STL Tutorial and Reference Guide, Second Edition

C++ Programming with the Standard Template Library

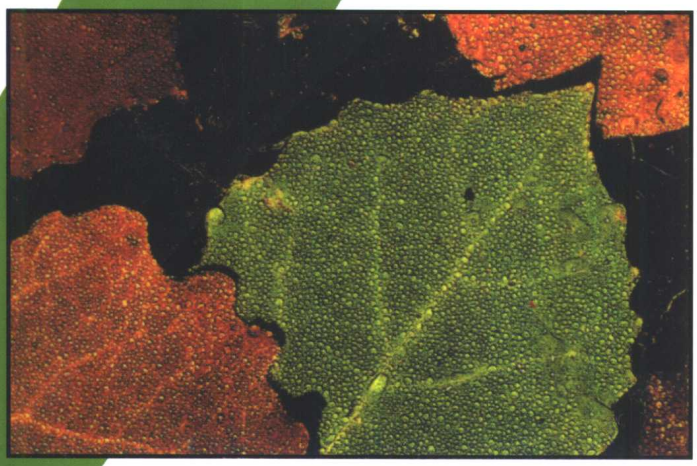
标准模板库自修教程与参考手册

STL 进行 C++编程

第二版

[美] David R.Musser
Gillmer J.Derge
Atul Saini 著

贺 民 王朝阳 译



Forward by Alexander Stepanov

科学出版社
www.sciencep.com

Addison Wesley

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

▶ Addison-Wesley 计算机专业教材

标准模板库自修教程与参考手册

——STL 进行 C++编程

（第二版）

[美]D.R.Musser, G.J.Derge, A.Saini 编著

贺民　王朝阳　译

科学出版社

北京

图字：01-2002-2605 号

内 容 提 要

本书译自 STL 经典教程《STL Tutorial and Reference Guide》最新版，本书作者最早提出了 STL 的基本理论，并对 STL 的应用和发展作出了卓越贡献。

本书内容分为 3 部分：第 1 部分是 STL 的入门知识，介绍了 STL 组件，STL 与其他软件库的区别，迭代器的概念，STL 类属算法，序列容器，有序关联容器，函数对象及容器、迭代器和函数适配器；第 2 部分是综合运用篇，其中给出了大量 STL 的应用实例，第 3 部分是 STL 参考指南，提供了迭代器、容器、类属算法、函数对象和函数适配器的参考信息，如文件、类的声明、示例、描述、构造函数和时间复杂度等。

本书内容全面、示例丰富，适合于用 C++语言编程的所有开发人员。

图书在版编目（CIP）数据

标准模板库自修教程与参考手册/（美）谬塞尔（Musser,D.R.）编著；
贺民，王朝阳译 —北京：科学出版社 2002 12
ISBN 7-03-011008-0
Ⅰ标… Ⅱ ①谬… ②贺… ③ 王 Ⅲ C 语言—程序设计 Ⅳ TP312
中国版本图书馆 CIP 数据核字（2002）第 094185 号

责任编辑：科海 / 责任校对：成昊
责任印刷：科海 / 封面设计：吕龙

科学出版社出版
北京东黄城根北街 16 号
邮政编码 100717
http //www sciencep com

北京市耀华印刷有限公司

科学出版社总发行 各地新华书店经销

*

2003 年 1 月第一版 开本：异 16 开
2003 年 1 月第一次印刷 印张：25
印数 1-5 000 字数：560 千字

定价：46 00 元

（如有印装质量问题，我社负责调换）

序

当 Dave Musser 邀请我为这本书作序时，我毫不犹豫地接受了。Dave 是我在事业上最亲密的伙伴，我们在一起合作了 20 多年，而且，没有 Dave 就不会有 STL。因此，他的这一邀请本身就是一种荣幸。同时，我也可以借此机会谈谈我在 STL 设计方面的看法。

在使用一门工具时，不但要懂得使用方面的规则，而且要了解一些指导其设计的原则。本序言的主要目的就是为读者提供 STL 幕后的设计原则。最后，我将以几点思考作为本篇序言的结尾。

STL 是基于如下几个原则进行设计的：

- 类属编程
- 在不损失效率的基础上进行抽象
- 冯• 诺伊曼计算模型
- 面向值的语义

类属编程 一些读者也许听说过这样一种说法，即 STL 就是利用一种称为“类属编程”的技术进行编程的方法，事实的确如此。一些读者也许还听说过类属编程就是运用 C++模板进行编程，这种认识则是片面的。类属编程的概念与 C++或模板无关。类属编程研究的是对软件组件的系统化组织。它的目标是研究出一种针对算法、数据结构和内存分配机制的分类方法，以及其他能够带来高度可重用性、模块化和可用性的软件工具。

为了最大限度地满足可重用性要求，就必须全面考虑各种可能的扩展情况。例如，当某位著名计算机科学家看到我编写的寻找两个数的最大公约数的算法

```
template <typename T> T gcd(T m, T n)  {
 while (n != 0)  {
  T t = m % n;
  m = n;
  n = t;
 }
 return m;
}
```

时，曾指出该算法是不正确的，因为当以 1 和–1 作为参数调用该函数时它返回–1，从而使得返回的公约数不满足最大条件。为解决这个问题，这位专家建议我将函数的最后一行修改成：

```
return m < 0 ? -m : m;
```

遗憾的是，如果做了这样的修改，该算法在很多扩展情况下就将无法使用，比如多项式、高斯整数等。它要求操作的数据集合是有序的。而如果我们采用一种更为抽象的（同时从算法上来讲也是更为有意义的）最大公约数定义，即最大公约数就是能被其他任何约数整除的约数，则这一问题就不存在了。在这一定义下，结果可能不惟一：对于整数，6 和–6 都是 24 和 30 的最大公约数。这与数百年来数学家们所做的工作不谋而合。

软件组件的类化应该只对那些有用的组件进行。比如，引入半序（有多个起点而只有一个终点的序列）的概念将是荒谬的，因为我们没有任何已知的数据结构来描述这种序列，也没有任何算法来对它们进行操作。

在我们将事物系统地组织起来之后，就可以保证它们之间接口的一致性。也就是说，两种组件在接口上的一致性程度应该大致等同于它们在行为上的一致性。这就使得我们可以建立能用于多种组件的算法——类属算法，同时也使得库的使用成为可能。如果程序员掌握了 STL 的向量，再学习使用 STL 的表将不会很难，掌握双端队列就更容易了。我坚持认为，能够最大限度地保证抽象编程的接口同时也是最容易掌握的接口（这里需要假定程序员是从零开始学习编程的，我们很难去说服习惯于使用 Lisp 的程序员，让他认为使用末尾后继（past-the-end）迭代器比测试 nil 的值更为有效）。

从很多方面来讲，类属编程都与抽象代数非常相似。假如有的读者学过涉及群、环、域的课程，就能够看出对迭代器的分类是出自何处了。[1]

就像数学把各种定理组织在不同的抽象理论体系当中一样，类属编程也把各种算法组织在不同的抽象概念中。因此，库的设计者的任务就是找出各种算法，同时也要找出这些算法能够有效工作的最低条件，并把各种算法组织到这些要求当中。通常，这类条件用某种合理的表达式及其语义描述。比如，STL 并未明确指出作用于迭代器的++要定义成某类的成员函数，它仅规定了如果 i 是迭代器且其所指向的对象可以被访问，则++i 是合法的表达式。

在不损失效率的基础上进行抽象 数学的研究对象经常是无法构造的，即便有时能够构造，也将需要无限长的时间。而在计算机科学中，效率至关重要。仅仅知道某一操作可以完成是不够的，更为重要的是要确保能够尽可能快地完成操作。STL 从几个方面保证了这一点。

首先，STL 对每一种接口都作出了复杂性方面的要求。当定义诸如迭代器一类的概念时，就给出了某种复杂性要求。程序员能够确定的是，对迭代器进行的黑体操作并不显著地依赖于迭代器在序列中的位置。而对迭代器指向的对象的访问也应该同样快速，因为将表的迭代器实现为包含指向表头的指针和一个整数索引的结构形式是不合法的（需要指出的是，尽管可以通

1. 我认为，一般来讲，数学修养对于优秀的程序员是不可或缺的。但遗憾的是，现在很多人在读大学（甚至读研究生）期间都无法接触到真正的数学。这里我主张，在您的整个职业生涯中，都要坚持不懈地学习数学。有些参考书是很优秀的。在此我想推荐下面三本由 John Stillwell 所著的书：*Numbers and Geometry*，*Mathematics and Its History*，以及 *Elements of Algebra*。在读完它们之后，还可以考虑 Robin Hartshorne 的 *Geometry: Euclid and Beyond* 和 Tristan Needham 的 *Visual Complex Analysis*。

过定义一组有效表达式及其语义来严格定义操作的含义，对复杂性的定义却非如此。要严格而有效地定义复杂性需求需要一种全新的方式)。

其次，STL 尽量不去隐藏数据结构中那些能使对数据的访问更加高效的部分。我们可以直接使用指向变量值的指针，从而可以直接改变某些域的值，而不是通过提供 get 和 put 方法来对容器操作，而这正是教科书作者们所乐于采用的方式。我们可以直接写

```
i->second = 5;
```

而不必写

```
pair<int, int> tmp = my_vector.get(i);
tmp.second = 5;
my_vector.put (i, tmp);
```

考虑到定期重新分配内存的问题，向量元素迭代器在重新分配内存后将不再指向原来的元素，我们不妨假定 STL 的使用者可以通过事先分配足够多的内存或者存储索引而不是迭代器来解决这一问题。

STL 尽了最大努力来保证其所有的类属算法是最优的并且和手工编码的算法同样有效率（确切地说，在使用优化良好的编译器（如 Kuck 和 Associates 的 C++编译器）的情况下将和手工编码的算法具有同样的效率)。

冯·诺伊曼计算模型 抽象数学是利用简单的数字事实来作为其抽象基础的，我们不应忘记数学也是一门实验科学，那么，应该用什么来作为建立一个类属的抽象架构的抽象基础呢？我认为，惟一合理的基础就是计算机的实际体系结构。重要的一点是，为了解决日益多样化的问题，经过了多年的发展才产生了现代计算机体系结构。字节寻址的存储器和指针并不是从早期的硬件设计中继承下来的——早期的硬件设计中并没有字节和指针的概念；要想实现循环，须借助于可以自己改变代码的程序——而是应用需求不断促进体系结构发展的结果。[2]如果要为各种数值类型设计一个类属架构，重要的是理解各种固有数据类型的工作方式，而不仅仅是关于整数和实数的数学原理。

计算机科学中最重要的新概念，同时也是数学中原来没有的，就是地址的概念。将地址，而不仅仅是值作为我们的计算模型的一部分是革命性的一步，从 Mark I 中的 72 个地址一直到成千上万的 Internet 地址，所有的进步都离不开这一概念的引入。从许多方面来讲，STL 中最具争议的一个方面就是它将地址以及地址在概念上的类化作为其整个体系的基石(对于实际进

2. 对于优秀的程序员来说，真正理解高级编程语言的精髓是颇为重要的。较好地理解若干种不同的体系结构也相当重要。我已经推荐过一些数学参考书，下面我将再推荐一些计算机参考书：我个人认为，John Hennessy 和 David Patterson 的 *Computer Architecture: A Quantitative Approach* 是最为重要的计算机科学方面的著作。此外，Gerrit Blaauw 和 Fred Brooks 的 *Computer Architecture: Concepts and Evolution* 将是一个很好的补充，特别是本书的第二部分，“The Computer Zoo”，其中提到了若干历史上著名的设计。

行编程的程序员来说，这一论调可能显得有些荒谬，但在学术界，几十年来一直在进行所谓的“功能编程”的研究，试图彻底摆脱地址这一概念）。如果用数学术语来表达，STL 的本质就是不同的数据结构对应不同的地址代数结构以及不同的地址连接方式。从数据结构的一个地址转向下一个地址的一系列操作就对应于迭代器。向数据结构添加和删除地址的一系列操作就对应于容器。

尽管 STL 迭代器的分类（输入、输出、单向、双向、随机存取）对所有的基本序列算法来讲已经足够，为了让 STL 能够正确处理多维结构，还需要定义其他类型的迭代器（事实上，在（1）非对称的访问，比如双端队列的迭代器或缓存行（cache line）和（2）多处理器系统的实现这两种情况下，即使对于许多基本的序列算法，为了加快其运行速度，二维迭代器也是必要的）。

面向值的语义 STL 将容器看作是结构的泛化。就像结构拥有其成员一样，容器同样也拥有成员。当复制结构时，其所有成员被复制。当结构被释放时，其所有成员也被释放。容器也是一样。这些属性使得结构和容器得以用来描述现实世界事物的关键属性，即整体和局部的关系。当然，整体和局部的关系并不是现实世界中惟一的关系，其他的关系则要靠迭代器来描述。[3]我认为，正是对局部和关系的混淆不清造成了对现实世界建模时概念上的混淆，同时也是我们需要碎片收集（即程序运行过程中对动态分配的内存进行回收的过程——译者注）的主要原因，然而这类混淆却在面向对象的语言和库中如此普遍地存在着。STL 并不是面向对象的，不仅因为它使用全局的类属算法，更为重要的是因为它把将对象作为一个组成部分和指向一个对象这两个概念进行了区分。它假定

```
T a = b;
```

创建了一个对象的副本，其各部分均不相同，而不是指向同一个对象的另一个指针。STL 中那些使用了赋值（sort，partition，remove 等等）的算法的规范要求使用这种面向值的语义。在 STL 中对象之间不可能共享成员（当然，一个对象是另一个对象的成员这种情况除外）。

一般来讲，在 STL 中认为任何类型的复制构造函数、析构函数、赋值和相等表达式操作的语义与固有类型的这些操作的语义相同。除此之外，STL 还假定对于那些定义了<、>、<=和>=运算符的对象，其语义也与定义在固有类型上的这些运算符的语义相同。用数学语言表述就是它们定义了一个全序（total ordering）（我认为，C++中一个比较令人苦恼的问题就是它并不要求基本操作的语义与固有数据类型的语义保持一致。我们甚至可以让运算符=来做乘法。只有应用在高度规则化的条件下，运算符重载才会带来方便，否则只会带来灾难）。

几点思考 STL 最初并不想设计成 C++标准库的一部分，而是打算设计成第一种通用算法和数据结构的库。之所以它后来成为 C++标准库的一部分，是因为 C++是惟一一种我可以

3. 例如，尽管我的腿是我的一个组成部分，我的律师却不是。如果我不存在了，我的腿也不存在了；如果我被复制了，我的腿也被复制了。我的律师是另一个人，尽管我的死可能从多方面对他产生影响——正如许多指向同一个死目标的指针一样，这称为悬空指针——但它并不会自动消失。

实现这样一个库来满足我个人要求的语言。在 STL 推广开来后的五年当中，许多人也宣称他们可以在他们所偏爱的语言上实现类似 STL 的东西：比如 Ada-95，ML，Dylan，Eiffel，Java 等等。也许他们能够做到，但直到目前为止还没有做到。我希望他们可以做到这一点，我也希望人们能够创造出一种比 C++更适合类属编程的语言。毕竟，在 C++中恰恰回避了类属编程。STL 中诸如迭代器和容器等基本概念并没有在 C++中得到描述，这是因为 STL 依赖于一系列严格定义的要求，而这些要求在 C++中没有以语言的形式表述出来（当然，在标准中对它们是有定义的，但那是用英语定义的）。

STL 的首要特征就是它是一种可扩展的架构。尽管 STL 已经得到了广泛的应用，我为类属组件建立大量库的愿望仍旧没有实现。我个人认为，没有建立起这些库的主要原因在于缺少一种机制来支持完成这项工作所需的资金。在基础性的算法工作当中是无法赚到钱的。需要由若干个小组的组件技术专家来为整个业界设计这些库。尽管我有时会比较幸运地从大型计算机公司那里得到一些资金，以进行 STL 方面的工作，但假如没有一种可靠的方式来资助这项工作，要想真正圆满地完成它是不可能的。在此我希望美国政府或者欧盟能够资助一个小而高效的组织来负责类属软件组件的开发工作。这里我指的不是科研工作，而是实际的开发工作，而且其目的是要开发出在体系结构、文档、通用性和效率方面都非常优秀的组件。

STL 将带来一种全新的计算机科学教育方式。99%的程序员并不需要了解如何建立组件，而只需学会如何去使用它们。STL 也将带来一种新的软件企业的运作方式。使用自己的代码，却不使用标准组件的人，将会被视为和设计非标准的、自己的 CPU 的人一样不可思议。我想知道，我们究竟能否进入软件的工业时代呢？……

Alexander Stepanov

前　言

在本书第一版问世之后的五年当中，C++语言标准的制定已经完成并且已被官方接受，各C++编译器厂商也在编译器和标准的兼容性方面有了较大的进展，此外还出现了许多关于这门标准化语言和库的书及杂志文章。其中很多书和文章都强调标准模板库（STL）是这一标准中最为重要的新特性。正如我们在本书第一版中所做的那样，人们对它给予了高度评价，认为STL 将会给人们编程的方式带来革命性变化。在过去五年当中，这一趋势已经比较明显地显现出来，其中该书对数万程序员所带来的影响不可忽视。我们在第一版已经提到了 STL 组件在当前软件开发工作中得到广泛使用的五个原因：

- C++已成为使用得最为广泛的编程语言（这得益于它提供了对建立和使用组件库的支持）。
- 由于 STL 已经纳入到 ANSI/ISO 的 C++及其库的标准中，编译器厂商也开始把 STL 作为其产品的标准发布版本的一部分。
- STL 中所有的组件都是类属的，因此也意味着它们可以用于各种不同的应用（需要编译器的语言支持）
- STL 组件高度的概括性并没有带来效率上的损失。
- STL 组件被精心设计成为可互换的编程模块，这使得可以以此为基础进一步开发用于数据库、用户界面等专门领域的组件。

我们很高兴地看到以上这些特性在过去五年的发展过程当中都得到了证实。

第二版的改进

在这次的新版中，我们显著增加了指导性素材，包括在第一部分增加了函数对象和容器、迭代器以及函数适配器方面的内容，在第二部分新增了两章内容，其中含有丰富的示例程序。我们还仔细检查了全部示例程序代码和相关的论述，包括第三部分的参考材料，以使它们符合最终的标准（尽管在标准定稿之后仍然发现了一些不甚严密之处，我们还是认为在大多数情况下关于 STL 组件规范的一些不确定之处不会给实际的编程人员带来严重问题。对于少数几种确实会带来问题的情况，我们都作了说明）。我们还在第三部分增加了一章实用工具方面的内容，其中讨论了 pair 和 comparison 类。新增了一个附录，其中讨论了标准串类（standard string class）的与 STL 相关的特性。

在这一版中，我们采用“文字式编程”风格来表示示例程序和代码段。对于那些不熟悉这种同时编写代码和文档的方式的读者，在第 2 章给出了简短的说明，并在第 12 章进行了详细的阐述。文字式编程方法的优势之一就是代码的细节可以只出现一次，之后就可以对其进行多次引用（通过名字和页码），从而读者可以不必重复阅读同一段代码的细节。另一个主要优势是我们可以进行比以前更加仔细的检查，以确保所有代码在语法上和逻辑上正确，因为文字式编程工具可以更方便地从书稿中直接提取代码并对其进行编译和测试。通过本书代码测试的编译器列在附录 D 中。

回 顾 历 史

几乎所有的 C++程序员都知道，这门语言是由 Bjarne Stroustrup 发明的。早在 1979 年，他就开始思考如何对 C 语言进行扩展，以使其支持对类和对象的定义。与此类似，STL 的体系结构也差不多是一个人的作品，这个人就是 Alexander Stepanov。

有趣的是，也是在 1979 年，几乎是在 Stroustrup 开始进行研究的同时，Alex 也开始着手进行类属编程方面的研究，并开始探究类属编程将可能对软件开发带来多大的影响。尽管早在 1971 年 Dave Musser 就已提出并推广了类属编程某些方面的理论，但仍局限于软件开发（或计算机代数）这一比较专业的领域内。Alex 充分认识到了类属编程的巨大潜力并说服了当时他在美国通用电气研究和开发部的同事（主要包括 Dave Musser 和 Deepak Kapur），他们认为软件开发应该全面地以类属编程为基础。但是当时任何一种编程语言都不提供对类属编程的支持。第一种提供这种支持的主要编程语言是 Ada，它通过其类属单元（generic unit）特性提供这种支持。到 1987 年 Dave 和 Alex 开发并发布了一个用于表处理的 Ada 库，并在其中体现了大多数他们在类属编程方面的研究成果。然而，Ada 在国防工业以外的领域并没有获得太广泛的接受。相反，当时 C++看起来更有可能获得广泛的应用并为类属编程提供更好的支持，尽管相对来讲这种语言在当时还不甚成熟（当时 C++语言中并没有模板的概念，模板是后来才加入到其中的）。Alex 很早就认识到的另一个原因也促使 STL 最终转向 C++语言，那就是 C/C++的计算模型允许用十分灵活的方式访问内存（通过指针），这对在获得通用性的同时又不损失效率至关重要。

尽管如此，仍然有大量的研究和实验工作有待开展，不仅仅包括开发个别的组件，更为重要的是要建立起基于类属编程的组件库的整个体系结构。Alex 先后在 AT&T 的贝尔实验室和惠普研究实验室试验了大量的体系结构和算法形式，开始是用 C 语言，后来用 C++语言。Dave Musser 也参与了这项研究，在 1992 年，Meng Lee 加入了 Alex 在惠普的项目并作出了重要贡献。

如果不是贝尔实验室的 Andrew Koenig 了解到了这项工作的价值并邀请 Alex 在 ANSI/ISO C++标准化委员会 1993 年 11 月的一次会议上介绍其主要理论的话，这项工作毫无疑问将只能作为一个研究项目继续进行一段时间，其最好结果也只能是最终开发出一个由惠普拥有专利的库。委员会对此的反应相当积极，最后 Andy 希望在 1994 年 3 月的会议上能够有一个正式的

提案。尽管面临着巨大的时间压力，Alex 和 Meng 还是在这次会议上拿出了一个草案，并获得了初步的认可。

委员会提出了一些修改和扩充意见（其中某些意见影响比较重大），然后委员会成员的一个小组与 Alex 和 Meng 进行了接触并帮助他们确定方案的细节。其中最为重要的一个扩充（关联容器）必须在全部实现后才能判断其是否可靠，Alex 把这一任务交给 Dave Musser 来完成。在这一紧要关头，整个工作很容易就会失去控制，但 Alex 和 Meng 又一次迎接了挑战并提出了一个方案，这一方案最终在 1994 年 7 月的 ANSI/ISO 委员会会议上获得了通过（关于这部分历史的其他信息，可以在 *Dr. Dobb's Journal* 杂志 1995 年 3 月号一篇 Alex 的谈话中找到）。

向世界推广

后来，Stepanov 和 Lee 的文档[17]包括到了 ANSI/ISO 的 C++标准草案（[1]，第 17 条到第 27 条）当中。它还影响了 C++标准库的其他方面，比如字符串工具，此外还有一些先前采纳的标准也做了相应修改。

尽管 STL 在 ANSI/ISO 委员会那里取得了成功，如何使 STL 走向实用化的问题依然存在。根据公开的标准草案中的 STL 部分，编译器厂商和独立软件库厂商可以开发出各自的库版本，可以作为单独的产品出售，也可以作为其他产品的卖点。本书第一版的作者之一 Atul Saini 最先认识到 STL 的商业潜力，早在 STL 被 ANSI/ISO 委员会接受为标准之前就已开始利用 STL 作为其公司 Modena Software Incorporated 的一种业务。

惠普在 1994 年 8 月决定将其 STL 实现放到 Internet 上免费发布，这一举动极大地促进了早期 STL 的传播发展。由 Stepanov、Lee 和 Musser 在标准化过程中开发的这一实现版本后来成为很多编译器厂商和库厂商的实现版本的基础。

同样是在 1994 年，Dave Musser 和 Atul Saini 推出了 STL 的第一个详尽的用户级文档——*STL++ Manual*，但他们不久就认识到需要一部更为全面的 STL 方面的论述，一本更好、更全面地涉及 STL 方方面面的著作。为了实现这一目标，在编辑 Mike Hendrickson 的大力支持和帮助下，他们写出了本书的第一版。

在第二版中，Gillmer J. Derge 加入了原来两位作者的行列。作为咨询公司 Toltec Software Services, Inc 的 CEO，Derge 有十多年的 C++开发经验，其中有七年在 General Electric Corporate R&D 度过，并且由于其技术成就而获得了惠特尼奖（Whitney Award）。

目 录

第 1 部分 STL 基础入门

第 2 部分 综合运用：示例程序

第 3 部分 STL 参考指南