PEARSON

C和C++实务精选
品味岁月积淀，读享技术菁华

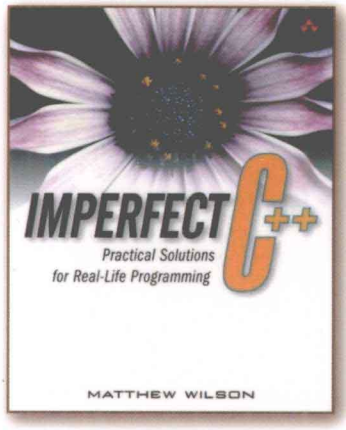
Imperfect C++

中文版

[美]Matthew Wilson 著　　荣耀 刘未鹏 译

Imperfect C++ Practical Solutions for Real-Life Programming

- 深刻认识C++不足，体会C++新标准
- 著译双馨，难以重现的译者组合



人民邮电出版社
POSTS & TELECOM PRESS

Imperfect C++ 中文版

Imperfect C++ Practical Solutions for Real-Life Programming

[美]Matthew Wilson 著
荣耀 刘未鹏 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Imperfect C++中文版 / (美) 威尔逊 (Wilson, M.) 著 ; 荣耀, 刘未鹏译. — 北京 : 人民邮电出版社, 2012.7
ISBN 978-7-115-27797-8

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

中国版本图书馆CIP数据核字(2012)第045630号

版 权 声 明

Imperfect C++中文版

◆ 著 [美] Matthew Wilson
译 荣 耀 刘未鹏
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· 39 5
字数· 875 千字 2012 年 7 月第 1 版
印数：1－3 000 册 2012 年 7 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-2438 号

ISBN 978-7-115-27797-8

定价：99 00 元（附光盘）

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

即便是 C++阵营里最忠实的信徒，也不得不承认：C++语言并不完美。实际上，世界上也没有完美的编程语言。

如何克服 C++类型系统的不足？在 C++中，如何利用约束、契约和断言来实施软件设计？如何处理被 C++标准所忽略的动态库、静态对象以及线程等有关的问题？隐式转换有何替代方案？本书将为你一一解答这些问题。针对 C++的每一个不完美之处，本书都具体地分析原因，并探讨实用的解决方案。书中也不乏许多作者创新的、你从未听说过或使用的技术，但这些确实能帮助你成为 C++方面的专家。

本书适合有一定经验的 C++程序员和项目经理阅读，也适合对 C++编程的一些专门或高级话题感兴趣的读者参考。

译序一

千万不要被书名所误导！这是一本拥抱（而非诋毁）C++的著作。它有着独特的定位：为现实世界中的程序员提供切合实际的解决方案，以解决C++语言自身的各种“不完美”。

世界上没有完美的编程语言。在本书中，Matthew Wilson不但为我们指出C++中诸多不完美之处，还提供了经过实践检验的应对技术和技巧，便于我们利用“不完美的C++”编写出近乎完美的代码——强健、高效、灵活、可移植、优雅的代码，而这些代码在声称为“完美的语言”中往往更难实现。

本书对给出的每一个“不完美”都进行了细致的探讨：为什么说它是一个“不完美”？对其修复的指导思想是什么？有时候只是告诫你避免做些什么，给出一些约束和建议，更多的时候则为你提供现实的解决方案，这些方案往往离不开对现代模板编程技术的使用。

书中包含有许多你未曾听过或用过的技术，有些属于作者的创新，有些则是对现有技术的精化，二者均被提升到“范式”的高度。例如：应用程序二进制接口（ABI）、垫片（Shim）、饰面（Veneer）、螺栓（Bolt-in）、区间（range）、属性（property）等。不少主题难度大，此前为其他C++专家所忽略。要探讨它们除了需要勇气外，第一手经验更是不可或缺。作为STLSoft库的主创者，Matthew在举例时，对Windows API、MFC、ATL、COM以及UNIX等都是信手拈来。

除了丰富的实践、扎实的理论以及缜密的逻辑外，Matthew的文笔流畅，语言幽默，说理直接，字里行间流露出过人的自信，使得本书极具阅读趣味。

本书具有一定的阅读门槛，目标读者为中、高级职业C++程序员。书中展示的代码示例、编程技术往往在几款甚至十几款编译器上进行验证，辅以表格对其各色特性加以比较，并针对不同编译器所表现出的差异性而给出高效、可移植的解决方案——就像很多现实世界中的C++程序员应该做（而没做到）的那样。如果你正在寻找一本真材实料的“C++实战”参考书，本书不会让你失望。

本书中文版由我和刘未鹏先生合译。未鹏思维敏捷，技术、文笔俱佳，我很高兴与他合作。

感谢陈冀康编辑给予的理解和支持。感谢朱艳的照料和热爱。荣珅则常常用他的小拳头乱砸书房的门，并大声地叫“爸爸”，这种干扰让我获得了必不可少的休息时间。

已有的经典名著使得C++新书问世难度加大，后来者若无过人之处就很难引起C++社群的注意，*Imperfect C++*、*C++ Common Knowledge* 以及 *C++ Template Metaprogramming* 等佳作一经问世便得到广泛的关注。作为译者（之一），我祝愿它们能够带给各位久违的快乐！

荣耀

2005年8月

于南京师范大学

www.royaloo.com

译序二

刀有很多种，有单刀，双刀，朴刀，戒刀，锯齿刀，砍山刀，鬼头刀，雁翎刀，五凤朝阳刀，鱼鳞紫金刀。

—— 古龙《飞刀，又见飞刀》

这里我们要说的刀，是瑞士军刀，瑞士军刀其实严格来说并不能算是一种刀，其功能的繁杂和精细已然超过了刀的范畴。它包含的工具一般有主刀、小刀、剪刀、开瓶器、木锯、小改锥、拔木塞钻、牙签、小镊子等，而在一些工具上还设计了多种功用，如开瓶器上，就具有开瓶、平口改锥、电线剥皮槽3种功用。随着时代的发展，一些新兴的电子技术也被引入瑞士军刀中，如内藏激光笔、电筒等。

瑞士军刀是军人在野外生存的必备工具，其小体积浓缩众多实用功能的精心设计能够将一把刀的容限发挥到最大，丝毫不逊于《第一滴血》中蓝博带在身上的那把锐利的寒光闪闪的钢刀。

那么现在你拿在手里的这本书就是一把瑞士军刀！

这是一本非常特别的C++图书，在市面上已经存在的大量经典C++书籍当中，这本书的着眼点和写作风格使它显得那么特立独行和标新立异，甚至有点另类。书中几乎巨细靡遗地涵盖了C++中大大小小的不完美之处，并以一系列成功案例证明C++的确不完美同时也提供了迂回之道、解决之道，再加上其用本主义的立场，正如同一把实用的瑞士军刀，功能繁杂而面面俱到，实用之至。同其他C++著作不一样，本书虽然尊重标准，但同时又超越标准，当标准不能满足需求或成为拦路石的时候，需求才是第一位的，于是有了作者所谓的“不完美主义的实践者”以及“不完美工具箱”之说。此外，作者的所谓“苦行僧式编程”哲学在我看来也是极其实用的一种编码方式！

我们以前看到的绝大部分C++书籍可说是统统走的“阳关大道”，然而Matthew这本书却偏要走他的“独木小桥”，蹊径虽小，然则别有一番风味和景观。我们意识到原来C++中也存在着如此多大大小小的不完美之处，就像宫崎俊电影中的那些打满补丁的海盗飞机一样。Bjarne本就说过，C++是为“用本”而设计的，诚然！而本书最大的趣味就在于它并不去一味抱怨这些缺点，而是积极地采取其他替代方案来达到同样的目的，并借此展现出C++自由强大的一面！

作者Matthew常用“survive”一词来描述在编码的现实世界中的境况，作为STLSoft库的主

要编写者，他十几年来积累的经验在书中充盈四溢，很多我们平常看不到的方面都会被他挑出来，甚至连我这个译者都觉得有点“啰嗦”。不过，对于喜欢他这种“唠叨”讲法的人，他那种辨证的严密论证法倒是能令你获益颇多。另外，书中随处可见具有作者个人特色的幽默，在大量平淡无奇的技术书籍当中可算是一个亮点。

本书的一个小缺憾就是它不适合初学者，某些地方甚至对于中级读者来说都有一定的难度，作者自己经验非常丰富，因此有些地方就不加解释地一带而过，为此译者适当添加了一些译注，以便读者理解和阅读。

最后，感谢荣耀先生在本书初译的过程中一直给予的支持和信任，并容忍我总是延期交付各章译稿。荣耀先生对技术的精益求精和一丝不苟也令我在翻译的过程中获益良多。

最大的感激要归于我的父母和我的爷爷，感谢他们一直以来对我的追求的支持和鼓励，没有他们我无法想象能够完成这项工作。

希望这本令我在翻译过程中获益匪浅的书也能够给你带来美妙而独一无二的阅读享受，Let's dig in!

刘未鹏

2005 年 8 月

前　言

或许我不像喜欢我的孩子们那样喜爱 C++，甚至或许我对 C++的喜爱都比不上我对骑自行车在坡度为 32°、光滑度为 10%的柏油路上爬坡的热衷，[1]尽管有时这些喜爱之情的确十分接近。我庆幸我有这样的人生，让我得以将生命中的部分时间用来实践或阐释 Frederick P.Brooks 的名言："尽量发挥想象力进行创造"。我更要感激的是我能够跟这门如此强大、危险却又诱人的语言相伴。

这些话听起来似乎蛮华丽动听，但你可能是因为看到本书的书名才买下它的，以为本书是一本抨击 C++的图书。你可能是 Java 或 C 或其他主流语言的热衷者，因而买这本书可能是想从中找到支持你远离 C++的理由。倘若果真如此，你会失望的，因为本书并不是举办一切 C++批判大会。不过先别急着离开，因为你或许能够从中找到令你开始接触 C++的理由。

你将从中学到什么

我写这本书的目的在于给予开发同行们一些能量。它以虽批评但具建设性的眼光来看待 C++及其不完美之处，并给出实际的措施以避免或改善这种不完美。我希望你在读完本书后能够对下面这些问题有一个更好的把握：

- 如何克服 C++类型系统中的某些不足。
- 模板编程在提高代码灵活性和健壮性上的强大能力。
- 如何在（当前标准尚不予置理的）未定义行为的险恶丛林中生存下来。后者包括动态库、静态对象以及线程等。
- 隐式转换的代价、它所带来的问题，及其替代方案，即基于显式转换的、高效且易于控制的泛用性编程。
- 如何编写能够与（或更容易令它们与）其他编译器、库、线程模型等兼容的软件。
- 编译器"在幕后"都做了些什么？你对编译器可以施加什么样的影响。
- 数组和指针之间微妙而棘手的互操作性，以及用于防止它们的行为变得彼此相似的技术。

---

[1] 自行车爱好者会明白我为什么这样说。

- C++对 RAII（Resource Acquisition Is Initialization，资源获取即初始化）机制的支持，以及该机制可以被运用到的各种问题领域。
- 如何通过最大限度地榨取编译器的侦错能力来节省你自己所需花费的工夫。

有了上面这些“装备”，你编写的代码将更有效率、更具可维护性、更健壮，也更具灵活性。

我的意图是，即便是经验丰富的 C++实践者也能从本书中发现新思想和新技术，从而引发他们的灵感并提高其现有的实战能力。经验较少的程序员则能领会其中包含的有关原则并将相关技术应用到自己的工作中，在增长见识的同时弥补自己对一些技术细节理解上的疏漏。

我并不指望你们所有人都完全同意我所说的一切，但我期望哪怕是最有争议的内容也能激发你去真正弄明白自己对这门强大语言的使用问题。

我假设你们已经知道

除非有人想写一本很厚的书，否则他必须假设许多知识是读者已经知道的。当然，规定大家必须首先读完某些书籍的做法过于粗鲁，但我假设你们的知识和经验能使你们轻松地理解 Scott Meyer 的 *Effective C++*系列和 Herb Sutter 的 *Exceptional C++*系列中讲述的绝大部分概念。我还假设你们拥有一本 C++“圣经”，即 Bjarne Stroustrup 的 *The C++ Programming Language*。我并不指望你们仔细阅读该书的每一页（我都还没有做到这一点），但你们应当将这本书作为这门语言的终极参考，因为它的确是字字珠玑。

本书中包含有相当多的模板代码（哪一本现代 C++书籍不是如此呢），但我并没有假设你是名大师级的人物，[1]或掌握了高阶元编程知识。话虽如此，我还是建议你们最好熟悉如何使用模板，比如组成 C++标准库中最流行的部分的那些内容。我努力将对模板的使用控制在一个合理的水平，但我们必须得承认，正是对模板的支持使得 C++具有“自我修复”的能力，而这也是本书得以诞生的主要原因。

由于灵活性和实用性对我而言非常重要，因此书中的代码并非只能在少数最新编译器上编译；实际上，书中几乎所有代码都可以在任意一款“还算可以”的现代编译器（见附录 A）上编译。当然，有一些很好的编译器是可以免费获得的，并且你也可以相信你的编译器能够编译这些代码。

只要有可能，我就会尽量避免涉及特定的操作环境、库和技术。然而我也略微谈到了一些，所以，了解以下内容将会有所帮助（尽管并非必不可少）：COM 和 CORBA、动态库（UNIX 和 Win32 的）、STL、线程（POSIX 和 Win32 的）、UNIX 以及 Win32。参考书目中包含有许多这些方面以及其他方面内容的好书。此外，熟悉不止一种机器架构也是有帮助的，当然这同样并非不可或缺。

由于 C 目前仍是语言间交互以及操作系统开发的通用语言，因此它继续作为一门极其重要的语言存在着。尽管本书是关于 C++的，但在某些领域，C 跟 C++之间的共性会变得很重要，我认

[1] “参考书目”中列出的若干书籍，可在你成为大师级人物的漫漫征途中助上一臂之力，如果你愿意付出努力的话。

为在这些领域选择兼顾 C/C++是合理的。事实上，正如我们将在本书的第二部分中看到的，我们有时需要求助于 C 来支持 C++的一些高级用法。

此外，我还作了一个重要的假定。我假定你们也认为工作的质量是非常重要的，并且有动力去寻找达到这一目标的新途径。本书不敢妄称是这些所谓新途径的惟一源泉。确切地说，它提供了一种实践性的、有时甚至是异端的视角来看待我们在 C++中遇到的问题。如果情况够好的话，本书或许也能成为你的精华书库中的成员之一。最终的责任落在你自己的肩上，剩下来要做的就是去寻找最好的工具来支持你的工作。

本书的组织方式

本书的主要内容分为 6 个部分，每一部分由一个绪论和 5～7 章组成，每一章又可以被进一步划分为若干节。

既然本书取名为“Imperfect C++”，那么我就会在书中尽量突显出实际的不完美之处，这就是你们在本书的通篇都会发现一些所谓的“不完美（Imperfection）”的原因。在书的前几部分，这些不完美之处出现得比较密集，这说明它们自身和它们的解决之道尚且是比较简单的。每个小节都对应着语言的某个特定的特性，并且通常会介绍它的一个不完美之处。只要有可能，我就会提供一些具体的技巧或技术来解决这些问题，或至少为开发者提供控制问题的方法。随着本书内容逐步展开，这些不完美之处将会变得不再像前面那样散碎，显得更为重大，从而伴随有更大篇幅、更加详细的讨论。

本书并没有采用时髦的“自助餐”式的写法，也不存在一条必须从头读到尾的贯穿全书的主线。当然，大部分后续章节的内容是根据前面章节的内容进行描述的，有时甚至建立在之前的章节之上，所以除非你故意作对，否则最好还是按顺序阅读。然而，一旦你读完一遍后，回头再来参考其中的某些部分时，你就可以根据需要跳至任意一处而不再需要通读所有内容了。在每一章中，各节一般是按内容顺序排列的，所以我建议你也应该按顺序来阅读每一节。

在难度方面，第一部分到第四部分显然经历了一个从易到难的过程，从相当简单明了直到有相当高的要求。[1]尽管第五部分和第六部分需要依赖第三部分和第四部分的一些内容，但它们相对来说不再那么具有挑战性了。你可以优哉游哉地一直读到附录。

主要内容介绍完之后是 4 个简短的附录。附录 A 介绍在探讨本书各项议题时使用的编译器和库的详细信息。附录 B 向你展示一个年轻的 C++工程师在刚踏入这一充满陷阱的领域时所犯下的一些令人目瞪口呆的错误。附录 C 介绍 Arturius 项目，这是一个免费的源码开放的编译器多路分发器，你也可以在随书光盘中找到它。最后，附录 D 介绍随书光盘的内容。

我的编码风格十分一致，甚至可以说是严格，你也可以说它过于“学究气”，我以前的同事和使用我库的人们早就这样说过。但我之所以采取这种风格是因为这么一来代码中的所有东西都

[1] 第三部分和第四部分的某些内容直到现在还让我感到头疼呢！

有其明确的位置，人家不至于会问“某某东西到哪里去了”这种问题，同时这也意味着当我几年后重新面对这些代码时，我还可以轻松搞定它。当然这么做也有缺点：我需要一个 21 英寸的显示器和一个工业用激光打印机。

为了尽量减少我的编码风格对阅读本书的影响，我在书中出现的代码示例中使用了一些自由风格。你将会在示例中看到很多省略号（...），它们一般表示前面的有关例子中已经包含了该段代码，或者表示这些省却的代码是我们司空见惯的样板式代码（例如，禁止客户代码访问某些方法，见 2.2 节）。只有编码风格中对可靠性有着显著影响的方面才会被加以讨论（见第 17 章）。[1]

参考资料

我在阅读其他 C++书籍时感到不满的一件事是：作者指出事实的同时从不提及标准的相关部分。所以我在写作本书的过程中，除了会不断引用一些相关的书籍和文章外，还会在描述 C++语言行为的同时提供其在 C++（C++98）或 C（C99）标准中的相关参考。

补充材料

光盘

随书光盘中包含有一些库、编译器（包括书中描述的大量编码技术）、测试程序、工具，以及其他有用的软件，另外还有许多摘自各种出版物的相关文章。关于光盘的详细内容，请参考附录 D。

网上资源

你也可以通过如下网址来获取补充资源：http://imperfectcplusplus.com。[2]

致谢

在几乎所有你看到或将要看到的书籍当中，你都会发现其中有一页或几页充满着对家人和朋友热情洋溢的致谢之辞，我可以向你保证这些言辞都是发自内心的真情流露。一本书要想得以完成，其背后必定隐藏着若干人的支持。

首先感谢我的母亲和 Suzanne，感谢她们长期以来的容忍、支持以及养育之恩，直到当年的唧唧喳喳的小布谷鸟到了早就成熟的年龄（27 岁），才离开温暖的巢，飞向一个不同的世界。感谢母亲和 Robert 在一段艰苦但最终有所收获的岁月里一直帮助小布谷鸟和他的家庭。尤其感谢

[1] 如果你非得完整见识一下我的编码风格的话，你可以到随书光盘中的库里找到足够多的样例。
[2] 这个站点上也有一个勘误表页面，当然，倒不是说这里一定会有错误。

Robert 在这段时间里的许多重要时刻帮我保持精神放松。

谢谢 Piker 填补了这个大家庭的空缺，并且不遗余力地照看孩子，鼓励我们，并提供免费的午餐。同样感谢 Dazzle，他总是告诉我他对我极有信心，并难能可贵地放弃那些诱人的 DBA 大师活动，坚定地帮我做那些沉闷的审稿工作；他在看其他 Perl 和 Python 脚本的时候可决不会是这个心态！也要感谢 Besso 一直以来对我的计划的浓厚兴趣、过分的自豪以及鼓舞人心的观点。谢谢 Al 和 Cynth（亲家）提供许多免费的饭菜和美味的巧克力（哦，我差点忘了我那辆自行车……）。

衷心感谢 808 State、Aim、Barry While、Billy Bragg、De La Soul、Fatboy Slim、George Michael、Level 42、Rush、Seal、Stevie Wonder 以及 The Brand New Heavies，没有他们我不可能从这个一年半的幻想中挣扎过来。

最重要的感谢要给予我的美丽爱妻 Sarah，感谢她抑制住合乎情理的担心和疑虑，而表现出完全的支持和信任。她是我心目中真正的明星！

感谢一些出色的人，他们对我的教育和事业的影响是深远的。感谢 Bob Cryan 教授，感谢他能够赏识一名天赋不错的学生，并在后来 3 年的研究生在读期间宽容他的逃课（去骑自行车）。

还要感谢 Richard McCormack 让我保持在概念上的优雅之外还看到了代码的高效之美。这段时间我有时会被别人责备说过于看重效率了，我就说你要怪就怪 Richard 去吧。另外，感谢 Graham Jones（绰号“Dukey”）教我设置 vi，在那疯狂的 6 个月的时间里和我保持深厚的友谊以及提供开心的玩笑。这些东西，无可替代！

同样还要感谢 Leigh 和 Scott Perry，感谢他们向我介绍“螺栓”概念以及其他优秀的技术。

特别感谢 Andy Thurling。在我怀揣博士文凭和名不副实的软件工程技能等级证书去找工作时，Andy 对我的潜能表现出充分的信心。[1] Andy 还教给我或许在这个奇妙而令人畏惧的职业中最最重要的一课：“我们只是在尽量充分利用手头拥有的信息而已（Skegging it Out）”。[2] Chuck Allison 则以更为亲切的形式来表述这个意思，那是一句来自古老的印第安人部落中的箴言：“向一个不断学习的人学知识就好比是在饮一条生生不息的河流”。

任何书籍的成功都离不开出版社、评审者以及其他给出指导和建议的人们的帮助。感谢我的编辑 Peter Gordon 给予鼓励并包容一个热情而任性的作者在写他的第一本书的过程中情绪起落。同样感谢 Peter 的得力助手 Bernard Gaffney，他很好地控制了整个计划进程，并耐心地答复我每天发去的几封电子邮件。还要感谢 Addison Wesley 出版社的产品和市场部门的其他职员，包括 Amy Fleischer、Chanda Leary-Coutu、Heather Mullane、Jacquelyn Doucette、Jennifer Andrews、Kim Boedigheimer 以及 Kristy Hart。衷心感谢（并致歉）我的项目经理 Jessica Balch 不厌其烦地帮助我纠正书中糟糕的句法、蹩脚的幽默以及英式英语的拼写（例如有很多“ize”被写成了“ise”）。还要特别感谢 Debbie Lafferty，2002 年的某一个晚上我在睡梦中冒出了“Imperfect C++”这个念头，是 Debbie Lafferty 鼓励我付诸实现的。

感谢忠诚的评审者们，他们是 Chuck Allison、Dave Brooks、Darren Lynch、Duane Yates、Eugene

[1] 当时我连实模式与保护模式的区别都弄不清！

[2] “Skegging it out”是北约克郡的一种用语，意思是“充分利用你当时拥有的信息”。

Gershnik、Gary Pennington、George Frasier、Greg Peet、John Torjo、Scott Patterson 以及 Walter Bright。没有他们我肯定会到处磕磕碰碰，踉踉跄跄。他们中的一些人让我保持心情愉快，有些人则使我质疑这个写作计划是否明智，但无论如何他们的反馈信息都极大地有助于我改善最终的成果。我们生活在一个奇妙的世界中，来自各个不同国家的人们可以建立起友谊，虽然其中大部分人我未曾谋面，却能够给予我如此之多的帮助，这真是奇妙！

同样要感谢 Addison Wesley 的审阅者们，包括 Dan Saks、JC van Winkel、Jay Roy、Ron McCarty、Justin Shaw、Nevin Liber 以及 Steve Clamage。他们的反馈信息对于本书的篇幅能够降到 1000 页以下起到了至关重要的作用，而且帮助我避免了一些无聊的闲话和粗心大意的错误。作为一名作者，我早已领略了写作和审稿过程的艰辛，我知道一次彻底的审阅需要付出多少精力，真的非常感谢他们投入的时间和精力。

感谢 Peter Dimov 允许我引用他的名言（在第 26 章），同时还对第五部分各章给出了极好的建议。感谢 Kevlin Henney 对第 19 章和智能强制的一些有趣的讨论所给予的关注。感谢 Joe Goodman 仔细审阅有关 C++ ABI 的那一部分（第 7、8 两章），使我最终能够呈上一份像样的讨论。感谢 Thorsten Ottosen 在第 1 章有关契约式设计部分提供的类似的帮助。

特别感谢 Chuck Allison、Herb Sutter、Joe Casad、John Dorsey 以及 Jon Erickson，他们在过去的几年中在各方面都给予我极大的支持和鼓励。

感谢 Bjarne Stroustrup 的鼓励，他还不时为我补上一点历史常识。哦，首先还是要感谢他发明了这门神奇的语言！

感谢 Walter Bright 在本书的写作过程中持续不断地改进他的优秀的 Digital Mars C/C++编译器，在世界上最缺乏耐心的家伙持续的质问面前保持责任心，而且还友好地将我引领入 D 语言开发世界，后者也给我这本书的写作带来了不少灵感。同样感谢 Greg Comeau，尽管他用不着对他的 Comeau 编译器做那么多的改进：Comeau 是目前业界最遵从标准的 C++编译器！他们除了是优秀而又反应迅速的编译器供应者之外，还在众多问题上持续不断地给予鼓励、建议以及其他信息。

感谢 CMP 出版公司允许我在书中使用我原先在他们刊物上发表的文章中的一些内容，并允许我将原先的几篇文章放在随书光盘中（见附录 D）。

感谢 Borland、CodePlay、Digital Mars、Intel、Metrowerks 以及 Microsoft 公司为我的研究、写作以及编写开源库提供编译器。

特别要感谢 Digital Mars、Intel 和 Watcom 公司授权我将他们的编译器放在随书光盘中（见附录 D）。另外，Greg Peet 在光盘的设计及内容方面的巨大帮助值得大大嘉赏。

谢谢 *C/C++ User's Journal*、*Dr. Dobb's Journal*、*BYTE* 以及 *Windows Developer Network* 的读者，他们为我的文章和专栏友好地提供了反馈信息和支持。

同样感谢整个 C++社群，以及那些关注各种与 C++有关的新闻组（见附录 A）的好心人。他们是 Attila Feher、Carl Young、Daniel Spangenberg、Eelis van der Weegen、Gabriel Dos Reis、Igor Tandetnik、John Potter、Massimiliano Alberti、Michal Necasek、Richard Smith、Ron Crane、Steven Keuchel、Thomas Richter 以及“tom_usenet”，而且或许还有好些人没有列出。尤其感谢 Ilya Minkov，

是他要求我为 STLSoft 库增加属性（Properties）实现的，而我自己以前可没有产生过这个念头。如果不是因为这个建议，我可能永远也不会实现这项我钟爱的技术（见第 35 章）。

最后，还要感谢 STLSoft 库的所有用户，没有他们的信息反馈，这个库中的许多特性就不会存在，而且本书中的某些部分也会变得难弄得多。

Matthew Wilson

不完美主义实践者的哲学

在本书中，C++语言技术与良好的实践方式占有同等重要的地位。本书并非仅仅讨论在某个特定场合下什么方案才是有效的或技术上正确的，更重要的还是要看最终哪种做法才是更安全的或者更切合实际的。本书要传达的意思有4个方面：

原则1——C++是卓越的，但并不完美。

原则2——穿上“苦行衣”。

原则3——让编译器成为你的仆从。

原则4——永不言弃，总会有解决方案的。

这四项原则构成了我所谓的“不完美主义实践者”的哲学。

C++并不完美

多年以前，一位为她最小的儿子过分骄傲的心理感到不安的母亲曾这样教导说：“如果你打算将一些好的东西告诉别人，那么你最好也准备承认其中那些糟糕的成分。”谢谢你，母亲！

C++是一门杰出的语言。它支持高阶概念，包括基于接口的设计、泛型、多态、自描述的软件组件以及元编程（meta-programming）等。此外，凭借对低阶特性的支持，它在提供对计算机的精细控制方面，包括位操作、指针以及联合（union）等，也比大多数语言更有能耐。借助于这些范围宽广的能力，外加保持对高效性的根本性支持，C++可以说是当今最杰出的通用编程语言。[1] 不过话说回来，C++并非完美无瑕，实际上远没到完美的程度，因而有了本书的名字——“Imperfect C++中文版”。

由于一些很好的原因（有些是历史原因，也有些是当前的原因），C++不仅是一个折衷[Stro1994]的产物，而且还是一些互不相干、有时甚至是互不兼容的概念的混合体，因而其中必然存在着一些缺陷。有些缺陷只是鸡毛蒜皮，但有些就不是那么无关紧要了。许多缺陷都是从它们的“祖辈”那里承袭而来的。其他则是由于语言将效率放在高优先级（幸好如此）才导致的。有些则可能是

[1] 请注意，我并不是说C++在所有特定问题领域都是最佳语言。例如，我绝不会建议你使用C++去编写“专家系统”，Prolog才是这方面的合适人选。而在系统脚本方面Python和Ruby则当仁不让。此外，我们还得承认Java在企业级电子商务系统开发中确有过人之处。

任何语言都无法摆脱的根本限制。正是由于如今的语言变得愈来愈复杂，也愈来愈多种多样，因而才出现了一些极有趣的问题，这些是任何人都始料未及的。

本书直面这种复杂的形势，坚信总能够克服复杂性，坚信控制权最终还是掌握在那些见多识广、经验丰富的计算机专家手中。我的目标是缓解那些使用C++的软件开发者日常经受的不知所措以及无法作出决断的痛苦之情。

本书致力于解决的并不是软件开发者因经验不足或知识不够而遇到的问题，而是这个职业中的所有成员，从初学者甚至到最有才干和经验的那些人，所共同遭遇的问题。这些问题中部分源于语言自身固有的不完美性，部分源于人们对语言所支持的一些概念的常见误用。无论如何，它们给我们所有人带来了麻烦。

本书并不仅仅是对语言中的不完美之处进行一些简单的论述并附带一些“别这么做”列表，你可以找到很多着眼于这方面的C++书籍。和它们不同，本书的重点在于如何为我所指出的缺陷（中的大部分）提供解决方案，并借此使得这门语言变得不再像它本来那么“不完美”。本书重在赋予开发者能量，讨论他们赖以谋生的手段——C++——中潜在的问题领域，给出了一些重要的相关信息，并为开发者提供了一些建议以及经过实践检验的技巧和技术，以帮助他们避免或应付这些问题。

苦行僧式编程

在我们阅读过的很多教科书中，即便是非常好的，也只是告诉我们C++能够提供的解决问题的手段，前提还得是你必须将C++中的有关特性有效地利用起来。然而，这些书常常又会在后面接着说道：“这样做其实没有实质性的意义”或者“严格来说那样做就稍微有点过头了”。不止一个以前的同事曾把我拖进类似的激烈争论之中。通常人们的理由可以归结为：“我是一名有经验的程序员，我并不会犯XYZ阻止我犯的那些错误，干嘛要为之烦神呢？”

唉！

这个论点简直不堪一驳。我也是一名有经验的程序员，但我每天至少会犯一个低级错误；假如不是养成了严格的习惯的话，则会是一天10个！他们的这种态度其实就是在假定他们的代码永远都不会被没有经验的程序员看到。此外，他们的说法要得以成立，等于是在说代码作者永远也不会学习或者改变观念、习惯以及方法论。最后一点，到底怎样才算是“有经验的程序员”呢？[1]

上面提到的这类人不喜欢引用、常量成员、访问控制、explicit、具体类（concrete classes）、封装、不变式，他们甚至在编码时根本就无视可移植性或可维护性。但他们就是喜欢重载、重写（override）、隐式转换、C风格强制（C-style casts），并到处使用int。他们还喜欢全局变量、混合式typedef、dynamic_cast、RTTI、专有的编译器扩展以及友元。他们在编码风格上总是不一致，

[1] 如今这个问题似乎变得没有定论，因为你看到每个人的履历表上的自我评价全都是10分。