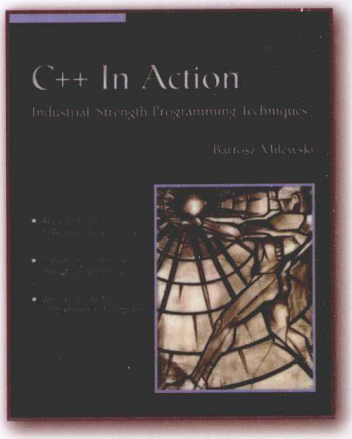
PEARSON

C和C++实务精选
品味岁月积淀，读享技术菁华

C++ 实践之路

[美] Bartosz Milewski 著　　周良忠 译

C++ In Action: Industrial Strength Programming Techniques



- 学习工业强度的C++编程技术
- 使用C++设计新思维编写Windows程序
- 学会编写清晰整洁、容易理解的C++程序

人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

C++ 实践之路

[美] Bartosz Milewski 著　　周良忠 译

人民邮电出版社
北京

图书在版编目（CIP）数据

C++实践之路 /（美）迈尔威斯基（Milewski,B.）著；周良忠译. -- 北京 : 人民邮电出版社，2012.11
ISBN 978-7-115-29134-9

Ⅰ. ①C… Ⅱ. ①迈… ②周… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第189529号

版权声明

C++实践之路

◆ 著　[美] Bartosz Milewski
译　周良忠
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：27
字数：598 千字　2012 年 11 月第 1 版
印数：1－3 000 册　2012 年 11 月北京第 1 次印刷
著作权合同登记号　图字：01-2011-7818 号
ISBN 978-7-115-29134-9

定价：79.00 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

内容提要

本书将带你领略C++作为工业编程语言的强大威力。

全书分四个部分，共23章。第一部分（第1章～第5章）从面向对象的角度讲解了C++的相关语言知识。第二部分（第6章～第13章）介绍了许多实用的工业强度的编程技术，如清理、隐藏实现细节、资源管理、重载运算符等技术。第三部分（第14章～第18章）探讨了编写和维护Windows应用程序的相关知识，是本书的特色部分。第四部分（第19章～第23章）在以前所学知识的基础上进一步对软件设计策略、团队协作开发、平台移植等内容进行简单介绍。

附录A提供了第一部分内容中部分练习的答案。附录B介绍了事务处理相关知识。

本书适合于不同层次的C++程序员阅读，无论是初学者还是高级程序员，都可从中汲取有用的C++知识营养。

Preface to the Chinese Edition

I was lucky to have a lot of feedback from my readers since this book was published in the United States. As my readers were trying to learn programming,I was learning about the strengths and weaknesses of my book. Here are some of the things I have learned.

The first part of the book, the introduction to C++, seems to be too difficult for most beginners. If you are a beginning programmer, don't get discouraged the first time your read it. I did use this material when teaching college students. I also used this book to teach C++ to a nephew of mine, who is a high school student, but the truth is I had to do a lot more explaining than the book does. A good teacher may use my book as a basis for a C++ course for the beginners, adding in-depth explanations and developing additional examples.

The traditional way of teaching C++, starting with the C subset, is now widely considered ineffective. My approach was to go straight into object-oriented programming and introduce procedural and generic programming later. This is why I didn't make much use of the C++ Standard Library in the beginning chapters.

There is another school of thought, which lets the students jump directly into the Standard Library and learn C++ as if it were Basic[1] . This way the students can start doing useful things from day one, but they have to accept a lot of features on faith. They learn to use templates, containers, iterators, and so on, without understanding how they work or how they fit into the language.

The best approach depends on the particular audience and probably falls somewhere in between the two schools—introducing a few staples of the Standard Library from the very beginning but keeping the main thrust of the initial course in the area of object-oriented programming. If I were to rewrite my book now, I would probably introduce standard strings in the introductory chapters. Strings and input/output streams are easy enough abstractions—they can be viewed as elementary building blocks of C++ programs.

Advanced programmers gave me a lot of positive feedback on the second part of the book—the Techniques. In particular, the Resource Management techniques described

---

[1]This modern approach was introduced in *Accelerated C++: Practical Programming by Example* by Andrew Koenig and Barbara Moo, Addison-Wesley 2002.

there turned out to be an eye opener for many. These are simple techniques that make an enormous difference to the quality of programs. I am becoming more and more convinced that Resource Management in C++ is more powerful than garbage collection in other languages.

The chapters on Windows programming brought the most interesting feedback. I've learned that a lot of programmers shy away from Windows programming because they believe that Windows applications can only be developed using one of the commercially available libraries. The quality of such libraries is often very poor, and, as it's often true with programs, whatever they lack in quality they make up for in complexity. Many readers of my book were relieved to see how Windows API can be encapsulated in uncomplicated modern C++ constructs that are both easy to understand and safe to use.

Finally, the last few chapters of the book, which talk about the complexity of software development, are getting more validation as time goes by. Programmers' time is getting progressively more expensive, making the search for better programming methodologies and tools even more urgent.

I am happy to see my book made available to Chinese programmers. I hope it will contribute to the quality and reliability of the software they create.

In my mind, the unique challenge for the Chinese programmer is to make the computer work seamlessly with the Chinese language and, in particular, with the Chinese script. I realize that my book falls short in this area. It doesn't cover such important topics as wide character strings and streams, Unicode, or localization. There are whole areas of C++ (and Windows) to support a wide variety of languages, scripts, and customs. I can only hope that my Chinese readers will supplement the material from this book with additional studies of these topics.

Bartosz Milewski

September 2002

中文版序

自从本书在美国出版后，我有幸收到了许多读者的反馈。读者在利用本书学习编程的同时，我也意识到了本书的优势与不足之处。下面将我的体会奉献给大家。

本书的第一部分“语言”对于大部分初学者来说似乎太难。如果你是一位入门级的程序员，开始阅读这部分内容时不要被吓倒。我在给大学生讲课时使用的就是这部分材料。我还使用本书来教授我的侄子，他只是一名高中生，但实际上，在书本内容之外，我还得进行额外的解释。有经验的教师可以使用本书作为C++初学者的基础课程，再辅之与相关解释和示例，完全可以达到理想的教学效果。

讲授C++的传统途径是从C的子集开始，现在人们普遍认为这种方式效率不高。我的教学方法是直接切入面向对象编程，并随后介绍过程化和泛型编程。这就是我为什么没有在开始章节中使用太多C++标准库的原因。

还有另一种教学法，即让学生直接跳到标准库并学习C++，把这部分内容当成基础知识[1]。按照这种教学法，学生第一天起就可以做一些有用的事，但他们必须首先无条件掌握许多知识点。他们必须学会使用模板、容器、迭代器等，而不一定理解它们的工作方式或如何适应于该语言。

最佳教学方式根据特定的读者而有所不同，也许应该是上述两种方法的折衷——从一开始就介绍一些标准库的主要知识，但重点放在面向对象编程上。如果让我现在重写本书，我大概会在入门章节中介绍标准字符串。字符串和输入/输出流是比较容易的抽象概念——它们可以看作C++程序的主要构建块。

高级程序员在反馈中对第二部分内容（技术）给予了肯定。特别是，这里描述的“资源管理技术”让许多程序员恍然大悟。这些技术虽然简单，但应用的好坏对程序质量产生的影

[1] Accelerated C++: Practical Programming by Example(Addison-Wesley 2002)一书采用的就是这种现代的方法，作者是Andrew Koenig 和 Barbara Moo。

响却是天壤之别。我越来越相信，C++中的资源管理比其他语言中的垃圾回收更强大。

“Windows”一部分提到的反馈最有趣。我了解到，许多程序员对于 Windows 编程退避三舍，因为他们认为只能使用商业库才能开发出 Windows 应用程序。这类库的质量往往非常低劣，而且事实表明，质量的低下往往带来程序的复杂性。许多读者在本书中看到 WindowsAPI 可以封装到并不复杂的现代 C++结构中，而且这种作法易于理解，使用安全；这使他们甚感欣慰。

最后谈谈本书的结尾几章。它们讲解了软件开发的复杂性，随着时间的延续，这种复杂性将得到进一步的证实。程序员的时间日益宝贵，找到更好的编程方法和工具甚至显得更为紧迫。

我很高兴看到本书能与中国程序员见面。我希望本书能为他们创建高质量、高可靠性软件出一份力。

在我的记忆中，中国程序员面临的独特挑战是让计算机能与中文，尤其是与中文字符无缝工作。我意识到，对于中国读者来说，本书缺少了这方面的知识。本书尚未涉及到宽位字符串和流、Unicode 或本地化这方面的重要话题。要支持各种不同的语言、字符和习俗，这涉及到 C++(和 Windows)的各个方面。对于这些主题，我只能希望中国读者在看过本书之后，额外地学习其他补充材料。

Bartosz Milewski

2002 年 9 月

序

为什么写这本书

的确，现在详细讲解C++各方面知识的优秀书籍为数不少，为什么还要写这本C++书呢？对于学习这一语言及各种编程技巧来说，市场上提供的书籍已相当饱和了。但本书不是一本语言参考书或高明技巧和模式的集合，本书旨在讨论编程方面的知识。

讲授编程与教授一种语言大不相同。程序员通常要面对一个必须通过编写程序来解决的问题——而不是面对他需要讲解的语言功能。在本书中，我将尽可能演示如何使用C++这一语言作为一种解决编程问题的工具。

本书以几个软件项目为中心展开。在每个项目中，我首先描述即将解决的问题。然后，我讨论程序应该做什么、应该是什么样，以及如何对用户的输入做出反应。在此基础上，我构建一个没有实现具体功能的程序框架。最后，我将按组件逐个实现相关功能。

不过，程序也不会到此停止不前。接下来，本书还要进行一系列的代码审查并进行改写。“怎样才能精益求精”？这是程序员经常自问的问题。再接着另一个需要解决的问题是“如何编写容易修改的代码”？

与现实生活一样，编程的途径也不是唯一的。这就是我们讨论不同解决方案优缺点的极其重要的原因。不知道如何比较不同方案的程序员将举步维艰。当我讨论“因为人们总是这样做……”这样的问题时，我会说出自己遇到的挫折。在本书中，这样的讨论有很多。我努力探讨每种解决方案的优缺点，而且在多数情况下，我设法找到我认为理想的方案。不过，我绝不会以“理想”作为评判目标。我相信，每个人都可以在这些所谓“理想”的背后提出一些非常实际的意见。在多数情况下，一个理想的解决方案进行了非常优秀的抽象或通用化。这样生成的代码易于理解、修改和调试。

最后，在本书中，我强调了编程中人的因素。我的座右铭是“程序是为程序员而写的，不是为计算机而写的。”程序员希望编写更优秀的程序，不是为了让它更易于计算机理解，而

是让程序更易于让人理解。不理解程序是不可能维护程序的。这一点看起来似乎明显，但许多程序员却忽视了这一不言而喻的真理。

为什么适合你阅读

你是哪一类读者？你可能是一个接触过编程但想学习 C++的初学者，可能是一个想要完成大学教育的学生，可能是一个想从学术型上升到工业环境的熟练程序员，或者是一个需要获取新创意的经验丰富的程序员。无论你属于哪一类，我都希望本书能成为你的至爱。

我为什么写这本书

为什么偏偏由我来撰写这本 C++编程书？可能成为本书读者的你有权知道我的资历，尤其因为我不是一个教育界的计算机科学家。

我获得了一个理论物理学博士学位——我的论文是关于超对称非线性 δ 模型。我过去擅长计算超对称费曼图。然后，大约在 1986 年，我开始迷恋计算机，于是我放弃了博士后学位并开始编程。

物理学家是一种介于数学家和工程师之间的人。一个物理学家习惯于穿梭在理论和实践之间。物理学家不应该相信理论，除非他或她在实践中证实了自己的理论，而且他或她总是尽量总结各方面经验来发现一条新原理。尤其重要的是，一个物理学家不应该不加批判地接受一切。

与数学家不同，物理学家必须更现实，更关心过程而不是形式上的理论。物理上的过程就是进行计算。而在编程中，过程就是软件维护。因为我是一个物理学家，所以在本书中，我的讲解重点是程序的改写而不是程序的创建就不足为奇了。

我从 1994 年开始讲授 C++并编写本书，当时我仍然在微软公司工作。我时常造访我的母校——波兰 Wroclaw 大学。每次造访都会促使我撰写更多的章节，我真想在那儿汲取更多的营养。在此期间，我开始将本书转换成 HTML 文档并在网上发布。因为 C++还在发展，我必须一而再、再而三地改写许多章节。例如，当 C++标准库中添加了 STL 时，我必须回过头来重新构思本书大部分代码的实现过程。

当我在微软从事设计并主持索引服务器的开发时，本书所描述的许多技术已经得到了发展。此后，我的新公司（Reliable Software）成为了测试新的编程方法的极好的实验室。本书所描述的所有技术都已集成到我们的产品中。事实上，当每一个更好的新想法出现时，我们都会改写我们的代码。

正因为是一个物理学家，所以我既是一个称职的程序员，同时也是一个不称职的程序员。之所以称职，因为我不断修改我所使用的技术、发现新的通用化方法并彻底探讨编程的各个方面；之所以不称职，是因为当我发现一个更好的解决方案时，我就想改写所有代码（我将尽量让你相信，这一想法实际上并不太坏）。我以前的工作主管总是挑我的刺，因为我质疑每个决定、惯例及习惯。我希望这一同样品质也能让我成为一个优秀的作者。

如果读者对本书提供的代码有更好的解决方案，请与我联系，我可以对本书进行第 n 次的重写。我的 E-mail 地址是：bartosz@relisoft.com。

致谢

真诚感谢 Wroclaw 大学和 Digipen 工学院的我的学生，因为讲授一种语言是学习它的最好途径。感谢 Jerzy Lukierski 教授，是他邀请我回到学校讲授计算机科学的。我同时感谢微软和 Reliable Software 公司的同事，是他们第一次尝试并改进了本书中所描述的技术。感谢我的妻子 Pam，除了不遗余力地支持我以外，她还帮助我编辑了本书的第 1 章，使这一章更加简单易懂。最后还要感谢 Addison-Wesley 出版社的编辑，尤其要感谢的是 Debbie Lafferty。

译者序

软件项目的成功与否与开发语言的选择息息相关。而在确定某种语言为开发工具前，我们通常要提出这样的问题“这一语言能高效地实现我们的软件目的吗？最终产品是否易于维护？它与其他语言相比优势何在？”当今的很多商业软件首选C++作为他们的开发语言，就是因为C++对以上问题能提供令人相当满意的答案。简言之，C++是ANSI标准C语言的增强版本，但它对C语言的扩充是革命性的。C++不仅支持面向对象编程，而且它还可以方便地构建一个由相关对象组成的层次等级树，同时，它出色的可维护性和可扩展性使得它成为商业软件开发中最流行的语言之一。

要想成为一名优秀的C++程序员，仅仅具备C的相关知识背景是远远不够的，而且C的一些习惯思维方法并不利于充分利用C++的高效与先进性。因此，学习一些C++编程技巧是每个合格C++程序员的必修课。本书正是一本为我们提供大量实用C++编程技巧的书籍。

本书第一部分从面向对象的角度讲解了C++的许多语言要点，初学者可以对C++面向对象编程获得更清晰的认识。第二部分则讲解了大量实用技术，这些技术经过广大程序员的多年实践证明是开发可靠程序的必备技巧。例如，资源管理技术是任何用C++开发的商业程序必须正确处理和应用的核心技术之一。本书的另一个可贵之处是对Windows应用程序开发的诸多技巧进行了一定深度的介绍。Windows应用程序的开发难度大，同时，相关资源也难以获取，所以，这部分知识对于专业程序员来说无疑具有宝贵的参考价值。最后，作为一名工业强度型程序员，他还必须具备软件设计、团队协作开发、平台移植等多方面的知识，本书也对这方面的知识进行了讨论。

为了方便读者的学习与实践，本书还附带一张光盘，你可以从光盘上找到本书所涉及的所有源代码及项目。

由于译者水平有限，错误在所难免，望广大读者不吝指正。

前言

我将本书分成4个部分：语言、技术、Windows和知识扩展。下面简要描述每部分的目标和结构。

语言

本书的第一部分重点讲解作为通用目的编程的C++语言，然而这不是你常见的C++指导。

对于那些对C或C++知之甚少的初学者来说，本部分内容只是作为一种面向对象的语言介绍了C++（尽管C++的内涵远非如此）。本部分没有重点介绍语法或语法规则，它只是说明如何在C++中表达一定的思想，就像通过对话而不是背单词和语法规则来学习一门外语一样（当我给学生授课时，我称这部分内容为“对话式C++”）。毕竟这是程序所需要的：能以特定语言编写程序的形式来表达自己的思想。当我学习一门外语时，我想知道的第一件事就是如何使用这样的问句“How much does it cost？”我不需要学习“cost”的过去时、现在时及将来时的所有动词时态，我只希望能在国外顺利地购物。

对于一个不太了解C++的C程序员来说（不等于说C++不活跃、很神秘，但C的其他一些子集往往有这种弊病），这是一个忘记C、有效运用C++编程的练习机会。为什么要忘记C？C++不是C的一个超集吗？的确C++是C的一个超集，让C++与C兼容完全是一个符合实际市场决策，而且这样做取得了成功!它不是一个全新的产品，它不必花费十年的时间来单独开辟市场，它是“3.1版本”的C。这样做也是利弊相随的。“利”在于C++及一些面向对象的编程元素与C向后兼容，这样可以迅速在编程领域占领一席之地“弊”在于它并不需要程序员更改编程习惯。

不必一次性重写所有的现有代码，许多公司可以逐步过渡到C++，现在很多公司仍然如此。通常的过渡方式就是将C++当成一种“更严格”的C来引入。从原则上讲，所有的C代码都可以作为C++代码重新编译。实际上，C++有着更严格的类型检查，而且编译器可以检测到更多的错误、发出更多的警告。所以，使用C++编译器重编译C代码是一个清理现有代

码的好办法。在此阶段对源代码要做的修改主要是更正错误、实施更严格的类型。如果代码是用前 ANSI C 写成的，则会生成所有函数的原型。在这一 ANSI 化的过程中发现的错误之多让人感到吃惊。为了顺利过渡，以上所有工作都是值得的。只有在缺少优秀的 C++编译器时（当然这种情况很少发生）人们才去使用 C 编译器。

一旦过渡到 C++编译器环境中后，程序员迟早要学习一些新技巧，并最终开发一些 C++编程方法（或通过自学、阅读一些辅助性书籍来达到目的）。但此时，许多程序员会受到一些不良的影响。因为 C++的一些子集是许多原来 C 程序员应用过的（我将此子集称为“C 特区”）。许多 C 程序员看到这些“C 特区”后开始憎恨 C++，但他们并没有意识到 C++同样具有很多优秀之处。

在本书的前言中，我想澄清一件事。对于一个“C 特区”程序员来说，本书对他们应该是一种冲击（我希望如此）。从本质上讲，“你到目前为止所做的一切都是错的”、“Kernighan 和 Ritchie 并非神”（Brian Kernighan 和 Dennis Ritchie 是 C 的创建者，也是著名的 *The C Programming Language* 一书的作者）。我知道，这类程序员的第一个自然反应就是合上本书立即要求退款。不要冲动!本书打破习俗的用意不在于伤害某个人的感情；而在于以一种不同的哲学思维来提醒读者重新思考他们的信条。

对于 C++程序员来说，“语言”部分提供了一个审视 C++的新角度。它演示了如何避免 C++的缺陷以及如何根据先设计后具体实现的方式来应用这一语言。如果我说 C++是一种优美的编程语言，这是不真实的。但它即将成为编写正规软件的最流行语言（至少在某些时候是）。我们最好充分利用它的丰富表达力来编写更出色的软件而不是用它不利的一面来伤害我们自己。对于一个 C++程序员来说，这一部分内容不难读懂。而且，虽然这里介绍的结构和技术已广为人知，我还是尽量从不同的角度来展示它们。我的主导思想是创建一个易于维护、可读性强的系统。这就是为什么我抓住每一次机会不仅演示不同的编程选择，而且还解释为什么优先考虑某种方案的原因。

最后，对于一个 Java 程序员来说，他们可以利用本书来扩展知识。事实表明，根据一定的原则，利用 C++可以编写出安全和可靠的代码。Java 能做到的，C++也能做到，而且能做得更多。另外，C++能提供无可匹敌的操作性能。

然而操作性能不是投身 C++的唯一原因。完善的资源管理在 C++中能实现，但在 Java 中不可能实现，因为 Java 要依赖于垃圾回收机制。在 C++中，你可以创建这样的对象：它的生存期由它所在的作用域准确定义。在退出这些作用域时，你要保证这些对象将被销毁。这就是为什么你可以让 C++对象作为一个信号量来负责处理这些重要资源（如文件句柄、数据库事务处理等等）的原因。Java 对象有一个未定义的生命期——只是在运行时决定回收它们时才释放这些资源。所以，说到 Java 中处理资源的方式就得重提旧式 C 的异常范例，在这种情况下，最后的子句必须要进行痛苦的显式垃圾回收。

没有“纯粹”C++出身的程序员。当我们“谈到”C++时，或多或少都会涉及其他一些编程背景。我们中间的一些人有着深厚的 C 背景，一些人使用类似 Smalltalk 的表达方式，还

有一些人在使用 LISP。“语言”部分的目标是尽可能让你成为一名正统的 C++程序员。语言是表达思想的一种工具。所以，我的讲解重点不是语法而是程序员表达自己的方式。在这里不是介绍“C++的结构是什么，你应该怎么用”，而更多的是以“我们有了一种思想，那么该怎样用 C++来表达呢？”的方式来讲解。最初，这一“思想”可能以“星球是一种天体”或“一个堆栈允许你用推入或弹出数据”的简单句子形式表达，以后再将这些句子组成“段落”和“章”来描述软件组件的功能。在需要解决相关问题的适当场合介绍不同的 C++结构。

技术

开发优秀软件不仅仅需要了解语言本身。首先，程序不是在真空中执行，它必须与计算机交互作用。交互意味着通过操作系统来发挥作用。若对操作系统一无所知，就不可能编写出像样的程序。其次，我们不仅需要编写能运行的程序，我们还希望程序小巧、快速、可靠、强健以及具有扩展性。最后，我希望在合理的时间内完成程序的开发，并且能在以后维护和改进此程序。

本书第二部分“技术”的目标是让你从“业余编程”过渡到“工业强度型编程”。我描述了一些可以使程序更强健、维护性更强的 C++编程技术。其中的“资源管理”重在介绍程序的创建、获取、拥有及释放各种资源。在程序执行的任一时刻，每一种资源都必须存在一个明确的拥有者负责释放它。这一简单的思想在设计和维护复杂的软件系统中相当有用。运用资源所有权关系分析法可以避免、发现和排除许多 bug。

资源管理与 C++异常处理能自然地吻合。事实上，如果不封装资源几乎不可能使用异常来编写合理的 C++程序。那么，什么时候使用异常呢？它们能为我们做些什么呢？这取决于你对下面简单问题的回答：你总是检查 new 的结果吗（或者对于 C 程序员来说就是检查 malloc 的结果）？这是一个反问句。除非你是一个格外小心的程序员，你才不用这样!这意味着不管你希望与否，你都已经使用了异常，因为访问一个空指针就会导致一个叫做“一般性保护”（程序员也称做 GP 过错或访问违例）的异常。如果你的程序没有添加防异常措施，当它遇到这种异常时就会死机。甚至操作系统会弹出一个消息框，证实你的应用程序所使用的技术不合格。

我的观点是，为了编写强健可靠的应用程序，使用异常是最终的选择——这也是本书的目的。当然，还有其他一些可成功地应用于开发强健可靠应用程序的编程技术（这些技术现在仍然被我们使用）。不过，结合应用 C++异常与资源管理带来的简单性和可维护性是这些技术所不及的。

Windows

就我所知，这是第一本深入探讨了使用现代 C++进行 Windows 编程的书籍。事实是 Windows API 很混乱，现有的库也同样不是完全规范一致，使得 Windows 应用程序的编写不是那么简单，但这并不意味着这一主题就没有意义。在 C++类、名字空间及模板中封装

Windows API 是一种挑战，这需要对整个 Windows 范例的重新深层构思。

这部分内容的主要目标是探讨创建和维护 Windows 程序的简单途径——更侧重于维护。我常常问我自己这样的问题："添加或修改一个消息句柄应该简单到什么程度？添加一个新菜单项、一条命令、一个对话框呢？而且如何才能将程序员犯错的可能性降到最低呢？"最后，还要考虑将库移植到例如像 Linux 的其他平台上（但这不是必需的）。

本书的 Windows 部分反映了一个不断改进的工作。事实上，这一工作多少年来一直在不断改进。对本部分所描述的每一个思想都有若干种不同的实现方法，并进行了测试（通常集成在某一产品中），也可能由于找到了更好的方案而最终放弃了。

知识扩展

在这一部分内容中，我涉及了软件开发的许多扩展知识。我侧重讨论软件项目的动态性，既从管理和规划的角度入手，同时着眼于开发策略和技巧。我从软件项目的概念到发行描述了其动态性。我讨论了文档、设计过程以及开发过程。不过，我不想提供一个现存的方案，因为这样的方案不会在任何时候都适用——原因如下所述。

成功开发一个应用程序（或系统）不是光靠学习一种语言、掌握其一定的技术就足够了。当今的商业软件项目是人类从事的最为复杂的工程之一。编程实质上是一门处理复杂性的艺术。很多人尝试使用传统的工程方法来控制软件的复杂性，模块化、软件重用、软件 IC（integrated circuit，集成电路）等——但我们要面对这样的事实：通常它们不能正常发挥作用。它们可能对提供低层构建块和库非常有用，但它们很难用作设计和实现复杂软件项目的指导原则。

原因很简单：软件块中的重复很少。试着比较一段程序的打印输出与一幅微处理器晶片图，你可以发现微处理器布局上的许多重复模式。这些重复块仿佛某种高科技晶体。另一方面，精简后的程序看起来更像一个高科技分形结构。你可以见到许多相似性——大模式与小模式相似。但你很少发现有完全准确的匹配或完全相同的重复。每个小块都像一个单独的手工艺品。程序中的重复不仅没有必要，而且还会给今后的维护带来困难。如果你修改某段代码或排除某个错误，你就要审查这一段代码的所有拷贝，而且还要同时对这些地方做出修改。

重复的弊病还反映在软件的生产过程中。软件工业中的研究、设计和制造不同于其他工业。例如制造的作用并非主导。严格来说，电子化发行渠道可以使制造阶段与之完全无关。R&D（研究与开发）扮演着重要角色，这一点比许多其他工业更突出。但真正区别软件开发与其他工业的是在产品中体现的设计成分。编程就是设计。设计、构建原型，一遍遍地测试。软件工业是最终的"设计工业"。

程序员在人们心中往往是这样一种灰色形象：独自夜战、封闭自我而且生活邋遢。我认识这样的程序员，但我仍然相信还有另一类程序员。这种旧文化的残留现在逐渐灭绝了，因为硬件的进步、软件的发展使得孤军奋战开发出有用的可靠程序已不可能。团队工作是软件开发必不可少的。

分解整个工作并在团队工作中协作努力总是一个大的挑战。在传统工业中，团队中的成员知道（至少在理论上）他们要做什么。他们知道例行程序。他们听着音乐，知道下面的步伐，并跳出同步的舞蹈。在软件工业中，团队成员即兴跳出舞步，同时为其余的队友谱出相应的“音乐”。

我提倡改变软件开发的重心。我将尽量让你信服这样的思想“程序是为程序员而写的”，而不应该遵循陈旧的“程序是为计算机而写的”的思想。这一论点是整本书的基点。如果你不把你编写的代码看作让其他程序员阅读、理解和修改的出版物，那么你就不可能开发出工业强度型软件。没有人希望自己编写的代码成为一段密码。计算机可以编译和运行你的程序，还要求你的同事可以理解它的意义。而且至关重要的是，只需极少的努力就可完成这一点，因为不理解代码就不可能开发和维护软件。

如何使用本书

缺乏足够的经验不可能学好编程。这是为什么本书附带光盘中包含所有源代码及相关软件的原因，它们可以供读者进行练习。如果你安装了我推荐的编译器之一，你就可以构建和运行本书涉及的所有程序。尤其是开始的一些例子以一套文件的形式提供给大家，它们保存在各自的目录中。你可以从光盘将这些文件复制到本地硬盘中并编译它们。不过，大部分例子需要一个版本控制系统。

版本控制系统

本书中的大部分代码示例属于一个软件项目，随着讲解的深入，这些代码得到了一步步的改进。提供这些逐级改进版本的最佳途径就是使用一个版本控制系统（Version Control System，VCS）。VCS 是软件开发的基本工具之一。它可以让你跟踪所有的变化、恢复到旧版本，而且更重要的是，可以与其他程序员相互协作。

随书光盘上赠送了 Reliable Software 公司开发的一种 VCS，这就是 Code Co-op。因为 VCS 是一个商业产品，所以通常你要购买使用许可证才能使用。不过，把它作为一个项目的“观察器”使用就不必获得使用许可证，而且这已足以让你学习本书中的项目了。

Code Co-op 有一些独特的功能，因此尤其适合于本书的学习。在安装 Code Co-op 的过程中，将你的计算机作为一个单机来配置。接下来，你就可以在学习软件项目过程中与我“交互协作”了。这种协作是单方面的：我开发了此项目，你将得到了同步脚本复制，它们描述了代码所作的修改。

注意，安装 Code Co-op 不会覆盖计算机系统中的任何 DLL 文件。运行卸载程序后，Code Co-op 将从你的计算机上彻底消失。

编译器

在本书成稿之时，集成了性能良好的 IDE（Interactive Development Environment，交互式