Sun公司核心技术丛书

HZ BOOKS

最佳UML入门图书!

Enterprise Java with UML

中文版

(美) C.T.Arrington 著
马波 李雄锋 译

机械工业出版社
China Machine Press

附赠 CD-ROM

Sun公司核心技术丛书

Enterprise Java with UML

中文版

（美） CT Arrington 著
马 波 李雄锋 译

本书配有光盘，需要的读者请到 http://210.34.51.1/tractate/index.asp
网页上申请，或到“网络与光盘检索实验室”联系。

机械工业出版社
China Machine Press

本书是第一本全面介绍用 UML 对 Java 应用程序进行建模的指南。作者通过具体的开发实例深入浅出地介绍了用 UML 这个建模工具开发面向对象系统的方法，对不同解决方案的优缺点进行比较，分析在开发过程中开发团队所碰到的各种常见问题。在介绍用建模工具开发系统时，本书还提供了开发企业级应用系统的策略以及相关技术，包括 XML、servlet、Enterprise JavaBeans、Swing Components、CORBA 及 RMI 等，并讨论如何在不同的技术组合中做出权衡、如何与相关的 Java 技术相结合进行系统开发。

本书内容翔实，讲解透彻。通过本书，读者可以对 UML 建模技术以及面向对象的分析和设计有一个全面而深入的认识和了解。

随书附带的光盘中包含书中的示例代码。

CT Arrington: Enterprise Java with UML (ISBN:0-471 - 38680-4).
Authorized translation from the English language edition published by John Wiley & Sons, Inc.

本书版权登记号：图字：01-2002-3599

图书在版编目（CIP）数据

Enterprise Java with UML 中文版/（美）阿林顿（Arrington, C）著；马波，李雄锋译．- 北京：机械工业出版社，2003 7
（Sun 公司核心技术丛书）
书名原文：Enterprise Java with UML
ISBN 7-111-12246-1

Ⅰ.E… Ⅱ.①阿… ②马… ③李… Ⅲ.①Java 语言 - 程序设计②面向对象语言，UML - 程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 039044 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）
责任编辑：贾 梅
北京忠信诚胶印厂印刷·新华书店北京发行所发行
2003 年 7 月第 1 版第 1 次印刷
787mm × 1092mm 1/16·26.75 印张
印数：0 001-4 000 册
定价：49.00 元（附光盘）

译 者 序

一个软件工程师的成长之路，拙见以为，简单的说可以是：从对数据结构和算法的认识为开始，对几门语言的熟练掌握为发展，对软件工程过程的理解为关键，对系统构架的认识和运用为目标。期间实现“质”的飞跃又在第三步，它是从 coder 提升到 developer 的必经之路。而现代软件工程的一个核心内容就是对开发全过程建模：从需求到分析，从设计到实现再到部署。对软件系统建模，目前来看，也就是对 UML 的理解和熟练运用。这必须是踏踏实实走好、不能也不可能绕开的一步。

同样，对任何一个企业、组织来讲，科学、严格的建模过程和基于模型的交流协作也是软件开发中非常重要的一环。任何不进行建模就着手开发的企图，只能以拙劣的设计，并由此导致不能满足用户需求，不可复用以及难以维护而收场。

统一建模语言（UML）是对面向对象系统建模的开发标准。对于一个软件开发人员而言，用 UML 进行系统设计，用 UML 与同事进行交流，并能够从 UML 模型中建立起系统。但是 UML 体系庞大，内容繁杂，学习使用都有一定的难度。

像众多渴望成长的软件工程师一样，译者在刚接触到 UML 时，也是兴奋加忙乱，逮住三位大师的三卷本一阵猛啃，但是却一直不得要领，其他许多这方面的书籍也大多隔靴搔痒，直到遇到 Martin Fowler 的《UML Distilled》和本书。Fowler 大师的书简洁、清晰，读之有醍醐灌顶、豁然开朗之感，但也正因为其简洁，适于反复揣摩、参考，而不适合做学习 UML 之样例使用。相比之下，本书内容翔实，循循善诱，著者思路清晰，讲解透彻，且耐心之至——从著者的“致谢”中就可以看出：“……我无法表达对我的家庭的感谢，感谢他们原谅我这种极端自私的行为（废寝忘食的著述——译者注）。我占用了多少个夜晚和周末？又有多少个早晨，我睡眼惺忪地醒来，因为睡眠不足和进度缓慢而脾气暴躁？……”对著者的孜孜不倦由此可见一斑。

作者具有极其丰富的面向对象设计和实践经验，他坚信：面向对象的分析和设计结合好的软件工程实践能够在健康、稳定的环境中产生优秀的系统。而本书正式对这一理念的最好诠释和示范。

本书的价值在于对整个软件系统开发过程进行了细致入微的分析、讨论，并通过开发一个样例系统的全过程示范了好的实践方式——作者完全从一个实践者的角度对系统开发中可能遇到的问题一一进行解答，从与相关人员的交流到相关技术的评估、选择；从每一张图如何做到每一段描述、说明如何写。从用户需求到最终实现，在整个开发的每一个环节上，作者都安排了“配套”内容：先讲解了 UML（用例、顺序图、类图等等）在此环节上的用途，然后又用考勤卡系统亲自示范相关内容。一路走来我们进行了技术选择（这是多么详细而有用的一个过程!），建立各种 UML 图表，直到完成代码——所有这一切都是有理有据，落到实处的，你在

下一个项目中就可以参考使用！译者就是这样做的，并感觉不但学习了 UML，对软件开发过程也有了新的理解，更重要的是澄清了许多盲点和错误观念，真是受益匪浅。

对于经验丰富的读者，可以关注建立模型的章节，这些章节包含了各种实用技巧和面临各种选择时的评估模板——这在一般的讲述 UML 理论的书籍中是很少见到的；对于初学者，可以详细揣摩随后的样例系统示范部分，体会理论是如何联系到实际的。

毫不夸张地说：本书讲述了每一个开发人员都应该具备的知识。It's a MUST read!

慢慢享受吧。

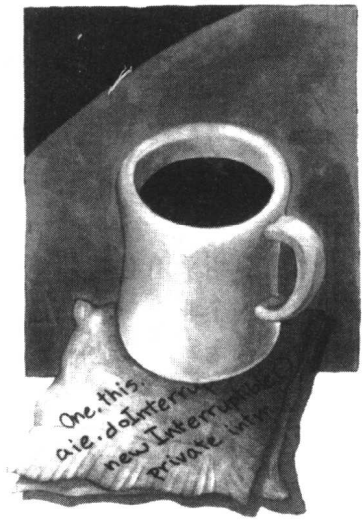
马波　李雄锋

2003 年 4 月

目　录

第 1 章

用UML 对Java 建模导论



当 Java 从一种新奇的语言完全地成长为一种支持网络的企业级计算候选语言之后，Java 开发人员也开始面临着许多的机遇和挑战。现在开发的系统必须随着潜在的商业应用的增长而增多，并能够与 Web 的发展速度并驾齐驱。客户对功能性（functionality）、可伸缩性（scalability）、可用性（usability）、可扩展性（extensibility）以及可靠性（reliability）的要求在逐年地增加。

幸运的是，Java 提供大量的支持使系统开发可以满足这些要求。首先，也可能是最重要的是，Java 是一种小巧紧凑的面向对象的（object-oriented）语言，为异常处理（exception handing）和并发处理提供了卓越的内建支持。而且，由于这种语言运行在一个平台独立（platform-independent）的虚拟机上，这样 Java 系统就可以在大约 12 种左右的操作系统上运行，包括 PalmPilot、网络浏览器，甚至是 AS400。有了这个可靠的基础，Sun 建立并开发了一个类库（class library）。这是最非凡的类库之一，它包含了国际化支持、日历管理、数据库访问、图像处理、联网、用户界面、2D 和 3D 图形等。最后，EJB（Enterprise JavaBeans）和 J2EE（Java 2 Enterprise Edition）提供了真正的跨平台（cross-platform）的企业级计算的规范（specification）。这些规范用一种前所未有的力度来描述数十年来一直困扰企业级开发人员的诸多问题，比如对象到关系的持久性（object-to-relational persistence）、对象高速缓存（object caching）、数据完整性（data integrity）和资源管理（resource management）等。这些规范以及实现这些规范的应用服务器，使我们可以充分利用这些丰富的实践经验和理论研究成果。我们在开发企业级系统方面得到了比以前更好的工具。

但是，强大的工具并不等于成功的保证。开发人员在能够掌握企业级 Java 技术的强大力量之前，需要对问题有一个清楚的认识，并对解决方案要有明确的计划。为了获取这种认识，需要一种途径来对系统进行可视化处理，并与各种用户交流他们的决策和创作。幸运的是，

近几十年来，我们对面向对象系统的理解和建模（model）已经取得了显著的进步。统一建模语言（Unified Modeling Language，UML）是一个开放的标准符号集，它允许开发人员构建软件系统的可视化表示。这些模型使得开发人员能够用它们来设计一流的解决方案，分享想法，并在整个开发周期中进行决策跟踪。而且，支持创建、反向工程（reverse-engineering）和分布式软件模型的UML工具在近两年来也日趋成熟，这使得建模可以无缝地衔接到软件开发生命周期中。

本书介绍采用UML进行软件建模，并向开发人员展示如何在软件开发过程中采用UML来创建更好的企业级Java系统和更有活力的企业级Java项目。本章的剩下部分将更详细地讨论软件建模。作为本书其他部分的基础，这里还将介绍一些面向对象的术语和UML符号。

注意 *本书是在总结我个人作为一个软件开发人员而得到的苦乐相伴的实践经验基础上，为有兴趣在构建软件之前先对它进行建模的Java开发人员提供指导。*

从这本书中，你将学到：

- 与其他人交流对OO建模理论和实践的理解；
- 与其他人交流对UML符号的理解；
- 用鉴定的眼光来评审（review）不同的UML软件模型；
- 从用户的角度（perspective）利用UML来获取对问题的详细的认识；
- 利用UML来可视化（visualize）并记录（document）从一整套Java技术中获取的折衷的解决方案；
- 利用UML来描述其他的技术和类库。

1.1 什么是建模

模型（model）是对事物进行有目的的简化（simplification）。它采用精确定义的符号来描述和简化一个复杂有趣的结构（structure）、现象（phenomenon）或关系（relationship）。我们通过建模来认识和控制周围的世界，避免被事物的复杂性所淹没。让我们考虑一些现实的例子。太阳系的数学模型使得像我们这样的凡夫俗子也可以计算出行星的位置，工程师们采用高级的建模技术来设计从航母到电路板等各种东西，而气象学家则利用数学模型来做天气预报。

软件系统的模型可以协助开发人员在进行大规模的投资之前审视、交流并校验系统。软件模型也可以帮助一个软件开发小组组织和协调他们的工作。以下部分，我们将介绍模型的一些特点和它们在软件开发上的作用。

1.1.1 简化

与组成最终系统的代码和组件相比，系统的模型显得简单得多，也更容易理解。对开发者

来说，构建、扩展和评估一个可视化模型比直接操作代码容易得多。想一想那些你在编码时所要做出的所有决定。在每一次编码时，你要决定哪些参数需要传递、需要采用哪些类型的返回值以及在什么地方设置某种功能等等许多其他的问题。一旦在编码的时候做出了这些决定，以后它们将保持不变。但是通过建模，特别是在一些可视化的建模工具的帮助下，你可以快速而有效地做出并修订这些决定。软件模型扮演着与艺术家的草图相似的角色，它提供一种快速而又相对简单的途径来帮助我们获取对实际解决方案的感性认识。

模型所固有的简单性使它们成为协作和评审的完美机制。涉及多个开发人员的编码过程是一件相当麻烦的事。在面对无所不在的进度压力时，我们需要为常规代码评审制定大量的规程（discipline）。我们可以对软件模型的某个特定部分的质量、是否易懂以及与模型的其他部分是否一致进行评审。而评审一个模型的准备时间比走审（walkthrough）一份相当的代码所需要的时间要少得多。一个熟练的开发者可以在一天之内完全了解某个子系统（subsystem）的全部详细模型。而对于同一个子系统，要完全了解它的实际代码动辄就要花上数周的时间。更多的开发者可以合作来评审整个模型的更多部分。总之，对软件模型的协作和评审可以降低出错的概率,减少整合的难度。而且，利用软件模型可以大量地减少花费在理解和评审代码上的时间。

1.1.2 不同的视角

一个软件系统的模型可以从不同的视角来描述系统。其中一个视角可能显示系统主要部件的交互（interact）和协作（cooperate）。另一个视角则可能对系统的某个特定部分的细节进行放大。还可能有一个视角，它是从用户的角度来描述系统的。通过高级的视图提供的上下文和导航功能，这些不同的视图可以让开发者把握系统的复杂性。一旦开发人员找到一个感兴趣的领域，她或他就可以通过放大该领域的细节来理解它。新来的开发人员在了解整个系统的过程中会发现这种方法特别有用。

我们在现实的世界中也采用这种方法。街道地图就是一个例子。它对城市的建筑物和街道进行建模。地图的一部分可能展示整个城市的主要的公路和干道，另一部分则对市区的某部分进行放大，详细地展示每一条街道。这些视图在不同的方面都是正确的，而且都很有用。

1.1.3 通用符号

通用符号（common notation）可以让开发人员把注意力放在认识解决方案的优点上而不是争论这个方案的含义上。当然，这有个前提，即对通用符号的用法和理解要相一致。很多其他的学科也采用通用符号来帮助进行交流。有经验的音乐家从不争论他们的音符的含义。他们利用这些符号来精确地描述声音，这样他们就无需再对某个音符所标识的音律进行讨论。

采用通用符号的精确的软件模型可以让软件开发人员并行地工作并合并他们的成果。只要每个人的工作都符合这个模型，他的工作就可以被整合到最终的系统中。现代的制造业采用这种技术来降低成本和减少产品周期。根据车辆的设计图，汽车厂商可以从上百个零件供应商

中采购零部件。只要这些零部件符合设计模型描述的规范，它就可以完美地结合到产品中去。

1.2 UML

统一建模语言（UML）是一种用来规范、可视化、构造和记录软件系统制品的语言。UML为我们提供软件系统建模所需要的精确的符号。很重要的一点是，UML绝不仅仅是记录已有想法的一种途径，它还帮助软件开发人员创造并交流想法。

UML发明的本意并不是为面向对象的软件系统进行建模提供符号语言。实际上，UML的发明只是为了结束相互竞争的符号带来的混乱。在20世纪90年代中期和后期，面向对象软件开发领域的许多最好的、最聪明的学者和从业者联合起来创造了一种通用的符号。现在它已经成为面向对象系统建模的国际标准。

UML是由OMG（Object Management Group，对象管理组织），而不是任何公司或个人控制的一个开放式的标准。这本书讨论的是当前的1.3版本的UML。UML的下一个版本，也就是2.0版，预计在2002年发布。

基础知识

在深入研究如何利用UML对系统进行建模之前，你需要理解一些面向对象相关的概念。

1. 抽象

抽象（*abstraction*）是对复杂的概念、过程或现实世界的对象的简化或模型。抽象和我们的生活息息相关。抽象让我们从简单开始了解这个世界，而不会被淹没在世界的复杂性之中。你会对个人电脑、电视机、CD播放器甚至十分简单的晶体管收音机的每一部分都了如指掌吗？会有人既通晓这些电子设备又洞悉细胞生物学和人类生理学的秘密吗？会有人清楚地知道开采矿石或进行专业足球赛等活动时人的施力情况吗？

抽象是一种简化，或者说是思维模型，它帮助人们在某个合适的层次上认识事物。这意味着，不同的人对同一个概念会建立完全不同的抽象。例如，我家的冰箱在我看来就是一个有门的大盒子，里面放着一些食品，还有一个小刻度轮让我可以知道内部的温度。而一个设计工程师则把它看做一个复杂的系统，它有一个蒸发器风扇、蒸发器、除霜器、压缩机以及冷凝器风扇，它们将热量从冰箱里传送到我的厨房中去。设计工程师需要这些丰富的视图来设计一个高效节能的电冰箱。而我就不管这么多了，只要它能提供一杯冰镇苏打水就行了。

对于那些整体上十分复杂、难以理解的事物，一个好的抽象应该能够突出有关的特性和行为。这个抽象的创建者的需求和兴趣决定了它的详细程度和重点所在。

抽象还可以帮助我们了解一个大模型中的不同部件间是如何交互的。在面向对象的世界中，这些交互的部件被称之为对象（*object*）。

2. 封装

在我那本尘封已久的韦氏字典上，封装（encapsulate）的意思是“to enclose in or as if in a capsule”（包装或者像是被包装在胶囊中）。在面向对象的系统中，数据细节和行为逻辑被隐藏在每一类对象中。封装（encapsulation）可以看成抽象的对立面。抽象突出对象的重要特性，而封装则隐藏它的繁琐的内部细节。我们在构造可重用（reusable）、可扩展（extensible）和可理解（comprehensible）的系统时，封装是一个强大的工具。

首先，将那些讨厌的细节封装在系统的内部会使这个系统更容易理解和重用（reuse）。在很多场合中，其他的开发者可能不关心一个对象是如何工作的，只要它能够提供他想要的功能就行了。在他使用这个对象时需要知道的东西越少，他就越愿意去重用它。简单地说，封装减轻了移植一个类或者类库到新系统中去的负担。

另外，封装还提高了系统的可扩展性。一个封装良好的对象可以让其他的对象使用它而无需考虑任何内部的细节。这样带来的好处是，我们可以根据新的需求来改变对象封装的细节，而不影响使用该对象的代码。

3. 对象

对象（object）是在一个较大的模型中的一个特定的元素。一个对象可能会是非常具体的事物，如在一个汽车经销商的库存系统中的某辆汽车。也可能是看不见的事物，如在银行系统中某个人的账户。也可能只有非常短的生命周期，如在银行系统中的一次交易。

系统中一组相似对象的抽象和对象本身的区别是很重要的。例如，在汽车经销商的库存系统中，汽车的抽象肯定就包括种类、车型、里程、年份、颜色、购买价和状况。而在该库存中的对象则可能是一辆 1996 年产的淡蓝色的本田雅阁，状况良好，里程表上已有54 000英里。

所有的对象都有“状态”（*state*），描述对象的特性和当前的状况。有些特性，如车的种类和车型，是保持不变的。而另一些特性，如车的状态和里程，则随时间改变。

对象还可以有“行为”（*behavior*），定义其他对象可能对该对象施加的动作。例如，银行账户对象会允许客户对象进行取款或存款。客户对象激活取款行为，但是，执行取款的逻辑却在账户对象内部。行为可能会依赖于对象的状态，如一辆没汽油的车不大可能会提供人们想要的行为。

而且，在一个系统中，每个对象必须被惟一地标识，按某些特性或一组特性来区分。以上述的汽车为例，每辆汽车都有惟一的车牌号。

在 UML 中，对象是用一个长方形和带下划线的名字来表示，如图 1-1 所示。

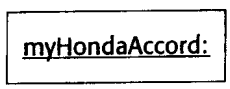


图 1-1 一个车对象

面向对象的系统中的工作是由大量的对象来分工完成。在系统中，每一个对象都被设定了一定的角色。由于每个对象承当的职责集相对来说比较小，要完成一个较大的目标，它们就必须相互协作。比如，一个用户想在 ATM 机上将钱从一个账户转移到另一个账户上，就是这么一个微不足道的例子，也需要一个用户界面对象、客户对象、支票账户对象以及储蓄账户对象。这种严格的分工和协作的组合可以保持这些对象简单易懂。*方法*（*method*）是一个对象向其他对象展示的服务或职责。这样，一个对象就可以调用另一个对象的方法。方法有点像面向过程编程中的函数（function）或子程序（subroutine）。但是，它必须在一个带有自己的状态的特定对象中被调用。这种数据和行为的紧密结合是面向对象软件开发的突出特点之一。

4. 类

类（class）是一组有共性的对象。它描述一个特定的抽象并提供创建对象的模板。类的名称一般都约定俗成地以大写字母打头，并交替使用大小写来标记单词的边界。从同一个类创建的对象有如下相似的地方：

- 对象拥有的数据类型。例如，汽车类可能规定每个汽车对象的颜色、种类和车型都用字符串类型的数据来表示。
- 这个对象可以知道的对象的类型和数量。汽车类可能规定每个汽车对象都可以知道一个或多个以前的车主对象。
- 对象提供的行为逻辑。

数据的真实值是由对象来决定的。这意味着，某辆汽车可能是蓝色的本田雅阁，有一个前车主，而另一辆车可能是绿色的富士传世，有两个前车主。而且，由于行为是状态相关的（state-dependent），因此两个不同的对象对同一个请求会有不同的反应。但如果这两个对象的状态是相同的，那么它们的反应将是一样的。

打个更详细的比方，虽然这个例子很幼稚。在制作玩具士兵时，先要将绿色或棕色的塑料融化，然后将其注入小模子中。模子的形状决定了玩具士兵的高度和外形，以及它是否能够手持一把微型的来福枪，还是背挎一台无线电。购买者无法改变这个玩具的高度，或者为他装备一个喷火器，因为这个类（我指的是模子）不支持这些配置。

然而，购买者还是可以有所作为的。他们可以用或不用来福枪和无线电。他们可以在广场中部署这些玩具来对付那些讨厌的 Ken 洋娃娃。他们在配置这些对象（哦，我说的是那些士兵），并决定它们之间的关联。

这些对象创造了面向对象的系统的应用价值。它们拥有数据，并执行工作。类，跟那个模子一样，对对象的创建十分重要，虽然没有人会拿它出来玩。

在 UML 中，类是用一个长方形来表示。类的名称在长方形的最上面一栏，其下是数据，第三栏是行为。图 1-2 显示一个用 UML 表示的 ToySoldier 类。需要注意的一点是，和对象的 UML 表示不一样，类的名称没有下划线。

5. 对象之间的关系

面向对象的系统中充满着各种不同的对象，它们相互协作完成各种不同的任务。每一个对

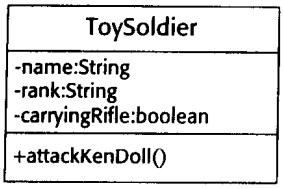


图 1-2 UML 中的 ToySoldier 类

象都有一个精确定义的责任集，所以它们必须一块合作来完成共同的目标。为了协作，对象之间必须存在某种关系，通过这些关系来进行通信。

我们在前面已经说过，对象的状态和行为是由对象的类来决定和约束（constrain）。类控制着对象拥有的状态，提供的行为，以及和它有关系的其他对象。从这里我们就会明白为什么要在类图（class diagram）中定义对象之间的关系。

对象之间有四种关系：

- 依赖（Dependency）
- 关联（Association）
- 聚合（Aggregation）
- 组合（Composition）

（1）依赖

依赖是对象之间最弱的一种关系。一个对象依赖于另一个对象是指这个对象和它之间存在短期的（short-term）关系。在这个短暂的（short-lived）关系中，依赖的对象通过调用被依赖对象的方法来获取它提供的服务，或者以此来配置被依赖的对象。现实生活中充满着依赖关系。我们向杂货店的出纳员购买食物，但我们并没有和那个人建立长期的（long-term）关系。在 UML 中，依赖是用一根带箭头的虚线来表示，箭头指向被依赖的类。

在面向对象的系统中，依赖关系有一些通用的模式（pattern）。作为方法的一个部分，一个对象可能创建另一个对象，让它执行一定的功能，然后就不再管它了。一个对象还可能在一个方法中创建另一个对象，对它进行配置，然后将它作为方法的返回值传给方法的调用者（caller）。一个对象也可以在调用方法的时候接收一个作为参数的对象，使用或修改它，然后在这个方法结束之后就不再理会这个对象。

图 1-3 显示了 Customer 类和 Cashier 类之间的依赖关系。这个依赖关系读作“每个 Customer 对象都依赖于 Cashier 对象”。改变 Cashier 类的接口可能会影响 Customer 类。

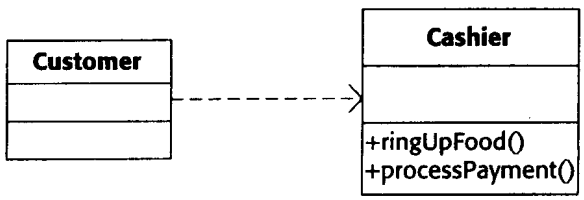


图 1-3 依赖关系的例子

（2）关联

关联是对象之间的长期关系。在关联中，一个对象保存对另一个对象的引用，并在需要的时候调用这个对象的方法。现实生活中也有很多关联的例子。比如人和他的车之间的关系。只要这个人记得他的车放在什么地方，他就可以上车然后开着它到想去的地方。在 UML 中，关联是由两个类之间的实线来表示。

在某些情况中，一个对象可能对另一个对象实例化（instantiate），并保存了对它的引用以备使用。一个对象也可以从配置方法中接收一个作为参数的对象，然后保存对这个对象的引用。

图 1-4 显示一个 Person 类和 Car 类之间的关联关系。这个关系读作“每一个 Person 对象都和不定数目的 Car 对象相关联”，以及“每一个 Car 对象都和不定数目的 Person 对象相关联”。把这个关系想像成一个“知道一”（knows-about-a）的关系可能会更容易理解。比如“每一个 Person 对象都知道一些 Car 对象”。

图 1-4 关联关系的例子

（3）聚合

聚合关系表明一个对象是一个更大的整体的一部分。这个被包含的对象可能会参与多个的聚合关系，并相对于整体而独立存在。例如，一个软件开发者可能会是两个不同的项目组的成员。但是当这两个项目组解散后，这个开发者仍然可以发挥作用。图 1-5 表示这种聚合关系。

图 1-5 聚合关系的例子

在 UML 中，聚合是由关联线上靠近“整体”类的一端加上一个空心的菱形来表示。这个关系读作“每一个 ProjectTeam 对象都有一些 SoftwareDeveloper 对象”，以及“每一个 SoftwareDeveloper 对象可能属于一个或多个 ProjectTeam 对象”。

（4）组合

组合关系表明一个对象是被一个更大的整体所拥有。这个被包含的对象可能不参与更多的组合关系，并且不能独立于这个整体而存在。在整体创建的时候创建这个部分，并在整体销毁的时候销毁。在 UML 中，组合的表示是在关联线上靠近“整体”类的一端加上一个实心的菱形。

来看一个在内燃机油腔中的小齿轮。它是内燃机的一个不可或缺的部分。在内燃机坏掉的时候，将这个齿轮取出来继续使用并不划算。而且，它也不容易更换。图 1-6 表明了这种组合