HZ BOOKS

PEARSON Addison Wesley

Modern C++ Design C++ 设计新思维

More Effective C++ 中文版

35个改善编程与设计的有效方法

More Effective C++: *35 New Ways to Improve Your Programs and Designs*

（美） Scott Meyers 著
刘晓伟 译

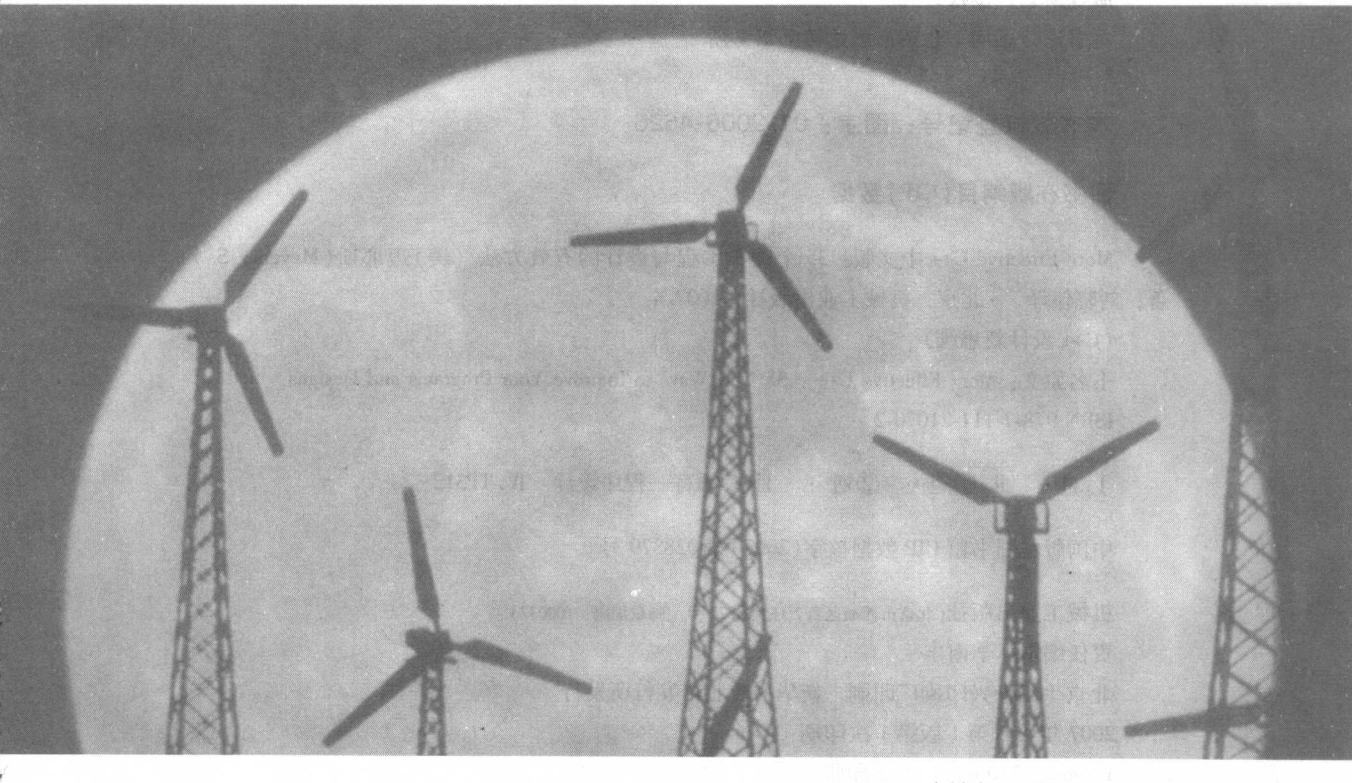
机械工业出版社
China Machine Press

C++ 设计新思维

More Effective C++ 中文版

35个改善编程与设计的有效方法

More Effective C++：*35 New Ways to Improve Your Programs and Designs*



（美） Scott Meyers 著
刘晓伟 译

机械工业出版社
China Machine Press

本书通过示例的方式详细论述了使用 C++ 进行程序设计的高效方法。全书共 6 章，包括分区指针、引用、类型转换、运算符、异常、效率、引用计数、代理类以及虚函数等内容。

本书内容全面丰富，论述详实清晰，作者权威且经验丰富，是 C++ 程序员必备读物。

本书版权登记号：图字：01-2006-4626

图书在版编目(CIP)数据

More Effective C++ 中文版：35 个改善编程与设计的有效方法/(美)迈耶斯(Meyers, S.)著；刘晓伟译. -北京：机械工业出版社，2007.4

(C++ 设计新思维)

书名原文：More Effective C++ : 35 New Ways to Improve Your Programs and Designs

ISBN 978-7-111-21070-2

Ⅰ. M…　Ⅱ. ①迈…　②刘…　Ⅲ. C 语言—程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2007)第 028879 号

机械工业出版社(北京市西城区百万庄大街 22 号　邮政编码　100037)

责任编辑：李南丰

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2007 年 4 月第 1 版第 1 次印刷

186mm × 240mm · 15.5 印张

定价：35.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：(010)68326294

“C++设计新思维”丛书前言

自C++诞生尤其是ISO/ANSI C++标准问世以来，以Bjarne Stroustrup为首的C++社群领袖一直不遗余力地倡导采用“新风格”教学和使用C++。事实证明，除了兼容于C的低阶特性外，C++提供的高级特性以及在此基础上发展的各种惯用法可以让我们编写出更加简洁、优雅、高效、健壮的程序。

这些高级特性和惯用法包括精致且高效的标准库和各种“准标准库”，与效率、健壮性、异常安全等主题有关的各种惯用法，以及在C++的未来占据更重要地位的模板和泛型程序设计技术等。它们发展于力量强大的C++社群，并被这个社群中最负声望的专家提炼、升华成一本本精彩的著作。毫无疑问，这些学术成果必将促进C++社群创造出更多的实践成果。

我个人认为，包括操作系统、设备驱动、编译器、系统工具、图像处理、数据库系统以及通用办公软件等在内的基础软件更能够代表一个国家的软件产业发展质量，迄今为止，此类基础性的软件恰好是C++所擅长开发的，因此，可以感性地说，C++的应用水平在一定程度上可以折射出一个国家的软件产业发展水平和健康程度。

前些年国内曾引进出版了一大批优秀的C++书籍，它们拓宽了中国C++程序员的视野，并在很大程度上纠正了长期以来存在于C++的教育、学习和使用方面的种种误解，对C++相关的产业发展起到了一定的促进作用。然而在过去的两年中，随着.NET、Java技术吸引越来越多的注意力，中国软件产业业务化、项目化的状况愈发加剧，擅长于“系统编程”的C++语言的应用领域似有进一步缩减的趋势，这也导致人们对C++的出版教育工作失去了应有的重视。

机械工业出版社华章分社决定继续为中国C++“现代化”教育推波助澜，从2006年起将陆续推出一套“C++设计新思维”丛书。这套丛书秉持精品、高端的理念，其作译者为包括Herb Sutter在内的国内外知名C++技术专家和研究者、教育者，议题紧密围绕现代C++特性，以实用性为主，兼顾实验性和探索性，形式上则是原版影印、中文译著和原创兼收并蓄。每一本书相对独立且交叉引用，篇幅短小却内容深入。作为这套丛书的特邀技术编辑，我衷心希望它们所展示的技术、技巧和理念能够为中国C++社群注入新的活力。

荣　耀
2005年12月
南京师范大学
www.royaloo.com

对本书的赞誉

这是一本在 C++ 的诸多方面都给人以启迪的书籍：不论是在很少用到的语言特性上，还是在你自以为十分熟悉的语言特性上。只有深刻了解 C++ 编译器是如何解释代码的，你才有可能使用这门语言编写出健壮的软件。本书是帮助你达到这种理解层次的一份极具价值的资料。读过本书之后，我感觉像是和一个 C++ 编程大师一起做了一次代码复查(code review)，并从他那里获得了许多真知灼见。

——Fred Wild，技术副总裁，
Advantage Software echnologies

本书包含许多重要的技术，这些技术是编写优良 C++ 程序不可或缺的。本书解释了如何设计和实现这些思想，并且指出了其他某些替代方案中潜在的一些陷阱。本书也详细解释了一些 C++ 的新特性。任何想要运用好这些新特性的人，最好手头有一本本书，以备查阅。

——Chrisopher J. Van Wyk，教授
Mahematics and Computer Science，Drew University

要编写具有工业强度的 C++ 软件，这是一本最好的书籍。对于已经阅读过《Effective C++》的人，本书是完美的姊妹篇。

——Eric Nagler，C++ 讲师及技术作者，
Univesity of California Santa Cruz Extension

这本书是 Scott 继他的第一本书《Effective C++》之后撰写的另一本全面且极具价值的书籍。我相信每一位专业 C++ 软件开发人员都应该阅读并记住《Effective C++》和《More Effective C++》两本书中的各个条款。这些条款涵盖了一些不容易理解但是又很重要的语言特性。我强烈推荐这两本书给软件开发人员、测试人员、管理人员等。每个人都可以从 Scott 专家级的知识与卓越的文字表达中获益。

——Steve Burkett，软件咨询师

译 者 序

《More Effective C++》英文版[⊖]自第一次印刷以来，至今销售已超过数十万册，译者手头的英文原书是2005年8月份第21次印刷的版本。本书的经典性和它的姊妹篇《Effective C++》一样毋庸置疑。相比《Effective C++》，本书对于运算符重载、默认构造函数、异常处理、延迟计算(lazy evaluation)、代理类(proxy class)、智能指针(smart pointer)、双重分派(double-dispatching)等主题进行了深入和详尽的讨论。毫无疑问，这些主题是每个从事C++程序开发的专业人员都应该掌握的内容。另一方面，正如作者在本书英文版的网页上所指出的那样，随着以模板为代表的C++新技术的逐步深入人心，本书的某些条款需要以发展的眼光去看待，比如说，boost::assign库以提供反例的形式为本书中关于运算符重载的条款给出了更好的注解；以shared_ptr为代表的boost智能指针库和《Modern C++ Design》一书提供的基于Policy的智能指针设计，都针对智能指针这个主题提供了更为完善的解决方案。此外，本书使用相当篇幅进行讨论的双重分派(double-dispatching)问题，在《Modern C++ Design》一书中也通过使用模板给出了更为完美的解决方案。鉴于此，对这些主题有兴趣的读者在阅读完本书以后，可以进一步研究《Modern C++ Design》一书中的相关内容。

在此，首先要感谢《Imperfect C++》和《Exceptional C++ Style》等书的译者刘未鹏把我介绍给机械工业出版社华章公司的陈冀康编辑。感谢我的同事吉子军和葛小果不厌其烦地与我讨论技术上的一些细节问题。另外，还要感谢陈高兵和乔茜在我翻译此书的过程中为我提供了舒适的工作环境。

关于本书，译者惟一的心愿就是希望这个译本没有辜负英文原版的名声。如果您对这个译本有任何意见或者建议，您可以通过liuxiaoweide@gmail.com与我联系。另外，我也会在http://blog.csdn.net/lxwde上维护本书的一个勘误表。

刘晓伟

2006年12月

⊖ 《More Effective C++》的英文影印版已由机械工业出版社引进出版。

引　言

对 C++ 程序员而言，现在是令人振奋的时代。尽管 C++ 商业化尚不足 10 年，却已然成为几乎所有主要计算平台的系统编程语言。越来越多的面临挑战性编程问题的公司和个人不断投入 C++ 的怀抱，那些尚未使用 C++ 的人们则通常被问及何时（而非是否）开始使用 C++。C++ 标准化工作本质上已经完成，其附带的标准库范围之广（涵盖并胜过 C 标准库），使我们得以在不牺牲移植性或不必从头实现常用算法和数据结构的情况下编写出丰富的复杂程序。C++ 编译器数量不断增加，它们提供的语言特性持续扩张，产生的代码质量也不断得到改善。用于 C++ 程序开发的工具和环境越发丰富、强大且健壮。商业库几乎可以消除在很多应用领域中编写代码的需要。

由于 C++ 语言已经成熟并且我们对其使用经验日益增多，我们需要的信息也发生了变化。在 1990 年，人们希望知道 C++ 是什么。而到了 1992 年，他们则希望知道如何使用它。今天，C++ 程序员则提出了更高级的问题：如何设计软件才能使其适应将来的需要？如何在不危及正确性和易用性的前提下提高代码的效率？如何实现语言未直接提供支持的复杂功能？

在本书中，我将回答这些问题以及其他许多诸如此类的问题。

本书向你展示如何设计和实现更有效的 C++ 软件，即，行为的正确性有着更好的保证、发生异常时表现更为健壮、更高效、移植性更好、更好地运用了语言特性、更优雅地适应变化、在混合语言环境中工作得更好、更易被正确使用、更不易被误用的 C++ 软件。简而言之，就是设计和实现出更好的软件。

本书内容被划分为 35 个条款。每一个条款都总结了 C++ 编程社群在特定主题上的智慧积累。大部分条款以指导方针的形式呈现，伴随每一个方针的解说则描述了该方针为何存在、倘若不遵循该方针将会发生什么后果，以及在什么情形下你有理由违反该方针。

这些条款被分为几大类。一些条款关注特定的语言特性，尤其是你可能缺乏使用经验的较新特性。例如，条款 9 ~ 条款 15 专注于异常主题。另外一些条款则解释如何结合运用语言的各色特性以实现更高级的目标。例如，条款 25 ~ 条款 31 描述如何约束对象创建的个数和地点，如何根据一个以上的对象类型创建表现出“虚拟性”的函数，如何创建“智能指针”，等等。还有其他一些条款讨论更广泛的主题，条款 16 ~ 条款 24 专注于讨论效率。不论一个特定的条款讨论的是什么主题，它们都提供了具有实效的途径。在本书中，你将学习到如何更有效地使用 C++。那些构成大多数 C++ 教材的语言特性描述，在本书中只是作为背景信息出现。

这种讲解方式意味着在阅读本书之前你就应该熟悉 C++。这里假定你已了解类、保护级别、虚函数和非虚函数等，还假定你已经熟悉模板和异常背后蕴藏的理念。但我并不期望你是一位语

言专家，所以当触及不那么为人熟知的 C++ 特性时，我总会给出必要的解释。

本书所述的 C++

本书中描述的 C++ 是 1998 年国际标准委员会定义的 C++ 语言。这意味着我可能使用了你手头的编译器尚不支持的一些语言特性。别担心，我猜对你而言惟一的“新”特性应该是模板，但现在几乎所有编译器都提供了对它的支持。我还使用了异常，但主要局限于条款 9 ~ 条款 15，这几个条款特别专注于讨论异常。如果你手头的编译器不支持异常机制，没关系，这并不会影响你学习本书其余部分内容。进一步而言，即便你手头没有支持异常的编译器，也应该阅读条款 9 ~ 条款 15，因为这些条款检视了任何情况下你都需要理解的议题。

我承认，仅凭标准委员会授意某一语言特性或认可某种实践，并不能保证该语言特性已得到目前编译器的支持，或该实践可应用于已有的开发环境中。当面临理论和实践之间的差异时，我对两者都加以讨论，尽管我更偏向于可以工作的实践。正因为两者都进行讨论，所以当你的编译器和 C++ 标准不一致时，本书可以助你一臂之力，并向你展示如何使用现有构造来模拟你手头的编译器尚未支持的语言特性。当你决定将一些迂回方式转换为新支持的语言特性时，本书亦将给你指导。

由于不同的编译器实现对 C++ 标准的遵从度不同，因此建议你至少在两种编译器环境下编写代码。这有助于让你避免无意中依赖于某个厂商的专有语言扩展或它对标准的曲解，还有助于让你避免使用“新锐”编译器技术（例如，只有一家厂商提供的新语言特性支持）。此类语言特性通常实现得不够好（充满 bug 或速度慢或兼而有之），而且在对它们的介绍方面，C++ 社群尚缺乏经验，无法为你提供如何正确地使用它们的忠告。摧枯拉朽固然令人兴奋，但当你的目标是生产可靠的代码时，最好还是能够让他人在一头扎入之前先试试水之深浅。

你将在本书中看到两个你可能不太熟悉的 C++ 构造，两者都是相对较晚出现的语言扩展。一些编译器支持它们，但如果你的编译器不支持，可以很容易地利用你熟悉的特性模拟之。

第一个构造是 bool 类型，其值为关键字 true 或 false。如果你的编译器尚未实现 bool，有两种模拟方式。其一是使用全局枚举：

```
enum bool { false, true };
```

这种方式允许你根据函数带有一个 bool 还是 int 对其进行重载，缺点是内建的比较操作符（即 == 、 < 、 >= 等）仍然传回 int。结果导致如下代码的行为不像我们预期的那样：

```
void f(int);
void f(bool);
int x, y;
...
f( x < y );    //调用 f(int)，其实应该调用 f(bool)
```

当你将代码提交给真正支持 bool 类型的编译器时，这种采用枚举模拟 bool 的方式可能会导致代码的行为发生改变。

另一种替代方式是使用 typedef 来定义 bool，并以常量对象表示 true 和 false：

```
typedef int bool;
const bool false = 0;
const bool true = 1;
```

这种方式和传统的 C/C++ 语义兼容，并且当使用这种模拟方式的程序被移植到一个支持 bool 类型的编译器时，其行为不会发生改变。缺点是当对函数进行重载时无法区分 bool 和 int。总之，这两种模拟方法都有道理，请选择最适合你所处环境的那一种。

第二个新构造其实包含四个构造，即转型操作符 static_cast、const_cast、dynamic_cast 以及 reinterpret_cast。如果你不熟悉这些转型操作，请翻到条款 2 并阅读全部内容。它们不仅仅比所取代的 C 风格的转型做得更多，而且做得更好。在这本书中，任何时候当需要执行一个转型操作时，我都使用这些新风格的转型操作符。

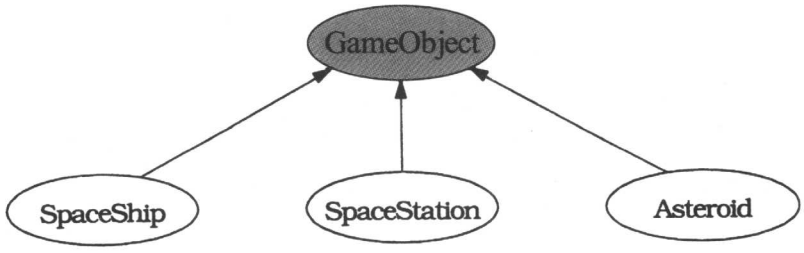
C++ 拥有的不仅仅是语言自身，它还有一个标准库。只要有可能，我就会使用标准 string 类来代替原生的 char * 指针，并且建议你也这么做。string 对象并不比基于 char * 的字符串难用，而且还使你避免考虑大部分内存管理问题。此外，如果抛出了异常，string 对象不大会发生内存泄漏问题（见条款 9 和条款 10）。一个有着良好实现的 string 类的效率可以与其 char * 等价物的效率相媲美，甚至更好（参见条款 29，洞察是如何做到这一点的）。如果你手头无标准 string 类可用，当然可以使用类似于 string 的类。建议使用它，因为几乎任何东西都比原生的 char * 要好。

任何时候只要有可能我就会使用来自标准库的数据结构。这类数据结构取自标准模板库（Standard Template Library，即 STL，见条款 35）。STL 包含 bitsets、vectors、lists、queues、stacks、maps、sets 以及更多的东西，应该优先使用这些标准化的数据结构，而不要抵制不住诱惑自己编写一个特殊的版本。你的编译器也许没有携带 STL，但不要因此就不使用它。感谢 SGI，你可以从 SGI 的 STL Web 站点（http：//www. sgi. com/tech/stl/）下载一份免费的拷贝，它可以和许多编译器协作。

如果你目前正使用一个包含各种算法和数据结构的库并且感到满意，那就不要仅仅因为 STL “标准” 就转而使用它。然而，如果你在使用一个 STL 组件和从头编写你自己的代码之间进行选择，那么应该选择使用 STL。还记得代码复用吗？STL（以及标准库的其他组件）中有大量很值得复用的代码。

约定与术语

在本书中，任何时候当我提到继承时，都是指公有继承。如果不是指公有继承，我会明确地予以说明。当绘制继承层次结构图时，我通过绘制从派生类指向基类的箭头来描述基类和派生类之间的关系。例如，以下是一幅来自条款 31 的继承层次结构图：



这种表示法和我在《Effective C++》第1版（不是第2版）中使用的约定相反，但我现在可以确信大多数C++实践者都是绘制从派生类指向基类的继承箭头线，我很高兴随大流。在这类图形中，抽象类（例如GameObject）被加上阴影而具体类（例如SpaceShip）则未加阴影。

继承导致对象的指针和引用具有两种不同的类型，分别是静态类型和动态类型。指针或引用的静态类型是指其声明时的类型。动态类型则由它实际指向的对象的类型确定。下面是基于上述类层次结构编写的一些例子代码：

```
GameObject * pgo =                  // pgo 的静态类型是 GameObject* ,
new SpaceShip;                      // 动态类型是 SpaceShip*
Asteroid * pa = new Asteroid;       // pa 的静态类型是 Asteroid* ,
                                    // 动态类型也是 Asteroid*
pgo = pa;                           // pgo 的静态类型仍然是(并且总是)GameObject* ,
                                    // 动态类型现在变成了 Asteroid*
GameObject& rgo = * pa;             // rgo 的静态类型是 GameObject,
                                    // 动态类型是 Asteroid
```

这些例子也示范了我喜欢的一种命名约定。pgo是一个指向GameObject的指针，pa是一个指向Asteroid的指针，rgo是一个指向GameObject的引用。我通常以这种方式为对象的指针和引用命名。

我特别喜欢的两个参数名称是lhs和rhs，它们分别是“left-hand side”和“right-hand side”的缩写。为了理解这些名字背后的理念，考虑一个用于表示有理数的类：

```
class Rational {… };
```

如果我想要一个用于比较一对Rational对象的函数，可以将其声明如下：

```
bool operator = = (const Rational& lhs, const Rational& rhs);
```

这让我能够写出如下所示的代码：

```
Rational r1, r2;
…
if (r1 == r2)…
```

在对operator ==的调用中，r1出现于“==”的左侧并被绑定到lhs，而r2则出现在“==”的右侧并被绑定到rhs。

我使用的其他缩写还有：ctor表示“constructor”，dtor表示“destructor”，RTTI则表示C++对

“runtime type identification”提供的支持（在此上下文中，dynamic_cast 是一个最常被使用的组件）。

当你分配内存而没有释放它时，就会面临内存泄漏问题。C 和 C++ 中都存在内存泄漏问题，但在 C++ 中内存泄漏“泄漏”的可能不仅仅是内存。因为当对象被创建时，C++ 会自动调用其构造函数，而构造函数自身可能会分配资源。例如，考虑以下代码：

```
class Widget {… };                    // 某个类(它是什么并不重要)
Widget * pw = new Widget;             // 动态分配一个 Widget 对象
…                                     // 假设 pw 从未被删除
```

这段代码会泄漏内存，因为 pw 指向的 Widget 对象从未被删除。而且，如果 Widget 构造函数分配了“本应在 Widget 对象被销毁时释放”的附加资源（例如，文件描述符、信号量、窗口句柄以及数据库锁等），那么这些资源也如同内存那样丢失了。为了强调在 C++ 中内存泄漏往往也会泄漏其他资源，在本书中，我通常说资源泄漏而不是内存泄漏。

你将不会在这本书中看到很多内联函数。这并不是因为我不喜欢内联。恰恰相反，我坚信内联函数是 C++ 的一个重要特性。然而，由于用于决定一个函数是否应被内联的准则可能会很复杂、微妙且与平台有关，因此我尽量避免内联，除非有一个我期望进行内联的关键之处。当你在本书中看到一个非内联函数时，并不意味着我认为把它声明为 inline 是个坏主意，只是因为是否内联该函数与我正在讨论的主题无关。

有一些 C++ 特性已经被标准委员会摒弃。这样的特性将最终被从语言中移除，因为较新的特性已经加入，它们可以做被废弃的特性所做的工作，而且做得更好。在这本书中，我将会介绍被摒弃的构造，并说明取代它们的语言特性。应该尽量避免使用被废弃的特性，但也没理由过度在意对它们的使用。原因在于，为了为顾客保持向后兼容性，编译器厂商往往会支持废弃的特性很多年。

书中所言的客户（Client）是指使用你编写的代码的人（程序员）或物（通常指类或函数）。例如，如果你编写了一个 Date 类（用于表示生日、最后期限等），任何使用该类的人就是你的客户，此外，任何使用了 Date 类的代码片断也是你的客户。客户很重要，实际上，客户正是问题实质之所在！道理很简单，如果没有人使用你编写的软件，那又编写它作甚？你会发现我处心积虑让客户的日子好过一些，通常这会让你的日子更难过，因为优秀的软件总是以客户为中心，客户就是上帝。如果这种说法让你感觉我用情太滥，不妨从利己主义的角度考虑一下。你曾使用过自己编写的类或函数吗？如果是，你就是你自己的客户。所以让客户更轻松，通常也就是让自己更轻松。

当讨论类模板或函数模板以及由它们产生的类或函数时，我保留了偷懒的权利，对模板及其实例之间的差别不加区分。例如，如果 Array 是一个接受类型参数 T 的类模板，我可能将该模板的特定实例称为 Array，尽管 Array < T > 才是该类的真正的名字。类似地，如果 swap 是一个接受类型参数 T 的函数模板，我可能以 swap 取代 swap < T > 来表示其实例。为了防止在某些情况下这种速记法不够清晰，我会在谈到模板实例时带上模板参数。

报告 bug、提供建议、获取更新

我已尽力使这本书精确、可读性好、有用，但我知道它必定有改善的余地。如果你发现任何种类的错误，不管是技术性的、语言上的、排版方面的，抑或任何其他方面的，请告诉我。我将努力在本书重印时予以纠正。如果你是某个错误的第一个报告者，我将很高兴将你的大名加入本书的致谢辞中。如果你有其他改善建议，我同样欢迎。

我将继续收集在 C++ 中有效地编程的指导方针。如果你有新指导方针的想法并愿意与我分享，我将非常高兴。请将你的指导方针、评论、批评以及 bug 报告邮寄到以下地址：

Scott Meyers

c/o Editor-in-Chief, Corporate and Professional Publishing

Addison-Wesley Publishing Company

1 Jacob Way

Reading, MA 01867

U. S. A.

或者，你也可以发送电子邮件到 mec++ @ awl. com。

我维护着一份自本书首次印刷以来的修订列表，其中包括错误修正、文字澄清以及技术更新。这个列表连同其他相关信息，可从本书网站（http：//www. awl. com/cp/mec++. html）获得。你也可以通过匿名 FTP 从 ftp. awl. com 的 cp/mec++ 目录中获取。如果你希望拥有这份修订列表，但无法上网，请向以上地址发信申请，我会寄一份给你。

如果你希望当我对本书做出修改时得到通知，可以考虑加入我的邮件列表，请访问 http：//www. aristeia. com/MailingList/index_ frames. html。

闲话少说，让我们开始揭示之旅！

致　　谢

本书在许多人的帮助下才得以完成。有些人为本书中一些相关技术的主题提供了自己的观点，有些人为本书的出版提供了很大的帮助，还有一些人在我撰写本书的时候给我的生活带来了更多的乐趣。

当一本书的贡献人数很多时，有时候作者就会选择不列出每个贡献者的名字，而是用下面这句话代替“本书的贡献者非常之多，所以在此不一一列出。”我倾向于 John L. Hennessy 和 David A. Patterson 在《Computer Architecture: A Quantitative Approach》（Morgan Kaufmann，第 1 版，1990）一书中所采用的做法：给出贡献者的完整列表。他们这本书除了提供详细的致谢名单值得我效仿以外，还提供了 90 - 10 准则的一些第一手数据，我在本书的条款 16 中也采用了这个做法。

本书中的条款

除了那些直接引用的文字，本书的其他文字均出自我的手笔。但是，我所讨论的很多思想则是由别人提出来的。我尽自己最大的努力记录谁贡献了什么（但是我知道，有些信息的来源我自己也想不起来了），尤其是很多在 comp. lang. c++ 和 comp. std. c++ 两个新闻组上发帖子的一些人。

C++ 社区中的许多思想都是由许多人各自独立地提出来的。因此，我只能告诉大家我最初是在哪里接触到某个思想的，而不一定会指出这些思想源自何处。

我在条款 2 提到的使用宏来模拟新的 C++ 类型转换操作符的语法，是由 Brian Kernighan 建议的。

在条款 3 里面，我针对使用基类指针删除一个派生类对象的数组提出了警告，这个条款是基于 Dan Saks 的“Gotchas”讲座的一些内容扩展而来的，他在好几个会议和内部讨论上讲到过这个问题。

条款 5 讲到，通过代理类来阻止使用非期望的单参数的构造函数，这个技术是基于 1994 年 1 月份《C++ Report》上 Andrew Koenig 的一篇专栏文章所讲到的内容。

James Kanze 在 comp. lang. C++ 上发过一个帖子，讲解如何通过自增操作符和自减操作符的前缀形式来实现其相应的后缀形式；我在条款 6 中用到了他给出的这种技术。

David Cok 曾经就《Effective C++》中的某些内容给我写过信，他的信让我注意到了 operator new 和 new operator 之间的区别，这也构成了条款 8 的关键内容。即便是在读了他的来信之后，我还是没能真正明白两者之间的区别，但是如果没有他一开始的提醒，我可能到现在还是不明白。

条款 9 中讲到的通过析构函数来阻止资源泄漏，这个思想来自于 Margaret A. Ellis 和 Bjarne

Stroustrup 的《The Annotated C++ Reference Manual》一书的 15.3 小节（见 285 页）。在那本书里，这种技术叫作“资源获得就是初始化（resource acquisition is initialization）”。Tom Cargill 建议我把这种方法的重点从资源获取转移到资源释放上来。

我在条款 11 中所讨论的内容受到了《Taligent's Guide to Designing Programs》（Addison-Wesley，1994）一书第 4 章中一些内容的启发。

我在条款 18 中描述的为 DynArray 类提前分配内存的方法是基于 Tom Cargill 的一篇文章《A Dynamic vector is harder than it looks》，这篇文章发表在 1992 年 6 月份的《C++ Report》上。针对动态数组类的更为复杂的设计可以从 Cargill 后来发表在 1994 年 1 月份的《C++ Report》上的专栏文章中找到。

条款 21 受到了 Brian Kernighan 在 1991 年 USENIX C++ 会议上的一篇论文《An AWK to C++ Translator》的启发。他使用了重载运算符（67%）来处理混合类型的数值计算，尽管他的设计所解决的问题与我在条款 21 中提到的问题无关，但是他的文章促使我考虑使用 multiple overloading 作为临时对象创建的解决方案。

在条款 26 中，我所设计的用于对象计数的模板类，基于 Jamshid Afshar 在 comp. lang. c++ 上发的一个帖子。

使用 mixin 类来跟踪由 operator new 产生的指针（参见条款 27），这个想法是由 Don Box 建议的。Steve Clamage 解释了如何使用 dynamic_cast 来找到一个对象在内存中的起始地址，从而使这个想法在实践中具有了可行性。

条款 28 中针对智能指针（smart pointers）所进行的讨论基于以下数篇文章：Steven Buroff 和 Rob Murray 发表在 1993 年 10 月份的《C++ Report》上的 C++ Oracle 专栏的部分内容；Daniel R. Edelson 发表在 1992 年 USENIX C++ 会议上的经典论文《Smart Pointers：They're Smart，but They're Not Pointers》；Bjarne Stroustrup 的《The Design and Evolution of C++》一书的 15.9.1 小节；Gregory Colvin 在 1995 年的 C/C++ Solutions 培训班上关于“C++ Memory Management”的课堂笔记；以及 Cay Horstmann 在 1993 年 3 月份到 4 月份在《C++ Report》上的专栏文章。另外，我自己也扩展了其中一部分内容。

在条款 29 中，使用基类来存储引用计数以及通过智能指针来操纵这些计数值，这种用法是基于 Rob Murray 在他的《C++ Strategies and Tactics》一书中的 6.3.2 和 7.4.2 小节对相同主题所进行的讨论。在条款 29 中随后所讨论的为已有类添加引用计数的方法来自于 Cay Horstmann 在 1993 年 3 月份到 4 月份在《C++ Report》上的专栏文章。

在条款 30 中，我对于左值内容（lvalue context）的讨论是基于 Dan Saks 1993 年 1 月份在《C User's Journal》（现在是《C/C++ Users Journal》）的专栏文章中的注解。当调用代理类时，非代理的成员函数是不可用的，这个问题的提出来自于 Cay Horstmann 的一篇未曾发表的文章。

使用运行时类型信息（runtime type information）来构建类似于虚函数表（vtbl）的关联数组用以存储函数指针（见条款 31），这种技术是基于 Bjarne Stroustrup 在 comp. lang. C++ 上发的一个帖

子，以及他的《The Design and Evolution of C++》一书的 13.8.1 小节的内容。

条款 33 的内容是基于我 1994 年和 1995 年在《C++ Report》上的一些专栏文章。在我的专栏文章里也包含了 Klaus Kreft 提出的一些意见，这些意见是关于如何使用 dynamic_cast 来实现一个虚 operator =，并且这个操作符还要能够检测到错误的类型参数。

条款 34 的大部分内容是受到 Steve Clamage 在 1992 年 5 月份的《C++ Report》上的文章“Linking C++ with other languages”的启发而写的。同样在该条款中，我谈到了类似于 strdup 这样的函数会带来的问题，这个讨论是受到一个匿名评论者的启发而写的。

关于本书

阅读一本书的草稿并提出意见是非常累人但是又极为重要的一件事情。我要感谢自愿为阅读我的书稿投入时间和精力的那些人。尤其是 Jill Huchital、Tim Johnson、Brian Kernighan、Eric Nagler 和 Chris Van Wyk，他们不止一次地阅读了本书（或者本书的大部分）。完整阅读本书初稿的人包括 Katrina Avery、Don Box、Steve Burkett、Tom Cargill、Tony Davis、Carolyn Duby、Bruce Eckel、Read Fleming、Cay Horstmann、James Kanze、Russ Paielli、Steve Rosenthal、Robin Rowe、Dan Saks、Chris Sells、Webb Stacy、Dave Swift、Steve Vinoski、和 Fred Wild。部分阅读本书初稿的人有 Bob Beauchaine、Gerd Hoeren、Jeff Jackson 和 Nancy L. Urbano。每个审阅本书草稿的人都提出了他们的意见，这些意见使得本书的内容更为准确、实用和详实。

本书出版之后，我收到了许多人发来的错误纠正意见和建议。我按照收到信件的先后顺序列出这些目光敏锐的读者：Luis Kida、John Potter、Tim Uttormark、Mike Fulkerson、Dan Saks、Wolfgang Glunz、Clovis Tondo、Michael Loftus、Liz Hanks、Wil Evers、Stefan Kuhlins、Jim McCracken、Alan Duchan、John Jacobsma、Ramesh Nagabushnam、Ed Willink、Kirk Swenson、Jack Reeves、Doug Schmidt、Tim Buchowski、Paul Chisholm、Andrew Klein、Eric Nagler、Jeffrey Smith、Sam Bent、Oleg Shteynbuk、Anton Doblmaier、Ulf Michaelis、Sekhar Muddana、Michael Baker、Yechiel Kimchi、David Papurt、Ian Haggard、Robert Schwartz、David Halpin、Graham Mark、David Barrett、Damian Kanarek、Ron Coutts、Lance Whitesel、Jon Lachelt、Cheryl Ferguson、Munir Mahmood、Klaus-Georg Adams、David Goh、Chris Morley、Rainer Baumschlager Christopher Tavares, Brian Kernighan, Charles Green, Mark Rodgers, Bobby Schmidt, Sivaramakrishnan J., Eric Anderson, Phil Brabbin, Feliks Kluzniak, Evan McLean, Kurt Miller, Niels Dekker, Balog Pal, Dean Stanton, William Mattison, Chulsu Park, Pankaj Datta, John Newell, Ani Taggu, ChristopherCreutzi, Chris Wineinger, Alexander Bogdanchikov, Michael Tegtmeyer, Aharon Robbins, Davide Gennaro, Adrian Spermezan, Matthias Hofmann, Chang Chen, John Wismar, Mark Symonds, Thomas Kim, and Ita Ryan.。他们的建议使得我在《More Effective C++》的后续印次中不断提升本书的质量，我非常感激他们的帮助。

在写作本书的过程中，对于新出现的 ISO/ANSI C++ 标准，我面临着许多的问题，非常感谢

Steve Clamage 和 Dan Saks 抽时间不厌其烦地给我回信。

John Max Skaller 和 Steve Rumsby 在 ANSI C++ 标准的草稿广泛传播之前，就为我弄到了这份草稿的 HTML 版本。Vivian Neou 告诉我在 16 位的 Microsoft Windows 上 Netsacpe 可以用来作为标准的 HTML 浏览器，我非常感谢 Netscape Communications 公司的那些人把这么好用的浏览器免费共享。

Bryan Hobbs 和 Hachemi Zenad 非常慷慨地给了我一份 MetaWare C++ 编译器内部测试版本的拷贝，这使得我可以检查本书中那些使用了最新的语言特性的代码。Cay Horstmann 帮助我在我不熟悉的 DOS 和 DOS 扩展环境下搭建了编译器的运行环境。Borland（现在是 Inprise）提供了他们最先进的编译器的一个拷贝，Eric Nagler 和 Chris Sells 帮助我在我没法弄到的编译器上测试了本书的代码。

如果没有 Addison-Wesley 出版社专业图书部的同仁们的帮助，本书不可能得以出版，我要特别感谢 Kim Dawley、Lana Langlois、Simone Payment、Marty Rabinowitz、Pradeepa Siva、John Wait 以及其他同仁给予我的鼓励、耐心和帮助。

Chris Guzikowski 帮助起草了本书的封底上的文字，从事低温物理学研究的 Tim Johnson 则挤时间对这些文字给出了批判性的建议。

Tom Cargill 慷慨地同意把他在《C++ Report》上发表的关于异常的文章放到 Addison-Wesley 的网站上。

其他人

Kathy Reed 最早把我带入了编程领域，她对我总是有足够的耐心。Donald French 在我还没有什么名气的时候，就坚信我具备编写 C++ 教程的能力。我要永远感谢 Donald French 把我介绍给 John Wait，后者是 Addison-Wesley 的编辑。在我写作本书的间隙，Jayni Besaw、Lorri Fields 和 Beth McKee 三个人提供了数不清的笑料。

我的妻子 Nancy L. Urbano，在我写作本书的漫长过程中，对我一再迁就。我也一再允诺写完书之后就和她去做一些轻松的事情。现在这本书完成了，我也兑现了我的诺言。我爱她。

最后，我还需要感谢我们的小狗 Persephone，它的存在改变了我们的生活。如果没有它，这本书可能会更早一些写完，我也不会少睡那么多觉，但是我写作的过程也会少了很多有趣的插曲。

目　录