# 第1章

## 关于本书

### 缘起

C++问世不久，就成为面向对象编程领域的事实标准（de facto standard）,因此正式标准化的呼声也就随之响起。只有有了可遵循的标准规格，我们才能写出跨PC乃至大型主机各种不同平台的程序。此外，如果能够建立起一个标准库（standard library）,程序员便得以运用可移植的通用组件（general component）和更高级的抽象性，而不必从头创建世界。

现在，偕同第二份标准，亦即所谓的C++11（见2.1节第7页关于C++standard的详细历史），我们有了一个巨大的C++标准库，其规格书的分量比起语言内核特性的规格书，两倍有余。这个标准库带来了：

* 输入/输出class
* String类型，和正则表达式（regular expression）
* 各种数据结构，例如dynamic array、linked list、binary tree、hash table
* 各种算法，例如众多排序算法（sorting algorithm）
* 多线程（multithreading）和并发（concurrency）calss
* 支持国际化的各种class
* 数值型（numeric）class
* 丰富的通用工具（utility）

然而，这个标准库的用法并非不言自明。想要从其强大能力中受惠，你需要一本好书；不能仅仅列出每个class和其成员函数了事。还必须详细解释各个组件（component）的概念和重要细节。本书正是以此为目标，首先从概念上介绍标准库及其所有组件，然后描述实际编程所需的细节。为了展示组件的确切用法，书中涵盖了大量实例。因此，本书不论对初学者或编程老手都是极详尽的C++标准库入门资料。搭配书中提供的资料，你应该能够掌握C++标准款的完全优势。

警告：我不担保本书所有内容都容易理解。标准库非常灵活，但这种非比寻常的灵活性是有代价的。标准库中有一些陷阱和缺陷，碰到它们时我会为你指出，并提出建议，帮助你回避问题。

### 阅读前的必要基础

想读懂本书的大部分内容，你需要先了解C++。本书讲述C++标准组件，而不是语言本身。你应该熟悉class（类）、template（模板）、继承（inheritance）和异常处理（exception handling）

的概念，但无须熟知这个语言的所有次要细节。重要细节已描述于本书之中；次要细节对程序库实现者可能很重要，对程序库的使用者就不那么重要了。

注意，C++标准化期间，这个语言又有变化，正如它在C++98标准化期间仍然持续变化一样，所以你的相关知识很有可能不再是最新的。第3章提供了一份简明扼要的的概览，为C++标准库用户介绍至关重要的语言特性。许多标准库新特性用上了这些语言新特性，所以你应该阅读第3章检阅所有C++新特性。当标准库用到语言新特性时，我也会提示你参照第3章。

### 本书风格和结构

C++标准库内的组件有相当程度的独立性，但彼此又有着关联，很难在描述某一部分时全然不提其他部分。我为本书考虑了几种不同的组织方式。一是按照C++standard的次序，但这并非完整介绍C++标准库的最佳选择。另一种方式是先纵览所有组件，再逐章详细介绍。

第三种方式是，依照“组件之交叉参考”程度高低，从最低者开始依次介绍。最终，我综合了三种方式：首先简短介绍标准库涉及的总体概念和通用小工具（utility），然后分章详述各组件，每个组件一或数章。首当其冲的是STL（standard template library，标准模板库）。STL无疑是标准库中最强大、最复杂、最激动人心的部分，其设计深刻影响了其他组件。

接下来我讲解较易理解的组件，例如特殊容器，string（字符串）和正则表达式（regular expression）。再来是你或许已经使用多时的老朋友：IOStream程序库。然后是国际化议题，

这部分对于IOStream程序库有些影响。最后，我描述数值（numeric）、并发（concurrency）和分配器（allocator）等部分。

讲述每个组件时，我首先给出其目的、设计和示例，然后是一份详细的说明，先以多种方式使用这个组件，继而说明相关陷阱和易犯错误。最后以一个参考小节（reference section）结束，其中有该组件的class和其成员函数的精确签名式（exact signature）和定义式（definition）。

**内容清单**

最前面的五章总体介绍本书及C++标准库：

* **第1章**：**关于本书** 介绍本书的主题和内容。
* **第2章：C++和标准库的简介** 简短概览C++标准库的历史和其标准化脉络，并介绍复杂度概念（concept of complexity）。
* **第3章：语言新特性** 针对阅读本书与使用C++标准库的需要，提供一分语言特性概览。
* **第4章：一般性概念** 描述标准库基本概念，这对于理解及使用所有标准库组件都是必要的。具体来说，本章介绍了namespace std，头文件（header file）格式、错误和异常处理（error and exception handling）的一般特性支持。
* **第5章：通用工具（Utility）**描述供应给标准库用户和标准库本身运用的数种小工具，更明确的说，本章描述class pair<>、tuple<>、smart pointer、数值上限、type trait 和 type utility、辅助函数、class ratio<>、clock和timer，以及可用的C函数。

第六章至第11章分别从个方面描述STL：

* **第6章：STL（Standard Template library，标准模板库）**STL提供了用于处理数据的容器和算法。本章逐步介绍STL的概念、问题、特殊编程技术，以及它们扮演的角色。
* **第7章：STL Container（容器）**本章解释STL容器类的概念和能力。首先通过详尽的例子分别讲解array、vector、deque、list、forward list、set、map、和unordered容器，然后介绍它们的共通能力、差异性、特定效益和缺点，并提供若干典型范例。
* **第8章：STL容器成员细论** 详列并描述所有容器的成员，包括其所提供的类型和操作函数，可用作手边参考资料。
* **第9章：STL Iterator（迭代器）**具体介绍迭代器的分类和辅助函数，以及适配器（iterator adapter），比如stream iterator、reverse iterator、insert iterator和move iterator。
* **第10章：STL Function Object及Lambda**详细讲解STL function object（函数对象）的诸多class及lambda，以及如何使用它们来定义容器和算法的行为。
* **第11章：STL Algorithm （算法）**罗列并描述STL算法。在简单介绍和比较这些算法后，借由一或多个范例详细解释每个算法。

第12章至第14章描述C++标准库中若干“结构相对单纯”的class：

* **第12章：特殊容器（Special Container）**描述容器适配器（container adapter）queue、stack，及calss bitset，后者可管理任意数量的bit或者flag。
* **第13章：String（字符串）**描述C++标准库的string类型（不止一种）。C++standard将string设计为一种“能够处理不同字符类型”的基础数据类型。
* **第14章：正则表达式（Regular Expression）**描述用来处理正则表达式的接口：它们可被用来查找和替换字符和子字符串（substring）。

第15章和第16章的主题是I/O和国际化（两者紧密相关）：

* **第15章 以Stream Class完成输入和输出** 涵盖众所周知的IOStream程序库的标准化形式。本章也讲述对程序员而言可能很重要却鲜为人知的若干细节，例如定义及整合特殊I/O通道。
* **第16章：国际化（Internationalization）**涵盖“程序国际化的概念”和class,例如对不同字符集（character set）的处理，以及使用不同格式的浮点数和日期。
* **第17章：数值（Numeric）**描述C++标准库中的数组组件，明确地说是随机数（random number and distribution）,复数（complex）及若干C数值函数。
* **第18章：并发（Concurrency）**描述用以支持并发和多线程的C++标准库特性。
* **第19章：分配器（Allocator）**描述C++标准中不同的内存模型（memory model）概念。

本书还包括一份参考书目（Bibliography）和一份索引。

篇幅所限，我不得不移走那些对一般程序员不那么切合，但实在应该被纳入补充章节（supplementary chapter）的内容，这份补充章节由本书官网http://www.cppstdlib.com提供，内容包括：

* bitset（简介于12.5节）细节
* Class valarry<> （扼要介绍于17.4节）
* 分配器（allocator）细节（简介于第19章）

### 如何阅读本书

本书即是引导式的使用指南，也是C++标准库的结构性参考手册。C++标准库的各个组件在相当程度上彼此独立，所以读完第2章至第5章后你可以任意跳读其他各章。不过切记，第6章至第11章讲述的是同一组东西，欲理解STL应该从导入性的第6章开始。

如果你是一位想认识C++标准库总体概念及其所有方面的程序员，你可以直接从头到为阅读全书，但应该跳过每章最后的参考小节（reference section）。如果你想运用C++标准库的某些组件来编写程序，那么，找出那个组件的最佳方式是运用索引——我已尽力做出了一份无所不包的最广泛索引，应足以节省你的查找时间。

根据我的经验，学习新东西的最佳方法就是看范例。为此，你会在本书发现许多范例。它们也行只是寥寥数行代码，也可能是完整程序。

如果是后者，其第一行注释包含有文件名。你可以在本书官网http://www.cppstdlib.com找到这些文件。

### 目前发展情势

本书撰写期间，C++11 standard完成了。请记住，有效编译器可能还无法与之兼容。这种情况可能在不久的将来获得巨幅改善。但现在，你可能会发现，本书所谈的东西在你的系统上并非一定有相同表现，或许得稍做修改才能在你的环境上正常运作。

### 范例代码及额外信息

你可以从我的网站http://www.cppstdlib.com上获得所有范例代码以及关于本书和C++标准库的更多信息。你也可以在互联网上找到许多相关信息，详见本书参考书目（bibliography）

# 第2章

## C++ 及标准库简介

本章中，我将讨论C++的历史及其不同版本，并介绍Big-O标记法，此法被用来具体显示标准库各项操作的效能（performance）和可扩展性（scalability）。

### 2.1 C++ Standard 的历史

C++标准化始于1989年，由国际标准化组织（the International Organization for Standardization,ISO）推动。ISO其实是一大群国家标准组织，成员机构包括美国的ANSI.。至今这项工作已经完成了四个里程碑，那就是遍及全世界不同平台上的各种C++ standard：

1. **C++98**，于1998年批准，是第一份C++标准规格。官方名称是Information Technology

Programming Language—C++，文档编号ISO/IEC 14882:1998。

1. **C++03**，这是个所谓“技术勘误”（technical corrigendum，TC），内含不甚严重的C++98 bug

修正。文档编号 ISO/IEC 14882:2003。不论C++98或C++03都被视为“第一份C++标准规格”。

1. **TR1**，内含大幅度的标准库扩充。官方名称是Information Technology—Programming

Language—Technical Report on C++ Library Extensions，文档编号ISO/IEC TR 19768:2007。

它所涵盖的扩充全部包含于namespace std::tr1内。

1. **C++11**，批准于2011年，是第二份C++标准。C++内含语言和标准库两方面皆十分重大的

改善和强化，而TR1所做的扩充亦被纳入成为namespace std的一部分。官方名称又一次是Information Technology Programming Language—C++，文档编号ISO/IEC 14882:1998。但它有一个新的文档编号ISO/IEC 14882:2011。

本书涵盖C++11，这一标准长时间被称为“C++0x”，带着一种“将于2009年前完成”的大众期待。所以，不论C++11或C++0x，其实是同一份东西，本书一律称之为C++11。

基于某些平台和环境尚未支持完整C++11（包括其语言特性和标准库），我会在谈到某个特性或行为时，提及它们是否只出现于C++11。

#### 2.1.1 C++11 Standard 常见疑问

**哪里可以获得这份标准**

最近的一份免费的C++11 standard 草案，文档名称为N3242 （见[C++Std2011Draft]）。这份草案应该足够满足大多数用户和程序员，但如果你需要真正的标准规格书，必须付费给ISO

或某个国属机构。

**为什么这份标准的制定花了这么长时间**

也许你会惊讶，为什么两次C++标准化过程之间竟花费了近10年甚至不止，可仍然未臻完美。请注意，这些标准规格是许许多多个人和公司的努力成果，这些努力包括提出改善和扩充建议，彼此讨论，等待实现，加以测试，解决“因为所有特性彼此交叉影响”而引发的每一个问题。这里面没有任何人是C++标准化的全职工作者。这一份标准并不是哪家公司拨出巨大预算和时间成本的研究成果。标准化机构没有（或几乎没有）付给那些为了开发标准规格而工作的人任何酬劳。所以，如果他们所处的公司对C++标准化漠不关心，那么他们就只能以兴趣为全部动力了。感天谢地，许多志愿者不但有时间也有余钱，可以全然义务地付出。大约50—100人的常态化聚会，每年3次，每次一星期，讨论所有的主题，然后在一整年的其他时间以电子邮件完成工作。这导致你得到的东西并非完美，或者具备完全一致的设计。整个成果是：实质堪用，但并非完美（世界上也不曾有完美的东西）。

标准库的描述占第一份标准规格书大约50%篇幅，在第二份标准规格书中升高至大约65%。就C++11而言，探讨标准库的页数从大约350页上扬至大约750页。

请注意，这份标准有着各式各样的源头。事实上，任何公司或国家，甚至个人，都有可能提出新特性和新扩充，那些提案必须被整个标准化组织接受。理论上没有任何东西的设计是从头做起的。因此整个成果的同质性（homogeneous）并不是很高。你会在不同的组件中发现不同的设计原理。例如string class 和STL 之间就有以下差异（STL是一个针对数据结构和算法而完成的框架）：

* String class 被设计为一个安全而便利的组件。因此它们提供一份几乎足以自我说明的接口，并检查许多可能发生的差错。
* STL 被设计为“将不同的数据结构与算法结合起来，产生最佳效能”。因此STL并不非常便利，也不对许多逻辑错误进行检查。为了获得威力强大的STL框架带来的好处和效能，你必须知道其所使用的概念，并且谨慎地套用它们。

上述两个组件都是标准的一部分。它们之间存在一些和谐，但依然遵循各自的基础设计哲学。

尽管如此，C++的另一个目标是简化事情。为此C++11引进了许多提案，用以解决各种问题与不一致性，以及人们在现实世界中找出的其他瑕疵。举个例子，数值和对象（value and object）的初始化在C++11中有了和谐性。另外，多少有点残破的smart pointer class auto\_ptr 现在已被多个改良后的smart pointer class 取代，后者在Boost中已臻成熟。Boost是个网站，特别强调自由、同行评审（peer-reviewed）、可移植、带源码的C++程序库（见[Boost]），主要是为了在任何议题被引入新标准或另一份技术勘误（TR）之前，先获得实战经验。

**这是最后一份C++ Standard吗**

C++ 不是终点。对于其中的新特性人们已经又有了bug修正、额外需求，以及各种提案。因此，或快或慢也许又会出现另一份带着bug修正和不一致性报告的“技术勘误”（TR），那也许会是TR2甚至第三份标准。

#### 2.1.2 C++98和C++11的兼容性

C++11的设计目标之一是，对C++98保持向后兼容。在C++98或C++03中可被编译的每一样东西，应该也都在C++中通过编译。然而还是有些例外，例如变量名称无论如何也不可以和新导入的关键字相同。

如果你的代码掺杂着C++各版本的身影，但你希望从C++11带来的改善中获利，那么可以的话，试着使用预定义好的宏\_\_cplusplus。对C++11而言，当编译一个C++转译单元（translation unit，也就是C++源码文件）时以下定义时有效的：

#define \_ \_cplusplus 201103L

与此相对，在C++98和C++03中则是：

#define \_ \_cplusplus 199711L

然而请注意，有时候，编译器厂商提供的值与上述不同。

注意，向后兼容仅适用于源码，不保证二进制兼容（binary compatibility），因为那会导致问题，特别是当一个原有操作（existing operation）取得一个新式返回类型（new return type）

，因为“基于返回类型而形成的重载（overloading）”是不被允许的（这适用于某些STL算法及STL容器的某些成员函数）。所以请以C++11编译器重新编译C++98程序的每一份源码，包括用到的所有程序库——通过都会成功。若以C++11编译器产生的二进制码链接（linking）C++98编译器产生的二进制码，则有可能失败。

### 2.2 复杂度与Big-O标记

C++标准库的某些部分，特别是STL，需要认真考虑算法和成员函数的效能，因此需要动用复杂度（complexity）概念。计算机科学家使用特殊符号来表现算法的相对复杂度，如此便可以很快依据算法的运行时间加以分类，进行算法之间的定性比较。这种衡量称为Big-O标记（notation）。

Big-O标记系将一个算法的运行时间以输入量n的函数表示。例如，当运行时间随元素个数呈线性增长时（亦即如果元素个数呈倍数增长，运行时间亦呈倍数增长），复杂度为O(n)；

如果运行时间与输入量无关，复杂度为O(1)。表2.1 列出了典型的复杂度和其Big-O标记。

|  |  |  |
| --- | --- | --- |
| **种类** | **标记** | **意义** |
| 常量 | O(1) | 运行时间和元素个数无关 |
| 对数 | O(log(n)) | 运行时间随元素个数的增加呈对数增长 |
| 线性 | O(n) | 运行时间随元素个数的增加呈线性增长 |
| n-log-n | O(nlog(n)) | 运行时间随元素个数的增加呈“线性与对数的乘积”增长 |
| 二次方 | O(n^2) | 运行时间随元素个数的增加呈平方增长 |

表2.1 五种典型的复杂度

请注意，Big-O标记隐藏（忽略）了指数较小的因子（例如常量因子），这一点十分关键。更明确地说，它不关心算法到底耗用多长时间。根据这种测量法则，任何两个线性算法都被视为具有相同的接受度。甚至可能发生一种情况：带有巨大常量的线性算法竟比带小常量的指数算法受欢迎（译注：因为Big-O标记无法显现真实的运算时间）。这是对Big-O标记法的一种合理批评。记住，这只是某种量度规则；它所显示的最佳（最低）复杂度算法，不一定就是真正的最佳（最快）算法。

表2.2列出了所有复杂度分类，并以某些元素个数说明了运行时间随元素个数增长的程度。一如所见，元素较少时，运行时间的差别很小，此时Big-O标记所隐藏的常量因子可能会带来很大影响。当元素个数愈多，运行时间差别愈大，常量因子也就变得无关紧要了。考虑复杂度时，请记住，输入量必须够大才有意义。

C++标准手册中提到的某些复杂度被称为amortized（分期摊还；折旧成本），意思是长期而言大量操作将如上述描述般进行，但单一操作可能花费比平均值更长的时间。举个例子，如果你为一个dynamic array 追加元素，运行时间将取决于array是否尚有备用内存。如果内存足够，就属于常量复杂度，因为在尾端加入一个新元素总是花费相同时间。但如果备用内存不足，那么就是线性复杂度，因为你必须分配足够的内存并搬动（复制）它们，实际耗用时间取决于当时的元素个数。内存重新分配动作并不常发生（译注：STL的dynamic array 容器会以某种设计哲学来保持备用内存），

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **种类** | **复杂度**  **标记** | **元素数目** | | | | | | | |
| **1** | **2** | **5** | **10** | **50** | **100** | **1000** | **10000** |
| 常量 | O(1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 对数 | O(log(n)) | 1 | 2 | 3 | 4 | 6 | 7 | 10 | 13 |
| 线性 | O(n) | 1 | 2 | 5 | 10 | 50 | 100 | 1000 | 10000 |
| n-log-n | O(nlog(n)) | 1 | 4 | 15 | 40 | 300 | 700 | 10000 | 130000 |
| 二次方 | O(n^2) | 1 | 4 | 25 | 100 | 2500 | 10000 | 100000 | 100000000 |

表2.2 运行时间与复杂度、元素个数对照表

所以任何“长度充分”的序列（sequence），元素附加动作几乎可说是常量复杂度。这种复杂度我们便称为amortized（分期摊还）常量时间。

# 第3章

## 语言新特性

C++语言内核与标准库，二者的标准化通常并行进行。这么一来，标准库可受益于语言的变化，而语言可受益于标准库实现过程中获得的经验。这导致C++标准库往往会用于若干“很可能在前一版标准中并不存在”的语言特性。

因此也可以说，C++11不是一个和C++98/C++03相同的语言，而C++98/C++03也和标准化之前的C++不同。如果你没有跟上演化，你可能会对标准库使用的语言新特性感到震惊。本章将给你一份扼要的C++11语言新特性概观，这些特性对于设计、了解、应用C++11标准库很重要。本章末端涵盖了若干在C++11之前已存在但一直没有被广泛认知的语言特性。

当我撰写本书时（2010年到2011年期间），并非所有编译器都能够提供C++11的所有语言新特性。我盼望这种情况能够很快获得改善，因为毕竟所有C++主流编译器厂商都参与了标准化过程。不过，在某段时间内，你或许会因为你所使用的程序库而受到局限。我将在整本书中以脚注提及任何典型且重要的局限。

### 3.1 C++11语言新特性

#### 3.1.1 微小但重要的语法提升

首先我要介绍C++11的两个新特性，它们虽然微小，对你每天的编程却十分重要。

Template表达式内的空格

“在两个template表达式的闭符之间放一个空格”的要求已经过时了：

vector<list<int> >; //*OK in each C++ version*

vector<list<int>>; //*OK since C++11*

上述两种形式都有可能出现于全书的任何代码中。

**nullptr** 和**std::nullptr\_t**

C++11允许你使用nullptr取代0或NULL，用来表示一个pointer（指针）指向所谓的no value（此不同于拥有一个不确定值）。这个新特性特别能够帮助你在“null pointer 被解释为一个整数值”时避免误解。例如：

void f(int);

void f(void\*);

f(0); //*calls f(int)*

f(NULL); //calls f(int) *if NULL is 0, ambiguous otherwise*

f(nullptr); //calls f(void\*)

nullptr是个新关键字。它被自动转换为各种pointer类型，但不会被转换为任何整数类型。

它拥有类型 std::nullptr\_t,定义于<cstddef> (5.8.1节第161页)，所以你现在甚至可以重载函数令它们接受null pointer。注意，std::nullptr\_t被视为一个基础类型（5.4.2节第127页）。

# 第4章

## 一般概念

本章讲述C++标准库的基本概念。几乎所有C++标准库组件都需要这些概念。

* 命名空间（namespace）std
* 头文件（header file）的名称与格式
* 差错与异常（error and exception）处理的一般概念
* 可被调用的对象（callabe object）
* 并发及多线程（concurrency and multithreading）的基本概念