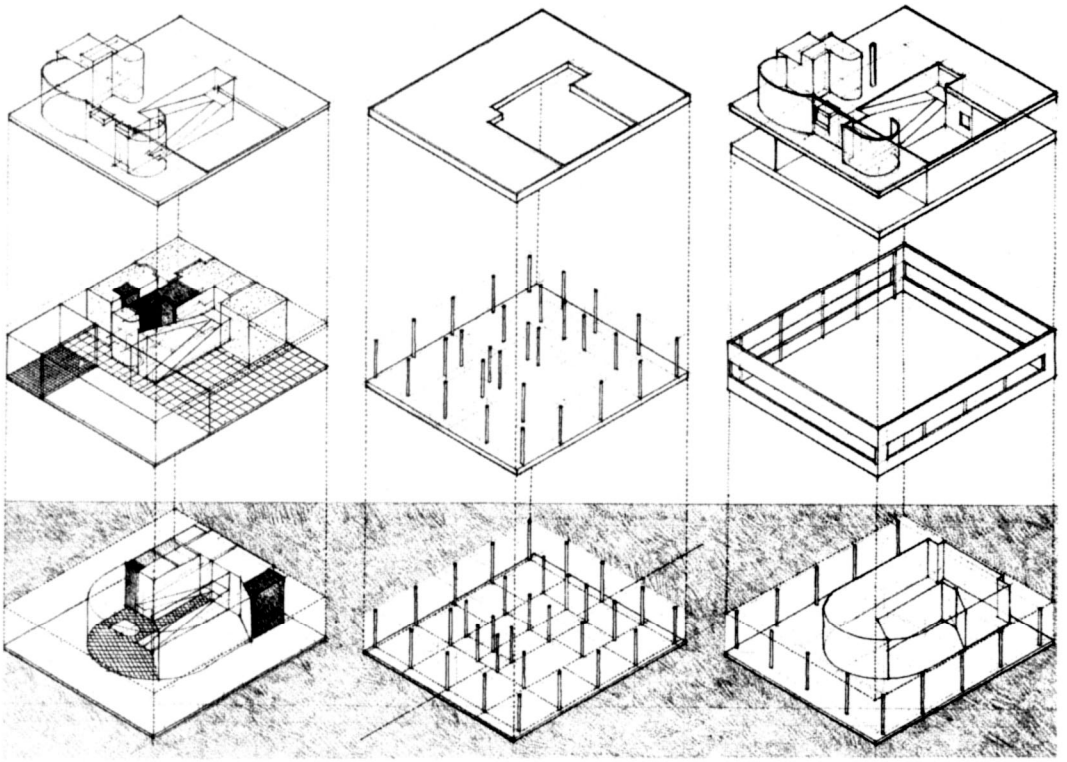
第二部分

对基本结构建模



第4章 类

本章内容：

- 类、属性、操作和责任
- 对系统的词汇建模
- 对系统中职责的分布建模
- 对非软件事物建模
- 对简单类型建模
- 进行质量抽象

类是任何面向对象系统中的最重要的构造块。类是对一组具有相同属性、操作、关系和语义的对象的描述。一个类实现一个或多个接口。

用类捕获正在开发的系统的词汇。这些类可以包括作为问题域的一部分的抽象，也可以包括构成实现的那些类。可以用类描述软件事物和硬件事物，甚至也可以用类描述纯粹是概念性的事物。【第9章讨论类的高级特性。】

结构良好的类具有清晰的边界，并形成了整个系统的职责均衡分布的一部分。

4.1 入门

对系统建模涉及到识别出对于你的特定视图来说是重要的事物。这些事物形成了正在建模的系统的词汇表。例如，如果正在建造一所房子，那么像墙、门、窗户、柜厨和灯对于房主来说就是重要的事物。这些东西中的每个东西都有别于其他东西，有自己的特性集。墙有高度和宽度，是固体的。门也有高度和宽度，是固体的，但它还有额外的行为，使它能朝一个方向开。窗户与门类似，二者都是在墙上开的洞，但它们稍有不同。窗户通常（但不总是）被设计得让人能向外看，而不是让人穿行。

墙、门和窗户很少作为个体单独存在，因此也必须要考虑怎样把这些事物的具体实例组合在一起。识别事物和选择它们之间要建立的关系将受下述因素影响：你希望如何使用住宅的各个房间，你希望各房间如何连通，这种要构造的布局的总体风格与感觉。

用户所关心的事物各不相同。例如，给你建房的水管工对排水沟、存水弯和排水口之类的东西感兴趣。作为房主，你就不需要关心这些东西，你只需关心水管工对这些东西的施工要满足你的要求，例如，排水沟要在地板下，或排水口要接在屋顶。

在UML中，所有的这些事物都被建模为类。一个类是对作为词汇表一部分的一些事物的抽象。类不是个体对象，而是描述一些对象的一个完整集合。这样，你可以把墙在概念上看作是一个对象类，它具有一定的共同属性，如高度、长度、厚度、能否负重等。你也可以考虑墙的个体实例，如“书房的西墙和南墙”。【在第13章中讨论对象。】

对于软件，有很多编程语言直接支持类的概念。这好极了，因为这意味着所建立的抽象经常可以直接地映射到编程语言，甚至可用于对非软件事物的抽象，如“顾客”、“贸易”和“会话”等。

UML也为类提供了图形表示，如图4-1所示。这种表示法允许对抽象进行可视化，而不考虑任何编程语言，这种方式使你可以强调抽象的最重要的部分：名称、属性和操作。

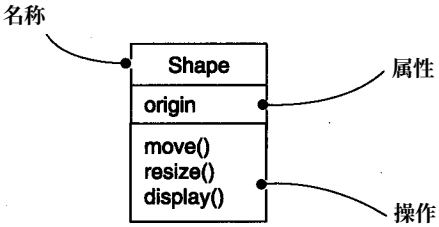


图4-1 类

4.2 术语和概念

类（*class*）是对一组具有相同属性、操作、关系和语义的对象的描述。在图形上，把一个类画成一个矩形。

1. 名称

每个类都必须有一个有别于其他类的名称。名称（*name*）是一个文本串。单独的名称叫做简单名（*simple name*）；用类所在的包的名称作为前缀的类名叫做路径名（*path name*）。所绘制的类可以仅显示它的名，如图4-2所示。【一个包中的各类的名称都必须是唯一的，第12章中要对此进行讨论。】

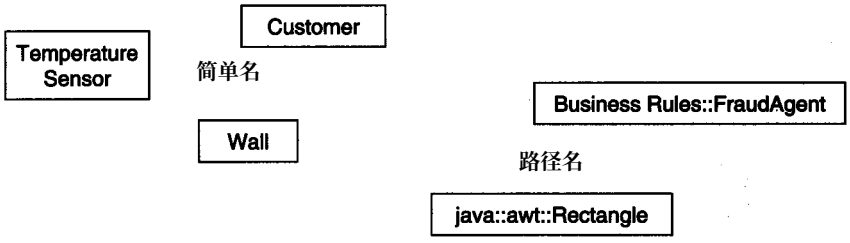


图4-2 简单名和路径名

注释 类名可以是由任何数目的字母、数字和某些标点符号（有些符号除外，例如用于分隔类名和包名的冒号）组成的正文，它可以延伸成几行。实际上，类名是从正在建模

的系统的词汇表中提取出来的短名词或名词短语。通常类名中的每个词的第一个字母要大写，如Customer或TemperatureSensor。

2. 属性

属性（attribute）是已被命名的类的特性，它描述了该特性的实例可以取值的范围。类可以有任意数目的属性，也可以没有属性。属性描述了正被建模的事物的一些特性，这些特性为类的所有对象所共有。例如，每一面墙都有高度、宽度和厚度；可以用这样的方式对顾客建模：每个顾客都有一个名称、地址、电话号码和出生日期。因此，一个属性是对类的一个对象可能包含的一种数据或状态的抽象。在一个给定的时刻，类的一个对象将对该类属性的每一个属性具有特定值。在图形上，将属性在类名下面的栏中列出。可以仅显示属性的名称，如图 4-3所示。【属性与聚合的语义有关，在第10章中对此进行讨论。】

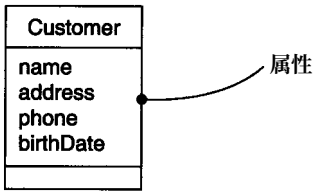


图4-3 属性

注释 属性名可以是像类名那样的正文。实际上，属性名是描述属性所在类的一些特性的短名词或名词短语。通常要将属性名中除第一个词之外的每个词的第一个字母大写，如name或loadBearing。

可通过声明属性的类以及属性可能的缺省初始值，来进一步地详述属性，如图 4-4所示。【可以详述属性的其他特征，例如可把它标记成只读的，或者是由本类的所有对象共享的，这些在第 9章中讨论。】

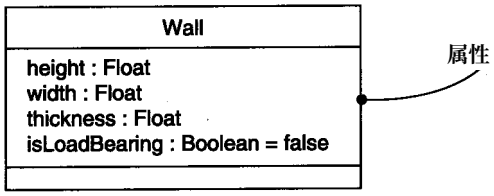


图4-4 属性和它们的类

3. 操作

操作（*operation*）是一个服务的实现，该服务可以由类的任何对象请求以影响其行为。换句话说，操作是你能对一个对象所做的事情的抽象，并且它由这个类的所有对象共享。一个类可以有任意数目的操作，也可以没有操作。例如，像 Java的包awt中那样的窗口库，类Rectangle的所有对象都能被移动或调整大小，还可以查询它们的特性。调用对象的操作经常（不总是）会改变该对象的数据或状态。在图形上，把操作列在类的属性栏下面的栏中。可以仅显示操作的名称，如图4-5所示。【可以进一步用节点或活动图详述操作的实现，在第 6章中描述节点，在第19章

中讨论活动图。】

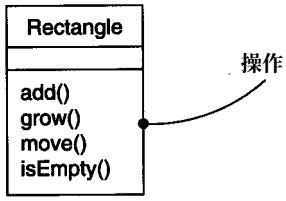


图4-5 操作

注释 操作名可以是像类名那样的正文。实际上，操作名是描述它所在类的一些特性的短动词或动词短语。通常要将操作名中除第一个词之外的每个词的第一个字母大写，如`move`或`isEmpty`。

可以通过阐明操作的特征标记来详述操作，特征标记包含所有参数的名称、类型和缺省值，如果是函数，还要包括返回类型，如图 4-6所示。【可以详述一个操作的其他特性，例如把操作标记为多态的、不变的或可视的，这些在第9章中讨论。】

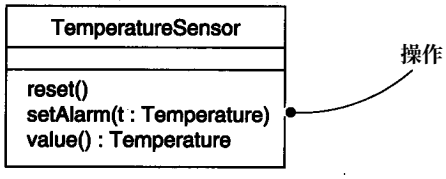


图4-6 操作和它的特征标记

4. 对属性和操作的组织

当画一个类时，不必马上把每个属性和操作都显示出来。事实上，在大多数情况下不能这样做（有太多的属性和操作以至于不能把它们放在一张图中），也可能不应该这样做（可能只有这些属性和操作的一个子集与特定的视图相关）。由于这些原因，可以对一个类进行省略，这意味着可以有选择地仅显示类的一些属性和操作，甚至也可以不显示任何属性和操作。空栏并不一定意味着没有属性或操作，只是没有选择要显示它们。通过在列表的末尾使用省略号（“...”），可以明确地表示出实际的属性和操作比所显示的要多。

为了更好地组织属性和操作的长列表，可以利用构造型在每一组属性和操作之前加一个描述其种类的前缀，如图4-7所示。【在第6章中讨论模板。】

5. 职责

职责（*responsibility*）是类的契约或责任。当创建一个类时，就声明了这个类的所有对象具有相同种类的状态和相同种类的行为。在较高的抽象层次上，这些相应的属性和操作正是要完成类的职责的特征。类`Wall`负责了解墙的重量、宽度和厚度；在信用卡应用系统中，类`FraudAgent`负责处理汇票，并决定汇票是否合法、是否值得怀疑或是否具有欺诈性；类`TemperatureSensor`负责测量温度，若温度达到一定度数，就要发出警报。【职责是一个已定义的构造型的例子，在第6章中讨论这方面的问题。】

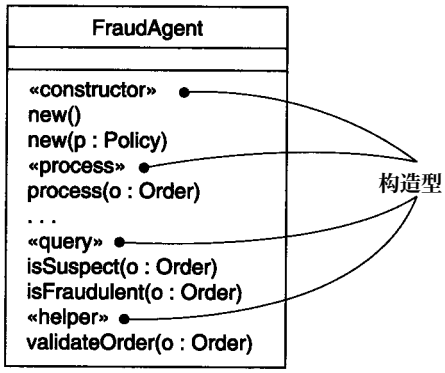


图4-7 用于类特征的构造型

对类建模的一个好的开始点是详述你的词汇表中的事物的职责。像 CRC卡和基于用况的分析等技术在这里是特别有用的。虽然实际上每个结构良好的类都最少有一个职责，最多也是可数的，但类可以有任何数目的职责。当精化模型时，要把这些职责转换成能很好地完成这些职责的一组属性和操作。【在第9章中讨论对类的语义建模。】

在图形上，把职责列在类图符底部的分隔的栏中，如图 4-8所示。【也可以在节点中画出类的职责，这在第6章中讨论。】

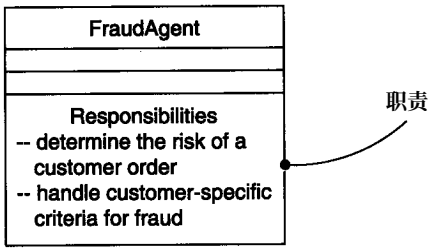


图4-8 职责

注释 职责是自由形式的文本。实际上，可以把单一的职责写成一个短语、一个句子或（最多）一段短文。

6. 其他特征

属性、操作和职责是创建抽象所需要的最常见的特征。事实上，对于大多数要建造的模型，这3种特征的基本形式足以传达类的最重要的语义。然而，有时需要可视化或详述其他特征，例如，对个体的属性和操作进行可视化，对与特定语言相关的操作特征（如多态性或一致性）进行可视化，甚至对类的对象可能产生或操纵的异常事件进行可视化。在 UML中能够表达这些特征以及很多其他特征，但它们被作为高级概念处理。【在第9章中讨论高级的类概念。】

在建造模型时，你很快会发现，几乎每个创建的抽象都是某种类。有时要把类的实现从规格说明中分离出来，对此，在 UML中可以用接口来表示。【在第11章中讨论接口。】

当开始建立更为复杂的模型时，你可能会一次又一次地遇见同样的类，如描述并发进程和

线程的类或描述物理事物（如 applet、Java Beans、COM+对象、文件、 Web页和硬件） 的类。因为这些种类的类是很常见的，并且它们描述了重要的体系结构抽象，所以 UML提供了主动类（表示进程和线程）、构件（表示物理软件构件）和节点（表示硬件设备）。【在第22、24和26章中讨论主动类、构件和节点。】

最后要说明的是，类很少单独存在。确切地讲，当建造模型时，通常要注重于相互作用的那些类群。在UML中，这些类的群体形成了协作，并且通常在类图中被可视化。【在第8章中讨论类图。】

4.3 普通建模技术

4.3.1 对系统的词汇建模

用类来对从试图解决的问题中或从用于解决该问题的技术中得到的抽象进行建模是很平常的。每个这样的抽象都是系统词汇表的一部分，这意味着它们在整体上描述了对用户和实现者来说是重要的事情。

对用户而言，大多数抽象并不难识别，因为通常这些抽象是从用户已经用于描述系统的事物中得出来的。诸如 CRC卡和基于用况分析的技术是帮助用户找到这些抽象的杰出方法。对于实现者来说，这些抽象通常是作为解决方案一部分的技术中的事物。【在第16章中讨论用况。】

为了对系统的词汇建模，需做如下工作：

- 识别用户或实现者用于描述问题或解决问题的那些事物。用 CRC卡和基于用况分析的技术帮助用户发现这些抽象。

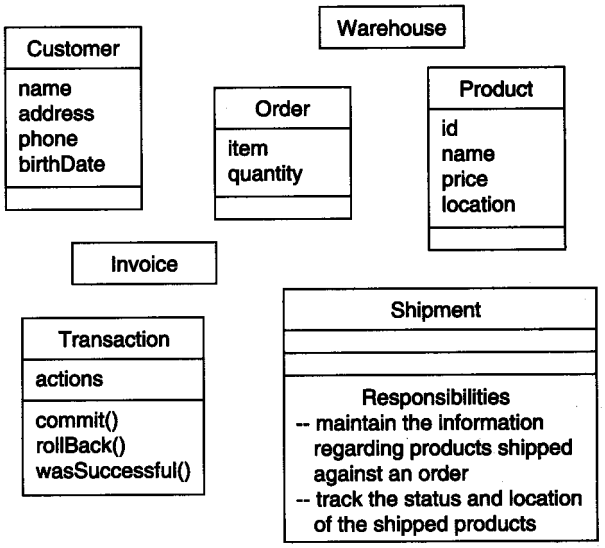


图4-9 对系统的词汇建模

- 对于每个抽象，识别一个职责集。确信要清楚地定义每个类，而且这些职责要在所有的类之间很好地均衡。
- 提供为实现每个类的职责所需的属性和操作。

图4-9描述了从一个零售系统抽取的一组类，其中包括Customer、Order和Product。这个图也包含了一些来自问题的词汇表中的其他的相关抽象，如Shipment（用于跟踪订单）、Invoice（用于按订单开发票）和Warehouse（在发货之前储存货物的地方）。还有一个与解相关的抽象Transaction，用于订货和发货。

随着模型的不断增大，你所发现的很多类将趋于簇集到一些在概念和语义上相关的组中。在UML中，可以用包来对这些类簇建模。【在第12章中讨论包。】

大多数模型很少是完全静态的。相反，系统词汇中的大多数抽象都动态地相互作用。在UML中，有一些对这种动态行为建模的办法。【在本书的第四部分和第五部分中讨论对行为建模。】

4.3.2 对系统中职责的分布建模

一旦开始对大量的类建模，就要保证你的抽象提供了一个均衡的职责集。这意味着不能让任何类过大或过小。每一个类应该做好一件事。若抽象出来的类过大，你会发现模型难以变化且很不容易复用；若抽象出来的类过小，则最后抽象会过多，难以合理地管理和理解。可以使用UML来帮助你可视化和详述这种职责的均衡。

为了对系统中的职责分布建模，要做如下工作：

- 识别一组为了完成某些行为而紧密地协同工作的类。
- 对上述的每一个类识别出一组职责。
- 从整体上观察这组类，把职责过多的类分解成较小的抽象，把职责过于琐碎的小类合成较大的类，重新分配职责以使每一个抽象合理地存在。
- 考虑这些类的相互协作方式，相应地重新分配它们的职责，以使协作中没有哪个类的职责过多或过少。【在第27章中讨论协作。】

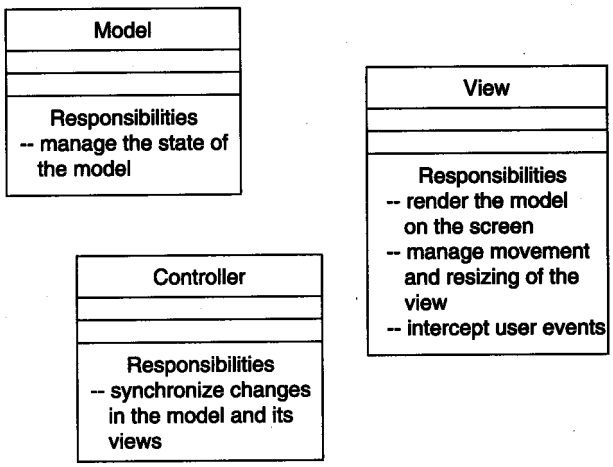


图4-10 对系统中职责的分布建模

例如，图4-10展示了一组取自Smalltalk的类，在图中显示了在类Model、类View和类Controller中的职责分布。请注意所有的这些类如何在一起工作，其中没有过大或过小的类。【这组类形成一个模式，这将在第28章中进行讨论。】

4.3.3 对非软件事物建模

有时，要建模的事物在软件系统中并无类似物。例如，送支票的人和在仓库中按订单对货物进行自动包装以供发货的机器人可能是你所建模的零售系统中的工作流的一部分。你的应用系统可能没有任何描述它们的软件（与上述例子中的顾客不同，因为你的系统可能要维护关于顾客的信息）。

为了对非软件事物建模，要做如下工作：

- 对被抽象为类的事物建模。
- 如果要将这些非软件事物与UML已定义的构造型相区别，就要创建一个新的构造型，并用这个构造型详述这些新语义，并给出明确的可视化提示。【在第6章中讨论构造型。】
- 如果被建模的事物是某种本身包含软件的硬件，考虑把它建模为一种节点，以便能进一步扩充它的结构。【在第26章中讨论节点。】

注释 UML主要用于对软件密集型系统建模，但将它与文本型硬件建模语言（如VHDL）相结合，对硬件系统建模也很有表达力。

如图4-11所示，把人（如AccountReceivableAgent）和硬件（如Robot）抽象成类是完全正常的，因为它们分别描述了具有共同结构和共同行为的一组对象。【系统外部的事物经常被建模为参与者，这要在第16章中进行讨论。】

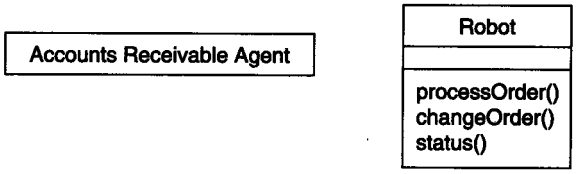


图4-11 对非软件事物建模

4.3.4 对简单类型建模

在其他极端情况下，所建模的事物可能直接取自用于实现一个解的编程语言。通常这些抽象包括简单类型，如整数、字符和串，甚至包括自定义的枚举类型。【在第11章中讨论类型。】

为了对简单类型建模，要做如下工作：

- 对被抽象为类型或枚举的事物建模，这可以用带有适当构造型的类表示符来表示。
- 若需要详述与该类型相联系的值域，可以使用约束。【在第6章中讨论约束。】

如图4-12所示，在UML中可以把这些事物建模为类型或枚举，就像表示类一样，但要显式地用构造型来标记。把像整数（用类int来描述）这样的事物建模为类型，用约束就可以显式地

说明这些事物所呈现出的值域。类似地，像 Boolean和Status这样的枚举类型都能被建模为枚举，它们都有各自作为属性的个体值。

注释 像C和C++这样的一些语言，让你对每一个枚举都设定一个等价的整数值。在UML中，可通过标记属性（表示带有恒定的缺省初始值的枚举）对此建模。

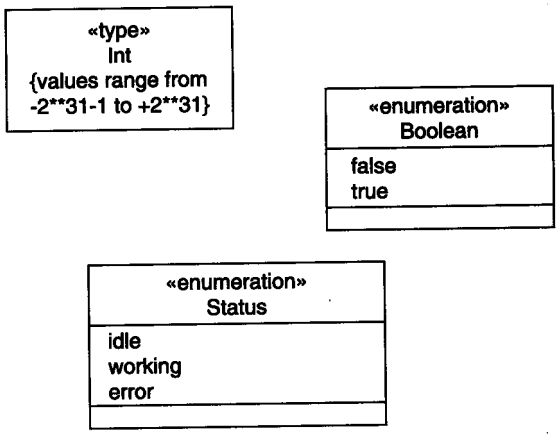


图4-12 对简单类型建模

4.4 提示和技巧

在UML中对类建模时要记住：对最终用户或实现者来说，各个类都应该映射到某个真实或概念性的抽象。一个结构良好的类，要遵循如下的策略：

- 为取自问题域或解域的词汇中的事物提供明确的抽象。
- 嵌入一个小的、明确定义的职责集，并且能很好地实现它们。
- 把抽象的规格说明和它的实现清楚地分开。
- 简单而且可理解，并具有可适应性和可扩展性。

当用UML绘制一个类时，要遵循如下的策略：

- 仅显示在该类的语境中对于理解抽象较为重要的类的特性。
- 通过按种类对属性和操作的长列表分组，来进行组织。
- 把相关的类显示在相同的类图中。