

前言

本书定位

C++ 标准程序库是个伟大的作品。它的出现，相当程度地改变了 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“入太庙每事问”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 以便拥有扩充能力的人，

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>

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，“入太庙每事问”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，宏观与微观兼备，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 源代码剖析》——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，俾得以提升对 STL 的扩充能力，或是希望藉由观察 STL 源代码，学习世界一流程式员身手，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，采循序渐进方式，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动作，尤其是定义于 `<stl_construct.h>` 之中用于对象建构与解构的基本函数，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 源代码剖析》，然而 STL 实作品百花齐放，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，任何人皆可从网际网络上下载 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 cygnus³ 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）为章节之进行依据。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份的基础。当然，技术之间的关连错综复杂，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 动作，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，加上少量输出动作，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 事实上是个高度可携产品，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《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.jjhq.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，我以个人阅历做为抉择依据。

做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。做为项技术的学习者，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。

绝大部份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++ 编译器的错误讯息并未中文化，万一错误讯息中出现以下字眼：*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。
- 有些动作关系到 *library functions*，而 *library functions* 的名称并未中文化[Ⓞ]，例如 *insert, delete, sort*。因此视状况而定，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”就比较容易会被采用。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分都与 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...

- 业界惯用词，不译：constructor, destructor, data member, member function, reference。
- 涉及 C++ 关键词者，不译：public, private, protected, friend, static,
- 意义良好，发音简短，流传颇众的译词，译之：多态 (polymorphism)，虚函数 (virtual function)、泛型 (genericity) ...
- 译后可能失掉原味而无法完全彰显原味者，中英并列。
- 重要的动词、形容词，时而中英并列：模棱两可 (ambiguous), 决议 (resolve), 覆盖 (override), 参数推导 (argument deduced), 具现化 (instantiated)。
- STL 专用术语：采用中文，如迭代器 (iterator)、容器 (container)、仿函数 (functor)、配接器 (adapter)、空间配置器 (allocator)。
- 数据结构专用术语：尽量采用英文，如 vector, list, deque, queue, stack, set, map, heap, binary search tree, RB-tree, AVL-tree, priority queue.

援用英文词，或不厌其烦地中英并列，获得的一项重要福利是：本书得以英文做为索引凭借。

<http://www.jjhou.com/terms.txt> 列有我个人整理的一份英中繁简术语对照表。

版面字型风格

中文

- 本文：细明 9.5pt
- 标题：华康粗圆
- 视觉加强：华康中黑

英文

- 一般文字，Times New Roman, 9.5pt，例如：class, object, member function, data member, base class, derived class, private, protected, public, reference, template, namespace, function template, class template, local, global
- 动词或形容词，Times New Roman 斜体 9.5pt 例如：resolve, ambiguous, override, instantiated
- class 名称，Lucida Console 8.5pt，例如：stack, list, map
- 程序代码识别符号，Courier New 8.5pt，例如：int, min(SmallInt*, int)

- 长串术语, *Arial 9pt*, 例如: member initialization list, name return value, using directive, using declaration, pass by value, pass by reference, function try block, exception declaration, exception specification, stack unwinding, function object, class template specialization, class template partial specialization...
- exception types 或 iterator types 或 iostream manipulators, *Lucida Sans 9pt*, 例如: bad_alloc, back_inserter, boolalpha
- 操作符名称及某些特殊动作, *Footlight MT Light 9.5pt*, 例如: copy assignment 操作符, dereference 操作符, address of 操作符, equality 操作符, function call 操作符, constructor, destructor, default constructor, copy constructor, virtual destructor, memberwise assignment, memberwise initialization
- 程序代码, *Courier New 8.5pt*, 例如:

```
#include <iostream>
using namespace std;
```

要在整本书中维护一贯的字形风格而没有任何疏漏, 很不容易, 许多时候不同类型的术语搭配起来, 就形成了不知该用哪种字形的困扰。排版者顾此失彼的可能也不是没有。因此, 请注意, 各种字形的运用, 只是为了让您阅读时有比较好的效果, 其本身并不具其它意义。局部的一致性更重于全体的一致性。

源代码形式与下载

SGI STL 虽然是可读性最高的一份 STL 源代码, 但其中并没有对实现程序乃至至于实现技巧有什么文字注释, 只偶而在文件最前面有一点点总体说明。虽然其符号名称有不错的规划, 真要仔细追踪源代码, 仍然旷日费时。因此本书不但在正文之中解说其设计原则或实现技术, 也直接在源代码中加上许多注释。这些注释皆以蓝色标示。条件式编译 (#ifdef) 视同源代码处理, 函数调用动作以红色表示, 嵌套定义亦以红色表示。classes 名称、data members 名称和 member functions 名称大多以粗体表示。特别需要提醒的地方 (包括 template 缺省参数、长度很长的嵌套定义式) 则加上灰阶底纹。例如:

```
template <class T, class Alloc = alloc> // 缺省使用 alloc 为配置器
class vector {
public:
    typedef T value_type;
```

```

    typedef value_type* iterator;
    ...
protected:
    // vector 采用简单的线性连续空间。以两个迭代器 start 和 end 分别指向头尾，
    // 并以迭代器 end_of_storage 指向容量尾端。容量可能比(尾-头)还大，
    // 多余的空间即备用空间。
    iterator start;
    iterator finish;
    iterator end_of_storage;

    void fill_initialize(size_type n, const T& value) {
        start = allocate_and_fill(n, value); // 配置空间并设初值
        finish = start + n; // 调整水位
        end_of_storage = finish; // 调整水位
    }
    ...
};

#ifdef __STL_FUNCTION_TMPL_PARTIAL_ORDER
template <class T, class Alloc>
inline void swap(vector<T, Alloc>& x, vector<T, Alloc>& y) {
    x.swap(y);
}
#endif /* __STL_FUNCTION_TMPL_PARTIAL_ORDER */

```

又如：

```

// 以下配接器用来表示某个 Adaptable Binary Predicate 的逻辑负值
template <class Predicate>
class binary_negate
    : public binary_function<typename Predicate::first_argument_type,
                          typename Predicate::second_argument_type,
                          bool> {
    ...
};

```

这些作法可能在某些地方有少许例外（或遗漏），唯一不变的原则就是尽量设法让读者一眼抓住源代码重点。花花绿绿的颜色乍见之下或许不习惯，多看几眼你就会喜欢它⁵。这些经过注释的 SGI STL 源代码以 Microsoft Word 97 文件格式，连同 SGI STL 源代码，置于侯捷网站供自由下载⁵。噢，是的，STL 涵盖面积广大，源代码浩繁，考虑到书籍的篇幅，本书仅能就具代表性者加以剖析，如果你感兴趣的某些细节未涵盖于书中，可自行上网查阅这些经过整理的源代码文件。

⁵ 下载这些文件并不会引发版权问题。详见 1.3 节关于自由软件基金会（FSF）、源代码开放（open source）精神以及各种授权声明。

在线服务

侯捷网站（网址见于封底）是我的个人网站。我的所有作品，包括本书，都在此网站上提供服务，包括：

- 勘误和补充
- 技术讨论
- 程序代码下载
- 电子文件下载

附录 B 对侯捷网站有一些导引介绍。

推荐读物

详见附录 A。这些精选读物可为你建立扎实的泛型（Genericity）思维理论基础与扎实的 STL 实务应用能力。