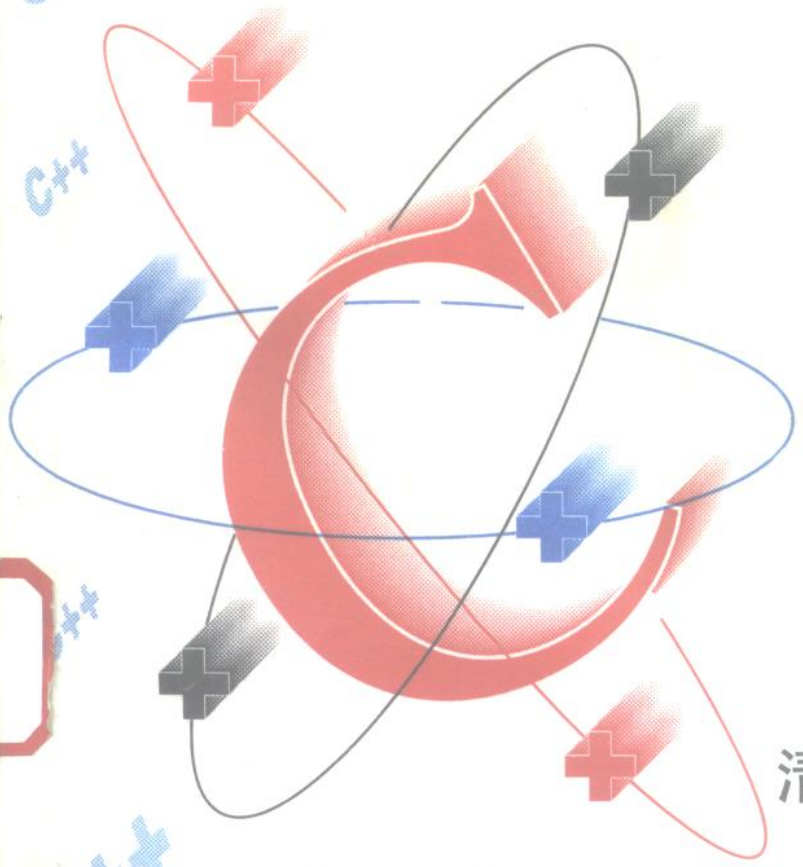
C++ 程序设计语言

张素琴 主编
蒋维杜 审



清华大学出版社

C++程序设计语言

张素琴　主编
蒋维杜　　审

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书介绍了面向对象的基本概念、C++语言中与 C 语言相同的部分以及扩充的面向对象部分，包括类、继承性、多态性、重载等机制，并通过大量实例程序予以说明。全书共分 11 章。

作者根据多年教学和开发的经验，针对初学者的特点，对书的内容作了周密的安排，把 C++语言写得深入浅出，易于掌握。本书体系合理，概念清晰，例题丰富，通俗易懂。

本书既适合 C++语言的初学者使用，也对已经了解 C++语言的读者很有帮助。既可作为大专院校的教材，也可供广大自学者使用。

图书在版编目(CIP)数据

C++程序设计语言/张素琴主编；蒋维杜审. —北京：清华大学出版社，1994
ISBN 7-302-01734-4

Ⅰ.C… Ⅱ.①张… ②蒋… Ⅲ.C 语言-程序设计语言学 Ⅳ.TP312C

中国版本图书馆 CIP 数据核字(94)第 16517 号

出 版 者：清华大学出版社(北京清华大学校内，邮编 100084)
印 刷 者：北京市海淀区清华园印刷厂
发 行 者：新华书店总店北京科技发行所
开　　本：787×1092 1/16　印张：30.25　字数：715 千字
版　　次：1995 年 8 月第 1 版　1995 年 8 月第 1 次印刷
书　　号：ISBN 7-302-01734-4/TP・759
印　　数：0001—8000
定　　价：26.60 元

前言

面向对象的设计方法是吸收了软件工程领域近30年来的有益的概念和有效的方法而发展起来的一种软件设计方法。它集抽象性、封装性、继承性、多态性于一体，与近代软件技术追求的数据抽象、数据隐藏、可复用、易修改、易扩充等特性相对应，将传统软件中的数据和操作组合成易于赋予语义的对象，使软件设计中人们普遍遵循的模块化、信息隐藏、抽象、代码共享等思想在面向对象语言机制的帮助下得以充分实现。

C++语言是目前出现的众多面向对象语言中最引人注目、有广泛发展前景的语言。C++保留了传统的和有效的结构化语言——C的特征，同时融合了面向对象的能力。它在C语言的基础上增加了数据抽象、继承、多态以及其它一些改善C语言程序设计结构的机制，既为程序员提供了面向对象的能力，也未丧失运行时间和空间效率，是一种灵活、高效、可移植的面向对象的语言。

本书向读者介绍了面向对象程序设计方法和C++语言。以C++语言中面向对象机制为主，同时，为了保证语言的完整性，并利于初学者学习，还简单介绍了C++语言中与C语言相同的部分。第一章介绍面向对象设计方法的基本概念；第二、三章介绍C++语言的非面向对象的特征；第四章重点讨论C++语言中用于实现数据抽象和封装的机制——类；第五章简单介绍C++语言中结构、联合和枚举相对于C语言的扩充；第六章在介绍函数机制的同时，重点介绍函数重载的概念；第七章着重讨论继承性，包括单继承和多继承；第八章介绍多态性的概念，主要有重载和虚函数；第九、十章介绍C++语言的I/O流类库和一个数据结构的类库，并讨论了面向对象程序设计环境的特点及类库在环境中所处的地位；最后，在第十一章，列举了C++语言应用的一些例子。

本书为便于初学者学习，认真构造了大量的程序示例，为说明某些概念，有些例子中的注释用中文给出。对例子的分析和研究有助于读者看到实际应用中面向对象问题的解决方法和C++语言的益处。

当前，人工智能、数据库、程序设计语言的研究有汇合之势，面向对象是一个很有希望的汇聚点。在国外，C++语言和面向对象的程序设计已很普及，而在我国，尚处于发展初期。我们希望这本书能起到抛砖引玉的作用，促进C++语言在我国的普及。

本书由张素琴主编，蒋维杜审阅。参加本书编写的还有曹晓峰、范伟、江晓晔；参加审阅工作的还有周庆辉。对于书中的不足之处，希望广大读者与专家批评指正。

作　者

1994年8月

目 录

第一章 面向对象的设计方法及C++简介

本章以软件开发的效率和质量为标准来探讨面向对象设计方法产生的背景。然后，围绕面向对象程序设计的核心概念——封装、继承、多态性及类属性进行扼要的讨论。最后，简单介绍C++语言的发展历史及现状。

1.1 面向对象程序设计的基本思想及基本概念

1.1.1 软件的外部质量

软件的外部质量是指软件中与用户和维护人员有关的质量因素。它主要包括：正确性、健壮性、可扩充性及可复用性等方面。软件的内部质量是指软件中与开发人员有关的质量因素。它主要包括：可读性和可维护性。本节中，我们将与读者共同探讨软件的外部质量；下一节将讨论模块化。

1. 正确性

如果软件能完全按照需求规格说明中的规定运行，那么软件就是正确的。显然，正确性是最重要的质量。如果一个软件不能按要求工作，其它任何要求都是没有意义的。事实上，完美的正确性实现起来不是一件容易的事。这主要是因为用自然语言描述的需求规格说明有许多含糊不清的内容，从而造成开发人员与用户之间在理解上的偏差。

2. 健壮性

健壮性是指软件处理非正常情况的能力。其中，非正常情况包括：用户的误操作和异常等。健壮性与正确性有很大的不同。正确性的要求在需求规格说明中阐述得很清楚，而在需求规格说明中并没有明确指出都有哪些非正常情况及各种情况都应怎样处理。健壮性要求在出现非法情况时系统不会产生灾难性的后果，例如死机、丢失或破坏重要数据等。

3. 可扩充性

可扩充性是指当规格说明变化时，相应的软件系统进行变更的难易程度。其中规格说明变化包括功能的变化、性能的变化以及环境的变化等。

可扩充性的难度与程序的规模有关。对于较小的程序来说，修改的问题不算大。但对于大型程序设计来说，问题却很严重。随着程序规模不断增大，修改的难度也随之加大。

为了提高可扩充性，应遵循两条基本的原则：

(1) 设计的简明性：一个简单的结构总比复杂的结构有更好的适应性。

(2) 分散化：程序各模块的独立性越好，程序变化时，所影响的模块越少，也就不容易造成连锁反应。

4. 可复用性

指在新的应用中，软件的部分或者全部可被重新使用的难易程度。

在开发不同的软件系统时，从设计到实现，常常会遇到许多相似的部分，为减少或免除由此而产生的重复性工作，所开发的软件或软件部件应该能支持在不同开发中重复地被利用。

可复用性的重要性是很明显的，可复用性对软件质量的其它方面也有影响。如果软件复用解决得好，就可以减少开发软件的工作量，而把更多的精力用于解决其它问题，例如，正确性及健壮性等。

以上论述的正确性、键壮性、可扩充性及可复用性仅仅是软件外部质量因素的关键内容，其它因素还包括兼容性、效率、可移植性、可检测性、完整性以及使用的方便性等。

1.1.2 模块化

模块化是在软件设计及实现中常用的方法。从设计角度讲，模块化降低了设计开发的复杂度并使设计步骤清晰。从软件的质量角度来讲，模块化利于提高健壮性、灵活性、可复用性等。在这里我们不仅仅要讨论程序中的模块，也要讨论设计中的模块。

在讨论中，不只局限于某种具体形式的模块，而主要探讨高质量模块应具有的普遍特性。

1. 模块化的标准

(1) 模块可分解性

一种好的设计方法应能把一个大的、复杂的问题分解为一些小的、简单的问题。通过解决各个小问题来解决大问题。通常这样的分解过程是可传递的。换句话说，各个小问题可以进一步分解。

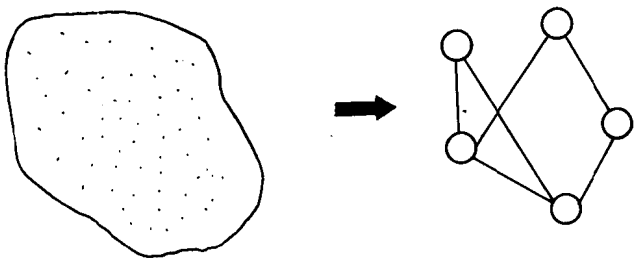


图 1.1 模块可分解性

对于大的系统来说，模块可分解性尤其显得重要。通常大的系统由许多人参加编写，这样就必须保证分解出的小问题能由不同的人员来完成。

目前，广泛采用的结构化设计方法(Structured Design，简称 SD)都是自顶向下进行分解的。结构化设计方法使软件开发人员从系统最抽象的功能入手进行设计，然后一步步进行细化，直到细化后的各个部分可以直接实现为止。整个过程可以表示为一棵树。显然，这种方法满足可分解性的要求。

(2) 模块可结合性

模块可结合性要求不同时期、不同项目、不同环境下设计出的模块应能自由地结合在

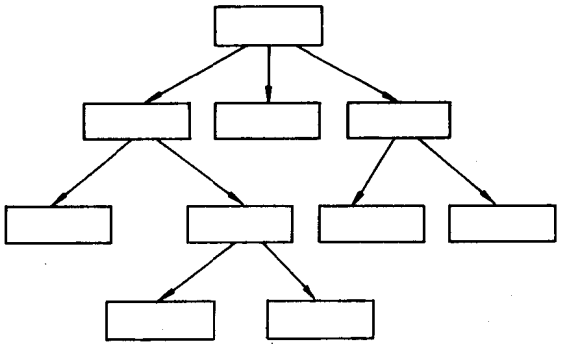


图 1.2　自顶向下分解

一起构成新的系统。如果说模块可分解性涉及的是如何从需求说明中分解出模块的话，那么模块可结合性则涉及如何把已有的模块结合起来构成新的系统。与模块可结合性密切相关的是软件外部质量中的可相容性：其主要目的是寻找好的设计方法，模块在完成特定项目要求的同时，有可能被用于更多的项目中。

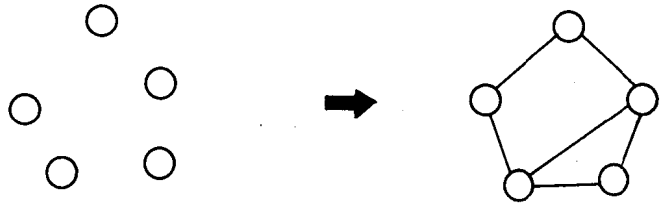


图 1.3　模块可结合性

模块可结合性与模块可分解性有很大不同。在有些情况下二者会发生冲突。例如，传统的自顶向下的结构化设计方法经常不考虑模块可结合性，因此设计出的模块没有很好的复用性。这主要是因为开发出的各个模块只用以完成在细化过程中分离出来的每个具体任务，而没有使模块更具有一般性。

(3) 模块可理解性

如果通过某种方法设计出的每个模块不需要参考相邻的模块，就能被人看懂，那么这种方法符合模块可理解性的要求。最坏的情况是为了理解每个模块而不得不参考相邻的各个模块。

很明显，对软件维护来说，模块可理解性是很重要的。因为，无论是更正错误还是再开发，都必须仔细阅读源代码。

(4) 模块连续性

如果通过某种方法设计出的模块，在需求说明发生变化时只影响一个或者少数几个模块，那么这种方法满足模块连续性的要求。模块发生的变化不应影响系统的结构。

显然，模块连续性与可扩充性密切相关。连续性标准要求小的变化仅仅影响各个独立

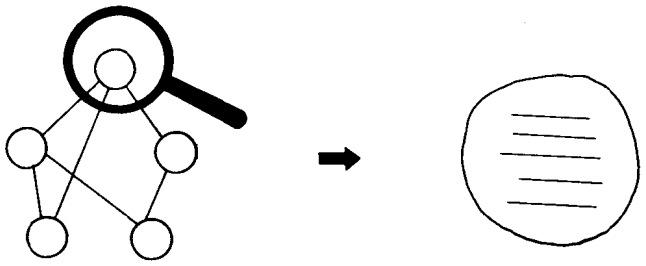


图 1.4　模块可理解性

的模块，而不影响系统自身的结构。

许多常用的语言(C、Pascal 及 Fortran)都提供了符号常量的定义方法。这样，当常量的值发生变化时，受影响的仅仅是常量的定义部分，而其它任何使用常量的地方都不用做任何变化。因此符号常量利于提高模块连续性。

(5) 模块保护

如果通过某种方法设计出的模块，在运行期间发生的错误被限制在这个模块内部或者仅仅传播到附近少数几个模块，那么这种方法满足模块保护的要求。显然，模块保护与健壮性密切相关。

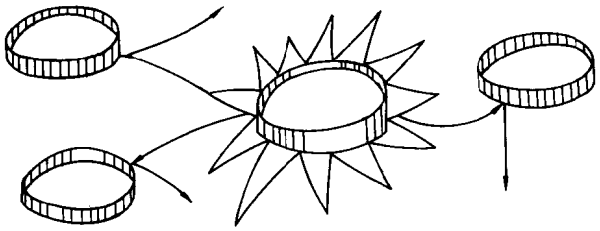


图 1.5　模块保护

以上讨论了模块化的五条基本标准：可分解性、可结合性、可理解性、连续性及模块保护。这五条标准有时会发生矛盾。任何一种完美的设计方法不应该只着重于其中某几条标准而应把这五条标准有机地结合起来。评价某种设计方法时，不应仅仅看利用这种方法是否容易确定系统的模型，更重要的是应当看所设计的模型的综合质量。在保证正确性及健壮性的基础上，应尽可能提高软件的可扩充性和可复用性。

下面针对这五条标准，讨论一下模块化的原则。遵循这些原则有利于达到模块化的标准，从而提高软件的质量。

2. 模块化的原则

(1) 语言模块单元

为了达到可分解性、可结合性及模块保护的标准，设计的模块应与程序设计语言中所提供的语言模块单元相对应。

在把大的系统分解为各个模块时，只有当这些模块与语言的语法单元相对应时，才利

于对各部分分别编译调试，最后连接构成大系统。另外一方面，只有封闭的语法单元才能结合起来构成系统。为了控制错误的影响范围，必须使各模块在语法上有明显的界限。

这一原则说明，如果没有合适的语言支持，很难实现真正的模块化。后面将看到，面向对象程序设计语言 C++中的类/对象便是一个非常好的语言模块单元。

(2) 信息隐藏原则

简单地说，这个原则要求把模块使用者没有必要知道的内容隐藏起来。做个形象的比喻：一台录音机的使用者不必知道其内部电路及传动机构，只要了解其各个开关的用途就可以充分使用它了。只有维修人员及设计者才需要了解其内部电路及机构。

细分起来，模块应由两部分组成：公有部分及私有部分。公有部分指用户在使用模块时直接使用的内容即外功能，主要指模块的接口，或者说是录音机上的开关及按键。私有部分指模块中那些与实现细节有关的内容，如数据结构及具体的算法等，或者说是录音机内部的电路和传动机构。

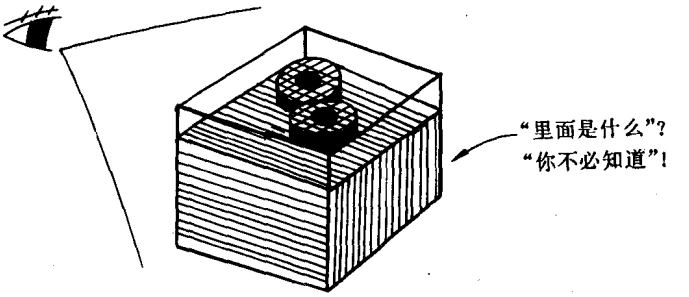


图 1.6 信息隐藏

虽然，设有绝对的规则来确定什么内容可以作为公有部分、什么内容可以作为私有部分，但是有条普遍原则：接口应是模块功能的描述，而私有部分应是这些功能的具体实现。

信息隐藏原则是为了达到模块连续性的标准。如果一个模块的变化(例如算法的变化及数据结构的变化)只影响模块的私有内容，那么此模块的使用者不用做任何变化。而且，一般来讲，接口越小，发生变化的概率越小。在第四章中将详细介绍接口的设计(接口的数量、接口的规模)。

1.1.3 软件复用

软件开发是一件很有创造性但重复性很大的工作。在各种各样的项目中，重复使用相同的算法、相同的数据结构，可借用相似的设计思想及模式。相比之下，设计硬件时，许多功能都可以通过购买集成电路来实现。近年来集成电路的功能越来越复杂、性能越来越高。

如果软件复用解决得好，在开发新软件时，就可以大量借用过去的成果，这样一方面可以缩短开发周期、节省费用，另一方面也可以用更多的人力、物力去研究其它课题，从而在总体上提高软件的质量。

软件复用可以分为以下几个基本层次：代码复用、设计过程复用、分析方案复用。其中代码复用最直接、应用最广。

目前应用最广泛的代码复用技术主要有：源代码的裁剪及例行程序库。

源代码裁剪是最原始的复用方式。由于文字编辑器的功能越来越强，裁剪代码的工作量越来越小，但是在修改过程中引入错误的可能性相当大。另外一方面，对各式各样修改过的代码进行管理是一件很困难的工作。

许多程序设计语言都提供了丰富的例行程序库完成输入/输出、文件管理、数学计算、图形显示、窗口管理等任务。其中数学计算最成功。但是由于例行程序本身的弱点，例行程序的应用受到很大限制。下面我们看一个例子。

例　排序

排序在各种软件中的使用频度是很大的。在科学计算、事务处理、管理信息系统、视图显示等各领域中广泛使用了排序。由于排序的广泛应用，对排序的各种算法也研究得很透彻。但是很难编写一个通用的排序程序。

假设为排序写一个例行程序 sort。为此可以写一个单一的例行程序来处理不同情况，也可以为每一种可能情况写一个例行程序。但是由于情况非常复杂，两种方案都是不切实际的。

对于第一种方案：程序要处理各种各样不同的可能情况。程序中会包含大量的变元，而且程序结构中会出现大量的 case 结构。这样一来从空间及时间的角度讲都是低效的。

对于第二种方案：会产生大量非常相似的例行程序。对于设计者来说，很难使用这些相似之处；用户必须花大量精力去熟悉所有例行程序。

产生以上这些问题的主要原因是，它超过了例行程序的能力。对于相互独立的问题，例行程序库应用起来很方便，但要受到下面一些限制：

(1) 每个问题的规格说明应是简明的，也就是说每个问题用少量参数就可以表达清楚。

(2) 相互独立的问题必须截然不同。例行程序库很难利用问题间的共性。

(3) 不应有复杂的数据结构。复杂的数据结构不得不分散在各个模块中，模块间的相互独立性会因此而被破坏。

鉴于例行程序的这些限制，使代码复用难以得到很好的实现。为了能更好地进行代码复用就必须打破这些限制，为此必须设计一种比例行程序更好的语法模块单元。后面将看到C++程序设计语言中的类/对象就满足了这些要求。

1.1.4　为什么要面向对象

无论通过什么方式实现的软件总是由被加工的对象及有关的功能构成。在开发一个软件的过程中，我们面临着两种选择：一个是把侧重点放在对象上，另一个是把侧重点放在功能上。在过去几十年的软件开发中，我们始终把注意力放在功能上。这是很合理的，因为软件开发出来总应能正确提供所要的功能。

这种基于功能的方法盛行了许多年后，人们渐渐发现了它的许多不足之处。首先，由于程序结构过于围绕事先确定好的功能，使得功能的扩充、删除及修改变得相当困难。这

样的软件结构脆弱、功能集中、耦合度大，很难满足可扩充性、可维护性的要求。而且，由于开发人员的精力过多花在功能的实现上，对今后的复用也没有什么考虑，使复用也很难实现。基于功能的软件之所以有诸多的毛病，主要是因为功能太具体，而且太易变！有人对大量解决具体实际问题的程序中的接口、功能、过程执行顺序、数据及对象的易变性进行了统计比较。结果如下：

接口：极为易变

功能：很易变

过程执行顺序：很易变

数据(从长期讲)：不易变

对象：最为稳定

很明显，对象比功能要稳定得多。

功能的易变性也是有其客观原因的。在软件开发的需求分析阶段，具体的功能都应有哪些、各自的要求是什么等等这些都很难制定得非常合理。有许多人是一边开发，一边增加功能、修改功能，用他们的话说是“走一步，看一步！”另外随着软件的发展，有些老的功能全过时，同时又需要增加一些新的功能，所有这一切都会引起软件的变化。

和功能相比，对象是最稳定的了。无论功能怎样千变万化，一个问题空间中的对象一般总能保持其稳定不变性。显然，围绕对象构造的软件系统也自然会有较好的稳定性，例如：一个高级语言编译器在其发展的各个阶段，总是或多或少地加工处理同一类的对象、源程序、词法标记、符号表、语法树、目标代码等等。但是编译器的外在功能却会有很大变化：编译器可被改写为语法校对器、静态分析器、动态分析器以及整齐格式输出器等。

为了评价一个软件结构(以及设计此结构时使用的方法)的好坏，我们不能只看这个软件结构当初是否很容易被开发出来，更重要的是应看看这个结构是否可以经得住各种变化。从这个角度讲，面向对象优于面向功能。

与这种基于功能的方法相对应的分析及设计方法有许多种。最成功的要算 Yourdon E.，Constantine L.，De Marco T. 等人提出并发展的结构化分析方法。其它的还有 J. D. Warnier 的结构化数据系统开发方法及 Jackson 的 JSD 等。这些方法在过去的十几年中得到了广泛的应用。这些方法非常适用于开发规模较小，生命期较短的软件：用这些方法开发软件时，许多工作都是相当机械的(这也许正是它们成功的重要原因)。但是随着生产及实践的需要，软件的规模不断增大，生命期不断加长，这一切都对可复用性、可扩充性等提出了更高的要求。事实证明，传统的基于功能的、结构化的开发方法已远远不能满足时代的要求。

为此从 80 年代初起，人们不断进行对面向对象技术的研究。目前，面向对象技术的领域有：面向对象的语言、面向对象的程序设计、面向对象的分析、面向对象的设计、面向对象的数据库管理系统及面向对象的基本理论。在本书中，我们以 C++为蓝本，与读者讨论面向对象的程序设计。

1.1.5 面向对象基本概念的介绍

本节将介绍面向对象程序设计中的几个基本概念：封装、继承、多态性和类属性。在论

述中，将从概念产生的背景及概念的实质两方面入手，并举些简单的例子。

1. 封装(Encapsulation)

封装是一种组织软件的方法。它的基本思想是把客观世界中联系紧密的元素及相关操作组织在一起，构造具有独立含义的软件实现，使其相互关系隐藏在内部，而对外仅仅表现为与其它封装体间的接口关系。

封装的目的是信息隐藏。不过，信息隐藏是原则，而封装是针对这一原则的实现。

封装的威力及吸引人的地方在于：在开发一个软件系统时，封装使得构造和组织系统更准确、方便并且能减少重复的工作。如果一个程序员把系统中最易变的部分(例如一个数据结构)封装起来，那么需求的变化(常常不可避免)对整体结构的影响就会非常小。把易变的部分局部化是个很基本的技术。封装把相关的内容组织在一起；使不同部分间的通信减到最少；并且它从需求中分离出更具体的内容。

封装不算一个全新的概念。但是，在面向对象的程序设计中提出了一种全新的封装方法：类/对象。后面我们还将看到类/对象也是构成程序的很好的模块：它满足模块的原则及标准，并且满足代码复用的要求。

对象是按照封装的方法构造的与客观世界具体成分相对应的软件模块。对象中所封装的是描述这些客观世界具体成分的一组数据(称为属性)，及这些数据上的一组操作(称成员函数)。类是对象的抽象及描述，它是具有统一属性和操作的多个对象的统一描述体。在类中必须给出生成一个对象的具体方法。

传统的语言已经提供给用户一些定义好的类，例如 C 中的 int、float、char 等。在生成一个对象时，首先要给出它所属的类，例如 int i。然后系统通过调用相应的过程来生成这个整型的对象 i。一个整型对象 i 的属性是它对应的整值，例如，1，2671，13 等相应的操作有＋，－，＊，/，％等。在使用这些系统定义好的类时，不用关心它内部实现机制是什么；所要关心的仅仅是如何生成一个对象(上面的 i)及与此对象有关的操作都有哪些，换句话说，用户仅仅是个使用者(client)。这正体现了封装性。

面向对象的程序设计语言的重要性在于：它提供给用户定义并实现对象的描述机制。这个过程通常是首先定义一个类 CLASS，然后再声明相应的对象 CLASS an-object。定义一个类时应首先确定表示对象特性的属性，再确定这些属性上的一组操作，见图 1.7。

为了定义一辆自行车，首先要定义自行车类，它的属性有 frame size(车身尺寸)、wheel size(车轮尺寸)、gear(齿轮)，material(材料)和 brand(牌子)。再定义自行车的操作有 shift(变速)、move(移动)、repair(修理)。

```
Bicycle Class:
  Attributes:
    frame size
    wheel size
    gear
    material
    brand
  Operation:
```