HZ BOOKS
Mc Graw Hill

面向对象技术实践丛书

Object-Oriented Analysis & Design

面向对象的分析与设计

Andrew Haigh 著　贾爱霞 等译

机械工业出版社
China Machine Press

对象技术实践丛书

Object-Oriented Analysis & Design

面向对象的分析与设计

Andrew Haigh 著 贾爱霞 等译

机械工业出版社
China Machine Press

面向对象是目前最通行的软件设计和编程方法，本书用基于面向对象的概念来讨论软件设计方法。通过阅读本书，读者可以理解面向对象的分析以及利用 UML v1.4 进行设计。本书还为那些已经在从事应用程序开发的程序员提供一些有价值的信息，纠正了一些不正确的观念，有助于他们提高设计水平。

本书讨论的设计方法不仅适用于 C++ 和 Windows 平台，也适用于 Java 和 UNIX 平台。

本书版权登记号：图字：01-2002-0353

图书在版编目（CIP）数据

面向对象的分析与设计/黑格（Haigh, A.）著；贾爱霞等译 . - 北京：机械工业出版社，2003.1

（面向对象技术实践丛书）

书名原文：Object - Oriented Analysis & Design

ISBN 7-111-11379-9

Ⅰ. 面… Ⅱ. ①黑…②贾… Ⅲ. 面向对象语言 - 程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2002）第 102869 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：程 耀 贾 梅

北京第二外国语学院印刷厂印刷·新华书店北京发行所发行

2003 年 1 月第 1 版第 1 次印刷

787mm × 1092mm 1/16 · 21.75 印张

印数：0 001-4000 册

定价：38.00 元

译 者 序

三个月的辛勤工作之后，整本书终于翻译完毕。即将付稿之际，颇有几点感触。

中国有很多程序员，但是软件设计人员相对较少，设计水平高的更是凤毛麟角。有不少从事软件开发的人没有软件设计的概念，遇到一个课题，上来就写代码。甚至一些软件公司的管理层也犯同样的错误，认为代码没写出来，什么都不算。

表面上看起来，有了代码，似乎就有了一切。但是一堆没有经过精心考虑的代码，到后来甚至会编不下去；就算完成了，后期的维护代价也会很高，最后还是要重写，这是非常可怕的重复性劳动。

没有软件设计阶段，软件的文档就不会齐全。目前，软件行业的人员流动非常普遍。如果没有齐全的文档，走一个关键人员就意味着扔掉一个软件，因为后面的人无法全面了解前任的思路。

有了明确的设计，可以在宏观上把握一个软件。虽然代码还没编出来，但我们可以非常清楚地知道软件的工作量、难点。一个好的设计可以将编码过程变成“蓝领”的劳动。同时，通过设计，我们可以提高宏观控制水平，可以积累有复用价值的模块。

总之，我们认为：设计是我们从事软件工作的人必须重视的。

面向对象是目前最流行的软件设计和编程方法。本书是用基于面向对象的概念讨论软件设计的。它适用于有一定面向对象编程或设计经验的读者。如果对面向对象的概念一点都不了解，恐怕阅读起来会比较困难。对于有一定经验的读者，本书有助于纠正一些不正确的观念，提高设计水平。另外，本书讨论的设计方法不仅适用于 C ++ 和 Windows 平台，也适用于 Java 和 Unix 系统。这一点很值得赞赏。

前五章是本书的主要理论，当然需要认真阅读。第 6 章中的测试方法是很多关于编程的书中所没有的，其中几种单元测试的方法比较有用。第 8 章中有关“移植”的内容很值得一读，因为它谈到如何使写出的代码能够方便地移植到另一个平台，这也是高级程序员与初级程序员之间的差别之一。

在我们看来，如果本书能对设计的讨论再深入一些就更好了。

在本书的翻译过程中，得到宋守许老师、朱士兰老师的热情指导，在此，对他们表示真挚的谢意。

王岚老师、秦豫川老师对本书进行了精心的校对，陈瑞先生、江宏先生完成了本书的画图工作，李淼小姐、张玉灵小姐完成了所有代码的录入，张力娟、周枚完成了本书的文字录入，在此一并致谢。

译 者

2002 年 8 月

前　言

当我去对一些刚毕业的学生和有一些工作经验的候选人进行面试时，我自然会问一些我很熟悉的问题。因此，我常问 Java 或 C ++，那正是我起步的地方，而且对大多数人来说并不难。在确信一位候选人是否懂得这些程序设计语言的语法之后，我就将问题转到了面向对象的分析和设计。这时，情况就很不一样了，尽管他们能写一些面向对象的应用程序，但他们不知道如何去设计一个面向对象的应用程序。在有几次的面试中，候选人甚至都没有理解面向对象的概念。

打个比方：你教会了一个人如何用锯子将木材锯开，并且教会他怎样用锤子和钉子将这些材料组合在一起，然后要求他们去盖一所房子。从技术上说，在给他们设计好之后，他们能够建成这所房子。然而，他们没有能力设计房子，或者说他们所建成的房子至少不是那种我愿意住进去的房子。他们也没有能力去检查一所已有的房子并解释它为什么要盖成现在的样子。

同样，大学刚毕业的程序员能够写出一个面向对象的应用程序，但这样的程序很难在任意一种环境下都有良好的表现。我也不会信赖这些程序员，不会拿一个已存在的应用程序去要求他们能理解该程序是如何工作的并能对它做一些改进。

这些不尽人意之处促使我写了这本书。这是一本与面向对象程序设计相关联的书，它提供基本的概念和分析以及设计技术。另外，本书还为那些已经在从事应用程序开发的程序员提供了一些有价值的信息。本书包含一些测试和调试。

关于读者

有许多人试图严格服从做增量开发时所要遵循的规则，但他们却对为什么需要如此长的时间才能写出一个成功的面向对象的应用程序感到疑惑不解，本书就是面向这些读者的。当一个增量被开发并被测试时，总有部分源代码需要被重写，以使应用程序进入设计中的下一个阶段。

当应用程序所需要的分析和设计完成之后，类及它们具有的功能就可以被适当地实现。常见的错误是写了一个类以完成当时所要求的任务，又不得不拆分这个类以满足应用程序在增量开发阶段的要求。

从书中可获得什么

通过阅读本书，可以理解面向对象的分析以及利用 UML v1.4 进行设计。

本书的内容

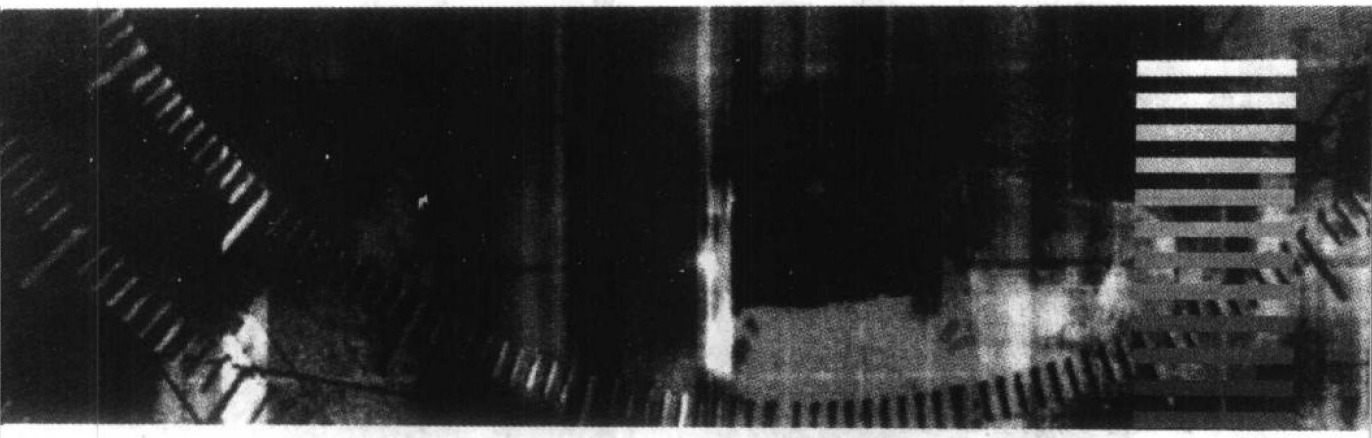
本书分为五个部分：

▶ 第一部分：什么是面向对象

目　录

第四部分 编 程

第五部分 实 例 学 习



第一部分　什么是面向对象

目标：

- ▶ 面向对象简介
- ▶ 学习面向对象的基础知识

第 1 章　面向对象简介

本章将讨论以下内容：

- ▶ 什么是对象
- ▶ 识别对象
- ▶ 类、属性和方法
- ▶ 面向对象的基础知识
- ▶ 将发现的一切形成文档

自从有计算机程序以来，程序员就一直在内存和外存容量的苛刻限制下“艰苦”劳作。尽管如此，程序员还是创造了许多令人惊奇的工程软件。他们能够利用最少的计算机资源来编制大多数的程序软件。在这样的程序中，除了必需的功能模块外，没有什么装饰性的或可有可无的东西。

后来，高性能的计算机越来越普及，它们拥有较多的内外存空间，编程也发展到一个较高的层次，不再对任一细节都斤斤计较。渐渐地，对程序员来说，有一点越来越清晰，那就是：要采用结构化的分析和设计技术来理顺编程中的混乱状况。这些结构化的技术具有革命性的作用，它们允许程序员用形式化的方法来编写应用程序。一个软件计划可顺利地被客户认可并最终被测试验证。遗憾的是，结构化技术引起一个问题，它们不允许最后完成的应用程序具有太大的灵活性，除非考虑到将来需要重写应用程序的重要部分。而面向对象技术就能够提供所需的灵活性。我们可以用家居来做个类比。

一般来说，我们之所以想到买房子，是因为感觉到目前的居室空间不够了。在目前的居室设计下，再也没有足够的空间来满足家庭成员的日常活动了。人们对于房子的需求会发生变化，就像一个企业对现有软件的需求也会改变一样。像对待房子一样，对于不能满足需求的软件，一般有两种解决方法：

- ▶ 你可能摒弃旧软件而购买新软件，当然，这需要再培训。遗憾的是，旧软件没有转卖的价值，另外，编一个新软件一般会比建造一座新房子更费时间。
- ▶ 另一个廉价的选择是在现有软件的基础上做一个扩展件。你可以用最低的培训代价获得新增的功能。

旧软件（或性能不佳、设计不好的软件）就像乡村中的一所旧房子。房子可以“生长”以满足主人不断变化的需求，如在已有房子的基础上增加更多的房间。新的扩展部分可附加在原结构上实现特有的目标。但是，扩展也要有空间才能实现，所以，从一个房间得到另一个房间也成为一种冒险。一些软件与这些房子类似：增加的新特性并不改进设计，而只是增强现有的功能。软件的特色多多，但它如同一场梦魇，任何人都试图改进它的功能和性能。改变软件的功能就像改变旧房子中的房间一样困难，改进的费用可能超过房子本身的价值。

对软件的解决方法要比重新设计改造旧房子（尤其要保持一定的空间次序）容易实施。对软件来说，你只需将错误废弃并从错误中学习，争取下一次创造出更好的软件。

一个现代建筑要考虑到人们如何居住、如何享受建筑提供的功能，那么，采用面向对象理念的程序员会考虑到如何让相关的信息同步变化、提供编程灵活性，如何使设计能够适应可能的变化。尽管结构化的方法论不像面向对象方法那样具有严格的学习曲线，但它不能带来适应现代商业变化的编程灵活性，也不能经受住“重复不断”的变化。

1.1　结构化技术和面向对象技术的比较

对那些使用早期的结构化方法的编程人员来说，掌握面向对象的编程技术可能会有一些精神上的压力。但是面向对象理念的优点远远超过学习过程中的困难。以下是一个结构化设计和一个面向对象设计的对比实例。

面向对象的技术出现以前，一切都是围绕结构化体系模型设计的。这种模型背后的推动力量就是“过程”，它无所不在。这种理念特别流行，因为模型化后的一切就已经是过程了。图 1-1 展示了一个大大简化了的订单处理过程。

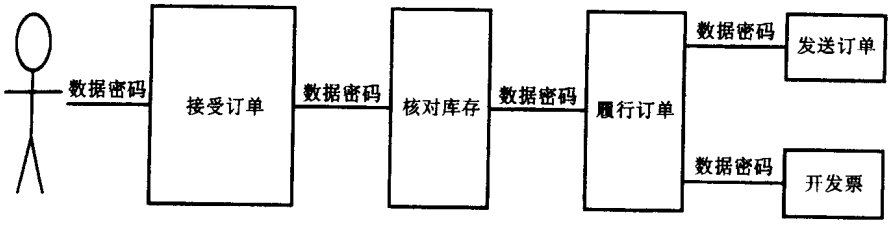


图 1-1　一个订单处理的简单例子

图 1-1 中并没有显示这样的事实：每一个过程都必须与数据库交互。通过使用数据库，每个持有密码的过程都可跟踪客户的订单。通常，对信息仓库的交互接口会提供一整套普遍意义的信息处理方法——例如，setSomething（设置访问信息的密码或某些属性值）。考虑到一些比较特别的问题，最后的接口会比应用程序使用的大得多。

相反，面向对象技术的驱动力是信息——它在系统中到处“流动”。相关的信息都“绑”为“一束”，每个新的订单都是系统中的一“束”新的信息，这就像是在数据库表中增加一行记录。

另外一个变化是，尽管不像外部过程那样易于直接处理信息本身，但每一“束”信息都有一套提供接口的方法，以允许其他“束”按照可控制的方式来处理信息。这如同数据库本身具有处理信息的方法一样，知道如何处理信息。

至此，我们就有了“对象”的基础，即信息束，它包含以可控制的方式处理信息的方法。

二者之间最后一个不同之处在于，如果一个不同类型的订单进入系统，面向对象系统更能适应这种变化。系统只需从现有订单派生出一个新订单，改变其中的数量和类型数据，并根据情况采用适当的方法。这就是面向对象系统的运作方式，图 1-2 展示了在面向对象系统中的订单处理的工作模式。

正如你看出的一样，这就是在系统中“流动”的对象。在此，过程的概念被对象的状态来标识——也就是说，原来那些被描述为“核对”的过程，现在被描述为“一个正在根据自己的

信息执行核对方法的对象”。旧的过程的完成也标志着一个新过程的开始。而在面向对象系统中，一个对象本身可以标明并改变自己的状态，如，从“正在核对的”到“核对完成的”。

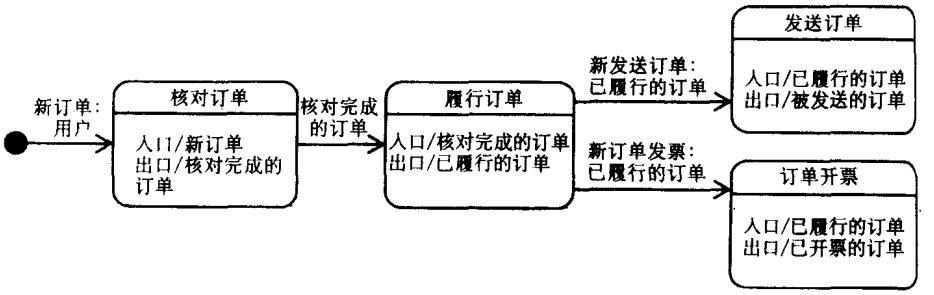


图 1-2　面向对象系统中的订单处理的工作模式

1.2　什么是面向对象

使用面向对象技术可使编程者全面理解问题模型的环境。对问题中的各个组件进行分别标识并细化各组件之间的关系，就可达此目的。

我们以一个空中交通控制系统（一个空中交通管理员将飞机控制权移交给下一段飞行路线的管理员）为例，见图 1-3，用面向对象技术，可以较容易地形成问题的抽象模型：

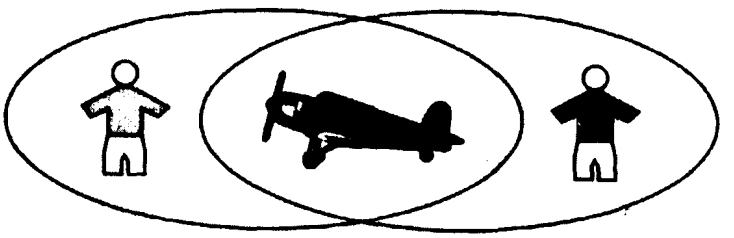


图 1-3　空中交通控制系统

在这个例子中，第一个管理员只需向第二个管理员传达足够的信息，以便他能够确定飞机的位置。所提供的信息必须足以识别飞机（即飞机发出的信号），并要提供通信使用的频率。使用面向对象技术，这些事务场景可以准确地模型化，如图 1-4 所示。

但是，如果第二个管理员需要的只是某一部分信息，如飞行高度和信号，利用结构化技术就能够满足要求，所以，在理想情况下，面向对象技术的优势不见得会显示出来。

现在，请考虑这样一种情况：管理员需要确认全体机组人员的某些细节，如机组人员是否满员。如果整个问题已经使用面向对象技术模型了，那么处理这个需求的功能就被包含在对象（在本例中，对象就是飞机）中。对象就像最初的数据项集，允许实现任何需求。在图 1-5 中，我们在飞机外画一个矩形盒，表明它是一个对象。

使用面向对象技术，飞机对象本身包含机组人员的信息，它自己能够确定目前的机组人员数量是否符合正常情况并做出适当的答复。

对结构化系统来说，传达给空中交通管理员的最初信息只是信息而已。任何待澄清的问题都需要由代表飞机概念的系统来处理，见图 1-6。要回答管理员的问题，系统需要确定飞机的类型，再确定这种飞机需要多少机组人员，最后与目前机上机组人员的实际数量相比较。

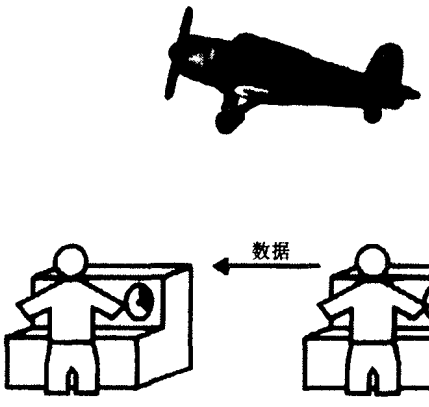


图 1-4　空中交通控制的面向对象技术

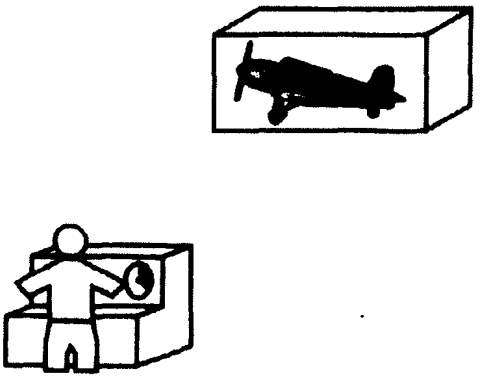


图 1-5　作为对象的飞机

看一看更为复杂的情况（这常常会引起系统的重新设计），如飞机有多种用途，进而可能有多种配置，如货机、客机或外交专机。对于每一种配置，“正常机组人员”的概念都是不同的。

空中交通管理员可以直接询问飞机信息，而不必从一些信息片段去推出另一些信息片段。也就是说，面向对象技术可以提供更符合客观现实的模型，系统中各对象之间的交互活动是符合客观事务的本来面貌的。

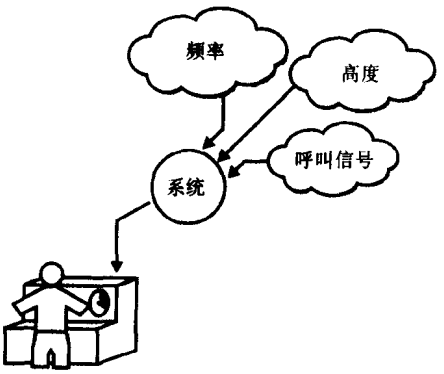


图 1-6　代表飞机概念的系统

1.2.1　面向对象技术是如何与用户关联的

从以上的例子可以看出，对象无所不在。另外还可以看出，对涉及到的人来说，面向对象的理念是自发的、源自天然的。所以，面向对象技术被设计得更遵循思维的自然方式。也就是说，用到的术语、定义、符号应该与每个人相关，包括用户（客户）。

基于面向对象的分析和设计技术的这些特点，用户可以从始至终参与到系统的分析中，包括找出系统中的对象、定义对象等。用户也能参与系统的设计，因为在讨论对象之间的交互作用时，大家都使用同样的术语、概念。用户还可以参加系统文档的创建、编写，因为他们理解已有对象的明确含义。他们还会在已有对象交互作用的层面上理解各个设计阶段，并最终跟踪对象和系统的设计，一直到实现阶段。

1.2.2　面向对象技术的其他优势

使用面向对象技术，还可获得其他一些优势，如下面谈到的源代码复用、源代码的可维护性、创建已有的对象和使用纯粹的面向对象的语言。

1. 复用

一旦系统被完成，不但设计和实现的方法成为一笔财富，创建的对象也将随项目一起继续

生存下来。例如，某项目中开发了一个支持图形接口的库，在未来的系统中，我们就不需要再重新开发这样的接口库，而只需调出以前的接口库就可以了，因为它是可以复用的。

复用是采用面向对象的方法和语言的一个推动力。这种信念逐渐变成了强迫性的，以致于人们经常将复用等同于面向对象。

复用有助于降低未来项目的成本，缩短未来项目的开发时间。从表面上看，复用似乎是一件不太难做的事情，但它常常不能按计划进行，因为几乎没人能写出非常符合未来需要的代码，所以总是需要重做一些工作。

虽然面向对象技术允许软件复用，但使用面向对象技术设计的软件不必非被复用不可。其实，软件的复用早在面向对象技术出现之前就存在了。有些公司向其他公司出售事务处理软件库，这也就是允许软件复用的一种体现。

不管用什么方式看待面向对象技术，复用都不是自动实现的。代码只有满足下面的条件才能被复用：

- ▶ 已有的代码必须有一个可管理的存储库（就像图书馆管理图书的方式一样，要有类似于图书管理员的人员，对提交到库中的代码检查它的可复用性）。
- ▶ 代码必须有完整有效的文档。
- ▶ 代码应该容易使用，也就是说它的对外接口要自然、支持较多的使用方式。比如表达日期的类 date，如果它只支持“MM/DD/YY”这样的日期格式，被复用的价值就不太大；如果它还支持另外三种日期格式“MM/DD/YYYY”、“DD/MM/YY”和“DD/MM/YYYY”，它就更容易被复用。
- ▶ 代码的存储库要采用适当的手段加以宣传，让相关人员易于获取。如在 Internet 上发布，或用 SDK（Software Development Kit）的方式定期更新版本。
- ▶ 代码的存储库要易于访问。不容易访问的库将使用户望而生畏，减少代码复用的机会。
- ▶ 鼓励使用者反馈意见。如果代码不能满足使用者的需求，就会出现反馈意见。可以改进现有代码，也可以再编写新的代码（当然新代码要加入代码存储库），以满足使用者的合理需求。
- ▶ 鼓励使用代码存储库，鼓励向存储库贡献新的思想或代码。

要在两种成本之间掌握好平衡，即自己实现可复用代码并管理存储库的成本与购买同样功能的具有技术支持和使用许可的代码库的成本。

2. 可维护性

维护任何源代码的最大障碍之一是要极力去理解两个问题：每一个组件在做什么、与系统中其他组件有什么关系。

从必要性上讲，前者应该说是一个技术问题。如果假定：因为某人是技术专家，他或她就可以比较容易地理解其他程序员编写的代码，那就太简单化了。比如，举一个例子，他们在代码中看到一点以前从没遇到的细微差别，那就需要花相当时间去理解这些差别可能引起的所有隐含效果。在这种情况下，文档和注释可能就没什么价值了。

问题的第二个方面关心代码与系统的其余部分如何关联。一段代码可能就是一个技术杰作，它可能具有很高的编码质量，有丰富的文档和有用的注释，但如果不能提供它为什么存

在、它能处理哪些事务或者如何使用它等线索，那么，这段代码与一个黑盒子没什么两样。

问题的第二个方面与问题域直接相关。只有充分理解问题域，一段段代码才能被放入系统的关联网中。面向对象应用程序的一个关键特征是可以向一个问题域的专家咨询有关对象之间的关系，因为它们与问题域的实质内容相关。

从以下例子看出，我们需要问题域专家的帮助：

一个程序员小组被要求维护一个国际象棋比赛的程序。这个系统有很多复杂的算法。除此之外，系统中还设计了很多复杂的对弈策略，需要请一位象棋大师才能确认系统是否真的在按设计的方式运行。

3. 在现有对象的基础上构建应用程序

面向对象技术的令人愉悦的一面是一切都是模块化的、有模板的。开始时，先要收集需要努力实现的系统中的各种对象，这些对象相互关联，形成系统的基本架构。接下来，设计过程将构造一个更大的模块——你的应用程序，该模块具有确定的接口，可被更大的应用程序使用。最初的能够实现一些常规处理的简单对象最终突然变成了复杂的应用系统的一部分。

4. 纯语言

分析首先要注重当前系统的需求，但好的设计要能够允许系统“生长”。面向对象的分析和设计技术是概念性的——即，在系统中发现的对象可用多种方式实现。倘若分析和设计完全遵从面向对象的原则，使用者就可以用任何编程语言实现应用程序。显然，使用面向对象的语言更能得到完美的结果，而使用其他语言将使实现过程变得更困难，因为这些语言可能不支持面向对象的某些基本特点。

比如，像 Visual Basic 这样的编程语言是基于对象的，但它不支持面向对象的所有特征。这并不意味着不能使用这种语言，只是在使用这种语言时，设计实际解决方案的过程会更困难。

1.2.3 面向对象技术的一些弱势

没有一种简单的面向对象方法论能满足任一类型的系统和项目的要求，所以，我们的一个诀窍就是以一种方法论为基础，必要时再参考其他可用的技术。虽然本书遵从 UML（通用建模语言，Unified Modeling Language）标准，但也用到了其他的技术，如将在第 2 章谈到的 CRC 卡片（CRC Card）和脚本。

但是，现有的分析和设计工具比较昂贵，而且不允许用户引进另外的表示法和技术。在这种形势下，分析和设计人员常常采用图形程序包或桌面出版程序包，正如我写本书采用的策略。

1.3 什么是对象

对象是一个真实的或抽象的元素项，它包含信息（即描述对象的属性）和用于处理对象的方法。任何对象都可包含其他对象，这些对象又可包含其他对象，直到系统中最基本的对象被揭示出来。

例如，小轿车可被看成一个对象，它包含许多组件，其中之一就是发动机。发动机可被看成一个对象，它也包含其他对象。至于对象要细化到哪一级，则取决于系统的需求。

1.3.1　识别对象

识别对象时，可以采用以下几个有效的技术：

▶ 仔细阅读功能说明书并在所有名词下划线。这有助于用潜在的类覆盖对系统的分析结果。我之所以用“潜在的类”这个词，是因为，比如，若“money”被识别为支付工资的应用程序中的一个对象，这时，用一个数表示“money”更容易。但是，在另外的情形下，“money”可能变成一个需要指定货币单位、不只是一个数量的类。这种“情形”常被称之为问题域，本书将在第 2 章中讨论。

一些必要的表示法的快速入门

每一个对象都有一个类（即对象的定义）与之对应。类将在本章随后的部分中讨论，但现在，我们引入本书中使用的一些表示法。这些方法基于 UML Version 1.4。

▶ UML 中，类的表示法为 ClassName。

▶ UML 中，一个类的非特定的对象表示为:ClassName，其中的冒号“:”用于界定非特定对象。

▶ 一个非特定类的对象表示为ObjectName:，注意，类是非特定的。

▶ 寻找实在的事物即那些与系统中的其他对象交互的事物，如：

▶ 人员（管理者、雇员、家庭、朋友）

▶ 地点（家、工作地点、度假目的地）

▶ 文件（购物单、法律协议、出生证或结婚证）

▶ 寻找对象之间的关系。

1. 实例：寻找一个家庭中的对象

一个:Male（男）对象和一个:Female（女）对象形成一个通常被描述为“婚姻”的关系。婚姻状况可以是:Male对象和:Female对象的属性。:Male对象和:Female对象中包含互相指向对方的句柄，如图 1-7 所示。

图 1-7　一个:Male对象和一个:Female对象的结合

一般需要有一个法定记录来维持婚姻关系。这种情况下，关系“married”（“已婚”）被表示为一个对象:Marriage，如图 1-8 所示。

如果 child（孩子）出生了，那么孩子就与夫妇一起组成了一个 Family（家庭），一种新的关系对象出现了，见图 1-9。

但是上述模型只表示了家庭成员，并没有表明他们与家庭的关系。UML 中有一个用于此目的的表示法，即链，它可用来强调特殊的关系。链的表示方式见图 1-10。

2. 另一个识别对象的实例

下一个例子涉及一组公司职员，其中每一位都扮演公司的不同角色。从分析的角度看，这