Mc Graw Hill Education

The Art of C++

C++ 编程艺术

(美) Herbert Schildt 著

曹蓉蓉 刘小荷 翻译
毕长剑 战晓苏 审校

Mc Graw Hill

清华大学出版社

C++编程艺术

(美) Herbert Schildt 著

曹蓉蓉 刘小荷 翻译

毕长剑 战晓苏 审校

清华大学出版社

北 京

Herbert Schildt
The Art of C++
EISBN: 0-07-225512-9

图书在版编目(CIP)数据

C++编程艺术/(美) 斯切尔特(Schildt, H.)著；曹蓉蓉 刘小荷翻译. —北京：清华大学出版社，2005.4

书名原文：The Art of C++

ISBN 7-302-10017-9

Ⅰ.C… Ⅱ. ①斯…②曹…③刘… Ⅲ. C 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2004)第 124674 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
http://www.tup.com.cn　　邮　　编：100084
社总机：010-62770175　　客户服务：010-62776969
组稿编辑：曹　康　　文稿编辑：于　平
封面设计：康　博　　版式设计：康　博
印 刷 者：北京市清华园胶印厂　　装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　印 张：21.25　字 数：544 千字
版　　次：2005 年 4 月第 1 版　2005 年 4 月第 1 次印刷
书　　号：ISBN 7-302-10017-9/TP・6875
印　　数：1~6000
定　　价：39.80 元

译 者 序

本书作者 Herbert Schildt，是公认的 C、C++、Java 和 C#等主流编程语言的程序设计大师和 Windows 程序设计专家，也是顶级编程图书作者；同时，他还是 ANSI/ISO 和 C++标准化组织的专家。他的编程书籍被翻译成多种语言版本广为流传，在全球世界范围内的销量已经超过三百万册。

本书的突出特点之一是编程技巧全面。本书以 C++国际标准语法为基础，从高级特性全面讲解 C++语言编程技术、技巧，充分展示了 C++语言的强大性、多样性、优美性、敏捷性和艺术性。作者结合多年软件开发和教学经验总结出非常有价值的完整示例，以行之有效的方法让读者快速精通 C++语言编程技巧。本书每章给出的示例代码都可以直接运行，无需修改，而且读者可以从 www.osborne.com 网站免费下载本书代码。相信读者通过研读本书可大大提高 C++编程能力。

本书的突出特点之二是内容丰富实用。在本书中，Herbert Schildt 给出了许多非常实用的高性能程序，每个程序分别侧重于 C++语言的不同方面。本书的实用示例程序主要包括垃圾回收器子系统、可以断点续传的 Internet 文件下载程序、线程控制面板、基于人工智能(AI)的搜索程序、通用 STL 容器和小型 C++解释程序等。作者对每个程序都给出了非常细致的分析和解释，这对于读者来说无疑是大有帮助的。本书代码示例易懂有趣、设计思想独特，从中读者可以学到很多在其他 C++书中无法学到的技巧，使读者能够掌握 C++高级编程的技巧，真正进入 C++高级编程领域，值得每位 C++程序员阅读和珍藏。

本书的突出特点之三是文笔透彻精确。Herbert Schildt 是全球著名的程序设计语言书籍作者，本书秉承他一贯的写作风格：简捷、清晰，非常适合读者的学习和阅读。通过本书的阅读，读者可在 C++程序设计大师的引领下探索编写高性能 C++程序的奥秘。本书让读者的 C++编程技术更上一层楼。

本书为 C++编程高级读物，面向有初级 C++语言基础和一定编程经历的程序设计者，适合作为高等院校计算机专业相关课程辅助教材，也可作为高年级本科生、研究生和广大编程爱好者深入学习 C++及其他面向对象语言的技术参考书。

本书由曹蓉蓉、刘小荷翻译；毕长剑、战晓苏审校。虽然与 Herbert Schildt 的高超造诣相比甚远，但是本书的译者和审校者还是凭借多年的 C++教学和编程经验，在翻译、校译此书的过程中，抱着对读者认真负责的态度，力争将原书的风格和思想原原本本地呈现给读者。还要特别指出的是，清华大学出版社对本书非常重视，从全书的翻译、编辑、排版到印刷质量上都下了很大的功夫。因此，可以相信这本中译本能够成为对读者大有裨益的好书。

此外，我们曾专门以电子邮件方式与原书作者进行了多次沟通，将原书的个别错误在翻译、校译和审校过程中加以改正。限于水平的原因，中译本中不妥或错误之处在所难免，敬请广大读者批评指正。我们的 e-mail 是 fwkbook@tup.tsinghua.edu.cn，读者有问题可随时联系。

前　言

从早期的 FORTRAN 语言开始，计算机语言就一直不断地发展演变。在此过程中，消除了不稳固的特性，取而代之的是一些功能强大的特性。多年过去后，这些进化的努力被精练为一种纯粹的形式，那就是程序设计语言应该具有的纯净本质。这么多年努力的结果就是 C++语言的产生，在程序设计历史上，任何其他语言都没有像 C++语言拥有如此重要的地位。

C++的成功取决于许多原因。语法简洁而优雅；对象模型简明流畅，容易理解；C++中提供了精心编写的库。然而，并非是这些特性为 C++赢得历史上的重要地位，而是 C++给予程序设计人员的强大的功能。从来没有其他的语言能够使得程序设计人员更加直接地控制计算机。通过使用 C++，程序设计人员就是机器的主人——这正是所有的程序设计人员所需要的。

没有边界，没有限制，没有约束。这就是 C++语言。

0.1　本书内容

本书不同于大多数其他的 C++书籍。其他的 C++书籍讲授语言的基础，而本书展示了如何应用 C++在更大的范围内完成有趣的、有用的、甚至是神秘的程序设计任务。在此过程中充分显示了 C++语言的强大功能和优雅性。

大致来说，本书包含两类应用程序，第一类称为“纯代码”，因为它们注重于扩展 C++程序设计环境本身。第 2 章的垃圾回收器，第 3 章的线程控制面板以及第 8 章的定制 STL 容器都是这种类型的示例。第二类应用程序显示了如何应用 C++来完成各种计算任务。例如，第 5 章开发了一个可以断点续传的网络下载工具，第6章给出了一个如何建立财务应用程序的范例，第 8 章中应用 C++实现了人工智能应用。

本书以一段独特有趣的代码结束：Mini C++解释程序，这个程序可以解释 C++的一个小子集。Mini C++解释程序揭示了 C++的关键字和语法是如何一起工作从而组成这门语言的语法的。更重要的是，这可以使您了解这门语言的内部机制，并且显示了隐藏在 C++设计背后的一些原因。使用 Mini C++解释程序不仅有趣，它还可以用作开发您自己的语言的起点，还可以用作其他语言的解释程序。

本书每一章都提供了可以直接使用的代码。例如，第 2 章的垃圾回收器可以适用于许多程序设计任务。然而，只有把这些应用程序作为自己的开发起点，才会获益匪浅。例如，可以进一步完善第 8 章的 Internet 文件下载工具，使其可以在某个指定的时间开始下载，或者可以监控一个下载站点，保持下载最新的文件。总之，可以将这些不同的程序和子系统作为您开发自己项目的跳板。

0.2 预备知识

本书假定读者具有坚实的C++语言基础知识。读者应该能够创建、编译并运行C++程序。应该能够使用指针、模板以及异常处理，理解复制构造函数并且熟悉标准库的常用部分。因此，本书假定读者具有可以从C++教程中获得的技巧。

如果读者需要复习或者加强基础知识，作者推荐下面几本书。

C++ From the Ground Up

C++: A Beginner's Guide

C++: The Complete Reference

这3本书都是McGraw-Hill/Osborne出版社发行的。其中*C++: A Beginner's Guide*一书的第1版和第2版已经由清华大学出版社出版发行，书名为《C++基础教程》和《C++基础教程(第2版)》。

0.3 源代码

本书所有的示例和项目的源代码都可以从网站www.osborne.com上免费下载。

目　录

第 1 章　C++的功能

C++的功能精深而广博，包括：可以直接对机器底层进行编程控制，生成高效的运行代码，具有直接与操作系统交互的能力。使用 C++，可以全面控制对象，包括对象的创建、销毁以及继承，可以访问指针，并且支持底层 I/O。可以通过定义类和重载运算符来加入新的功能。可以创建自己的库并手工优化代码。甚至可以在需要的时候“打破规则”。C++并不是为谨小慎微者准备的语言，它是为需要世界上最强大编程语言的程序员准备的。

当然，C++并不只是具有原始功能。它的功能是有条理的、集中的并且是直接的。它的精心设计、功能丰富的库以及微妙的语法形成了灵活的编程环境。尽管 C++的特长在于创建高性能的系统代码，但是它也可以用来完成任何其他编程任务。例如，它的字符串处理能力是其他语言无法比拟的，它的数学和数字运算能力使其特别适用于科学计算编程，C++可以生成高效的目标代码，可以很好地完成需要大量复杂运算的任务。

这本书的目标是展示 C++的功能、应用范围以及它的灵活性。我们通过用 C++实现各种不同的应用程序来做到这一点。一些应用程序说明了语言本身的能力。这些应用程序被称为“纯代码”示例，因为它们体现了 C++的语法规则以及使用 C++的方便性、灵活性和简洁性。第 2 章的垃圾回收器和第 9 章的 C++解释程序就是这样的示例。其他应用程序说明了 C++可以很容易地应用到普遍的程序设计任务中。例如，第 5 章的下载管理器体现了 C++创建高性能的网络代码的能力。第 6 章将 C++应用于各种财务计算。总之，这些应用程序显示了 C++是一种功能强大的语言。

然而，在开始接触应用程序之前，花费一点时间思考一下是什么使得 C++成为如此优秀的语言，这对您的学习是有好处的。为此，本章花一些时间来指出赋予了“C++功能”的几个特性。

1.1　简洁而丰富的语法

C++的一个主要特征是它语法的简洁。C++语言只定义了 63 个关键字。C++定义了丰富但简洁的语法来提供控制语句、运算符、数据类型以及任何现代语言所需要的面向对象的特征。因此，C++的语法是简练、一致并紧凑的。

这种够用就好的哲学有两个重要的优点。首先，C++的关键字和语法可以应用于所有可以使用 C++的环境。也就是说，C++的核心特征对于所有的应用程序都有效，而不依赖于执行环境。对于依赖于执行环境的功能，如多线程，C++交给操作系统来处理，操作系统能够有效地处理这些问题。因此，C++没有试图采用一种“万能”的解决方案，因为那样做会降低运行效率。

其次，一种改进的、逻辑上一致的语法应能够清晰地表达复杂的结构。当程序变得庞大的时候，这一点就显得尤其重要。当然，笨拙的程序员编写笨拙的 C++代码，优秀的程序员编写清晰而简明的代码。这种能清楚地表示出复杂逻辑的能力是 C++成为广泛应用的程序设计语言

的原因之一。

1.2 功能强大的库

当然，现代程序设计环境需要许多超出 C++关键字和语法支持的功能。C++通过标准库提供了这些功能。C++定义了任何主流语言都可以使用的设计良好的库。它的函数库由 C 语言沿袭而来，包含了范围广泛的一组非面向对象的函数，如 char*字符串的处理、字符处理以及转换函数，这些函数被程序员广泛应用。C++类库为 I/O、字符串以及 STL 等提供面向对象的支持。

由于依赖于库例程而不是关键字，因此只需要简单地扩展库，而不是发明新关键字，就可以把新的功能加入到库中。这使得 C++可以适应程序设计环境的改变，而不需要改变核心语言。因此，C++兼顾了看起来有点冲突的特性：稳定性和灵活性。

即使在它的函数库和类库中，C++也采用了一种够用就好、“化繁为简”的方法，从而避免了“万能”陷阱。这个库仅提供了可以为广泛的编程环境合理实现的功能。对于适用于特定环境的特殊功能，C++通过操作系统提供。因此，C++程序员可以使用执行平台的所有功能。这种方法使您可以编写最大限度使用执行环境的属性和能力的高效代码。

1.3 STL

标准类库中有一个部分特别重要，因而有必要拿出来单独讨论：这就是 STL(Standard Template Library，标准模板库)，STL 的创建改变了程序员思考和使用语言库的方式。其影响如此深远，以至于影响了后出现的语言的设计。例如，Java 和 C#的 Collection 架构就是直接按照 STL 的模式创建的。

标准模板库(STL)的核心是一套完善的模板类与函数，它实现了许多常用的数据结构，标准模板库将其称为容器。例如，STL 包含支持向量、链表、队列和堆栈的容器。由于 STL 是由模板类和函数构成的，因此其容器几乎可用于任何数据类型。因此，STL 提供了解决各种编程问题的“即用”解决方案。

尽管 STL 对于 C++程序员的实际重要性不会被低估，但是仍然有一个很重要的原因使得它非常重要。它是软件组件革命的先驱。由此开始，程序员开始寻找重用代码的方法。因为开发和调试是一个代价很高的过程，所以代码重用非常符合需求。在早期，代码重用是通过将一个程序的源代码剪切并粘贴到另一个程序中完成的。(当然，现在这个方法也有效)。后来，程序员创建了可以重复使用的函数库，如 C++提供的那些库。紧接着就是标准类库。

在类库的基础上，STL 更进一步地采用了一个重要的概念：将库模块化为适用于多种数据的通用组件。另外，由于 STL 是可扩展的，因此您可以定义自己的容器，添加自己的算法，甚至改写内建的容器。扩展、修改以及重用功能的能力是软件组件的本质。

现在，组件软件革命已经接近尾声，很容易忘记是 STL 革命，包括模块化的功能、标准化的接口以及通过继承的可扩展性。计算的历史将把 STL 作为语言设计中重要的里程碑编入史册。

1.4 程序员控制一切

在此有两种程序设计语言的理论。一些人认为语言应该具有避免会在第一“现场”引发问题的特性，从而阻止程序员出现错误。这听起来很不错，但是这样做通常会将某些功能强大的特性被限制、削弱甚至被完全排除，虽然这些特性具有潜在的危险性。这种特性的两个示例是指针和显式内存分配。指针被认为是危险的，因为初学编程的程序员经常滥用它，并且会打破(在某些情况下)安全屏障。显式内存分配(如通过new和delete分配和释放内存)被认为是危险的，因为程序员可能不明智地分配一大块的内存，并且在不需要这块内存的时候忘记释放它，从而引发内存泄漏。尽管这两种特性都具有危险性，但它们使程序员可以直接操作内存，创建高效代码。幸运的是，C++不赞成这种理论。

第二种理论(C++坚持这种理论)就是“程序员是国王”。这意味着程序员控制一切。语言与您是否成为一位糟糕的程序员无关。语言的主要目标是给程序员一个灵活的、谨慎的工作环境。如果您是一位优秀的程序员，您的工作将会表现出您的优秀。如果您是一位糟糕的程序员，也会通过工作表现出来。总之，C++给您提供这些功能，然后就由您自己发挥才能。C++程序员永远不会“和语言发生冲突”。

显然，大多数程序员都赞成C++的理论。

1.5 细节控制

C++不仅给程序员提供控制的能力，它还提供细节控制。例如，考虑自增运算符：“++”。您知道，C++定义了它的前缀和后缀版本。前缀版本在获取其值之前增加操作数。后缀版本在获取值之后增加操作数。当执行包含“++”运算符的表达式时，您获得对表达式执行方式的精确控制。另一个细节控制的示例是register说明符。通过使用register说明符修改变量声明，通知编译器优化对此变量的访问。这样，您可以控制哪一个变量具有最高的优化优先权。C++给予程序员的细粒度控制能力是C++取代汇编语言成为编写系统代码的首选语言的原因之一。

1.6 运算符重载

C++最重要的特性之一就是运算符重载，因为它支持类型扩展。类型扩展可以将新数据类型加入并完全整合到C++程序设计环境中。类型扩展是创建在两个特性之上。第一个是类，允许定义新的数据类型。第二个是运算符重载，允许定义与类相关的多种运算符的含义。通过运算符重载和类，可以创建新的数据类型，然后用操作内建类型相同的方式来操作这种类型：通过运算符。

类型扩展功能非常强大，因为它使得C++成为一个开放而不是封闭的系统。例如，假定需要管理三维坐标。可以通过创建名为ThreeD的新数据类型，然后定义基于这种类型对象的运算符。例如，可以使用运算符“+”来相加两个ThreeD坐标，或者使用运算符“==”来判断两组坐标是否相等。这样，可以编写操作ThreeD的代码，就像操作任何内建类型那样，如下所示：

```
ThreeD a(0, 0, 0), b(1, 2, 3), c(5, 6, 7);
```

```
a = b + c;
// ...
if(a == c) // ...
```

如果没有运算符重载，那么对 ThreeD 对象的操作将不得不通过调用函数来处理，如 addThreeD()或者 isEqualThreeD()，这是一种可读性差的方法。

1.7 一种简洁精练的对象模型

C++的对象模型是简洁的杰作。在 C++的 ISO 标准中，对于对象模型的描述还不到一页(准确地讲，是 6 段)。在如此少的段落中，这个标准解释了对象的本质、对象的生存期以及多态性。例如，这个标准给出了对象的定义，“对象是一个存储区域”。正是这种基础的定义使得 C++的对象模型如此的出众。

当然，在 C++标准中花费了很多的页面来描述为了支持对象所必须的语法和语义，包括对象的创建、销毁和继承等。然而，这是因为 C++给予了对对象的深入控制，而不是因为对象模型的怪异或者不一致。更重要的是，由于优雅的设计，C++对象模型也是被 Java 和 C#语言采用的模型。

1.8 C++发展史

由 Dennis Ritchie 于 20 世纪 70 年代创建的 C 语言标志着程序设计的根本性转变的开始。尽管某些早期的语言，特别是 Pascal，已经获得了巨大的成功，然而是 C 语言创建了影响计算机语言产生的范例。C 语言标志着程序设计新时代的开始。

在 C 语言创建之后不久，出现了新的概念：面向对象的程序设计(OOP)。尽管我们现在认为 OOP 的出现是理所当然的，但是在发明它的那个时代，这确实向前迈出了重要的一步。面向对象的理念很快吸引了程序员的注意，因为它提供了一种强大的新方法来完成程序设计工作。在那个时候，程序变得越来越大，并且其复杂度也在增加。因此需要采取一些措施来处理这种复杂性，OOP 提供了一种解决方案。OOP 使得复杂的大程序可以划分为功能性的单元(对象)。这样做使得复杂的系统分解为容易管理的部分。随之出现的问题是 C 语言不支持对象。

由 Bjarne Stroustrup 设计的 C++语言建立在 C 语言的基础之上。Stroustrup 向 C 语言中加入了面向对象程序设计需要的新的关键字和语法。通过向流行的 C 语言加入面向对象特性，Stroustrup 使得成千上万的程序员转向 OOP 成为可能。随着 C++语言的创建，程序设计的新纪元完全实现了。用一个权威人士的话来说，Stroustrup 创建了世界上功能最强大的计算机语言，并且指明了未来语言发展的方向。

尽管 C++语言的发展刚刚开始，但它已经导致了两种重要语言的出现：Java 和 C#。除了稍有区别之外，Java 和 C#的语法、对象模型以及全部的“外观和感受”都非常类似于 C++。另外，Java 和 C#的库的设计中也有 C++的影子，Java 和 C#的 Collection 架构直接由 STL 派生而来。C++的奠基设计对于整个程序设计影响巨大。

C++给程序员提供的强大功能是 C++如此重要的原因。它广泛的影响是它一直成为全世界程序员的卓越语言的原因。

第2章 简单的C++垃圾回收器

回顾整个计算的历史，关于管理动态分配内存使用的最优方法一直存在争议。动态分配的内存是在运行期间从堆中获得的内存，堆是供程序使用的自由存储区域。堆通常也称为自由存储区或者动态内存。动态分配非常重要，因为它使得程序在执行期间可以获取、使用、释放，然后重用内存。因为几乎所有实际的程序都会以某种形式使用动态分配，所以管理动态分配的内存的方法对于程序的结构和性能有着深远的影响。

通常，有两种方法处理动态内存。第一种方法是手工方式，程序员必须显式地释放不再使用的内存，从而使得这块内存可以重用。第二种方法依赖于一种自动处理，通常称其为垃圾回收，当某块内存不再需要时会自动被回收器回收。两种方法各有千秋，随着时间的推移，比较好的策略是交替使用这两种方法。

C++使用手工的方法管理动态内存。Java 和 C#使用垃圾回收机制。Java 和 C#是比较新的语言，当前计算机语言设计的倾向是使用垃圾回收器。然而，这并不意味着C++的程序员被放置在“历史的对立面”。因为C++具有内建的强大功能，为C++建立一个垃圾回收器是可能的，甚至是很容易的。因此，C++程序员拥有两者的优点：既可以手工管理动态内存的分配，也可以在需要时使用自动的垃圾回收器。

本章为C++开发一个完整的垃圾回收子系统。在开始时，很重要的一点是要理解这个垃圾回收器并没有取代C++内建的动态分配方法。而只是对它进行了补充。因此，在同一个程序中可以使用手工的管理和垃圾回收系统。

本书选用垃圾回收器作为第一个示例，是因为不仅代码本身是有用的(并且是迷人的)，而且它还显示了C++不可超越的强大功能。通过使用模板类、运算符重载以及C++的继承能力来处理计算机操纵的底层元素，如内存地址，可以透明地向C++中加入核心特征。对于大多数的其他语言来说，改变处理动态分配的方式需要改变编译器本身。然而，由于C++给予程序员的空前强大的功能，这个任务可以在源代码层次上完成。

这个垃圾回收器还显示了如何将一个新的类型定义并整合到C++的程序设计环境中。这种类型可扩展性(type extensibility)是C++的关键部分，经常会被忽略。最后，这个垃圾回收器证明了C++“与机器联系紧密”的能力，因为它操纵并管理了指针。不同于其他阻止对底层细节访问的语言，C++允许程序员在需要的时候充分接近硬件。

2.1 两种内存管理方法的比较

在为C++开发垃圾回收器之前，比较垃圾回收和内建于C++中的手工方法是有好处的。通常，在C++中使用动态内存需要两个步骤。首先，通过new从堆中分配内存。然后在不需要这块内存的时候，使用delete释放它。因此，每一次动态分配都要遵循下面的顺序：