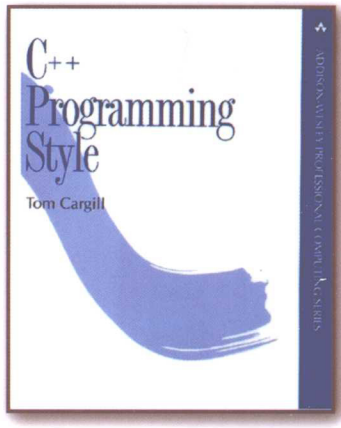
PEARSON

C和C++实务精选
品味岁月积淀，读享技术菁华

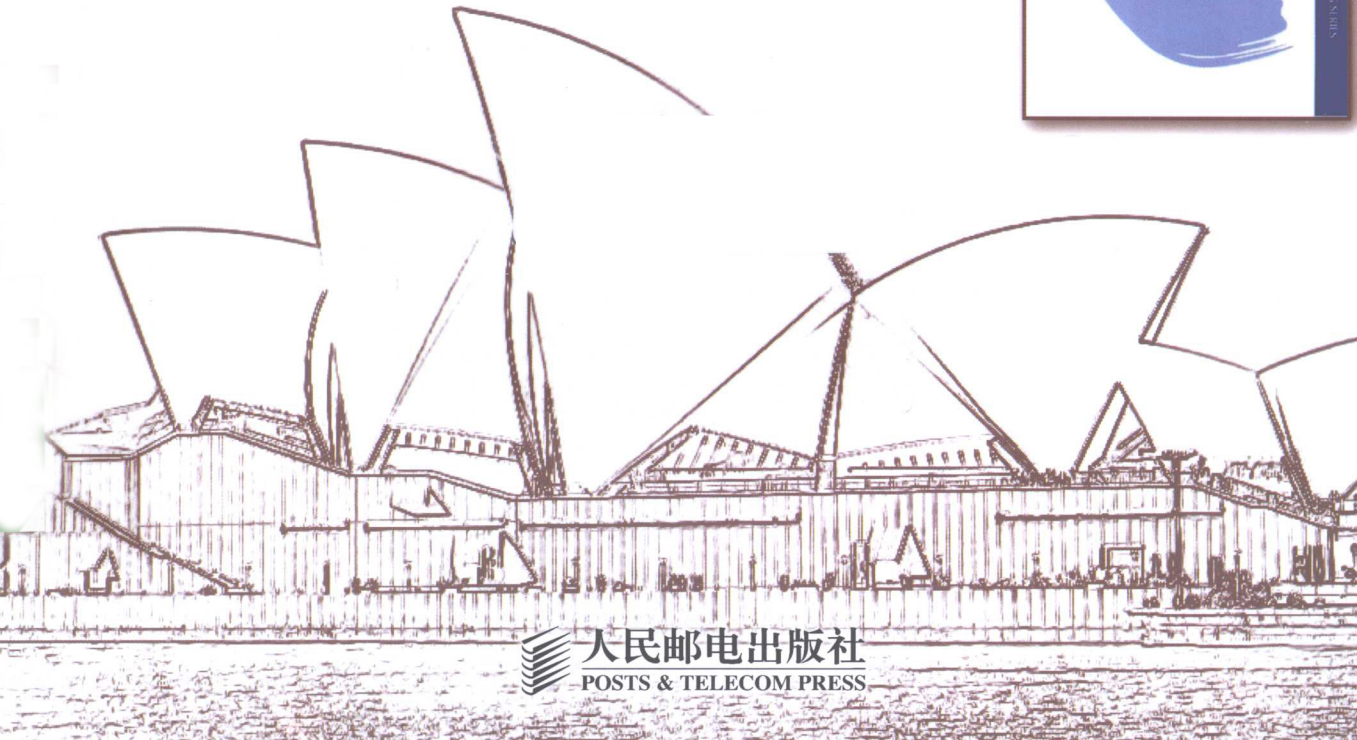
C++ 编程风格

[美] Tom Cargill 著　　聂雪军 译

C++ Programming Style



- C++标准委员会成员Bruce Eckel推荐阅读
- 深入理解C++编程规范，培养良好编程风格
- 全面提升C++大规模编程功力



人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

C++ 编程风格

[美] Tom Cargill 著　　聂雪军 译

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

C++编程风格 / (美) 卡吉尔 (Cargill, T.) 著 ; 聂雪军译. -- 北京 : 人民邮电出版社, 2013.1
ISBN 978-7-115-29506-4

Ⅰ. ①C… Ⅱ. ①卡… ②聂… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP312.

中国版本图书馆CIP数据核字(2012)第232226号

版权声明

C++编程风格

◆ 著 [美]Tom Cargill
译 聂雪军
责任编辑 傅道坤

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：13.75
字数：303 千字 2013 年 1 月第 1 版
印数：1- 3 000 册 2013 年 1 月北京第 1 次印刷

著作权合同登记号 图字：01-2011-6243 号
ISBN 978-7-115-29506-4

定价：39.00 元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

内容提要

本书讲解了 C++ 语言中较深层次的程序设计思想和使用方法，包含大量的软件工程概念和设计模式，重点介绍大规模编程相关的内容，例如增加代码的可读性、可维护性、可扩展性以及提高代码执行效率等的方法。本书的示例代码都是从实际程序中抽取出来的，作者通过对这些代码进行分析，讲解了如何正确地编写代码以及避开一些常见的误区和陷阱，并提炼出了一些关于程序设计风格和编码风格的规则。如果开发人员在编程时能够遵循这些规则，将有助于开发出更好的 C++ 程序。

本书描述平实，示例丰富，适合有一定编程经验的计算机程序设计与开发人员参考。

2012年再版译序

C++的语法特性并不复杂，但想要用好这些特性却不容易。Bjarne Stroustrup曾说过："……学习C++最难的部分莫过于体会编程语言构件的内涵以及集中于应用中的概念，即要学会抽象思维，并且设计时要着重于类而非操作序列……"因此，要想写出优秀的C++代码，掌握C++语法固然重要，但更重要的是掌握面对不同问题时的分析思路和解决方案，这通常也可以统称为编程风格。

本书介绍了一些常见的C++编程风格，重点在于讲解如何掌握C++程序的设计原理和编程实践。本书的内容与特点在2006年版的译序中已经给出了简要介绍，在此不再赘述。在2012年版的重新修订过程中，完成的工作如下所示。

（1）内容勘误

该书在2006年出版后，陆续有读者指出书中存在一些翻译有误或者不到位的地方。在这次修订过程中，我收集了网上论坛、电子邮件以及平时与同事交流中反馈的问题，一并进行了修正。

（2）语言提炼

在此次修订过程中，我将2006年的译稿逐字逐句重新阅读了一遍，试着从读者的角度来发现译稿中语言繁琐或表达不畅的地方，力争在确保翻译内容精确的同时，尽可能提高文字的简洁性和阅读的流畅性。

与6年前相比，我在重新阅读本书时对相同问题有了更深的理解和思考，正可谓"温故而知新"。这是我在修订过程中的最深体会，希望无论是新读者还是老读者，都能从本书中获得新的收获。本书的内容较为浅显易懂，对于想进一步提高编程水平的新手C++程序员来说，可以作为一个不错的开端。

在经过此次修订后，本书的质量将进一步提高，但其中或许还有不妥或者疏漏的地方，欢迎各位读者随时反馈与指正。

聂雪军

2012年7月于武汉

2006 年译序

这是一本能够提升你 C++ 编程功力的书籍。

C++ 是一种功能强大的编程语言，这已是不争的事实。目前，许多关于 C++ 的书籍都只是对语言本身进行了详细的介绍，而对于如何正确地使用 C++ 语言却介绍得不多。这就好比我们拥有了一种强大的武器，却无法有效地发挥它的最大威力。例如，虚函数和动态绑定是 C++ 的强大功能之一，但我们必须首先识别出正确的抽象，才能有效地使用虚函数；否则，盲目地使用虚函数反而会降低程序的性能。与其他的 C++ 语言书籍相比，本书更像是一本通过 C++ 来阐述软件工程思想和设计模式的书籍。书中所讲述的大多数问题都是与大规模编程相关的，更多的重点是放在程序中不同组件之间的交互行为上，而不是诸如命名规则、注释风格等细节问题上。对程序员来说，实现某个功能并不是件难事，困难的是如何使设计和编码更为合理，这其中包括：代码的可读性、可维护性、可扩展性以及执行效率等。通过对本书的学习，你将会掌握正确的设计思想和使用 C++ 的方法。

本书的最大特点就是与实际紧密相连。书中的示例代码都是从实际程序中抽取出来的。译者在翻译本书的时候，总有一种似曾相识的感觉：书中所提到的许多错误，在译者还是新手的时候都曾遇到过，因此能够与作者的分析和讲解产生共鸣。在任何一名新手的成长过程中，总是要经过痛苦的实践和许多不眠之夜，才能够积累一些经验和教训。而在本书中，作者将这些最宝贵的东西一并呈现给了读者。

本书的另一特点就是平实性。书中既没有华丽的辞藻，也没有尖锐的观点，而只是通过普通的叙述方法，将我们在编码过程中所遇到的问题加以剖析。细细读来，就好像是经验丰富的大师正在给新手们讲解如何正确地编写代码，以及避开一些常见的误区和陷阱。

书中所给出的编程规则对于程序员在设计和编码时是很有帮助的。但是切记，任何规则都不是绝对的，在每一条规则后面都有着特定的使用条件。即使是同一条规则，在不同的环境中，所起到的作用也可能是截然不同的。我们应该灵活地使用这些规则，而不是生搬硬套。否则，规则将会变成教条，这就违背了作者的初衷。

译者深知“一份耕耘，一份收获”，因此在翻译过程中不敢有一丝懈怠。对每个不易理解的地方，译者总是仔细斟酌，反复推敲，尽最大努力去将那些精妙的思想用浅显的语言表达出来。然而，由于时间和水平有限，翻译中的疏漏和错误在所难免，还望读者和同行不吝指正。

聂雪军

2006 年 5 月于北京

序

Kernighan 和 Plauger 的经典之作——*The Elements of Programming Style*——从出版至今，已经有大约 20 年的时间，书中给出的一组规则直到现在仍然可以作为最好的编程指导思想。当前，计算机程序正变得日益庞大和复杂，而我们使用的编程语言也已经发生了很大的变化。现在，我们不仅需要关注程序中每个模块的算法和数据结构，还要关注如何将程序中所有的模块更好地组合在一起。DeRemer 和 Kron 提出了两个术语："大规模编程（programming-in-the-large）"和"小规模编程（programming-in-the-small）"。这两个术语分别用来描述程序的"大规模"特性和"小规模"特性。"小规模编程"主要研究程序中"代码只有几页长"的组件，例如一个 C++ 类。而"大规模编程"则是研究如何将"小规模"的组件组合成一个完整的程序——用 C++ 的术语来说，就是研究类之间的关系。在 Kernighan 和 Plauger 的书中，重点讲解了"小规模编程"，对"大规模编程"的讨论虽然有所涉及，但却非常有限，基本上可以概括为以下两点。

1. 将程序模块化。

2. 有效地使用子程序。

本书介绍的编程风格侧重于"大规模编程"，但仅限于 C++ 编程领域。本书所面向的读者，是那些已经学习过 C++ 语言，并正在努力将语言的各种特性——尤其是面向对象的特性——应用于解决编程问题的程序员。虽然书中的讨论仅限于 C++，但对于其他语言来说，其中许多编程经验同样是适用的。本书尽可能地保持了语言的独立性以方便读者阅读。

本书采用了 Kernighan 和 Plauger 的讲解方法，通过对程序进行分析和改写来提炼出编程风格的规则。书中使用的所有程序都是从介绍 C++ 编程的教科书、期刊和手册中节选出来的，我

并没有专门为本书编写示例程序。其中有些程序是原封不动地照搬过来，而有些程序则做了一些修改。这些修改包括改正一些无伤大雅的小错误，以及在保留程序结构的基础上，对那些未获得版权的程序进行的修改。

本书本着“就事论事”的态度来分析代码。我们都是通过阅读和分析彼此的代码来达到学习目的。本书并非要对某个程序员提出批评，而只是努力指出好的程序与坏的程序之间的差别。毫无疑问，本书中的“好”程序也可能有其自身的缺陷。我们鼓励读者对这些程序进行更严格的分析，并找出进一步改善编程风格的方法。

致谢

本书是基于 C++ At Work 和各种 USENIX 会议提供的资料而逐渐形成的。感谢 SIGS 的 Rick Friedman 以及 USENIX 的 John Donnelly 和 Dan Klein，是他们为我提供了这些资料，同时也感谢编写这些资料的人们。特别感谢 Solbourne 公司的程序员们，他们为了给我提供第一份资料而绞尽脑汁。本书还得益于 *The C++ Journal* 中的文章，因此我要感谢编辑 Livleen Singh。我与 Dave Taenzer 曾有过大量的讨论，这些讨论对于本书中的许多主题都有着很好的影响。同样还要感谢 Addsion-Wesley 出版社的 John Wait 对本书的信心和耐心。

David Cheriton、James Coggins、Cay Horstmann、David Jordan、Brian Kernighan、Doug Lea、Scott Meyers、Rob Murray、Kathy Stark 和 Mike Vilot 等审阅了本书的初稿，并提出了许多提高本书质量和消除错误的建议。衷心地感谢他们的工作。

感谢我挚爱的 Carol Meier，在我编写本书的时候，她自始至终都支持着我。在本书的构思和写作期间，我们迎来了我们的第一个孩子。

最后，我还要感谢以下的作者和出版社，感谢他们允许我引用以下书籍中的资料：

Davis，S.R. 1991，Hands-On Turbo C++.Reading, MA:Addison-Wesley.

Dewhurst, S.C. 和 Stark，K.T. 1989. Programming in C++. Englewood Cliffs，NJ：Prentice Hall.

C++ for C Programmers, Ira Pohl(Redwood City, CA:Benjamin/Cummings Publishing Company,1989)p.92.

Stroustrup,B.1991 .The C++ Programming Language,2d ed.Reading,MA:Addsion-Wesley/

Wiener,R.S.，和 Pinson，L.J.1988. An Introduction to Object-Oriented Programming and C++.

Reading,MA:Addsion-Wesley.

Wiener,R.S., 和 Pinson，L.J.1990,The C++Workbook. Reading,MA:Addision-Wesley.

参 考 文 献

Kernighan 和 Plauger[2] 的书籍对所有程序员来说都是值得推荐的。Deremer 和 Kron[1] 提出了“大规模编程”和“小规模编程”这两个术语。

1. DeRemer,F. and Kron，H.“Programming-in-the-large versus Programming-in-the-samll”，Proceeding of the International Conference on Reliable Software，ACM/IEEE，April 1975，Los Angeles，California.

2. Kernighan，B.W. and Plauger，P.J. 1974（2d ed., 1978）.The Elements of Programming Style. New York, NY: McGraw-Hill.

Tom Cargill

Boulder，Colorado

前言

本书采用一种统一的方法来给出所要学习的内容。通过研究示例程序——“编程风格示例”——来引入每个学习主题，这些示例程序通常在某些重要的方面存在着缺陷。在分析程序时，我们采取了与做代码交叉审查时一样的思路：在审查同事的代码时，我们要找出哪些问题是最需要改正的，以及对程序的哪些部分进行修改才能最大程度提升程序的整体性能。在本书中，我们将对每个示例程序做详尽的阅读和分析。读者在阅读书中对示例程序的分析之前，可以首先从自己的角度去分析程序中的问题，然后试着给出自己的解决方案。在分析完示例程序后，我们还将把最初的程序和修改后的程序进行比较；在某些示例程序中，随着对程序的逐步改进，程序的代码量和复杂性都会显著降低。最后，我们还可以对修改后的最终程序做更严格的分析，并努力去找出更多的改进方法。

对于从分析代码中学到的每个主要知识点，都可以用一条规则来进行描述。所有这些规则在内容和侧重点上都有着很大的区别。其中一些规则是具体的、技术上的规则，它们告诉程序员如何避开 C++ 中常见的编程陷阱。而其他规则是关于如何在面向对象程序设计中构建抽象以及抽象之间的关系。所有这些规则都与 C++ 程序设计中的某些特性有着密切关系。

我们需要重点理解在每条规则后面的动机和背景，而不是盲目坚持这些规则。在某些情况下，程序员可以为了达到一个更重要的目标而打破这些规则。事实上，本书中的某些规则之间是相互冲突的。例如，有的规则强调程序的简单性，而有的规则强调程序的完整性，它们往往是互相冲突的。程序员不仅要知道这些规则，而且还要理解其中的具体含义和应用环境，并且能够判断在什么时候应该打破这些规则。

在本书给出的示例程序中有着许多不同的地方，我们将在本章对程序中各种不同的编写方式进行解释。

C++ 是一种特殊的编程语言，因为在许多不同层次的抽象上，都可以用 C++ 来编程。例如，我们可以通过 C++ 对计算机硬件设备进行控制，这通常是需要用汇编语言才能达到的低层次抽象；此外，C++ 支持类、继承和多态，而这又是一种层次非常高的抽象。因此，在所有的面向对象语言中，从程序员所能控制的抽象层次上来看，C++ 是一种非常优秀的语言。

C++ 对 C 进行了许多扩展，其中大多数扩展都是和大规模编程相关的。C++ 中引入了成员函数、访问控制、重载、继承和多态等特性，这些特性都是用来描述程序中不同组件之间或者一组程序之间的关系，而并不是用来描述如何在单个组件中进行编码。因此，在本书的 C++ 编程风格中，我们讨论的大部分问题都是关于如何描述组件之间的关系。事实上，如果要在编程风格和软件设计之间划分出明显的界限，那么将是一件很困难的工作。这是很自然的，因为与像 C 这样的语言相比，用 C++ 来编程能够更明显地反映出软件的设计思想。我们通常是通过编程语言来表达具体的软件设计思想，而 C++ 程序总是能够告诉我们大量重要的设计细节。

本书假定读者对 C++ 语言已经有了不错的掌握，这样我们就只需要解释少量的细节问题。读者可以通过自己喜欢的入门书籍来学习那些尚不熟悉的语法和语义。只有当遇到一些深奥的技术细节时，我们才会进行明确解释。对于某些 C++ 的细节问题，我们建议读者去参考 Eillis 和 Stroudtrup 的著作，或者翻阅 The Annotated C++ Refrence Manual，Addisio-Wesley，1990。作为参考手册来说，我们很难学完 Ellis 和 Stroustrup 的著作中所有的内容，因此应该学会针对特定的问题有选择性地进行阅读。

词法风格

在 C++ 中可以使用许多不同的词法风格，例如缩进约定、在什么地方使用空格等。在本书中，我并没有指出哪一种词法风格是更好的。大多数的程序都是遵循着最初的词法风格。例如，在一些程序中将指向字符的指针声明为 char* p，而在有些程序中使用的是 char *p。有时候用 0 来表示空指针，而有时候则用 NULL 来表示。不过，在每个示例程序中，我们都将只用一种词法规则，并且在修改代码时也遵循这个规则。

我们并没有指出哪一种词法风格是更好的，理由有两个。首先，虽然在程序的可读性上，词法风格很重要，但与大规模编程的问题相比，这种重要性就显得苍白无力。例如，与“程序中是需要一个类还是两个类，或者这些类是否通过继承来进行关联”相比，我们在类的开始部分到底是声明公有成员函数还是私有成员函数就显得不那么重要。事实上，从某种意义上说，词法风格也可以是无足轻重的，我们可以通过一些自动工具——其中包括多种排版技术——将任意程序从一种词法风格转换到另一种词法风格。因此，在本书中将不会对词法风格进行讨论。

其次，许多关于词法风格的问题都是主观性的。例如，对某些程序员来说，多余的括号可以带来帮助，而对其他的程序员来说，多余的括号则妨碍了程序的编写。有些程序员喜欢看到返回表达式被写为 return (x)，而有些程序员则更喜欢使用 return x。对于不同的程序员来说，所有这些做法都是基于同一个理由："这样写更易于阅读"。程序员，无论是个人还是处于某个团队，必须在词法风格的问题上作出他们自己的选择。换句话说，即使我在词法风格上有我自己的观点，但其他程序员也可以不考虑它们。或许，唯一一条客观的原则就是：在软件项目的整个开发过程都应该只采用一种词法风格，并且应该自始至终都遵循这种词法风格。

内联函数与 const 声明

在示例程序中，有些成员函数被声明为内联函数，虽然这样的声明看上去对程序整体性能的提高并没有什么帮助。一些程序员只是例行公事地将代码量较小的成员函数声明为内联函数：将这些成员函数的定义放在类的声明中，这样可以减少一些源代码的大小。然而，这通常是一种误解。在编译的时候，内联函数将在被调用的地方进行展开，这通常比相应的函数调用要占据更多的代码空间，尤其当我们在内联函数中又调用了其他的内联函数时。由于内联函数展开所增加的代码量有可能降低程序的执行速度，因为代码量的增加将可能妨碍有效的代码缓冲操作。只有当内联函数确实能够带来程序性能的提升时，才应该被用于代码中。然而，对于那些无法判断是否会带来性能提升的内联成员函数，我们还是应该将它们保留在最初的代码中，并且不予以分析。通常，在每个示例程序中都会有一个关于正确性或者一致性的问题需要讨论，这个问题远比任何关于程序性能的问题要重要。因此，我们将内联函数的相关问题交由读者来分析。

在本书的示例程序中，对于 const 修饰符在类型和成员函数上的用法也是不同的。如果在程序中小心地使用 const，那么就能在编译期检测出一些错误。不过，由于在使用 const 的时候存在着不同的风格，而且在一个已经完成的程序中加入 const 是非常困难的，因此，我们并没有将 const 修饰符增加到程序中去。const 的出现或缺少并不会影响程序的整体结构，而整体结构才是我们关心的主要问题。

虚函数

虚函数的动态绑定是 C++ 中一个强大并且重要的功能。本书的大部分内容都是讨论在什么情况下应该将虚函数从程序中删除，而不是讨论在什么情况下将虚函数增加到程序中。我们将首先给出一些使用了虚函数的 C++ 程序，然后对这些程序进行观察和分析，最后将重点放在如何消除虚函数上。使用虚函数的示例程序是非常多的，然而能够通过虚函数来改进性能的程序却是非常少的。因此，概括起来就是：在示例程序中，我们将重点说明如何避免虚函数，而不是说明如何使用虚函数。

注释

在本书中，大多数程序的注释都很少。事实上，从软件产品的设计标准来看，这些注释是不够的。如果有更多的注释，那么读者在阅读程序时可能会更快，但这却违背了我们研究代码的

初衷。如果注释不是很多，那么读者就必须直接从代码中获得更多的信息，从而把更多的注意力放在程序的细节上。通过对程序细节的分析来推断程序的设计思想，这对于理解大多数的示例程序来说都是非常重要的，这也是一个值得我们去努力培养的技术。

标准 I/O 与流

在本书的一些程序中，我们使用了标准的 C 语言 I/O 库（例如，printf）对字符进行格式化输出，而在其他程序中则使用了流类库（例如，cout）。在大多数情况下，无论程序使用哪种方式，都可以编写出简洁的代码。

参数化类型和异常

在编写本书的时候，在已发布的 C++ 编译器中都能支持参数化类型（模板），而异常处理则还不支持[1]。对于这些 C++ 特性来说，我们应该持谨慎的态度。我们应该等到大多数程序员都接受了这些特性后，再去观察和分析使用这些特性的程序，而不是凭主观想象去推测这些特性所可能带来的混乱。只有当这些特性在程序中出现以后，才可以给出一些有意义的指导原则，用来告诉程序员在使用这些特性进行编程时如何去避开常见的陷阱。

练习

在大多数章节的后面都给出了练习，其中一些练习是进一步阅读和分析程序。在一些练习的代码中，所存在的缺陷与该章示例程序中的缺陷是相同的。在其他的练习中，则给出了一些写得不错的程序。然而，这些练习并非要求读者按照所给出的方式来编写代码。读者应该通过实践来找到适合自己的编程方式，而不是盲目遵从书中给出的方式。

程序清单

在本书中，所有示例程序都是从实际的源文件中节选出来的，这些源文件至少能够在一种 C++ 平台上进行编译和运行。其中大多数程序都是在 Cfront 2.0 下运行的，还有一些是在 G++ 和 Borland C++ 3.0 下运行的。

参 考 文 献

在编写本书的时候，对 C++ 的定义描述参考了 Ellis 和 Stroustrup 的著作 [1]。

Ellis,M.A.,and Stroustrup,B.1990.*The Annotated C++ Reference Manual.* Reading,MA:Addison-Wesley.

[1] 译者注：这是因为作者编写本书的时候较早，这些特性在目前所有主流的 C++ 编译器上都已完全支持。

目录