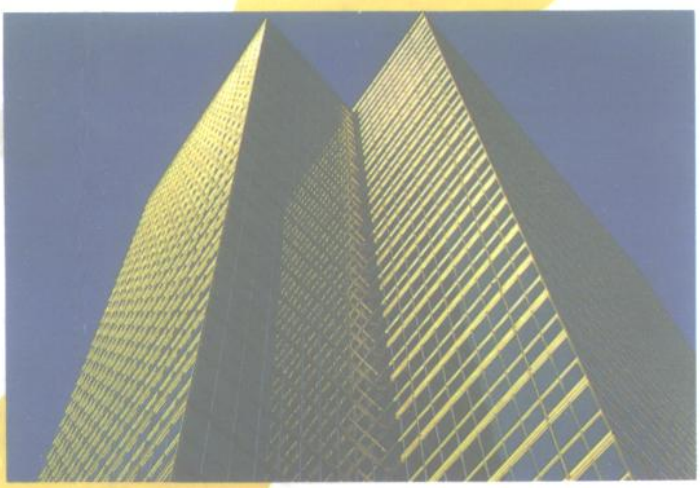
C++编程技巧
——改进程序设计50法

Effective C++
50 Specific Ways to Improve Your Programs and Designs

Scott Meyers



Scott Meyers 著　陈迅雷　黄　榕　译
戴　锣　审校

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES
上海科学普及出版社

C++编程技巧

——改进程序设计50法

Scott Meyers 著
陈迅雷　黄　榕　译
戴　锷　审校

上海科学普及出版社

(沪)新登字第 305 号

责任编辑　毕淑敏　胡名正
封面设计　毛增南

C++编程技巧
——改进程序设计 50 法
Scott Meyers 著
陈迅雷　黄　榕　译
戴　愕　审校

上海科学普及出版社出版
（上海曹杨路 500 号　邮政编码 200063）

新华书店上海发行所发行　　上海市委党校印刷厂印刷
开本 787×1092　1/16　印张 10.5　字数 242000
1994 年 7 月第 1 版　1994 年 7 月第 1 次印刷

ISBN 7-5427-0802-3/TP・191　　定价：12.00 元

内 容 提 要

本书集教学经验写成，它指明C和C++的异同，以丰富的程序实例说明：什么用法是不允许的，哪些用法比别的用法更好，如何充分利用C++的优势，怎样避免编程失误等，还着重介绍高质量程序的编程方法和面向对象的程序设计技术。全书提出50个专题，分为7章，内容包括：从C到C++的修改过渡，全局变量、指针成员等的定义和释放等内存管理问题，构造函数、析构函数和赋值运算，类和函数的设计、说明及实现，继承，面向对象的程序设计等。本书指导读者运用最新编程技术、增强编程优化意识。

读者对象：微机用户，程序员，大专院校有关专业师生。

Effective C++.

50 Specific Ways to Improve Your Programs and Designs

Scott Meyers

ADDISON-WESLEY PUBLISHING COMPANY, INC.

For Nancy,
without whom nothing
would be much worth doing.

Wisdom and beauty form a very rare combination
-Petronius Arbiter
Satyricon, XCIV

序

在给 Institute for Advanced Professinal Studies 讲授C++语言以后，我用自己的经验写成了这本书。我发现，经过一星期的强化指导，大多数的学生已能通晓该种语言的基本构造，但是还不大会有效地将这些构造结合起来运用。所以我试图为使用C++语言开发有实效的软件编写一些简短的、专门的、又易记忆的指南：即集中地指出老资格的C++程序员总需要做的，或者总需要避免做的是哪些事。

作为一个计算机科学工作者，我感兴趣的是可以由机器来检测的那些规则。为此，我编写了一般程序来检测一些"几乎总是含有错误"的C++构造的软件。当前正在开发中的这个检测程序被称为 lint++。但是，一些素质良好的C++程序员所使用的大多数指南过于复杂，因此难以规格化，或者是重要的例外太多，常被 lint++之类的程序盲目地限制。

于是就使我萌发了一个想法，写一本有关虽不如计算机程序那么精确，但又比普通C++教材更专门一些的书，阐述如何提高C++编程和设计，提出50题专门的方法。

本书提出了你应该做些什么，为什么要这样做，还有不应该做些什么，为什么不应该这样做的道理。当然，从根本上说，明白道理比"怎样做"更重要。然而，从实际运用的观点来看，手头有一系列指南，比记住一、两本教材的内容更方便、更实用。

与许多C++教材不同的是，对这本书我并不按照某个语言特点来进行编排。也就是，不是在一个地方讲述构造函数，而在另一个地方讲述虚拟函数，又在其它地方再讲述继承，等等。相反，每一种解释都围绕一种特定的指南，而且对某个语言特点的各个方面的描述，则贯串于整本书中。

这种编排方法的优点在于能较好地反映C++是一种可选的软件系统，具有复杂性，而且使你懂得仅仅理解系统的个别语言特点是远远不够的。例如，有经验的C++开发人员会知道，光了解内嵌函数和虚拟析构函数并不意味着就一定能理解内嵌虚拟析构函数。理解C++中各种语言特性间的相互作用是高效使用语言的一个最重要因素。我就是根据这个原则来编排这本书的。

但这种编排方法的缺点则在于，它要求你必须全面理解C++的构造，因此就必须查找书中的好多处地方。为了便于查找，书中都加以提示，并在书后提供了索引表。(译注：索引表见本书原版。)

本书中的一系列指南远不能说是最详尽的，但也给出了一些好的规则。当然，要使它们在任何时候都适合于任何的应用，也是不容易的。你可能已经知道了其它的一些指南，也有了另外一些有效地编写C++程序的方法。如果是这样，我将对此感到非常高兴。

另一方面，作为普遍推行的建议，书中的某些方法还不能算是尽善尽美的，或许还有更好的方法来执行书中的测试工作，或者是在技术方面的讨论仍有不清楚、不完整、以及易误解的地方，则请读者向我指出。

Donald Knuth 曾经长期给予能指出他书中错误的人若干奖励，这种做法是值得赞赏的。尤其当今有不少错误百出的C++教材冲击市场的时候，Donald Knuth 的做法更值得

提倡。所以，对于书中的技术上、语法上、印刷上以及其它方面的错误，请来信告诉我。在本书以后的修订再版时，我将致谢第一个向我指出错误的人。

把你的建议、评价以及批评寄至下面的地址：

Scott Meyers
c/o Editor—in—Chief, Corporate and professional
Publishing
Addison—Wesley Publishing Company
1Jacob Way
Reading, MA 01867
U. S. A.

Scott Douglas Meyers　　Providence. Rhode Island
1991 年 11 月

鸣　谢

差不多整整二十年以前，Kathy Reed 教我懂得什么是计算机，还教会我怎样编程。二十年光阴荏苒，尽管我已经离开了 110 波特的电传打字机，拥有了每秒 12 兆指令的工作站，使用的语言从 BASIC 升级到了 C++，但往事历历在目，她的谆谆教诲令我难忘，永远深深铭刻在我的脑海里。

1989 年，Donald French 邀我为 Institute for Advanced Professional Studies 编写 C++培训教材。从那时起，他给了我使用、修改、补充这些教材的机缘。如果没有他，我不会对 C++知道得这么多，以致能公开进行探讨，恐怕也不会编写这本书了。当我决定要写这本书时，Don 给了我许多帮助。他还介绍我认识了 Addison—Wesley 出版公司的 John Wait。我非常感激他们。

在 1991 年 6 月 3 日的那一周里，Stratus Computer 公司的那一班学生建议我写一本书总结我在教学时迸发出来的智慧结晶，就是他们促使我最终决定写这本书的。就在他们的课程结束之后，我便开始着手编写。为此，我也要感谢他们。

笼统地说，这本书中的这些专题和实例，并没有特别的来源，至少我已经记不得了。它们更多的是 来之于我使用和讲授 C++的经验，我的同事们，以及 Usenet newsgroups 的同行提供的关于 comp. lang. C++和 comp. std. C++的建议。现在作为 C++教学的许多标准例子（例如串和复数）来源于 Bjarne Stroustrup 的《The C++ Programming Language》的第一版（Addison—Wesley. 1986）；有些内容（例如本书专题第 17）也能在那本书中找到。专题第 9 受到《The Annotated C++ Reference Manual》（参见专题第 50）的启发，John Shew—chuk 也对专题第 10 和第 13 提出了建议。在专题第 17 之末那个使用别名的例子是 Doug Lea 所提供的。专题第 43 中蚱蜢和蟋蟀（grasshoppers and crikets）的例子的构思，我则是直接取自于 Mark Linton。当然，“Hello World”来之于 Brian W. Kernighan 和 Dennis M. Ritchie 的《The C Programming Language》（Prentice—Hall，1978）。Alexander Wolf 和 stan Lippman 在最后一分钟关于模板方面提供了极有价值的帮助。专题第 10 中的 operator new 操作的实现的原型来自 Stroustrup 的《The C++ Programming Language》第二版（Addison—Wesley，1991）。

Tom Cargill，Glenn Carroll，Tony Davis，Brian Kernighan，Jak Kirman，Doug Lea，Moises Lejter，Eugene Santos，Jr.，John Shewchuk，John Stasko，Bjarne Stroustrup，Barbara Tilly 和 Nancy L. Urbano 对本书手稿作了部分或者全部的校阅。他们在技术上的评述，以及语法方面的解释和建议，为此书增色不少。

我还要感谢下列人士，他们指出了此书以往各次印刷中的错误，他们是 Nancy L. Urbano，Chris Treichel，David Corbin，Paul Gibson，Steve Vinoski 和 Tom Cargill。不用说，如还有错误也在所难免，当然责任应由我来负。

Addison—Wesley 公司的编辑 John Wait 先生自始至终参与了这项工作。他在提高趣味性和丰富本书的内容方面给了我意想不到的帮助，甚至包括许多我前所未知的内容。

最后，我要感谢我的妻子 Nancy L. Urbano，以及我的家人对我的热情鼓励和支持。编写这本C++书，耗去了我许多业余时间，是他们使我相信，为此书付出的种种努力是值得的。因此，本书的成功离不开他们。

目　录

引　言

学习一种程序设计语言的基本原理是一回事，而学习怎样运用该种语言设计和编写高效的程序则是另一回事。学习和运用C++语言正是这样，C++是一种有着非同寻常的功能和表达能力的一种语言。建立了常用语言C的这样一种功能齐全的顶峰C++，也就提供了一个范畴广阔的面向对象的特性。对模板(参数化类型)的支持近来功效日趋强化，并且对处理例外性的构造，也具有了专业语言的特征(虽然它们中大多数还没有用于商业化的编译软件中)。

只要使用恰当，C++能使工作变得趣味无穷。无论是面向对象的和常规的软件设计，C++都能直接进行表示并高效地实现。你可以用它定义新的数据类型，它们能与系统的内含定义的数据类型无甚逊色，然而却灵活得多。通过理智慎重的推敲，精心构思的技巧设计一组class(类)，它们就能自动地做好内存管理、别名使用、初始化和清除、类型转换，并能解决其他一些足以难倒许多程序员的一系列难题。这一切可使得应用程序设计变得很容易，更直观，非常高效甚至能做到几乎不出错。简而言之，只要你知道了解怎样去做，编写出高效的C++程序并不是一件太难的事情。

但是如果未经充分培训就去使用，C++也会生成这样的一些代码，它们难以理解，无法维护，没有可扩展性，效能低下，或者甚至是错误百出。

编写这本书的用意在于帮助你掌握C++这两方面的决窍：很容易找到你的出错所在，并且学会怎样避免这些错误的发生。本书假定你早就懂得作为一种编程语言的C++，而且还积累了一些使用经验，这里提供的只是怎样高效地使用语言的一种指南，让你设计的软件具有高效能、可维护、可扩展的特点，并且能按你预期的要求运行。

我提出的建议大致可分成两种：一是通用的设计策略，另一个是专门的语言功能的核心。

本书关于设计的讨论着重于在不同方法之间怎样进行选择来完成C++中的某些事情。例如，在继承与模板间你怎样选择？在模板和类属指针，共用和私有的继承，私有继承和分层，函数超载和参数缺省间你如何选择？还有在虚拟和非虚拟函数，值传递和引用传递间的你又是如何选择？一开始如能对这些作出正确决择是很重要的，因为一个错误的选择起初并非总是很明显的，往往要到开发进程的最后阶段才会显露出来，而此时修改常常是艰难、费时、纷乱并且是昂贵的，为时已晚。

即使你能够确切地知道你想要做什么，但真要把它做好仍然是棘手的。对一个赋值运算符来说，取哪一种返回类型才是恰当的？当operator new找不到足够的内存时，它应该怎么运转？何时析构函数该用虚拟型的？怎样操作一个类常量？怎样编写一个成员初始表？详尽地做好这些是至关重要的。如果处理不当的话，常常会导致预期不到的，极可能是令人困惑的程序状态。更重要的是，含错误的运行常常并不是马上显露端倪，由于隐含着各种难以检测的错误，会增加质量控制中的不必要的程序代码。超时的等待也会使时间记录溢出。

本书的内容不必一页一页地弄清楚，甚至不必从头至尾顺序地阅读。本书的所载50个

专题，每题或多或少带有各自的主题方向。然而，在一个专题里往往会涉及到另一些专题，所以，阅读这本书的一个方法是可以从你感兴趣的那条专题开始，循着它所提供的指引进行阅读。

这许多专题已按各自所属的相通的主题分类成章，因此如果你对其中某方面的主题的讨论有兴趣，比如关于内存管理或者面向设计的内容，就可以从有关的章节看起，可以从头到尾阅读一遍，亦可以跳跃式地选读。但是，你将会发现，这本书中所有的材料对于用C++作高效的编程来说都是相当基本的。所以，几乎每样东西最终不是这样就是那样地同别的事情联系起来的。

本书不是一本C++的语言参考书，它不供你从头开始学习C++的语言。譬如说，我很愿意告诉你有关你自己的operator new(见专题7、8和9)中所有的gatchas该如何写，但是，我又猜想你在别的地方也能看到那个函数返回的必须是一个void*类型，它的第一个参数必须是size—t类型。述及这方面资料较好的C++入门书为数不少。

本书的宗旨趋向于介绍C++程序设计中某些常常会被浮浅地一带而过或者易遭完全忽视的那些方面内容。别的许多书本为你描述了该语言的各种部分，而这本书将告诉你怎样把这些不同的部分综合起来，让你完成一个高效的程序。别的书可以告诉你怎样编译你的程序，而这本书却能告诉你怎样避免编译程序所未能报告你的那些问题。

像大多数的语言一样，C++也有丰富的"民间"知识，它作为语言口头表达传统的一种，由一代又一代程序员流传至今。在这本书里，我尝试着用更易接受的形式把那些聚沙成塔的智慧结晶记录下来。

同时，本书的内容只限于"法定化"的，允许移植的C++规定。只有符合ANSI创建的标准(见The Annotated C++Reference Manual一书即"ARM"，请参阅专题50)特性的版本的语言才能结合本书使用。有时，这个规定看来是很严格的。例如专题32在编译时，要检查那些被用到的操作特殊类常量如何实现的问题。一个通用的编译程序常会允许你像这样地编写程序：

```
//操作类常量的一个不可移植方法
    class X{
    private:
      const BUFFER—SIZE=1024;   //安排的类常量
      int buffer[BUFFER—SIZE];
      …
    };
```

虽然这是既简单又直截了当的，但是它却不被ANSI标准所允许，所以在专题32中提出了另一种不同的机制来实现同样的事情，这种机制能获上述标准的支持。在这本书中，融汇贯通是个关键的问题。所以，如果你试图按图索骥，那就找错了地方。

遗憾的是，一个明白不过的事实是，即ARM指定的C++标准语言规格同你关系密切的编译程序供应商所能提供的C++版本有时是有很大差异的。尤其是类型嵌套模板和例外处理(见专题49)在ARM里都有。但是就在我在1991年秋季写这本书的时候，没有一个通用的编译程序能支持所有的这三种功能，甚至能够支持其中两种的也极少。

所以，当我指出这些功能是非常有用处的时候，我同样也讲明如果缺少这些功能的话，

你该怎样来面对如何生产高效的软件这样一个现实问题。说到底，完全不顾将来必然会诞生的事物将是十分愚蠢的。但是，你不能死死抱住一块王牌不放，“舍命陪君子”，等待最新颖、最强大、尽善尽美的C++编译程序魔幻般地跻身到你的计算机上。你只能利用你能够得到的工具来工作，这本书能帮助你的就在于此。

在这本书里，有一样东西你是找不到的，那就是C++福音，即一条真正通向完美无缺的C++软件的路径。在本书50个专题中，每一个都给你指出怎样作出更好的设计、怎样避免共同面临的问题，怎样获得更高的效率，但是这些专题中没有一种万试万灵的方法。软件的设计和实现是复杂的活计，它不可避免地要受到硬件、操作系统、应用的制约，所以在这里尽我所能可以做到的就是提供如何编制程序更好的指南。

如果任何时候都遵循所有的指南，你不可能陷入有关C++最常见的陷井。但是根据它们的本质特性，各指南也有例外，这就是为什么本书每一条专题里都需要有一条解释。解释是这本书最重要的部分，因为只有确实理解了，在专题里蕴含的基本原理，才能合理地确定它是否适用于你正在开发的程序，是否适用于使你受累的那些独特的限制。

那么，这本书最好的用途就在于洞悉C++是怎样的一种语言，以及为什么是这样，怎样使它的运行能为你所用。盲从本书上介绍的指南显然是不合适的，但另一方面，如果没有很好的理由，你也不要无端地对这些指南反其道而行之。

在书中罗列堆砌术语词汇毫无意义，那种文字游戏最好留给语言律师们去玩。可是，一个小规模的C++术语词汇量则是人人都应该掌握的。下面要叙述的六个词语，它们使用频繁，因此值得花笔墨说服我们接受它们表示的含义。

鉴于此书的目的，C++程序的declaration(说明)部分告诉编译程序对象或函数的名字，但不需提供细节。下面就是说明的例子：

```
extern int x;                          //对象说明
void makeUpperCase(char *s);  //函数说明
```

但是相反地，程序的definition(定义)部分则向编译程序提供了具体的细节。譬如说关于对象，其定义就是编译程序给对象分配的内存空间。又譬如说，函数的定义则提供了其代码体。

```
int x;                                  //对象定义
void makeUpperCase(char *s)       //函数定义
{
unsigned length=strlen(s);
for (unsigned i=0;i<length;i++)
  s[i]=toupper(s[i]);
}
```

我们再去看看constructor(构造函数)。一个default constructor(缺省的构造函数)就是可以不带参数而调用的函数。这样的构造函数不是根本没有参数，就是其每个参数都具有缺省值。当你要说明对象数组的时候，就需要使用缺省构造函数了：

```
class A{
public:
  A();                          //缺省构造函数
```

```
};
A arrayA[10];              //被调用的10个构造函数
class B{
public:
  B(int x=0);              //缺省构造函数
};
B  arrayB[10];             //被调用的10个构造函数
                           //每个构造函数带有参数
class C{
public:
  C(int x);                //不是缺省构造函数
};
C arrayC[10];              //出错!
```

你可能会发现,当类的缺省构造函数具有缺省参数值时,你的编译程序会不接受对象数组。例如,许多编译程序会在上面的array B说明处阻塞,即使ARM保佑array B也不行。这就是存在于ARM的C++标准描述和具体编译程序的语言实现之间的差异的一个例子。我们知道,每个编译程序都有这样的缺陷。编译程序供应商只有追赶(仍在演变着的)标准的潮流,具有灵活头脑,那么在不远的将来肯定会有所安慰,此时在ARM中描述的C++将和被C++编译程序接受的语言一样了。

偶尔,如果你想建立一个对象数组,但它没有相应的缺省构造函数,通常的办法是说明一个指针数组来代替,接着你可以通过调用new来分别初始化每个指针。

```
C *ptrArray[10];           //没有被调用的构造函数
ptrArray[0]=new C(22);     //分配并构造一个C对象
ptrArray[1]=new C(4);      //分配并构造一个C对象
...
```

让我们回过头来继续谈论下面的术语。一个copy constructor(复制构造函数)可以用来以另一个相同类型的对象来初始化某一个对象。

```
class String{
private:
  char *data;
public:
  string(const char *value=0);        //缺省构造函数
  string(const String& x);             //复制构造函数
};
string s1;                             //调用缺省构造函数
String s2(s1);                         //调用复制构造函数
string s3=s2;                          //调用复制构造函数
```

复制构造函数一个最重要的用途是定义用值作传递和返回的对象的意义。作为一个例子,考虑一下下面这个编写一个函数来连接两个String对象的方法(但效率并不好):

```
String operator+(string s1,string s2)
{
  String temp;
  temp.data=
    new char[strlen(s1.data)+strlen(s2.data)+1];
  strcpy(temp.data,s1.data);
  strcat(temp.data,s2.data);
  return temp;
}
String a("Hello");
String b("word");
String c=a+b;                    //c=String("Hello World")
```

这个函数operator+把两个String对象作为参数，且返回一个String对象作为结果。参数和结果两者都通过值传递，所以这里调用一个复制构造函数，用a来初始化s1，调用另一个函数用b来初始化s2，再调用第三个函数用temp来初始化c。实际上，如果编译程序决定生成中间性的临时对象，这是允许的，那么另外还有一些对复制构造函数的调用。这里很重要的一点是，值传递的意思就是"调用复制构造函数"。

顺便一提，你实际上并不需要像这样为所有的string对象编写operator+函数。返回一个新的string对象是对的(见专题23)，但你会想在两个参数中通过引用来作传递(见专题22)。

下面需要掌握的两个术语是initialization(初始化)和assignment(赋值)。一个对象的初始化就是指对象首次被给定一个值。对于带有构造函数的类对象或者结构对象，初始化总是要调用一个构造函数来实现。它同对象的赋值是完全不同的，对象的赋值则是指给已经初始化的对象一个新值。

```
String s1;                       //初始化
String s2("Hello");              //初始化
String s3 =s2;                   //初始化
s1=s3;                           //赋值
```

从纯粹的操作观点看，初给化和赋值间的不同在于前者由构造函数来执行，而后者则由operator=来执行。换而言之，这两个过程相对应于两种完全不同的函数调用。

要区分它们的理由是因为这两种函数必须关心不同的事物，构造函数常常必须检查它们赋值的有效性，而大多数赋值运算符理所当然地认为它们的赋值是合法的(因为赋值已经被构造了)。另一方面，同正在构造中的对象不同的是，一个赋值的目标对象可以已经被分配有资源，一般这些资源必须在新资源被分配前被释放。经常是这样，其中一个资源是内存，在赋值运算符给内存指定一个新值元前，原来旧值的内存分配必须解除。

这里请看一个String构造函数和赋值运算符是怎样实现的：

```
//一个可能的String构造函数
String::String(const char *value)
{
```