HUSTP

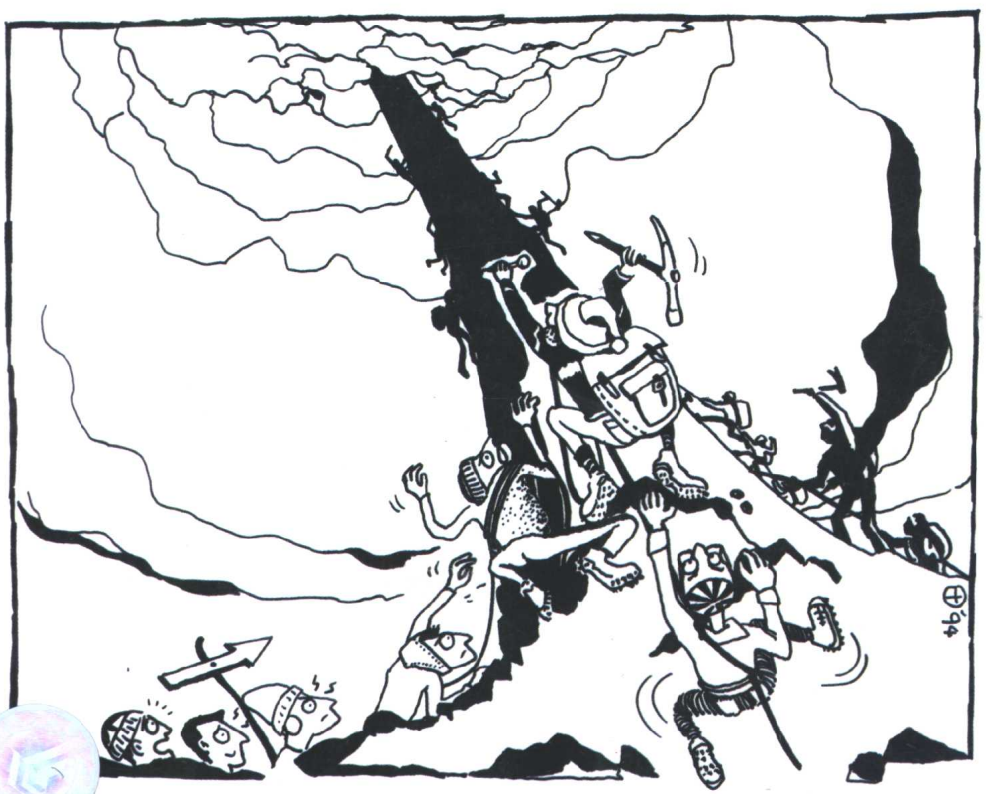
无限延伸你的视野

庖丁解牛 恢恢乎游刃有余

STL 源码剖析

The Annotated STL Sources (using SGI STL)

向专家学习
型别技术、内存管理、算法、数据结构、STL各类组件
之高阶实现技巧



侯 捷

华中科技大学出版社
http://press.hust.edu.cn

The Annotated STL Sources

SGI STL 源码剖析

The Annotated STL Sources

向专家学习
型别技术、内存管理、算法、数据结构、STL 各类组件
之高阶实现技巧

侯 捷

华中科技大学出版社

图书在版编目(CIP)数据

STL 源码剖析/侯捷
武汉:华中科技大学出版社,2002 年 6 月
ISBN 7-5609-2699-1

Ⅰ.S…
Ⅱ. 侯…
Ⅲ. 计算机软件-STL 源码
Ⅳ. TP311.1

STL 源码剖析 侯捷

责任编辑:周 筠 http://yeka.xilubbs.com junzhou@public.wh.hb.cn
技术编辑:孟 岩
责任校对:张兴田
封面设计:潘 群
责任监印:张正林

出版发行:华中科技大学出版社 武昌喻家山 邮编:430074 电话:(027)87545012

录 排:华中科技大学惠友科技文印中心
印 刷:湖北新华印务有限公司

开本:787×1092 1/16 印张:33 插页:2 字数:600 000
版次:2002 年 6 月第 1 版 印次:2002 年 6 月第 1 次印刷 印数:1—10 000
ISBN 7-5609-2699-1/TP·464 定价:68.00 元

源码之前
了无秘密

献给每一位对 GP/STL 有所渴望的人

天下大事　必作于细

－ 侯 捷 －

高屋建瓴 细致入微

——《STL 源码剖析》引介

身为C++标准库最重要的组成部分，STL（标准模板库）不仅是一个可复用组件库，而且是一个包罗算法与数据结构的软件框架（framework）。“框架”这个词，本身就有庞大、稳定、完整而可扩展的涵义。软件框架，则是用一行行精细准确的源码，构造一个庞大、稳定、完整而可扩展的软件架构。稍有软件开发经验的人都知道，要做到这些，谈何容易！ STL 在 1994 年走入 C++标准，使得原本即将推出的 C++标准延迟 4 年问世而无怨无悔，并为之对内容做巨幅改进。而今 STL 不仅为千千万万 C++程序员所日常运用，而且获得极高的学术赞誉，成为一个典范、一种境界。作为一个软件框架，STL 所取得的成功，实在可以用“辉煌”来形容，其所内涵的软件思想和技术经验，更是无比的深厚与精致。

学习编程的人都知道，阅读、剖析名家代码乃是提高水平的捷径。源码之前，了无秘密。大师们的缜密思维、经验结晶、技术思路、独到风格，都原原本本体现在源码之中。在你仔细推敲之中，迷惑不解之时，恍然大悟之际，你的经验、思维、视野、知识乃至技术品位都会获得快速的成长。特别是面对 STL 这样优秀而普遍的作品，无论你是为了满足作为程序员第二天性的求知欲，还是在日常工作中解决实际问题，总有一天，你会打开一个叫做<vector>或者<algorithm>的头文件，想把 STL 背后的秘密看个究竟。英文里有一个常用短语，叫做“under the hood”，钻进魔术师的帐篷，屏住呼吸，瞪大眼睛，把那些奇妙的魔法看个通透，让自己的理解和技艺获得巨幅的提升，这种诱惑，任何一个程序员都无法抵挡！

不过，要想研读 STL 源码，绝对没有那么简单。STL 是精致的软件框架，是为优化效率而无所不用其极的艺术品，是数据结构与算法大师经年累月的智能结晶，是泛型思想的光辉诗篇，是 C++高级技术的精彩亮相！这些灿烂的赞誉，体现在数万行源码里，对于一个初涉此道的学习者来说，就是一个感觉：“难！”无论你是浅尝辄止，便退出

这次探险，还是勇敢地向浓雾中前进，当你受困于 STL 精致的大网之中，为那些迷一般的结构和操作感到茫然无措的时侯，所有人都会冒出一个念头："如果有这样一本书，既能够提纲挈领，为我理顺思绪，指引方向，同时又能够照顾小节，阐述细微，帮助我更快更好地理解 STL 源码，那该有多好！"

望着长长的 STL 著作列表，一个"真正"的 C++程序员，多少会有一点遗憾。自从 STL 问世以来，出版了大量的书籍，帮助读者了解它的思想，学习它的用法，掌握它的技巧。其中佼佼者如 Matt Austern 的《Generic Programming and STL》，Nicolai Josuttis 的《The C++ Standard Library》，Scott Meyers 的《Effective STL》，已成 C++经典名著。然而，定位在引导学习者进行 STL 源码分析的著作，可以说是凤毛麟角。毕竟，既要能高屋建瓴，剖析大架构，不为纷繁琐碎之细节而迷乱，又能具体而微，体现细致之处的精妙缜密，不因为宏大体系而失之粗略，无论对于专家高手还是技术作家，都是太难达到的目标。

读了这本《STL 源码剖析》之后，我认为，这个遗憾终于被补足了！

本书的作者侯捷先生是蜚声海峡两岸的著名 IT 技术作家，在 C++、Windows 系统原理、泛型理论和 STL 等技术领域有极深的造诣。然而，侯先生最令人称道之处，乃是他剖析大架构的能力。所谓剖析大架构，就是要在洋洋洒洒数以万行计的源码中，精准定位，抽取核心观念，高屋建瓴，纲举目张，将看上去乱麻一般的源码梳理得头绪清晰，条理分明，同时又照顾细节，参透精微，把一个个关键动作阐述得通通透透。这种能力，我以为至少在华人技术作家中，侯先生堪执牛耳！在他的名作《深入浅出 MFC》中，侯先生将自己这方面的能力展现得淋漓尽致，而在这本《STL 源码剖析》中，我们看到了又一次更加精彩的表现。

我有机会作为大陆最早的几个读者之一，详细拜读了侯先生的这本 STL 专着，内心产生了一种强烈的技术冲动。说得俗一点，就是觉得很过瘾！具体来说，我认为这本书至少有四大特点，使它成为我所见过的最出色的一本 STL 源码剖析类著作。

首先，选材精当，立足高远。STL 是一个标准，因而有各种实现版本。本书所剖析的 SGI STL，可以说是设计最巧妙、思想最深刻、获得赞誉最盛、认同最广的 STL 实现品。当然， 这份出自 STL 之父 Alex Stepanov，以及 Matt Austern，David Musser 等巨匠之手的经典作品，剖析阐述起来自然也需要花费更大的心力。侯先生藉其扎实的理论与技术素养，毅然选择这份作品来剖析，是需要极大勇气与自信的。同样，本书对读者的预期，也是很高的，读者不但要有扎实的基本功，更要有掌握 STL 的兴趣与坚韧意志。读这本书，你可以有充分的信心，学到的是超一流大师的思想和经验，所谓名门正派，高屋建瓴。

其次，脉络清晰，组织顺序匠心独具。任何人打算系统阅读 STL 源码，所必须做出的第一个决定就是，从何处开始？我在初读此书时，一个最感疑惑的地方就是侯先生竟然把 allocator 放在所有组件之前讲述。要知道，allocator 这个东西，对一般的使用者完全透明，根本感觉不到其存在，以至于在名著《The C++ Standard Libaray》中，Nicolai Josuttis 将这一部分放在全书最后。既然如此，又何必让这个无名小卒占据头版头条？我一开始还真是不理解。直到后来，我自己有一些扩展 STL 的实践，才发现，用的时候你固然可以对 allocator 不闻不问，但一旦要领悟 STL 的工作原理，或者要自己扩展 STL 的功能，则对于 allocator 的掌握几乎是第一先决条件。不了解 allocator，则无论剖析也好，扩展也罢，必然处处碰壁。侯先生毫不迟疑，首先帮读者搬开这块绊脚石，理出头绪，实在是匠心独具。紧接着的第三章 iterator 及 traits，直入 STL 的核心观念与关键技术，剑走中锋，直取要害，高举高打，开诚布公，直接把理解 STL 的钥匙交到读者手上。此章一过，读者神完气足，就可以大刀阔斧地打通 STL 的重重关隘。此布局只要稍有变化，读者的学习难度势必猛增。侯先生的此种安排，实在是大家手笔！

此外，本书在技术上迎难而上，详略得当，完整而重点突出。了解 SGI STL 的读者都知道，这份作品对 C++标准中的 STL 做了大量的扩充，增加了专用的高效 allocator，用以操作巨型字符串的 rope，单链表 slist，以及万众企盼的 hash 容器等等，再加上 STL 本身就有很多精微之处，技术上的难点不少。此类书籍的作者，但凡稍有一丝懈怠之心，大可以冠冕堂皇地避重就轻。然而侯先生在此书中对重点难点毫不避讳，无论是标准功能还是非标准功能，只要对读者理解 STL 架构有益，只要有助于提高读者的技术，增长读者的视野与经验，书中必然不畏繁难，将所有技术细节原原本本和盘托出。另一方面，所谓剖析源码，其目的在于明理、解惑，提高自身水平，并不是要穷经皓首，倒背如流。因此，一旦道理讲清楚，书中就将重复与一般性的内容一笔带过，孰轻孰重，一目了然，详略十分得当，这一点对于提高读者的学习效率，有着巨大的意义。

最后一点，本书通过大量生动范例和插图讲解基本思想，在同类书籍中堪称典范。虽然我把这一点放在最后，但我相信大部分读者站在书店，随手翻过这本书，得到的第一印象便是这一点。STL 之所以为大家所津津乐道，除了其思想深刻之外，最大的因素是它实用。它所包装的，是算法与数据结构的基本功能。作为一个程序员，如果你是做数据库编程的，大可以不懂汇编语言，如果你是写驱动程序的，大可以不必通晓人工智能，写编译器的可以不用懂什么计算机图形学，操作系统内核高手不用精通网站架设，然而，如果你不懂数据结构与算法的基础知识，不具备数据结构与算法的基本技能，那就完全丧失称为一个程序员的资格！市面上讲述算法与数据结构的专着汗牛充栋，俯拾皆是。相比之下，本书倒并不是以此为核心目标的。但是，可曾有哪位读者看到任何一

本书像本书一样，将红黑树用一张张清晰生动的图解释得如此浅显易懂？所谓一图胜千言，在教授基本数据结构与算法方面，我想不出还有任何一种方法，能够比幻灯般的图片更生动更令人印象深刻了。读过此书的每一位读者，我想都会为书中那一幅幅插图所打动，作者细致严谨的作风，时刻为读者考虑的敬业精神，也许是更值得我们尊敬的东西。我非常荣幸有机会与侯先生和华中科技大学出版社的周筠女士再次合作，担任了这本书的繁简转译工作。在术语转译方面，我们基本上保持了与《Effective C++ 中文版》相一致的标准。其中有一些术语不完全符合国内的习惯译法，下面是一个简单的对比表：

英文术语	大陆惯用译法	本书译法
adapter	适配器	配接器
argument	实参（实质参数）	引数
by reference	传参考,传地址	传址
by value	传值	传值
dereference	反引用,解参考	提领
evaluate	评估,计算	评估，核定
instance	案例,实例	实体
instantiated	实例化	实体化、具现化
library	库,函数库	程序库
range	范围	区间（使用于 STL 时）
resolve	解析	决议
parameter	形参（形式参数）	参数
type	类型	型别

侯先生在一篇影响颇为广泛的 STL 技术杂文中，将 STL 的学习境界划分为三个阶段：会用，明理，能扩展。阅读 STL 源码是由第一层次直贯第二层次，而渐达于第三层次的一条捷径，当然也是一条满是荆棘之路。如果你是一个勇于征服险峰的程序员，如果你是一个希望了解 under the hood 之奥秘的程序员，那么当你在攀登 STL 这座瑰丽高山的时候，这本书会大大地帮助你。我非常热情地向您推荐这本着作。当然，再好的书籍也只是工具，能不能成功，关键，还在你自己！

孟岩

2002 年 4 月于北京

庖丁解牛[1]

侯捷自序

这本书的写作动机，纯属偶然。

2000 年下半年，我开始为计划中的《泛型思维》一书陆续准备并热身。为了对泛型编程技术以及 STL 实现技术有更深的体会，以便在讲述整个 STL 的架构与应用时更能虎虎生风，我常常深入到 STL 源码中去刨根究底。2001 年 2 月的某一天，我突然有所感触：既然花了大把精力看过 STL 源码，写了眉批，做了整理，何不把它再加一点功夫，形成一个更完善的面貌后出版？对我个人而言，一份批注详尽的 STL 源码，价值不菲；如果我从中获益，一定也有许多人能够从中获益。

这样的念头使我极度兴奋。剖析大架构本是侯捷的拿手，这个主题又可以和《泛型思维》相呼应。于是我便一头栽进去了。

我选择 SGI STL 作为剖析对象。这份实现版本的可读性极佳，运用极广，被选为 GNU C++ 的标准程序库，又开放自由运用。愈是细读 SGI STL 源码，愈令我震惊抽象思维层次的落实、泛型编程的奥妙，及其效率考虑的缜密。不仅最为人广泛运用的各种数据结构（data structures）和算法（algorithms）在 STL 中有良好的实现，连内存配置与管理也都重重考虑了最佳效能。一切的一切，除了实现软件积木的高度复用性，让各种组件（components）得以灵活搭配运用，更考虑了实用上的关键议题：效率。

---

[1] 庄子养生主：“彼节者有间，而刀刃者无厚；以无厚入有间，恢恢乎其于游刃必有余地矣。”侯捷不让，以此自况。

这本书不适合 C++ 初学者，不适合 Genericity（泛型技术）初学者，或 STL 初学者。这本书也不适合带领你学习面向对象（Object Oriented）技术——是的，STL 与面向对象没有太多关联。本书前言清楚说明了书籍的定位和合适的读者，以及各类基础读物。如果你的 Generic Programming/STL 实力足以阅读本书所呈现的源码，那么，恭喜，你踏上了基度山岛，这儿有一座大宝库等着你。源码之前了无秘密，你将看到 vector 的实现、list 的实现、heap 的实现、deque 的实现、RB-tree 的实现、hash-table 的实现、set/map 的实现；你将看到各种算法（排序、搜寻、排列组合、数据移动与复制……）的实现；你甚至将看到底层的 memory pool 和高阶抽象的 traits 机制的实现。那些数据结构、那些算法、那些重要观念、那些编程实务中最重要最根本的珍宝，那些蜇伏已久仿佛已经还给老师的记忆，将重新在你的大脑中闪闪发光。

人们常说，不要从轮子重新造起，要站在巨人的肩膀上。面对扮演轮子角色的这些 STL 组件，我们是否有必要深究其设计原理或实现细节呢？答案因人而异。从应用的角度思考，你不需要探索实现细节（然而相当程度地认识底层实现，对实务运用有绝对的帮助）。从技术研究与本质提升的角度来看，深究细节可以让你彻底掌握一切；不论是为了重温数据结构和算法，或是想要扮演轮子角色，或是想要进一步扩张别人的轮子，都可因此获得深厚扎实的基础。

天下大事，必作于细！

参观飞机工厂不能让你学到流体力学，也不能让你学会开飞机。然而如果你会开飞机又懂流体力学，参观飞机工厂可以带给你最大的乐趣和价值。

The Annotated STL Sources

我开玩笑地对朋友说，这本书的出版，给大学课程中的“数据结构”和“算法”两门授课老师出了个难题。几乎对所有可能的作业题目（复杂度证明题除外），本书都有了详尽的解答。然而，如果学生能够从庞大的 SGI STL 源码中干净抽出某一部分，加上自己的包装，做为呈堂作业，也足以证明你有资格获得学分和高分。事实上，追踪一流作品并于其中吸取养分，远比自己关起门来写个三流作品，价值高得多——我的确认为 99.99 % 的程序员所写的程序，在 SGI STL 面前都是三流水准 ☺。

侯捷 2002/03/30 新竹 • 台湾

http://www.jjhou.com （繁体）
http://jjhou.csdn.net （简体）
jjhou@jjhou.com

p.s. 以下三本书互有定位，互有关联，彼此亦相呼应。为了不重复讲述相同的内容，我会在适当时候提醒读者在哪本书上获得更多资料：

- 《多态与虚拟》，内容涵括：C++ 语法、语意、对象模型，面向对象精神，小型 framework 实现，OOP 专家经验，设计模式（design patterns）导引。
- 《泛型思维》，内容涵括：语言层次（C++ templates 语法、Java generic 语法、C++ 操作符重载），STL 原理介绍与架构分析，STL 现场重建，STL 深度应用，STL 扩充示范，泛型思考。
- 《STL 源码剖析》，内容涵括：STL 所有组件之实现技术和其背后原理解说。

前言

本书定位

C++ 标准程序库是一个伟大的作品。它的出现，相当程度上改变了 C++ 程序的风貌以及学习模式[1]。纳入 STL（Standard Template Library）的同时，标准程序库的所有组件，包括大家早已熟悉的 string、stream 等等，亦全部以 template 覆盖过。整个标准程序库没有太多的 OO（Object Oriented），倒是无处不存在 GP（Generic Programming）。

C++ 标准程序库中隶属 STL 范围者，粗估当在 80% 以上。对软件开发而言，STL 是尖甲利兵，可以节省你许多时间。对编程技术而言，STL 是金柜石室——所有与编程工作最有直接密切关联的一些最被广泛运用的数据结构和算法，STL 都有实现，并符合最佳（或极佳）效率。不仅如此，STL 的设计思维，把我们提升到另一个思想高点，在那里，对象的耦合性（coupling）极低，复用性（reusability）极高，各种组件可以独立设计又可以灵活无罅地结合在一起。是的，STL 不仅仅是程序库，它其实具备 framework 格局，允许使用者加上自己的组件，与之融合并用，是一个符合开放性封闭（Open-Closed）原则的程序库。

从应用角度来说，任何一位 C++ 程序员都不应该舍弃现成、设计良好而又效率极佳的标准程序库，却“人太庙每事问”地事事物物从轮子造起——那对组件技术及软件工程将是一大嘲讽。然而，对于一个想要深度钻研 STL 以便拥有扩充能力

1 请参考 *Learning Standard C++ as a New Language*, by Bjarne Stroustrup, C/C++ Users Journal 1999/05。中译文 http://www.jjhou.com/programmer-4-learning-standard-cpp.htm

的人来说，相当程度地追踪 STL 源代码是必要的功课。是的，对于一个想要充实数据结构与算法等固有知识，并提升泛型编程技法的人，“人太庙每事问”是必要的态度，追踪 STL 源代码则是提升功力的极佳路线。

想要良好运用 STL，我建议你看《*The C++ Standard Library*》by Nicolai M. Josuttis；想要严谨认识 STL 的整体架构和设计思维，以及 STL 的详细规格，我建议你看 《*Generic Programming and the STL*》by Matthew H. Austern；想要从语法层面开始，学理与应用得兼，宏观与微观齐备，我建议你看《泛型思维》by 侯捷；想要深入 STL 实现技法，一窥大家风范，提升自己的编程功力，我建议你看你手上这本《STL 源码剖析》——事实上就在下笔此刻，你也找不到任何一本相同定位的书[2]。

合适的读者

本书不适合 STL 初学者（当然更不适合 C++ 初学者）。本书不是面向对象（Object Oriented）相关书籍。本书不适合用来学习 STL 的各种应用。

对于那些希望深刻了解 STL 实现细节，从而得以提升对 STL 的扩充能力，或是希望藉由观察 STL 源代码，学习世界一流程序员身手，并藉此彻底了解各种被广泛运用之数据结构和算法的人，本书最适合你。

最佳阅读方式

无论你对 STL 认识了多少，我都建议你第一次阅读本书时，采取循序渐进的方式，遵循书中安排的章节先行浏览一遍。视个人功力的深浅，你可以或快或慢并依个人兴趣或需要，深入其中。初次阅读最好循序渐进，理由是，举个例子，所有容器（containers）的定义式一开头都会出现空间配置器（allocator）的运用，我可以在最初数次提醒你空间配置器于第 2 章介绍过，但我无法遍及全书一再一再提醒你。又例如，源代码之中时而会出现一些全局函数调用操作，尤其是定义于 `<stl_construct.h>` 之中用于对象构造与析构的基本函数，以及定义于

[2] *The C++ Standard Template Library*, by P.J.Plauger, Alexander Al. Stepanov, Meng Lee, David R. Musser, Prentice Hall 2001/03，与本书定位相近，但在表现方式上大有不同。

<stl_uninitialized.h> 之中用于内存管理的基本函数，以及定义于 <stl_algobase.h> 之中的各种基本算法。如果那些全局函数已经在先前章节中介绍过，我很难保证每次都提醒你——那是一种顾此失彼、苦不堪言的劳役，并且容易造成阅读上的累赘。

我所选择的剖析对象

本书名为《STL 源码剖析》，然而 STL 实现版本百花齐放，不论就技术面或可读性，皆有高下之分。选择一份好的实现版本，就学习而言当然是极为重要的。我选择的剖析对象是声名最盛，也是我个人评价最高的一个产品：SGI（Silicon Graphics Computer Systems, Inc.）版本。这份由 STL 之父 Alexander Stepanov、经典书籍《*Generic Programming and the STL*》作者 Matthew H. Austern、STL 巨匠 David Musser 等人投注心力的 STL 实现版本，不论在技术层次、源代码组织、源代码可读性上，均有卓越的表现。这份产品被纳为 GNU C++ 标准程序库，任何人皆可从因特网上下载 GNU C++ 编译器，从而获得整份 STL 源代码，并获得自由运用的权力（详见 1.8 节）。

我所选用的是 cygnus[3] C++ 2.91.57 for Windows 版本。我并未刻意追求最新版本，一来书籍不可能永远呈现最新的软件版本——软件更新永远比书籍改版快速，二来本书的根本目的在于建立读者对于 STL 宏观架构和微观技术的掌握，以及源代码的阅读能力，这种核心知识的形成与源代码版本的关系不是那么唇齿相依，三来 SGI STL 实作品自从搭配 GNU C++2.8 以来已经十分稳固，变异极微，而我所选择的 2.91 版本，表现相当良好；四来这个版本的源代码比后来的版本更容易阅读，因为许多内部变量名称并不采用下划线（underscore）——下划线在变量命名规范上有其价值，但到处都是下划线则对大量阅读相当不利。

网络上有一个 STLport（http://www.stlport.org）站点，提供一份以 SGI STL 为蓝本的高度可移植性实现版本。本书附录 C 列有孟岩先生所写的文章，是一份 STLport 移植到 Visual C++ 和 C++ Builder 的经验谈。

---

3 关于 cygnus、GNU 源代码开放精神，以及自由软件基金会（FSF），请见 1.3 节介绍。

各章主题

本书假设你对 STL 已有基本认识和某种程度的运用经验。因此，除了第一章略作介绍之外，立刻深入 STL 技术核心，并以 STL 六大组件（components）为章节的进行依据。以下是各章名称，这样的次序安排大抵可使每一章所剖析的主题能够于先前章节中获得充分的基础。当然，技术之间的关联错综复杂，不可能存在单纯的线性关系，这样的安排也只能说是尽最大努力。

第 1 章　STL 概论与实现版本简介

第 2 章　空间配置器（allocator）

第 3 章　迭代器（iterators）概念与 traits 编程技法

第 4 章　序列式容器（sequence containers）

第 5 章　关联式容器（associated containers）

第 6 章　算法（algorithms）

第 7 章　仿函数或函数对象（functors, or function objects）

第 8 章　配接器（adapter）

编译工具

本书主要探索 SGI STL 源代码，并提供少量测试程序。如果测试程序只做标准的 STL 操作，不涉及 SGI STL 实现细节，那么我会在 VC6、CB4、cygnus 2.91 for Windows 等编译平台上分别测试它们。

随着对 SGI STL 源代码的掌握程度增加，我们可以大胆做些练习，将 SGI STL 内部接口打开，或是修改某些 STL 组件，加上少量输出操作，以观察组件的运作过程。这种情况下，操练的对象既然是 SGI STL，我也就只使用 GNU C++ 来编译[4]。

[4] SGI STL 事实上是个高度可移植性的产品，不限使用于 GNU C++。从它对各种编译器的环境组态设定（1.8.3 节）便可略知一二。网络上有一个 STLport 组织，不遗余力地将 SGI STL 移植到各种编译平台上。请参阅本书附录 C。

中英术语的运用风格

我曾经发表过一篇题为“技术引导乎 文化传承乎”的文章，阐述我对专业计算机书籍的中英术语运用态度。文章收录于侯捷网站 http://www.jjhou.com/article99-14.htm。以下简单叙述我的想法。

为了学术界和业界的习惯，也为了与全球科技接轨，并且也因为我所撰写的是供专业人士阅读的书籍而非科普读物，我决定适量保留专业领域中被朗朗上口的英文术语。朗朗上口与否，见仁见智，我以个人阅历作为抉择依据。

作为一个并非以英语为母语的族裔，我们对英文的阅读困难并不在单字，而在整句整段的文意。作为一项技术的学习者，我们的困难并不在术语本身（那只是个符号），而在术语背后的技术意义。

熟悉并使用原文术语，至为重要。原因很简单，在科技领域里，你必须与全世界接轨。中文技术书籍的价值不在于“建立本国文化”或“让它成为一本道地的中文书”或“完全扫除英汉字典的需要”。中文技术书籍的重要价值，在于引进技术，引导学习，扫平阅读障碍，增加学习效率。

绝大部分我所采用的英文术语都是名词，但极少数动词或形容词也有必要让读者知道原文（我会时而中英并列，并使用斜体英文），原因是：

- C++ 编译器的错误信息并未中文化，万一错误信息中出现以下字眼：*unresolved*, *instantiated*, *ambiguous*, *override*，而编写程序的你却不熟悉或不懂这些动词或形容词的技术意义，就不妙了。
- 有些操作关系到 library functions，而 library functions 的名称并未中文化☺，例如 *insert*, *delete*, *sort*。因此，视情况而定，我可能会选择使用英文。
- 如果某些术语关系到语言关键词，为了让读者有最直接的感受与联想，我会采用原文，例如 static, private, protected, public, friend, inline, extern。

版面像一张破碎的脸？

大量中英文夹杂的结果，无法避免造成版面的 “破碎”。但为了实现合宜的

表达方式，牺牲版面的 “全中文化” 在所难免。我将尽量以版面手法来达到视觉上的顺畅，换言之，我将采用不同的字形来代表不同属性的术语。如果把英文术语视为一种符号，这些中英夹杂但带有特殊字形的版面，并不会比市面上琳琅满目的许多应用软件图解使用手册来得突兀（而后者不是普遍为大众所喜爱吗☺）。我所采用的版面，都已经过一再试验，获得许多读者的赞同。

英文术语采用原则

就我的观察，人们对于英文词或中文词的采用，隐隐有一个习惯：如果中文词发音简短（或至少不比英文词繁长）并且意义良好，那么就比较有可能被业界用于日常沟通；否则业界多半采用英文词。

例如，polymorphism 音节过多，所以意义良好的中文词 “多态” 就比较有机会被采用。例如，虚函数的发音不比 virtual function 繁长，所以使用这个中文词的人也不少。 “多载” 或 “重载” 的发音比 overloaded 短得多，意义又正确，用的人也不少。

但此并非绝对法则，否则就不会有绝大多数工程师说 data member 而不说 “数据成员”、说 member function 而不说 “成员函数” 的情况了。

以下是本书采用原文术语的几个简单原则。请注意，并没有绝对的实践，有时候要看上下文情况。同时，容我再强调一次，这些都是基于我与业界和学界的接触经验而做的选择。

- 编程基础术语，采用中文。例如：函数、指针、变量、常数。本书的英文术语绝大部份都与 C++/OOP/GP（Generic Programming）相关。
- 简单而朗朗上口的词，视情况可能直接使用英文：input, output, lvalue, rvalue...
- 读者有必要认识的英文名词，不译：template, class, object, exception, scope, namespace。
- 长串、有特定意义、中译名称拗口者，不译：explicit specialization, partial specialization, using declaration, using directive, exception specialization。
- 操作符名称，不译：copy assignment 操作符，member access 操作符，arrow 操作符，dot 操作符，address of 操作符，dereference 操作符……