C++语言基础与编程技术

李京山　宋建云　郭爱民　王　理等编著

宇航出版社

C++ 语言基础与编程技术

李京山　宋建云　郭爱民
王　理　吴建瑜　赵士杰　著
李　杨　　张　宏

学报出版社

(京)新登字 181 号

内 容 提 要

本书共分为两大部分。第一部分有上、中、下三篇，全面而系统地讲解了 C++ 语言基础知识和程序设计方面的知识。第二部分提供了便于读者快速查阅的 C++ 语言快速参考。全书内容全面详实，结构完整，语言清晰明了，是一本既通俗易懂，又有一定深度的 C++ 语言程序设计读本。

本书适用于计算机编程人员、技术人员及广大的计算机用户，同时也适用于各大专院校师生及计算机爱好者。

C++语言基础与编程技术

李京山　宋建云　郭爱民

王　理　吴建瑜　赵士杰　　编著

李　杨　　张　宏

责任编辑：赖巧玲

*

宇航出版社出版发行

北京和平里滨河路 1 号　邮编：100013

各地新华书店经销

河北地质六队美术胶印厂印刷

开本：787×1092 1/16　印张：26.25　字数：649 千字

1994 年 5 月第 1 版　1994 年 5 月第 1 次印刷　印数：1—6000 册

ISBN 7-80034-623-4/TP・039　定价：22.00 元

前　言

C++ 语言目前在我国计算机界已广泛使用，并得到众多专家学者的一致好评。C++ 之所以能如此成功地在我国推广，是因为这样一个简单事实：简单性是 C++ 的重要设计原则，其丰富的工具、库和软件开发环境使程序员能够更方便且更愉快地编写出令人满意的大型程序。可以说，在程序设计方面，C++ 语言已向前迈出了重要一步，在我国定会越来越普及。在此，我们也要重申一点：C 语言是 C++ 获此成功的基础，C 语言本身在近几年也在发展，从某种程度上说也受到 C++ 发展的影响。

本书由浅入深，注重程序结构，而不是编写算法，通过实例详细描述了 C++ 语言基础与程序设计技术，并在此基础上对 C++ 语言进行了全面地归纳和总结。因此，本书既有实用性，又具备一定的系统性，无论是对初级程序员还是对有一定经验的程序员都会有所裨益。

全书共分为两大部分：第一部分为 C++ 程序设计，第二部分为 C++ 语法和语义。

第一部分共 12 章，主要内容包括：(1)上篇 1－4 章，概述 C++ 语言特性与技术。内容包括基本类型、表达式、语句、控制结构。(2)中篇 5－9 章，内容主要包括类概念、派生类、多态性、继承、对象、模板及其应用，输入输出与数据流等。(3)下篇 10－12 章，主要讨论异常处理技术以及在设计和实现大型软件系统时使用 C++ 所涉及的有关事宜，如设计管理、C++ 与设计等较深层的问题。

第二部分共 8 章，给出 C++ 程序设计语言的完整语法及语义说明。该部分是一个相对独立的部分，相当于一个参考手册，适用于读者在编写程序时快速查阅。

在本书的编写过程中，得到了李晓、王佳明、张杰、韩东升、赵荣、杨敏芝、刘冰等同志的大力支持和帮助，在此对他们表示衷心地感谢。

编　者

1993.5

目　录

第一部分　C++ 程序设计

上　篇

此为试读,需要完整PDF请访问: www.ertongbook.com

中 篇

下 篇

第二部分 快速参考
—— C++语法和语义

第一部分

C++ 程序设计

上 篇

第一章 C++概述

这一章大致描述一下C++ 程序设计语言的主要概念和特性。其意图就是尽快使读者对C++有个整体印象。后面章节将详细讲解C++的语言特性和技术。本章的讨论注重于支持数据抽象和面向对象编程的语言特性，但也简要地介绍一下适用于过程编程的关键特性。

1.1 C++简介

C++ 是C的一个扩展版本。C++ 首先是由 Bjarne Stroustrup 于1980年在贝尔实验室发明的。起初，称这一新的语言为“带类的C语言”。然后，在1983年，将其名字改为C++ 。

尽管C++ 的前身，即C语言，目前在世界上是受欢迎且被广泛使用的专业编程语言之一，但C++ 的发明是编程中最主要的一个因素。在C中，一旦程序从25000行扩展到100000行，它就会变得复杂，且很难合为一个整体。C++ 的目的就是要打破这一障碍。C++ 的本质就是让程序员充分理解并管理更大型且更复杂的程序。

当发明C++ 时，Stroustrup 深知，最为重要的是保留C的原始精华，其中包括有效性、灵活性和基本原理。而与此同时添加了对面向对象编程的支持。令人高兴的是，它的目标都达到了。C++ 仍给程序员提供了C语言的灵活和控制，以及对象的强功能。用 Stroustrup 的话说，C++ 中的面向对象特性“使程序构造得很清晰，可扩充，易维护，且不失有效性。”

尽管C++ 在开始时是为管理非常大型的程序而设计的，但无法限制这一使用。实际上，C++ 的面向对象属性可有效地适用于任意编程任务。人们常常会看到C++用于像编辑程序、数据库、个人文件系统和通讯程序之类的项目。此外，由于C++ 共享C的有效性，所以更高性能的软件常用C++ 编写。

1.2 面向对象编程

面向对象编程(Object－Oriented Programming，缩写为OOP)是完成编程作业的一种新方法。自从发明计算机以来，编程方法戏剧性地发生了变化，主要是为了适应程序越来越复杂的状况。从二进制机器指令编写程序到汇编语言的发明，以至高级语言应运而生，给程序员提供了更多的工具，以处理更为复杂的程序。当然，第一个广为应用的高级语言是 Fortran。尽管 Fortran 语言迈出了令人记忆犹新的第一步，但用其还是难以编写出清晰

此为试读，需要完整PDF请访问：www.ertongbook.com

而易理解的程序。

结构化编程于60年代诞生。C和Pascal这样的语言使用这一方法。结构化语言使程序员有可能非常方便地编写出有一定难度的程序。然而，一旦项目达到某一程度，使用结构化编程的方法就变得无法控制。其复杂度达到了程序员无法管理的地步。

在编程研制的过程中，所建立的方法允许程序员处理日益复杂的问题。其中，每走一步，新方法都沿用了以前方法的最佳成分。如今，许多设计方法仍停留在不再适用的结构化方法上。为解决这一问题，人们发明了面向对象编程方法。

面向对象编程吸取了结构化编程的最佳思想，并把它们与几个强有力的新概念结合在一起，以一个新方法完成编程任务。一般来说，当用面向对象方法编程时，可把一个问题分为几个彼此相关的部分，并注意到与每个部分相关的代码和数据。此外，把这些部分组织成一个层次结构。最后，把这些部分变为叫作对象的自包含单元。

所有面向对象编程语言有三个公共部分：对象、多态性和继承。

1. 对象

面向对象语言的最重要特性是对象。简而言之，对象(object)是一个逻辑实体，它含有数据及处理该数据的代码。在一个对象中，有些代码和(或)数据可能是私用于该对象的，且不得由该对象外的任意成分存取。这样的话，对象提供一个有意义的保护级，以防止程序中不相关的部分偶尔修改或不正当使用该对象的私用部分。代码和数据的这一链接常称为封装(encapsulation)。

事实上，对象是用户定义类型的变量。初看它有些怪(链接代码和数据的对象看作变量)，然而在面向对象编程中，这是极为精确的表现形式。当定义对象时，便隐式地建立了一个的数据类型。

2. 多态性

面向对象编程语言支持多态性(polymorphism)。实际上，一个名字可用于几个相关但稍有不同的目的。多态性允许一个界面与一般类一起用。界面的一般概念是：将数据向堆栈推入并从堆栈中推出数据。函数为每个数据类型定义特定方法。

第一个面向对象的编程语言是解释编程程序，因此，多态性理所当然地在运行时得到支持。不过，C++ 是一个编译语言。它支持运行时编译时的多态性。

3. 继承

继承(inheritance)是一个进程，通过这一进程，对象可获取到另一对象的特性。这是重要的，因为它支持分类的概念。如果考虑这一点的话，绝大多数知识都可运用层次分类这一概念。例如，红香蕉苹果属于苹果类，苹果又属于水果类，而水果又属于更大的食品类。如果不使用分类，则不得不显式地定义每个对象的所有特性。然而，通过使用分类概念，对象便仅需定义那些使其在类中的唯一属性。这是一种继承机制，使对象有可能成为更一般情况的一个特定事例。

继承机制在面向对象编程中极为重要。请见如下举例：

```
class shape {
        point center;
        color col;
        // ...
public:
```

```
        point where() { return center; }
        void move(point to) { center = to; draw(); }
        virtual void draw();
        virtual void rotate(int);
        // ...
};
```

要定义一种特殊形状，必须指出，它是一种形状，并指定其特殊特性(包括虚函数)：

```
class circle : public shape {
        int radius;
public:
        void draw() { /* ... */ };
        void rotate(int)
};
```

在C++ 中，circle 类称为从 shape 类派生，而类 shape 称为 circle 类的基。编程范例是：

决定需要哪些种类；

为每个类提供操作全集。

检验一下过程编程、数据隐藏、数据抽象和面向对象编程所需的最小支持，我们将详细描述可使数据抽象和面向对象编程更有效的特性。

1.3 C++程序设计风格与编程方法

由于C++ 是C 的超级，所以我们可编写看似C 程序的C++ 程序。不过，这样做会妨碍充分地利用C++ 特性，这就好比看不带色的彩色电视机。相反，绝大多数C++ 程序员使用一种风格和某些唯一于C++ 的特性。C 程序和C++ 程序之间的绝大多数区别都与利用C++ 的面向对象功能有关。但是，使用唯一于C++ 编程风格的另一优点在于它有助于我们开始用C++ 而不是用C 思考。即，当编写C++ 代码时，通过一种不同的风格，告诉自己停止用C 思考，应开始用C++ 进行思考。

学会如何编写地道的C++ 程序是很重要的，所以这一节介绍其中几个特性。请看如下C++ 程序：

```
#include "iostream.h"
main()
{
 int i;
 cout << "This is output.\n";  //这是单行注释
 /*  也可以使用C 式注释 */
 //input a number using >>
 cout << "enter a number: ";
 cin << i << " squared is " << i*i << " \n ";
 return 0;
}
```

该程序与一般C程序有很大不同。开头包含头文件iostream.h。该文件是由C++定义的，并用于支持C++ I/O操作。

面向对象的程序设计是一种程序设计技术。如果"面向对象的程序设计语言"这一术语表示某种内容，那么它必须表示提供支持面向对象编程形式机制的程序设计语言。

在此存在一个重要区别，如果一种语言提供使其宜于(方便、安全和有效)使用程序设计风格的设施，那么就称该语言支持这一程序设计风格。

1.3.1 过程编程

原始编程范例是：

决定需要哪种过程；

使用可寻找到的最佳算法。

侧重面在如何处理、执行计算所需的算法。语言通过为传送函数变元并从函数返回值提供设施支持这一范例。Fortran是原始过程语言，Algol60，Algol68，Pascal和C是随后发明的。

平方根函数便是其中一例，根据变元，产生结果。为此，执行易于理解的算术计算：

```
double sqrt(double arg)
{
    // 计算平方根的代码
}
void some_function()
{
    double root2 = sqrt(2);
    // ...
}
```

注释以两个反斜线开头。

1.3.2 模块编程

多少年来，程序设计注重于从过程设计移至数据组织。其中，它反映了程序大小在不断增长。它们所处理并带有数据的相关过程集合常称为模块。编程范例为：

决定需要哪种模块；

划分程序，这样数据隐藏在模块中。

这一模式也称为数据隐藏原则。设计良好过程的技术此时适用于模块中的每个过程。定义堆栈模块是模块的最常见举例。要解决的主要问题是：

(1) 为堆栈(例如，函数push( )和pop( ))提供用户界面；

(2) 确保堆栈表示(如元素数组)仅可通过该用户界面存取；

(3) 确保在第一次使用堆栈时对其初始化。

C仅允许这种使用方法。下面是C或堆栈模块的外部界面：

```
void push(char);
char pop();
const int stack_size = 100;
```

假设在 stack.h 文件中存在该界面，实现可这样定义：

```
#include "stack.h"
static char v[stack_size];
static char * p = v;
void push(char c)
{
}
char pop()
{
}
```

我们可像下面这样使用堆栈：

```
#include "stack.h"
void some_function()
{           push('c');
      char c = pop();
      if (c ! = 'c') error("impossible");
}
```

由于数据仅是可隐藏的内容，所以数据隐藏表示扩展为信息隐藏表示；即：变量名、常量、函数和类型也可局部于模块。尽管设计 C++ 不是专用支持模块编程，但其类概念对模块概念给予支持。

1.3.3 数据抽象

用模块编程导致在类型管理模块控制下集中某一类型的所有数据。如果需要两个堆栈，则定义堆栈管理模块如下：

```
class stack_id { /* ... */ };
stak_id create_stack(int size);
void push(stack_id, char);
char pop(stack_id);
destroy_stack(stack_id);
```

这是原来基础上的一大进展，但以这种方式实现的类型显然有别于内部类型。每个类型管理模块必须定义一种独立机制，以创建其类型变量。这样的类型变量没有编译程序所知的名字，也没有编程环境，这样的变量更不遵循一般的作用域规则或变元传送规则。

通过模块机制创建的类型在许多重要方面有别于内部类型。例如：

```
void f()
{
      stack_id s1;
      stack_id s2;
      s1 = create_stack(200);
      push(s1, 'a');
      char c1 = pop(s1); // pretty ugly
```

```
        destroy_stack(s2);
        s1 = s2;
    }
```

换句话说，支持数据隐藏的模块概念允许这种编程风格，但不支持它。像 Ada，Clu 和 C++这样的语言允许用户定义几乎与内部类型同样操作的类型。这样的类型常称为抽象数据类型。也可以称为用户定义类型。编程范例如下：

决定需要哪些种类型；

为每种类型提供操作全集。

算术类型是常见的用户定义类型举例：

```
class complex {
        double re, im;
public:
        complex(double r, double i) { re=r; im=i }
        complex(double r)    { re=r; im=0; }
        friend complex operator+(complex, complex);
        friend complex operator-(complex, complex);        // 二目
        friend complex operator-(complex);                 // 一目
        friend complex operator*(complex, complex);
        friend complex operator/(complex, complex);
        //...
};
```

说明类(即用户定义类型)complex 指定复数表示和对复数的操作集合。表示是 private；即，re 和 im 仅对在说明 complex 类时所指定的函数可存取。

这样的函数可定义如下：

```
complex operator+(complex a1, complex a2)
{
        return complex(a1.re+a2.re,a1.im+a2.im);
}
```

并这样使用：

```
void f()
{
        complex a = 2.3;
        complex b = 1/a;
        complex c = a+b*complex(1,2.3);
        //...
        c = -(a/b)+2;
}
```

绝大多数(但不是全部)模块最好表示为用户定义类型。

1.3.4 对数据抽象的支持

对用数据抽象编程的基本支持由如下设施构成：定义类型的操作集合（函数和算符）和限制对类型对象的存取。

1. 隐藏类型

当隐藏类型表示时，必须为用户提供某些机制，以便初始化该类型变量。简单的解决方法是：需要用户调用某一函数，在使用变量之前将其初始化。例如：

```
class vector {
        // ...
public:
        void init(int size);
        // ...
};
void f()
{
        vector v;              // 此处不使用 v
        v.init(10);
        // 此处使用 v
}
```

这是一种错误倾向且不优雅。最佳的解决方法是允许类型设计人员提供不同的函数，以便进行初始化。根据这样的函数，变量的分配与初始化变为单一操作，而不是两种独立操作。这样的初始化函数称为构造函数。通过与类同名可标识构造函数。在构造类对象为非平凡的地方，人们常常需要在其最后一次使用后删除对象。在C++ 中，这样的删除函数称为析构函数。析构函数名与类名相同，只是前面加一个～ 符号。例如：

```
class vector {
        int sz;
        int * v;
public:
        vector(int);                    // 构造函数
        ~vector();                      // 析构函数
        int& operator{}(int index);
};
```

我们可以定义 vector 构造函数，以检查是否有错，并分配空间：

```
vector::vector(int s)
{
        if (s<=0) error("bad vector size");
        sz = s;
        v = new int{s};
}
```

vector 析构函数释放所用的空间：

此为试读，需要完整PDF请访问：www.ertongbook.com