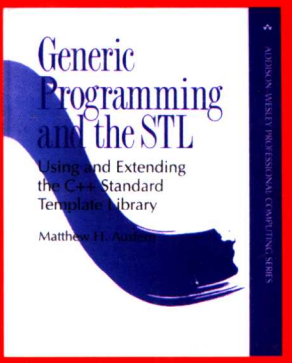
深入C++系列



Generic Programming and the STL

泛型编程与STL

Matthew H. Austern 著
侯 捷 译

Addison Wesley

中国电力出版社
www.infopower.com.cn

深入C++系列

Generic Programming and the STL

泛型编程与 STL

Matthew H. Austern 著
侯 捷 译

中国电力出版社

Generic Programming and the STL: Using and Extending the C++ Standard Template Library (ISBN 0-201-30956-4)

Matthew H. Austern

图书在版编目（CIP）数据

泛型编程与STL/（美）奥斯滕著；侯捷译．—北京：中国电力出版社，2003

（深入C++系列）

ISBN 7-5083-1487-5

Ⅰ.泛… Ⅱ.①奥…②侯… Ⅲ.C语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2003）第017798号

责任编辑： 姚贵胜

丛 书 名： 深入C++系列

书　　名： 泛型编程与STL

编　　著： （美）Matthew H. Austern

翻　　译： 侯捷

出版发行： 中国电力出版社

地址：北京市三里河路6号 邮政编码：100044

电话：（010）88515918 传真：（010）88423191

印　　刷： 北京地矿印刷厂

开　　本： 787×1092 1/16　　**印　　张：** 35.75　　**字　　数：** 806千字

书　　号： 7-5083-1487-5

版　　次： 2003年4月北京第一版

印　　次： 2003年4月第一次印刷

印　　数： 1～6000 册

定　　价： 72.00 元

译序（侯 捷）

泛型编程（generic programming）和其第一份重要实现品 STL（Standard Template Library），对许多人而言并不陌生，但是对更多人而言却非常遥远。

自从被全世界软件界广泛运用以来，C++ 有了许多演化与变革。然而就像人们总是把目光放在艳丽的牡丹而忽略了花旁的绿叶，作为一个广为人知的面向对象程序语言（Object Oriented Programming Language），C++ 的另一面 — 泛型编程思维 — 被严重地忽略了。不能说面向对象思维和泛型思维有什么主从之分，但是红花绿叶相辅相成，肯定能对程序开发有更大的突破。

泛型思维在 C++ 身上主要以 templates 及相关特性来表现。许多人以为 template 是多么高阶的技巧，**望之俨然**；接触后才发现，**即之也温**。更深入研习泛型思维后又有另一层体会：**其言也厉**。

C++ templates 相关特性不难掌握，然而以此技术为载具所发展出来的一套泛型程序库：STL（现已纳入 C++ 标准程序库），背后却另有一套严密逻辑。本书一语道破 STL 的精神：一个严谨的软件概念（*concepts*）分类学。是的，这些 *concepts* 和任何程序语言无关，但就像 OO 思维需要 OO 语言的协助才得以顺利展现一样，泛型思维需要泛型语言的协助，才得以顺利展现。Ada 和 C++ 都支持泛型语法和语义，Java 也已跟进[1]。

换言之，泛型编程和 C++ templates 和 STL 不该并为一谈。然以现今发展看来，三者又几乎被划上等号。本书即以 C++ 为载具，第一篇讲述泛型思维的演进及 STL 的组织概念，第二篇整理 STL 所涵盖的全部 concepts 的完整规格，第三篇整理 STL 所涵盖的全部 components 的完整规格。

人们对于 STL 的最大误解是**效率**。事实上 STL 提供的是一个不损及效率的抽象性。泛型编程和面向对象编程不同，它并不要求你通过额外的间接层来调用函数；它让你撰写完全一般化并可重复运用的算法，其效率和「针对特定数据型别而设计」的算法旗鼓相当。每一个算法、每一个容器的操作行为，其复杂度都有明确规范 — 通常是最佳效率或极佳效率。在接受规格书明定的复杂度之后，我想你不会认为自己亲手撰写的码，能够更胜通过严格检验的世界通用程序库。

---

[1] 请参考 "GJ : A Generic Java - java may be in for some changes". Philip Wadler, DDJ, Feb., 2000。

人们对 STL 效率的误解，有一大部分是把编译期效率和运行期效率混为一谈了。的确，大量而嵌套地运用 templates，会导致编译器在进行 template 引数推导（argument deduction）及具现化（instantiation）时耗用大量时间。但它绝不影响运行效率。至于对项目开发时程所带来的影响，我要说，和 STL 为我们节省下来的开发时间相比，微不足道。

STL 的严重缺点在于，它尚未支持 persistence（对象持久性）。在良好的解决方案尚未开发出来之前，persistence 必须由用户自行完成。

作者 Austern 在本书「前言」对泛型思维及其技术演化还有更多精辟讨论，我建议你此刻（噢不，至少看完这篇译序之后）立即翻看「前言」，一定能够对整个技术的来龙去脉有所掌握。

永远记住，面对新技术，程序员最大的困难在于心中的怯弱。To be or not to be, that is the question! 不要和哈姆雷特一样犹豫不决，当你面对一个有用的技术，应该果断。

我在台湾元智大学开授一门「泛型编程」课程。本书便是我为同学列出的高阶参考书。本书在 (1) 泛型观念之深入浅出、(2) STL 组织之井然剖析、(3) STL 参考文件之详实整理 三方面有卓越的表现。可以这么说，在泛型技术和 STL 的学习道路上，本书并非万能（至少它不适合初学者），但如果你希望彻彻底底掌握泛型技术和 STL 的深层观念，没有此书万万不能 ☺。

侯捷 2003.03.08 于台湾新竹

jjhou@jjhou.com
http://www.jjhou.com（繁）
http://jjhou.csdn.net（简）

p.s. 本书采页页对译方式，并保留原文索引。简体版系以繁体版为基础。繁体版第一阶段初译工作由黄俊尧先生负责，同挂译者之名；由于俊尧未涉简体版之大小事务，因此未同列简体版译者。但我仍要在此感谢俊尧的贡献。

p.s. 本书根据英文版第四刷（4th printing）制作，并修订前三刷之错误。请上侯捷网站（网址如上）观看后续的讨论、勘误、程序样例。网页所列之 2003/ 02/20 前的繁体版勘误，已于本简体版修正完毕。

本书保留大量简短易读之英文术语；时而中英并陈。以下是我所有书籍（无论著译）的习惯用语，特请读者注意：

英文术语	本书译词	英文术语	本书译词
argument	引数 （i.e. 实参）	instantiated	实体化、具现化
by reference	传址	library	程序库
by value	传值	resolve	决议
dereference	提领 （i.e. 解参考）	parameter	参数 （i.e. 形参）
evaluate	评估、核定	type	型别 （i.e. 类型）
instance	实体		
equality	相等	equivalence	等价

以下是 STL 六大组件（components）之本书译名：

英文术语	本书译词	意义
adapters	配接器	用来修饰其他组件。包括 iterator adapters、function adapters、container adapters 三大类。
allocators	配置器	用来分配空间。空间可来自于内存或磁盘 — 取决于配置器如何实现。主要用来服务容器。
algorithms	算法	就是计算机科学一般意义上的「算法」。例如 sort, bineary search, permutation...。
containers	容器	就是数据结构，用来存放元素。例如 vector, list, set, map, red black tree, hashtable...。
function object	函数对象（意义同下）	同下。
functor	仿函数（意义同上）	一种行为类似函数的东西。具体就是「重载了"function call" 操作符」的 classes。
iterators	迭代器	一种行为类似指针的东西。具体就是「重载了++, --, ->, * 等操作符」的 classes。

前 言

这不是一本讨论面向对象编程（object-oriented programming）的书。

你或许会觉得奇怪。毕竟，你可能在书店的 C++ 分类区看到这本书，也可能听过别人把面向对象与 C++ 当同一件事来讲，但是这并非 C++ 语言的唯一用途。基本上 C++ 支持多种不同的设计思维模型（paradigms），其中最新也最鲜为人知的一项就是泛型编程（generic programming）。

正如许多新的观念一样，泛型编程实际上已有很长一段历史了。早期的泛型编程研究论文大约在 25 年前就已经出现；第一个实验性质的泛型程序库并非以 C++ 写成，而是以 Ada [MS89a, MS89b] 和 Scheme [KMS88] 完成。由于泛型算法太新，所以还没有相关教科书。

第一个走出研究圈的样例显得格外重要，它就是 STL：C++ Standard Template Library。STL 是由 Alexander Stepanov（他后来成为 Hewlett-Packard 实验室的一员）和 Meng Lee 共同发展，于 1994 年被纳入 C++ 标准程序库的一部分。可免费取得之「HP STL 实现品」[SL95] 也于这一年发行，它是展示 STL 强大威力的范本。

当 Standard Template Library 开始成为 C++ 标准规格的一部分，C++ 族群立刻意识到这是一个高品质而且高效率的容器类程序库（container classes library）。要在一堆物品中找出熟悉的东西是最容易的了，而每一位 C++ 程序员都熟悉容器类。每个具相当水准的程序也都需要某种管理对象的方法，甚至每位 C++ 程序员都曾实现过 `strings`、`vectors` 或 `lists`。

C++ 早期就已经有容器类程序库的存在。这个语言加入 template classes（亦即「可参数化的型别」）之后，第一个用途 — 事实上也是引入 template 的主要原因 — 就是将容器类参数化。很多厂商包括 Borland、Microsoft、Rogue Wave 与 IBM，都有自己的程序库，其中包括 `Array<T>` 或其对等物。

容器类的观念是如此根深蒂固，以致于 STL 给人的初步印象似乎不比其他容器类程序库多了些什么。这种印象使大家疏忽了 STL 的独特性。

STL 是一种高效、泛型、可交互操作的软件组件；巨大，而且可以扩充。它包含计算机科学中的许多基本算法和数据结构，而且它把算法和数据结构完全分离开来，互不耦合。STL 不只是一个容器类程序库，更精确地说它是一个泛型算法（generic algorithms）库；容器的存在使这些算法有东西可以操作。

你可以在程序中使用现有的 STL 算法，正如你使用现有的容器一样。举例来说，当你想要使用 C 标准库中的 `qsort` 函数，你也可以使用泛型的 STL `sort`（而且 `sort` 更简单、更弹性、更安全，也更有效率）。很多书籍，包括 David Musser 与 Atul Saini 合著的 *STL Tutorial and Reference Guide* [MS96]，以及 Mark Nelson 的 *C++ Programmer's Guide to the Standard Template Library* [Nel95]，都有说明如何以这种方式使用 STL。

这实际上是非常有用的。重复运用代码总比重新撰写代码来得好，现在你可以在你的程序中重复运用既有的 STL 算法。然而这还只是以某个角度来使用 STL 而已。STL 的设计具有扩充性；也就是说你可以自行撰写一些组件，与 STL 组件交互作用，就像不同的 STL 组件之间可以交互作用一样。有效地运用 STL，意味着对它进行扩充。

泛型编程（Generic Programming）

STL 并非只是一些有用组件的集合。它还有鲜为人知而未被了解的一面：它是描述软件组件抽象需求条件的一个正规而有条理的阶层架构（formal hierarchy）。由于所有 STL 组件都是精确符合某些特定条件而写成，所以 STL 组件可以相互作用，可以扩充，你可以在增加新算法和新容器的同时，对新旧代码之间的协同作用抱持巨大信心。

计算机科学的重要进步，许多是由于发掘了新的抽象性质而促成。一个被当代所有程序语言支持的决定性抽象性质就是副例程 subroutine（又名过程 procedure，或函数 function — 不同的语言使用不同的词汇）。C++ 支持另一种抽象性质：抽象数据型别（abstract data typing，ADT）。是的，在 C++ 中，我们可以定义新的数据型别，以及该型别的基本操作行为。

代码与数据的结合形成了抽象数据型别，它必须通过一个具有明确定义的接口来操作。subroutine 是一种重要的抽象性质，因为一旦使用它，你就不需要仰赖（甚至不必知道）其实际做法；同样道理，你可以使用抽象数据型别，可以用来操作甚至产生新值，而不必在意数据的实际表现方式。唯一重要的是其接口。

C++ 也支持面向对象编程（OOP）[Boo94, Mey97]，此技术涉及「与继承相关」、「由多型（polymorphic）数据型别构成」的阶层体系。面向对象编程（OOP）拥有比抽象数据型别（ADT）更多的间接性，因而达到更进一步的抽象性。某些情况下你可以访问某值并操作之，却不必指明其精确型别。你可以撰写单一函数，用以操作继承阶层体系中的不同型别。

泛型编程（Generic Programming）意味一种新的抽象性质。其中心抽象性比早期如副例程（subroutine）或类（class）或模块（module）的抽象性更难捉摸。它是数据型别的一组需求条件。这很难让人领悟，因为它并非与 C++ 的某个性质系结在一起。C++（甚或任何当代程序语言）并没有任何关键字可用来宣告一组抽象需求条件。

对于这种「一开始令人泄气的含糊情况」，泛型编程的回报是前所未有的弹性，以及不会损及效率的抽象性。泛型编程和面向对象编程不同，它并不要求你通过额外的间接层来调用函数；它让你撰写完全一般化并可重复运用的算法，其效率与「针对某特定数据型别而设计」的算法旗鼓相当。

泛型算法抽离（抽象化）于特定型别和特定数据结构之外，使得以接受尽可能一般化的引数型别。这意味一个泛型算法实际上具有两部分：(1) 用来描述算法步骤的实际指令；(2) 正确指定「其引数型别必须满足之性质」的一组需求条件。

STL 的创新在于认知这些型别条件可以被具体指明并加以系统化。也就是说，我们可以定义一组抽象概念（所谓 ***concept***s），只要某个型别满足某组条件，我们就说此型别符合某个 concept。这些 concepts 非常重要，因为算法对于其引数型别的大部分假设，都可以藉由「符合某些 concepts」以及「不同 concepts 之间的关系」来陈述。此外，这些 concepts 形成一个明确定义的阶层架构，这让人联想到传统面向对象编程中的继承（inheritance），只不过它是纯然的抽象。

这个 concepts 阶层架构是 STL 的一个概念性结构，也是 STL 最重要的部分。由于它，使得重复运用和交互操作变得可能。此一概念性结构作为软件组件的正规分类法也十分重要 — 即使没有具体代码。STL 确实包含有具象数据结构如 pair 和 list，但要有效率地运用这些数据结构，你必须了解其所依据的概念结构。

定义抽象的 concepts，并根据抽象的 concepts 来撰写算法与数据结构，是泛型编程的本质。

如何阅读本书

本书把 Standard Template Library 当作抽象概念库（library of abstract concepts）来描述。本书定义出 STL 基本的各个 concepts 与抽象性质，并指出所谓「某个型别模塑出某个 concept」是什么意思，「以某个 concept 的接口写成一个算法」又是什么意思。本书讨论 STL 涵盖的各个类和算法，并阐明如何撰写属于你自己并相容于 STL 的类和算法。此外本书还包含一份所有 STL concepts、classes、算法的完整参考手册。

每个人都应该阅读第一篇，这一部分介绍 STL 与泛型编程的主要概念。说明如何使用以及撰写一个泛型算法，并解释「算法之所以为泛型」的意义。所谓泛型（Genericity），具有「在多种数据型别上皆可操作」的含意。

探究泛型算法，自然而然便导出了 *concepts*、*modeling* 与 *refinement* 的核心观念，这些观念之于泛型编程，就像多型与继承之于面向对象编程，同样地基础，同样地重要。泛型算法作用于一维区间上，导出 STL 的数个基本概念：*iterators*、*containers*、*function objects*。

第一篇同时也介绍了本书通用的符号及排版习惯：术语 *modeling* 和 *refinement*、*ranges* 的非对称标示法，以及 concept 名称的特殊字体。

STL 定义了很多 concepts。某些 concepts 只因技术上的细节而互不相同。第一篇是个概论，概略讨论 STL concepts 的全貌。第二篇是详细的参考手册，包含每个 STL concept 的严格定义。你可能不会想要遍读第二篇；当你需要参考某个 concept 时才到第二篇查询，应该是更好的做法。（每当需要撰写符合某个 STL concept 的新型别时，你都应该参考第二篇。）

第三篇同样是参考手册，提供 STL 预定义的算法和类的说明。这一部分仰赖第二篇的「concept 定义」甚多。STL 所有的算法和几乎所有的具象型别都是 templates，每个 template 的参数都能以某种 concept 的 model 加以描述。第三篇的定义可以和第二篇对应的章节交互参考。

理想状况下，本书到达第三篇就可以结束了。不幸的是现实需求使我必须写出更多章节：一份有关可移植性议题的附录。STL 问世之初并没有可移植性问题，因为只有一份实现品存在。这种情况已不复存在。一旦某种语言或程序库有一个以上的实现品存在，任何关心可移植性的人都必须知道每个实现品之间的差异。

你仍然可以从 anonymous FTP 站台 `butler.hpl.hp.com` 下载早期的 HP 实现品，但此作品已无人维护。Silicon Graphics Computer Systems (SGI) 有一份较新的免费作品，可以从 `http://www.sgi.com/Technology/STL` 下载。至于 SGI STL 在不同编译器上的移植版本，可以从 `http://www.metabyte.com/~fbp/stl` 取得。此外这个世界还存在有其他商业化 STL 实现品。

如果你正在撰写真正的程序，光了解函数库的设计理论是不够的；你还必须知道不同的 STL 实现品以及不同的 C++ 编译器之间的差别。这些不怎么吸引人但却必要的细节，构成了附录 A 的主题。

谁该阅读本书

虽然本书谈的几乎都是以 C++ 写成的算法，但这不是一本算法导入型教科书，也不是一本 C++ 语言教本。本书确实对于上述两方「不为人熟知的某些观点」有所解释。更明确地说，由于 STL 对 templates 的使用方式迥异其他 C++ 程序，所以本书讨论了某些 templates 高阶技术。本书不应该是你的第一本 C++ 书籍，也不应该是你的第一本算法分析入门书。你应该知道如何撰写基本的 C++，同时也应该知道 *O(N)* 表示法的意义。

算法与数据结构方面，两本标准参考书是 Donand Knuth 的 *The Art of Computer Programming* [Knu97, Knu98a, Knu98b] 和 Cormen, Leiserson, and Rivest 的 *Introduction to Algorithm* [CLR90]。两本最佳的 C++ 入门书籍则是 Bjarne Stroustrup 的 *The C++ Programming Language* [Str97] 和 Stanley Lippman, and Josee Lajoie 的 *C++ Primer* [LL98]。

本书由来

我于 1996 年加入 Silicon Graphics Computer Systems 的编译器研发团队。那时候 Alex Stepanov 已经离开 HP 并加入 SGI 数个月了。当时的 SGI C++ 编译器并不包含 Standard Template Library 实现品。以 HP 的原始作品为基础，Alex、Hans Boehm 和我开始撰写一个新的 STL 版本，用来和 SGI MIPSpro 编译器 7.1（以及后续版本）搭配出货。

SGI Standard Template Library [Aus97] 包含很多崭新的扩充性质，例如高效且「对多线程而言安全（thread-safe）」的内存配置能力、散列表（hash tables），以及某些算法的改良。这些增强功能如果留做私用，对 SGI 的客户而言没有任何价值，所以 SGI STL 把它们全部免费公开。所有源码及文件都放在 `http://www.sgi.com/tech/stl`。

网路上的这份文件以网页形式呈现，将 STL 的概念结构视为核心，描述组成此结构之各个抽象概念（concepts），并以这些 concepts 来说明 STL 的算法和数据结构。我们收到很多要求，希望能够扩充此文件，本书就是对这些要求的回应。本书的「参考手册」部分，也就是二、三两篇，是 SGI STL 网页的分支。

整个网页是为 SGI 而写，其版权属于 SGI 所有。我在我所属的管理部门的宽容许可下使用那些网页内容于本书上。

致谢

首先，也是最重要的，如果没有 Alex Stepanov 的努力，这本书不可能诞生。Alex 参与这本书的每个阶段：他把我带进 SGI，我对泛型编程的每一个知识几乎都是他教导的，他参与 SGI STL 及其网页的开发，并鼓励我将网页的内容写成一本书。我由衷感谢 Alex 的帮助与鼓励。

我也要感谢 Bjarne Stroustrup 和 Andy Koenig 帮助我了解 C++，并感谢 Dave Musser 对于泛型编程、STL 及本书的诸多贡献（其中有些贡献可自参考书目中找到）。Dave 使用早期的 SGI STL 网页内容作为其课程教材，而通过他和他的学生的意见，这个网页有了很大的改进。

同样地，本书通过校阅者的意见而有了很大的改进，校阅者包括 Tom Becker、Steve Clamage、Jay Gischer、Brian Kernighan、Andy Koenig、Angelika Langer、Dave Musser、Sibylla Schupp 和 Alex Stepanov，

他们阅读过本书的每个版本。通过他们的协助，这本书更为清楚，错误也减少了许多。剩下的任何错误都是我的责任。

本书第一刷和第二刷中的许多错误已经更正，我要感谢 Sam Bradsher、Bruce Eckel、Guy Gascoigne、Ed James-Beckham、Jon Jagger、Nate Lewis、Shawn D. Pautz、John Potter、George Reilly、Manos Renieris、Peter Roth、Andreas Scherer 和 Jürgen Zeller，使我注意到这些错误。

我也要感谢 Addison-Wesley 的工作人员，包括 John Fuller、Mike Hendrickson、Marina Lang 与 Genevieve Rajewski，他们在我写作的过程中指导我。感谢 Karen Tongish 细心的编辑。

最后，我要感谢我的未婚妻 Janet Lafler 给我的爱与支持，以及对我在很多夜晚和周末写作的容忍。

我们的猫儿 Randy 与 Oliver，在我的键盘上走来走去试着想帮助我，不过最后我删掉了它们大部分的贡献。

他们阅读过本书的每个版本。通过他们的协助，这本书更为清楚，错误也减少了许多。剩下的任何错误都是我的责任。

本书第一刷和第二刷中的许多错误已经更正，我要感谢 Sam Bradsher、Bruce Eckel、Guy Gascoigne、Ed James-Beckham、Jon Jagger、Nate Lewis、Shawn D. Pautz、John Potter、George Reilly、Manos Renieris、Peter Roth、Andreas Scherer 和 Jürgen Zeller，使我注意到这些错误。

我也要感谢 Addison-Wesley 的工作人员，包括 John Fuller、Mike Hendrickson、Marina Lang 与 Genevieve Rajewski，他们在我写作的过程中指导我。感谢 Karen Tongish 细心的编辑。

最后，我要感谢我的未婚妻 Janet Lafler 给我的爱与支持，以及对我在很多夜晚和周末写作的容忍。

我们的猫儿 Randy 与 Oliver，在我的键盘上走来走去试着想帮助我，不过最后我删掉了它们大部分的贡献。

目 录