# C++语言和面向对象程序设计

宛延闿

清华大学出版社

# C++语言和面向对象程序设计

宛延闿 编

清华大学出版社

## 内容简介

C++程序设计语言和面向对象程序设计代表了旨在使计算机问题解更符合人的思维活动，这是一场软件开发方法的革命。全书共分八章。第一章概论；第二章C语言；第三、四章是C到C++的过渡；第五、六、七章分别介绍数据封装和隐藏、继承和导出类、多态性和虚拟函数；第八章给出综合的四个有代表性的C++和面向对象典型实例分析。

本书由浅入深围绕面向对象的主要特征为主干线，通过大量典型例子循序渐进地介绍C语言、C++语言和面向对象程序设计。适用于广大软件工作者，是大专院校学生和研究生的教科书，也是学习C、C++及面向对象程序设计的科技人员较全面的参考书，同时本书也可以作为现代软件工程课程的补充教材。

**(京)新登字158号**

**C++语言和面向对象程序设计**

宛延闿 编

☆

清华大学出版社出版

北京 清华园

北京昌平环球印刷厂印刷

新华书店总店科技发行所发行

☆

开本：787×1092 1/16 印张：18.5 字数：436千字

1993年6月第1版 1993年6月第1次印刷

印数：00001—13000

ISBN 7-302-01192-3/TP·447

定价：11.70元

# 前　言

面向对象问题解和面向对象程序设计代表了新颖的计算机程序设计方法和思维方法，此方法与通常的结构程序设计方法十分不同。面向对象语言具有一个强有力的特征：支持一种旨在使得计算机问题解能更符合人的思维活动的概念。人们能够利用C++语言充分挖掘硬件潜在能力，并能在减少开销的前提下，提供更强有力的软件开发工具。

C++是一种混合性语言，它既具有独特的面向对象特征，又保留传统的高效结构语言C的主要特征。C++提供给程序开发者的面向对象能力，而又不失去内存运行效率，并能在普通计算机硬件上产生高质量的软件产品。

C++早在1980年就由AT&T贝尔实验室开发，迄今仍在演变中。C++是包含支持面向对象程序设计和C的一个超集，C++面向对象语言全面支持数据抽象、数据封装、继承性和多态性，C++语言比支持面向对象语言Ada和Modula-2更前进了一大步，并保留了C程序设计语言（编写UNIX操作系统的主语言）的简洁性和高效性，C++语言代替C语言是肯定的事情，特别在大型程序设计软件项目和大型软件系统方面，而且在中、小微机和超级微机上也都很适用。

本书的所有概念均通过许多细致的结构以及充分的测试程序而得阐明。经验告诉我们：一个好的优选的典型实例能帮助读者澄清与C++的高级特征和面向对象程序设计相关的困难概念，事实上，典型实例本身允许读者通过在实际应用中的理解和研究，更深刻了解C++和面向对象问题解及其优越之点。

鉴于编者水平有限，错误之处在所难免，欢迎同行们给以批证指正。

希望本书能对我国软件开发和新的程序设计方法的革新方面起抛砖引玉作用。

本书在编写过程中，得到了许多专家的指正和本单位领导、同行的支持，在此一并表示感谢。

宛延闿
北京计算机应用和仿真技术研究所
一九九一年于北京

# 目　录

# 第一章 绪 论

## 1.1 综 述

计算机软件开发一直被两大难题所困扰：一是如何超越程序复杂性障碍；二是如何在计算机系统中自然地表示客观世界，即对象模型。由C++语言写的面向对象程序设计是软件工程学中的结构化程序设计、模块化、数据抽象、信息隐藏、知识表示、并行处理等各种概念的积累与发展，是解决上述两大难题的90年代最有希望、最有前途的方法。面向对象程序设计是软件开发方法的一场革命，它代表了新颖的计算机程序设计的思维方法。该方法与通常结构程序设计十分不同，它支持一种概念，即旨在使得计算机问题的求解更接近人的思维活动，人们能够利用C++语言充分挖掘硬件潜在能力，在减少开销的前提下，提供更强有力的软件开发工具。

面向对象程序设计是软件系统的设计与实现的新方法。这种新方法是通过增加软件可扩充性和可重用性，来改善并提高程序员的生产能力，并能控制维护软件的复杂性和软件维护的开销。当使用面向对象程序设计方法时，软件开发的设计阶段更加紧密地与实现阶段相联系。在软件设计与实现中当今有许许多多方法，面向对象方法是在实践中超越其他许多方法的潜在的最有前途的方法。虽然在各个应用领域中报道了面向对象程序设计的巨大成功，但就一般的软件设计和开发舞台来评价面向对象方法学的成功尚为时太早。面向对象方法学包含了分析、设计和实施的面向对象方法。这部分是当今最弱的部分，面向对象对软件系统开发起着关键作用。本书的目的是介绍软件开发崭新的方法和C++语言，C++语言非常适合于面向对象程序设计。

C++这个名字表示了从C语言进化的特征。“++”是C的增量操作符，顾名思义，C++是C的扩充，即C++语言在提供面向对象程序设计的同时保留C语言的高度简洁和高效率等优点。我国已法定用UNIX操作系统，而UNIX操作系统90%的代码用C语言编写，C语言具有高效灵活、易于理解、可移植性好等优点，C++语言不仅保留这些优点而且包含了几乎所有面向对象特征，C++将代替C是肯定的。所以，本书为了学习与研究的一致性和连贯性，第二章将简洁地介绍C语言知识，对于非常熟悉C语言的读者可跳过此章。

本书的逻辑梗概是：绪论→C语言→C++语言→面向对象特征→典型实例剖析。

## 1.2 面向对象程序设计

面向对象程序设计中心是围绕几个主要概念：抽象数据类型和类，类型层次（子类），继承性和多态性。类和继承是符合人们一般思维方式的描述模式。

什么叫对象（object）？

对象通常作为计算机模拟思维，表示真实世界的抽象。一个对象像一个软件构造块，

它包含了数据结构和提供相关的行为(操作)。对象本身可为用户提供一系列服务，可以改变对象状态、测试、传递消息等等，用户无需知道服务的任何实现细节，操作完全是封闭的。

抽象数据类型是面向对象程序设计的中心概念之一。一个抽象数据类型是一个模型，此模型包含一个类型和与之相关的操作集。定义这些操作集的是基本类型的行为。

在大多数面向对象语言中，类的定义描述以抽象数据类型为基础的对象行为，抽象数据类型定义了能以类型为基础执行所有操作的接口。类定义也指定了实现细节或类型的数据结构。通常这些实现细节仅在该类范围内存取，我们称这种类型为私有(private)类型。当数据类型的全部或部分在该类范围外存取，我们称这样的类型为公共(public)类型。

按面向对象的说法，为一个类而定义的所有操作称之为方法(Methods)。这些方法类似于非面向对象语言中的过程和函数。如果一个类的 Public 操作能在许多应用领域中被充分应用，那么可以说类是组成可重用软件程序块的基础。

一个对象(object)是被一个特定类说明的变量。这样一个对象通过包含在类定义中的所有域的拷贝(private 和 public 两部分)来封装。通过访问一个或几个在类定义上的方法，可以对这个对象实施各种操作。引用一个方法的过程称之为向这个对象发送一个消息(Message)。一个典型的消息包含的一些参数，正像在非面向对象语言中过程或函数调用(或引用)的一些参数一样，一个典型引用方法的操作是修改存储于特定对象中的数据。

每个类变量或对象表示该类的一个实例(instance)。如果几个对象被定义具有同样的类，它们将是包含有不同值的一个集合。

面向对象方法通过子类提供类型的等级层次。一个子类(subclass)定义一个对象集合的行为，该对象继承父类(parent class)的各种特征，子类反过来又将它自己的或继承来的特征传递到它的子类中。借助允许建立子类的方法，可使软件开发的周期缩短，从而降低开发复杂性和费用。

子类能导致增加问题解的能力。用基本类的集合建立子类代替修改现存的软件，或坏的软件，或重写的软件。这些新的子类的对象组成了软件构造的从属结构。

## 1.3 面向对象问题解

一个面向对象的软件系统的结构建立在一系列类上，这些类刻划了系统中所有要处理的基本数据类型的行为。每个类的许多对象可通过引用该类的方法来操纵，即向这些对象发送消息，这些消息表示了可在这个类的对象集合上发生的操作(活动)。

面向对象程序设计集中在对数据操作而不是对过程操纵。数据构成了软件分解的基础。的确，面向对象软件设计的主要挑战是将软件系统分解为基本数据类型或者类和子类，以及对每个基本类和子类特性的定义。对象即类(或子类)变量相应于实际问题领域中的物理的或逻辑的实体。

定义一个面向对象软件系统的结构框架和构成一个高层系统设计，最终只表现为一系列类(子类)、它们的定义和对象。用方法接口描述每个类的行为特征。这个方法的实现

细节不是系统高层设计所关心的部分。

在典型的面向对象语言中，类中定义与方法的接口可以和实现细节的定义分开，允许系统的设计和系统实现分开。概念（与方法接口有关的类定义）和它的实现（实现这个类的数据结构和算法的代码）的分离在面向对象的程序设计中是非常重要的。

概念和它的实现的分离在构成重复使用性以及控制维护的花费上也是很重要的。

可重用性在面向对象程序设计中十分重要，其原因为在类中的封装概念由方法接口形式提供。用户仅需要了解类的对象作为在方法接口上的特定行为，而无需关心他们的实现。从用户观点来说，方法实现被包含在隐藏细节的“黑盒”中。

在面向对象程序设计中可维护性也很重要。这是因为改变数据结构或算法（即实现类的代码）仅局限于实现这个类（或类的一部分）的代码区域内。这对程序的维护提供了方便。

面向对象软件开发的主要目的是：

- 用可重用软件分解（基于基本类）和用子类加快问题求解，缩短开发时间和减少软件开发费用。
- 通过改变一个或多个类的实现，使其影响局部化，从而降低软件维护的花费。

面向对象系统的可靠性可以得到增强，因为初始设计阶段就注重高层的整体性，构成系统的主要部件从一开始就被正确组合在一起。每个主要部件是用其抽象特性定义，高层集成整体性的测试在许多低层详细设计之前已经设计好或已实现了，这些贡献改善了可靠性。

面向对象程序设计提供了一个快速原型可使用平台。在软件系统高层分解进入类并从这些类的对象被完成之后，许多重要方法（赋予系统行为）能快速简单地实现，最后可达到细节的完美实现。

## 1.4 类、对象和封装

一个类描述涉及到定义该类任一对象的一个实例的所有行为特征。实际上，一个类定义的是一种对象类型。

一个类定义的私有部分通常定义了以数据类型为基础的数据结构。仅仅在该类范围内部可以存取指定的方法，这些方法经常支持公共（public）方法的实现。有时一个类的私有部分可以包含其他类的对象，这个其他类仅在给定类的范围内部才能被操纵。

一个类的公共部分通常指定对方法的接口，在许多交叉应用领域中它们构成了这个类的可重用性的基础。这些方法能引用到该类的范围之外，当然是用发送消息到给定类的对象的方式。

封装是用于被定义的各个软件对象的过程中。封装定义为：

1. 所有对象内部软件范围具有清晰的边界；
2. 描述该对象与其他对象如何相互作用的一个接口；
3. 受保护的内部实现。该实现给出了由软件对象提供的功能的细节，实现细节不能在定义该对象的类的范围外进行访问。

封装概念不仅涉及到类的描述，而且如何将问题解的各个组件组装在一起的求精过程。封装的单位是对象，该对象的特性由它自己的类说明来描述。这些特性为相同类的其他对象所共享，对象的封装比讲一个类表示的封装更具体化。有了封装这个定义，一个类的每个实例(instance)在一个问题求解中是一个独立的封装，或称作组件(问题解的分量)。

下例介绍类、对象和封装。在此简要研究一下二叉查找树(a binary search tree)的抽象(详见本书第五章)。

在最高层上，两个对象由基本表示二叉查找树来标识。这些对象的第一个是由称之为类树的实例来表示，因为二叉查找树是由许多节点组成，对象的第二个用称为树节点类的实例来表示。

树节点类赋于下述特点：

- 树节点类的实例是由二叉查找树构成的对象
- 每个树节点包含了确定在结构中数据排序关系的一个对象
- 一个二叉查找树包含了指向左子树和指向右子树的许多对象

在树节点类的对象定义的方法为：

- new_node　建立一个树节点类的新实例
- key　在确定树排序的节点中，返回数据存储域
- left　返回给定节点的左子节点
- right　返回给定节点的右子节点

以下图 1.1 描绘了树节点类。

```
Class Tree  Node
    private
      object ordering relation
      object left_subtree
      object right_subtree
    public
      method new_node
      method key
      method left
      method right
```

图 1.1　树节点类的描述

树类有以下特征：

- 树类的实例是二叉查找树
- 一个二叉查找树是以称为根节点的一个具体节点及由许多树节点组成，对树对象的所有存取均通过根节点进行
- 在二叉查找树的节点均相对于某些关键对象排序，在左子树中具有较小键值而在右子树中具有较大键值
- 在二叉查找树的节点的插入和删除必须产生二叉查找树的一个对象

在树类的对象定义的方法为：

- define　　建立一个新树
- insert　　在树中插入一个新节点
- remove　　从树中删除已存的节点
- is_present　确定是否是树中一个具体节点
- display　　在树中按序输出节点的集合

图 1.2 描绘了树类

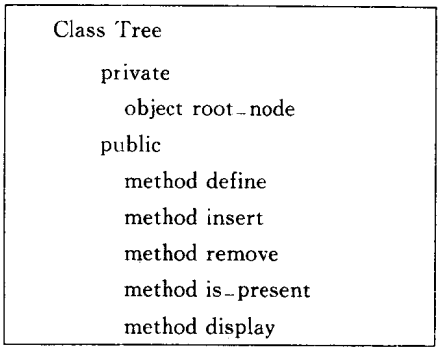


图 1.2　树类的描述

## 1.5　子类——继承性和多态性

我们现在介绍类层次概念。在层次体系中，某些类是从属于其他类，或称为子类，在 C++ 中称为导出类(derived classes)。子类是被认为层次体系组合下的类的具体情况，通常在类层次体系中的下层表示一个增加的详细说明，而高层通常表示更高的概括。

在大多数面向对象语言中，如果类 P 是子类 S 的一个父辈，则子类 S 的一个对象 s 能使用父类 P 的一个对象 p。这蕴含着消息(即操作)公共集合能发送到类 P 和类 S 的各个对象。当同样消息能被发送到父类的对象和它的子类的对象时，我们把此定义为多态性(polymorphism)。

多态性允许每个对象响应公共消息格式，即用合适的方式从一个对象取来送到子类对象去。例如，一个打印方法可定义输出到赋予一个对象状态的某个数据域。多态性的相同消息 print，被送到一个类和它的子类里面的所有对象，这样，每个对象知道如何响应这个消息，也允许用与其他子类对象的不同方式响应。例如，不同数据域可以从每个不同的子类的对象输出。在不同类型对象上，对一个类似的操作，使用相同消息的能力是与问题解的人的思维方式相一致的。对打印整数、浮点数、字符、字符串和数据记录等使用不同的术语是不自然的。多态性是对问题解的面向对象方法中的关键特征之一。

不同的面向对象语言提供了一个能力范围，此能力涉及到一个子类的对象的规模，即可继承、可扩充或取而代之父类的特征等。某些面向对象语言支持多重继承，即在一个子类中可多于一个父类。这些内容在后面几章将陆续介绍。

## 1.6 面向对象程序设计的挑战

面向对象程序设计的挑战的某些方面是，通过寻找共通性把软件系统划分为类，建立子类来处理具体个性，这样一步步对现存的软件系统构造出一个合适的类和子类的层次体系结构。

### 1.6.1 划分软件分类

对初学的面向对象程序员，用类设计建立一个程序，似乎是不自然的，也比较困难。例如，写一个 rollbook 数据库程序，在此程序中有存储、修改和删除学生的名字和成绩，以及打印这些学生的名字和成绩单。典型的方法是定义 rollbook 的数据类型，然后定义插入、删除、修改和打印方法作为以 rollbook 数据类型为基础的相联系的操作。

插入、删除和修改操作可以用比 rollbook 的类型更合适的与过程相联系的 edit 来代替，这样过程的 edit 能说明为一个类，这个 edit 类能在其他要求数据库信息联机交互编辑应用中重用。rollbook 类能包含 edit 类的一个对象，去完成插入、删除和修改学生的名字和成绩单。

### 1.6.2 对已存的软件系统增加功能

在面向对象程序中每个函数或过程是与一个类相联系的。对已存在的软件系统增加一个新功能，那么设计者或程序员必须决定是否增加一个特殊过程到已存在的类中，或定义一个新类或一个子类。

可重用性问题是决定新功能应当加到面向对象系统什么地方。如果增加的功能从几个已存在类中被对象重用，则有理由基于函数或过程建立一个新类。如果新功能的实现要求访问已存的类型的内部细节，则有理由增加新功能作为添加方法加到现存类中。如果新功能表示对一个给定类中一个现存函数(方法)的修改，则有理由作为一个方法去建立一个子类和增加功能到这个子类中。这个方法就取而代之父类中的类似的方法。

### 1.6.3 类型和子类型的层次结构

建立类和子类的一种方法是自顶向下数据类型分解，在分解中程序员应在类的系统中辨认并按模型建造主要数据元素。从高层的数据分量(类)一层一层地划分为更专门的子类，此过程一直继续到类和对象的层次结构构造完成为止。

作为一个例子，考虑作为系统中主要类的一个汽车，汽车类可以划分成子类发动机，变速器，闸，教练设备，排气装置，悬置等。子类发动机，像其他子类一样，能进一步划为点火装置，燃料喷射装置和起动装置。点火装置子类也能进一步划分为火花活塞和螺线管。对每个这样的类和子类，必须定义相应的操作(方法)集合，汽车类分解图的一部分由图 1.3 示出。

另外一个方法是建立类和子类的所谓自底向上分解，在此分解中许多子类生长(扩充)成一个父类，并对每个相继的子类的对象提供更大的功能。

通常采用自底向上的分解例子是一个屏幕窗口类(screen_window)作为一个父类。这样一个类概括了所有屏幕窗口的最小特性。我们能构造一系列子类,对基本的 screen_window 类加上许多功能,并具有各种各样子类的对象,诸如作为前景和后景颜色的属性、窗口边界的限定,等等。对 screen_window 的部分类分解在图 1.4 描绘。

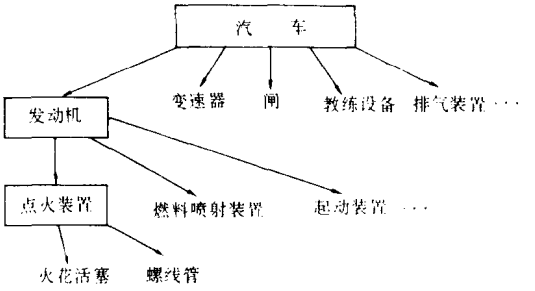


图 1.3 汽车部分类分解图

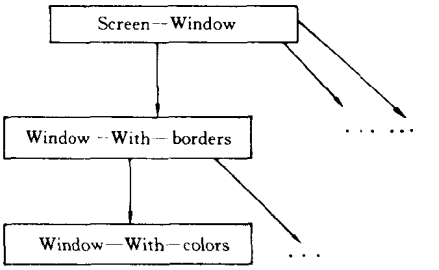


图 1.4 屏幕窗口部分类分解图

通常是两种方法共同使用,即通过从一个父类开始逐步建立子类,当软件工程师了解到几个不相同的子类的之间关系后,再抽象出它们的共有特征到一个或几个父类中,如此反复从而设计出较完美的面向对象程序。

### 1.6.4 应改变典型软件开发过程

当今典型软件开发包含三个独立活动,在它们之间有高墙隔开。

(1) 第一个独立活动是分析和设计活动,它使用各种概念模型,如 ER 模型和 SASD 模型。此活动是把现实世界用 ER 模型描述并进行分析和设计。

(2) 第二个独立活动是程序设计。它使用不同的程序设计语言(如 COBOL,FORTRAN,C 等)。每种语言有自己的概念模型,因而模型的设计和分析技术是十分不同的。程序设计是从一种概念空间(设计现实世界特定表示)到另一种概念空间(编译或解释理解)的事务处理一大部分工作。

(3) 第三个活动是用另外的语言和概念模型定义和存取数据库。对关系型数据库来说,COBOL,FORTRAN 和 C 与处理记录的 SQL 语言有着严重的不匹配。这种典型软件开发,效率不高,应用受到了极大限制。

另一种开发方法称之为无保留库的软件开发过程(No_Vault Software development)。这种开发过程使用面向对象分析和设计方法以及相应的工具,再加上面向对象程序设计语言($C^{++}$)和面向对象数据库。这样具有单一的、又统一的概念模型——对象——在所有开发和维护阶段都使用。其结果是:高生产率(两堵高墙拆除,无需中间保留库,过去那些方法是人为加入的,不符合人们的思维活动);高质量(避免了有保留库时发生的各种错误和不匹配问题);高灵活性(对象组装程序,可重用,移植容易)。

### 1.6.5 对"算法+数据结构=程序设计"的挑战

我们知道,非面向对象的过程语言,如 FORTRAN、C 和 PASCAL,其数据结构是问

题解的中心。一个软件系统的结构是围绕一个或几个关键数据结构为核心而组成的。在这种情况下，计算软件开发一直被两大难题所困扰：一是如何超越程序的复杂性障碍；二是计算机系统中如何自然地表示客观世界即对象模型。诚然，Niklans wirth 提出的“算法＋数据结构＝程序设计”，在软件开发进程中产生了深远的影响。但软件系统的规模越来越大，复杂性不断增长，以致不得不对“关键数据结构”重新评价。数据结构的主要欠缺是应用范围和可视性。在这种系统中，许多重要的过程和函数（子程序）的实现严格依赖于关键数据结构，如果这些关键数据结构的一个或几个数据有所改变，则涉及到整个软件系统，许多过程和函数必须重写，甚至因为几个关键数据结构改变，导致软件系统整个结构彻底崩溃。

近三十年来，计算机科学家为提高软件生产率所作的种种探索与努力，或多或少与面向对象的程序设计这一思想有关联。作为克服复杂性的手段，在面向对象程序设计中，把密切相关的数据与过程定义为一个整体（即对象），而且一旦作为一个整体（包）定义了之后，就可以使用它而无需了解其内部的实现细节。面向对象软件系统的构造不依赖于对象的内部结构，而仅依赖于定义能在对象内部数据上实行操作的方法。

封装和数据隐藏是面向对象问题解和面向对象程序设计的基本要素。它可使一个软件工程师避免许多维护问题。一旦数据（类型）结构修改了，仅对实现模型的代码局部区域作适当变化，软件系统其余部分原封不动，因为修改部分相关联的数据仅仅是在定义模块中的过程和函数的接口部分。

面向对象技术提供给人们的封装和数据隐藏，表示了真正的面向对象语言的事务——在此事务中，是对象而不是函数或过程代表了问题解的中心环节。只有当我们重点不是送数据到函数，而送消息到对象的时候，我们才有了真正的面向对象语言。