HZ BOOKS

PEARSON Addison Wesley

Modern C++ Design C++ 设计新思维

C++跨平台开发技术指南

Cross-Platform Development in C++

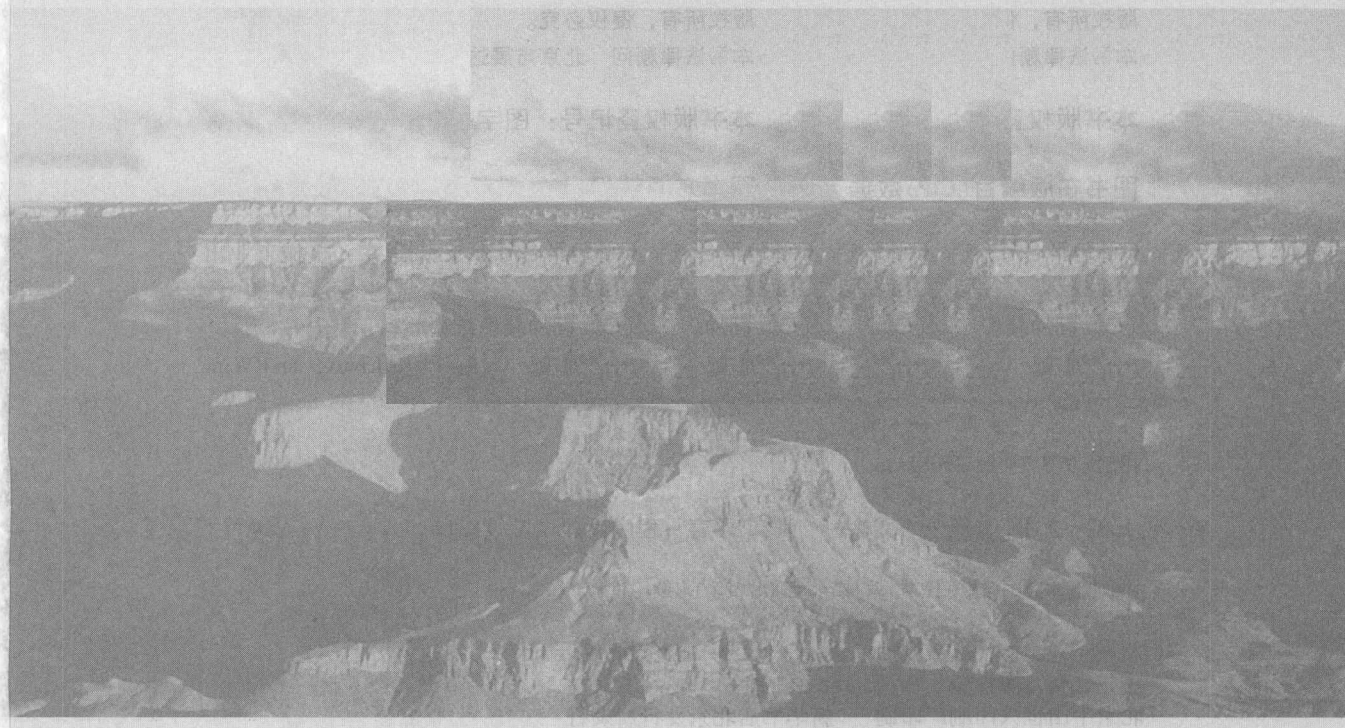
（美） Syd Logan 著
徐旭铭 译

机械工业出版社
China Machine Press

C++
设计新思维

C++跨平台开发技术指南

Cross-Platform Development in C++



（美） Syd Logan 著
徐旭铭 译

机械工业出版社
China Machine Press

本书详细介绍C++语言的跨平台技术，包含的主要内容有：Netscape在向数百万Windows、Mac OS和Linux用户发布浏览器时采用的策略和过程；如何使用基于标志的API，包括POSIX和STL；如何避免隐晦的移植性陷阱，相关的如浮点数、char类型、数据序列化，以及C++的类型；如何建立一个有效的跨平台bug报告和跟踪系统等。本书内容详实，实例丰富。适合软件开发技术人员参考。

本书版权登记号：图字：01-2008-1781

图书在版编目(CIP)数据

C++跨平台开发技术指南/(美)隆甘(Logan, S.)著；徐旭铭译. —北京：机械工业出版社，2008.9

(C++设计新思维)

书名原文：Cross-Platform Development in C++: Building Mac OS X, Linux, and Windows Applications

ISBN 978-7-111-25082-1

Ⅰ.C…　Ⅱ.①隆…　②徐…　Ⅲ.C语言－程序设计　Ⅳ.TP312

中国版本图书馆CIP数据核字(2008)第140940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：陈佳媛

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2009年1月第1版第1次印刷

186mm×240mm · 21.25印张

标准书号：ISBN 978-7-111-25082-1

定价：49.00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

译者序

这是一个 Java 横行，LAMP 肆虐的年代。很多人说 C 和 C++早已落伍了，将在网络大潮中默默隐去，但当你读完本书，便会体会到 C 和 C++的魅力所在。

本书向读者完整地呈现了 C 和 C++在跨平台领域上的方方面面。从丰富的工具，完善的软件工程，到开发、部署、安装以及系统设计。什么问题应该避免，什么地方需要注意，十几年的经验让本书作者深谙此道。他不用理论来证明自己，而是用作品来展示理论。只有高手才能达到这种境界。

说到本书作者，Syd Logan 是一位资深工程师，在 C 和 C++领域耕耘数载。他全程参与了 Netscape 和 Firefox 浏览器的开发工作。此外，他还负责为 AOL 开发 VOIP 和点对点的视频功能。

作为本书的译者，我也在翻译过程中获益匪浅。作者展示的是多年来在跨平台开发中的心得体会，即便你开发的不是 C 和 C++的跨平台项目仍可从中受益。毕竟语言只是工具，设计思维才是最宝贵的财富。

无论 Internet 怎样发展，一切内容都要通过浏览器展示给用户。当年微软正是意识到这一点，才对 Netscape 发动了浏览器战争。出色的技术和商业成功没有必然联系，Netscape 倒下了。然而其中的积淀却借着 Mozilla 凤凰涅槃。相比 IE 的不思进取，Firefox 3 如火如荼的发布正说明了优秀的作品最终将得到用户的肯定。

我也希望大家能从本书中找到自己想要的知识。

如发现任何纰漏之处，敬请指正。

在此感谢华章图书的陈冀康老师对我的信任和鼓励，还要感谢 Lydia 和 Ada 的支持。

徐旭铭

序

作为 Firefox、Mozilla 和 Netscape 浏览器解析和渲染超文本标记语言/可扩展标记语言/层叠样式表(HTML/XML/CSS)的引擎，Gecko 是全世界使用最广泛的渲染引擎之一。

而我身为 Netscape 的工程师以及后来 Mozilla Gecko 团队的开发经理，有幸从一开始就参与了 Gecko 引擎的开发。

Gecko 诞生时的愿望是要创建一个跨平台的、小巧快速的、先进的可嵌入 Web 浏览引擎，而这一点正是我们在“浏览器大战”中争夺优势的砝码。当时笨拙的 Netscape 4. x 引擎显然已经无法再完全支持 CSS2、CSS3 和 XML Web 标准了。所以有人提出只使用原有的引擎的一部分库来重新开发。在 Gecko 项目的早期，我们曾经讨论过要采用拥有跨平台能力的 Java 而不是 C++。但最后还是觉得只有 C++以及它特有的开发过程、工具和设计技术，才能产生最好的解决方案。本书将会把这些过程、工具和设计技术作为最佳实践来逐一描述。

在进入 Netscape 工作之前，我曾为很多公司开发过跨平台软件。然而，Mozilla 项目把这些经历提升到了一个完全不同的高度。我们使用和开发了一系列软件架构、工具和过程来实现大范围内的跨平台开发工作。

我的第一个任务是把 Gecko 从微软 Windows 系统上移植到 Motif/Xlib 上去。写过跨平台软件的人都知道，刚开始的移植工作是最有挑战性的。你会在那时发现软件的移植性到底如何。即使 Gecko 从设计的时候就是以可移植为目标，然而平台和编译器之间的细微差异还是搞得我们很头疼。这就是为什么需要有一个像 Mozilla Tinderbox 那样的工具来验证 check-in 代码的可移植性，并且还需要一个软件开发过程来要求工程师在向代码仓库提交新的源代码之前至少在两个平台上验证过。

Gecko 引擎开发的动机之一，是要在它上面重现 Netscape Communicator 的用户界面体验。这就要求有一个跨平台的用户界面方案，因为 Netscape Communicator 就是在多个平台上提供了图形用户界面的环境。所以我就有了这样一个机会来设计一个用户界面策略以解决这个棘手的跨平台问题。我撰写了一份文档，解释了如何在 Gecko 渲染引擎里把描述用户界面元素的 XML 元描述(XML meta description)与作为控件和事件逻辑的 JavaScript 相混合，以达到跨平台用户界面的目的。这份文档后来成为了 XUL(XML User Interface Language)的原型。再后来，Firefox 的开发人员用 XUL 和 Gecko 引擎开发出了小巧快速、广受欢迎的跨平台 Web 浏览器。第 9 章描述了如何在 XUL 上创建你自己的跨平台用户界面。

作为 W3C SVG(Scalable Vector Graphics)最早的成员之一，我对 Gecko 能不断地进化和解决跨平台问题感到非常兴奋。最近添加的对 SVG 的原生支持则又是 Gecko 可移植性的一大胜利。

Syd Logan 在这里要展示的信息是无数工程师对于那些和创建跨平台软件产品相关的特殊问题的真知灼见。虽然它是以 C++为蓝本，不过很多技术都是可以为其他非 C++软件项目所借鉴。我希望你能从中发现有用的工具、技术或是过程，从而避免跨平台开发中的陷阱，让你的项目取得巨大成功。

Kevin McCluskey

前言

在1998年加入Netscape前十余年的职业生涯里，我有幸在很多不同的平台上参与了很多不同的项目。我曾在一种很难懂的CPU(TMS34020)上用自己设计的嵌入式内核工作。我从一个编写文件系统驱动程序以支持网络文件系统(Network File System，NFS)的客户端开发Windows NT和Windows 98内核的项目中获得了Windows内核开发的经验。在用户端，我最擅长的是用户界面开发，从最初为UNIX开发Motif(Z-Mail)和OpenWindows程序，到最后在Windows平台上接触Win32以及MFC。我甚至还曾经有机会为Apple的一个在早期Mac OS上的项目用Mac Toolbox API编写代码。所有的这些代码都是用C写成，全部都是高度不可移植，只是为了特定的工作和平台编写的代码。

然后，我作为一名UNIX专家加入了Netscape。刚开始的时候，我的任务是修复Netscape 4.x浏览器上的bug，特别是处理一个移植到IBM AIX平台上的版本的问题。Netscape和IBM签订了一份合同，保证AIX版本的Netscape浏览器上的bug或者IBM认为重要的bug都要在一定的时间内修复。我就是因此而被雇用的。类似的和SGI、HP、Sun或者其他平台签订的合约让Netscape雇用了很多额外的员工。我们有两到三个人专门对付AIX平台上的bug。

在当时，可移植性还没有得到Netscape的重视。虽然Netscape的很多代码都是可移植的，但是项目都还没有一个统一的构建系统，用户界面的代码也完全是平台相关的。很多bug都是由于平台相关的天性所造成(所以需要不同的团队支持不同的平台)。在Windows上完美运行的Netscape在缺乏支持的平台上就跑得一塌糊涂。不是所有的平台都拥有相同的特性，特性在平台与平台之间的表现也不尽相同。

加入Netscape为AIX修复了一年bug后，我转到了Netscape即时通信工具(Instant Messenger)的团队里，在新的基于开源的Mozilla平台上工作。这个三人团队的工作是把AOL Instant Messenger客户端移植到Netscape浏览器里。Netscape IM团队是在AOL收购Netscape后仓促成立的，目的是为了把基于AOL的功能整合到应用程序里。(另一个整合到Netscape里的关键项目是AOL Mail。)

就像前面提到的，开发中新的Netscape客户端是基于开源的代码库Mozilla。这个代码库在那个时候绝大多数是由位于圣地亚哥和山景城的Netscape工程师开发，由开源社区贡献代码的。(我在此后把这个项目叫做Netscape/Mozilla。)

当时Netscape和微软在浏览器市场上争夺得非常激烈，所以浏览器必须在Windows上工作正常，按时发布。Netscape从Netscape的门户网站上获得了大量的广告收入，当新版本发布的时候点击率是最高的，成千上万的用户都涌入网站下载最新版本的Netscape。不光Windows，在Mac OS和Linux上支持Netscape也促成了高访问量和高收入。所以在Netscape内部，Linux和Mac OS与Windows拥有同等待遇。不仅仅因为这是“正确”的事情(就像我们大多数人相信的一样)，也因为对公司来说，每个访问都是非常重要的。

Netscape/Mozilla在我看来是一个全新的起点。首先，新版本的Netscape不只是一个浏览器，它是一个包含应用程序的平台。(随Netscape发布的主要应用有AIM、邮件/新闻客户端、一个名为Composer的所见即所得的HTML编辑器、Chatzilla IRC客户端和浏览器自身。这就类似于今天Firefox的扩展。)

Netscape/Mozilla采用自己开发的技术来构建图形用户界面而不是用本地平台提供的MFC或者Gtk+等C/C++GUI工具。像HTML描述网页布局一样，用户界面的布局使用静态XML文件来描述。

为这个目的而开发的 XML 叫做 XUL。像网页里链接到 HTML 元素的 JavaScript 一样，通过 attribute 链接到 XML 元素的 JavaScript 代码用来响应菜单选择和按钮点击。只要有这个 XML 和 JavaScript 的组合就能为 Netscape/Mozilla 构建应用程序。而且因为 XML 和 JavaScript 生来就是可移植的，所以采用这两项技术设计出来的用户界面同样是可移植的。如果 JavaScript 不足以完成某些功能(可能是任何真正的应用程序，就像 Netscape 和 Mozilla 发布的那些)，JavaScript 代码可以从共享库里去调用 C++函数。这些共享库，或者说组件，在 Netscape/Mozilla 架构里通过两种技术来直接支持：XPConnect 和 XPCOM。这些技术允许组件开发人员用界面定义语言(Interface Description Language，IDL)来定义未知平台的界面。JavaScript 代码可以使用 XPCOM 和 XPConnect 来查询一个组件是否存在，然后查询一个特定的界面。如果一切顺利，JavaScript 代码会获得一个可以和其他任何对象一样调用的对象(只要它不是用 C++写的)，可以做到任何 JavaScript 程序员能想到的事情。这些界面自身就是高度平台无关的。

老实说，Netscape/Mozilla 架构里那些为了支持移植性而做的工作对我来说一开始不是很明显。但是渐渐地，我变得非常感谢这个方法所带来的力量。提出这个架构的决定所带来的积极影响是无可争议的，Netscape 不单单向 Windows、Mac 和 Linux，还向 SunOS、AIX、HP-UX、SGI Irix 和其他许多类 UNIX 平台的用户提供了千百万份浏览器。针对“tier－1”平台(Mac OS、Linux 和 Windows)的产品基本上是同一时间发布的。这些版本基本上拥有相同的特性，当然，也拥有相同的 bug。要在如此大的范围里达到可移植性需要一个非常特殊的架构。这也是本书的目标之一，让你对 Netscape/Mozilla 架构对代码移植性的直接影响有一个清晰的认识。

不过，让浏览器和相关应用(AIM、Mail、Composer)可移植的不仅仅是 Netscape/Mozilla 的架构。要想实现这个目标，光有一个可靠的架构是不够的，还需要在一系列的策略和过程中把跨平台开发作为高优先级的考量，以及严格的纪律来保证这些准则得以遵守。Netscape 和 Mozilla 在 Tinderbox 和 Bugzilla 这样的工具上的巨额投资都迅速得到了回报。工程师被强制为其他平台而不仅仅为他们自己的平台考虑问题，如果在日常测试中发现某个平台有问题则可能会导致所有平台上开发的停滞，而不只是那个平台受到影响而已。因为 Netscape 和 Mozilla 意识到要真正实现可移植性的惟一途径就是有问题当场解决。本书尽量避免涉及代码而偏向描述准则，就是因为无论架构在支持跨平台上有多出色，如果要最终完成相同质量的产品，你必须在所有要支持的平台上巨细靡遗，倾入相同的关注。

和我们编写的数据结构和算法的程序一样，在我看来，可移植性很大程度上包含了架构和过程。这个信念正是你手上这本书的基石。

本书的组织方式

本书由一系列章节组成。大多数章节都包含一组条款。每个条款涵盖了一个支撑章节主题的特定话题。在书本的开头，你会发现一些小节包含了这样一些条款，它们所介绍的最佳实践必须和整个开发组织沟通，包括管理部门、开发部门和测试部门。之后的章节则覆盖了管理人员应该了解的软件工程，不过它们其实主要是写给那些编写代码的读者看的。在前面的几章里，一共介绍了 23 条条款。

用户界面的实现在开发跨平台的桌面应用中占有很重要的地位。条款 23 将会介绍这个主题。最后两章则讨论了跨平台 GUI 的相关论题。其中的第 1 章全面介绍了了跨平台 GUI 工具包 wxWidgets。你还可以查阅 Prentice Hall 出版的关于这个主题更详细的书籍，Julian Smart 等人所著的《Cross-platform GUI Programming with wxWidgets》。wxWidgets 并非惟一可用的跨平台 GUI 工具包。另一个可选的，同时也十分流行的跨平台 GUI 工具包是 Qt，它不在本书所讨论的范围之内。但是，如果你对 Qt 有兴趣，市面上有很多介绍其详细细节的书籍，包括最著名的《C++GUI Programming with Qt4》，作者是 Jasmin Blanchette 和 Mark Summerfield，同样也是由 Prentice Hall 出版(还可以查阅他们关于 Qt3 的书)。

本书最后一章第9章“用C++开发跨平台GUI工具包”，从介绍Netscape和Mozilla浏览器套件里的一个重要的跨平台GUI工具包组件XPToolkit开始，详细介绍了我专门为本书创建的一个工具包——Trixul。Trixul具有很多和我们在Netscape用来开发GUI的Netscape/Mozilla XPToolkit相同的属性。比如，允许用XML和JavaScript来描述用户界面；都支持让用户界面调用C/C++写成的共享库的基于组件的模型；高度可移植等。Trixul和Mozilla提供的工具包有两个最大的不同。第一，Trixul是一个桌面的GUI工具包，而XPToolkit应用只在Web浏览器里执行。第二，Trixul的设计（我觉得）比XPToolkit要简单得多，（我确信）这让我在描述工具包的架构和背后的思想时更得心应手。虽然我没有真的希望你为自己的项目设计一个跨平台GUI工具包，但是从观察Trixul是如何设计和实现的过程中，还是有很多东西可以学习的。

本书的大多数章节都可以按照任意顺序阅读，如果你是技术经理，我推荐你仔细阅读下列各章：

- 第1章，“策略与管理”
- 第2章，“Build系统和Toolchain”
- 第3章，“软件配置管理”

技术经理还至少应该浏览以下章节：

- 第4章，“安装和部署”
- 第5章，“操作系统接口和库”
- 第6章，“其他可移植性主题”
- 第7章，“用户界面”

开发人员应该从头读到尾，不过你可以采用和技术经理相反的顺序，省略他们应该精读的部分，细读他们可以浏览的部分。如果你专注于用户界面的开发，我推荐阅读条款22、23，以及第8章，“wxWidgets”。如果你对GUI工具包的内部实现有兴趣，或者打算帮助开发Trixul（下一节会说到），那绝对不能错过第9章，“用C++开发跨平台GUI工具包”。

关于Trixul

Trixul是我专门为了撰写本书所做的一个开源项目。在某种程度上，我有和Gtk+的原作者相同的想法，边做边学。但是，Trixul背后更主要的目的是要开发一个简单的、跨平台的工具包，其架构和设计可以简单到在100页内即可描述，让大多数人不用阅读大段的代码就能理解。Trixul设计的灵感很大程度上来自Netscape/Mozilla（文档对象模型DOM，Gecko的布局引擎，XUL，XPConnect和XPCOM等Netscape/Mozilla的技术都在Trixul里有相似的实现）。并非所有人都要编写他们自己的GUI工具包，但是Netscape需要，AOL也需要（在Netscape被收购后开发的一个不那么可移植的玩意，名为Boxely），或许你的公司也会用得着。如果不讨论用户界面问题是怎么解决的话，Mozilla/Netscape可移植性的故事是不完整的。而我觉得用合理的篇幅来解释Trixul是最好的办法。

但是，Trixul不仅仅是以教学为目的的。我真诚地希望Trixul能发展成为下一代的桌面GUI工具包。在撰写本书的时候，这个项目正处于襁褓之中。如果你喜欢Trixul并且有兴趣参与开发或是移植工作，我是非常愿意接受的。你可以访问www.trixul.com或者http://sourceforge.net/projects/trixul来了解更多。

参考文献

下面是一个在本书中直接提到的，或者是受其影响的简短的书目列表。

Andrei Alexandrescu, *Modern C++Design: Generic Programming and Design Patterns Applied*（Reading, MA: Addison-Wesley, 2001）.

Jasmine Blanchette and Mark Summerfield, C++*GUI Programming with Qt*3 (Upper Saddle River, NJ: Prentice Hall, 2004).

Randal E. Bryant and David O'Hallaron, *Computer Systems A Programmer's Perspective* (Upper Saddle River, NJ: Prentice Hall, 2003).

David R. Butenhof, *Programming with POSIX Threads* (Upper Saddle River, NJ: Prentice Hall, 1997).

Paul Dubois, *Software Portability with imake* (Sebastopol, CA: O'Reilly Media, Inc., 1996).

Erich Gamma, et al., *Design Patterns* (Reading, MA: Addison-Wesley, 1995).

Simson Garfinkel and Michael K. Mahoney, *Building Cocoa Applications: A Step-by-Step Guide* (Sebastopol, CA: O'Reilly Media, Inc., 2002).

Ian Griffiths, et al., *.NET Windows Forms in a Nutshell* (Sebastopol, CA: O'Reilly Media, Inc., 2003).

Greg Lehey, *Porting UNIX Software* (Sebastopol, CA: O'Reilly Media, Inc., 1995).

Syd Logan, *Developing Gtk + Applications in C* (Upper Saddle River, NJ: Prentice Hall, 2001).

Scott Meyers, *Effective* C++ (Reading, MA: Addison-Wesley, 2005).

Andrew Oram and Steve Talbot, *Managing Projects with make* (Sebastopol, CA: O'Reilly Media, Inc., 1993).

Eric S. Raymond, *The Art of UNIX Programming* (Reading, MA: Addison-Wesley, 2003).

Julian Smart, et al., *Cross-Platform CUI Programming with wx Widgets* (Upper Saddle River, NJ: Premtice Hall, 2006).

Bjarne Stroustrup, *The C++ Programming Language* (Reading, MA: Addison-Wesley, 2000).

致谢

有人可能会觉得写书是一件很孤立的事情。但实际上，一本书从构思到完成需要很多人的才智、时间和贡献。所以在此我要感谢以下帮助我完成此书的人。

首先，我要感谢 Prentice Hall 的 Greg Doench 对我的信任，让我有机会再次把我的想法写出来。以及感谢他的助手 Michelle Housley 的大力协助。我还要感谢 Pearson Education 的 Keith Kline、Anne Goebel 和 Gina Kanouse，他们做了大量的编辑工作让我的手稿最终变成了书。我特别要感谢 Anne 在最后的校阅中不厌其烦地和我一起做了大量的“小修小补”。这些无疑大大增加了她的工作量（更不要提收到每封都标着“最终修改”的 email 的痛苦了）。无论如何，她都仔细考虑并执行我的要求，毫无怨言。就凭这一点，我还要再说一声“谢谢”。按照惯例，所有未发现的错误都是由我负责。（请访问 www. crossplatformbook. com 来获取本书的勘误。）

接下来，我要感谢本书的读者，他们大多是和我一起在 Netscape 工作过的同事。特别地，我要感谢现在在 AOL 工作的 Sean Su，他阅读了条款 15 的草稿并给了我他的反馈信息。我在 Netscape 的第一位经理，Jim Dunn，帮助我了解了 Netscape 4. x 浏览器开发的幕后工作。同样是来自 Netscape 的 Doug Turner，现在为 Mozilla 工作的他向我提供了大量建设性的反馈意见，大大丰富了我的原稿。曾经是 Netscape 资深国际化工程师的 Roy Yokohama，给我在 wxWidgets 国际化和本地化的部分提供了许多非常有用的建议。其他提供了有用的反馈的还有 Netscape 的 Stephen Morse 和 Namachivayam Thirumazshusai（Shiva）。我同样还要感谢 Kevin McCluskey 为本书作序。当年我们一起在圣地亚哥为 Netscape 工作时，他教了我很多东西。

感谢我所有的朋友，你们帮助了我在成书的日子里能保持清醒的头脑。感谢 Mei，在项目完成最后的阶段里，给我了时间、空间和我最需要的鼓励。

引　言

一个典型的可移植性的定义是这样的：

可移植性是指让一个软件在超过一个操作系统或平台上成功运行的难易程度。

不过，这个定义只是一个开始。真正要定义可移植性，只考虑在其他操作系统上运行是不够的。比如下面的 C++代码：

```
#include <iostream>

using namespace std;

int
main( int argc,char * argv[])
{
    cout < < "Hello\nWorld";
}
```

大多数读者都会同意以上代码基本上可以在任何有优秀 C++编译器支持的平台上编译执行。

然而即使这样简单的代码还是潜伏着问题的。试着在 Windows 命令行模式下运行这个程序(假设这段代码编译后的名字是 foo. exe)，把输入重定向到一个文本文件里去：

```
c:\ > foo > foo. txt
```

然后用 wc(1)命令查看结果有多少个字符：

```
c:\ > wc -c foo. txt
12 foo. txt
```

接下来在 Mac OS X 上再做一遍：

```
% foo > foo. txt
% wc -c foo. txt
11 foo. txt
```

注意到什么了吗？在 Windows 上 wc 表明文件有 12 个字符，而在 Mac OS X 上只有 11 个。现在我们用 od(1)来看看到底发生了什么。首先，在 Windows 上运行 od(1)：

```
C:\ > od -c foo. txt
0000  H  e  l  l  o  \r  \n  W  o  r  l  d
0014
```

然后是 Mac OS X：

```
% od -c foo. txt
0000  H  e  l  l  o  \n  W  o  r  l  d
0013
```

这里的区别是 Windows 把我们字符串里的 \ n 转换成了 DOS 下的行终结符 \ r \ n，而在 Mac OS X 上则保持原样。

在大多数情况下，这个差异是无关紧要的。毕竟两个平台在终端上看到的输出是一样的。如果只是

以这个为标准的话，你完全可以认为这段代码是可移植的。不过要是输出的大小是有关系的呢？比如，如果这段代码是系统的一部分，另一个系统上的某些进程要根据其文件内容的大小来分配缓存的话，怎么办？要么(Windows 计算数据大小，Mac OS 进行分配)浪费一个字节的空间，因为我们把 11 个字节的数据量写进了 12 个字节的缓存里。要么(在 Mac OS X 上计算结果，在 Windows 上分配缓存)我们就要面对缓冲区溢出一个字节的未定义行为。

所以如果使用不当，就算是“Hello World”这样的老古董也会产生移植性问题。现在让我们来看看能影响编写可移植软件的主要因素。

影响软件可移植性的方方面面

语言

虽然本书主要涵盖了 C++的内容，不过由于 C 的近亲因素，C 也会列入讨论范围。C 从 20 世纪 70 年代末期首次在 K&R 的书里介绍时就被认为是可移植性良好的语言。它闻名于世的一个主要原因是 UNIX 系统发现它能在那么多不同的硬件平台上运行，而这是因为操作系统的绝大部分是用 C 写成的。而标准化的努力(特别是 ANSI 和最近的 C99)更让 C 成为了一个可移植性很强的编程语言。遵循 ANSI 标准，避免使用编译器开发商的语言扩展是消除 C 可移植性问题的重要步骤。你可以通过(命令行标记或相关设置)命令编译器只接受基于标准的代码，拒绝所有的编译器开发商提供的语言扩展来提升可移植性。这项建议对 C++同样有效。

编译器

另一个和 C/C++语言的可移植性紧密相关的当然就是负责将源码变成可执行形式的编译器了。我前面提过编译器可用于控制代码遵循标准的程度，但是编译器的作用不止于此。本书提及了很多目前最流行的编译器，如在 Windows 平台上的微软 Visual C++6.0 和 Visual C++7.0 .NET。而在很多平台上都有的开源编译器 GNU 的 GCC 则支持 Mac OS X、Linux 和通过 Cygwin 项目支持 Windows。

由于 C/C++语言的定义，许多语言特性的实现细节都留给了编译器开发商自行处理。结果造成了使用这些特性会引入不可移植性。这些特性包括：

- 内建类型 short、int 和 long 的长度

根据定义，这些类型的长度是与编译器相关的。C 标准规定 short 类型必须至少为 16 位；而 int 类型必须至少和 short 类型一样大；最后 long 类型必须至少和 int 类型一样大。这样一来，在 32 位的机器上可以是 16 位的 short 类型，32 位的 int 类型和 32 位的 long 类型；也可以是 32 位的 short 类型和 int 类型，64 位的 long 类型。当然还有其他符合标准的组合。一般来说，典型的 32 位机器会支持 32 位 int 类型，64 位 int 类型则出现在 64 位机器上，因为 int 类型一般就是定义为一个原生字长。但就是这一点也是没有保证的。本书中我介绍的 Netscape 可移植运行库(Netscape Portable Runtime Library，NSPR)专门为这个问题提供了解决方案。

- 位操作符

假设所操作的 short 类型、int 类型和 long 类型的长度会引入错误的结果。我再次强调，这些类型的长度是由编译器决定的。向右位移一个值可以是向右传播符号位，也可以是最左位填零，当然这个操作的实现也是取决于编译器。

- 有符号和无符号的 char 类型

C 和 C++都不指定一个 char 型是不是有符号的。这个是由编译器的作者自行决定的。当代码里的 char 类型和 int 类型混在一起的时候就有可能产生问题，典型的例子是在 C 里用 getchar( )函数从标准输入(stdin)读入一串字符到一个无符号的 char 变量里。getchar( )的返回值是一个 int 值，通常 -1 代表读

到了文件结尾。如果你在循环里比较 -1 和那个无符号的 char 值，那无论 getchar( ) 是不是遇到了 EOF，这个循环永远都不会结束。要避免这个问题，可以如下显式地声明字符类型为有符号的：

```
signed char foo; // signed,range -128 to 127;
unsigned char fee; // unsigned,range 0 to 255;
```

只在同类型之间进行赋值和比较（比如只比较 char 类型和 char 类型的变量）；一律使用 C++风格的类型转换；遵循函数原型（例如不要把 getchar( ) 的返回值赋值给不是 int 的其他类型）；修复每一个编译器产生的警告——这些都是克服这类移植性问题的方法。

二进制数据

除了多位整数的存储方式（位和字节在内存中的存储顺序）外，二进制数据还得面对编译器所选择的 struct 在内存里的布局问题。通常这是完全取决于编译器和目标架构的。不同的编译器所生成出来的代码里的 struct 在二进制下（写入内存或磁盘的），或者在内存里的组织方式可以是完全不同的。要避免这个问题的最好办法就是避免读写二进制数据，而代之以文本方式。当然这个办法不总是可行的，所以我在条款 20 会详细讨论如何处理二进制数据。

标准库

标准库（及其头文件）扩充了 C/C++语言核心的能力。而可移植性正是开发一个标准库的主要动机之一。C 标准库包含了熟悉的 < stdio. h > 、< ctype. h > 、< string. h > 和其他一些头文件。其中像 strncmp( ) 、getchar( ) 、printf( ) 、malloc( ) 、fopen( ) 等无数的函数和宏几乎出现在每一个 C 程序之中。不仅因为标准库给普通程序员所带来的巨大价值，还因为每个 C 的实现都支持标准库，使用标准库所带的函数、常量和宏能大大提升你的 C 代码被成功移植到另一个编译器上的机会。

在 C++的世界里，标准模板库（Standard Template Library，STL）则正式定义为标准 C++语言的一部分。STL 保留了 C 标准库（不过头文件都加上了 c 前缀，并且去掉了 . h 后缀，如 < cstdio > 、< cctype > 和 < cstring > ），并扩充了它的功能使其和 C++在很多方面完美合作。STL 不仅提供了对输入/输出的支持，还设计一整套有标准保证性能的容器类。大体上，我推荐你尽可能地使用 STL 提供的类似功能而不是其他（包括你自己写的代码）。当别人在 STL 里已经为你提供了一个最优化的实现后，再绞尽脑汁去实现一个高效的链表是完全没有意义的。而且使用 STL 还可以使你的代码更容易移植。这建议也适用于标准库的其他部分。只要可能就尽量地学习使用标准库。

和 STL 相关的还有一个开源的 Boost 项目（www. boost. org），它致力于填补 STL 的空白。它所做的很多工作可能最终都会成为 STL 的一部分。

操作系统接口

核心语言和标准库里提供的操作系统接口（也叫系统调用，system call）让应用程序有能力进行系统相关的任务。标准库里有很多属于这个类别的函数，如创建进程、进程间通信（interprocess communication，IPC）、底层输入输出、设备驱动接口和网络输入输出等。可以想象，这些功能都是高度系统相关的。

现在用一个在 UNIX 和 Win32 上创建进程函数的例子来展示系统调用的实现可以有多大的差别。要在 Win32 创建进程，使用的是 CreateProcess( ) 函数：

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName,
  LPTSTR lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
```

```
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);
```

举例来说，如果要运行 Notepad. exe 记事本程序，可以在 Windows 下编写执行以下代码：

```
#include <windows.h>

STARTUPINFO si;
PROCESS_INFORMATION pi;

ZeroMemory( &si,sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi,sizeof(pi) );

// Start the child process.
if(! CreateProcess(NULL,"notepad",NULL,NULL,
    FALSE,0,NULL,NULL,&si, &pi))
        return E_FAIL;
else
    return S_OK;
```

在 UNIX 下，则使用 fork(2)和 exec(3)函数族来创建执行进程。创建进程的函数 fork(2)的原型是：

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

命令进程载入并执行一个特定程序的函数 execl(3)的原型是：

```
#include <unistd.h>

extern char * * environ;
int execl(const char * path,const char * arg,...);
```

下面的程序展示了如何使用 fork 和 exec 来运行 UNIX 的 date(1)程序。fork(2)的返回值决定哪个进程是被创建的(子进程)以及哪个进程是调用 fork(2)的(父进程)。这里，子进程通过调用 execl(3)来执行 date 程序，父进程则简单退出：

```
#include <sys/types.h>
#include <unistd.h>

int
main(int argc,char * argv[])
{
    pid_t pid;

    pid = fork();
    if (pid = = 0) {
        /* child process */
        execl("/bin/date","date",NULL);
    }
```

```
    return(0);
}
```

从这段源代码看上去，进程的创建是可移植的，可惜事实并非如此。在条款 16 和 17 中，我会讨论如何使用标准接口和 NSPR 那样的移植库来把不可移植的原生系统功能变得可移植。这些条款覆盖的标准接口包括了 IEEE 1003 标准操作系统接口(standard interface for operating system)POSIX、System V Interface Description(SVID)和所有自称“UNIX”的系统都支持的 X/Open Portability Guide(XPG)。GCC 编译器家族在本书所论及的平台上支持上述所有的三个标准。所以如果你决定要在 Windows 上用 Cygwin 和 GCC，基本上没什么问题。如果你要用微软或其他厂商提供的编译器和支持库的话，要获得纯源代码兼容还需要一些其他的策略。NSPR(在条款 17 里讨论)和 Boost 都可以在这里派上用场。还有一个办法就是在相应的 Win32 接口上构建一套你自己的抽象层，我会在条款 22 里讨论如何开发这个抽象层。

用户界面

用户界面可能是现代桌面平台(应用层)上移植性最差的部分了。本书会花费大量篇幅来讨论怎样克服这个限制来编写高度可移植的软件。

每个平台都有它自己的用户界面工具包来支持原生的图形用户界面(GUI)开发。在 Windows 平台上有 Win32、MFC 和终会取代前二者的 . Net API。在 Mac OS X 上有基于 Object-C 的 Cocoa 框架和被设计成与旧有的 Mac OS 工具箱相似的，用来简化从 Classic Mac OS 7/8/9 向 Mac OS X 移植的 Carbon API。而在 Linux 上，选择的范围非常大，从 Gtk + (GNOME)、Qt(KDE)到其他诸如 Xt/Motif(CDE)等。它们都是基于 X Window 系统的。

这些工具包的源代码互相都不兼容，看上去和用起来感觉也都不一样。前面提到的两个工具包 Qt 和 Gtk + 也有 Windows 版，但是很少用到，因为微软提供的工具绝对统治了 Windows 平台。

关于原生 GUI 工具包和 API 的编程可以有很多要说的东西。你的应用程序运用原生 API 的时候可以优雅地和本地环境集成。用户会发现你的应用程序易于学习和使用，你的应用程序相比使用非原生工具包也更容易无缝地与其他桌面应用程序交流。如果不用考虑移植性，通常使用原生 API 是正确的方法。但是，你阅读本书的原因就是因为移植性是重要的考量，所以我们需要找到办法来最大化代码移植性的同时尽量减少不用原生工具包而带来的负面影响。本书的第 7、8、9 章会探索解决这个问题的办法。

编译系统

编译系统可以简单到一个执行编译器和连接器的命令行脚本，也可以复杂到基于 automake 和 Imake 的庞大系统。使用一个标准的可共享的编译系统的关键原因是开发人员可以在不同机器之间轻易地移动。举例来说，如果你是一个 Win32 的程序员，你可以轻易地移到 OS X 上执行编译你的修改，因为你使用的工具基本上是一样的。我在本书里描述的 Imake 主要关注如何支持代码的移植性。流行于开源社区的 automake 和 autoconf，提供了另一套处理跨平台 Makefile 生成的策略，则在别处有详细的文档。集成开发环境(Integrated development environment，IDE)如微软的 Visual Studio. NET，或是 Apple 的 Interface Builder 和 Project Builder 等则束缚了可移植的编译系统开发，更不要说可移植的代码了。条款 8 讨论了 IDE 在跨平台开发中的位置。

配置管理

想象一下你是项目里惟一的一个开发人员，只有二十几个源文件，且全部都在硬盘上的一个目录下。你负责编辑、编译和测试它们。定时将源目录用 zip 或 gzip 打包存到别的地方以防止灾难发生(误删文件或是硬件损坏)。为讨论方便，假设每个星期做一次这样的备份。

现在假设你发布了软件的第一个版本，从其他人那里得到了反馈，花费了几个星期来修复 bug 和添加一些被请求的特性。发布一个新版本后得到了一个坏消息，一个原来正常的特性现在不工作了。显然在第一版之后的某些修改让这个原本正常的特性不工作了。你应该怎么找到是什么修改导致了这个问题？一个办法是从每个星期的备份开始，把 bug 定位某个星期的工作上。通过比较那个 build 里的源代码来确定到底做了什么修改，来逐渐推断出是哪里出了错。

但是我们有更好的办法，可以比上面的方案解决更多的问题。这就是配置管理。一个配置管理系统对一个就算只有一个开发人员的项目来说依然是很有益的。而在多人开发的项目里，配置管理会变得非常关键，因为当超过一个开发人员向同一个代码库里提交代码时，不单现有的问题会被放大，还会引入新的问题。可能最值得注意的就是如何有效地把多人所做的修改合并到一个代码库里而不引入新错误的过程。

市面上有很多这样的配置方案，大多数都是平台相关的，不过目前最出色的是两个开源软件，并发版本系统(Concurrent Version System，CVS)和 Subversion。除此之外，它们在几乎所有值得考虑的平台上都同时有命令行和 GUI 方式。CVS 是开源社区的标准源码控制系统，在企业领域也十分流行。我会在条款 13 介绍 CVS，让你了解有效使用它所需要知道的一切。条款 14 则介绍了一个在多人协作的项目里提升配置系统作用的相关工具：patch(1)。

抽象的作用

抽象是本书所展示的很多技巧和技术的中心主旨。如果不是在我们生活的这个世界里的话，无法理解抽象对于工程学的重要性。所谓的抽象是指处理事物时不需要关心它的具体细节。如果没有抽象，那再简单的东西都会变得异常复杂，让人迷失在琐碎的细节之中。拿下面一段 PowerPC 的二进制码来做例子：

```
0011540 0000 0000 7c3a 0b78 3821 fffc 5421 0034
0011560 3800 0000 9001 0000 9421 ffc0 807a 0000
0011600 389a 0004 3b63 0001 577b 103a 7ca4 da14
```

你知不知道这段代码是干嘛的？(要是你看懂了，真是太落伍)虽然很难想象直接敲入对应机器指令的数字，但事实上人们曾经真的写过这么底层的代码。这样的编程方式居然能让计算机做有用的事情是非常令人惊叹的。(当然相比现在对计算机的需求来说，在那个机器语言的时代，人们要求计算机做的事情并不多。)

就算是硬件高度发达的今天，如果还是让程序员在那么低级的抽象层上编码的话，程序员的数量将仍然很稀少，他们的程序的功能也还会是相当匮乏的。像上面的这种程序非常容易出错，任何人想要搞清楚它的作用并试图维护这段二进制代码几乎是不可能的。

以二进制编程的困难最终由于抽象的引入而被克服，这就是：汇编语言。

下列的代码是以 PowerPC 汇编写成，是以上二进制代码的一个抽象：

```
_F:
  mflr r0
    stmw r29, -12(r1)
  stw r0,8(r1)
  stwu r1, -80(r1)
  mr r30,r1
  stw r3,104(r30)
  lwz r0,104(r30)
  cmpwi cr7,r0,0
  bne cr7,L8
```

```
    li r0,0
    stw r0,56(r30)
    b L10
L8:
    lwz r0,104(r30)
    cmpwi cr7,r0,1
    bne cr7,L11
    li r0,1
    stw r0,56(r30)
    b L10
L11:
    lwz r2,104(r30)
    addi r0,r2,-1
    mr r3,r0
    bl _F
    mr r29,r3
    lwz r2,104(r30)
    addi r0,r2,-2
    mr r3,r0
    bl _F
    mr r0,r3
    add r29,r29,r0
    stw r29,56(r30)
L10:
    lwz r0,56(r30)
    mr r3,r0
    lwz r1,0(r1)
    lwz r0,8(r1)
    mtlr r0
    lmw r29,-12(r1)
    blr
```

但是这样程度的抽象依然远远不够。花一点时间阅读这段代码，看看你能不能看出来这段程序生成的是哪一段著名的数列。不太简单吧？就算这是我的代码(实际上是GNU C编译器从我的C代码上生成的汇编码)，我也觉得很难看懂这段汇编码是干嘛的，或者把它映射回原来的C代码也非常困难。

事实上虽然汇编码相对机器码来说是一大进步，但是对抽象来说还远远不够。对大多数人来说，它和机器码其实没什么两样。最好不要在这个层面上编写应用程序。不过当年汇编语言出现的时候，确实产生了许多有意义的应用程序，而且使代码的调试和维护都变得简单很多，只是这些都是很小的进步。

很显然，汇编语言并不是故事的结尾。现在来看看进一步的抽象，友好的C语言：

```
int F(const int n)
{
  if (! n ||n = = 1)
      return n;
  return (F(n - 1) + F(n - 2));
}
```

C语言第一次给了我们编写易于理解的、可维护的代码的希望。这里的F( )函数是一个产生整数的递归函数。结果是一个斐波那契数，它的递推关系如下：

$$f(X)=\begin{cases}0 & \text{if } X=0\\ 1 & \text{if } X=1\\ f(X-1)+f(X-2) & \text{if } X>1\end{cases}$$