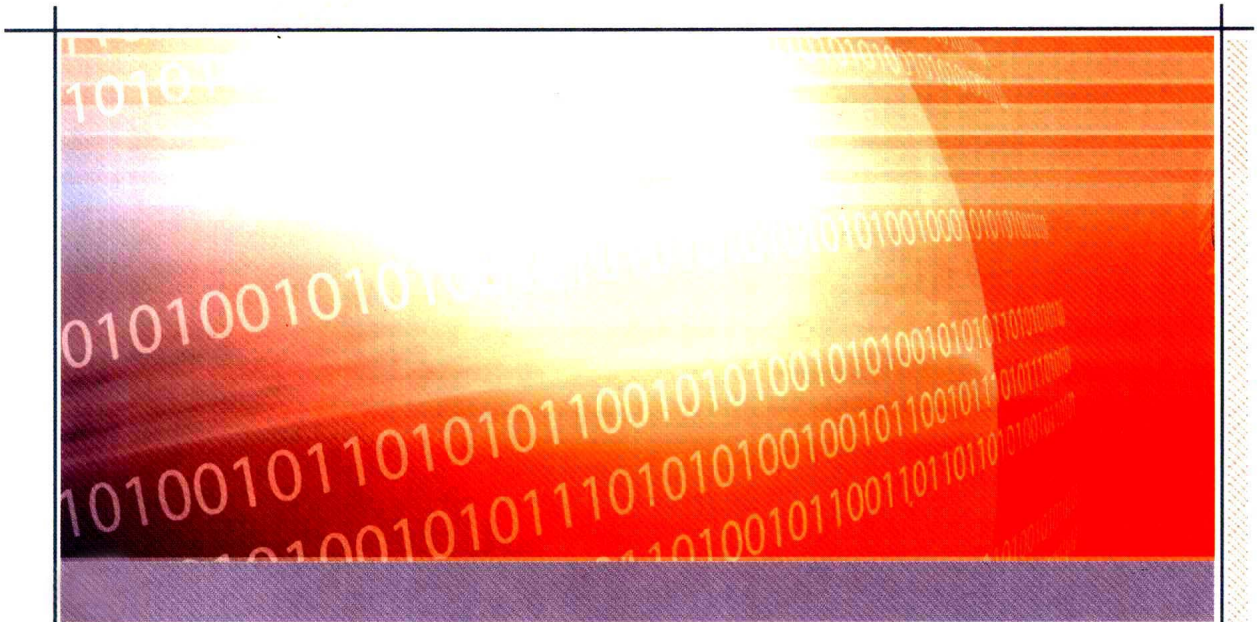
C++
YUYAN JIAOCHENG

（第三版）

C++语言教程

吴祖峰　陈文宇　张松梅　编著



电子科技大学出版社

C++语言教程

（第三版）

吴祖峰　陈文宇　张松梅　编著

电子科技大学出版社

图书在版编目（CIP）数据

C++语言教程/吴祖峰，陈文宇，张松梅编著. —3 版.

成都：电子科技大学出版社，2008.8

ISBN 978-7-81114-761-2

Ⅰ.C… Ⅱ. ①吴… ②陈… ③张… Ⅲ.C 语言—教材

Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2008）第 016912 号

内 容 简 介

C++语言是国内外广泛使用的计算机语言，它保持了 C 语言的简洁、高效，又支持面向对象的程序设计，是目前非常受欢迎的一个面向对象语言，同时也是计算机应用人员应掌握的一种程序设计工具。

本书第一版于 2000 年出版，全书针对学习对象的特点，精心策划，准确定位，概念清晰，例题丰富，深入浅出，受到专家和读者的一致好评。本书共九章，介绍了 C++语言的主要语言特性，并用直观的方法讲述了面向对象的设计技术。

本书适合作为大专院校、培训班和自考班教材，也适合从事计算机软件开发和应用的人员参考。

C++语言教程

（第三版）

吴祖峰 陈文宇 张松梅 编著

出　　版： 电子科技大学出版社（成都市一环路东一段 159 号电子信息产业大厦 邮编：610051）
策划编辑： 吴艳玲
责任编辑： 吴艳玲
主　　页： www.uestcp.com.cn
电子邮箱： uestcp@uestcp.com.cn
发　　行： 新华书店经销
印　　刷： 郫县犀浦印刷厂
成品尺寸： 185mm×260mm　　**印张** 20.25　　**字数** 490 千字
版　　次： 2008 年 8 月第三版
印　　次： 2008 年 8 月第三次印刷
书　　号： ISBN 978-7-81114-761-2
定　　价： 34.90 元

前　言

（第三版）

C++语言是著名的 C 语言的面向对象的扩展，同 C 语言具有完全的兼容性，自发表以来，引起很大的反响，在商业上取得了巨大的成功，是目前应用最广泛的一种面向对象的语言。

学习 C++，不仅要掌握 C++的语言成分，更重要的是要学习一种与传统结构化程序设计完全不同的程序设计语言；因此，本书分为两大部分：面向对象的设计方法和 C++语言的主要语法细节。

本书内容安排如下：第一章介绍面向对象的基础知识；第二章介绍 C++作为更好的 C 的语言特点；第三章介绍类类型；第四章介绍运算符重载；第五章介绍派生类；第六章介绍流库；第七章介绍模板；第八章介绍面向对象技术；第九章介绍命名空间与异常。附录中对 Visual C++集成环境的使用和 C#进行介绍。

本书是在第二版的内容上进行修订而成的，在此感谢第二版的作者陈文宇、张松梅，以及在资料收集、文字审核方面给予大量帮助的曹明生、周整茂、丁玲、黄耀先、胡云鹏。还要感谢电子科技大学出版社的吴艳玲老师，她为本书的出版做了大量工作，才使得本书得以同广大读者见面。

由于作者水平有限，恳请广大读者不吝赐教。

吴祖峰

2008 年 7 月

目　录

第一章 引 论

C++语言是一个面向对象语言，它所支持的面向对象的概念容易将问题空间直接映射到程序空间，为程序员提供了一种与传统结构程序设计不同的思维方式。因此，学习 C++语言面临两个问题，如何建立面向对象的思维方式？如何用 C++语言编程？即需要学习面向对象的设计方法和使用 C++语言的编程方法。

面向对象的设计方法尚在探索中，本章试图从面向对象的目标、面向对象的核心概念、面向对象的思想和方法几个方面，给读者以面向对象概念的总体印象，并通过 CRC（Class_ Responsibility_ Collaborator）方法，介绍一种面向对象的设计方法。在第八章，将进一步叙述面向对象的直观设计方法和设计原则。本书的其余部分讨论 C++语言的具体细节，尽力反映 ANSI C++标准草案的主要内容。书中主要的例子均在 Visual C++ 6.0 上调试通过。

1.1 面向过程和面向对象程序设计方法

1.1.1 面向过程程序设计方法

面向过程的程序设计，也称为结构化程序设计，它的基本思想是：自顶向下、逐步求精。具体来说，就是从问题出发找出解决问题所需要的步骤，然后一步一步地按步骤实现。先做全局性问题分析，然后再把问题分解成相对独立的子问题，最后把每个子问题进一步精确化，直到得到一个计算机能理解的程序。

根据基本思想，面向过程程序设计通常采用模块化设计，即把程序分解成若干个功能独立的模块。每个模块实现独立的子功能。然后以函数为单位实现每个子功能。

但由于面向过程的程序设计是按过程步骤来实现的，它的数据和处理数据的函数是相互独立分离的实体，因而可能出现其他不相关函数访问或者修改某数据的情况，使得数据的隐藏性降低。而同样的处理对于不同的数据类型，必须编写不同的程序，所以可重用性也较低。而且当一段代码被局部修改后，因为面向过程设计是按步骤来的，前后代码之间存在联系，造成其他部分代码也将改变，使软件编码变得复杂。

对于小规模程序，可以用面向过程的程序设计，但程序规模变大到一定程度时，面向过程程序设计的可重用性、可修改性较差，增大了软件编码的复杂性。

1.1.2 面向对象程序设计

面向对象程序设计的出现，解决了随着软件规模以及复杂性增加，面向过程程序

设计所遇到的问题。

面向对象程序设计的基本思想是把数据和处理数据的方法（函数）封装在一个称为类的数据结构中，作为一个相互依存、不可分割的整体来处理，这就不同于面向过程设计数据和处理数据的函数相互分离的情况。一个“对象”就是“类”的一个实例。面向对象设计，把构成问题的事务分解成各个对象，程序是由对象组成的。

面向对象程序设计优于面向过程设计的方面是，它把数据与处理它的方法封装在一起，再通过把数据私有化，使得只有封装在同一个类中的处理它的方法才能访问或者修改它，从而保证了信息的隐藏性，使得数据受到了保护，避免了一些错误的修改或访问，提高了软件的可维护性；另一方面，当相同的处理对不同的数据类型进行处理时，不必编写不同的程序，可以通过面向对象程序设计中的函数重载来实现，增加了可重用性。

面向对象程序设计还有一个特征，就是继承性。它有助于实现代码的重用。例如当产品更新换代时，不必重头编写软件，只需在原有的类的基础上在类中增加新的功能函数，使新的类具有更多的功能。而倘若用面向过程的程序设计，则不能容易的去增加新的功能，因为数据与过程是分离的。

综上所述，面向对象的程序设计克服了面向过程程序设计存在的问题，有利于开发出易于扩展、维护的应用软件。

1.2　C 语言与 C++语言

1.2.1　C 语言

C 语言是目前使用最广泛、最流行的高级程序设计语言之一。C 语言从一开始，就是作为 UNIX 操作系统的开发语言而闻名的。目前 UNIX 和 Linux 操作系统都是用 C 语言来编写的。

C 语言是一种面向过程的语言。它具有以下一些特点。

（1）C 语言简洁、方便、灵活。它只有 32 个关键字。

（2）数据类型和运算符丰富。具有整型、字符型、数组类型、指针类型、结构、联合等数据类型，适用于实现复杂数据结构。括号、赋值等运算符的灵活使用，可以实现复杂的运算。

（3）C 语言允许直接访问内存地址、通过控制计算机寄存器端口等硬件资源可以直接对硬件操作。因此它是最适合编写操作系统的语言。

（4）可移植性好。C 语言源代码经过编译后的可执行程序可在不同操作系统上运行。

1.2.2　C++语言

C++是由 C 语言扩充而来的。它包含了 C 语言所有的特点。因此，一个 C 程序也是一个 C++程序。

C++不是抛弃了 C 语言的传统，而是继承和发展了 C 语言丰富的数据类型、运算符，可以直接访问内存地址等优点。

除此以外，C++最重要的一点是引入了类这一构造数据类型，用它来实现面向对象的程序设计。这也是 C++与 C 语言的主要区别。类把数据与处理数据的方法封装在一起，数据被限定在类的范围内，使它对类外部不可见，实现了信息隐藏性。对象就是类的一个实例。C++程序把任何事物都看成对象，程序就是由对象组成，简单的对象经过组合成为复杂的对象。而客观世界就是由各种对象组成的，C++面向对象的思想，使程序设计与人们现实世界的客观规律及人们普遍的思维方式一致。

C++还具有类继承的特征，使程序员可以重复利用以前的类来编写高层次的类，提高了编写程序的效率。

C++支持函数重载，可以使用相同的函数名来处理不同的数据类型，提高了软件的可读性和可理解性。

所以学习 C++语言，不但要学习本身语法，更重要的是要掌握面向对象程序设计方法。

1.3 面向对象的目标

面向对象语言的所谓“对象”，最广泛的解释是将某组数据和使用该组数据的一组基本操作或过程封装在一起，而将此封装体看作一个实体，如图 1-1 所示。

面向对象的基本想法就是把要构造的系统表示为对象的集合。这里所说的系统，不仅应考虑为程序和软件系统，而且应广泛解释为计算模型、CAD/CAM 中的成分模型以及人工智能中的知识表示形式等。

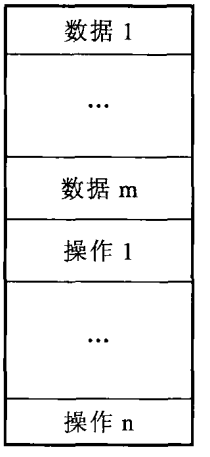


图 1-1 对象的模型结构

面向对象的思想与解决计算机软件面临的两大课题有关。一个问题是，怎样克服软件的复杂性；另一个问题是，怎样将现实世界模型在计算机中自然地表示出来。

客观世界的问题都是由客观世界的实体及其相互间的联系构成的，我们把客观世界的实体称之为问题空间的对象。显然，“对象”因问题的特殊性是不固定的，一本书可以是一个对象，一家图书馆也可以是一个对象，每一个对象都有自己的运动规律和内部状态，不同对象之间的相互作用和互相通信构成了完整的客观世界。因此，从思维模型的角度，面向对象很自然地与客观世界相对应。

从软件的角度，计算可以看作仿真。考虑一个雇员数据库，库中每个记录都代表一个雇员，可以说这是对公司的某些方面的一种仿真或模拟。同样地，一个操作系统的数据结构常反映了真实世界中某些对象的状态。例如，可以用来反映这样一个事实：一个磁带机正在回绕或正处于奇偶校验状态，或者系统中的磁带机正分配给某个

特定的作业。

如果每个被仿真的实体都由一个数据结构来表示，并且将相关的操作信息封装进去，仿真将被简化，可以方便地刻化对象的内部状态和运动规律。面向对象就是这样一种适用于直观模型化的设计方法。这意味着系统设计者从现实世界所得到的图像，或设计者头脑中形成的模型里所出现的物理图像与构成系统的一组对象之间有近乎一对一的对应关系。这一思想非常利于实现大型的软件系统。

作为解决软件复杂性问题的手段，在面向对象中，利用了如下对象的性质：

（1）将密切相关的数据和过程封装起来定义为一个实体；

（2）定义了一个实体后，即使不知道此实体的功能是怎样实现的，也能使用它们。

这相当于软件工程和程序设计方法论中的抽象化、抽象数据类型和信息隐藏等概念所具有的性质。它把系统中所有资源，如数据、模块以及系统都看作对象。每个对象把一组数据和一组过程封装在一起，这组过程对这组数据进行处理。使用这一方法，设计人员可以依照自己的意图创建自己的对象，并将问题映射到该对象上。这一方法直接、自然，可以使设计人员把主要精力放在系统一级，而对细节问题可以较少的关心。

使用对象将信息局部化，并使程序结构与设计结构相吻合的优点，有利于在完善和维护阶段对软件进行修改，也有利于其他人（非设计人员）来清除软件错误。程序员容易确定程序的哪些部分是否依赖于正要修改的片断，而且正在修改的部分对其他部分影响很小。这对大型、复杂软件的维护和改进是很重要的。

面向对象设计非常注重设计方法，因为它要产生一种与现实具有自然关系的软件系统，而现实就是一种模型。实际上，用面向对象方法编程的关键是模型化。程序员的责任是构造现实的软件模型。此时，计算机的观点是不重要的，而现实生活的观点才是最重要的。

因此，可以将面向对象的目标归纳为：对试图利用计算机进行问题求解和信息处理的领域，尽量使用对象的概念，将问题空间中的现实模型映射到程序空间，由此所得到的自然性可望克服软件系统的复杂性，从而得到问题求解和信息处理的更高性能。

在进一步叙述面向对象设计方法之前，先了解一下面向对象的几个核心概念。

1.4　面向对象语言的核心概念

在问题空间中，将客观世界的实体称为对象，不同对象之间的相互作用和互相通信构成了完整的客观世界。如何将问题空间的这一思维模型直接映射到程序空间，也就是说，面向对象语言提供什么概念来支持这一思维模型？纵观诸多面向对象的程序设计语言，最核心的概念是数据封装、继承和多态性。

1.4.1　数据封装

使用传统设计方法的程序员往往有着共同的弱点。用一个较为夸张的例子来说，当人们问及这些程序员时间的时候，他们会详细地解释钟表的概念。这是由他们使用

计算机的经验所致。这里，必须搞清楚的概念是，告诉别人时间和让他人理解时间的概念是两个不同的知识领域。比方说，某人有一块手表，并常常告诉别人现在是几点了，这并不等于说他知道这块表的内部工作原理。手表是一个对象，只需它提供时间状态，无需了解它如何运转。数据封装的概念反映的就是这一简单原理。

数据封装将一组数据和与这组数据有关的操作集合封装在一起，形成一个能动的实体，称为对象。用户不必知道对象行为的实现细节，只需根据对象提供的外部特性接口访问对象。例如，可能使用一个数组来存储在屏幕上画一个字符所需要的字形信息：int font [num]；如何显示、缩小、放大、增加亮度和设置字符颜色呢？在传统 C 语言中，常用的解决办法是，将数据结构和相关的函数放入一个可编译的源文件中，把数据和函数作为模块看待。这当然是朝正确的方向迈出了一步。但这还不够好，在数据和函数之间没有明确的关系。若不用上述提供的相关函数也能直接操纵 font 的数据，这样可能导致一些问题。假如你决定用链表取代数组，而另一位做同一工作的程序员则可能认为，他有更好的方法来访问这些字形信息，于是他编写了一些自己能直接操纵这些数组的函数。可问题是，那里再也没有数组了！当程序联调的时候，将产生错误。实际上，在数据 font 和操纵它的函数（显示、缩小、放大、旋转等）之间存在着明确的关系。数据 font 以及这些函数的实现细节对使用者并不重要，重要的是这些函数的界面，使用者只需根据这些函数的界面（函数名及其参数）和函数的功能进行访问。那么，在设计、实现、维护和重用程序时就有很大的帮助。

C++通过建立一个合适的 Font 类，将字形数据（称为数据成员）和函数（称为成员函数）结合在一起，形成一个新的类型，称为类类型（class）：

```
class Font
    {
        private:
            字形数据;
        public:
            操作字形数据的成员函数;
    };
```

在这样建立的类 Font 中，能确保私有的字形数据只能由类中的成员函数进行访问和处理。在任何时候，都可以自由地把字形数据从数组变为链表，只需改变成员函数的实现细节。由于这些成员函数的界面不改变，系统其他部分的程序（及使用者）就不会由于改动而受到影响。

类的概念将数据和与这个数据有关的操作集合封装在一起，建立了一个定义良好的接口，人们只关心其使用，不关心其实现细节。这反应了抽象数据类型的思想。

1.4.2 继承

继承是面向对象语言的另一个重要的概念。在客观世界中，存在着整体和部分的关系、一般和特殊的关系，继承将后者模型化。举一个比喻的例子。昆虫学家对昆虫的分类如图 1-2 所示。

在试图对一些新的动物或对象进行分类以前，关心的问题是：它与其他一般的类有多少相似之处？差别有多大？每一个不同的类都由一组描述它的行为和特征组成，最高层（根结点）是最普遍的，问题也最简单：有翅膀还是没翅膀？每一层都比它之前的层更具体。一旦某个特征定义下来，所有在它之下的种类都要包含该特征。所以，一旦确定苍蝇为昆虫有翅类中的一种，就不必指出苍蝇是昆虫，有一对翅膀，因为苍蝇从它的目中继承了这个特征。

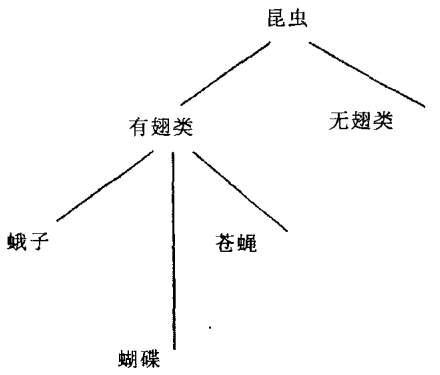


图 1-2　昆虫分类图标

在面向对象语言中，类功能支持这种层次机制；除了根结点外，每个类都有它的超类（superclass），又称为父类或基类。除了叶结点外，每个类都有它的子类（subcalss），又称为派生类。

一个子类可以从它的基类那里继承所有的数据和操作，并扩充自己的特殊数据和操作。基类抽象出共同特征，子类表达其差别。有了类的层次结构和继承性，不同对象的共同性质只需定义一次，用户就可以充分利用已有的类，此概念符合软件重用的目标。

1.4.3　多态性

面向对象的另外一个核心概念是多态性。

所谓多态，是指一个名字，有多种语义；或一个相同界面，有多种实现。

下面我们来考察多态性问题的一个类比问题。当一汽车司机为避免撞车时刹车，他关心的是快速刹车（效果），而不关心刹车是鼓式刹车还是盘式刹车（实现方法的细节）。这里，刹车的使用与刹车的结构是分离的概念，可能有多种结构的刹车，它们的使用方法是相同的。相同的使用方法（相同界面）对应于不同种类的刹车结构（多种实现），这反映了多态性的思想。

与此类似，用户在使用函数编程时，他关心的是该函数的功能及其使用界面，并不需要了解该函数的使用界面与函数的哪一种实现方法相匹配（binding）。也就是说，在设计这一级上，软件人员只关心“施加在对象上的动作是什么”，而不必牵涉“如何实现这个动作”以及“实现这个动作有多少种方法”的细节。在面向对象语言中，重载（或称为超载）表达了最简单的多态性。比较容易理解的是函数重载。例如：

```
void   fun(int,int,char);
void   fun(char,float);
void   fun(int,int);
void   fun(float,float);
```

这几个函数都具有相同的函数名 fun，但其参数不同，它们的函数体也可以完全

不同，编译能根据其函数参数的不同而自动选择相应的函数体进行匹配。因此，一个函数名代表了多种函数的实现（函数体）。

对于函数重载，若函数调用（界面）与哪一个函数体（函数实现）相匹配，是在编译时确定的，称为早期匹配（early binding）。如果函数调用与哪一个函数体的匹配是在运行时动态进行的，称之为晚期匹配（lately binding）。一般来说，早期匹配执行速度比较快，晚期匹配提供灵活性和高度的问题抽象。

C++中的虚函数提供了晚期匹配带来的良好特征。函数重载强调的是函数名相同，函数参数和函数体不相同（编译能根据参数的差别进行识别和匹配）。虚函数则强调单界面多实现版本的方法，亦即函数名、返回类型、函数参数的类型、顺序、个数完全相同，但函数体可以完全不同，这在编译阶段是无法识别的，只能由系统在运行时刻动态地寻找所需的函数体进行匹配。

在 C++中，通过继承关系，基类及其派生类之间构成一个树形结构（多重继承为图结构），树中的每个类（基类或派生类）都可以说明一个具有虚特性的函数，称为虚函数。那么在这个类及其派生类中都可以定义这个虚函数的不同实现，但要求这些不同实现必须遵守相同的函数界面，否则虚特性丢失。使用时，系统能在运行时刻动态地寻找所需的实现版本。

C++语言中的继承机制和虚函数使得软件可扩充性更为自然。可以继承基类拥有的所有特性，然后加进派生类所需要的特殊的东西，使派生类对象以相似的方式工作并具有新的功能。派生类定义的类型及其虚函数版本是有规则的功能等级的真正扩充。由于这是语言设计的一部分，不是编程时才有的想法，所以很容易实现软件的可扩充性。

1.4.4 类属

类属并不是面向对象语言特有的成分，由于它的重要性以及许多 C 程序员对它很陌生，故在这里介绍这个概念。

类属的概念来源于 ALGOL 68，是 ADA 语言中的核心概念。在 ADA 语言中，类属最重要的形式是：类型参数化，即参数化一个软件分量的能力。

实际中有这样一些程序，从它们的逻辑功能（或算法）看，彼此是相同的，所不同的主要是处理对象（数据）的类型。如果提供具有相同逻辑功能的程序正文（保存相同性），然后将数据类型作为参数传递（指出不同性），这就是类属机制的思想，又称为参数化模板。例如，一个典型的类属模板是

```
Binary_Tree [T]
```

其中，类属的形参 T 代表二叉树元素的数据类型。现在，可以用这个单独的类属模板 Binary_Tree [T]来代替许多非常相似的模块。例如：

```
Binary_Tree[Integer]        /*  说明一棵二叉树，其元素为整型      */
Binary_Tree[float]          /*  说明一棵二叉树，其元素为浮点型    */
```

这里，类属模板 Binary_Tree[T]与实际模块 Binary_Tree[Integer]和 Binary_Tree[float]非常相似，其差别在于类属形参 T 适应了不同的类型：Integer 和 Real，后者称为前