素如何展示，UML总是提供很多可选选项，不是所有的这些可选选项都会被每一种建模工具支持。我们将在本书中一致地使用最通用的语法变体，并且突出我们认为在通常建模场合有用的其他变体。一些语法选项非常特殊，我们将一笔带过。

尽管UML 2.0相比较UML1.x 存在很多语义改变，但基本的原则几乎相同。习惯于使用UML 先前版本的建模者迁移到UML 2.0会非常容易。实际上，UML 2.0中最深刻的变化在于UML的元模型，而且大多数建模者不会直接碰到这种变化。UML 元模型是UML 语言的模型，它本身也是采用UML的子集进行描述的。UML元模型精确地定义了你将在本书中遇到的所有UML建模元素的语法和语义。对于UML元模型的这些修改主要涉及改善UML 规范的精确性和一致性。

Grady Booch 在他的一本书中说“如果你有好的思想，那么它也是我们的”，这其实从一方面概括了UML的哲学——它吸取已有的精华并且在其上进行整合和构造。这是最广泛意义上的复用，UML 同许多来自于“史前”方法的最好思想融合，而拒绝那些特质极端的东西。

1.4 MDA——UML的未来

目前OMG启动模型驱动的构架（Model Driven Architecture，MDA）的项目可能定义UML 的未来。尽管本书不是一本有关MDA 的书籍，但是我们将在本节简要介绍MDA。在OMG的 MDA 网站（www.omg.org/mda）上以及在[Kleppe 1]和[Frankel 1]的书中，我们能够找到更多信息。

MDA定义如何基于模型开发软件的愿景。愿景的精髓是模型驱动可执行软件构架的产生。这个愿景在目前一定程度上已经发生，但是MDA主导该过程自动化程度的成就颇少。

在MDA中，通过遵循MDA的建模工具的一系列模型转换来生产软件。抽象的、独立于计算机的模型（Computer-Independent Model,CIM）被用做独立于平台的模型PIM（Platform-Independent Model,PIM）的基础。独立于平台的模型PIM）被转换为特定平台的模型PSM（Platform-Specific

Model,PSM），然后被转换成代码。

模型的MDA记号非常普通，代码被看作是非常具体的模型。图1-3演示MDA模型转换链。

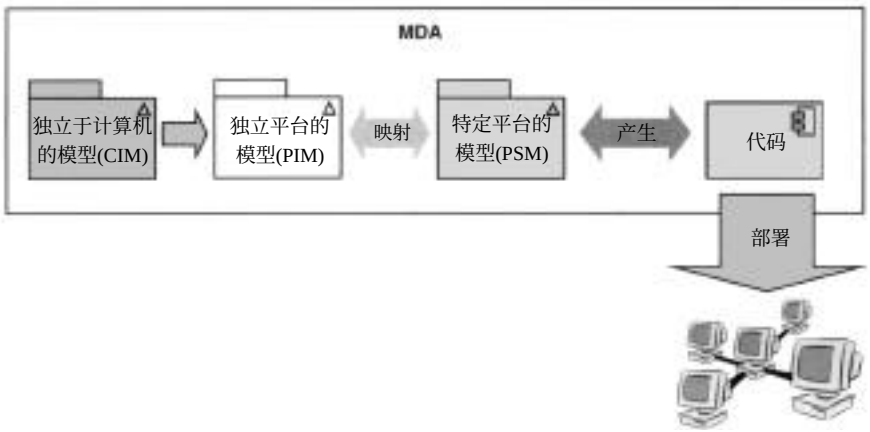


图 1-3

CIM是非常高级层次的抽象模型，它以一种独立于计算机的方式捕获了系统的关键需求以及问题域的词汇。它是真正的、你想要自动化的业务模型。该模型的创建是可选的，如果你选择创建它，那么你可以把它作为产生PIM的基础。

PIM是独立于任何底层平台的（例如,EJB、.NET等等）、表达软件系统业务语义的模型。PIM通常是与分析模型处于相同抽象层次的模型，我们将在本书稍后讨论这些抽象模型,PIM更加完整。因为PIM必须提供将被转换成PSM（从PSM 可以产生代码）的足够完整的基础。给你提个醒儿：术语“独立于平台”缺失了重要信息，除非你定义出想要独立的平台。不同的MDA 工具支持不同层次的平台独立。

PIM带有特定平台的信息用以创建PSM。针对目标平台从PSM 生成代码。

从原则上讲，只要PSM 信息充分完整，就能够产生100%的代码和辅助制品，例如，文档、测试制品、编译文件和部署描述。如果需要这些发生,UML模型必需具有计算上的完整性—换言之，所有操作的语义必须采用动作语义指定。

如前所述，一些MDA 工具已经提供了动作语言。例如，Kennedy Carter（www.kc.com）的 iUML工具提供了动作说明语言（Action Specification Language，ASL），ASL 遵循UML 2 的动作语义。该动作语言比诸如Java和 C++处于更高的抽象层次上，你可以采用它创建计算上完整的UML模型。

其他的工具，例如，ArcStyler（www.io-software.com）允许产生70%-90%的代码和其他制品，但是操作的主体仍然需要采用目标语言（例如,Java）来补充完整。

在MDA看来，源代码，例如Java和C#代码，仅仅是从UML模型编译产生的“机器代码”这种代码按需直接从PSM生成。因而，在采用MDA开发中，代码在本质上比UML模型具有更低的价值。MDA 把UML从现在的、手工创建源代码的先驱角色转变为代码生成的根本机制。

在本书出版过程中，越来越多的建模工具商在他们的产品中加入了MDA能力。最新信息，请察看OMG MDA网站。而且，有许多非常有前途的开源MDA首创者，例如,Eclipse Modeling

Framework（www.eclipse.org/emf）以及 AndroMDA（www.andromda.org）。

在本书中，我们仅限于提供MDA 的“概图（big picture）”。我们提到的MDA规范实在有限，我们鼓励你查看在本书开始时提到的参考，以获得更多的信息。

1.5 为什么“统一”

UML统一不仅是历史的范畴，而且UML尝试（并且取得巨大成功）统一了几种不同领域。

- 开发生命期—UML提供用于从需求分析工程到实现，贯穿整个软件开发生命期的可视化建模语法。
- 应用领域—UML已被用于从关键实时嵌入系统到管理决策支持系统中任何事物的建模。
- 实现语言和平台—UML中立于语言和平台。当然，它对纯粹的OO语言（Smalltalk、Java、C#等等）具有极好的支持，但对于混和OO语言，如C++和基于面向对象的语言Visual Basic同样有效。它甚至已被用于非OO语言，如C的建模。
- 开发过程—尽管UP及其变体可能是OO系统的首选开发过程，UML能够（并且的确）支持很多其他软件工程过程。
- 它本身内部概念—UML在其内部概念的一个小集合的应用上，勇于尝试保持一致和统一。它不总是（直到目前为止）成功，但它仍是先前尝试的很大进步。

1.6 对象和UML

UML的基本前提是我们能够把软件和其他系统建模为交互对象的集合。显然，这对OO 软件系统和语言非常适合，但是对于商务过程和其他应用，它也工作得很好。

UML 模型具有两个方面。

- 静态结构—描述什么类型的对象对于建模系统是重要的，它们是如何相关的。
- 动态行为—描述了这些对象的生命周期以及它们是如何交互以提供系统所需的功能。

UML模型的这两个方面关系紧密，它们之间不是真正的竞争关系。

我们将在第7章看到对象（和类）的全部细节。直到目前，我们仅认为对象是数据和行为的紧密伴生。换而言之，对象包含信息并且能够执行功能。

1.7 UML结构

你可以通过看看它的结构，开始理解UML是如何作为可视化语言工作的，请参见图1-4（与你以后将要看到的一样，这是合法的UML图）。这个结构包括：

- 构造块—这些是基本UML建模元素、关系和图。
- 公共机制—达到特定目标的UML 公共方法。
- 构架—系统构架的UML 视图。

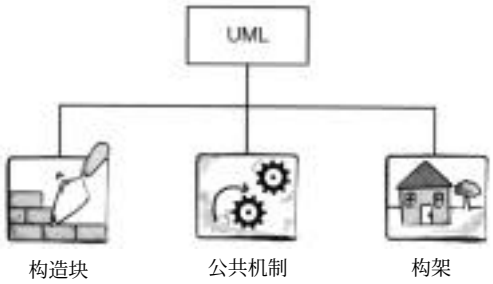


图 1-4

理解UML的结构为本书中展示的其他信息提供了有用的组织原则。同样，它突出了UML本身是事先

设计好的和有构架的系统。实际上，UML是使用UML建模和设计的！这个设计是UML的元模型。

1.8 UML构造块

根据《UML用户指南》(《*The Unified Modeling language Usen Guide*》) [Booch 2]，UML仅由三个构造块组成（见图1-5）：

- 物件 —— 这些是建模元素本身。
- 关系 —— 这些把物件联系在一起。关系说明了一个或多个物件是如何语义相关的。
- 图 —— 这些是UML模型的视图。它们展示物件的集合，“讲述关于软件系统的故事”是可视化系统将作什么（分析级图）或者系统如何作（设计级图）的方法。

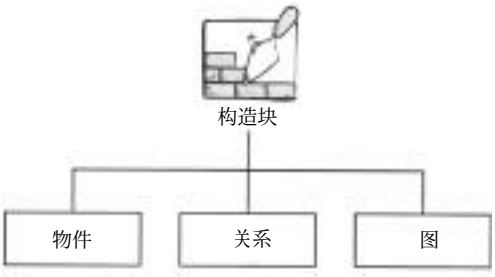


图 1-5

我们将在接下来的三个部分中稍微更详细地看看物件、关系和图。

1.8.1 物件

UML物件可以分成：

- 结构物件 —— UML模型的名词，如类、接口、协作、用例、活动类、组件、节点；
- 行为物件 —— UML模型的动词，如交互、活动、状态机；
- 分组物件 —— 包，它用于把语义上相关的建模元素分组为紧密联系的单元；
- 注解物件 —— 注解，它附加到模型上以捕获特殊信息，同黄色便笺很像。

我们将在第二部分中看到这些物件怎样应用在UML建模中。

1.8.2 关系

关系允许你显示模型中两个或者多个物件互相是如何相关的。考虑家庭以及家庭中成员之间的关系提供给你UML模型中有关关系的很好思想—它们让你捕获物件之间有意义（语义上）的联系。例如，作用于模型中的结构物件和分组物件的关系描述在表1-1中。

理解不同类型关系的确切语义是UML建模中非常重要的部分，但我们将在后面章节中详细揭示它们的语义。

表 1-1

关系类型	UML 语法 源 目标	简要语义	章 节
依赖	---------->	源元素依赖于目标元素，目标元素的改变可以影响源元素	9.5
关联	————	描述对象之间的一组链接	9.4
聚合	◇————	目标元素是源元素的部分	18.4
组合	◆————	强形式的聚合（约束更强）	18.5
包含	⊕————	源元素包含目标元素	11.4
泛化	————▷	源元素是更加通用的目标元素的特化，源元素可以替换目标元素	10.2
实现	----------▷	源元素保证实现由目标元素所说明的契约	12.3

1.8.3 图

在所有UML建模工具中，当你创建了新物件或者新关系时，它被加入到模型中。模型是所有物件和关系的知识库，创建模型帮助你描述正在设计的软件系统所需的行为。

图是模型的窗口或者视图。图不是模型本身。这是非常重要的区别，因为物件或者关系可以从图中删除，或者甚至从所有的图中删除，但是它仍然可以存在于模型中。实际上，它存在于模型中直到显式地被删除。UML建模新手的普遍错误是在图中删除了物件，而把它们留在模型中。

UML一共有13种不同类型的图。它们列举在图1-6中。在该图中，每个方框表示一种图。方框中的文本是斜体的，它表示抽象类型的图。因此，例如，存在六种不同的结构图Structure Diagram）。正常文本表示可以创建的具体的图。阴影方框表示引入UML 2的新图。

我们能够把这组图划分为两类，一类是为系统的静态结构建模的图（静态模型），另一类是为系统的动态结构建模的图（动态模型）。静态模型捕获物件以及物件之间的静态关系；动态模型捕获物件是如何交互以产生软件系统所需的行为。我们将在第二部分中见到静态模型和动态模型。

创建UML图没有特定顺序，尽管通常你从使用用例图定义系统范围开始。实际上，当你揭示越来越多的有关正在设计的软件系统的信息和细节时，你常常并行工作在几种图上，精化这些图。图既是模型的视图，又是向模型中输入信息的主要机制。

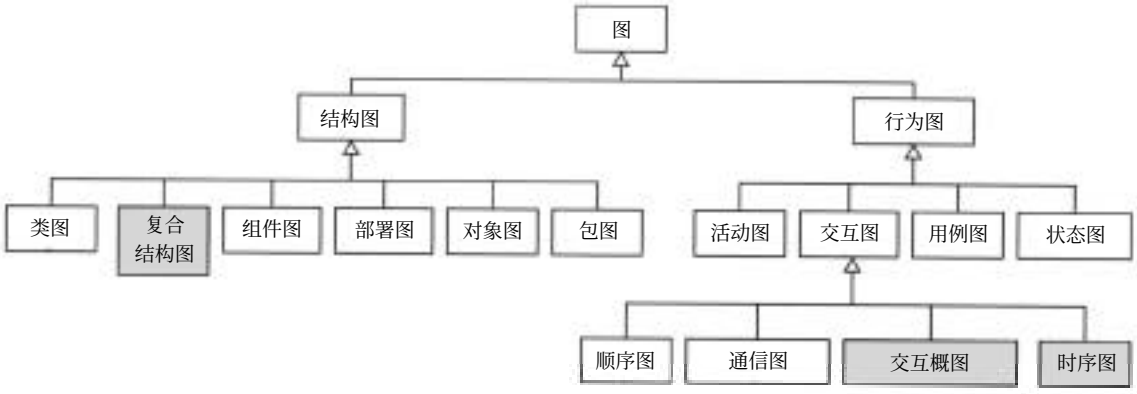
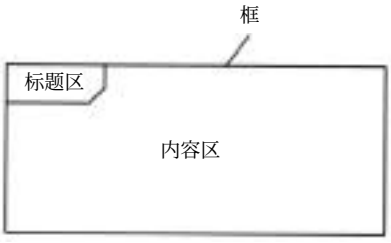


图 1-6

UML 2为图引入了新的语法，如图1-7所示。每个图可以具有框、标题区和内容区。标题区是一个不规则的五角形，它包括图的类型（可选）、名称以及参数（可选）。



标题区语法：<kind><name><parameters>
注意：<kind>和<parameters>可选

图 1-7

<kind>说明该图的类型，应该是图1-6中所列出的、具体图中的一种。UML规范说明<kind>可以缩写，但却没有提供缩写的列表。你几乎不需要显式地指定<kind>，因为从可视化语法看它通常非常清楚。

<name>应该描述图的语义（例如,CourseRegistration），并且<parameters>提供图中建模元素所需的信息。你将在看到使用<parameters>的示例。

随便说一下，图可以具有暗示框。这是在建模工具中表示图的区域。你可以在图1-8中看到暗示框的示例。

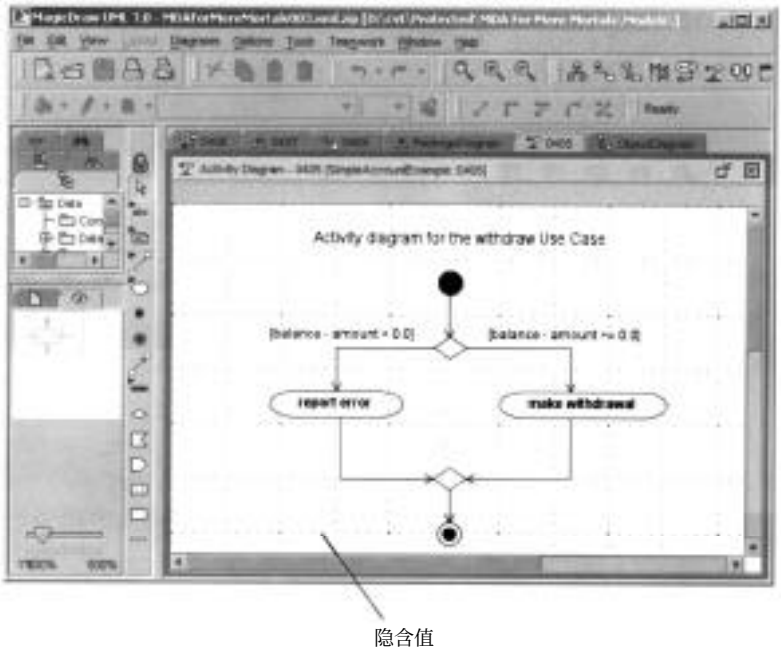


图　1-8

1.9　UML公共机制

UML具有四种公共机制，它们一致地应用在语言当中。它们描述为达到对象建模目的的四种策略，它们在UML的不同语境中反复运用到。我们再一次看到，UML 具有简洁和雅致的结构（图1-9）。

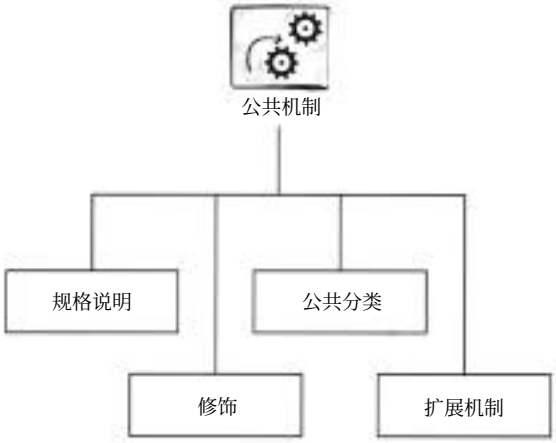


图　1-9

1.9.1 规格说明

UML模型元素具有至少两种维度—图形维度，它允许你使用图和图标可视化模型；文本维度，它由各种建模元素的规格说明所组成。规格说明是元素语义的文本描述。

例如，我们可以用带有各种分栏的方框（图1-8）可视化类，如类BankAccount，但是这个表示实际上没有告诉我们任何有关该类的业务语义的任何信息。建模元素背后的语义是用它们的规格说明来捕获的，没有这些规格说明，你只能猜想建模元素实际上代表什么。

规格说明集合是模型的真正的“内核”，且形成了承载模型的语义背板（semantic backplane），赋予模型意义。各种图仅仅是该背板的视图或者可视化投影。

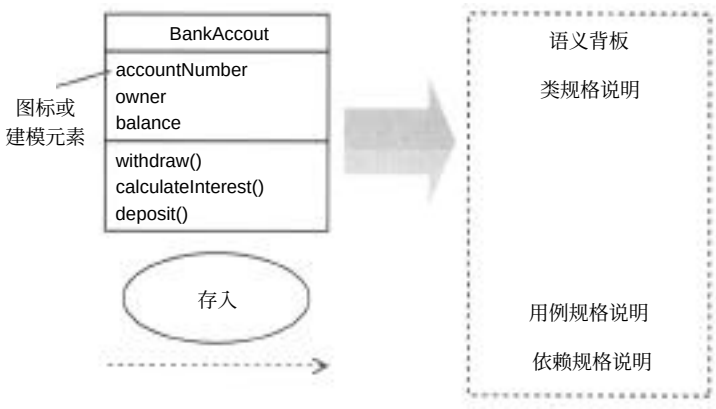


图 1-10

通常，使用UML建模工具维护这个语义背板，它为每个建模元素输入、查看和修改规格说明提供方法。

UML为创建模型提供了大量的灵活手段。特别的，模型可能是：

- 省略的 —— 某些元素存在于背板中，但是为了简化视图，隐藏于特殊图中；
- 不完整的 —— 模型中的某些元素可能完全丢失；
- 不一致 —— 模型可能包含矛盾。

事实上，完整性和一致性约束是不严格的，这是重要的，正如你知道模型随着时间推移不断进化并且经历很多修改。然而，目标总是朝着为构造软件系统的足够完整的一致性模型前进。

使用UML建模，通常的实践是开始于一个主要的图形模型，它允许你可视化系统，然后，随着模型进化，向背板中加入越来越多的语义。然而，对于任何一个有用或者完整的模型，在背板中必须存在模型语义。如果不存在，你没有有模型，仅仅是由线所连接的无意义的方框和圆块的集合！实际上，新手所犯的通常错误可能被称作“因图而亡（death by diagram）”—— 模型被过度图形化而没有说明。

1.9.2 修饰

UML非常棒的一个特征是每个模型元素都具有一个非常简单的符号，你可以在符号上添加许多修饰，这些修饰使得元素的规格说明可见。使用这种机制，你可以根据特定的需要剪裁可视化信息的数量。

开始，你可以使用基本符号加上一个或者多个修饰创建高级层次的图，然后，随着时间的推移，你可以精化该图，添加越来越多的修饰，直到图足够详细地描述了你的目标。

记住，任何UML图仅是模型的视图，这是很重要的。因此，当你增强图的整体清晰性和可读性或者突出模型的某些重要特征时，你仅应显示那些修饰。通常没有必要表示图中所有的东西。相反，清晰的、准确地演示出你想要展示的要点、容易阅读，这是更加重要的。

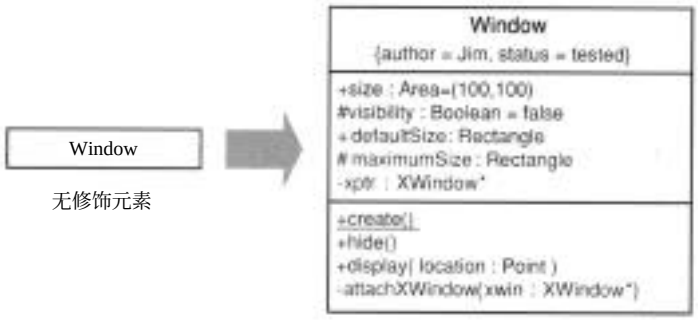


图 1-11

图1-11显示类的小图标是带有类名称的方框。然而，你能够使用修饰扩展这个小视图以暴露底层模型的各种特征。灰色文本表示可选的、可能的修饰。

1.9.3 公共分类

公共分类描述看待世界的特殊方法。UML 中有两种公共分类——类元/实例和接口/实现。

1.9.3.1 类元和实例

UML 考虑，我们可能具有一类事物的抽象概念（如bank account），然后再有抽象的、特定的、具体的实例（如“my bank account ”或者“ your bank account ”）。一类事物的抽象概念是类元，特定、具体事物是实例。这是非常重要的概念，实际上，很容易掌握。类元和实例无处不在。想像这本书——我们可以说，本书的抽象概念是“统一建模语言和统一过程”，而你正在阅读的是这多本书的实例，如你正在阅读的这个。我们将看到类元/实例的概念是渗透UML中的重要概念。

UML中，实例通常具有与相应类元相同的图标，但是实例图标的名称具有下划线。这是首先要掌握的非常微妙的可视化差异。

UML2提供了类型丰富的13个类元。比较常用的一些类元请参见表1-2，我们将在以后章节中看到所有这些（和其他）类元的细节。

表 1-2

类 元	语 义	章 节
Actor（参与者）	由系统的外部用户扮演的角色，系统传递给它们值	4.3.2
Class（类）	共享相同属性的一组对象的描述	7.4
Component（组件）	系统的模块化和可替换的部分，它封装了内容	19.8
Interface（接口）	操作的集合，用于说明由类或组件所提供的服务	19.3

（续）

类　元	语　义	章　节
Node（节点）	物理的、运行时元素，它表示可计算资源，例如，PC就是节点的一个例子	24.4
Signal（信号）	对象之间传递的异步消息	15.6
Use case （用例）	动作序列的描述，系统执行用例向用户产生值	4.3.3

1.9.3.2 接口和实现

这个原则是分离作什么（接口）和如何作（实现）。例如，当你开车时，你与非常简单和良好定义的接口交互。不同的车以不同方式实现该接口。

接口定义了一份契约（它与一份法律合同很相似），特定的实现保证符合它。契约和契约的实现相分离是重要的UML概念。我们将在17章中详细讨论它。

接口和实现的具体例子随处可见。例如，录像机前端按钮为实现非常复杂的机制提供了（相对）简单的接口。接口通过隐藏复杂性把我们与它们隔离。

1.9.4 扩展机制

UML的设计者意识到，不可能简单地设计一种完全统一的，能够满足现在和将来所有人的需要的建模语言，因此UML整合了三种简单扩展机制，我们将它们总结在表1-3中。

我们将在接下来的三个章节中看到关于这三种扩展机制的详细信息。

表 1-3

UML 扩展机制	
约束	通过添加新规则来扩展元素的语义
构造型	构造型允许我们基于已有模型元素定义新的UML建模元素 —— 我们自己定义构造型的语义 构造型给UML元模型添加新元素
标记值	通过允许我们添加新的特殊信息来扩展模型元素的规格说明

1.9.4.1 约束

约束是花括号中（{}）中的文本字符串，它说明必须维持为真的那些有关建模元素的条件和规则。换言之，它在某些方面约束该元素的某些特征，你将在通篇中碰到约束的示例。

UML定义了约束语言—对象约束语言（Object Constraint Language，OCL）作为标准的扩展。我们将在 25 章中介绍 OCL。

1.9.4.2 构造型

《UML参考手册》(《The UML Reference Mannual》)[Rumbaugh 1]说：“构造型表示已有模型元素的变体，它们具有相同的形式、不同的目的”。

构造型允许你基于已有元素引入新的建模元素，你能够给新元素在符号（例）中添加构造型的名称。每个模型元素可以具有零个或多个构造型。

每个构造型可以定义一组标记值和约束，它们作用于构造型化的元素。你也可以给构造型赋予图标、颜色和纹理。通常，在UML模型中，避免使用颜色和纹理，因为某些用户（例如，色盲）可能

难于理解该图，并且图常常是以黑白形式打印出来的。然而，一般的做法是给预定义构造型赋予新的图标，这允许你以受控的方式来扩展UML图形符号。

因为构造型引入了面向不同目的的新建模元素，你必须定义这些元素的语义。如何作到？如果建模工具没有提供存档构造型的内嵌支持，大多数建模者仅在模型中加入一个注解，或者在被定义构造型中添加外部文档一条引用。目前，支持构造型的CASE 工具相当可怜——大多数工具在一定程度上支持构造型，但不是所有工具都提供捕获构造型语义的功能

你能够使用带有特殊的UML预定义的构造型《stereotype》的类元素（第7章）来建模构造型。这创建了系统构造型的元模型。它是元模型，因为它是建模元素的模型，它与通常UML系统或者业务模型是处于完全不同的抽象层次。因为它是元模型，你永远不要把它同你的正常模型归并在一起你必须把它保存在独立的模型中。为构造型而创建的新模型，只有在有很多构造型的情况下，才能是值得去作的事。这很少见，因此大多数建模者倾向用注释或者外部文档存档构造型。

如何显示构造型有很多方法。然而，大多数建模者仅使用在《》中的构造型名称或者图标。其他变体使用的不多，并且CASE工具常常限制你。一些示例显示在图1-12（星号不是UML语法的一部分——它们仅是突出最有用的显示选项）中。

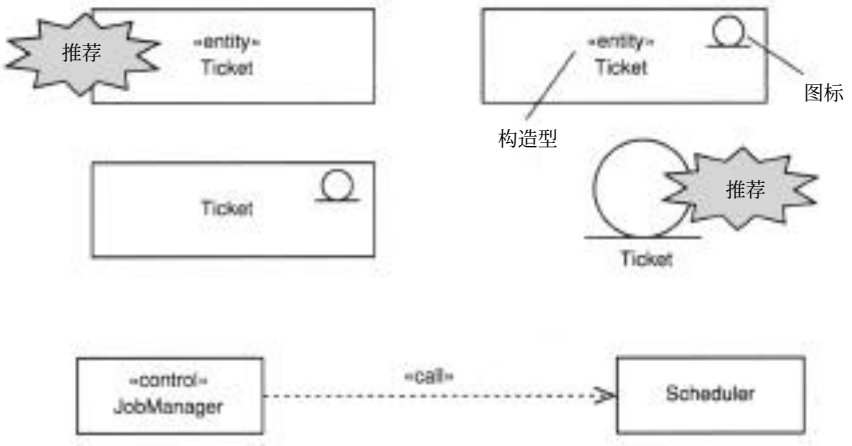


图 1-12

注意，你能够像类一样构造型化关系。你将在书中看到很多这样的用法。

1.9.4.3 标记值

UML中，特性是缚到模型元素的任何值。大多数元素具有许多预定义的特性。其中一些显示在图中，另一些是模型中语义背板的一部分。

UML 允许你使用标记值为建模元素添加你自己的特性。标记值是个非常简单的思想——它仅是能够具有值的关键字。标记值的语法表示如下：{tag1 = value1, tag2 = value2, ⋯, tagN = valueN}。这是用逗号分隔的、标记/值的列表，标记和值之间由等号分隔。标记值列表由花括号括起来。

一些标记仅是作用于模型元素的附加信息，如{author = Jim Arlow}。其他标记表示由构造型定义的新建模元素的特性。你不应该直接为模型元素应用这些标记，而应该把它们同构造型关联起来，然后，当构造型作用于模型元素时，它同样获得与该构造型相联系的标记。

1.9.4.4 UML档案

UML档案是构造型、标记值和约束的集合体。你可以使用UML档案为特殊目的而定制UML。

UML档案允许你定制 UML，以便你可以把它高效地应用于不同的领域。档案允许你以一致和良好定义的方式使用构造型、标记值和约束。例如，如果你使用UML建模 .NET应用，那么你使用UML.NET 档案，它列表在表1-4中。

表 1-4

构造型	标记	约束	扩展	语义
《NETComponent》	没有	没有	组件	表示.NET框架中的组件
《NETProperty》	没有	没有	属性	表示组件的属性
《NETAssembly》	没有	没有	包	.NET 运行时程序包
《MSI》	没有	没有	制品	组件自安装文件
《DLL》	没有	没有	制品	可移植的执行程序DLL
《EXE》	没有	没有	制品	可移植的执行程序EXE

该档案是UML2.0 规范当中的示例UML档案之一，它定义了新的UML建模元素，被采用用于建模.NET 应用程序。

档案中的每个构造型扩展了UML元模型元素（例如，类（Class）和关联（Association）），以创建新的、客户化的元素。构造型可以定义原始元素没有的、新的标记值和约束。

1.10 构架

《UML参考手册》(《The UML Reference Manual》) [Rumbaugh 1] 把系统构架定义为："系统的组织结构，包括系统分解的组成部分、它们的关联性、交互、机制和指导原则，这些提供系统设计的信息"。IEEE 把系统构架定义为"在其环境中，系统的最高级概念"

构架是关于捕获系统高级层次结构的策略。审视架构存在很多方法，但是一个非常通用的方式是"4+1视图"，它由Philippe Kruchten[Kruchten 2]所描述。系统构架的主要方面被捕获为该系统的四张视图——逻辑视图、进程视图、实现视图和部署视图。它们由第五个视图——用例视图整合到一起，图1-13演示了它们的关系。

让我们依次看看这些视图。

- 逻辑视图——捕获问题域的词汇，作为类和对象集合。重点是展示对象和类是如何组成系统、实现所需系统行为。
- 进程视图——建模系统中作为活动类（有自己控制线程的类）的可执行线程和进程。其实，它是面向进程的逻辑视图的变体，包含所有相同的制品。
- 实现视图——建模组成系统的物理代码的文件和组件。它同样展示出组件之间的依赖，展示一组组件的配置管理以定义系统的版本。
- 部署视图——建模把组件物理地部署到一组物理的、可计算节点上，如计算机和外设上。它允许你建模横跨分布式系统节点上的组件的分布。
- 用例视图——该视图把系统的基本需求捕获为一组用例（请参见第4 章）以及提供构造其他视图的基础。

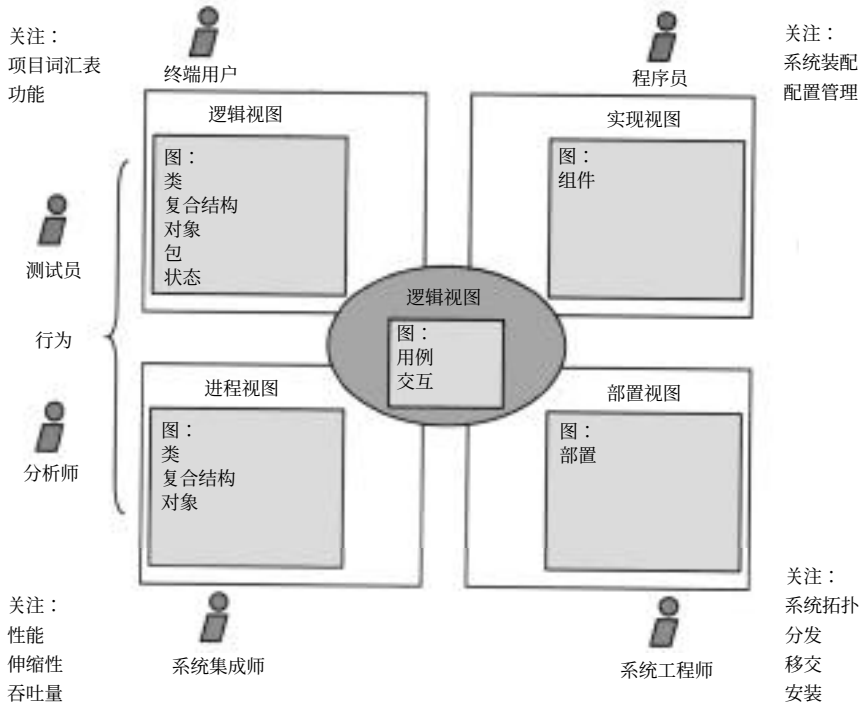


图1-13　源自 [Kruchten 1] 的图 5-1 获得 Addison-Wesley 的授权

正如你将在本书后面看到的那样，UML为4+1视图提供了极好地支持，并且UP 是需求驱动的方法，它与4+1模型非常匹配。

一旦你创建了4+1视图，你已经用UML模型浏览了系统的所有主要方面。如果你遵循UP生命周期，这个4+1构架不是一次创建即可，而是不断演进的。在UP的框架内UML的建模过程是一个逐步精化的过程，最终生成捕获允许构造系统的、足够信息的4+1构架。

1.11　小结

本章介绍了UML的历史、结构、概念和主要特征。至此，你已经学习了以下内容：

- 统一建模语言（UML）是一个开放的、可扩展的工业标准的可视化建模语言，由OMG 所批准。
- UML 不是方法论。
- 统一过程（UP） 或其变体是一种方法论，它是UML的最好补充。
- 对象建模把世界看作交互对象的系统。对象包含信息并且能够执行功能。UML 模型有：
  —静态结构—— 什么类型的对象是重要的，它们是如何相关的；
  —动态结构—— 对象是如何协作来执行系统的功能。
- UML 由三个构造块组成：
  —物件；
  - 结构物件是UML模型的名词；
  - 行为物件是UML模型的动词；

- 仅有一个分组物件，包—运用它来分组语义相关的物件；
- 仅有一个注解物件，注解—它就像一个黄色便笺；

—关系把物件链接到一起；

—图展示了模型的有趣视图。

• UML具有四种公共机制：

—规格说明，它是模型元素的特征和语义的文本描述—模型的“肉”。

—修饰，它是图中建模元素上暴露的信息项以表现某个要点；

—公共分类：

- 类元和实例：
  - 类元—一类事物的抽象概念，例如bank account；
  - 实例—一类事物的特定实例，例如my bank account；
- 接口和实现：
  - 接口—说明事物行为的契约；
  - 实现—事物是如何工作的特殊细节。

—扩展机制

- 约束允许对模型元素添加新的规则；
- 构造型基于已有的建模元素引入新的建模元素；
- 标记值允许为模型元素添加新的特性—标记值是带有相关值的关键字。
- UML profile是一组约束、构造型和标记值—它允许你为特定用途定制UML。

• UML是基于系统构架的4+1视图：

—逻辑视图—系统功能和词汇；

—进程视图—系统性能、可伸缩性和吞吐量；

—实现视图—系统组装和配置管理；

—部署视图—系统的拓扑结构、分布、移交和安装；

—这些视图由用例视图所统一，它描述利益相关人（stakeholder）的需求。