HZ BOOKS
华章科技

资深C++专家、C++11布道师、金山软件资深工程师撰写

深度剖析C++11中最常用新特性，从程序简洁性、性能、代码质量、内存泄露、多线程等多方面给出了代码优化的方法和建议

深入讲解了C++11在线程池开发、流行框架和库的开发、库的封装等各种工程级项目中的应用，包含大量实现源码并开源，可直接使用

华章原创精品

深入应用C++11

代码优化与工程级应用

In-Depth C++ 11

Code Optimization and Engineering Level Application

祁宇 著

机械工业出版社
China Machine Press

华章原创精品

深入应用C++11

代码优化与工程级应用

In-Depth C++11

Code Optimization and Engineering Level Application

祁宇 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

深入应用 C++11：代码优化与工程级应用 / 祁宇著 . —北京：机械工业出版社，2015.5
（华章原创精品）

ISBN 978-7-111-50069-8

I. 深… II. 祁… III. C 语言 – 程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2015）第 085822 号

深入应用 C++11：代码优化与工程级应用

出版发行：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：姜 影 责任校对：董纪丽
印 刷：三河市宏图印务有限公司 版 次：2015 年 5 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：26.75
书 号：ISBN 978-7-111-50069-8 定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzit@hzbook.com

Preface 前　言

为什么要写这本书

2011 年 C++11 标准刚发布时，广大 C++ 开发者奔走相告，我也在第一时间看了 C++ 之父 Bjarne Stroustrup 的 C++11 FAQ（http://www.stroustrup.com/C++11FAQ.html），虽然只介绍了一部分特性，而且特性的用法介绍也很简短，但给我带来三个震撼：第一个震撼是发现我几乎不认识 C++ 了，这么多新特性，与以前的 C++ 很不同；第二个震撼是很多东西和其他语言类似，比如 C# 或者 Java，感觉很酷；第三个震撼是很潮，比如 lambda 特性，Java 都还没有（那时 Java 8 还没出来），C++11 已经有了。我是一个喜欢研究新技术的人，一下子就被 C++ 那么多新特性吸引住了，连续几天都在看 FAQ，完全着迷了，虽然当时有很多地方没看明白，但仍然很兴奋，因为我知道这就是我想要的 C++。我马上更新编译器尝鲜，学习新特性。经过一段时间的学习，在对一些主要特性有一定的了解之后，我决定在新项目中使用 C++11。用 C++11 的感觉非常好：有了 auto 就不用写冗长的类型定义了，有了 lambda 就不用定义函数对象了，算法也用得更舒服和自然，初始化列表让容器和初始化变得很简便，还有右值引用、智能指针和线程等其他很棒的特性。C++11 确实让项目的开发效率提高了很多。

相比 C++98/03，C++11 做了大幅度的改进，增加了相当多的现代编程语言的特性，使得 C++ 的开发效率有了很大的提高。比如，C++11 增加了右值引用，可以避免无谓的复制，从而提高程序性能；C++11 增加了可变模板参数，使 C++ 的泛型编程能力更加强大，也大幅消除了重复模板定义；C++11 增加了 type_traits，可以使我们很方便地在编译期对类型进行计算、查询、判断、转换和选择；C++11 中增加的智能指针使我们不用担心内存泄露问题了；C++11 中的线程库让我们能很方便地编写可移植的并发程序。除了这些较大的改进之外，C++11 还增加了很多其他实用、便利的特性，提高了开发的便利性。对于一个用过 C# 的开发者来说，

学习 C++11 一定会有一种似曾相识的感觉，比如 C++11 的 auto、for-loop 循环、lambda 表达式、初始化列表、tuple 等分别对应了 C# 中的 var、for-loop 循环、lambda 表达式、初始化列表、tuple，这些小特性使我们编写 C++ 程序更加简洁和顺手。C++11 增加的这些特性使程序编写变得更容易、更简洁、更高效、更安全和更强大，那么我们还有什么理由不去学习这些特性并充分享受这些特性带来的好处呢？

学习和使用 C++11 不要背着 C++ 的历史包袱，要轻装上阵，把它当作一门新的语言来学习，才能发现它的魅力和学习的乐趣。C++11 增加的新特性有一百多项，很多人质疑这会使本已复杂的 C++ 语言变得更加复杂，从而产生一种抗拒心理，其实这是对 C++11 的误解，C++11 并没有变得更复杂，恰恰相反，它在做简化和改进！比如 auto 和 decltype 可以用来避免写冗长的类型，bind 绑定器让我们不用关注到底是用 bind1st 还是 bind2nd 了，lambda 表达式让我们可以不必写大量的不易维护的函数对象等。

语言都是在不断进化之中的，只有跟上时代潮流的语言才是充满活力与魅力的语言。C++ 正是这样一门语言，虽然它已经有三十多年的历史了，但是它还在发展之中。C++14 标准已经制定完成，C++17 也提上了日程，我相信 C++ 的未来会更加美好，C++ 开发者的日子也会越来越美好！

作为比较早使用 C++11 的开发者，我开始在项目中应用 C++11 的时候，可以查阅的资料还很有限，主要是通过 ISO 标准（ISO/IEC 14882:2011）、维基百科、MSDN 和 http://en.cppreference.com/w/ 等来学习 C++11。然而，这些资料对新特性的介绍比较零散，虽然知道这些新特性的基本用法，但有时候不知道为什么需要这个新特性，在实际项目中该如何应用，或者说最佳实践是什么，这些东西网上可没有，也没有人告诉你，因为当时只有很少的人在尝试用 C++11，这些都需要自己不断地去实践、去琢磨，当时多么希望能有一些指导 C++11 实践的资料啊。在不断实践的过程中，我对 C++11 的认识加深了，同时，也把应用 C++11 的一些心得和经验放到我的技术博客（http://www.cnblogs.com/qicosmos/）上分享出来，还开源了不少 C++11 的代码，这些代码大多来自于项目实践。技术分享得到了很多认识的或不认识的朋友的鼓励与支持，曾经不止一个人问过我同一个问题，你坚持写博客分享 C++11 技术是为了什么，有什么好处吗？我想最重要的原因就是 C++11 让我觉得 C++ 语言是非常有意思和有魅力的语言，不断给人带来惊喜，在窥探到 C++11 的妙处之后，我很想和更多的人分享，让更多的人领略 C++11 的魅力。另外一个原因是我的一点梦想，希望 C++ 的世界变得更加美好，C++ 开发者的日子变得更美好。我希望这些经验能帮助学习 C++11 的朋友，让他们少走弯路，快速地将 C++11 应用起来，也希望这些代码能为使用 C++ 的朋友带来便利，解决他们的实际问题。

“独乐乐，与人乐乐，孰乐乎？与少乐乐，与众乐乐，孰乐？”，这是我分享技术和写作此书的初衷。

读者对象

❑ C++ 开发人员。

C++11 新标准发布已经 4 年了，C++11 的使用也越来越普及，这是大势所趋，普通的 C++ 开发者不论是新手还是老手，都有必要学习和应用 C++11，C++11 强大的特性可以大幅提高生产率，让我们开发项目更加得心应手。

❑ C++11 爱好者。

其他语言的开发人员，比如 C# 或者 Java 开发人员，想转到 C++ 开发正是时机，因为新标准的很多特性，C# 和 Java 中也有，学起来也并不陌生，可以乘着新标准的“轻舟”学习 C++11，事半功倍，正当其时。

如何阅读本书

虽然 C++11 的目的是为了提高生产率，让 C++ 变得更好用和更强大，但是，这些新特性毕竟很多，面对这么多特性，初学者可能会茫然无措，找不到头绪。如果对着这些特性一一去查看标准，不仅枯燥乏味，还丧失了学习的乐趣，即使知道了新特性的基本用法，却不知道如何应用到实际开发中。针对这两个问题，本书试图另辟蹊径来解决。本书的前半部分将从另外一个角度去介绍这些新特性，不追求大而全，将重点放在一些常用的 C++11 特性上，有侧重地从另外一个角度将这些特性分门别类，即从利用这些新特性如何去改进我们现有程序的角度介绍。这种方式一来可以让读者掌握这些新特性的用法；二来还可以让读者知道这些特性是如何改进现有程序的，从而能更深刻地领悟 C++11 的新特性。

如果说本书的前半部分贴近实战，那么本书后半部分的工程级应用就是真正的实战。后半部分将通过丰富的开发案例来介绍如何用 C++11 去开发项目，因为只有在实战中才能学到真东西。后半部分实战案例涉及面比较广，是笔者近年来使用 C++11 的经验与心得的总结。这些实践经验是针对实际开发过程中遇到的问题来选取的，它们的价值不仅可以作为 C++11 实践的指导，还可以直接在实际开发中应用（本书开发案例源码遵循 LGPL 开源协议），相信这些实战案例一定能给读者带来更深入的思考。

通过学习本书基础知识与实战案例，相信读者一定能掌握大部分 C++11 新特性，并能应用于自己的实际开发中，充分享受 C++11 带来的好处。

C++ 之父 BjarneStroustrup 曾说过：C++11 看起来像一门新的语言。这个说法是否夸张，读者不妨看完本书之后再来回味这句话。

本书示例代码需要支持 C++11 的编译器：

- Windows：Visual Studio 2013。
- Linux：GCC 4.8+ 或者 Clang 3.4。

由于少数代码用到了 boost 库，还需要编译 boost 1.53 或最新的 boost 库。

勘误和支持

除封面署名外，张轶（木头云）参与了第 1 章大部分内容和 7.4 节的整理，还负责了本书大部分的审稿工作。由于笔者的水平有限，书中错漏之处在所难免，敬请读者批评指正，如有更多宝贵意见请发到我的邮箱 cpp11book@163.com，同时，我们也会把本书的勘误集中公布在我的博客上（http://www.cnblogs.com/qicosmos/）。本书中有少数内容来自 en.cppreference.com、MSDN 和 http://www.ibm.com/developerworks/cn/，以及一些网络博客，虽然大部分都注明了出处，但也可能存在疏漏，如果有些内容引用了但没注明出处，请通过邮箱 cpp11book@163.com 与我联系。

书中的全部源文件除可以从华章网站[⊖]下载外，还可以从 github(https://github.com/qicosmos/cosmos) 上下载，同时我也会将相应的功能更新及时更正出来。

致谢

首先感谢你选择本书，相信本书会成为你学习和应用 C++11 的良师益友。

感谢 C++ 之父 Bjarne Stroustrup 和 C++ 标准委员会，正是他们推动着 C++ 不断改进和完善，才使 C++ 变得更有魅力。

还要感谢一些热心朋友的支持，其中，史建鑫、于洋子、吴楚元、胡宾朔、钟郭福、林曦和翟懿奎审阅了部分章节的内容，并提出了宝贵的意见；还要感谢刘威提供了一些论文资料。

感谢机械工业出版社华章公司的两位编辑杨福川和姜影，在这一年多的时间里始终支持我的写作，他们的帮助与鼓励引导我能顺利完成全部书稿。

接下来我要感谢我的家人：感谢我的父母和妻子，没有他们承担所有的家务和照顾孩子，我不可能完成此书；感谢弟弟和弟妹对我的鼓励与支持。还要对一岁多的女儿说声抱歉，为

⊖ 参见华章网站 www.hzbook.com.——编辑注

了完成本书，已经牺牲了很多陪女儿玩耍的时间，记得女儿经常跑到我写作的书房拉着我的手往外走，边走边说：“爸爸一起玩一下”。在这要对我的家人说声抱歉，在这一年的时间里，由于专注于写作，对他们一直疏于关心和照顾。

谨以此书献给我最亲爱的家人，以及众多热爱 C++11 的朋友们！

祁宇（qicosmos）

目　录 Contents

第一篇 Part 1

C++11 改进我们的程序

Chapter 1 第 1 章

使用 C++11 让程序更简洁、更现代

本章要讲到的 C++11 特性可以使程序更简洁易读，也更现代。通过这些新特性，可以更方便和高效地撰写代码，并提高开发效率。

用过 C# 的读者可能觉得 C# 中的一些特性非常好用，可以让代码更简洁、易读。比如 var 可以在编译期自动推断出变量的类型；range-base for 循环非常简洁清晰；构造函数初始化列表使创建一个对象变得非常方便；lambda 表达式可以简洁清晰地就定义短小的逻辑，等等。

现在的 C++11 中也增加了类似的特性，不仅实现了上面的这些功能，而且在一些细节的表现上更加灵活。比如 auto 不仅可以自动推断变量类型，还能结合 decltype 来表示函数的返回值。这些新特性可以让我们写出更简洁、更现代的代码。

1.1 类型推导

C++11 引入了 auto 和 decltype 关键字实现类型推导，通过这两个关键字不仅能方便地获取复杂的类型，而且还能简化书写，提高编码效率。

1.1.1 auto 类型推导

1. auto 关键字的新意义

用过 C# 的读者可能知道，从 Visual C# 3.0 开始，在方法范围中声明的变量可以具有隐式类型 var。例如，下面这样的写法（C# 代码）：

```
var i = 10;                    // 隐式（implicitly）类型定义
```

```
int i = 10;                          //显式(explicitly)类型定义
```

其中，隐式的类型定义也是强类型定义，前一行的隐式类型定义写法和后一行的显式写法是等价的。

不同于 Python 等动态类型语言的运行时变量类型推导，隐式类型定义的类型推导发生在编译期。它的作用是让编译器自动推断出这个变量的类型，而不需要显式指定类型。

现在，C++11 中也拥有了类似的功能：auto 类型推导。其写法与上述 C# 代码等价：

```
auto i = 10;
```

是不是和 C# 的隐式类型定义很像呢？

下面看下 auto 的一些基本用法[⊖]：

```
auto x = 5;                          //OK: x是int类型
auto pi = new auto(1);               //OK: pi被推导为int*
const auto *v = &x, u = 6;           //OK: v是const int*类型, u是const int类型
static auto y = 0.0;                 //OK: y是double类型
auto int r;                          //error: auto不再表示存储类型指示符
auto s;                              //error: auto无法推导出s的类型
```

在上面的代码示例中：字面量 5 是一个 const int 类型，变量 x 将被推导为 int 类型（const 被丢弃，后面说明），并被初始化为 5；pi 的推导说明 auto 还可以用于 new 操作符。在例子中，new 操作符后面的 auto(1) 被推导为 int(1)，因此 pi 的类型是 int*；接着，由 &x 的类型为 int*，推导出 const auto* 中的 auto 应该是 int，于是 v 被推导为 const int*，而 u 则被推导为 const int。

v 和 u 的推导需要注意两点：

- 虽然经过前面 const auto*v=&x 的推导，auto 的类型可以确定为 int 了，但是 u 仍然必须要写后面的“=6”，否则编译器不予通过。
- u 的初始化不能使编译器推导产生二义性。例如，把 u 的初始化改成“u=6.0”，编译器将会报错：

```
const auto *v = &x, u = 6.0;
error: inconsistent deduction for 'const auto': 'int' and then
'double'
```

最后 y、r、s 的推导过程比较简单，就不展开讲解了。读者可自行在支持 C++11 的编译器上实验。

由上面的例子可以看出来，auto 并不能代表一个实际的类型声明（如 s 的编译错误），只是一个类型声明的“占位符”。

使用 auto 声明的变量必须马上初始化，以让编译器推断出它的实际类型，并在编译时将 auto 占位符替换为真正的类型。

⊖ 部分示例来自 ISO/IEC 14882:2011，7.1.6.4 auto specifier，第 3 款。

细心的读者可能会发现，auto 关键字其实并不是一个全新的关键字。在旧标准中，它代表“具有自动存储期的局部变量”，不过其实它在这方面的作用不大，比如：

```
auto int i = 0;          //C++98/03，可以默认写成 int i = 0;
static int j = 0;
```

上述代码中的 auto int 是旧标准中 auto 的使用方法。与之相对的是下面的 static int，它代表了静态类型的定义方法。

实际上，我们很少有机会这样直接使用 auto，因为非 static 的局部变量默认就是“具有自动存储期的”[⊖]。

考虑到 auto 在 C++ 中使用的较少，在 C++11 标准中，auto 关键字不再表示存储类型指示符（storage-class-specifiers，如上文提到的 static，以及 register、mutable 等），而是改成了一个类型指示符（type-specifier），用来提示编译器对此类型的变量做类型的自动推导。

2. auto 的推导规则

从上一节的示例中可以看到 auto 的一些使用方法。它可以同指针、引用结合起来使用，还可以带上 cv 限定符（cv-qualifier，const 和 volatile 限定符的统称）。

再来看一组例子：

```
int x = 0;

auto * a = &x;           //a -> int*，auto 被推导为 int
auto   b = &x;           //b -> int*，auto 被推导为 int*
auto & c = x;            //c -> int&，auto 被推导为 int
auto   d = c;            //d -> int ，auto 被推导为 int

const auto e = x;        //e -> const int
auto f = e;              //f -> int

const auto& g = x;       //e -> const int&
auto& h = g;             //f -> const int&
```

由上面的例子可以看出：

- a 和 c 的推导结果是很显然的，auto 在编译时被替换为 int，因此 a 和 c 分别被推导为 int* 和 int&。
- b 的推导结果说明，其实 auto 不声明为指针，也可以推导出指针类型。
- d 的推导结果说明当表达式是一个引用类型时，auto 会把引用类型抛弃，直接推导成原始类型 int。
- e 的推导结果说明，const auto 会在编译时被替换为 const int。
- f 的推导结果说明，当表达式带有 const（实际上 volatile 也会得到同样的结果）属性时，

⊖ ISO/IEC 14882:2003，7.1.1 Storage class specifiers，第 2 款。