第 9 章 面向对象方法学引论

传统的软件工程方法学曾经给软件产业带来巨大进步，部分地缓解了软件危机，使用这种方法学开发的许多中、小规模软件项目都获得了成功。但是，人们也注意到当把这种方法学应用于大型软件产品的开发时，似乎很少取得成功。

在 20 世纪 60 年代后期出现的面向对象编程语言 Simdla_67 中首次引入了类和对象的概念，自 20 世纪 80 年代中期起，人们开始注重面向对象分析和设计的研究，逐步形成了面向对象方法学。到了 20 世纪 90 年代，面向对象方法学已经成为人们在开发软件时首选的范型。面向对象技术已成为当前最好的软件开发技术。

9.1 面向对象方法学概述

9.1.1 面向对象方法学的要点

面向对象方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程，也就是使描述问题的问题空间(也称为问题域)与实现解法的解空间(也称为求解域)在结构上尽可能一致。

客观世界的问题都是由客观世界中的实体及实体相互间的关系构成的。我们把客观世界中的实体抽象为问题域中的对象(object)。因为所要解决的问题具有特殊性，因此，对象是不固定的。一个雇员可以作为一个对象，一家公司也可以作为一个对象，到底应该把什么抽象为对象，由所要解决的问题决定。

从本质上说，用计算机解决客观世界的问题，是借助于某种程序设计语言的规定，对计算机中的实体施加某种处理，并用处理结果去映射解。我们把计算机中的实体称为解空间对象。显然，解空间对象取决于所使用的程序设计语言。例如，汇编语言提供的对象是存储单元；面向过程的高级语言提供的对象，是各种预定义类型的变量、数组、记录和文件等等。一旦提供了某种解空间对象，就隐含规定了允许对该类对象施加的操作。

从动态观点看，对对象施加的操作就是该对象的行为。在问题空间中，对象的行为是极其丰富多彩的，然而解空间中的对象的行为却是非常简单呆板的。因

此，只有借助于十分复杂的算法，才能操纵解空间对象从而得到解。这就是人们常说的“语义断层”，也是长期以来程序设计始终是一门学问的原因。

通常，客观世界中的实体既具有静态的属性又具有动态的行为。然而传统语言提供的解空间对象实质上却仅是描述实体属性的数据，必须在程序中从外部对它施加操作，才能模拟它的行为。

众所周知，软件系统本质上是信息处理系统。数据和处理原本是密切相关的，把数据和处理人为地分离成两个独立的部分，会增加软件开发的难度。与传统方法相反，面向对象方法是一种以数据或信息为主线，把数据和处理相结合的方法。面向对象方法把对象作为由数据及可以施加在这些数据上的操作所构成的统一体。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，相反，它是进行处理的主体。必须发消息请求对象主动地执行它的某些操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。

面向对象方法学所提供的“对象”概念，是让软件开发者自己定义或选取解空间对象，然后把软件系统作为一系列离散的解空间对象的集合。应该使这些解空间对象与问题空间对象尽可能一致。这些解空间对象彼此间通过发送消息而相互作用，从而得出问题的解。也就是说，面向对象方法是一种新的思维方法，它不是把程序看作是工作在数据上的一系列过程或函数的集合，而是把程序看作是相互协作而又彼此独立的对象的集合。每个对象就像一个微型程序，有自己的数据、操作、功能和目的。这样做就向着减少语义断层的方向迈了一大步，在许多系统中解空间对象都可以直接模拟问题空间的对象，解空间与问题空间的结构十分一致，因此，这样的程序易于理解和维护。

概括地说，面向对象方法具有下述4个要点：

(1)认为客观世界是由各种对象组成的，任何事物都是对象，复杂的对象可以由比较简单的对象以某种方式组合而成。按照这种观点，可以认为整个世界就是一个最复杂的对象。因此，面向对象的软件系统是由对象组成的，软件中的任何元素都是对象，复杂的软件对象由比较简单的对象组合而成。

由此可见，面向对象方法用对象分解取代了传统方法的功能分解。

(2)把所有对象都划分成各种对象类(简称为类，class)，每个对象类都定义了一组数据和一组方法。数据用于表示对象的静态属性，是对象的状态信息。因此，每当建立该对象类的一个新实例时，就按照类中对数据的定义为这个新对象生成一组专用的数据，以便描述该对象独特的属性值。例如，荧光屏上不同位置显示的半径不同的几个圆，虽然都是Circle类的对象，但是，各自都有自己专用的数据，以便记录各自的圆心位置、半径等等。

类中定义的方法，是允许施加于该类对象上的操作，是该类所有对象共享的，并不需要为每个对象都复制操作的代码。

(3)按照子类(或称为派生类)与父类(或称为基类)的关系，把若干个对象类组成一个层次结构的系统(也称为类等级)。在这种层次结构中，通常下层的派生类具有和上层的基类相同的特性(包括数据和方法)，这种现象称为继承(inheritance)。但是，如果在派生类中对某些特性又做了重新描述，则在派生类中的这些特性将以新描述为准，也就是说，低层的特性将屏蔽高层的同名特性。

(4)对象彼此之间仅能通过传递消息互相联系。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，相反，它是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，而不能从外界直接对它

的私有数据进行操作。也就是说，一切局部于该对象的私有信息，都被封装在该对象类的定义中，就好像装在一个不透明的黑盒子中一样，在外界是看不见的，更不能直接使用，这就是“封装性”。

综上所述，面向对象的方法学可以用下列方程来概括：

OO=objects+classes+inheritance+communication with messages

也就是说，面向对象就是既使用对象又使用类和继承等机制，而且对象之间仅能通过传递消息实现彼此通信。

如果仅使用对象和消息，则这种方法可以称为基于对象的(object-based)方法，而不能称为面向对象的方法；如果进一步要求把所有对象都划分为类，则这种方法可称为基于类的(class-based)方法，但仍然不是面向对象的方法。只有同时使用对象、类、继承和消息的方法，才是真正面向对象的方法。

9.1.2 面向对象方法学的优点

1.与人类习惯的思维方法一致

传统的程序设计技术是面向过程的设计方法，这种方法以算法为核心，把数据和过程作为相互独立的部分，数据代表问题空间中的客体，程序代码则用于处理这些数据。

把数据和代码作为分离的实体，反映了计算机的观点，因为在计算机内部数据和程序是分开存放的。但是，这样做的时候总存在使用错误的数据调用正确的程序模块，或使用正确的数据调用错误的程序模块的危险。使数据和操作保持一致，是程序员的一个沉重负担，在多人分工合作开发一个大型软件系统的过程中，如果负责设计数据结构的人中途改变了某个数据的结构而又没有及时通知所有人员，则会发生许多不该发生的错误。

传统的程序设计技术忽略了数据和操作之间的内在联系，用这种方法所设计出来的软件系统其解空间与问题空间并不一致，令人感到难于理解。实际上，用计算机解决的问题都是现实世界中的问题，这些问题无非由一些相互间存在一定联系的事物所组成。每个具体的事物都具有行为和属性两方面的特征。因此，把描述事物静态属性的数据结构和表示事物动态行为的操作放在一起构成一个整体，才能完整、自然地表示客观世界中的实体。

面向对象的软件技术以对象(object)为核心，用这种技术开发出的软件系统由对象组成。对象是对现实世界实体的正确抽象，它是由描述内部状态表示静态属性的数据，以及可以对这些数据施加的操作(表示对象的动态行为)，封装在一起所构成的统一体。对象之间通过传递消息互相联系，以模拟现实世界中不同事物彼此之间的联系。

面向对象的设计方法与传统的面向过程的方法有本质不同，这种方法的基本原理是，使用现实世界的概念抽象地思考问题从而自然地解决问题。它强调模拟现实世界中的概念而强调算法，它鼓励开发者在软件开发的绝大部分过程中都用应用领域的概念去思考。在面向对象的设计方法中，计算机的观点是不重要

的，现实世界的模型才是最重要的。面向对象的软件开发过程从始至终都围绕着建立问题领域的对象模型来进行：对问题领域进行自然的分解，确定需要使用的对象和类，建立适当的类等级，在对象之间传递消息实现必要的联系，从而按照人们习惯的思维方式建立起问题领域的模型，模拟客观世界。

传统的软件开发方法可以用“瀑布”模型来描述，这种方法强调自顶向下按部就班地完成软件开发工作。事实上，人们认识客观世界解决现实问题的过程，是一个渐进的过程，人的认识需要在继承以前的有关知识的基础上，经过多次反复才能逐步深化。在人的认识深化过程中，既包括了从一般到特殊的演绎思维过程，也包括了从特殊到一般的归纳思维过程。人在认识和解决复杂问题时使用的最强有力的思维工具是抽象，也就是在处理复杂对象时，为了达到某个分析目的集中研究对象的与此目的有关的实质，忽略该对象的那些与此目的无关的部分。

面向对象方法学的基本原则是按照人类习惯的思维方法建立问题域的模型，开发出尽可能直观、自然地表现求解方法的软件系统。面向对象的软件系统中广泛使用的对象，是对客观世界中实体的抽象。对象实际上是抽象数据类型的实例，提供了比较理想的数据抽象机制，同时又具有良好的过程抽象机制（通过发消息使用公有成员函数）。对象类是对一组相似对象的抽象，类等级中上层的类是对下层类的抽象。因此，面向对象的环境提供了强有力的抽象机制，便于用户在利用计算机软件系统解决复杂问题时使用习惯的抽象思维工具。此外，面向对象方法学中普遍进行的对象分类过程，支持从特殊到一般的归纳思维过程；面向对象方法学中通过建立类等级而获得的继承特性，支持从一般到特殊的演绎思维过程。

面向对象的软件技术为开发者提供了随着对某个应用系统的认识逐步深入和具体化的过程，而逐步设计和实现该系统的可能性，因为可以先设计出由抽象类构成的系统框架，随着认识深入和具体化再逐步派生出更具体的派生类。这样的开发过程符合人们认识客观世界解决复杂问题时逐步深化的渐进过程。

2.稳定性好

传统的软件开发方法以算法为核心，开发过程基于功能分析和功能分解。用传统方法所建立起来的软件系统的结构紧密依赖于系统所要完成的功能，当功能需求发生变化时将引起软件结构的整体修改。事实上，用户需求变化大部分是针对功能的，因此，这样的软件系统是不稳定的。

面向对象方法基于构造问题领域的对象模型，以对象为中心构造软件系统。它的基本作法是用对象模拟问题领域中的实体，以对象间的联系刻画实体间的联系。因为面向对象的软件系统的结构是根据问题领域的模型建立起来的，而不是基于对系统应完成的功能的分解，所以，当对系统的功能需求变化时并不会引起软件结构的整体变化，往往仅需要作一些局部性的修改。例如，从已有类派生出一些新的子类以实现功能扩充或修改，增加或删除某些对象等。总之，由于现实世界中的实体是相对稳定的，因此，以对象为中心构造的软件系统也是比较稳定的。

3.可重用性好

用已有的零部件装配新的产品，是典型的重用技术，例如，可以用已有的预制件建筑
一幢结构和外形都不同于从前的新大楼。重用是提高生产率的最主要的方法。

传统的软件重用技术是利用标准函数库，也就是试图用标准函数库中的函数作为“预
制件”来建造新的软件系统。但是，标准函数缺乏必要的“柔性”，不能适应不同应用场合
的不同需要，并不是理想的可重用的软件成分。实际的库函数往往仅提供最基本、最常用的功能，在开发一个新的软件系统时，通常多数函数是开发者自己编写的，甚至绝大多数函数都是新编的。

使用传统方法学开发软件时，人们认为具有功能内聚性的模块是理想的模块，也就是说，如果一个模块完成一个且只完成一个相对独立的子功能，那么这个模块就是理想的可重用模块。基于这种认识，通常尽量把标准函数库中的函数做成功能内聚的。但是，即使是具有功能内聚性的模块也并不是自含的和独立的，相反，它必须运行在相应的数据结构上。如果要重用这样的模块，则相应的数据也必须重用。如果新产品中的数据与最初产品中的数据不同，则要么修改数据要么修改这个模块。

事实上，离开了操作便无法处理数据，而脱离了数据的操作也是毫无意义的，我们应该对数据和操作同样重视。在面向对象方法所使用的对象中，数据和操作正是作为平等伙伴出现的。因此，对象具有很强的自含性，此外，对象固有的封装性和信息隐藏机制，使得对象的内部实现与外界隔离，具有较强的独立性。由此可见，对象是比较理想的模块和可重用的软件成分。

面向对象的软件技术在利用可重用的软件成分构造新的软件系统时，有很大的灵活性。有两种方法可以重复使用一个对象类：一种方法是创建该类的实例，从而直接使用它；另一种方法是从它派生出一个满足当前需要的新类。继承性机制使得子类不仅可以重用其父类的数据结构和程序代码，而且可以在父类代码的基础上方便地修改和扩充，这种修改并不影响对原有类的使用。由于可以像使用集成电路(IC)构造计算机硬件那样，比较方便地重用对象类来构造软件系统，因此，有人把对象类称为“软件 IC”。

面向对象的软件技术所实现的可重用性是自然的和准确的，在软件重用技术中它是最成功的一个。关于软件重用问题，在第 11.3 节中还要详细讨论。

4.较易开发大型软件产品

在开发大型软件产品时，组织开发人员的方法不恰当往往是出现问题的主要原因。用面向对象方法学开发软件时，构成软件系统的每个对象就像一个微型程序，有自己的数据、操作、功能和用途，因此，可以把一个大型软件产品分解成一系列本质上相互独立的小产品来处理，这就不仅降低了开发的技术难度，

而且也使得对开发工作的管理变得容易多了。这就是为什么对于大型软件产品来说，面向对象范型优于结构化范型的原因之一。许多软件开发公司的经验都表明，当把面向对象方法学用于大型软件的开发时，软件成本明显地降低了，软件的整体质量也提高了。

5.可维护性好

用传统方法和面向过程语言开发出来的软件很难维护，是长期困扰人们的一个严重问题，是软件危机的突出表现。

由于下述因素的存在，使得用面向对象方法所开发的软件可维护性好:

(1)面向对象的软件稳定性比较好。

如前所述，当对软件的功能或性能的要求发生变化时，通常不会引起软件的整体变化，往往只需对局部作一些修改。由于对软件所需做的改动较小且限于局部，自然比较容易实现。

(2)面向对象的软件比较容易修改。

如前所述，类是理想的模块机制，它的独立性好，修改一个类通常很少会牵扯到其他类。如果仅修改一个类的内部实现部分(私有数据成员或成员函数的算法)，而不修改该类的对外接口，则可以完全不影响软件的其他部分。

面向对象软件技术特有的继承机制，使得对软件的修改和扩充比较容易实现，通常只须从已有类派生出一些新类，无须修改软件原有成分。

面向对象软件技术的多态性机制(见 9.2.2 节)，使得当扩充软件功能时对原有代码所需作的修改进一步减少，需要增加的新代码也比较少。

(3)面向对象的软件比较容易理解。

在维护已有软件的时候，首先需要对原有软件与此次修改有关的部分有深入理解，才
能正确地完成维护工作。传统软件之所以难于维护，在很大程度上是因为修改所涉及的部分分散在软件各个地方，需要了解的面很广，内容很多，而且传统软件的解空间与问题空间的结构很不一致，更增加了理解原有软件的难度和工作量。

面向对象的软件技术符合人们习惯的思维方式，用这种方法所建立的软件系统的结构与问题空间的结构基本一致。因此，面向对象的软件系统比较容易理解。

对面向对象软件系统所做的修改和扩充，通常通过在原有类的基础上派生出一些新类来实现。由于对象类有很强的独立性，当派生新类的时候通常不需要详细了解基类中操作的实现算法。因此，了解原有系统的工作量可以大幅度下降。

(4)易于测试和调试。

为了保证软件质量，对软件进行维护之后必须进行必要的测试，以确保要求修改或扩充的功能按照要求正确地实现了，而且没有影响到软件不该修改的部分。如果测试过程中发现了错误，还必须通过调试改正过来。显然，软件是否易于测试和调试，是影响软件可维护性的一个重要因素。

对面向对象的软件进行维护，主要通过从已有类派生出一些新类来实现。因此，维护后的测试和调试工作也主要围绕这些新派生出来的类进行。类是独立性

很强的模块，向类的实例发消息即可运行它，观察它是否能正确地完成要求它做的工作，对类的测试通常比较容易实现，如果发现错误也往往集中在类的内部，比较容易调试。

9.1.3 喷泉模型

迭代是软件开发过程中普遍存在的一种内在属性。经验表明，软件过程各个阶段之间的迭代或一个阶段内各个工作步骤之间的迭代，在面向对象范型中比在结构化范型中更常见。

一般说来，使用面向对象方法学开发软件时，工作重点应该放在生命周期中的分析阶段。这种方法在开发的早期阶段定义了一系列面向问题的对象，并且在整个开发过程中不断充实和扩充这些对象。由于在整个开发过程中都使用统一的软件概念“对象”，所有其他概念(例如功能、关系、事件等)都是围绕对象组成的，目的是保证分析工作中得到的信息不会丢失或改变，因此，对生命周期各阶段的区分自然就不重要、不明显了。分析阶段得到的对象模型也适用于设计阶段和实现阶段。由于各阶段都使用统一的概念和表示符号，因此，整个开发过程都是吻合一致的，或者说是“无缝”连接的，这自然就很容易实现各个开发步骤的多次反复迭代，达到认识的逐步深化。每次反复都会增加或明确一些目标系统的性质，但却不是对先前工作结果的本质性改动，这样就减少了不一致性，降低了出错的可能性。

图 9.1 所示的喷泉模型，是典型的面向对象的软件过程模型。

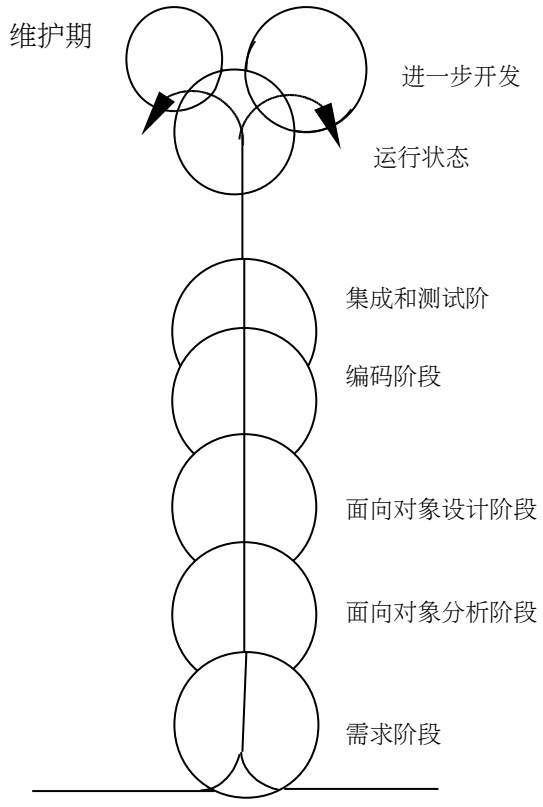

图 9.1　喷泉模型

“喷泉”这个词体现了面向对象软件开发过程迭代和无缝的特性。图中代表不同阶段的圆圈相互重叠，这明确表示两个活动之间存在交迭；而面向对象方法在概念和表示方法上的一致性，保证了在各项开发活动之间的无缝过渡，事实上，用面向对象方法开发软件时，在分析、设计和编码等项开发活动之间并不存在明显的边界。图中在一个阶段内的向下箭头代表该阶段内的迭代(或求精)。图中较小的圆圈代表维护，圆圈较小象征着采用了面向对象范型之后维护时间缩短了。

为避免使用喷泉模型开发软件时开发过程过分无序，应该把一个线性过程(例如，快速原型模型或图 9.1 中的中心垂线)作为总目标。但是，同时也应该记住，面向对象范型本身要求经常对开发活动进行迭代或求精。

9.2 面向对象的概念

“对象”是面向对象方法学中使用的最基本的概念，前面已经多次用到这个概念，本节再从多种角度进一步阐述这个概念，并介绍面向对象的其他基本概念。

9.2.1 对象

在应用领域中有意义的、与所要解决的问题有关系的任何事物都可以作为对象，它既可以是具体的物理实体的抽象，也可以是人为的概念，或者是任何有明确边界和意义的东西。例如，一名职工、一家公司、一个窗口、一座图书馆、一本图书、贷款、借款等等，都可以作为一个对象。总之，对象是对问题域中某个实体的抽象，设立某个对象就反映了软件系统具有保存有关它的信息并且与它进行交互的能力。

由于客观世界中的实体通常都既具有静态的属性，又具有动态的行为，因此，面向对象方法学中的对象是由描述该对象属性的数据以及可以对这些数据施加的所有操作封装在一起构成的统一体。对象可以作的操作表示它的动态行为，在面向对象分析和面向对象设计中，通常把对象的操作称为服务或方法。

1.对象的形象表示

为有助于读者理解对象的概念，图9.2形象地描绘了具有3个操作的对象。

看了图9.2之后，读者可能会联想到一台录音机。确实，可以用一台录音机比喻一个对象，通俗地说明对象的某些特点。

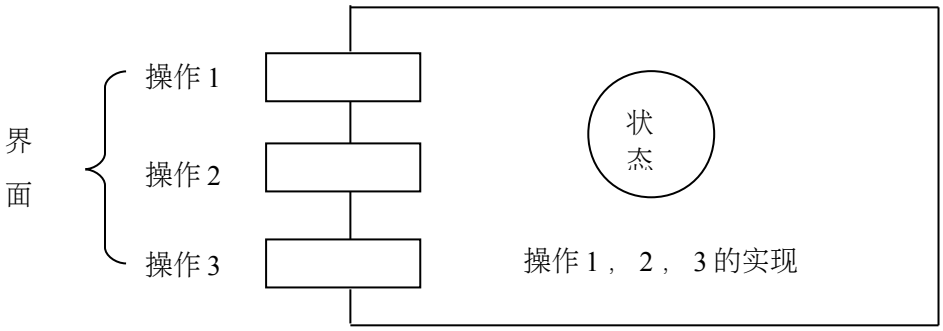

图9.2　对象的形象表示

当使用一台录音机的时候，总是通过按键来操作：按下“Play(放音)”键，则录音带正向转动，通过喇叭放出录音带中记录的歌曲或其他声音；按下“Record(录音)”键，则录音带正向转动，在录音带中录下新的音响……完成录音机各种功能的电子线路被装在录音机的外壳中，人们无须了解这些电子线路的工作原理就可以随心所欲地使用录音机。为了使用录音机根本没有必要打开外壳去触动壳内的各种零部件，事实上，不是专业维修人员的一般用户，完全不允许打开录音机外壳。

一个对象很像一台录音机。当在软件中使用一个对象的时候，只能通过对象与外界的界面来操作它。对象与外界的界面也就是该对象向公众开放的操作，例如，C++语言中对象的公有(public)成员函数。使用对象向公众开放的操作就好像使用录音机的按键，只须知道该操作的名字(好像录音机的按键名)和所需要的参数(提供附加信息或设置状态，例如听录音前先装录音带并把录音带转到指定位置)，根本无须知道实现这些操作的方法。事实上，实现对象操作的代码和数据是隐藏在对象内部的，一个对象好像是一个黑盒子，表示它内部状态的数据和实现各个操作的代码及局部数据，都被封装在这个黑盒子内部，在外面是看不见的，更不能从外面去访问或修改这些数据或代码。

使用对象时只需知道它向外界提供的接口形式而无须知道它的内部实现算法，不仅使得对象的使用变得非常简单、方便，而且具有很高的安全性和可靠性。对象内部的数据只能通过对象的公有方法(如C+十的公有成员函数)来访问或处理，这就保证了对这些数据的访问或处理，在任何时候都是使用统一的方法进行的，不会像使用传统的面向过程的程序设计语言那样，由于每个使用者各自编写自己的处理某个全局数据的过程而发生错误。

此外，录音机中放置的录音带很像一个对象中表示其内部状态的数据，当录音带处于不同位置时按下"P1ay"键所放出的歌曲是不相同的，同样，当对象处于不同状态时，做同一个操作所得到的效果也是不同的。

2.对象的定义

目前，对对象所下的定义并不完全统一，人们从不同角度给出对象的不同定义。这些定义虽然形式不同，但基本含义是相同的。下面给出对象的几个定义。

(1)定义 1：对象是具有相同状态的一组操作的集合。

这个定义主要是从面向对象程序设计的角度看“对象”。

(2)定义 2：对象是对问题域中某个东西的抽象，这种抽象反映了系统保存有关这个东西的信息或与它交互的能力。也就是说，对象是对属性值和操作的封装。

这个定义着重从信息模拟的角度看待“对象”。

(3)定义 3：对象：：=(ID，MS，DS，MI)。其中，ID 是对象的标识或名字 MS 是对象中的操作集合，DS 是对象的数据结构，MI 是对象受理的消息名集合(即对外接口)。

这个定义是一个形式化的定义。

总之，对象是封装了数据结构及可以施加在这些数据结构上的操作的封装体，这个封装体有可以惟一地标识它的名字，而且向外界提供一组服务(即公有的操作)。对象中的数据表示对象的状态，一个对象的状态只能由该对象的操作来改变。每当需要改变对象的状态时，只能由其他对象向该对象发送消息。对象响应消息时，按照消息模式找出与之匹配的方法，并执行该方法。

从动态角度或对象的实现机制来看，对象是一台自动机。具有内部状态 S，操作 f_i(i=1，2，…，n)，且与操作 f_i 对应的状态转换函数为 g_i(I=1，2，…，咒)的一个对象，可以用图 9.3 所示的自动机来模拟。

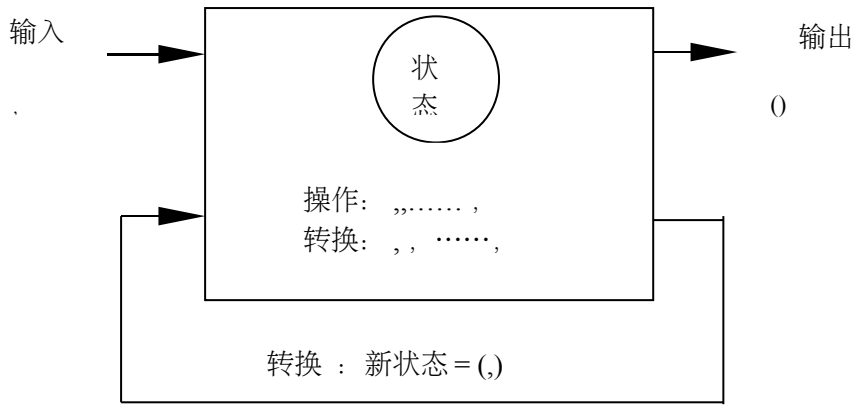

图 9.3 用自动机模拟对象

3.对象的特点

对象有如下一些基本特点：

(1)以数据为中心。操作围绕对其数据所需要做的处理来设置，不设置与这些数据无关的操作，而且操作的结果往往与当时所处的状态(数据的值)有关。

(2)对象是主动的。它与传统的数据有本质不同，不是被动地等待对它进行处理，相反，它是进行处理的主体。为了完成某个操作，不能从外部直接加工它的私有数据，而是必须通过它的公有接口向对象发消息，请求它执行它的某个操作，处理它的私有数据。

(3)实现了数据封装。对象好像是一只黑盒子，它的私有数据完全被封装在盒子内部，对外是隐藏的、不可见的，对私有数据的访问或处理只能通过公有的操作进行。为了使用对象内部的私有数据，只需知道数据的取值范围(值域)和可以对该数据施加的操作(即，对象提供了哪些处理或访问数据的公有方法)，根本无须知道数据的具体结构以及实现操作的算法。这也就是抽象数据类型的概念。因此，一个对象类型也可以看作是一种抽象数据类型。

(4)本质上具有并行性。对象是描述其内部状态的数据及可以对这些数据施加的全部操作的集合。不同对象各自独立地处理自身的数据，彼此通过发消息传递信息完成通信。因此，本质上具有并行工作的属性。

(5)模块独立性好。对象是面向对象的软件的基本模块，为了充分发挥模块化简化开发工作的优点，希望模块的独立性强。具体来说，也就是要求模块的内聚性强，耦合性弱。如前所述，对象是由数据及可以对这些数据施加的操作所组成的统一体，而且对象是以数据为中心的，操作围绕对其数据所需做的处理来设置，没有无关的操作。因此，对象内部各种元素彼此结合得很紧密，内聚性相当强。由于完成对象功能所需要的元素(数据和方法)基本上都被封装在对象内部，它与外界的联系自然就比较少，因此，对象之间的耦合通常比较松。

9.2.2 其他概念

1.类(class)

现实世界中存在的客观事物有些是彼此相似的，例如，张三、李四、王五…

…虽说每个人职业、性格、爱好、特长等等各有不同，但是，他们的基本特征是相似的，都是黄皮肤、黑头发、黑眼睛，于是人们把他们统称为“中国人”。人类习惯于把有相似特征的事物归为一类，分类是人类认识客观世界的基本方法。

在面向对象的软件技术中，“类”就是对具有相同数据和相同操作的一组相似对象的定义，也就是说，类是对具有相同属性和行为的一个或多个对象的描述，通常在这种描述中也包括对怎样创建该类的新对象的说明。

例如，一个面向对象的图形程序在屏幕左下角显示一个半径 3cm 的红颜色的圆，在屏幕中部显示一个半径 4cm 的绿颜色的圆，在屏幕右上角显示一个半径 1cm 的黄颜色的圆。这三个圆心位置、半径大小和颜色均不相同的圆，是三个不同的对象。但是，它们都有相同的数据(圆心坐标、半径、颜色)和相同的操作(显示自己、放大缩小半径、在屏幕上移动位置，等等)。因此，它们是同一类事物，可以用"Circle 类”来定义。

以上先详细地阐述了对象的定义，然后在此基础上定义了类。也可以先定义类再定义对象，例如，可以像下面这样定义类和对象：类是支持继承的抽象数据类型，而对象就是类的实例。

2.实例(instance)

实例就是由某个特定的类所描述的一个具体的对象。类是对具有相同属性和行为的一组相似的对象的抽象，类在现实世界中并不能真正存在。在地球上并没有抽象的“中国人”，只有一个个具体的中国人，例如，张三、李四、王五……同样，谁也没见过抽象的“圆”，只有一个个具体的圆。

实际上类是建立对象时使用的“样板”，按照这个样板所建立的一个个具体的对象，就
是类的实际例子，通常称为实例。

当使用“对象”这个术语时，既可以指一个具体的对象，也可以泛指一般的对象，但是，
当使用“实例”这个术语时，必然是指一个具体的对象。

3.消息(message)

消息就是要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明。通常，一个消息由下述 3 部分组成:

·接收消息的对象;

·消息选择符(也称为消息名);

·零个或多个变元。

例如，MyCircle 是一个半径 4cm、圆心位于(100，200)的 Circle 类的对象，也就是 Circle 类的一个实例，当要求它以绿颜色在屏幕上显示自己时，在 C++语言中应该向它发下列消息:

```
MyCircle.Show(GREEN);
```

其中 MyCircle 是接收消息的对象的名字，Show 是消息选择符(即消息名)，圆括

号内的GREEN是消息的变元。当MyCircle接收到这个消息后，将执行在Circle类中所定义的Show操作。

4.方法(method)

方法就是对象所能执行的操作，也就是类中所定义的服务。方法描述了对象执行操作的算法，响应消息的方法。在C++语言中把方法称为成员函数。

例如，为了Circle类的对象能够响应让它在屏幕上显示自己的消息Show(GREEN)，在Circle类中必须给出成员函数Show(int color)的定义，也就是要给出这个成员函数的实现代码。

5.属性(attribute)

属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。类的每个实例都有自己特有的属性值。

在C++语言中把属性称为数据成员。例如，Circle类中定义的代表圆心坐标、半径、颜色等的数据成员，就是圆的属性。

6.封装(encapsulation)

从字面上理解，所谓封装就是把某个事物包起来，使外界不知道该事物的具体内容。

在面向对象的程序中，把数据和实现操作的代码集中起来放在对象内部。一个对象好像是一个不透明的黑盒子，表示对象状态的数据和实现操作的代码与局部数据，都被封装在黑盒子里面，从外面是看不见的，更不能从外面直接访问或修改这些数据和代码。

使用一个对象的时候，只需知道它向外界提供的接口形式，无须知道它的数据结构细节和实现操作的算法。

综上所述，对象具有封装性的条件如下：

(1)有一个清晰的边界。所有私有数据和实现操作的代码都被封装在这个边界内，从外面看不见更不能直接访问。

(2)有确定的接口(即协议)。这些接口就是对象可以接受的消息，只能通过向对象发送消息来使用它。

(3)受保护的内部实现。实现对象功能的细节(私有数据和代码)不能在定义该对象的类的范围外访问。

封装也就是信息隐藏，通过封装对外界隐藏了对象的实现细节。

对象类实质上是抽象数据类型。类把数据说明和操作说明与数据表达和操作实现分离开了，使用者只需知道它的说明(值域及可对数据施加的操作)，就可以使用它。

7.继承(inheritance)

广义地说，继承是指能够直接获得已有的性质和特征，而不必重复定义它们。在面向对象的软件技术中，继承是子类自动地共享基类中定义的数据和方法的机制。

面向对象软件技术的许多强有力的功能和突出的优点，都来源于把类组成一个层次结构的系统(类等级)：一个类的上层可以有父类，下层可以有子类。这种层次结构系统的一个重要性质是继承性，一个类直接继承其父类的全部描述(数据和操作)。为了更深入、具体地理解继承性的含义，图9.4描绘了实现继承机制的原理。

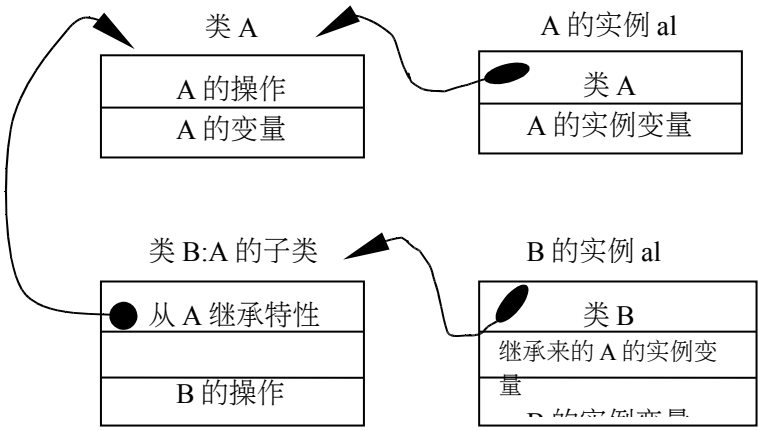

图 9.4　实现继承机制的原理

图中以A、B两个类为例，其中B类是从A类派生出来的子类，它除了具有自己定义的特性(数据和操作)之外，还从父类A继承特性。当创建A类的实例a1的时候，a1以A类为样板建立实例变量(在内存中分配所需要的空间)，但是它并不从A类中复制所定义的方法。

当创建B类的实例b1的时候，b1既要以B类为样板建立实例变量，又要以A类为样板建立实例变量b1所能执行的操作既有B类中定义的方法，又有A类中定义的方法，这就是继承。当然，如果B类中又定义了和A类中同名的数据或操作，则b1仅使用B类中定义的这个数据或操作，除非采用特别措施，否则A类中与之同名的数据或操作在b1中就不能使用。

继承具有传递性，如果类C继承类B，类B继承类A，则类C继承类A。因此，一个类
实际上继承了它所在的类等级中在它上层的全部基类的所有描述，也就是说，属于某类的对象除了具有该类所描述的性质外，还具有类等级中该类上层全部基类描述的一切性质。

当一个类只允许有一个父类时，也就是说，当类等级为树形结构时，类的继承是单继承；当允许一个类有多个父类时，类的继承是多重继承。多重继承的类可以组合多个父类的性质构成所需要的性质，因此功能更强、使用更方便；但

是，使用多重继承时要注意避免二义性。

继承性使得相似的对象可以共享程序代码和数据结构，从而大大减少了程序中的冗余信息。在程序执行期间，对对象某一性质的查找是从该对象类在类等级中所在的层次开始，沿类等级逐层向上进行的，并把第一个被找到的性质作为所要的性质。因此，低层的性质将屏蔽高层的同名性质。

使用从原有类派生出新的子类的办法，使得对软件的修改变得比过去容易得多了。当需要扩充原有的功能时，派生类的方法可以调用其基类的方法，并在此基础上增加必要的程序代码；当需要完全改变原有操作的算法时，可以在派生类中实现一个与基类方法同名而算法不同的方法；当需要增加新的功能时，可以在派生类中实现一个新的方法。

继承性使得用户在开发新的应用系统时不必完全从零开始，可以继承原有的相似系统的功能或者从类库中选取需要的类，再派生出新的类以实现所需要的功能。

有了继承性以后，还可以用把已有的一般性的解加以具体化的办法，来达到软件重用的目的：首先，使用抽象的类开发出一般性问题的解，然后，在派生类中增加少量代码使一般性的解具体化，从而开发出符合特定应用需要的具体解。

8.多态性(polymorphism)

多态性一词来源于希腊语，意思是“有许多形态”。在面向对象的软件技术中，多态性是指子类对象可以像父类对象那样使用，同样的消息既可以发送给父类对象也可以发送给子类对象。也就是说，在类等级的不同层次中可以共享(公用)一个行为(方法)的名字，然而不同层次中的每个类却各自按自己的需要来实现这个行为。当对象接收到发送给它的消息时，根据该对象所属于的类动态选用在该类中定义的实现算法。

在C++语言中，多态性是通过虚函数来实现的。在类等级不同层次中可以说明名字、参数特征和返回值类型都相同的虚拟成员函数，而不同层次的类中的虚函数实现算法各不相同。虚函数机制使得程序员能在一个类等级中使用相同函数的多个不同版本，在运行时刻才根据接收消息的对象所属于的类，决定到底执行哪个特定的版本，这称为动态联编，也叫滞后联编。

多态性机制不仅增加了面向对象软件系统的灵活性，进一步减少了信息冗余，而且显著提高了软件的可重用性和可扩充性。当扩充系统功能增加新的实体类型时，只须派生出与新实体类相应的新的子类，并在新派生出的子类中定义符合该类需要的虚函数，完全无须修改原有的程序代码，甚至不需要重新编译原有的程序(仅需编译新派生类的源程序，再与原有程序的.OBJ文件连接)。

9.重载(0verloading)

有两种重载：函数重载是指在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名字；运算符重载是指同一个运算符可以施加于不同类型

的操作数上面。当然，当参数特征不同或被操作数的类型不同时，实现函数的算法或运算符的语义是不相同的。

在 C++语言中函数重载是通过静态联编(也叫先前联编)实现的，也就是在编译时根据函数变元的个数和类型，决定到底使用函数的哪个实现代码；对于重载的运算符，同样是在编译时根据被操作数的类型，决定使用该算符的哪种语义。

重载进一步提高了面向对象系统的灵活性和可读性。

9.3 面向对象建模

众所周知，在解决问题之前必须首先理解所要解决的问题。对问题理解得越透彻，就越容易解决它。当完全、彻底地理解了一个问题的时候，通常就已经解决了这个问题。

为了更好地理解问题，人们常常采用建立问题模型的方法。所谓模型，就是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。通常，模型由一组图示符号和组织这些符号的规则组成，利用它们来定义和描述问题域中的术语和概念。更进一步讲，模型是一种思考工具，利用这种工具可以把知识规范地表示出来。

模型可以帮助我们思考问题、定义术语、在选择术语时作出适当的假设，并且可以帮助我们保持定义和假设的一致性。

为了开发复杂的软件系统，系统分析员应该从不同角度抽象出目标系统的特性，使用精确的表示方法构造系统的模型，验证模型是否满足用户对目标系统的需求，并在设计过程中逐渐把和实现有关的细节加进模型中，直至最终用程序实现模型。对于那些因过分复杂而不能直接理解的系统，特别需要建立模型，建模的目的主要是为了减少复杂性。人的头脑每次只能处理一定数量的信息，模型通过把系统的重要部分分解成人的头脑一次能处理的若干个子部分，从而减少系统的复杂程度。

在对目标系统进行分析的初始阶段，面对大量模糊的、涉及众多专业领域的、错综复杂的信息，系统分析员往往感到无从下手。模型提供了组织大量信息的一种有效机制。

一旦建立起模型之后，这个模型就要经受用户和各个领域专家的严格审查。由于模型的规范化和系统化，因此比较容易暴露出系统分析员对目标系统认识的片面性和不一致性。通过审查，往往会发现许多错误，发现错误是正常现象，这些错误可以在成为目标系统中的错误之前，就被预先清除掉。

通常，用户和领域专家可以通过快速建立的原型亲身体验，从而对系统模型进行更有效的审查。模型常常会经过多次必要的修改，通过不断改正错误的或不全面的认识，最终使软件开发人员对问题有了透彻的理解，从而为后续的开发工作奠定了坚实基础。

用面向对象方法成功地开发软件的关键，同样是对问题域的理解。面向对象方法最基本的原则，是按照人们习惯的思维方式，用面向对象观点建立问题域的模型，开发出尽可能自然地表现求解方法的软件。

用面向对象方法开发软件，通常需要建立 3 种形式的模型，它们分别是描

述系统数据结构的对象模型，描述系统控制结构的动态模型和描述系统功能的功能模型。这3种模型都涉及到数据、控制和操作等共同的概念，只不过每种模型描述的侧重点不同。这3种模型从3个不同但又密切相关的角度模拟目标系统，它们各自从不同侧面反映了系统的实质性内容，综合起来则全面地反映了对目标系统的需求。一个典型的软件系统组合了上述3方面内容：它使用数据结构(对象模型)，执行操作(动态模型)，并且完成数据值的变化(功能模型)。

为了全面地理解问题域，对任何大系统来说，上述3种模型都是必不可少的。当然，在不同的应用问题中，这3种模型的相对重要程度会有所不同，但是，用面向对象方法开发软件，在任何情况下，对象模型始终都是最重要、最基本、最核心的。在整个开发过程中，3种模型一直都在发展、完善。在面向对象分析过程中，构造出完全独立于实现的应用域模型；在面向对象设计过程中，把求解域的结构逐渐加入到模型中；在实现阶段，把应用域和求解域的结构都编成程序代码并进行严格的测试验证。

下面分别介绍上述3种模型。

9.4 对象模型

对象模型表示静态的、结构化的系统的“数据”性质。它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，描述了系统的静态结构。正如9.1节所述，面向对象方法强调围绕对象而不是围绕功能来构造系统。对象模型为建立动态模型和功能模型，提供了实质性的框架。

在建立对象模型时，我们的目标是从客观世界中提炼出对具体应用有价值的概念。

为了建立对象模型，需要定义一组图形符号，并且规定一组组织这些符号以表示特定语义的规则。也就是说，需要用适当的建模语言来表达模型，建模语言由记号(即模型中使用的符号)和使用记号的规则(语法、语义和语用)组成。

一些著名的软件工程专家在提出自己的面向对象方法的同时，也提出了自己的建模语言。但是，面向对象方法的用户并不了解不同建模语言的优缺点，很难在实际工作中根据应用的特点选择合适的建模语言，而且不同建模语言之间存在的细微差别也极大地妨碍了用户之间的交流。面向对象方法发展的现实，要求在精心比较不同建模语言的优缺点和总结面向对象技术应用经验的基础上，把建模语言统一起来。

曾对面向对象方法学的发展做出过重要贡献的Booch，Rumbaugh和Jacobson经过合作研究，于1996年6月设计出统一建模语言UML 0.9。截止到1996年10月，在美国已有700多家公司表示支持采用UML作为建模语言，在1996年年底，UML已经稳定地占领了面向对象技术市场的85%，成为事实上的工业标准。1997年11月，国际对象管理组织OMG批准把UMI1.1作为基于面向对象技术的标准建模语言。

通常，使用UML提供的类图来建立对象模型。在UMI中术语“类”的实际含义是，“一个类及属于该类的对象”。下面简要地介绍UML的类图。

9.4.1 类图的基本符号

类图描述类及类与类之间的静态关系。类图是一种静态模型，它是创建其他UML图的基础。一个系统可以由多张类图来描述，一个类也可以出现在几张类图中。

1.定义类

UML中类的图形符号为长方形，用两条横线把长方形分成上、中、下3个区域(下面两个区域可省略)，3个区域分别放类的名字、属性和服务，如图9.5所示。

类名是一类对象的名字。命名是否恰当对系统的可理解性影响相当大，因此，为类命名时应该遵守以下几条准则：

(1)使用标准术语。应该使用在应用领域中人们习惯的标准术语作为类名，不要随意创造名字。例如，“交通信号灯”比“信号单元”这个名字好，“传送带”比“零件传送设备”好。

(2)使用具有确切含义的名词。尽量使用能表示类的含义的日常用语作名字，不要使用空洞的或含义模糊的词作名字。例如，“库房”比“房屋”或“存物场所”更确切。

图9.5 表示类的图形符号

(3)必要时用名词短语作名字。为使名字的含义更准确，必要时用形容词加名词或其他形式的名词短语作名字。例如，“最小的领土单元”、“储藏室”、“公司员工”等都是比较恰当的名字。

总之，名字应该是富于描述性的、简洁的而且无二义性的。

2.定义属性

UML描述属性的语法格式如下：

可见性属性名：类型名=初值{性质串}

属性的可见性(即可访问性)通常有下述3种：公有的(public)、私有的(private)和保护的(protected)，分别用加号(+)、减号(-)和井号(#)表示。如果未声明可见性，则表示该属性的可见性尚未定义。注意，没有默认的可见性。

属性名和类型名之间用冒号(：)分隔。类型名表示该属性的数据类型，它可以是基本数据类型，也可以是用户自定义的类型。

在创建类的实例时应给其属性赋值，如果给某个属性定义了初值，则该初值可作为创建实例时这个属性的默认值。类型名和初值之间用等号(=)隔开。

用花括号括起来的性质串明确地列出该属性所有可能的取值。枚举类型的属性往往用性质串列出可以选用的枚举值，不同枚举值之间用逗号分隔。也可以用

性质串说明属性的其他性质，例如，约束说明{只读}表明该属性是只读属性。

例如，“发货单”类的属性“管理员”，在 UML 类图中像下面那样描述：

-管理员：String=“未定”

类的属性中还可以有一种能被该类所有对象共享的属性，称为类的作用域属性，也称为类变量。C+k 语言中的静态数据成员就是这样的属性。类变量在类图中表示为带下划线的属性，例如，发货单类的类变量“货单数”，用来统计发货单的总数，在该类所有对象中这个属性的值都是一样的，下面是对这个属性的描述：

-货单数：Integer

3.定义服务

服务也就是操作，UML 描述操作的语法格式如下：

可见性操作名(参数表)：返回值类型{性质串}

操作可见性的定义方法与属性相同。

参数表是用逗号分隔的形式参数的序列。描述一个参数的语法如下：

参数名：类型名=默认值

当操作的调用者未提供实在参数时，该参数就使用默认值。

与属性类似，在类中也可定义类作用域操作，在类图中表示为带下划线的操作。这种操作只能存取本类的类作用域属性。

9.4.2 表示关系的符

如前所述，类图由类及类与类之间的关系组成。定义了类之后就可以定义类与类之间的各种关系了。类与类之间通常有关联、泛化(继承)、依赖和细化等 4 种关系。

1.关联

关联表示两个类的对象之间存在某种语义上的联系。例如，作家使用计算机，我们就认为在作家和计算机之间存在某种语义连接，因此，在类图中应该在作家类和计算机类之间建立关联关系。

(1)普通关联

普通关联是最常见的关联关系，只要在类与类之间存在连接关系就可以用普通关联表示。普通关联的图示符号是连接两个类之间的直线，如图 9.6 所示。

图 9.6　普通关联示例

通常，关联是双向的，可在一个方向上为关联起一个名字，在另一个方向上起另一个名字(也可不起名字)。为避免混淆，在名字前面(或后面)加一个表示关联方向的黑三角。

在表示关联的直线两端可以写上重数(multiplicity)，它表示该类有多少个对象与对方的一个对象连接。重数的表示方法通常有：　　，

0..1　　表示　0到1个对象
0..*或*　　表示　0到多个对象
1+或1..*　　表示　1到多个对象
1..15　　表示　1到15个对象
3　　表示3个对象

如果图中未明确标出关联的重数，则默认重数是1。

图9.6表示，一个作家可以使用1到多台计算机，一台计算机可被0至多个作家使用。

(2)关联的角色

在任何关联中都会涉及到参与此关联的对象所扮演的角色(即起的作用)，在某些情况下显式标明角色名有助于别人理解类图。例如，图9.7是一个递归关联(即一个类与它本身有关联关系)的例子。一个人与另一个人结婚，必然一个人扮演丈夫的角色，另一个人扮演妻子的角色。如果没有显式标出角色名，则意味着用类名作为角色名。

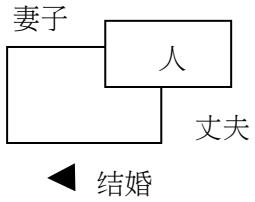

图9.7　关联的角色

(3)限定关联

限定关联通常用在一对多或多对多的关联关系中，可以把模型中的重数从一对多变成一对一，或从多对多简化成多对一。在类图中把限定词放在关联关系末端的一个小方框内。

例如，某操作系统中一个目录下有许多文件，一个文件仅属于一个目录，在一个目录内文件名确定了惟一一个文件。图9.8利用限定词“文件名”表示了目录与文件之间的关系，可见，利用限定词把一对多关系简化成了一对一关系。

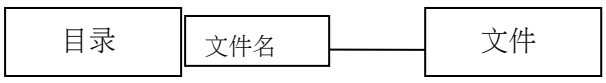

图9.8　一个受限的关联

限定提高了语义精确性，增强了查询能力。在图 9.8 中，限定的语法表明，文件名在其目录内是惟一的。因此，查找一个文件的方法就是，首先定下目录，然后在该目录内查找指定的文件名。由于目录加文件名可惟一地确定一个文件，因此，限定词“文件名”应该放在靠近目录的那一端。

(4)关联类

为了说明关联的性质可能需要一些附加信息。可以引入一个关联类来记录这些信息。关联中的每个连接与关联类的一个对象相联系。关联类通过一条虚线与关联连接。例如，图 9.9 是一个电梯系统的类模型，队列就是电梯控制器类与电梯类的关联关系上的关联类。从图中可以看出，一个电梯控制器控制着 4 台电梯，这样，控制器和电梯之间的实际连接就有 4 个，每个连接都对应一个队列(对象)，每个队列(对象)存储着来自控制器和电梯内部按钮的请求服务信息。电梯控制器通过读取队列信息，选择一个合适的电梯为乘客服务。关联类与一般的类一样，也有属性、操作和关联。

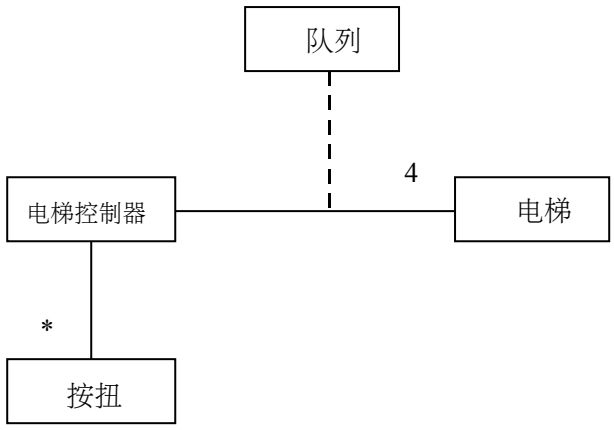

图 9.9　关键类示例

2.聚集

聚集也称为聚合，是关联的特例。聚集表示类与类之间的关系是整体与部分的关系。在陈述需求时使用的“包含”、“组成”、“分为……部分”等字句，往往意味着存在聚集关系。除了一般聚集之外，还有两种特殊的聚集关系，分别是

共享聚集和组合聚集。

(1)共享聚集

如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集。例如，一个课题组包含许多成员，每个成员又可以是另一个课题组的成员，则课题组和成员之间是共享聚集关系，如图 9.10 所示。一般聚集和共享聚集的图示符号，都是在表示关联关系的直线末端紧挨着整体类的地方画一个空心菱形。

图 9.10 共享聚集示例

(2)组合聚集

如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失(或失去存在价值了)，则该聚集称为组合聚集(简称为组成)。例如，在屏幕上打开一个窗口它就由文本框、列表框、按钮和菜单组成，一旦关闭了窗口，各个组成部分也同时消失，窗口和它的组成部分之间存在着组合聚集关系。图 9.11 是窗口的组成，从图上可以看出，组成关系用实心菱形表示。

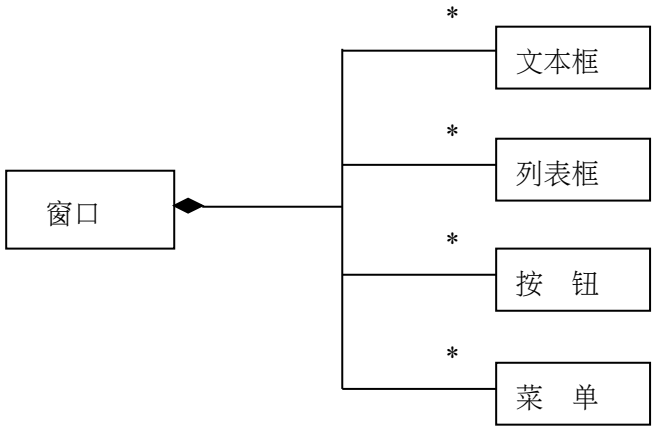

图 9.11 组合聚集示例

3.泛化

UML 中的泛化关系就是通常所说的继承关系，它是通用元素和具体元素之间的一种分类关系。具体元素完全拥有通用元素的信息，并且还可以附加一些其他信息。

在 UML 中，用一端为空心三角形的连线表示泛化关系，三角形的顶角紧挨着通用元素。

注意，泛化针对类型而不针对实例，一个类可以继承另一个类，但一个对象不能继承另一个对象。实际上，泛化关系指出在类与类之间存在“一般一特殊”关系。泛化可进一步划分成普通泛化和受限泛化。

(1)普通泛化

普通泛化与 9.2.2 节中讲过的继承基本相同，对普通泛化的概念此处不再赘述。

需要特别说明的是，没有具体对象的类称为抽象类。抽象类通常作为父类，用于描述其他类(子类)的公共属性和行为。图示抽象类时，在类名下方附加一个标记值{abstract}，如图 9.12 所示。图下方的两个折角矩形是模型元素“笔记”的符号，其中的文字是注释，分别说明两个子类的操作 drive 的功能。

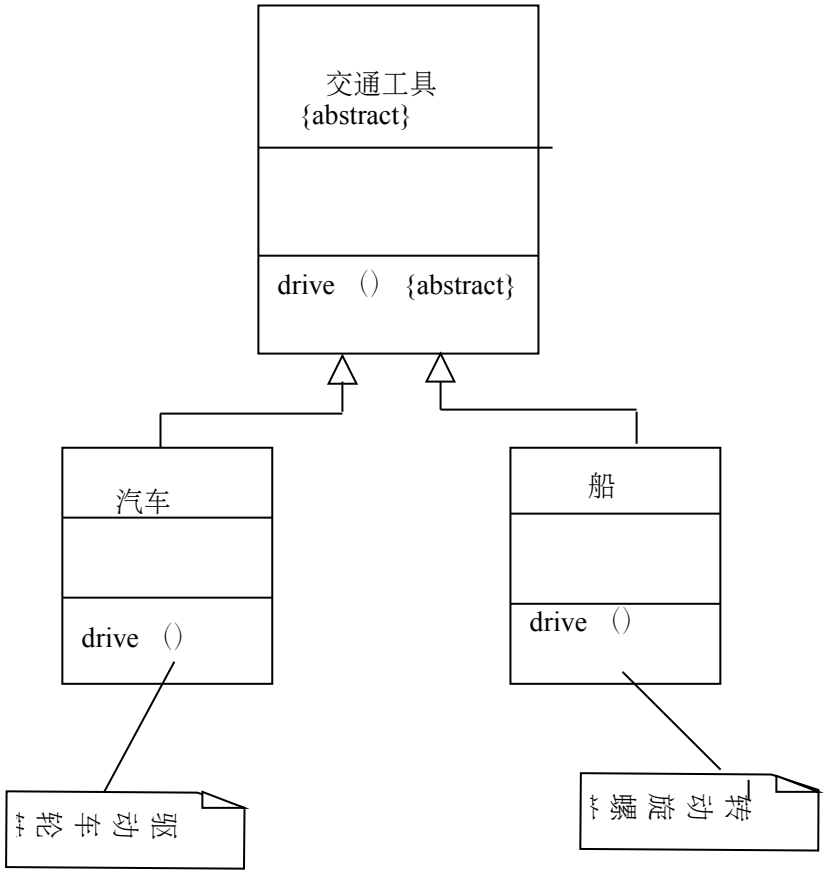

图 9.12 抽象类示例

抽象类通常都具有抽象操作。抽象操作仅用来指定该类的所有子类应具有哪些行为。抽象操作的图示方法与抽象类相似，在操作标记后面跟随一个性质串{abstract)。

与抽象类相反的类是具体类，具体类有自己的对象，并且该类的操作都有具体的实现方法。

图 9.13 给出一个比较复杂的类图示例，这个例子综合应用了前面讲过的许多概念和图示符号。图 9.13 表明，一幅工程蓝图由许多图形组成，图形可以是直线、圆、多边形或组合图，而多边形由直线组成，组合图由各种线型混合而成。当客户要求画一幅蓝图时，系统便通过蓝图与图形之间的关联(聚集)关系，由图形来完成画图工作，但是图形是抽象类，因此当涉及到某种具体图形(如，直线、圆等)时，便使用其相应子类中具体实现的 draw 功能完成绘图工作。

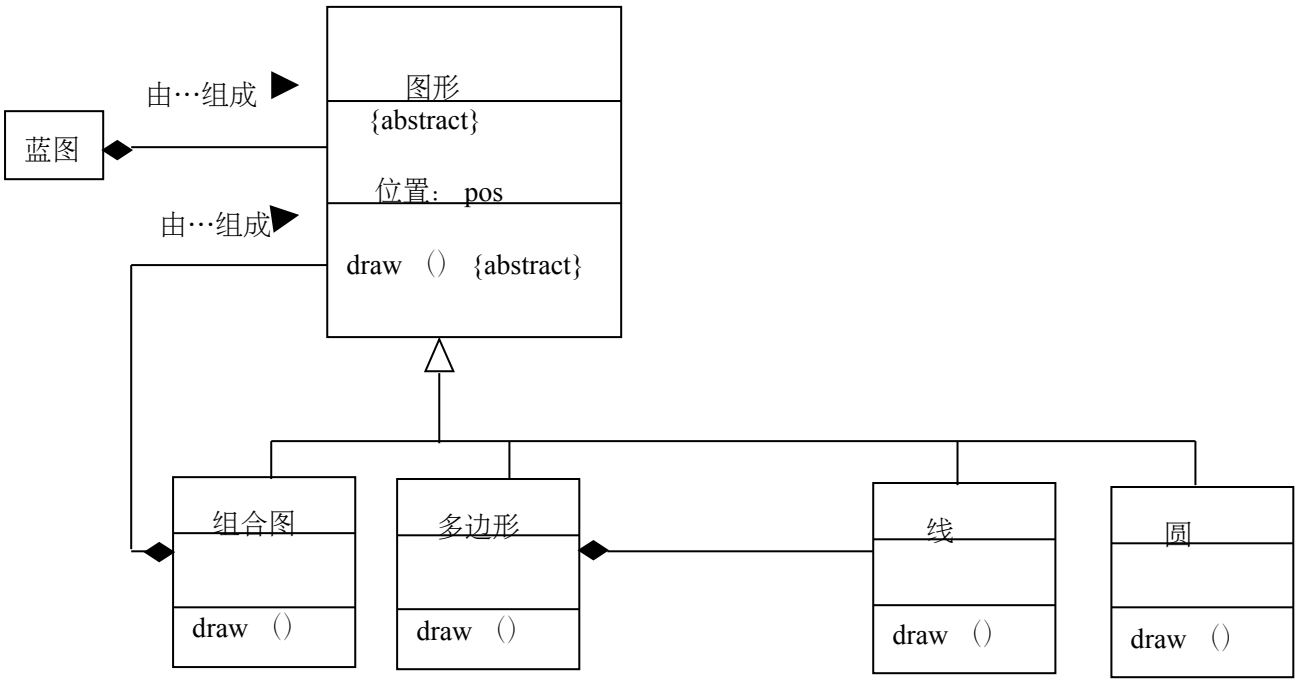

图 9.13 复杂类图示例

(2)受限泛化

可以给泛化关系附加约束条件，以进一步说明该泛化关系的使用方法或扩充方法，这样的泛化关系称为受限泛化。预定义的约束有4种：多重、不相交、完全和不完全。这些约束都是语义约束。

多重继承指的是，一个子类可以同时多次继承同一个上层基类，例如图9.14中的水陆两用类继承了两次交通工具类。

与多重继承相反的是不相交继承，即一个子类不能多次继承同一个基类(这样的基类相当于c++语言中的虚基类)。如果图中没有指定{多重)约束，则是不相交继承，一般的继承都是不相交继承。

完全继承指的是父类的所有子类都已在类图中穷举出来了，图示符号是指定{完全}约束。

不完全继承与完全继承恰好相反，父类的子类并没有都穷举出来，随着对问题理解的深入，可不断补充和维护，这为日后系统的扩充和维护带来很大方便。不完全继承是一般情况下默认的继承关系。

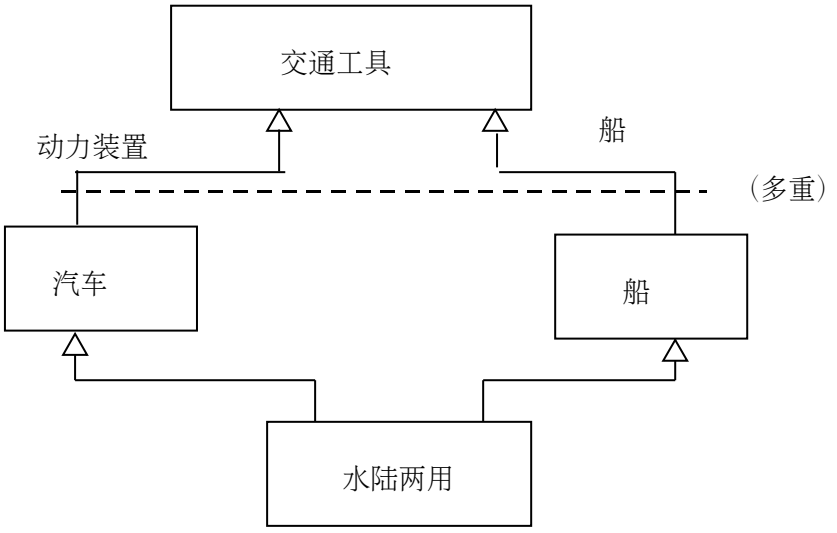

图 9.14 多重继承示例

4.依赖和细化

(1)依赖关系

依赖关系描述两个模型元素(类、用例等)之间的语义连接关系：其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。例如，一个类使用另一

个类的对象作为操作的参数，一个类用另一个类的对象作为它的数据成员，一个类向另一个类发消息等，这样的两个类之间都存在依赖关系。

在 UML 的类图中，用带箭头的虚线连接有依赖关系的两个类，箭头指向独立的类。在虚线上可以带一个版类标签，具体说明依赖的种类，例如，图 9.15 表示一个友元依赖关系，该关系使得 B 类的操作可以使用 A 类中私有的或保护的成员。

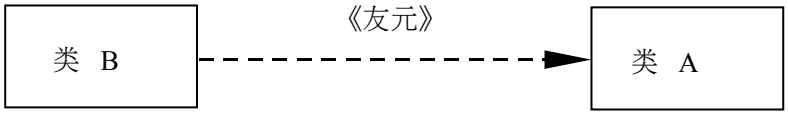

图 9.15 友元依赖关系

(2)细化关系

当对同一个事物在不同抽象层次上描述时，这些描述之间具有细化关系。假设两个模型元素 A 和 B 描述同一个事物，它们的区别是抽象层次不同，如果 B 是在 A 的基础上的更详细的描述，则称 B 细化了 A，或称 A 细化成了 B。细化的图示符号为由元素 B 指向元素 A 的、一端为空心三角形的虚线(注意，不是实线)，如图 9.16 所示。细化用来协调不同阶段模型之间的关系，表示各个开发阶段不同抽象层次的模型之间的相关性，常常用于跟踪模型的演变。

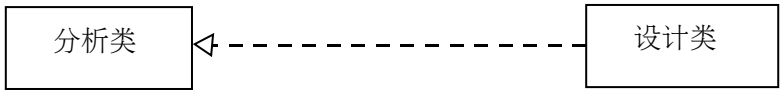

图 9.16 细化关系示例

9.5 动态模型

动态模型表示瞬时的、行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。

一旦建立起对象模型之后，就需要考察对象的动态行为。所有对象都具有自己的生命周期(或称为运行周期)。对一个对象来说，生命周期由许多阶段组成，在每个特定阶段中，都有适合该对象的一组运行规律和行为规则，用以规范该对象的行为。生命周期中的阶段也就是对象的状态。所谓状态，是对对象属性值的一种抽象。当然，在定义状态时应该忽略那些不影响对象行为的属性。各对象

之间相互触发(即作用)就形成了一系列的状态变化。我们把一个触发行为称作一个事件。对象对事件的响应，取决于接受该触发的对象当时所处的状态，响应包括改变自己的状态或者又形成一个新的触发行为。

状态有持续性，它占用一段时间间隔。状态与事件密不可分，一个事件分开两个状态，一个状态隔开两个事件。事件表示时刻，状态代表时间间隔。

通常，用 UML 提供的状态图来描绘对象的状态、触发状态转换的事件以及对象的行为(对事件的响应)。

每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型。也就是说，动态模型是基于事件共享而互相关联的一组状态图的集合。

本书 3.6 节已经介绍过状态图，此处不再赘述。

9.6 功能模型

功能模型表示变化的系统的“功能”性质，它指明了系统应该“做什么”，因此更直接地反映了用户对目标系统的需求。

通常，功能模型由一组数据流图组成。在面向对象方法学中，数据流图远不如在结构分析、设计方法中那样重要。一般说来，与对象模型和动态模型比较起来，数据流图并没有增加新的信息，但是，建立功能模型有助于软件开发人员更深入地理解问题域，改进和完善自己的设计。因此，不能完全忽视功能模型的作用。

在本书第 2 章中已经详细讲述了数据流图的符号和画法，此处不再赘述。

UML 提供的用例图也是进行需求分析和建立功能模型的强有力工具。在 UML 中把用用例图建立起来的系统模型称为用例模型。

用例模型描述的是外部行为者(actor)所理解的系统功能。用例模型的建立是系统开发者和用户反复讨论的结果，它描述了开发者和用户对需求规格所达成的共识。

9.6.1 用例图

一幅用例图包含[illegible]模型元素有[illegible]及用例之间的关系。图 9.17 是自动售货机系统的[illegible]代表系统，椭圆代表用例(售货、供货和取货款[illegible]条人代表行为者，它们之间的连线表示关系。

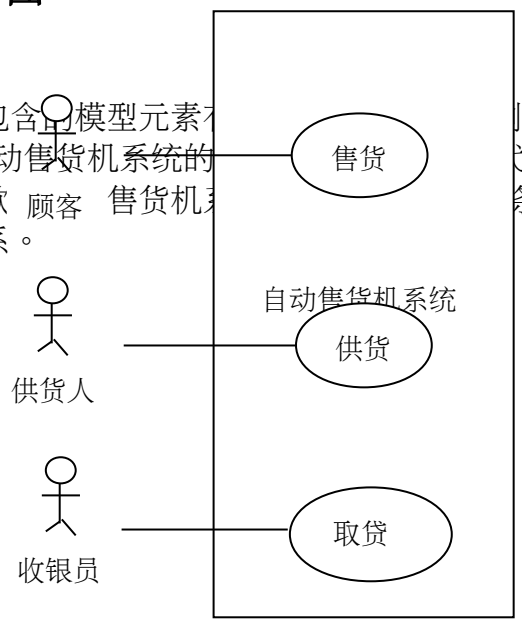

图 9.17　自动售货机系统用例图

1.系统

系统被看作是一个提供用例的黑盒子，内部如何工作、用例如何实现，这些对于建立用例模型来说都是不重要的。

代表系统的方框的边线表示系统的边界，用于划定系统的功能范围，定义了系统所具有的功能。描述该系统功能的用例置于方框内，代表外部实体的行为者置于方框外。

2.用例

一个用例是可以被行为者感受到的、系统的一个完整的功能。在 UML 中把用例定义成系统完成的一系列动作，动作的结果能被特定的行为者察觉到。这些动作除了完成系统内部的计算与工作外，还包括与一些行为者的通信。用例通过关联与行为者连接，关联指出一个用例与哪些行为者交互，这种交互是双向的。

用例具有下述特征：

(1)用例代表某些用户可见的功能，实现一个具体的用户目标；

(2)用例总是被行为者启动的，并向行为者提供可识别的值；

(3)用例必须是完整的。

注意，用例是一个类，它代表一类功能而不是使用该功能的某个具体实例。用例的实例是系统的一种实际使用方法，通常把用例的实例称为脚本。脚本是系统的一次具体执行过程，例如，在自动售货机系统中，张三投入硬币购买矿泉水，系统收到钱后把矿泉水送出来，上述过程就是一个脚本；李四投币买可乐，但是可乐已卖完了，于是系统给出提示信息并把钱退还给李四，这个过程是另一个脚本。

3.行为者

行为者是指与系统交互的人或其他系统，它代表外部实体。使用用例并且与系统交互的任何人或物都是行为者。

行为者代表一种角色，而不是某个具体的人或物。例如，在自动售货机系统中，使用售货功能的人既可以是张三(买矿泉水)也可以是李四(买可乐)，但是不能把张三或李四这样的个体对象称为行为者。事实上，一个具体的人可以充当多种不同角色。例如，某个人既可以为售货机添加商品(执行供货功能)，又可以把售货机中的钱取走(执行取货款功能)。

在用例图中用直线连接行为者和用例，表示两者之间交换信息，称为通信联系。行为者触发(激活)用例，并与用例交换信息。单个行为者可与多个用例联系；反之，一个用例也司与多个行为者联系。对于同一个用例而言，不同行为者起的作用也不同。可以把行为者分成主行为者和副行为者，还可分成主动行为者和被动行为者。

实践表明，行为者对确定用例是非常有用的。面对一个大型、复杂的系统，要列出用例清单往往很困难，可以先列出行为者清单，再针对每个行为者列出它的用例。这样做可以比较容易地建立起用例模型。

4.用例之间的关系

UML 用例之间主要有扩展和使用两种关系，它们是泛化关系的两种不同形式。

(1)扩展关系

向一个用例中添加一些动作后构成了另一个用例，这两个用例之间的关系就是扩展关系，后者继承前者的一些行为，通常把后者称为扩展用例。例如，在自动售货机系统中，“售货”是一个基本的用例，如果顾客购买罐装饮料，售货功能完成得很顺利，但是，如果顾客要购买用纸杯装的散装饮料，则不能执行该用例提供的常规动作，而要做些改动。我们可以修改售货用例，使之既能提供售罐装饮料的常规动作又能提供售散装饮料的非常规动作，但是，这将把该用例与一些特殊的判断和逻辑混杂在一起，使正常的流程晦涩难懂。图 9.18 中把常规动作放在“售货”用例中，而把非常规动作放置于“售散装饮料”用例中，这两个用例之间的关系就是扩展关系。在用例图中，用例之间的扩展关系图示为带版类《扩展》的泛化关系。

(2)使用关系

当一个用例使用另一个用例时，这两个用例之间就构成了使用关系。一般说来，如果在若于个用例中有某些相同的动作，则可以把这些相同的动作提取出来单独构成一个用例(称为抽象用例)。这样，当某个用例使用该抽象用例时，就好像这个用例包含了抽象用例中的所有动作。例如，在自动售货机系统中，“供货”和“取货款，，这两个用例的开始动作都是去掉机器保险并打开它，而最后的动作都是关上机器并加上保险，可以从这两个用例中把开始的动作抽象成

“打开机器”用例，把最后的动作抽象成“关闭机器，，用例。于是，“供货”和“取货款”用例在执行时必须使用上述的两个抽象用例，它们之间便构成了使用关系。在用例图中，用例之间的使用关系用带版类《使用》的泛化关系表示，如图 9.18 所示。

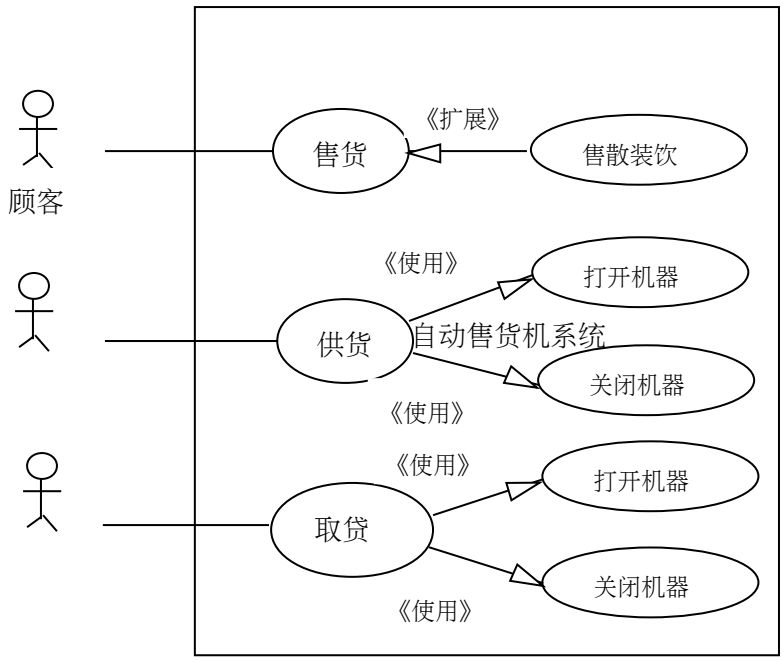

图 9.18　含扩展和使用关系的用例图

请注意扩展与使用之间的异同：这两种关系都意味着从几个用例中抽取那些公共的行为并放入一个单独的用例中，而这个用例被其他用例使用或扩展，但是，使用和扩展的目的是不同的。通常在描述一般行为的变化时采用扩展关系；在两个或多个用例中出现重复描述又想避免这种重复时，可以采用使用关系。

9.6.2 用例建模

几乎在任何情况下都需要使用用例，通过用例可以获取用户需求，规划和控制项目。获取用例是需求分析阶段的主要工作之一，而且是首先要做的工作。大部分用例将在项目的需求分析阶段产生，并且随着开发工作的深入还会发现更多用例，这些新发现的用例都应及时补充进已有的用例集中。用例集中的每个用例都是对系统的一个潜在的需求。

一个用例模型由若干幅用例图组成。创建用例模型的工作包括：定义系统，寻找行为者和用例，描述用例，定义用例之间的关系，确认模型。其中，寻找行为者和用例是关键。

1.寻找行为者

为获取用例首先要找出系统的行为者，可以通过请系统的用户回答一些问题的办法来发现行为者。下述问题有助于发现行为者：

·谁将使用系统的主要功能(主行为者)?
·谁需要借助系统的支持来完成日常工作?
·谁来维护和管理系统(副行为者)?
·系统控制哪些硬件设备?
·系统需要与哪些其他系统交互?
·哪些人或系统对本系统产生的结果(值)感兴趣?

2.寻找用例

一旦找到了行为者，就可以通过请每个行为者回答下述问题来获取用例：
，行为者需要系统提供哪些功能?行为者自身需要做什么?
·行为者是否需要读取、创建、删除、修改或存储系统中的某类信息?
·系统中发生的事件需要通知行为者吗?行为者需要通知系统某些事情吗?从功能观点看，这些事件能做什么?
·行为者的日常工作是否因为系统的新功能而被简化或提高了效率?

还有一些不是针对具体行为者而是针对整个系统的问题，也能帮助建模者发现用例，
例如：

·系统需要哪些输入输出?输入来自何处?输出到哪里去?
·当前使用的系统(可能是人工系统)存在的主要问题是什么?

注意，最后这两个问题并不意味着没有行为者也可以有用例，只是在获取用例时还不知道行为者是谁。事实上，一个用例必须至少与一个行为者相关联。

9.7 3种模型之间的关系

面向对象建模技术所建立的3种模型，分别从3个不同侧面描述了所要开发的系统。这3种模型相互补充、相互配合，使得我们对系统的认识更加全面：功能模型指明了系统应该“做什么”；动态模型明确规定了什么时候(即在何种状态下接受了什么事件的触发)做；对象模型则定义了做事情的实体。

在面向对象方法学中，对象模型是最基本最重要的，它为其他两种模型奠定了基础，我们依靠对象模型完成3种模型的集成。下面扼要地叙述3种模型之间的关系。

(1)针对每个类建立的动态模型，描述了类实例的生命周期或运行周期。

(2)状态转换驱使行为发生，这些行为在数据流图中被映射成处理，在用例图中被映射成用例，它们同时与类图中的服务相对应。

(3)功能模型中的处理(或用例)对应于对象模型中的类所提供的服务。通常,复杂的处理(或用例)对应于复杂对象提供的服务,简单的处理(或用例)对应于更基本的对象提供的服务。有时一个处理(或用例)对应多个服务,也有一个服务对应多个处理(或用例)的时候。

(4)数据流图中的数据存储,以及数据的源点/终点,通常是对象模型中的对象。

(5)数据流图中的数据流,往往是对象模型中对象的属性值,也可能是整个对象。

(6)用例图中的行为者,可能是对象模型中的对象。

(7)功能模型中的处理(或用例)可能产生动态模型中的事件。

(8)对象模型描述了数据流图中的数据流、数据存储以及数据源点/终点的结构。

9.8 小 结

近年来,面向对象方法学日益受到人们的重视,特别是在用这种方法开发大型软件产品时,可以把该产品看作是由一系列本质上相互独立的小产品组成,这就不仅降低了开发工作的技术难度,而且也使得对开发工作的管理变得比较容易了。因此,对于大型软件产品来说,面向对象范型明显优于结构化范型。此外,使用面向对象范型能够开发出稳定性好、可重用性好和可维护性好的软件,这些都是面向对象方法学的突出优点。

面向对象方法学比较自然地模拟了人类认识客观世界的思维方式,它所追求的目标和遵循的基本原则,就是使描述问题的问题空间和在计算机中解决问题的解空间,在结构上尽可能一致。

面向对象方法学认为,客观世界由对象组成。任何事物都是对象,每个对象都有自己的内部状态和运动规律,不同对象彼此间通过消息相互作用、相互联系,从而构成了我们所要分析和构造的系统。系统中每个对象都属于一个特定的对象类。类是对具有相同属性和行为的一组相似对象的定义。应该按照子类、父类的关系,把众多的类进一步组织成一个层次系统,这样做了之后,如果不加特殊描述,则处于下一层次上的类可以自动继承位于上一层次的类的属性和行为。

用面向对象观点建立系统的模型,能够促进和加深对系统的理解,有助于开发出更容易理解、更容易维护的软件。通常,人们从3个互不相同然而又密切相关的角度建立起3种不同的模型。它们分别是描述系统静态结构的对象模型、描述系统控制结构的动态模型、以及描述系统计算结构的功能模型。其中,对象模型是最基本、最核心、最重要的。

统一建模语言UML是国际对象管理组织OMG批准的基于面向对象技术的标准建模语言。通常,使用UML的类图来建立对象模型,使用UML的状态图来建立动态模型,使用数据流图或UML的用例图来建立功能模型。在UML中把用用例图建立起来的系统模型称为用例模型。

本章所讲述的面向对象方法及定义的概念和表示符号,可以适用于整个软件开发过程。软件开发人员无须像用结构分析、设计技术那样,在开发过程的不同阶段转换概念和表示符号。实际上,用面向对象方法开发软件时,阶段的划分

是十分模糊的，通常在分析、设计和实现等阶段间多次迭代。喷泉模型是典型的面向对象软件过程模型。

习 题 9

1.什么是面向对象方法学?它有哪些优点?
2.什么是“对象”?它与传统的数据有何异同?
3.什么是“类”?
4.什么是“继承”?
5.什么是模型?开发软件为何要建模?
6.什么是对象模型?建立对象模型时主要使用哪些图形符号?这些符号的含义是什么?
7.什么是动态模型?建立动态模型时主要使用哪些图形符号?这些符号的含义是什么?
8.什么是功能模型?建立功能模型时主要使用哪些图形符号?
9.试用面向对象观点分析、研究本书第2章中给出的定货系统的例子。在这个例子中有哪些类?试建立定货系统的对象模型。
10.建立定货系统的用例模型。