第一部分 UML-F档案

这部分提出构成 UML-F 档案的符号元素。

第1章指出档案（profile）是什么以及为何框架体系结构的开发和适配需要档案。

第2章中的 UML 表示法遵循了 UML 界的趋势。由于 UML 已经变得相当复杂，开发团队经常需要使用子集对系统建模。本章归纳了在几个实际框架项目中已证明有效的 UML 子集，并由此形成 UML-F 基础。UML 子集强调类、对象和顺序图。

第3章讨论了 UML-F 的扩展机制。所谓标记（tag）就是对 UML 扩展机制中的标记值和构造型的一种统一化和改进。本章还给出了用于框架建模的基本 UML-F 标记。

第4章建立在第3章的基本标记之上。它定义了框架构造原则的标记，并讨论了如何定义对设计模式进行建模和注释的标记。

第5章描述了如何将 UML-F 标记和适配配方（adaptation cookbook）结合使用以便在适配过程中支持框架用户。它为第4章所讨论的某些框架构造原则提出了配方。

第1章为何需要框架的 UML 档案

在过去十年中，对象技术广泛运用于软件开发中。 Java 语言及其相关产品对面向对象范型（ paradigm ）的广泛采纳作出了非常重要的贡献。总的说来，对象技术由三个核心概念构成：信息隐藏、继承／多态以及动态绑定。这些组成部分按不同比例混合使用，就定制出了对象技术的各种风格。基于对象的系统强调信息隐藏。面向对象的系统加入继承和动态绑定。面向对象的框架以扩展性为关键特征，从而形成了面向对象系统的特别分支。本章首先描述 UML 档案是什么。然后概述面向对象框架的特点。这些特点提供了比标准 UML 更为简洁的方法来描述这些必需的制品（ artifact)。

1.1 UML 档案

UML 是一种庞大和复杂的语言。但使用当前UML版本仍有许多特性很难方便地表达。因此， UML 提供了允许扩展的机制，比较典型的如标记值（ tagged value ）和构造型(stereotype)。这些扩展可能在所谓的档案（ profile ）中被定义和分组。 UML-F 表示的就是这样一类档案，它着重于框架的描述。我们把UML档案定义为带有特定元素的标准 UML 语言扩展。档案提供了新的符号元素，并且一般对某些元素的语义进行了特例化。它也可能限制 UML 元素的使用。

UML 试图覆盖广泛的应用领域。 它不仅限于描述软件，而且提供描述硬件和真实世界需求的概念。很明显， 这些不同的领域需要不同的建模元素。

因此，UML 不会成为适合所有目的的语言，这一点在很早就非常清楚，尽管它努力地统一语法、语义和用法。相反， UML 是具有一个公共核心的语言族。这个核心定义在 UML 标准文档中，文档由对象管理组织（ OMG，2001 ）制订。

UML 语言族由许多独立成员构成，这些成员扩展、适配、约束、和/或修改 UML 元素的含义。本书提到的 UML-F 变体就是这样的成员。为了系统地引入新家族的语义，UML 提供了档案机制。档案描述了对 UML 标准的修改。这种档案的目标可能在于某一特定应用领域， UML-RT 实时档案（ Powel Douglass 2000 ）就是一个典型示例。其他档案提供了特定工具扩展。例如，它们可能修改 UML 以更好地适用于基于 Web 系统的建模：Java 档案将约束 UML 为单类继承；JavaBean 子档案应该额外提供符号元素以使类标记符合 JavaBean 需求。我们建议读者访问 OMG 的网站以了解当前在定义档案中所做的工作。

档案通常构造为一种层次结构。图 1-1 显示了档案结构的示例。这种档案形成从特定工具扩展、特定项目扩展到已有档案的基础。 UML 标准化过程的当前挑战在于为那些重要的和广泛使用的概念定义标准的含义。仅仅少数扩展是特定于项目或者工具的。从图 1-1 中可推断出某些档案将被标准化为 UML 的一部分。UML-RT 是具有这种趋势的主要档案。更深层次的档案将在 OMG 标准化前公布。其他一些档案可能只在公司内部使用，或者甚至只在项目中使用。特定项目档案可能建立在更多的通用档案上，有时仅仅只需添加一些构造型或者特例化对象序列的含义。 UML-F 档案是计划公开使用的，其设计目标是通用档案工具，特定的项目档案可以此为基础。

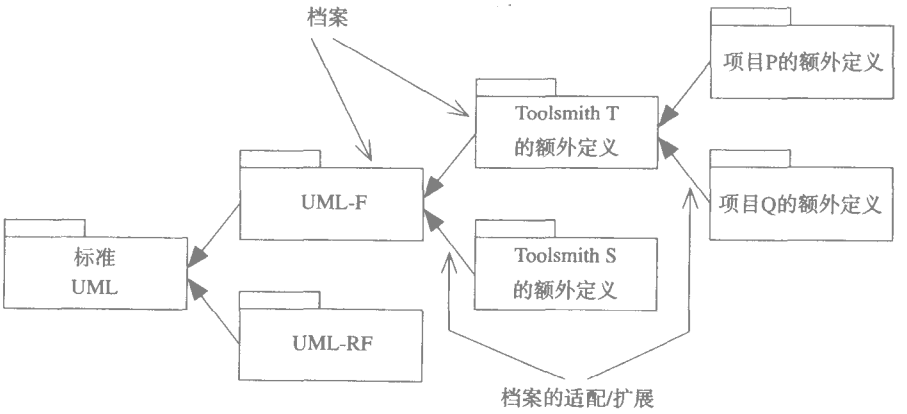

图 1-1 档案层次示例为 UML 包图描述

Cook 等（1999）的文章更详细地描述了档案化机制及其扩展，称之为 preface。

1.2 面向对象的框架扩展是关键

UML-F 档案着重于面向对象的框架，为此我们将在本节定义相关术语。什么是框架呢？面向对象框架并不是使用积木似的构造块来搭建系统，而是预制的类／组件的扩展集合[⊖]。例如，想像一个万能厨房设备。该设备可以加入一个部件作为搅伴器。加入另一个部件可以将该设备转化为绞肉机。这个多功能的设备描述了面向对象框架的特点。它由一些独立组件所构成：如电机、驱动轴、电源插座、开关和包装，所有这些被组装为一个单元，它提供了明确的扩展接口。框架的适配仅需要一个可兼容的插件程序。顺应于接口的任何扩展能

⊖ 虽然 component-ware 一词在软件工程社区变得很流行，但 component 存在很多不同的使用。我们将 componen 定义为使用编程接口的，可作为单元配置的软件的一部分（Pree 和 Templ，2000）。关于对 componen 标准的讨论，请参考 Szyperski（1998）。

够被插入框架中。我们将此同使用类似积木的多个组件构造设备的过程进行比较。只需选择合适的组件，将其装配而无需清楚地定义扩展接口。换言之，框架有效地提高复用，不仅是代码复用，还有体系结构设计的复用。

通常，软件框架是软件的一部分，该软件可以通过编程的回调形式（ Pree 和 Templ，2000）进行扩展。用传统语言所建构的框架依赖于函数和过程参数，为了改变面向对象框架的行为，程序员在框架类的子类中使用继承以覆盖动态绑定方法。注意，从这个观点看来， Java 接口构造的概念也与之相似，它提供概念上的类似扩展机制。这种方法（重）定义类似于在传统编程过程中提供特定过程或函数，在这两个案例中，应用体系结构驻留于框架中。

图 1-2（b）描绘了编程的回调形式。结构化编程语言，如 Pascal，提供过程和函数作为构造模块。图 1-2（a）显示了调用一些小的库例程的应用。将其同图 1-2（b）比较，在图 1-2（b）中，库包含框架，该框架通过将特定过程或函数作为参数传递给框架而被扩展。该框架回调这些过程或函数。

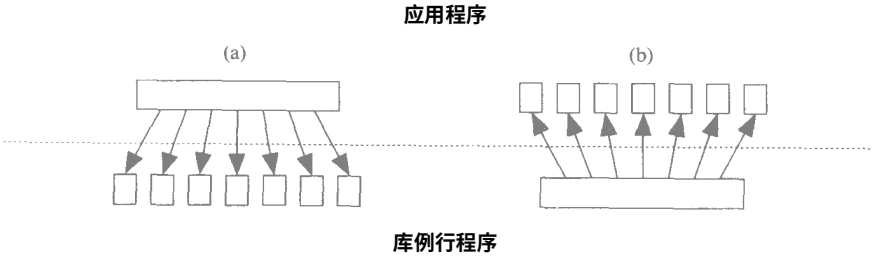

图1-2 编程的结构化形式（a）和回调形式（b）

面向对象框架通常由组件集合构成，这些组件和扩展接口之间存在预定义的协作。我们称预定扩展的点为热点（ Pree，1995 ）或变化点。图 1-3用模式性（schematic）的运行时快照（不是 UML ）显示了带有两个灰度变化点的框架特征。

框架的适配通过插入特定对象而不是占位符。连接方法接口的箭头表示具体和抽象组件之间的交互。包含预装配对象的框表示整体可作为一个参数化的组件。

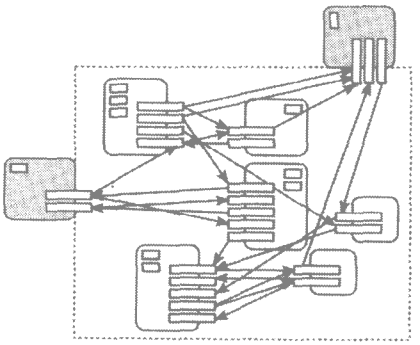

图1-3 具有两个变化点的框架的示意性运行时快照

如果框架提供特定领域变化点以便通过适配获得所需灵活性，那么框架应该有良好设计的属性。良好设计的框架也预先定义了整个体系结构中的大部分，即其组件的组合和交互。构造于框架之上的应用不仅复用代码，而且复用体系结构设计，我们认为这是框架非常重要的特点。

1.2.1 框架的白箱组件

框架不同于普通的类库，因为它预定义了对一些组件之间的交互进行建模的体系结构。然而，框架的类所形成的类层次与单个组件的类层次并没有什么不同。它只是可能含有更高级的抽象类和／或接口。

框架的白箱组件是不完整类，即包含方法的类不含有有意义的默认实现。图 1-4 中所描绘的示例框架类层次中的类 Promotion 表明这个特点。我们假设该框架支持基于 Web 的购物应用的构造。 Promotion 的子类定义了客户接收某

些优惠如折扣或礼品券的规则。内含对角线的方法框表示抽象方法⊖。类 Promotion 具有一个抽象方法，它必须在子类中被覆盖。

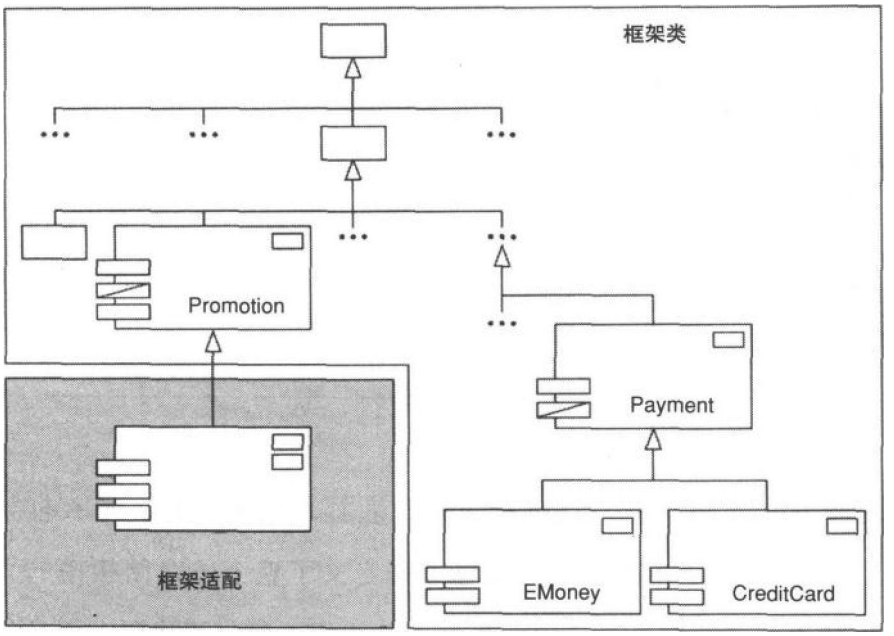

图1-4 框架类层次示例

程序员通过对框架中被其他方法所调用的方法进行覆盖以适配框架。覆盖方法的需要意味着程序员必须对框架设计和实现有一定程度的了解。这使得变化点的文档对框架的成功至关重要。

1.2.2 框架的黑箱组件

黑箱组件是框架适配的成品组件。修改通过组合和参数定义完成，而不是通过传统编程。

⊖ 该符号在第3章介绍。

图1-4中的框架类层次当前提供抽象类 Payment的两个子类EMoney 和 Credit Card，它们提供 Payment 抽象方法的默认实现。假设框架组件的交互如图1-5（a）所描绘，适配插入类 FreqShopping 和 EMoney 的实例中，如图1-5（b）所示。在类 Payment 的案例中，框架提供合适、易用的子类，在类 Promotion 案例中，必须先创建合适的子类。由于类 EMoney 和类 Credit Card 可以随便使用，它们表示黑箱组件。

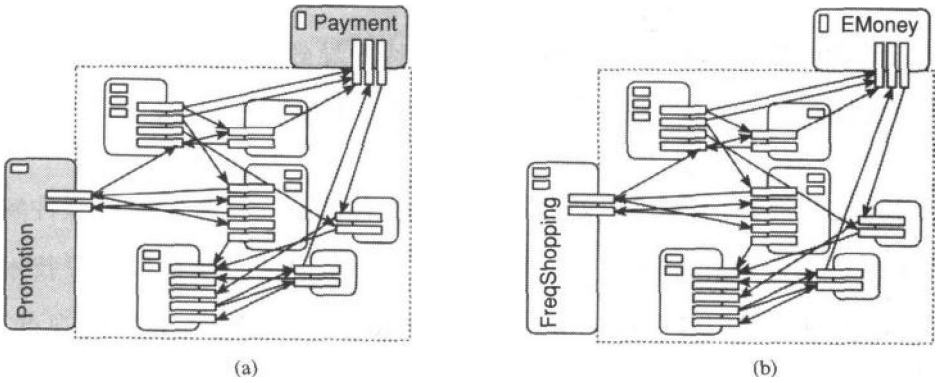

图1-5 通过组合特化之前的框架（a）和之后的框架（b）

该示例说明了框架既不仅仅提供白箱组件，也不只提供黑箱组件。如果框架高度复用，大量的适配将表明可能提供哪些默认黑箱，而不是仅提供一个白箱接口。所以随着框架逐渐成熟，它们可能提供越来越多的黑箱组件。

1.3 框架的优点和缺点

框架有许多优点。除了体系结构设计的复用等同于标准化应用结构这一事实外，适配框架以便生成特定的应用，意味着源代码的数量将明显减少，这些源代码必须由应用程序员来编写。与通过使用传统函数库编写的软件相比，成熟框架

的代码量减少可高达90%（ Weinand等，1989；Fayad 等，1999a，b，c）。

更好的消息是以框架为中心的软件开发并不限于特定领域，如图形用户界面（ GUI）。实际上，框架几乎适宜于所有的商业或技术领域。例如，过程控制系统、基于 Web 软件的商务系统、预定系统以及银行软件。

坏消息是框架开发需要付出更多的开发精力。开发框架比开发特定应用的花费高出很多。如果类似的应用已经存在，它们必须被仔细研究以提出通用的半成品系统，即特定领域的框架。最后，框架的适配导致迭代重设计。仅当类似应用在某一领域重复开发时，此时框架作为一项长期投资才能获得回报。

框架开发和复用同当前的项目文化并不一致，它试图优化特定领域软件解决方案的开发而不是通用软件解决方案的开发。由于本书着重于框架开发技术方面的讨论，以上讨论我们参考由 Goldberg 和 Rubin（1995）所发表的观点。但是，从一些框架所选取的提示和指导能支持框架开发和适配过程。第 7章通过提出在实践中证明有用的并和 UML-F 档案相关的提示和指导而完成本书。

将 UML-F 用作支持组件开发和适配的方法

学习和理解框架会耗费大量精力。如果不了解框架提供的功能和交互基础，应用开发人员既不会使用白箱框架，也不会使用黑箱框架。这表明不仅提供一个高质量的框架很重要，而且提供简单清楚的学习方法更为重要。标准 UML 并没有提供为此目的而裁减的符号语言。作为 UML-F 档案而归纳的扩展支持框架开发和适配。UML-F 档案着重于指明框架的变化点。

所以称为“ 食谱 ”文档框架并支持其适配。它们包含一组配方，这些配方

通过源代码样例为特定的适配提供逐步的指导。第5章讨论“食谱”如何从UML-F档案中受益。

1.4 UML-F 档案的目的

框架体系结构的 UML-F 档案的目的在于定义 UML 子集，通过一些同 UML 兼容的扩展而得到丰富，这些扩展允许对这种制品的注释。因此，称为 UML-F 的档案并不对应于特定领域，而对应于框架技术。本书所表示的 UML-F 档案寻求以下目标：

- UML-F 提供符号元素以准确注释和记录那些众所周知的设计模式。当前 UML 对此的支持极为有限。
- UML-F 本身就是框架的灵魂，它清楚直接的扩展关键在于为任何框架模式编制文档，包括那些将来可能产生的，提供了适宜方法。
- UML-F 包含精简的、易于记忆的符号元素集合。
- UML-F 构造于 UML 标准之上，即扩展应该定义于现存 UML 扩展机制的基础之上。
- 符号元素适宜集成到 UML 工作环境中。例如，工具应该可以创建被注释的框架模式和对应的在线模式文档之间的超链接。

更多的档案将在以后由 OMG 标准化，来自于不同社区的合理提议将开始定义和标准化 UML 档案过程。从这点来讲，本书为框架体系结构的 UML 档案打下了基础。

第2章 框架文档的 UML 要素

许多面向对象的建模语言，诸如由 Rumbaugh 等（1994）、Coleman 等（1994）、Booch（1994）和Jacobson（1993）所描述的语言，都是在20世纪90年代的早期定义的。经过艰苦卓绝的努力，这些建模技术最好的一些特性被融合到由 OMG 发布的 UML[⊖]中。而且 CASE[⊜]工具提供商也已经采用 UML 作为它们的标准符号。

本章介绍 UML 图的符号元素：类、对象和顺序图，它们将会在全书中使用。我们不会讨论这些图的所有特征，而主要讨论那些对于框架开发和适配特别重要的特性上。因此，本章至今为止并不打算介绍另外一种 UML，相反，它重点放在那些已被证实对于框架开发和适配有用的 UML 符号元素的子集。通常，这样一组符号元素对于建模框架已经足够，但是这并不意味其他的 UML 元素就不会被使用。特别地，包图在 UML 工具语境中对于将类模型构造成子系统，进而构造成子图是非常有用的。如果需要 UML 详细完整的概述，你可以参考由 OMG（2001）发布的技术建议，或者由 D'Souza 和 Wills（1998）、Booch 等（1998）和 Rumbaugh 等（1998）出版的书籍。在由 Bézivin 和 Muller（1998）、France 和 Rumpe（1999）和 Evans 等（2000）出版的 UML 会议系列中，涉及到了大量详细成熟的技术和方法的论点。

⊖ UML 1.4 标准是当前（2001年9月）一份最终草案，主要是对它前一版本 UML规范 1.3的巩固。

⊜ CASE是计算机辅助软件工程（Computer Aided Software Engineering ）的缩写。它的基本思想是使用精致的工具来辅助开发过程。到目前为止，许多CASE 工具已经使用了它们自己的符号。

2.1 UML 概述

如今，软件系统就像有机物一样太过复杂而无法仅用单个的蓝图来描述。相反，存在需要许多的描述，其中的每个描述就可以聚焦于系统的不同方面。与建筑物体系结构类似，软件系统的核心结构是系统所有其他方面构造所依据的基础。然而建筑物是静态制品，软件系统主要是从它们本身的行为而产生价值。如果情况更加复杂，软件系统的架构不是具体的、可以触摸的事物，这使得它的行为更加难以描述和掌握。

UML 提供的各种符号，每一个符号都是着重于软件系统一个方面而设计的。软件的静态结构是通过 UML 最重要的部分类图来描述的。类图提供了整个系统的概述，它捕获结构以及组件之间的关系。对象图与类图紧密联系，它们对于描述系统快照是非常有效和重要的技术。

UML 提供了一些符号描述行为的不同方面。顺序图是其中最重要的，因为它们捕获对象之间的交互，描述几个对象如何通过发送和接收消息而相互之间产生交互。它们通常不考虑参与对象的状态。

协作图提供几乎相同的信息，但是使用的方法不同。根据我们的经验，协作图通常要比顺序图更加难于阅读，协作图更加难于理解整个的消息流。此外，协作图更不容易操纵，例如，直接添加新的参与对象、过滤不相关的消息或者整合两张图。因此在本书中没有使用协作图，并且我们也不会遵循 UML 文档中建议的为展示模式而特别地使用这些图的指导。

状态图描述了单一对象的生命周期：状态图[⊖] 结合对象状态的信息，包括

⊖ 首先由 David Harel 提出（Harel，1987）。

它的行为信息。其他种类的图聚焦于工作流（活动图）、软件组件和它们在几个计算机或节点上的物理分布（组件图和配置图）以及描述用户如何理解系统（用例图）。本书没有使用任何一种这些类型的图。

尽管为软件系统的不同方面提供许多符号的 UML 是一种复杂的语言，但是这些符号没有必要紧密地耦合。这是个问题，因为它使得一些符号在开发过程中的一致和系统使用变得更加困难。但同时它也提供了机会，因为它允许聚焦于 UML 的子集。与使用自然语言一样，没有必要为了沟通而去理解和使用完整的 UML 语言。相反，UML 子集（或者 UML 档案）可以聚焦在使用语境元素上的基本元素。在本书中，焦点是框架相关方面的描述。我们识别类、对象和顺序图的子集作为我们的基本语言。这些图涉及一般层次（类图）和实例层次（对象图）上的结构以及对象之间潜在的交互（顺序图）。以下小节介绍了这些图的相关特性。

2.2 类图

类图是最重要的对象建模符号。类图有着几个重要的用途：

- 类图提供系统类型结构的概述。
- 在类图中显示的关联和泛化描述了类之间的结构关系。
- 类图提供了系统体系结构的描述，因为组件是由（一组）类实现的。
- 几种其他种类的 UML 图，诸如顺序图和对象图，都是以类图为基础的。
- 类图描述了一组整个系统可能的状态，因此类图可以作为系统可能状态的约束。
- 附属注解包含类图元素的约束或者实现体，允许我们在一定程度上描述

行为——注解也可以用于描述附加的信息，诸如最新改动的时间、评审的状态等。

图2-1显示了一个简单的 UML 类图，它包含了 UML 标准所提供的所有元素。类图由六个类（Human、Child、Employee、House、Roof 和 Door）以及两个接口（Mammal 和 Thing）构成。接口以不同的形状表示， Thing 被描绘成具有构造型《interface》的类，但是 Mammal 被描绘成没有任何接口方法指示的小圆圈。

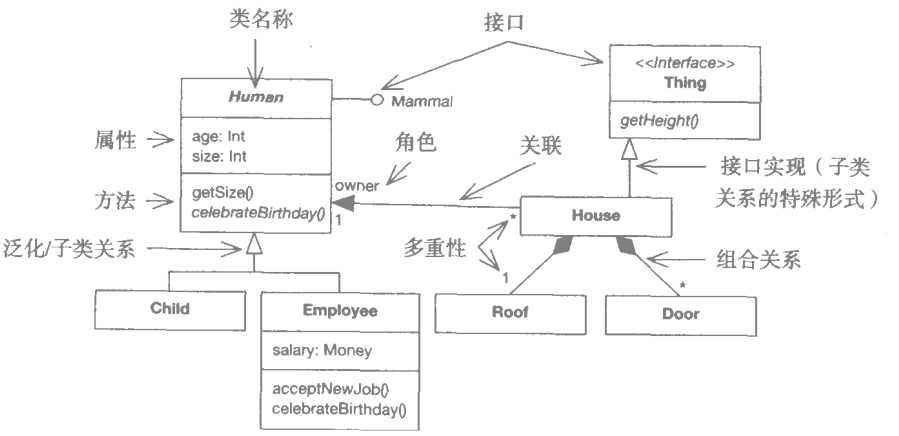

图2-1 一个简单的类图

类 House 实现 Thing 接口。类 Child 和 Employee 是 Human 的子类——泛化关系以空心三角形箭头的形式表示。泛化是子类和超类之间的关系。 Java 中的子类只有一个直接的超类， UML 标准允许类从任意数目的超类继承。但是在 Java 中每个类可以实现任意数量的接口。当沿着不同的路径数次继承相同的属性或方法时，这就足够允许普通的类型层次，但同时避免技术混乱。

此外，类和接口之间的实现关系与类之间的泛化关系非常地相似。两者都

是以相同的方法使用实心三角形来可视化的[㊀]。抽象类用斜体的类名称来描绘例如，类 Human。

类 Human 和 House 之间没有命名的关联表示所有权关系，有一个角色和两个多重性标识附属于关联。角色名 owner 能够用于从类 House 的对象到相应的 Human 对象（它的所有者）导航。用实心的箭头显示关联在这个方向上的导航性。多重性 1 表示恰好只存在一个这样的对象。注意：关联可以有许多不同的实现，但是在许多情况下使用属性[㊁] 作为实现是一个比较好的选择。在另外一个方向上的多重性 * 表明许多类 House 的对象可以属于一个 Human 对象。但是，因为没有箭头和它联系在一起，所以这个方向上的导航是否可能，是不清楚的[㊂]。

类和接口都有特征标记。接口特征标记由方法构成，而类的特征标记是由方法和属性构成。每个方法通过它的名称、参数类型和返回类型来描述。属性特征标记由名称和类型构成。UML 把方法称为操作，在本书余下的部分我们将交替地使用术语“方法”和“操作”[㊃]。

图2-1中的类图的大部分类省略了方法和属性特征标记列表，除了类 Human 和 Employee 以及接口 Thing 具有特征标记。因为类 Employee 从类 Human 继承，所以类 Employee 自动继承了类 Human 的特征标记，特征标记通过新定

㊀ UML标准1.4建议了用虚线表示接口实现，但是与继承的这个区别并没有什么必要，因为从语境就可以清楚地区别它们。我们提出在UML-F中用这个简化形式来表示它，从概念的角度，接口和类仅仅是相互间的变体（它们的能力有所不同）。

㊁ 属性是UML对实例变量的行话，在本书中，两者可以同义地使用。

㊂ 我们后面会开发一个概念可以允许我们来区分信息缺失（例如，通过默认）以及显式地确定没有导航性。这个概念通用于许多情况，并且很好地扩充了UML标准。

㊃ 一般而言，操作的概念包含方法实现的概念。特殊情况，在早期开发阶段确定的操作没有必要在实现阶段变成方法。