HZ BOOKS

PEARSON Addison Wesley

Modern C++ Design C++ 设计新思维

Accelerated C++中文版

Accelerated C++: *Practical Programming by Example*

（美） Andrew Koenig Barbara E. Moo 著
靳志伟 等译

机械工业出版社
China Machine Press

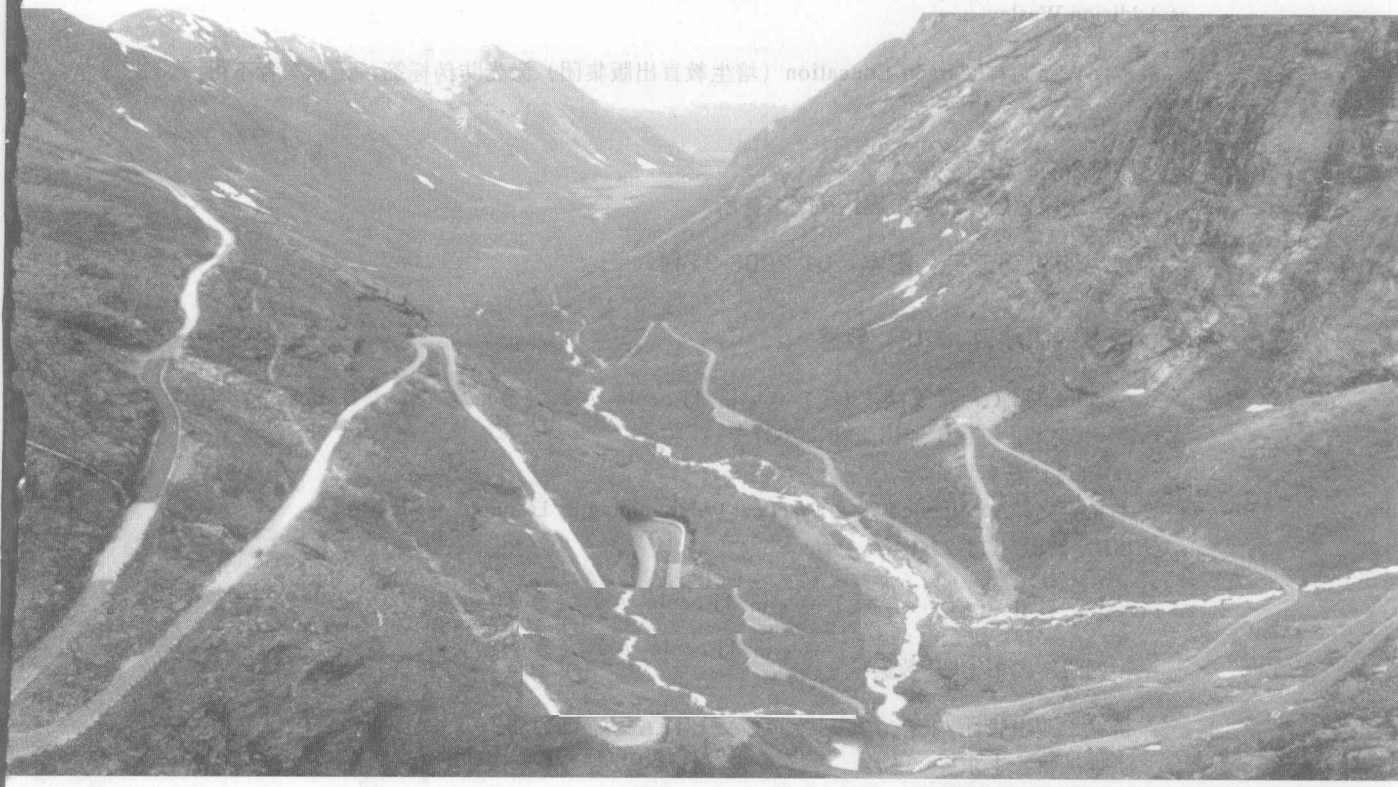
Accelarated C++中文版

Modern C++ Design
C++
设计新思维

Accelerated C++中文版

Accelerated C++: *Practical Programming by Example*

（美） Andrew Koenig Barbara E. Moo 著
靳志伟 等译



机械工业出版社
China Machine Press

本书系统介绍C++程序设计，是美国斯坦福大学的经典教材。从使用C++标准库中的高级抽象开始，使读者很快掌握编程方法。每一章都有很经典独特的例子以及非常到位的讲解，覆盖了C++非常多的内容，从标准库容器、泛型算法的使用，到类的设计、泛型算法的设计，本书都进行了详细的讲解。

本书作者有丰富的C++开发、研究和教学经验，内容由浅入深，讲解精炼巧妙。无论是刚入门的新手还是有经验的开发人员都能从本书中受益。

本书版权登记号：图字：01-2006-3144

图书在版编目（CIP）数据

Accelerated C++中文版/（美）克尼格（Koenig, A.），（美）莫（Moo, B. E.）著；靳志伟译. －北京：机械工业出版社，2007.10

（C++设计新思维）

书名原文：Accelerated C++: Practical Programming by Example

ISBN 978-7-111-22404-4

Ⅰ. A…　Ⅱ. ①克…　②莫…　③靳…　Ⅲ. C语言－程序设计　Ⅳ. TP312

中国版本图书馆CIP数据核字（2007）第147998号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：李东震

北京京北制版厂印刷 · 新华书店北京发行所发行

2008年1月第1版第1次印刷

186mm × 240mm · 19.25印张

定价：39.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

“C++设计新思维”丛书前言

自C++诞生尤其是ISO/ANSI C++标准问世以来，以Bjarne Stroustrup为首的C++社群领袖一直不遗余力地倡导采用“新风格”教学和使用C++。事实证明，除了兼容于C的低阶特性外，C++提供的高级特性以及在此基础上发展的各种惯用法可以让我们编写出更加简洁、优雅、高效、健壮的程序。

这些高级特性和惯用法包括精致且高效的标准库和各种“准标准库”，与效率、健壮性、异常安全等主题有关的各种惯用法，以及在C++的未来占据更重要地位的模板和泛型程序设计技术等。它们发展于力量强大的C++社群，并被这个社群中最负声望的专家提炼、升华成一本本精彩的著作。毫无疑问，这些学术成果必将促进C++社群创造出更多的实践成果。

我个人认为，包括操作系统、设备驱动、编译器、系统工具、图像处理、数据库系统以及通用办公软件等在内的基础软件更能够代表一个国家的软件产业发展质量，迄今为止，此类基础性的软件恰好是C++所擅长开发的，因此，可以感性地说，C++的应用水平在一定程度上可以折射出一个国家的软件产业发展水平和健康程度。

前些年国内曾引进出版了一大批优秀的C++书籍，它们拓宽了中国C++程序员的视野，并在很大程度上纠正了长期以来存在于C++的教育、学习和使用方面的种种误解，对C++相关的产业发展起到了一定的促进作用。然而在过去的两年中，随着.NET、Java技术吸引越来越多的注意力，中国软件产业业务化、项目化的状况愈发加剧，擅长于“系统编程”的C++语言的应用领域似有进一步缩减的趋势，这也导致人们对C++的出版教育工作失去了应有的重视。

机械工业出版社华章分社决定继续推动中国C++“现代化”教育，从2006年起将陆续推出一套“C++设计新思维”丛书。这套丛书秉持精品、高端的理念，其作译者为包括Herb Sutter在内的国内外知名C++技术专家和研究者、教育者，议题紧密围绕现代C++特性，以实用性为主，兼顾实验性和探索性，形式上则是原版影印、中文译著和原创兼收并蓄。每一本书相对独立且交叉引用，篇幅短小却内容深入。作为这套丛书的特邀技术编辑，我衷心希望它们所展示的技术、技巧和理念能够为中国C++社群注入新的活力。

荣　耀

2005年12月

南京师范大学

www.royaloo.com

译者序

进入大学学习编程，开始接触的语言是C语言，有了C语言的基础后，慢慢地过渡到C++语言。这个过程是很多人都走过的，也总以为是最自然的过程。但是当我翻开《Accelerated C++: Practical Programming by Example》时，顿时感到焕然一新：很少有C++的基础读物以如此顺序来编排，更难能可贵的是，这样的顺序安排得是那样自然，一气呵成。

在翻译本书的过程中，我越来越深地体会到这本书章节之间安排的巧妙：它从使用C++标准库中的高级抽象开始，可以让你很快地进入编程的状态，开门见山，而非曲径通幽。毕竟枯涩的曲径往往会消磨掉人的兴趣和意志。同时，我更加感到这本书的内容设计的精致——每一章都有很经典独特的例子以及非常到位的讲解。正因为如此，才使大多数人认为枯燥晦涩，甚至难以驾驭的C++变得生动有趣，条理清晰。也难怪本书会成为斯坦福大学的经典C++教材。

Andrew Koenig和Barbara E. Moo夫妇有着多年的C++开发、研究和教学经验，还亲身参与了C++的变革和演进的工作，在C++领域，他们是相当有影响力的人。正因为如此，本书讲解得通俗易懂，既像老师的循循善诱，又像是亲密朋友的指点与支持。翻译的过程也是我重新学习的过程，本书覆盖了C++中非常多的内容，从标准库容器、泛型算法的使用，到类的设计、泛型算法的设计，本书都进行了详细的讲解。而且，本书所讲述的是标准C++，无论你使用的是什么样的系统，本书都适用。

最后，谨以此感谢Andrew Koenig和Barbara E. Moo这一对“神雕侠侣”，正是因为他们的工作才有了本书的传奇问世。感谢北京师范大学的陈硕，同济大学软件学院的周闻钧老师，是他们给我很多翻译细节上的指点；感谢北京邮电大学的所有老师和同学为我营造的良好的翻译环境。因为你们，我才可以走到现在！

参加本书翻译的有靳志伟、何官卿、李大伟和姚海鹏。翻译过程中难免会有疏漏，如果读者有任何疑问，可以与我联系，希望在交流的过程中，大家共同进步！我的邮箱：jinzita177@gmail.com。

靳志伟

于北京

2007年7月1日

前　言

学习C++编程的新方法

我们假定你想要快速地学习如何编写实用的C++程序。因此，我们首先讲述了C++中最有用的部分。这种学习的策略虽然显而易见，但是这种策略的根本含义是指，即便C++是以C语言为基础的，我们也不会从教授C语言开始。相反，我们从开始就使用了高级的数据结构，而在后面才解释这些数据结构的基础。这种方法可以让你马上开始编写地道的C++程序。

我们的方法还有一个地方比较特殊：我们关注如何解决问题，而不是关注探讨语言和标准库的特征。当然，我们也讲述了这些特征，但是这样做只是为了给程序提供支持，而不是用程序来证明这些特征。

由于本书教授C++编程，而不仅仅是教授语言特征，所以对于已经有一些C++基础，而且想以更自然、更高效的风格来使用这门语言的读者来说，本书非常有用。初学C++语言的人，往往学习了语言技巧，但是并不知道如何使用这门语言来解决日常的问题。

我们的方法对于初学者和有经验的程序员同样有效

在过去的每个暑假，我们常常在斯坦福大学教授为期一周的C++强化课程。我们最初也选用传统的方法来教授这门课程：假定学生已经了解C语言，我们就从展示如何定义类开始，有系统地过渡到语言的其他方面。我们发现学生在头两天就会感到困惑和挫败感—直到他们学到足够的知识，能够开始编写有用的程序时，困惑和挫败感才会消失。而且一旦达到这种境界，他们就会学习得非常快。

当我们接触到对崭新的标准库提供了足够支持的C++实现时，我们就开始重新设计这门课程。新的课程从一开始就使用标准库，关注编写实用的程序，在学生掌握了足以让他们有效地使用语言细节的时候，我们才开始教授这些细节。

在采用这种方法后，我们发现了戏剧化的结果：学习了一天之后，学生就能够编写程序，而如果采用原先的课程方式，这可能要花费几乎一周的时间。另外，学生开始的挫败感也消失了。

抽象

我们的方法之所以有效，只是因为C++已经成熟了，我们对它的理解也成熟了。这种成熟条件使得我们可以忽略早期的C++程序和程序员所依赖的低级概念。

这种可以忽略细节的能力就是成熟技术的特征。比如，早期的汽车经常会出现故障，使得

每个司机都不得不成为一个业余的汽车修理工。如果一个司机在汽车出现问题后，就不知道如何回家的话，是非常危险的。但是今天的司机无须了解详细的工程知识，就可以驾驶汽车了。当然今天的司机也可能由于别的原因而想学习详细的工程知识，但那是另一回事了。

我们把抽象定义为可选择的忽略，即仅仅关注那些与手头工作相关的概念，而忽略其他东西。我们认为这是现代编程中最重要的思想。要想编写一个成功的程序，关键在于了解问题的哪些部分应该考虑，哪些部分应该忽略。每种编程语言都提供了创建有用的抽象的工具，而且每个成功的程序员都知道如何使用这些工具。

我们认为抽象非常有用，所以我们在本书中到处都可以看到抽象的概念。当然，我们并不总是直接把它们叫做抽象，因为抽象的形式太多了。相反，我们提到了函数、数据结构、类和继承，所有这些都是抽象。而且我们并不只是提到这些概念，本书到处都会使用这些概念。

如果可以很好地设计抽象并选取适当的抽象，我们相信，即便不理解这些抽象工作的细节，也可以使用它们。我们无须成为汽车修理工就能驾驶汽车，所以我们也无须理解C++工作的所有细节就能使用它们。

本书覆盖面

如果你的确要用C++编程的话，就需要掌握本书的所有知识，尽管本书并没有告诉你你想知道的一切。

这句话听起来有些荒谬，但事实确实如此。像本书这样厚的书不可能涵盖你想知道的C++的一切知识，因为不同的程序员和不同的应用程序需要不同的知识。因此，任何一本覆盖C++所有知识的书籍，比如Stroustrup写的《The C++ Programming Language》(Addison-Wesley出版社，2000)，都不可避免地告诉你很多你并不需要的知识。但是即便你不需要了解这些知识，还是有些人会有这个需要。

另一方面，C++的许多部分非常重要，如果不理解它们，就很难有效地编写程序。我们会关注这些部分。仅使用本书的知识，就完全可以编写各种各样的实用程序。事实上，我们的一个审校，他也是C++编写的一个大型商业系统的主要程序员，他告诉我们，本书基本上覆盖了他们工作中使用到的所有技术。

使用这些技术，你可以编写真正的C++程序，而不是C语言风格，或者其他语言风格的C++程序。一旦掌握了本书的内容，你就会知道还应该学习些什么知识以及如何来学习。外行的望远镜制造者们常常这样说，先制造一个3英寸的镜片，然后再制造一个6英寸的镜片，要比从头开始制造一个6英寸的镜片容易得多。

本书讲述的仅仅是标准C++，并且忽略了其他扩展。这样做的好处是，我们教你编写的程序可以在任何环境下运行。然而，这也暗示了我们不会讲述如何编写在窗口环境下运行的程序，因为这样的程序总是与特定的环境，往往也与特定的厂商密切地联系在一起。如果你想编写只在特定环境下运行的程序，你就只能在其他地方学习这样的知识，但是千万不要因此就把本书抛弃！因为我们的方法是通用的，你可以把这里所学的所有知识用在任意的环境中。当然，你

也应该尽量阅读GUI应用程序的书籍，但是请先阅读本书！

给有经验的C或C++程序员的提示

学习一种新的编程语言时，可能会不自觉地用已经了解的语言风格来编写程序。为了避免这种习惯，我们从一开始就使用C++标准库中的高级抽象。如果你是有经验的C或者C++程序员，这种方法对你来说，既有好消息，也有坏消息——其实都是同样的消息。

这些消息就是，你会惊奇地发现，你已有的知识对理解我们所讲述的C++知识来说用处不大。你需要学习的东西远远超出你的意料（这就是坏消息），但是你会学得比预期快得多（这就是好消息）。特别地，如果你已经知道C++，那么你首先学的可能是C语言编程，也就是说你的C++编程风格是基于C语言的。这样做并没有什么错，但是我们的方法与众不同，我们相信你会看到你从未见过的C++的另一面。

当然，很多语法细节都是相似的，但是这仅仅是细节而已。对于重要的概念，我们会以完全不同的顺序来安排，这很可能是你以前从未见过的顺序。比如，我们到第10章才会讲述指针和数组，我们也并不讨论你的旧爱`printf`和`malloc`。另一方面，我们在第1章就会开始讲述标准库的`string`类。我们说我们会采用一种新的方法，这并没有言过其实！

本书的结构

本书分为两部分。第一部分是从开始到第7章，关注使用标准库抽象的程序。第二部分从第8章开始，讲述如何定义抽象。

一开始就讲述标准库是与众不同的，但是我们认为这样做是正确的。C++语言的很多部分（尤其是比较难的部分）几乎都是为了库作者的利益来考虑的。库的使用者根本无须知道语言的这些部分。通过在开始忽略这些部分，直到第二部分才讲述这些内容，我们很快就可以编写实用的C++程序，这种方法比起通常的方法要快得多。

一旦理解了如何使用标准库，就可以学习标准库所依赖的底层技术，并且可以使用这些技术来编写自己的库。另外，你会有一种感觉，知道如何使一个库有用，以及何时避免完全编写新的代码。

虽然本书比很多C++书都要薄，但是我们对每个重要概念都至少使用了两次，关键概念更是不断使用。这样使得本书的很多地方都会引用其他内容。我们所采用的形式类似第3.1小节，表示参见第3.1节的内容。当我们第一次解释某个概念时，我们会用黑体来表示，这样做会使得它比较明显，并且可以让你注意到它是一个重点。

本书每章（除了最后一个章）结束的时候都有一个小结。小结有两个用处：可以让你加深对本章所介绍的概念的印象，并且小结还包含了其他相关的一些知识，我们认为你迟早需要了解这些知识。我们建议你第一次阅读本书的时候跳过这些内容，在以后需要的时候再回头来看。

两个附录详细总结并说明了C++语言和标准库的重要部分，希望对于编写程序会有一定的帮助。

发挥本书的最大功效

每本讲述编程的书籍都会包含示例程序，本书也是如此。为了理解这些程序是如何工作的，就需要在计算机上运行它们。这样的计算机非常多，新的计算机也在不断地出现——也就是说，直到你读到这句话为止，我们所说的任何关于它们的东西都是不精确的。因此，如果还不知道如何编译并执行一个C++程序的话，请访问http://www.acceleratedcpp.com，看看我们在那里是如何说的。我们会随时更新这个网站，为它添加运行C++程序的信息和建议。这个网站也提供了机器可读的一些示例程序版本，以及其他一些你可能会感兴趣的信息。

感谢

我们谨以此感谢下面的人们，没有他们就不可能有这本书。本书的审校对本书功不可没：Robert Berger，Dag Brück，Adam Buchsbaum，Stephen Clamage，Jon Kalb，Jeffrey Oldham，David Slayton，Bjarne Stroustrup，Albert Tenbusch，Bruce Tetelman和Clovis Tondo。Addison-Wesley出版社的很多人也参与了本书的出版工作；我们所知道的有Tyrrell Albaugh，Bunny Ames，Mike Hendrickson，Deborah Lafferty，Cathy Ohala和Simone Payment。Alexander Tsiris核对了第13.2.2节中的希腊语语源。最后，从高级程序开始教授C++的想法已经酝酿了好多年，这也是受到听过我们课程的成百上千的学生和数以万计的参与我们讨论的人们的激励而产生的。

Andrew Koenig
Barbara E. Moo
Gillette, New Jersey
2000年6月

目 录

第0章 入门

让我们从一个C++小程序开始：

```
// a small C++ program
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

程序员们常常把这个程序叫做“Hello,world!”程序。尽管这个程序很短，但在深入阅读它之前，应该先花时间在电脑上编译并运行这个程序。这个程序会在标准输出中输出

```
Hello, world!
```

标准输出通常是显示器上的一个窗口。如果有什么疑问，可以请教懂C++的人，或者在作者的网站上寻求帮助。网址是：http://www.acceleratedcpp.com。

这个程序很有用，因为它非常的简单，如果在编译和运行中遇到什么问题，最有可能的原因是排版的错误或者不知道如何执行它。此外，深入地理解这个非常短小的程序能学会非常多的C++基础知识。为了充分理解这个程序，我们将详细地讲述程序每一行。

0.1 注释

这个程序的第一行是：

```
// a small C++ program
```

以//开始的是一个**注释**，直到这一行的结束。编译器会忽略注释；注释的目的是为读程序的人解释这些程序。在本书中，为了能让读者更容易地分辨出程序中的注释，我们会用斜体来显示每个注释文本。

0.2 #include指令

在C++中，很多基本的程序，比如输入-输出，都是**标准库**（standard library）的一部分，而不是语言核心的一部分。这个区别非常重要，因为语言核心对所有C++程序都是适用的，但是标准库不同，必须显式地说明需要使用标准库的哪些部分。

程序中使用#include**指令**来说明需要的标准库程序，#include**指令**通常出现在一个程序的开始部分。这个程序使用的库程序仅仅是输入-输出库，可以通过下面的语句来指定：

```
#include <iostream>
```

iostream支持连续输入-输出，或者流输入-输出，而不是随机访问或者图形输入-输出。因为iostream出现在#include指令中，并且嵌入在尖括号中（<>），所以它表示C++标准库中的一个**标准头文件**。

C++标准没有准确地说明标准头文件是什么，但是它定义了每个头文件的名字和行为。在一个程序中包含一个标准头文件就可以使用相应的库程序，至于这个库程序是如何实现的，并不是我们现在所关心的。

0.3 main函数

一个**函数**就是一段程序，它有一个名字，而且程序的其他部分可以调用它，或者使它运行起来。每个C++程序都必须包含一个叫做main的函数。当请求执行一个C++程序时，就要调用main函数。

main函数的返回值是一个整数类型，它用来指示程序是否成功地运行。返回0表示运行成功；返回其他值表示有错误。下面的语句用来定义一个main函数，并且指定其返回值为int类型：

```
int main()
```

在这里，int是语言核心中用来表示整数的名称。main后面的圆括号中存放的是程序执行时从系统接收的参数。这个特殊的例子中没有参数，所以圆括号中是空的。我们将在第10.4节中学习如何使用main函数的参数。

0.4 花括号

接下来继续介绍如何定义main函数，跟在圆括号之后的，是封闭在**花括号**中的一连串的**语句**（常简称括号）。

```
int main()
{                       // left brace
                        // the statements go here
}                       // right brace
```

在C++中，花括号之间的语句是一个单元。在上面这个例子中，左花括号标志着main函数中语句的开始，右花括号标志着语句的结束。换句话说，花括号之间所有语句都属于同一个函数。

当花括号中有两个或者更多语句时，系统会按照语句出现的顺序来执行它们。

0.5 使用标准库来输出

花括号中的第一条语句完成了程序的实际工作：

```
std::cout << "Hello, world!" << std::endl;
```

这条语句使用标准库的输出**操作符<<**，在标准输出中输出"Hello,world!"，然后再输出std::endl值。

在一个名字前加上std::，表明这个名字属于**名字空间**std。名字空间是相关名字的集合；标准库使用std来包含它定义的所有名字。比如，iostream标准头文件定义了cout和

endl，这里通过std::cout 和 std::endl来使用它们。

std::cout指的是**标准输出流**，它使得从程序进行通常的输出很便利。在窗口式操作系统的C++程序运行中，std::cout 会把输出显示在与程序相关的窗口上。

std::endl会结束当前的输出行，所以，如果这个程序还有别的输出的话，将会从新的一行开始。

0.6 return语句

return语句就像下面这样：

```
return 0;
```

它结束当前执行的函数，并且把return和分号之间的数值（这里是0）返回到调用这个函数的程序处。返回的值的类型必须和函数声明的返回类型一致。在这个main函数的例子中，返回值类型是int，并且返回到C++实现本身。因此，从main函数中返回的值必须是一个结果为整数值的表达式，这个值会返回给实现。

当然，有很多理由要求我们终止一个程序时必须要小心谨慎。例如，有的程序有多个return语句。如果函数的定义要求返回某种特殊类型的值，那么这个函数中所有return语句都必须返回相应类型的值。

0.7 稍微深入分析“Hello, world!”程序

"Hello, world!" 程序中有两个贯穿C++的概念：表达式和生存空间（scope）。随着这本书的继续，关于这两个概念会有更详细的介绍，但是在这里，有必要先做一些基本的说明。

表达式的作用是请求系统进行计算。计算后会生成一个**结果**，同时也可能会有一些**副作用**——也就是说，它会影响程序或者系统的状态，而这些影响并不是结果的直接作用。比如，3+4是一个表达式，生成的结果是7，而且没有任何副作用。而

```
std::cout << "Hello, world!" << std::endl
```

也是一个表达式，它的副作用是在标准输出流上输出"Hello, world!"并且结束当前行。

每个表达式都包含操作符和操作数（operand），操作符和操作数都有很多种形式。在输出"Hello,world!"的表达式中，两个<<符号都是操作符，而剩下的std::cout，"Hello,world!"和std::endl都是操作数。

每个操作数都有一种**类型**。关于类型，有很多东西都需要介绍，但本质上，每种类型都表示一种数据结构和它适用的操作。操作数的类型决定了作用于它的操作符所产生的结果。

每种类型都有自己的名字。例如，语言核心定义int表示整数类型，标准库把std::ostream定义为流输出。在"Hello, world!"程序中，std::cout的类型就是std::ostream。

操作符<<有两个操作数，可是在程序中写了两个<<操作符，却只有三个操作数，这是怎么回事呢？原来操作符<<是**左结合**的（left-associative），左结合的大概意思是，当一个表达式