HZ BOOKS

PEARSON Addison Wesley

Modern C++ Design C++ 设计新思维

Exceptional C++中文版

47个C++工程难题、编程问题和解决方案

Exceptional C++: *47 Engineering Puzzles, Programming Problems, and Solutions*

（美） Herb Sutter 著
聂雪军 译

机械工业出版社
China Machine Press

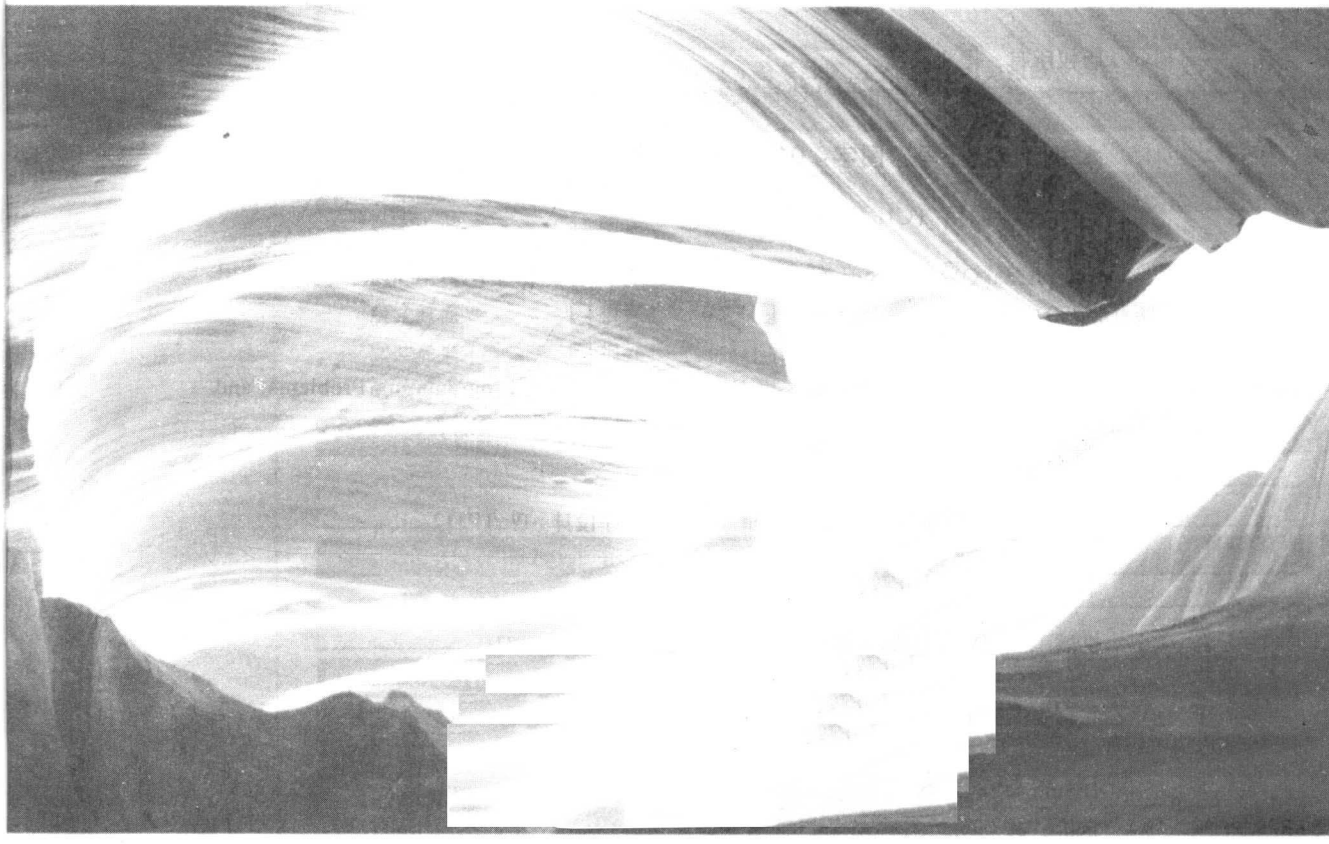
C++ 设计新思维

Exceptional C++中文版

47个C++工程难题、编程问题和解决方案

Exceptional C++: *47 Engineering Puzzles, Programming Problems, and Solutions*

（美） Herb Sutter 著
聂雪军 译



机械工业出版社
China Machine Press

本书通过示例的方式来讲述如何用标准C++进行正确的软件开发。全书共分8章，包括范型程序设计与C++标准库、异常安全性的问题与技术、类的设计与继承、编译器防火墙和Pimpl惯用法、名字查找名字空间和接口规则、内存管理、误区陷阱以及错误的惯用法等。本书内容深入，论证严谨，作者权威，可帮助读者编写效率更高、更加健壮的C++代码。

本书适合有一定编程经验的C++程序员阅读，也可作为提高C++编程水平的参考书籍。

本书版权登记号：图字：01-2006-3143

图书在版编目（CIP）数据

Exceptional C++中文版（美）萨特（Sutter, H.）著；聂雪军译. －北京：机械工业出版社，2007.1

（C++设计新思维）

书名原文：Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions

ISBN 7-111-20258-9

Ⅰ.E… Ⅱ.①萨… ②聂… Ⅲ.C语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2006）第127767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1月第1版第1次印刷

186mm×240mm · 12.5印张

定价：29.00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

这是一本大师编写的C++书籍。

能够成为一名大师，是每个C++程序员的梦想。在经过了几年的实践之后，许多程序员都能在日常工作中很好地应用C++来解决编程问题和实现功能模块。然而，在处理一些较为复杂的问题时，却往往还是会感到有些力不从心。例如，我们很难编写出非常稳定的代码；在设计大型系统时，总会有一些不完善的地方。有时候虽然能够从直觉上感觉到一些东西，但却始终无法找到问题的关键所在，也很难确定该从哪些方面来入手。究其原因，这是因为虽然我们已经掌握了C++，但对于一些深层次的细节，我们还没有进行深入和细致的研究。对于上述这些问题，本书将会给出很好的解答。

本书主要有以下的特点：

(1) 深入性。这是一本颇具深度的书籍。本书对C++的分析深度是很多C++书籍所不能达到的。对于书中的每个主题，作者首先从理论上进行了严谨的论证，然后再给出实际的代码进行全面和细致的分析，做到了详尽无遗。对于每一种应用，作者都非常清楚地解释了为什么要这么做，以及这种做法的使用环境。如果你已经具备了一定的C++基础，那么本书的内容能够对你的C++编程能力产生质的提升。

(2) 拓展性。C++中的一些特性，例如异常安全性，对于编写高质量的程序来说，是非常必要的。但遗憾的是，到目前为止，了解并使用这些特性的程序员还是不多。大多数程序员都还是停留在一些普通的C++特性上，而很少会去研究C++的高级特性。本书对一些C++的高级特性进行了详细的阐述和分析，特别指出了在哪些情况下，这些特性的优势能够得到最大的发挥，以及这些特性能够为我们带来怎样的好处。通过对这些高级特性的学习，我们不仅能够从更全面的角度来看待问题，而且能够拓展新的设计思路。

(3) 纠错性。在学习C++的过程中，我们免不了会犯一些错误，例如对继承的过度使用，好像“没有大量继承的OO就不是OO，而我们自己是很难发现这些错误的。久而久之，我们会不知不觉地陷入到一些思维陷阱中，并因此养成不好的编程习惯。本书的一大特点就是，在分析问题的同时，还指出了一些不好的编程习惯和常见的错误，并且解释了为什么这些做法是不好的或错误的。在阅读了这部分内容之后，你肯定会发现其中有些错误是你原来没有觉察到的，并且在今后的工作中，你也会有意识地去改正这些错误。

对于本书的翻译工作，译者投入了大量的时间和精力，工作到凌晨是常有的事情。在翻译的过程中，对于每一个难以理解的地方，译者首先保证自己能够充分地理解，然后再用浅显的语言表达出来，从而帮助读者能够更快、更准确地理解书中的内容。参加本书翻译工作的还有李杨、吴汉平、徐光景、童胜汉、陈军、胡凯、李斌、张玮、陈红和刘红。

由于译者的时间和水平有限，翻译中的疏漏和错误在所难免，还望读者和同行不吝指正。

书山有路勤为径。只有让自己不断地学习，不断地提高，成为大师的梦想才有可能实现。祝愿所有的C++程序员都能实现自己的梦想。

致谢

首先要感谢陈冀康编辑对我的信任和耐心，使得我能够顺利地完成本书的翻译工作。感谢李南丰编辑对本书的审阅，帮助我改正了译稿中的一些错误。在翻译本书的时候，妻子云兰正在准备迎接我们的第一个孩子。感谢你们，你们是我所有的动力。感谢我的父母，你们一直都在默默地支持着我。

聂雪军

2006年8月于北京

序　言

这是一本非凡的书，不过直至快要读完全书我才意识到它是多么不平凡。这可能是第一本写给已经熟悉C++——熟悉C++的一切——的人看的书。从语言特性到标准库组件再到编程技术，本书从一个主题跳到另一个主题，总是使你处于些微失衡的状态，总是确保你专心致志。就像现实C++程序一样，类设计撞上虚函数的行为，迭代器协定碰到名字查找规则，赋值操作符擦上异常安全，编译依赖遭遇导出模板。就像在现实程序中的一样，语言特性、库组件和编程技术形成的混乱的大漩涡，精彩而令人眩晕。

我将GotW（Guru of the Week）发音为“Gotcha（got you，抓到你了）”可能并无不妥。当我将自己针对书中测验给出的解决方案与Sutter的答案进行比较时，我往往会落入他（和C++）设置的陷阱中，次数之多，使我羞于承认。每当我犯错时，我几乎能看到Herb面带微笑轻声说道：“Gotcha！”也许有人争论这证明我对C++知之甚少，另有人可能宣称这说明C++太复杂，以至于任何人都很难精通它。我则认为这表明在使用C++进行工作时，你必须小心谨慎地思考你正在做的事情。C++是一门威力极大的语言，它被设计用于解决需求苛刻的问题，尽可能细致地磨练你在语言、库和编程惯用法方面的知识，至关重要。本书讨论主题范围之广，内容安排之独特（采用了基于测验的格式），对你的磨练过程将会助一臂之力。

C++新闻组的老读者都知道选出“Guru of the Week”有多么困难，那些有经验的参与者对此更是深有体会。尽管在网上每周只能产生一名guru，然而有了本书提供的知识撑腰，每当编程时你都可冀望产出有着guru质量的代码。

Scott Meyers

1999年6月

前　言

本书通过例子展示如何实施可靠的软件工程。本书包括因特网上流行的C++专题“Guru of the Week”（或简写为GotW）前30个议题的扩充版本，并补充了大量的其他材料。“Guru of the Week”包含一系列独立的C++工程问题和解决方案，描述了具体的设计和编程技术。

本书并非盛有代码谜题的摸彩袋，它首先是对现实世界的C++企业软件设计的指南。它使用了问题/解决方案的形式，因为这是我知道的将您——文雅的读者——引入问题背后的思想和指导方针背后的理由最有效的方式。尽管这些条款涵盖了形形色色的主题，然而你会注意到反复出现的主题集中于企业级开发议题上，尤其是异常安全、健全的类和模块设计、适当的优化以及编写遵从标准的可移植代码。

我希望本书这些内容对你的日常工作有用，我还希望你至少从中发现一些极好的思想和优雅的技术，并且在阅读本书的过程中有时突然喊出“啊哈，原来如此！”是谁说软件工程枯燥乏味的？

如何阅读本书

我期望你已经掌握了C++基础知识，如果你还没有，可以从一本介绍性和概览性的好书（像Bjarne Stroustrup的《The C++ Programming Language》（第3版）[⊖]或Stan Lippman和Josée Lajoie合著的《C++ Primer》（第3版）[⊜]这样的经典著作都是不错的选择）开始学习，然后选读一本编程风格指南，例如Scott Meyers的经典著作《Effective C++》（我发现基于浏览器的CD版方便且实用）[⊜]。

书中的每一个条款都以谜题或问题的形式呈现，并带有一个介绍性的头部，如下所示：

Item ##. 谜题的标题	Difficulty:x

标题和难度等级（通常从3～$9\frac{1}{2}$，最高为10）提示你将要遭遇到什么。注意，难度等级只是我自己预期大多数人认为问题有多难的主观猜测，因此，你也许会发现对你而言一个难度等级为7的问题比一个难度等级为5的问题更简单。话虽如此，当你看到一个难度为$9\frac{1}{2}$的怪

⊖ Stroustrup B. *The C++ Programming Language*，第3版（Addison Wesley Longman，1997）。

⊜ Lippman S. and Lajoie J. *C++ Primer*，第3版（Addison Wesley Longman，1998）。

⊜ Meyers S. *Effective C++ CD: 85 Specific Ways to Improve Your Programs and Designs*（Addison Wesley Longman，1999）。可访问http://www.meyerscd.awl.com获得在线演示版。

物浮出水面时，最好还是做最坏的思想准备。

你无需按顺序阅读每一个小节和问题，但有几个地方存在一些相关问题的“小型系列”，你可以看到它们被标以“之一”、“之二”等，有些地方甚至多至“之十”。最好按组阅读这些小型系列。

最后，关于URL有必要多说一句：在Web上，内容会经常变动。尤其是，我无法控制的一些内容会动来动去。这使得在印刷书籍上刊印随意的Web URL就变成了真正的痛苦：恐怕在该书下厂印刷之前那些URL就已经过时了，更不要说等它在你的书桌上躺上5年之后了。当我在本书中引用他人的文章或Web站点时，我是通过自己的Web站点（www.gotw.ca）上的URL做到这一点的——我自己的Web站点是我所能控制的，它只包含对实际Web网页的重定向链接。如果你发现印刷在本书中的一个链接不再有效，请写邮件告诉我，我将更新该链接，使其指向新的网页位置（如果我还能找到该网页的话），或者告诉你该网页已不复存在（如果我找不到的话）。不管怎么说，本书的URL将会保持为最新，尽管在这个因特网世界中印刷媒体的日子是如此难过。呜呼！

来龙去脉：GotW与PeerDirect

C++“Guru of the Week”系列已经由来已久。GotW最初创建于1996年底，为我们自己在PeerDirect的开发团队提供有趣的挑战和继续教育。我编写它是为了提供有趣味的学习工具，包括对继承和异常安全之类特性的正确用法的讲解。随着时间的推移，我还将它作为一种工具，用于向我们的团队介绍C++标准会议正在进行的改动。从那以后，GotW作为因特网新闻组comp.lang.c++.moderated的定期专栏对一般C++公众开放，在那儿你可以找到每一个新议题的问题和答案（以及大量有趣的讨论）。

在PeerDirect用好C++非常重要，这与在你的公司用好C++的重要性有很多相同的理由，尽管要达到的目标可能并不相同。我们构建用于分布式数据库和数据库复制的系统软件，在这些领域，诸如可靠性、安全性、可移植性、效率，以及其他很多企业级议题，都是生死攸关的考虑。我们编写的软件要能够移植到不同的编译器和操作系统上，当发生数据库事务死锁、通信中断以及程序异常时，它要保持安全、强健。顾客用它来管理位于智能卡和pop终端或PalmOS和WinCE设备上的微型数据库，或管理部门级Windows NT和Linux、Solaris服务器，甚至管理Web服务器和数据仓库的大规模并行Oracle后端。这些都使用同样的软件、同样的代码，对可靠性有着同样的要求。现在，当我们费力地工作于大量的密集而未加注释的代码之上时，就会面临可移植性和可靠性的挑战。

对于读者中过去几年来已经读过因特网上“Guru of the Week”的人，我有两句话要说：

- 感谢你们的关心、支持、电子邮件、赞扬、纠正、评论、批评、提问，特别感谢你们对GotW系列印行成书的要求。它来了，希望你们喜欢。
- 本书包含的内容比你在网上看过的多得多。

本书并不只是对已经浮现于网络空间的陈旧的GotW议题的剪贴。所有问题和解决方案

都已在相当大程度上修订和重写，例如，条款8到条款17讨论的异常安全起先出现于单个GotW谜题中，现在则变成了一个具有10个部分的小型深入系列。每一个问题和解决方案都已经被检视，使其跟上修改后的正式C++标准。

所以，如果你以前是GotW的定期读者，本书中仍然有大量的新东西。再一次对所有忠实的读者表示感谢，希望这份材料能够帮助你继续磨炼、扩展你的软件工程和C++编程技能。

致谢

首先当然要感谢comp.lang.c++.moderated 上的所有GotW的读者和热爱者，尤其是那些参加为本书起名竞赛的胜出者。有两位对引导我们定出最终书名帮助尤巨，我要特别对他们表示感谢：Marco Dalla Gasperina建议取名“Enlightened C++”，Rob Stewart则建议取名“Practical C++ Problems and Solutions”。鉴于这里反复强调异常安全，更进一步加入双关语“exceptional”是自然而然的。

非常感谢丛书编辑Bjarne Stroustrup，感谢Marina Lang、Debbie Lafferty以及Addison Wesley Longman的其余编辑人员，感谢他们对此项目投入持续的热情以及在1998年Santa Cruz C++标准会议上给予的周到的招待。

我还要感谢很多审稿人（其中不少人是C++标准委员会的成员），他们提供了深刻而尖锐的评论，对改善你将要看到的正文内容很有帮助。特别感谢Bjarne Stroustrup和Scott Meyers，以及Andrei Alexandrescu、Steve Clamage、Steve Dewhurst、Cay Horstmann、Jim Hyslop、Brendan Kehoe、Dennis Mancl，感谢他们宝贵的洞察力和评论。

最后，特别感谢我的家人和朋友，感谢你们一直以各种方式陪伴着我。

Herb Sutter

1999年6月

目　录

"C++设计新思维"丛书前言
译者序
序言
前言

第1章　泛型程序设计与C++标准库 ……1
条款1：迭代器 ……1
条款2：大小写不敏感的字符串——之一 ……3
条款3：大小写不敏感的字符串——之二 ……7
条款4：可重用性最大的泛型容器——之一 …9
条款5：可重用性最大的泛型容器——之二 …9
条款6：临时对象 ……16
条款7：标准库的使用（再论临时对象）……21
第2章　异常安全性的问题与技术 ……23
条款8：编写异常安全的代码——之一 ……23
条款9：编写异常安全的代码——之二 ……27
条款10：编写异常安全的代码——之三 ……30
条款11：编写异常安全的代码——之四 ……34
条款12：编写异常安全的代码——之五 ……36
条款13：编写异常安全的代码——之六 ……41
条款14：编写异常安全的代码——之七 ……46
条款15：编写异常安全的代码——之八 ……48
条款16：编写异常安全的代码——之九 ……50
条款17：编写异常安全的代码——之十 ……54
条款18：代码的复杂性——之一 ……55
条款19：代码的复杂性——之二 ……58
第3章　类的设计与继承……63
条款20：类的编写技巧 ……63
条款21：对虚函数进行重载 ……68
条款22：类之间的关系——之一 ……73
条款23：类之间的关系——之二 ……76
条款24：继承的使用和误用 ……81
条款25：面向对象程序设计 ……89
第4章　编译器防火墙和Pimpl惯用法 ……91
条款26：将编译期依赖性降到最低——之一 ……91
条款27：将编译期依赖性降到最低——之二 ……93
条款28：将编译期依赖性降到最低——之三 ……97
条款29：编译防火墙 ……99
条款30：Fast Pimpl惯用法 ……102
第5章　名字查找、名字空间和接口规则 ……109
条款31：名字查找与接口规则——之一 …109
条款32：名字查找与接口规则——之二……111
条款33：名字查找和接口规则——之三 …119
条款34：名字查找与接口规则——之四 …122
第6章　内存管理 ……129
条款35：内存管理——之一 ……129
条款36：内存管理——之二 ……131
条款37：auto_ptr ……136
第7章　误区、陷阱以及错误的惯用法 …147
条款38：对象标识 ……147
条款39：自动转换 ……149
条款40：对象的生存期——之一 ……151
条款41：对象的生存期——之二 ……153
第8章　其他主题 ……161
条款42：变量的初始化 ……161
条款43：正确地使用const ……162
条款44：类型转换 ……169
条款45：bool ……174
条款46：转调函数 ……177
条款47：控制流程 ……179
参考文献 ……186
后记 ……187

第 1 章　泛型程序设计与C++标准库

在本书的开始部分，我们首先来看一些泛型程序设计领域中具有代表性的问题。这些问题主要是关于如何有效地使用模板、迭代器和算法，以及如何对标准库进行使用和扩展。在我们解决了这些问题并且掌握了其中的思想后，就会很自然地引出后面章节的内容。在这些章节中，我们将分析在编写异常安全模板（exception-safe template）时的异常安全性（exception safety）。

条款1：迭代器　　难度系数：7

每个程序员在使用标准库的时候，都必须知道下面这些关于迭代器的常见的以及不常见的错误用法。在这些错误中，你能找出几个？

在下面的程序中，至少有四个与迭代器相关的问题。你能找出几个？

```
int main()
{
  vector<Date> e;
  copy( istream_iterator<Date>( cin ),
        istream_iterator<Date>(),
        back_inserter( e ) );
  vector<Date>::iterator first =
        find( e.begin(), e.end(), "01/01/95" );
  vector<Date>::iterator last =
        find( e.begin(), e.end(), "12/31/95" );
  *last = "12/30/95";
  copy( first,
        last,
        ostream_iterator<Date>( cout, "\n" ) );
  e.insert( --e.end(), TodaysDate() );
  copy( first,
        last,
        ostream_iterator<Date>( cout, "\n" ) );
}
```

解答

```
int main()
{
  vector<Date> e;
  copy( istream_iterator<Date>( cin ),
        istream_iterator<Date>(),
        back_inserter( e ) );
```

到目前为止，程序都还是正确的。Date类的设计者定义了一个提取函数，其形式为

operator>>(istream&, Date&)，这也是istream_iterator <Date>用来从cin流中读取Date对象的函数。算法copy()只是将每个Date对象填充到vector中。

```
vector<Date>::iterator first =
      find( e.begin(), e.end(), "01/01/95" );
vector<Date>::iterator last =
      find( e.begin(), e.end(), "12/31/95" );
*last = "12/30/95";
```

错误：上面代码中的最后一行可能是非法的，因为last的值可能等于e.end()，而这个值是不能被解引用的（dereferenceable）。

算法find()的工作流程是：如果在指定的范围内没有找到所要查找的值，find()将返回它的第二个参数（也就是范围的终止迭代器）。在本例中，如果在容器e中并没有包含“12/31/95”，那么last就将等于e.end()，这个值指向的是容器中最后一个元素的下一个位置，因此并不是一个有效的迭代器。

```
copy( first,
      last,
      ostream_iterator<Date>( cout, "\n" ) );
```

错误：这行代码也可能是非法的，因为[first,last)可能并不是一个有效的范围；事实上，first有可能在last之后。

举例来说，如果在容器e中并没有包含“01/01/95”，但却包含了“12/31/95”，那么迭代器last将指向容器中等于“12/31/95”的Date对象，而此时的迭代器first指向的是容器e中最后一个元素的下一位置，这样last就位于first之前。然而，copy()要求first所指向的对象必须位于last指向的对象之前——也就是说，[first,last)必须是一个有效的范围。

如果你所使用的是一个带有检测机制的标准库，那么应该可以检测出上述问题。不然的话，程序很可能在copy()函数的执行期间或者执行之后，出现一个难以分析的内存信息转储（core dump）。

```
e.insert( --e.end() , TodaysDate() );
```

第一处错误：表达式“--e.end()”可能是非法的。

理由很简单，不过略有些不太好理解：在通常标准库的实现中，都是用Date*类型的指针来表示vector<Date>::iteraor，而在C++中并不允许对内部类型的临时变量进行修改。例如，下面的代码同样是非法的：

```
Date* f();    // 返回类型为Date*的函数
p = --f();    // 错误，应该写为"f() - 1"
```

幸运的是，我们知道vector<Date>::iteraor是个随机访问的迭代器，所以将代码修改成下面这样并不会对程序的性能带来损失：

```
e.insert( e.end() - 1, TodaysDate() );
```

第二处错误：如果容器e是空的，当我们想获得“指向e.end()之前一个位置的迭代器”时（不论是写--e.end()还是写成e.end()-1），所得到的都不是一个有效的迭代器。

```
    copy( first,
          last,
          ostream_iterator<Date>( cout, "\n" ) );
}
```

错误：frist和last可能不再是有效的迭代器。

vector在内存中所占的空间是以块（chunk）为单位来增长的，因此它并不需要在每次插入新的元素时都重新分配缓冲区。只有当vector已经被填满并且还需要插入新的元素时，才会引发内存的重新分配。

本例中，在经过e.insert()的操作之后，vector所占的内存空间可能会增长，也可能不会增长；也就是说，vector所占用的内存可能会移动，也可能没有移动。正是由于这种不确定性，使得我们必须认为现有的迭代器都将不再有效。如果在本例中，内存确实移动了，那么在执行copy()时将会再次产生难以分析的内存信息转储。

指导原则

绝对不要对无效的迭代器进行解引用操作。

小结：在使用迭代器时，我们必须清楚以下四点。

1. 迭代器的有效值：这个迭代器可以解引用吗？如果你写成*e.end()，那么这绝对是个错误。

2. 迭代器的有效生存期：这个迭代器在使用时还是有效的吗？它是否会因为某些操作而变得无效？

3. 迭代器的有效范围：两个迭代器是否能构成一个有效范围？first是否确实在last之前（或者first等于last）？二者所指向的是否是同一个容器？

4. 对内部类型的非法操作：程序是否试图去修改内部类型的临时变量，例如在--e.end()这样的表达式中？幸运的是，编译器通常能够发现这种类型的错误，而如果迭代器的类型是一个类（并非内部类型），那么库的编写者通常会允许这种写法，因为这可以带来语法上的便利。

条款2：大小写不敏感的字符串——之一　　难度系数7

你是否需要一个大小写不敏感（Case-Insensitive）的字符串类？如果你乐意的话，可以自己来编写一个。

在本条款中包括以下三个相关的问题：

1. “大小写不敏感”的含义是什么？

2. 编写一个ci_string类，并且这个类的用法与标准库中std::string类的用法应该是一致的。所不同的是，ci_string对大小写不敏感，这一点与我们通常提供的标准库扩展函

数strimp()[⊖]的行为是一致的。ci_string的用法应该可以像下面这样：

```
ci_string s( "AbCdE" );
// 大小写不敏感
//
assert( s == "abcde" );
assert( s == "ABCDE" );
// 当然，大小写将保持不变
//
assert( strcmp( s.c_str(), "AbCdE" ) == 0 );
assert( strcmp( s.c_str(), "abcde" ) != 0 );
```

3. 将大小写敏感性作为对象的属性是否合适？

解答

上述三个问题的解答如下所示：

1. “大小写不敏感”的含义是什么？

“大小写不敏感”的确切含义完全取决于你的应用程序和所使用的自然语言。例如，在许多自然语言中根本就没有大小写的概念。而对于那些支持大小写的自然语言，你可能还需要决定重音字符与非重音字符在比较时是否相等，以及诸如此类的情况。本条款所要解答的问题是：如何在标准的字符串上实现大小写不敏感性，而不考虑你所处的语言环境是什么。

2. 编写一个ci_string类，并且这个类的用法与标准库中std::string类的用法应该是一致的。所不同的是，ci_string对大小写不敏感，这一点与我们通常提供的标准库扩展函数strimp()的行为是一致的。

像“我如何实现一个大小写不敏感的字符串类”这样的问题是非常普遍的，我们甚至可以为其建立一个FAQ——因此就有了本条款。

以下是我们想要实现的功能：

```
ci_string s( "AbCdE" );
// 大小写不敏感
//
assert( s == "abcde" );
assert( s == "ABCDE" );
// 当然，大小写将保持不变
//
assert( strcmp( s.c_str(), "AbCdE" ) == 0 );
assert( strcmp( s.c_str(), "abcde" ) != 0 );
```

这里的关键之处在于：我们要理解在标准C++中，string类到底是由什么来表示的。如果去查看string的头文件，你将会看到像下面这样的代码：

```
typedef basic_string<char> string;
```

从上面我们可以看到，string并不是一个类：它实际上是一个由模板所生成的类型。接

⊖ 函数stricmp()在比较字符串的时候对大小写是不敏感的。虽然stricmp()并不是C或者C++标准的一部分，但在许多C和C++的编译器中都提供了这个扩展函数。

下来，让我们来看看下面模板basic_string<>的声明，其中可以指定其他的模板特化参数：

```
template<class charT,
         class traits = char_traits<charT>,
         class Allocator = allocator<charT> >
class basic_string;
```

因此，“string”的真实类型是“basic_string<char, char_traits<char>, allocator<char>>”，在使用这个模板的时候也可以指定其他默认的模板特化参数。这里我们不用关心allocator部分，而应把重点放在char_traits部分，因为char_traits定义了字符之间如何来进行交互——包括字符的比较。

现在让我们来对字符串进行比较。basic_string模板类提供了一些有用的比较函数，通过这些函数，我们可以知道一个字符串是否等于、小于、或者大于另外一个字符串等等。这些字符串比较函数都是建立在char_traits模板所提供的字符比较函数之上的。例如，在char_traits模板中提供了字符比较函数eq()和lt()来进行两个字符之间的相等比较和小于比较，并且还提供了compare()和find()这两个函数来对字符序列进行比较和查找。

如果我们希望这些字符串的比较有着不同的行为，那么我们所要做的就是提供一个不同于char_traits的模板。以下是最简单的方法：

```
struct ci_char_traits : public char_traits<char>
              // 对于不需要进行替换的函数，
              // 只需继承下来即可
{
  static bool eq( char c1, char c2 )
    { return toupper(c1) == toupper(c2); }
  static bool lt( char c1, char c2 )
    { return toupper(c1) <  toupper(c2); }
  static int compare( const char* s1,
                      const char* s2,
                      size_t n )
    { return memicmp( s1, s2, n ); }
          // 如果你的系统支持memicmp,则直接使用
          // 否则，你要自己去实现这个功能
  static const char*
  find( const char* s, int n, char a )
  {
    while( n-- > 0 && toupper(*s) != toupper(a) )
    {
        ++s;
    }
    return n >= 0 ? s : 0;
  }
};
```

最后，我们将这些关键部分组合起来：

```
typedef basic_string<char, ci_char_traits> ci_string;
```

我们在上面的工作就是创建了一个ci_string类，这个类与标准库中的string类（在大多数情况下，string所表示的就是标准字符串）的行为是非常类似的，只不过在

ci_string类中是通过ci_char_traits而不是char_traits<char>来获得字符之间的比较运算规则。由于我们在ci_char_traits规则中实现了大小写不敏感性，因此也就使得ci_string本身具有了大小写不敏感的属性，除此之外我们并没有做任何其他的改动——也就是说，我们没有对basic_string进行任何修改，就获得了一个大小写不敏感的字符串类。这就是所谓的扩展性。

3. 将大小写敏感作为对象的属性是否合适？

通常，我们都是将大小写敏感作为比较函数的属性而不是对象的属性，这种作法在一般情况下都是更为有用的。例如，我们来考虑以下的代码：

```
string    a = "aaa";
ci_string b = "aAa";
if( a == b ) /* ... */
```

假定我们有一个函数operator==(),那么对表达式“a==b”的求值结果是真还是假？我们可以很容易得出结论：如果两个操作数中的任何一个是大小写不敏感的，那么比较运算就应该也是大小写不敏感的。但是，如果我们稍微修改一下这个示例，引入basic_string的另一个模板特化类型，并进行比较：

```
typedef basic_string<char, yz_char_traits> yz_string;

ci_string b = "aAa";
yz_string c = "AAa";
if( b == c ) /* ... */
```

现在，我们再来考虑这个问题：对表达式"a == b"的求值结果是真还是假？在上面的情况中，我想你也会同意，这两个对象之间的比较运算语义并不是显而易见的。

相反，如果将示例写成下面这样的话，比较运算的语义就会变得很清晰：

```
string a = "aaa";
string b = "aAa";
if( stricmp( a.c_str(), b.c_str() ) == 0 ) /* ... */
string c = "AAa";
if( EqualUsingYZComparison( b, c ) ) /* ... */
```

在许多情况下，将大小写敏感性作为比较运算的属性是更为有用的。但在实际的编码过程中，我也遇到过将大小写敏感性作为对象属性的情况（尤其当大多数或者所有的比较运算都与C风格的char*字符串有关时），在这些情况中，将大小写敏感性作为对象的属性则是更为有用的。因为你可以使用“自然的”方式去对字符串进行比较（例如，“if(a==“text”)...”），而无需提醒自己每次都要使用大小写敏感的比较函数。

通过对本条款的学习，你应该能够了解basic_string模板的使用方法以及它在实际中的灵活性。如果你希望获得不同于在函数memicmp()和toupper()中所提供的比较运算，那么你只需用你自己的代码来替换这里所给出的5个函数，并且在代码中实现适合你的应用程序的字符比较运算。