Broadview®
www.broadview.com.cn
PEARSON

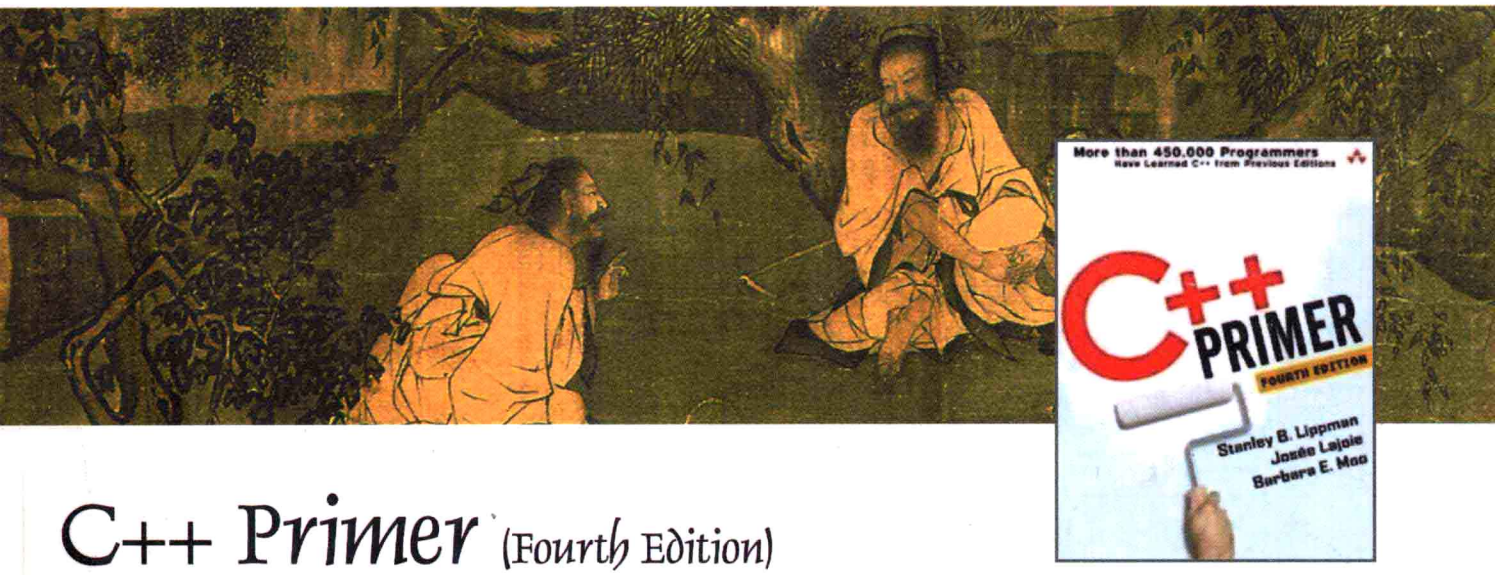
传世经典书丛
Eternal Classics

C++ Primer (第4版)

(评注版)

Stanley B. Lippman
[美] Josée Lajoie 著
Barbara E. Moo

评注 陈硕



C++ Primer (Fourth Edition)

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

C++ Primer（第4版）(评注版）

C++ Primer(Fourth Edition)

Stanley B. Lippman
[美] Josée Lajoie 著
Barbara E. Moo

陈硕 评注

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

《C++ Primer》是一本系统而权威的C++教材，它全面而深入地讲解了C++语言及其标准库。本书作者Stanley B. Lippman在20世纪80年代早期即在C++之父Bjarne Stroustrup领导下开发C++编译器，另一作者Josée Lajoie曾多年担任C++标准委员会核心语言组主席，他们对这门编程语言的理解与把握非常人可比。本书对C++语法和语义的阐释兼具准确性与可读性，在坊间无出其右者。第4版更吸收了先进的C++教学经验，在内容组织上对初学者更加友好，详略得当且重点突出，使读者能更快上手编写有用的程序，也更适合自学。全球已有45万人通过该书的各个版本学习了C++编程。

对于国外技术图书，选择翻译版还是影印版，常常让人陷入两难的境地。本评注版力邀国内资深专家执笔，在英文原著基础上增加中文点评与注释，旨在融合二者之长，既保留经典的原创文字与味道，又以先行者的学研心得与实践感悟，对读者阅读与学习加以点拨、指明捷径。

经过评注的版本，更值得反复阅读与体会。希望这本书能够帮助您跨越C++的重重险阻，领略高处才有的壮美风光，做一个成功而快乐的C++程序员。

版权贸易登记号　图字：01-2012-4456

图书在版编目（CIP）数据

C++ Primer：第4版：评注版/（美）李普曼（Lippman,S.B.），（美）拉乔伊（Lajoie,J.），（美）莫（Moo,B.E.）著；陈硕评注. —北京：电子工业出版社，2012.7

（传世经典书丛）

ISBN 978-7-121-17441-4

Ⅰ.①C… Ⅱ.①李… ②拉… ③莫… ④陈… Ⅲ.①C语言－程序设计－教材 Ⅳ.①TP312

中国版本图书馆CIP数据核字（2012）第138153号

策划编辑：张春雨

责任编辑：李云静

印　　刷：北京天宇星印刷厂

装　　订：三河市皇庄路通装订厂

出版发行：电子工业出版社

北京市海淀区万寿路173信箱　　邮编：100036

开　　本：850×1168　1/16　印张：43　　字数：1320千字

印　　次：2012年7月第1次印刷

定　　价：108.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

出版说明

悦读上品　得乎益友

孔子云："取乎其上，得乎其中；取乎其中，得乎其下；取乎其下，则无所得矣"。

对于读书求知而言，这句古训教我们去读好书，最好是好书中的上品——经典书。其中，科技人员要读的技术书，因为直接关乎客观是非与生产效率，阅读选材本更应慎重。然而，随着技术图书品种的日益丰富，发现经典书越来越难，尤其对于涉世尚浅的新读者，更为不易，而他们又往往是最需要阅读、提升的重要群体。

所谓经典书，或说上品，是指选材精良、内容精练、讲述生动、外延丰盈、表现手法体贴入微的读品，它们会成为读者的知识和经验库中的重要组成部分，并且拥有从不断重读中汲取养分的空间。因此，选择阅读上品的问题便成了有效阅读的首要问题。当然，这不只是效率问题，上品促成的既是对某一种技术、思想的真正理解和掌握，同时又是一种感悟或享受，是一种愉悦。

与技术本身类似，经典 IT 技术书多来自国外。深厚的积累、良好的写作氛围，使一批大师为全球技术学习者留下了璀璨的智慧瑰宝。就在那个年代即将远去之时，无须回眸，也能感受到这一部部厚重而深邃的经典著作，在造福无数读者后从未蒙尘的熠熠光辉。而这些凝结众多当今国内技术中坚美妙记忆与绝佳体验的技术图书，虽然尚在国外图书市场上大放异彩，却已逐渐淡出国人的视线。最为遗憾的是，迟迟未有可以填补空缺的新书问世。而无可替代，不正是经典书被奉为圭臬的原因？

为了不让国内读者，尤其是即将步入技术生涯的新一代读者，就此错失这些滋养过先行者们的好书，以出版 IT 精品图书，满足技术人群需求为己任的我们，愿意承担这一使命。本次机遇惠顾了我们，让我们有机会携手权威的 Pearson 公司，精心推出"传世经典书丛"。

在我们眼中，"传世经典"的价值首先在于——既适合喜爱科技图书的读者，也符合专家们挑剔的标准。幸运的是，我们的确找到了这些堪称上品的佳作。丛书带给我们的幸运颇多，细数一下吧。

得以引荐大师著作

有恐思虑不周，我们大量参考了国外权威机构和网站的评选结果，并得到了 Pearson 的专业支持，又进

一步对符合标准之图书的国内外口碑与销售情况进行细致分析，也听取了国内技术专家的宝贵建议，才有幸选出对国内读者最富有技术养分的大师上品。

向深邃的技术内涵致敬

中外技术环境存在差异，很多享誉国外的好书未必适用于国内读者；且技术与应用瞬息万变，很容易让人心生迷惘或疲于奔命。本丛书的图书遴选，注重打好思考方法与技术理念的根基，旨在帮助读者修炼内功，提升境界，将技术真正融入个人知识体系，从而可以一通百通，从容面对随时涌现的技术变化。

翻译与评注的双项选择

引进优秀外版著作，将其翻译为中文供国内读者阅读，较为有效与常见。但另有一些外语水平较高、喜好阅读原版的读者，苦于对技术理解不足，不能充分体会原文表述的精妙，需要有人指导与点拨。而一批本土技术精英经过长期经典熏陶及实践锤炼，已足以胜任这一工作。有鉴于此，本丛书在翻译版的同时推出融合英文原著与中文点评、注释的评注版，供不同志趣的读者自由选择。

承蒙国内一流译(注)者的扶持

优秀的英文原著最终转化为真正的上品，尚需跨越翻译鸿沟，外版图书的翻译质量一直屡遭国内读者诟病。评注版的增值与含金量，同样依赖于评注者的高卓才具。好在，本丛书得到了久经考验的权威译(注)者的认可和支持，首肯我们选用其佳作，或亲自参与评注工作。正是他们的参与保证了经典的品质，既再次为我们的选材把关，更提供了一流的中文表述。

期望带给读者良好的阅读体验

一本好书带给人的愉悦不止于知识收获，良好的阅读感受同样不可缺少，且对学业不无助益。为让读者收获与上品相称的体验，我们在图书装帧设计与选材用料上同样不敢轻率，惟愿送到读者手中的除了珠玑章句，还有舒适与熨帖的视觉感受。

所有参与丛书出版的人员，尽管能力有限，却无不心怀严谨之心与完美愿望。如果读者朋友能从潜心阅读这些上品中偶有获益，不啻为对我们工作的最佳褒奖。若有阅读感悟，敬请拨冗告知，以鼓励我们继续在这一道路上贡献绵薄之力。如有不周之处，也请不吝指教。

电子工业出版社博文视点

从《C++ Primer（第 4 版）》入手学习 C++

为什么要学习 C++ ？

2009 年本书作者 Stanley Lippman 先生来华参加上海祝成科技举办的 C++ 技术大会，他表示人们现在还用 C++ 的唯一理由是其性能。相比之下，Java/C#/Python 等语言更加易学易用并且开发工具丰富，它们的开发效率都高于 C++。但 C++ 目前仍然是运行最快的语言[1]，如果你的应用领域确实在乎这个性能，那么 C++ 是不二之选。

这里略举几个例子[2]。对于手持设备而言，提高运行效率意味着完成相同的任务需要更少的电能，从而延长设备的操作时间，增强用户体验。对于嵌入式[3]设备而言，提高运行效率意味着：实现相同的功能可以选用较低档的处理器和较少的存储器，降低单个设备的成本；如果设备销量大到一定的规模，可以弥补 C++ 开发的成本。对于分布式系统而言，提高 10% 的性能就意味着节约 10% 的机器和能源。如果系统大到一定的规模（数千台服务器），值得用程序员的时间去换取机器的时间和数量，可以降低总体成本。另外，对于某些延迟敏感的应用（如游戏[4]、金融交易），通常不能容忍垃圾收集（GC）带来的不确定延时，而 C++ 可以自动并精确地控制对象销毁和内存释放时机[5]。我曾经不止一次见到，出于性能原因，用 C++ 重写现有的 Java 或 C# 程序。

C++ 之父 Bjarne Stroustrup 把 C++ 定位于偏重系统编程（system programming）[6]的通用程序设计语言，开发信息基础架构（infrastructure）是 C++ 的重要用途之一[7]。Herb Sutter 总结道[8]，C++ 注重运行效率（efficiency）、灵活性（flexibility）[9]和抽象能力（abstraction），并为此付出了生产力（productivity）方面

1 见编程语言性能对比网站（http://shootout.alioth.debian.org/）和 Google 员工写的语言性能对比论文（https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf）。

2 C++ 之父 Bjarne Stroustrup 维护的 C++ 用户列表：http://www2.research.att.com/~bs/applications.html。

3 初窥 C++ 在嵌入式系统中的应用，参见 http://aristeia.com/TalkNotes/MISRA_Day_2010.pdf。

4 Milo Yip 在《C++ 强大背后》提到大部分游戏引擎（如 Unreal/Source）及中间件（如 Havok/FMOD）是 C++ 实现的。参见 http://www.cnblogs.com/miloyip/archive/2010/09/17/behind_cplusplus.html。

5 参见孟岩的《垃圾收集机制批判》：C++ 利用智能指针达成的效果是，一旦某对象不再被引用，系统刻不容缓，立刻回收内存。这通常发生在关键任务完成后的清理（cleanup）时期，不会影响关键任务的实时性；同时，内存里所有的对象都是有用的，绝对没有垃圾空占内存。参见 http://blog.csdn.net/myan/article/details/1906。

6 有人半开玩笑地说："所谓系统编程，就是那些 CPU 时间比程序员的时间更重要的工作。"

7 《Software Development for Infrastructure》（http://www2.research.att.com/~bs/Computer-Jan12.pdf）。

8 Herb Sutter 在 C++ and Beyond 2011 会议上的开场演讲：《Why C++?》，参见 http://channel9.msdn.com/posts/C-and-Beyond-2011-Herb-Sutter-Why-C。

9 这里的灵活性指的是编译器不阻止你干你想干的事情，比如为了追求运行效率而实现即时编译（just-in-time compilation）。

的代价[10]。用本书作者的话来说，就是“C++ is about efficient programming with abstractions”（C++ 的核心价值在于能写出“运行效率不打折扣的抽象[11]”）。

要想发挥 C++ 的性能优势，程序员需要对语言本身及各种操作的代价有深入的了解[12]，特别要避免不必要的对象创建[13]。例如下面这个函数如果漏写了 &，功能还是正确的，但性能将会大打折扣。编译器和单元测试都无法帮我们查出此类错误，程序员自己在编码时须得小心在意。

```
inlineintfind_longest(conststd::vector<std::string>& words)
{
  // std::max_element(words.begin(), words.end(), LengthCompare());
}
```

在现代 CPU 体系结构下，C++ 的性能优势很大程度上得益于对内存布局（memory layout）的精确控制，从而优化内存访问的局部性[14]（locality of reference）并充分利用内存阶层（memory hierarchy）提速[15]，这一点优势在近期内不会被基于 GC 的语言赶上[16]。

C++ 的协作性不如 C、Java、Python，开源项目也比这几个语言少得多，因此在 TIOBE 语言流行榜中节节下滑。但是据我所知，很多企业内部使用 C++ 来构建自己的分布式系统基础架构，并且有替换 Java 开源实现的趋势。

学习 C++ 只需要读一本大部头

C++ 不是特性（features）最丰富的语言，却是最复杂的语言，诸多语言特性相互干扰，使其复杂度成倍增加。鉴于其学习难度和知识点之间的关联性，恐怕不能用“粗粗看看语法，就撸起袖子开干，边查 Google 边学习[17]”这种方式来学习 C++，那样很容易掉到陷阱里或养成坏的编程习惯。如果想成为专业 C++ 开发者，全面而深入地了解这门复杂语言及其标准库，你需要一本系统而权威[18]的书，这样的书必定会是一本八九百页的大部头[19]。

兼具系统性和权威性的 C++ 教材有两本，C++ 之父 Bjarne Stroustrup 的代表作《The C++ Programming Language》和 Stanley Lippman 的这本《C++ Primer》。侯捷先生评价道：“泰山北斗已现，

10 我曾向 Stanley Lippman 介绍目前我在 Linux 下的工作环境（编辑器、编译器、调试器），他表示这跟他在 20 世纪 70 年代的工作环境相差无几，可见 C++ 在开发工具方面的落后。另外，C++ 的编译运行调试周期也比现代的语言长，这多少影响了工作效率。

11 可参考 Ulrich Drepper 在《Stop Underutilizing Your Computer》中举的 SIMD 例子。参见 http://www.redhat.com/f/pdf/summit/udrepper_945_stop_underutilizing.pdf。

12 《Technical Report on C++ Performance》（http://www.open-std.org/jtc1/sc22/wg21/docs/18015.html）。

13 可参考 Scott Meyers 的《Effective C++ in an Embedded Environment》（http://www.artima.com/shop/effective_cpp_in_an_embedded_environment）。

14 我们知道 std::list 的任一位置插入的是 O(1) 操作，而 std::vector 的任一位置插入的是 O(*N*) 操作，但由于 std::vector 的元素布局更加紧凑（compact），很多时候 std::vector 的随机插入性能甚至会高于 std::list。参见 http://ecn.channel9.msdn.com/events/GoingNative12/GN12Cpp11Style.pdf，这也佐证了 std::vector 是首选容器。

15 可参考 Scott Meyers 的技术报告《CPU Caches and Why You Care》和任何一本现代的计算机体系结构教材（http://aristeia.com/TalkNotes/ACCU2011_CPUCaches.pdf）。

16 Bjarne Stroustrup 有一篇论文《Abstraction and the C++ machine model》，对比了 C++ 和 Java 的对象内存布局。参见 http://www2.research.att.com/~bs/abstraction-and-machine.pdf。

17 语出孟岩的《快速掌握一个语言最常用的 50%》（http://blog.csdn.net/myan/article/details/3144661）。

18 “权威”的意思是说你不用担心作者讲错了，能达到这个水准的 C++ 图书作者全世界也屈指可数。

19 同样篇幅的 Java/C#/Python 教材可以从语言、标准库一路讲到多线程、网络编程、图形编程。

又何必案牍劳形于墨瀚书海之中！这两本书都从 C++ 盘古开天以来，一路改版，斩将擎旗，追奔逐北，成就一生荣光[20]。”

从实用的角度，这两本书读一本即可，因为它们覆盖的 C++ 知识点相差无几。就我个人的阅读体验而言，Primer 更易读一些，我 10 年前深入学习 C++ 正是用的《C++ Primer（第 3 版）》。这次借评注的机会仔细阅读了《C++ Primer（第 4 版）》，感觉像在读一本完全不同的新书。第 4 版的内容组织及文字表达比第 3 版进步很多[21]，第 3 版可谓“事无巨细、面面俱到”；第 4 版则重点突出、详略得当，甚至篇幅也缩短了，这多半归功于新加盟的作者 Barbara Moo。

《C++ Primer（第 4 版）》讲什么？适合谁读？

这是一本 C++ 语言的教程，不是编程教程。本书不讲八皇后问题、Huffman 编码、汉诺塔、约瑟夫环、大整数运算等经典编程例题，本书的例子和习题往往都跟 C++ 本身直接相关。本书的主要内容是精解 C++ 语法（syntax）与语意（semantics），并介绍 C++ 标准库的大部分内容（含 STL）。“这本书在全世界 C++ 教学领域的突出和重要，已经无须我再赘言[22]。”

本书适合 C++ 语言的初学者，但不适合编程初学者。换言之，这本书可以是你的第一本 C++ 书，但恐怕不能作为第一本编程书。如果你不知道什么是变量、赋值、分支、条件、循环、函数，你需要一本更加初级的书[23]，本书第 1 章可用做自测题。

如果你已经学过一门编程语言，并且打算成为专业 C++ 开发者，从《C++ Primer（第 4 版）》入手不会让你走弯路。值得特别说明的是，学习本书不需要事先具备 C 语言知识。相反，这本书教你编写真正的 C++ 程序，而不是披着 C++ 外衣的 C 程序。

《C++ Primer（第 4 版）》的定位是语言教材，不是语言规格书，它并没有面面俱到地谈到 C++ 的每一个角落，而是重点讲解 C++ 程序员日常工作中真正有用的、必须掌握的语言设施和标准库[24]。本书的作者一点也不炫耀自己的知识和技巧，虽然他们有十足的资本[25]。这本书用语非常严谨（没有那些似是而非的比喻），用词平和，讲解细致，读起来并不枯燥。特别是如果你已经有一定的编程经验，在阅读时不妨思考如何用 C++ 来更好地完成以往的编程任务。

尽管英文原版书篇幅近 900 页，但其内容还是十分紧凑的，很多地方读一个句子就值得写一小段代码去验证。为了节省篇幅，本书经常修改前文代码中的一两行，来说明新的知识点，值得把每一行代码敲到机器中去验证。习题当然也不能轻易放过。

《C++ Primer（第 4 版）》体现了现代 C++ 教学与编程理念：在现成的高质量类库上构建自己的程序，而不是什么都从头自己写。这本书在第 3 章中介绍了 string 和 vector 这两个常用的类，立刻就能写出很

20 侯捷《大道之行也——C++ Primer 3/e 译序》（http://jjhou.boolan.com/cpp-primer-foreword.pdf）。

21 Bjarne Stroustrup 在《Programming: Principles and Practice Using C++》的参考文献中引用了本书，并特别注明“use only the 4th edition”。

22 参见侯捷的《C++ Primer 4/e 译序》。

23 如果没有时间精读注 21 中提到的那本大部头，短小精干的《Accelerated C++》亦是上佳之选。另外，如果想从 C 语言入手，我推荐裘宗燕老师的《从问题到程序：程序设计与 C 语言引论（第 2 版）》（机械工业出版社出版）。

24 本书把 iostream 的格式化输出放到附录，彻底不谈 locale/facet，可谓匠心独运。

25 Stanley Lippman 曾 说：“Virtual base class support wanders off into the Byzantine...The material is simply too esoteric to warrant discussion...”

多有用的程序。但作者不是一次性把 string 的上百个成员函数一一列举，而是有选择地讲解了最常用的那几个函数。

《C++ Primer（第 4 版）》的代码示例质量很高，不是那种随手写的玩具代码。第 10.4.2 节实现了带禁用词的单词计数；第 10.6 利用标准库容器简洁地实现了基于倒排索引思路的文本检索；第 15.9 节又用面向对象方法扩充了文本检索的功能，支持布尔查询。值得一提的是，这本书讲解继承和多态时举的例子符合 Liskov 替换原则，是正宗的面向对象。相反，某些教材以复用基类代码为目的，常以“人、学生、老师、教授”或“雇员、经理、销售、合同工”为例，这是误用了面向对象的“复用”。

《C++ Primer（第 4 版）》出版于 2005 年，遵循 2003 年的 C++ 语言标准[26]。C++ 新标准已于 2011 年定案（称为 C++11），本书不涉及 TR1[27] 和 C++11，这并不意味着这本书过时了[28]。相反，这本书里沉淀的都是当前广泛使用的 C++ 编程实践，学习它可谓正当时。评注版也不会越俎代庖地介绍这些新内容，但是会指出哪些语言设施已在新标准中废弃，避免读者浪费精力。

《C++ Primer（第 4 版）》是平台中立的，并不针对特定的编译器或操作系统。目前最主流的 C++ 编译器有两个，GNUG++ 和微软的 Visual C++。实际上，这两个编译器阵营基本上“模塑[29]”了 C++ 语言的行为。理论上讲，C++ 语言的行为是由 C++ 标准规定的。但是 C++ 不像其他很多语言有“官方参考实现[30]”，因此 C++ 的行为实际上是由语言标准、几大主流编译器、现有不计其数的 C++ 产品代码共同确定的，三者相互制约。C++ 编译器不光要尽可能符合标准，同时也要遵循目标平台的成文或不成文规范和约定，例如高效地利用硬件资源、兼容操作系统提供的 C 语言接口等。在 C++ 标准没有明文规定的地方，C++ 编译器也不能随心所欲地自由发挥。学习 C++ 的要点之一是明白哪些行为是由标准保证的，哪些是由实现（软硬件平台和编译器）保证的[31]，哪些是编译器自由实现、没有保证的；换言之，明白哪些程序行为是可依赖的。从学习的角度，我建议如果有条件，不妨两个编译器都用[32]，相互比照，避免把编译器和平台特定的行为误解为 C++ 语言规定的行为。尽管不是每个人都需要写跨平台的代码，但也大可不必自我限定在编译器的某个特定版本，毕竟编译器是会升级的。

本着“练从难处练，用从易处用”的精神，我建议在命令行下编译运行本书的示例代码，并尽量少用调试器。另外，值得了解 C++ 的编译链接模型[33]，这样才能不被实际开发中遇到的编译错误或链接错误绊住手脚。（C++ 不像现代语言那样有完善的模块（module）和包（package）设施，它从 C 语言继承了头文件、源文件、库文件等古老的模块化机制，这套机制相对较为脆弱，需要花一定时间学习规范的做法，避免误用。）

就学习 C++ 语言本身而言，我认为有几个练习非常值得一做。这不是“重复发明轮子”，而是必

---

26 基本等同于 1998 年的初版 C++ 标准，修正了编译器作者关心的一些问题，与普通程序员基本无关。

27 TR1 是 2005 年 C++ 标准库的一次扩充，增加了智能指针、bind/function、哈希表、正则表达式等。

28 作者正在编写《C++ Primer（第 5 版）》，会包含 C++11 的内容。

29 G++ 统治了 Linux 平台，并且能用在很多 Unix 平台上；Visual C++ 统治了 Windows 平台。其他 C++ 编译器的行为通常要向它们靠拢，例如 Intel C++ 在 Linux 上要兼容 G++，而在 Windows 上要兼容 Visual C++。

30 曾经是 Cfront，本书作者正是其主要开发者。参见 http://www.softwarepreservation.org/projects/c_plus_plus。

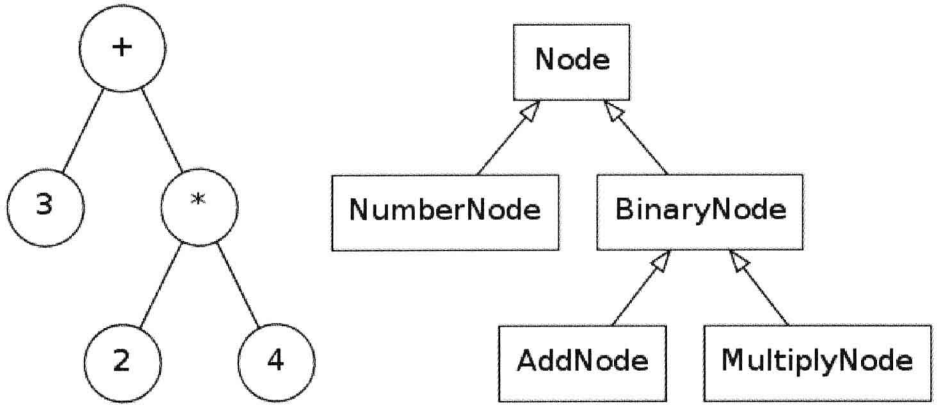
31 包括 C++ 标准有规定，但编译器拒绝遵循的。参见 http://stackoverflow.com/questions/3931312/value-initialization-and-non-pod-types。

32 G++ 是免费的，可使用较新的 4.x 版，最好 32-bit 和 64-bit 一起用，因为服务端已经普及 64 位。微软也有免费的编译器，可考虑 Visual C++ 2010 Express，建议不要用老掉牙的 Visual C++ 6.0 作为学习平台。

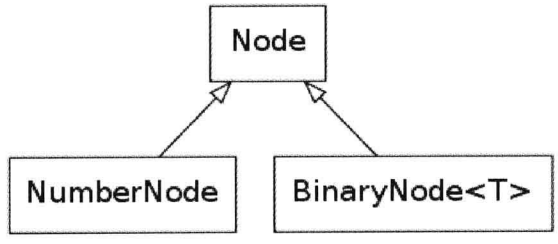
33 可参考陈硕写的《C++ 工程实践经验谈》中的“C++ 编译模型精要”一节。

要的编程练习，帮助你熟悉、掌握这门语言。一是写一个复数类或者大整数类[34]，实现基本的运算，熟悉封装与数据抽象。二是写一个字符串类，熟悉内存管理与拷贝控制。三是写一个简化的 vector<T> 类模板，熟悉基本的模板编程，你的这个 vector 应该能放入 int 和 string 等元素类型。四是写一个表达式计算器，实现一个节点类的继承体系（右图），体会面向对象编程。前三个练习是写独立的值语义的类，第四个练习是对象语义，同时要考虑类与类之间的关系。

表达式计算器能把四则运算式 3+2×4 解析为左图的表达式树[35]，对根节点调用 calculate() 虚函数就能算出表达式的值。做完之后还可以再扩充功能，比如支持三角函数和变量。



在写完面向对象版的表达式树之后，还可以略微尝试泛型编程。比如把类的继承体系简化为下图，然后用 BinaryNode<std::plus<double>> 和 BinaryNode<std:: multiplies<double>> 来具现化 BinaryNode<T> 类模板，通过控制模板参数的类型来实现不同的运算。



在表达式树这个例子中，节点对象是动态创建的，值得思考：如何才能安全地、不重不漏地释放内存。本书第 15.8 节的 Handle 可供参考。（C++ 的面向对象基础设施相对于现代的语言而言显得很简陋，现在 C++ 也不再以“支持面向对象”为卖点了。）

C++ 难学吗？“能够靠读书、看文章、读代码、做练习学会的东西没什么门槛，智力正常的人只要愿意花工夫，都不难达到（不错）的程度。[36]” C++ 好书很多，不过优秀的 C++ 开源代码很少，而且风格迥异[37]。我这里按个人口味和经验列几个供读者参考阅读：Google 的 protobuf、leveldb、PCRE 的 C++ 封装，我自己写的 muduo 网络库。这些代码都不长，功能明确，阅读难度不大。如果有时间，还可以读一读 Chromium 中的基础库源码。在读 Google 开源的 C++ 代码时要连注释一起细读。我不建议

34 大整数类可以以 std::vector<int> 为成员变量，避免手动资源管理。

35 “解析”可以用数据结构课程介绍的逆波兰表达式方法，也可以用编译原理中介绍的递归下降法，还可以用专门的 Packrat 算法。可参考 http://www.relisoft.com/book/lang/poly/3tree.html。

36 参见孟岩的《技术路线的选择重要但不具有决定性》（http://blog.csdn.net/myan/article/details/3247071）。

37 从代码风格上往往能判断项目成型的时代。

一开始就读 STL 或 Boost 的源码，因为编写通用 C++ 模板库和编写 C++ 应用程序的知识体系相差很大。另外可以考虑读一些优秀的 C 或 Java 开源项目，并思考是否可以用 C++ 更好地实现或封装之（特别是资源管理方面能否避免手动清理）。

继续前进

我能够随手列出十几本 C++ 好书，但是从实用角度出发，这里只举两三本必读的书。读过《C++ Primer》和这几本书之后，想必读者已能自行识别 C++ 图书的优劣，可以根据项目需要加以钻研。

第一本是《Effective C++（第 3 版）》[38]。学习语法是一回事，高效地运用这门语言是另一回事。C++ 是一个遍布陷阱的语言，吸取专家的经验尤为重要，既能快速提高眼界，又能避免重蹈覆辙。《C++ Primer》加上这本书包含的 C++ 知识足以应付日常应用程序开发。

我假定读者一定会阅读这本书，因此在评注中不引用《Effective C++（第 3 版）》的任何章节。

《Effective C++（第 3 版）》的内容也反映了 C++ 用法的进步。第 2 版建议“总是让基类拥有虚析构函数”，第 3 版改为“为多态基类声明虚析构函数”。因为在 C++ 中，“继承”不光只有面向对象这一种用途，即 C++ 的继承不一定是为了覆写（override）基类的虚函数。第 2 版花了很多笔墨介绍浅拷贝与深拷贝，以及对指针成员变量的处理 [39]。第 3 版则提议，对于多数 class 而言，要么直接禁用拷贝构造函数和赋值操作符；要么通过选用合适的成员变量类型 [40]，使得编译器默认生成的这两个成员函数就能正常工作。

什么是 C++ 编程中最重要的编程技法（idiom）？我认为是“用对象来管理资源”，即 RAII。资源包括动态分配的内存 [41]，也包括打开的文件、TCP 网络连接、数据库连接、互斥锁等。借助 RAII，我们可以把资源管理和对象生命期管理等同起来，而对象生命期管理在现代 C++ 里根本不困难（见注 3），只需要花几天时间熟悉几个智能指针 [42] 的基本用法即可。学会了这三招两式，现代的 C++ 程序中可以完全不写 delete，也不必为指针或内存错误操心。现代 C++ 程序里出现资源和内存泄漏的唯一可能是循环引用，一旦发现，也很容易修正设计和代码。这方面的详细内容请参考《Effective C++（第 3 版）》的第 3 章（资源管理）。

C++ 是目前唯一能实现自动化资源管理的语言，C 语言完全靠手工释放资源，而其他基于垃圾收集的语言只能自动清理内存，而不能自动清理其他资源 [43]（网络连接、数据库连接等）。

除了智能指针，TR1 中的 bind/function 也十分值得投入精力去学一学 [44]。让你从一个崭新的视角，重新审视类与类之间的关系。Stephan T. Lavavej 有一套 PPT 介绍 TR1 的这几个主要部件 [45]。

---

38 Scott Meyers 著，侯捷译，电子工业出版社出版。

39 Andrew Koenig 的《Teaching C++ Badly: Introduce Constructors and Destructors at the Same Time》（http://drdobbs.com/blogs/cpp/229500116）。

40 能自动管理资源的 string、vector、shared_ptr 等，这样多数 class 连析构函数都不必写。

41 “分配内存”包括在堆（heap）上创建对象。

42 包括 TR1 中的 shared_ptr、weak_ptr，还有更简单的 boost::scoped_ptr。

43 Java 7 有 try-with-resources 语句，Python 有 with 语句，C# 有 using 语句，可以自动清理栈上的资源，但对生命期大于局部作用域的资源无能为力，需要程序员手工管理。

44 参见孟岩的《function/bind 的救赎（上）》（http://blog.csdn.net/myan/article/details/5928531）。

45 参见 http://blogs.msdn.com/b/vcblog/archive/2008/02/22/tr1-slide-decks.aspx。

第二本书，如果读者还是在校学生，已经学过数据结构课程[46]的话，可以考虑读一读《泛型编程与 STL》[47]；如果已经工作，学完《C++ Primer》立刻就要参加 C++ 项目开发，那么我推荐阅读《C++ 编程规范》[48]。

泛型编程有一套自己的术语，如 concept、model、refinement 等，理解这套术语才能阅读泛型程序库的文档。即便不掌握泛型编程作为一种程序设计方法，也要掌握 C++ 中以泛型思维设计出来的标准容器库和算法库（STL）。坊间面向对象的书琳琅满目，学习机会也很多，而泛型编程只有这么一本，读之可以开拓视野，并且可加深对 STL 的理解（特别是迭代器[49]）和应用。

C++ 模板是一种强大的抽象手段，我不赞同每个人都把精力花在钻研艰深的模板语法和技巧上。从实用角度，能在应用程序中写写简单的函数模板和类模板即可（以 type traits 为限），并非每个人都要去写公用的模板库。

由于 C++ 语言过于庞大复杂，我见过的开发团队都对其剪裁使用[50]。往往团队越大，项目成立时间越早，剪裁得越厉害，也越接近 C。制订一份好的编程规范相当不容易。若规范定得太紧（比如定为团队成员知识能力的交集），则程序员束手束脚，限制了生产力，对程序员个人发展也不利[51]。若规范定得太松（定为团队成员知识能力的并集），则项目内代码风格迥异，学习交流协作成本上升，恐怕对生产力也不利。由两位顶级专家合写的《C++ 编程规范》一书可谓是现代 C++ 编程规范的范本。

《C++ 编程规范》同时也是专家经验一类的书，这本书的篇幅比《Effective C++（第 3 版）》短小，条款数目却多了近一倍，可谓言简意赅。有的条款看了就明白，照做即可：

- 第 1 条，以高警告级别编译代码，确保编译器无警告。
- 第 31 条，避免写出依赖于函数实参求值顺序的代码。C++ 操作符的优先级、结合性与表达式的求值顺序是无关的。裘宗燕老师写的《C/C++ 语言中表达式的求值》[52]一文对此有明确的说明。
- 第 35 条，避免继承“并非设计作为基类使用”的 class。
- 第 43 条，明智地使用 pimpl。这是编写 C++ 动态链接库的必备手法，可以最大限度地提高二进制兼容性。
- 第 56 条，尽量提供不会失败的 swap() 函数。有了 swap() 函数，我们在自定义赋值操作符时就不必检查自赋值了。
- 第 59 条，不要在头文件中或 #include 之前写 using。
- 第 73 条，以 by value 方式抛出异常，以 by reference 方式捕捉异常。
- 第 76 条，优先考虑 vector，其次再选择适当的容器。
- 第 79 条，容器内只可存放 value 和 smart pointer。

有的条款则需要相当的设计与编码经验才能解其中三昧：

---

46 最好再学一点基础的离散数学。

47 Matthew Austern 著，侯捷译，中国电力出版社出版。

48 Herb Sutter 等著，刘基诚译，人民邮电出版社出版（这本书的繁体版由侯捷先生和我翻译）。

49 侯捷先生的《芝麻开门：从 Iterator 谈起》（http://jjhou.boolan.com/programmer-3-traits.pdf）。

50 参见孟岩的《编程语言的层次观点——兼谈 C++ 的剪裁方案》（http://blog.csdn.net/myan/article/details/1920）。

51 一个人通常不会在一个团队工作一辈子，其他团队可能有不同的 C++ 剪裁使用方式，程序员要有“一桶水”的本事，才能应付不同形状大小的水碗。

52 参见 http://www.math.pku.edu.cn/teachers/qiuzy/technotes/expression2009.pdf。

- 第 5 条，为每个物体（entity）分配一个内聚任务。
- 第 6 条，正确性、简单性、清晰性居首。
- 第 8、9 条，不要过早优化；不要过早劣化。
- 第 22 条，将依赖关系最小化。避免循环依赖。
- 第 32 条，搞清楚你写的是哪一种 class。明白 value class、base class、trait class、policy class、exception class 各有其作用，写法也不尽相同。
- 第 33 条，尽可能写小型 class，避免写出“大怪兽”。
- 第 37 条，public 继承意味着可替换性。继承非为复用，乃为被复用。
- 第 57 条，将 class 类型及其非成员函数接口放入同一个 namespace。

值得一提的是，《C++ 编程规范》是出发点，但不是一份终极规范。例如 Google 的 C++ 编程规范[53] 和 LLVM 编程规范[54] 都明确禁用异常，这跟这本书的推荐做法正好相反。

评注版使用说明

评注版采用大 16 开印刷，在保留原书版式的前提下，对其进行了重新分页，评注的文字与正文左右分栏并列排版。另外，本书已依据原书 2010 年第 11 次印刷的版本进行了全面修订。为了节省篇幅，原书每章末尾的小结、术语表及书末的索引都没有印在评注版中，而是做成 PDF 供读者下载，这也方便读者检索。评注的目的是帮助初次学习 C++ 的读者快速深入掌握这门语言的核心知识，澄清一些概念、比较与其他语言的不同、补充实践中的注意事项等。评注的内容约占全书篇幅的 15%，大致比例是三分评、七分注，并有一些补白的内容[55]。如果读者拿不定主意是否购买，可以先翻一翻第 5 章。我在评注中不谈 C++11[56]，但会略微涉及 TR1，因为 TR1 已经投入实用。

为了不打断读者阅读的思路，评注中不会给 URL 链接，评注中偶尔会引用《C++ 编程规范》的条款，以 [CCS] 标明，这些条款的标题已在前文列出。另外评注中出现的 soXXXXXX 表示 http://stackoverflow.com/questions/XXXXXX 网址。

网上资源

代码下载：http://www.informit.com/store/product.aspx?isbn=0201721481

豆瓣页面：http://book.douban.com/subject/10944985/

术语表与索引 PDF 下载：http://chenshuo.com/cp4（本序的电子版也发布于此，方便读者访问脚注中的网站）。

我的联系方式：giantchen@gmail.com
http://weibo.com/giantchen

陈硕
2012 年 5 月
中国 • 香港

53 参见 http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml#Exceptions。

54 参见 http://llvm.org/docs/CodingStandards.html#ci_rtti_exceptions。

55 第 10 章绘制了数据结构示意图，第 11 章补充了 lower_bound 和 upper_bound 的示例。

56 通过 Scott Meyers 的讲义可以快速学习 C++11（http://www.artima.com/shop/overview_of_the_new_cpp）。

To Beth,
who makes this,
and all things,
possible.

To Daniel and Anna,
who contain
virtually
all possiblities.
—SBL

To Mark and Mom,
for their
unconditional love and support.
—JL

To Andy,
who taught me
to program
and so much more.
—BEM

Preface

C++ Primer, Fourth Edition, provides a comprehensive introduction to the C++ language. As a primer, it provides a clear tutorial approach to the language, enhanced by numerous examples and other learning aids. Unlike most primers, it also provides a detailed description of the language, with particular emphasis on current and effective programming techniques.

Countless programmers have used previous editions of *C++ Primer* to learn C++. In that time C++ has matured greatly. Over the years, the focus of the language—and of C++ programmers—has grown beyond a concentration on runtime efficiency to focus on ways of making *programmers* more efficient. With the widespread availability of the standard library, it is possible to use and learn C++ more effectively than in the past. This revision of the *C++ Primer* reflects these new possiblities.

Changes to the Fourth Edition

In this edition, we have completely reorganized and rewritten the *C++ Primer* to highlight modern styles of C++ programming. This edition gives center stage to using the standard library while deemphasizing techniques for low-level programming. We introduce the standard library much earlier in the text and have reformulated the examples to take advantage of library facilities. We have also streamlined and reordered the presentation of language topics.

In addition to restructuring the text, we have incorporated several new elements to enhance the reader's understanding. Each chapter concludes with a Chapter Summary and glossary of Defined Terms, which recap the chapter's most important points. Readers should use these sections as a personal checklist: If you do not understand a term, restudy the corresponding part of the chapter.

We've also incorporated a number of other learning aids in the body of the text:

- Important terms are indicated in **bold**; important terms that we assume are already familiar to the reader are indicated in ***bold italics***. Each term appears in the chapter's Defined Terms section.

- Throughout the book, we highlight parts of the text to call attention to important aspects of the language, warn about common pitfalls, suggest good programming practices, and provide general usage tips. We hope that these notes will help readers more quickly digest important concepts and avoid common pitfalls.

- To make it easier to follow the relationships among features and concepts, we provide extensive forward and backward cross-references.

- We have provided sidebar discussions that focus on important concepts and supply additional explanations for topics that programmers new to C++ often find most difficult.

- Learning any programming language requires writing programs. To that end, the primer provides extensive examples throughout the text. Source code for the extended examples is available on the Web at the following URL:

```
http://www.awprofessional.com/cpp_primer
```

What hasn't changed from earlier versions is that the book remains a comprehensive tutorial introduction to C++. Our intent is to provide a clear, complete and correct guide to the language. We teach the language by presenting a series of examples, which, in addition to explaining language features, show how to make the best use of C++. Although knowledge of C (the language on which C++ was originally based) is not assumed, we do assume the reader has programmed in a modern block-structured language.

Structure of This Book

C++ Primer provides an introduction to the International Standard on C++, covering both the language proper and the extensive library that is part of that standard. Much of the power of C++ comes from its support for programming with abstractions. Learning to program effectively in C++ requires more than learning new syntax and semantics. Our focus is on how to use the features of C++ to write programs that are safe, that can be built quickly, and yet offer performance comparable to the sorts of low-level programs often written in C.

C++ is a large language and can be daunting to new users. Modern C++ can be thought of as comprising three parts:

- The low-level language, largely inherited from C
- More advanced language features that allow us to define our own data types and to organize large-scale programs and systems
- The standard library, which uses these advanced features to provide a set of useful data structures and algorithms

Most texts present C++ in this same order: They start by covering the low-level details and then introduce the the more advanced language features. They explain the standard library only after having covered the entire language. The result, all too often, is that readers get bogged down in issues of low-level programming or the complexities of writing type definitions and never really understand the power of programming in a more abstract way. Needless to say, readers also often do not learn enough to build their own abstractions.

In this edition we take a completely different tack. We start by covering the basics of the language and the library together. Doing so allows you, the reader, to write significant programs. Only after a thorough grounding in using the library—and writing the kinds of abstract programs that the libary allows—do we move on to those features of C++ that will enable you to write your own abstractions.

Parts I and II cover the basic language and library facilities. The focus of these parts is to learn how to write C++ programs and how to use the abstractions from the library. Most C++ programmers need to know essentially everything covered in this portion of the book.

In addition to teaching the basics of C++, the material in Parts I and II serves another important purpose. The library facilities are themselves abstract data types written in C++. The library can be defined using the same class-construction features that are available to any C++ programmer. Our experience in teaching C++ is that by first using well-designed abstract types, readers find it easier to understand how to build their own types.

Parts III through V focus on how we can write our own types. Part III introduces the heart of C++: its support for classes. The class mechanism provides the basis for writing our own abstractions. Classes are also the foundation for object-oriented and generic programming, which we cover in Part IV. The *Primer* concludes with Part V, which covers advanced features that are of most use in structuring large, complex systems.

Acknowledgments

As in previous editions of this *Primer*, we'd like to extend our thanks to Bjarne Stroustrup for his tireless work on C++ and for his friendship to these authors throughout most of that time. We'd also like to thank Alex Stepanov for his original insights that led to the containers and algorithms that form the core of the standard library. Finally, our thanks go to the C++ Standards committee members for their hard work in clarifying, refining, and improving C++ over many years.

We also extend our deep-felt thanks to our reviewers, whose helpful comments on multiple drafts led us to make improvements great and small throughout the book: Paul Abrahams, Michael Ball, Mary Dageforde, Paul DuBois, Matt Greenwood, Matthew P. Johnson, Andrew Koenig, Nevin Liber, Bill Locke, Robert Murray, Phil Romanik, Justin Shaw, Victor Shtern, Clovis Tondo, Daveed Vandevoorde, and Steve Vinoski.

This book was typeset using LaTeX and the many packages that accompany the LaTeX distribution. Our well-justified thanks go to the members of the LaTeX community, who have made available such powerful typesetting tools.

The examples in this book have been compiled on the GNU and Microsoft compilers. Our thanks to their developers, and to those who have developed all the other C++ compilers, thereby making C++ a reality.

Finally, we thank the fine folks at Addison-Wesley who have shepherded this edition through the publishing process: Debbie Lafferty, our original editor, who initiated this edition and who had been with the *Primer* from its very first edition; Peter Gordon, our new editor, whose insistence on updating and streamlining the text have, we hope, greatly improved the presentation; Kim Boedigheimer, who keeps us all on schedule; and Tyrrell Albaugh, Jim Markham, Elizabeth Ryan, and John Fuller, who saw us through the design and production process.