# 从《C++ Primer 第四版》入手学习 C++

陈硕 (giantchen@gmail.com)

最后更新 2012-7-11



## 1 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++ 的惟一理由是其性能。相比之下，Java/C#/Python 等语言更加易学易用并且开发工具丰富，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（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。

[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，请见 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。http://www.cnblogs.com/miloyip/archive/2010/09/17/behind_cplusplus.html

[5] 孟岩《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 (clean up) 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。
http://blog.csdn.net/myan/article/details/1906

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) 方面的代价 [10]。用本书作者的话来说，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。

```
inline int find_longest(const std::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 (locality of reference) [14] 并充分利用内存阶层 (memory hierarchy) 提速。可参考 Scott Meyers 的 PPT《CPU Caches and Why You Care》、Herb Sutter 的 PPT《Machine Architecture》[15] 和任

[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++?》
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)。

[10] 我曾向 Stan 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 1970 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。
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》讲义
http://www.artima.com/shop/effective_cpp_in_an_embedded_environment

[14] 我们知道 `std::list` 的任一位置插入是 O(1) 操作，而 `std::vector` 的任一位置插入是 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`。见 http://ecn.channel9.msdn.com/events/GoingNative12/GN12Cpp11Style.pdf ，这也佐证 `std::vector` 是首选容器。

[15] 分别位于 http://aristeia.com/TalkNotes/ACCU2011_CPUCaches.pdf
http://www.nwcpp.org/Downloads/2007/Machine_Architecture_-_NWCPP.pdf

何一本现代的计算机体系结构教材（《计算机体系结构：量化研究方法》、《计算机组成与设计：硬件/软件接口》、《深入理解计算机系统》等）。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。

## 2 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的书，这样的书必定会是一本八九百页的大部头 [18]。

兼具系统性和权威性 [19] 的 C++ 教材有两本，C++ 之父 Bjarne Stroustrup 的代表作《The C++ Programming Language》和 Stan Lippman 的这本《C++ Primer》。侯捷先生评价道 [20]：“泰山北斗已现，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。”

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，我十年前深入学习 C++ 正是用的《C++ Primer 第三版》。这次借评注的机会仔细阅读了《C++ Primer 第四版》，感觉像在读一本完全不同的新书。第四版内容组织及文字表达比第三版进步很多 [21]，第三版可谓“事无巨细、面面俱到”，第四版重点突出详略得当，甚至篇幅也缩短了，这多半归功于新加盟的作者 Barbara Moo。

[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。http://www2.research.att.com/ bs/abstraction-and-machine.pdf

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

[18] 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。

[19] “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。

[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.

## 2.1 《C++ Primer 第四版》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 章可用作自测题。

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 第四版》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 第四版》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。

尽管本书篇幅近 900 页，其内容还是十分紧凑，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 第四版》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。这本书在第三章介绍了 `std::string` 和 `std::vector` 这两个常用的类，立刻就能写出很多有用的程序。作者没有一次性把 `std::string` 的上百个成员函数一一列举，而是有选择地先讲解了最常用的那几个函数，更好地体现了本书作为教材而不是手册的用途。

[22] 侯捷《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)》

[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...

《C++ Primer 第四版》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 第四版》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 第四版》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，GNU G++ 和微软 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

[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 第五版》，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。
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。http://stackoverflow.com/questions/3931312

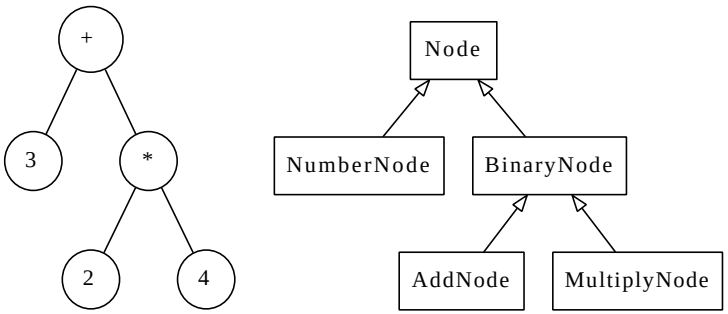
[32] G++ 是免费的，可使用较新的 4.x 版，最好 32-bit 和 64-bit 一起用，因为服务端已经普及 64-bit 编程。微软也有免费的 C++ 编译器，可考虑用 Visual C++ 2010 Express，建议不要用老掉牙的 Visual C++ 6.0 作为学习平台。

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]，实现基本的加减乘运算，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 `std::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。



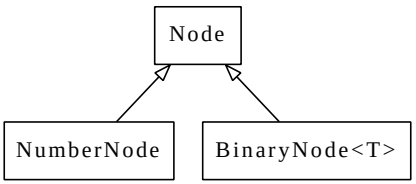

在写完面向对象版的表达式树之后，还可以略微尝试泛型编程。比如把类的继承体系简化为下图，然后用 `BinaryNode<std::plus<double> >` 和 `BinaryNode<std::multiplies<double> >` 来具现化 `BinaryNode<T>` 类模板，通过控制模板参数的类型

[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。

[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

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功夫，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）。

# 3 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++ 第三版》[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。

[36] 孟岩《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。

[38] Scott Meyers 著，侯捷译，电子工业出版社。

我假定读者一定会阅读这本书，因此在评注中不引用《Effective C++ 第三版》的任何章节。

《Effective C++ 第三版》的内容也反映了 C++ 用法的进步。第二版建议“总是让基类拥有虚析构函数”，第三版改为“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。第二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]。第三版则提议，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++ 里根本不是困难（见注 5），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惟一可能是循环引用，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++ 第三版》第 3 章资源管理。

C++ 是目前惟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]。

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]，可以考虑读一读

[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 `std::string`、`std::vector`、`boost::shared_ptr` 等等，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] 孟岩《function/bind 的救赎（上）》http://blog.csdn.net/myan/article/details/5928531

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

[46] 最好再学一点基础的离散数学。

《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。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），不是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。制定一份好的编程规范相当不容易。规范定得太紧（比如定为团队成员知识能力的交集），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]。规范定得太松（定为团队成员知识能力的并集），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，这本书篇幅比《Effective C++ 第三版》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。

[47] Matthew Austern 著，侯捷译，中国电力出版社

[48] Herb Sutter 等著，刘基诚译，人民邮电出版社。（这本书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] 孟岩《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] http://www.math.pku.edu.cn/teachers/qiuzy/technotes/expression2009.pdf

- 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：

- 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。

# 4 评注版使用说明

评注版采用大开本印刷，在保留原书板式的前提下，对原书进行了重新分页，评注的文字与正文左右分栏并列排版。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和术语表还有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

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

[54] 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 lower_bound 和 upper_bound 的示例。

[56] 从 Scott Meyers 的讲义可以快速学习 C++11
http://www.artima.com/shop/overview_of_the_new_cpp

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。

## 4.1 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/
本文电子版发布于 https://github.com/chenshuo/documents/downloads/LearnCpp.pdf，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

评注者简介：

略