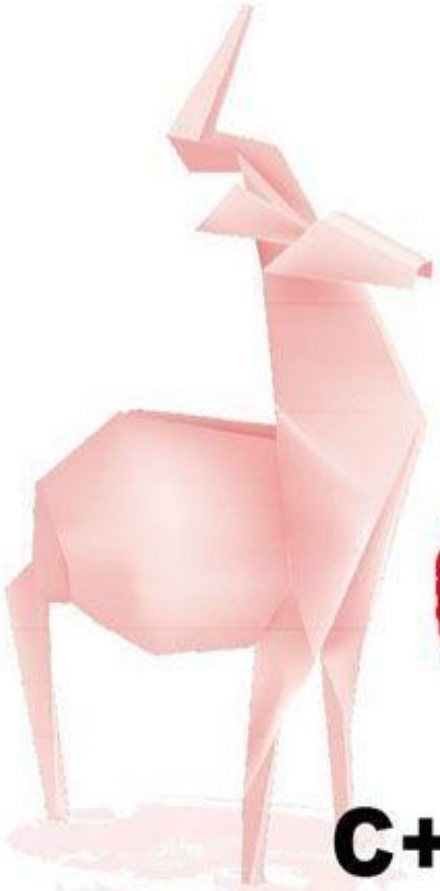
HZ BOOKS
华章科技

C++标准委员会成员和IBM XL编译器中国开发团队共同撰写，权威性毋庸置疑

系统、深入、详尽地讲解了C++11新标准中的新语言特性、新标准库特性、对原有特性的改进，以及所有这些新特性的应用

华章原创精品



深入理解C++11

C++11新特性解析与应用

Understanding C++11

Analysis and Application of New Features

（加）Michael Wong
IBM XL编译器中国开发团队 著

机械工业出版社
China Machine Press

免责声明

本书言论仅为作者根据自身实践经验所得，仅代表个人观点，不代表IBM公司的官方立场和看法，特此声明。

IBM XL编译器中国开发团队简介

IBM拥有悠久的编译器开发历史（始于20世纪80年代），在全球拥有将近400名高素质工程师组成的研发团队，其中包括许多世界知名的研究学者和技术专家。IBM一直以来都是编程语言的制定者和倡导者之一，并将长期在编译领域进行研发和投资。IBM XL编译器中国开发团队于2010年在上海成立，现拥有编译器前端开发人员（C/C++）、后端开发人员，测试人员，以及性能分析人员等。团队与IBM北美编译器团队紧密合作，共同开发、测试和发布基于POWER系统的AIX及Linux平台下的XL C/C++和XL Fortran系列产品，并对其提供技术支持。虽然团队成立时间不长，但已于2012年成功发布最新版本的XL C/C++for Linux V12.1&XL Fortran for Linux V14.1，并获得7项发明专利。团队成员拥有丰富的编译器开发经验，对编译技术、编程语言、性能优化和并行计算等都有一定的研究，也对C++11标准的各种新特性有较早的研究和理解，并正在实际参与C++11新特性的开发工作。

欢迎广大读者关注IBM XL编译器中国开发团队博客：http://ibm.co/HKOGCx，及新浪微博：www.weibo.com/ibmcompiler，并与我们一起学习、探讨C++11和编译技术。

作者个人简介

官孝峰　毕业于上海交通大学软件学院。多年致力于底层软件开发，先后供职于AMD上海研发中心，IBM上海研发中心。在嵌入式系统及应用、二进制翻译、图形驱动等领域有丰富的实践经验。2010年加入IBM XL中国编译器中国开发团队，负责XL C/C++/Fortran编译器后端开发工作，专注于编译器后端优化、代码生成，以及语言标准等领域。在C++11标准制定后，率先对标准进行了全面深入的研究，并组织团队成员对新语言标准进行学习探讨。是数项国内外专利的发明人，并曾于DeveloperWorks社区发表英文论文一篇。

陈晶　毕业于西安交通大学通信与信息工程专业，有多年的编译器文档开发写作经验。2008年起供职于IBM上海研发中心，一直致力于研究C++11标准的各项新特性，并负责该部分的技术文档撰写。精通C/C++语言，对编译器领域有浓厚的兴趣。负责工作部门内部的编译器产品技术培训。在DeveloperWorks社区发表过多篇论文。拥有一项国内专利。

任剑钢　毕业于复旦大学计算机专业。2010年加入IBM XL中国编译器中国开发团队，先后从事XL Fortran编译器前端、XL C/C++/Fortran编译器后端等各种开发工作。对于技术敏感而热衷，擅长C/C++/Java等多种编程语言，现专注于编译器代码优化技术。拥有一项专利，并带领团队在IBM Connections平台的技术拓展大赛中赢得大奖。

朱鸿伟　毕业于浙江大学计算机科学与技术专业。资深软件开发工程师，有多年系统底层软件开发和Linux环境开发经验。一直致力于C/C++语言、编译器、Linux内核等相关技术的研究与实践，关注新技术和开源社区，对于Linux内核以及Linux平台软件的开发有着浓厚的兴趣，曾参与Linux开源软件的开发与设计。2010年加入IBM XL中国编译器中国开发团队，现专注于编译器前端技术研究与开发工作。

张青山　毕业于福州大学计算机系。从事嵌入式开发多年，曾致力于Linux内核和芯片驱动程序的开发、及上层应用程序的编写。2010年加入IBM XL编译器中国开发团队，负责XL C++编译器前端的研发工作。对C99、C++98、C++11等语言标准及编译理论有深入理解，并实际参与C++11前端各种特性的实现。此外还致力于编译器兼容性的研究和开发。

纪金松　中国科学技术大学计算机体系结构专业博士，有多年编译器开发经验。2008年起先后就职于Marvell（上海）、腾讯上海研究院、IBM上海研发中心。一直致力于系统底层软件的研究与实践，在编译器后端优化、二进制工具链、指令集优化、可

重构计算等相关领域有丰富的实战经验。在国内外杂志和会议中发表过10多篇论文，并拥有多项国内外专利。

郭久福　毕业于华东理工大学控制理论与控制工程专业。拥有近10年的系统软件开发经验，曾就职于柯达开发中心、HP中国以及IBM中国软件实验室。对C语言标准以及C++语言标准有深入研究，近年来致力于编译器前端的开发和研究。

林科文　毕业于复旦大学计算机软件与理论专业，有多年底层系统开发经验。2010年起供职于IBM上海研发中心，现从事编译器后端开发工作。一直致力于系统软件的研究和实践，以及编译器代码优化等领域。活跃于DeveloperWorks社区，是三项国内专利的发明人。

班怀芸　毕业于南京理工大学计算机应用技术专业。资深测试工程师，在测试领域耕耘多年，曾任职于Alcatel-Lucent公司，有丰富的项目经验。2011年加入IBM XL编译器中国开发团队，现从事和C/C++语言相关的测试领域研究，负责用编译器实现相关需求分析、自动化测试方案制定及实现等工作。持续关注语言和测试技术的发展，对C/C++新特性和语言标准有较深的理解。

蒋健　毕业于复旦大学计算机科学系，资深编译器技术专家。先后供职于Intel、Marvell、Microsoft及IBM等各公司编译器开发部门，参与并领导业界知名的编译器后端多种相关研发工作，并且拥有近十项编译器方面的专利。现负责XL C/C++/Fortran编译器后端开发，并领导IBM XL中国编译器中国开发团队各种技术工作。

宋瑞　毕业于北京大学微电子学院，在软件测试领域工作多年，对于测试架构以及软件的发布流程具有丰富的经验，曾就职于Synopsys和Apache design solution。2011年加入IBM XL编译器中国开发团队，从事和编译器、C/C++语言相关的研究与测试，对C/C++新语言特性和标准有较深的理解和研究。

刘志鹏　毕业于南京理工大学计算机应用专业，2010年加入IBM XL中国编译器中国开发团队。先后从事Fortran、C++前端的开发工作。现致力于C++语言新标准、语言兼容性等研究与开发。对编译器优化技术、Linux内核开发有浓厚兴趣，擅长C/C++/Java等多种语言。

毛一赠　毕业于上海交通大学，现任职于IBM编译器中国开发团队，从事XL C/C++编译器前端的开发工作，在此领域有多年研究经验。具有丰富的实际项目经验并持续关注语言发展。擅长C++、C、Java等语言，对C++模板有深入研究。此外对C#/VB/perl

/Jif/Fabric等语言也均有所涉猎。热爱编程与新技术，活跃于DeveloperWorks等社区，发表过技术博客数篇。

张寅林　毕业于上海交通大学信息安全工程专业。2010年起加入IBM XL编译器中国开发团队，专注于编译器后端性能优化开发工作。对于编译器优化算法，Linux操作系统体系结构与实现有浓厚的兴趣，擅长C/C++/Python编程语言。目前从事IBM企业级存储服务器的开发工作。

刘林　东南大学计算机科学与工程学院硕士，有多年底层系统开发经验。2010-2012年间就职于IBM XL编译器中国开发团队，先后从事编译器测试及后端开发工作。对嵌入式、Linux操作系统体系结构与实现有浓厚的兴趣。

Preface

If you are holding this book in the store,you might be wondering why you should read this book among many C++books.First,you should know this book is about the latest new C++11(codenamed C++0x)Standard ratified at the end of 2011.This new Standard is almost like a new language,with many new language and library features but there is a strong emphasis for compatibility with the last C++98/03 Standard during design.At the time of printing of this book in 2013,this is one of the first few C++11 books published.All books that do not mention C++11 will invariably be talking about C++98/03.

What makes this book different is that it is written by Chinese writers,in its original Chinese language.In fact,all of us are on the C++compiler team for the IBM xlC++compiler,which has been adding C++11 features since 2008.

For my part,I am the C++Standard representative for IBM and Canada and have been working in compilers for 20 years,and is the author of several C++11 features while leading the IBM C++Compiler team.

For C++users who read Chinese,many prefer to read an original Chinese language book,rather than a translated book,even if they can read other languages.While very well written also by experts from the C++Standard Committee,these non-Chinese books'translation can take time,or result in words or m

eanings that are loss in translation.The translator has a tough job as tech nical books contain many jargon and new words that may not have an exact me aning in Chinese.Different translators may not use the same word,even withi n the same book.These are reasons that lead to a slow dissemination of C++k nowledge and slows the adoption of C++11 in Chinese.

These are all reasons that lead to weak competitiveness.We aim to impro ve that competiveness with a book written by Chinese-speaking writers,with a uniform language for jargons,who understand the technology gap as many of the writers work in the IBM Lab in Shanghai.We know there are many Chinese-speaking C++enthusiasts who are eager to learn and use the updates to their favorite language.The newness of C++11 also demands a strong candidate in t he beginner to intermediate level of C++11,which is the level of this book.

You should do well reading this book,if you are:

❑ an experienced C programmer who wants to see what C++11 can do for y ou.

❑ already a C++98/03 programmer who wants to learn the new C++11 langu age.

❑ anyone interested in learning the new C++11 language.

We structure this book using the design principles that Professor Bjarn e Stroustrup,the father of C++followed in designing C++11 through the Stand ard Committee.In fact,we separated this book into chapters based on those d esign principles.These design principles are outlined in the first chapter. The remaining chapters separate every C++11 language features under those d esign classifications.For each feature,it will explain the motivation for t he feature,the rules,and how it is used,taking from the approved C++11 pape rs that proposed these features.A further set of appendices will outline th e current state of the art of compiler support for C++11,incompatibilities, deprecated features,and links to the approved papers.

After reading this book,you should be able to answer questions such as:

❑ What is a lambda and the best way to use it?

❑ How is decltype and auto type inference related?

❑ What is move semantics and how does it solve the forwarding problem?

❑ I want to understand default and deleted as well as explicit overrides.

❑ What did they replace exception specifications with and how does noexcept work?

❑ What are atomics and the new memory model?

❑ How do you do parallel programming in C++11?

What we do not cover are the C++11 changes to the Standard library. This part could be a book itself and we may continue with this as Book II. This means we will not talk about the new algorithms, or new class libraries, but we will talk about atomics since most compilers implement atomics as a language feature rather than a library feature for efficiency reason. However, in the C++11 Standard, atomics is listed as a library feature simply because it could be implemented at worst as a library, but very few compilers would do that. This book is also not trying to teach you C++. For that, we particularly recommend Professor Stroustrup's book Programming principles&Practice Using C++which is based on an excellent course he taught at Texas A&M University on programming.

This book could be read chapter by chapter if you are interested in every feature of C++11. More likely, you would want to learn about certain C++11 feature and want to target that feature. But while reading about that feature, you might explore other features that fall under the same design guideline.

We hope you find this book useful in your professional or personal learning. We learnt too during our journey of collaborating in writing this book, as we wrote while building the IBM C++compiler and making it C++11 compliant.

The work of writing a book is long but it is well worth it.While I have been thinking about writing this book while working on the C++Standard,it was really Xiao Feng Guan who motivated me to start really stop thinking and start doing it for real.He continued to motivate and lead others through their writing assignment process and completed the majority of the work of organizingthis book.I also wish to thank many who have been my mentors officially and unofficially.There are too many to mention but people such as Bjarne Stroustrup,Herb Sutter,Hans Boehm,Anthony Williams,Scott Meyers and many others have been my teachers and great examples of leaders since I started reading their books and watching how they work within large groups.IBM has generously provided the platform,the time,and the facility to allow all of us to exceed ourselves,if only just a little to help the next generation of programmers.Thank you above all to my family Sophie,Cameron,Spot the Cat,and Susan for lending my off-time to work on this book.

Michael

序

当你在书店里拿起这本书的时候，可能最想问的就是：这么多C++的书籍，为什么需要选择这一本？回答这个问题首先需要知道的是，这是一本关于在2011年年底才制定通过的C++11（代码C++0x）的新标准的书籍。这个新标准看起来就像是一门新的语言，不仅有很多的新语言特性、标准库特性，而且在设计时就考虑了高度兼容于旧有的C++98/03标准。在2013年出版的C++的书籍中，本书是少数几部关于C++11的书籍之一，而其他的，则会是仅讲解C++98/03而未提及C++11的书籍。

相比于其他书籍，本书还有个显著特点——绝大多数章节都是由中国作者编写。事实上，本书所有作者均来自IBM XL C++编译器开发团队。而团队对于C++11新特性的开发，早在2008年就开始了。

而我则是一位IBM和加拿大的C++标准委员会的代表。我在编译器领域已工作了20多年。除了是IBM C++编译器开发团队的领导者之外，还是一些C++11特性的作者。

对于使用中文的C++用户而言，很多人还是喜欢阅读原生的中文图书，而非翻译版本，即使是在他们具备阅读其他语言能力的时候。虽然C++标准委员会的专家也在编写一些高质量的书籍，但是书籍从翻译到出版通常需要较长时间，而且一些词语或者意

义都可能在翻译中丢失。而翻译者通常也会觉得技术书籍的翻译是门苦差，很多行话、术语通常难以找到准确的中文表达方式。这么一来不同的翻译者会使用不同的术语，即使是在同一本书中，有时同一术语也会翻译成不同的中文。这些状况都是C++知识传播的阻碍，会拖慢C++11语言被中国程序员接受的进程。

基于以上种种原因，我们决定本书让母语是中文，并且了解国内外技术差距的IBM上海实验室的同事编写。我们知道，在中国有非常多的C++狂热爱好者正等着学习关于自己最爱的编程语言的新知识。而新的C++11也会招来大量的初级、中级用户，而本书也正好能满足这些人的需求。

所以，如果你属于以下几种状况之一，将会非常适合阅读本书：

- C语言经验非常丰富且正在期待着看看C++11新功能的读者。
- 使用C++98/03并期待使用新的C++11的程序员。
- 任何对新的C++11语言感兴趣的人。

在本书中，我们引述了C++之父Bjarne Stroustrup教授关于C++11的设计原则。而事实上，本书的章节划分也是基于这些设计原则的，读者在第1章可以找到相关信息，而剩余章节则是基于该原则对每个C++11语言进行的划分。对于每个特性，本书将根据其相关的论文展开描述，讲解如设计的缘由、语法规则、如何使用等内容。而书后的附录，则包括当前的C++11编译器支持状况、不兼容性、废弃的特性，以及论文的链接等内容。

在读完本书后，读者应该能够回答以下问题：

- 什么是lambda，及怎么样使用它是最好的？
- decltype和auto类型推导有什么关系？
- 什么是移动语义，以及（右值引用）是如何解决转发问题的？
- default/deleted函数以及override是怎么回事？
- 异常描述符被什么替代了？noexcept是如何工作的？
- 什么是原子类型以及新的内存模型？
- 如何在C++11中做并行编程？

对于标准程序库，我们在本书中并没有介绍。这部分内容可能会成为我们下一本书的内容。这意味着我们将在下一本书中不仅会描述新的算法、新的类库，还会更多地描述原子类型。虽然出于性能考虑，大多数的编译器都是通过语言特性的方式来实现原子类型的，但在C++11标准中，原子类型却被视为一种库特性，因其可以通过库的方式来实现。同样的，这样一本书也不会教读者基础的C++知识，如果读者想了解这方面的内容，我们推荐Stroustrup教授的《Programming principles＆Practice Using C++》（中文译为：《C++程序设计原理与实践》，华章公司已出版）。该书是Stroustrup教授以其在德克萨斯A＆M大学教授的课程为基础编写的。

对C++11特性感兴趣的读者可以顺序阅读本书。当然，读者也可以直接阅读自己感兴趣的章节，但是读者阅读时肯定会发现，这些特性基本和其他的特性一样，遵从了相同的设计准则。

我们也希望本书对你的职业或者个人学习起到积极的作用。当然，我们在合作写作本书，以及在为IBM C++编译器开发C++11特性时，也颇有收获。

本书的编写经历了较长的时间，但这是值得的。我在C++标准委员会工作的时候，只是在考虑写这样一本书，而官孝峰则让我从这样的考虑转到了动手行动。继而他还激励并领导其他成员共同参与，最终完成了本书。

此外，我要感谢我的一些正式的以及非正式的导师，比如Bjarne Stroustrup、Herb Sutter、Hans Boehm、Anthony Williams、Scott Meyers，以及许多其他人，通过阅读他们的著作，或观察他们在委员会中的工作，我学会了很多。当然，更要感谢IBM为我们提供的平台、时间，以及各种便利，因为有了这些最终我们才能够超越自我，为新一代的程序员做一些事情，即使这样的事情可能微不足道。还要感谢的是我的家人，Sophie、Cameron、Spot（猫）和Susan，让我能够在空闲时间完成书籍编写。

Michael

前言

为什么要写这本书

相比其他语言的频繁更新，C++语言标准已经有十多年没有真正更新过了。而上一次标准制定，正是面向对象概念开始盛行的时候。较之基于过程的编程语言，基于面向对象、泛型编程等概念的C++无疑是非常先进的，而C++98标准的制定以及各种符合标准的编译器的出现，又在客观上推动了编程方法的革命。因此在接下来的很多年中，

似乎人人都在学习并使用C++。商业公司在邀请C++专家为程序员讲课，学校里老师在为学生绘声绘色地讲解面向对象编程，C++的书籍市场也是百花齐放，论坛、BBS的C++板块则充斥了大量各种关于C++的讨论。随之而来的，招聘启事写着“要求熟悉C++编程”，派生与继承成为了面试官审视毕业生基础知识的重点。凡此种种，不一而足。于是C++语言“病毒性”地蔓延到各种编程环境，成为了使用最为广泛的编程语言之一。

十来年的时光转瞬飞逝，各种编程语言也在快马加鞭地向前发展。如今流行的编程语言几乎无一不支持面向对象的概念。即使是古老的语言，也通过了制定新标准，开始支持面向对象编程。随着Web开发、移动开发逐渐盛行，一些新流行起来的编程语言，由于在应用的快速开发、调试、部署上有着独特的优势，逐渐成为了这些新领域中的主流。不过这并不意味着C++正在失去其阵地。身为C的“后裔”，C++继承了C能够进行底层操作的特性，因此，使用C/C++编写的程序往往具有更佳的运行时性能。在构建包括操作系统的各种软件层，以及构建一些对性能要求较高的应用程序时，C/C++往往是最佳选择。更一般地讲，即使是由其他语言编写的程序，往往也离不开由C/C++编写的编译器、运行库、操作系统，或者虚拟机等提供支持。因此，C++已然成为了编程技术中的中流砥柱。如果用个比喻来形容C++，那么可以说这十来年C++正是由“锋芒毕露”的青年时期走向“成熟稳重”的中年时期。

不过十来年对于编程语言来说也是个很长的时间，长时间的沉寂甚至会让有的人认为，C++就是这样一种语言：特性稳定，性能出色，易于学习而难于精通。长时间使用C++的程序员也都熟悉了C++毛孔里每一个特性，甚至是现实上的一些细微的区别，比如各种编译器对C++扩展的区别，也都熟稔于心。于是这个时候，C++11标准的横空出世，以及C++之父Bjarne Stroustrup的一句“看起来像一门新语言”的说法，无疑让很多C++程序员有些诚惶诚恐：C++11是否又带来了编程思维的革命？C++11是否保持了对C++98及C的兼容？旧有的C++程序到了C++11是否需要被推倒重来？

事实上这些担心都是多余的。相比于C++98带来的面向对象的革命性，C++11带来的却并非“翻天覆地”式的改变。很多时候，程序员保持着“C++98式”的观点来看待C++11代码也同样是合理的。因为在编程思想上，C++11依然遵从了一贯的面向对象的思想，并深入加强了泛型编程的支持。从我们的观察来看，C++11更多的是对步入“成熟稳重”的中年时期的C++的一种改造。比如，像auto类型推导这样的新特性，展现出的是语言的亲和力；而右值引用、移动语义的特性，则着重于改变一些使用C++程序库时容易发生的性能不佳的状况。当然，C++11中也有局部的创新，比如lambda函数的引入，以及原子类型的设计等，都体现了语言与时俱进的活力。语言的诸多方面都在C++11中再次被锤炼，从而变得更加合理、更加条理清晰、更加易用。C++11对C++语言改进的每一点，都呈现出了经过长时间技术沉淀的编程语言的特色与风采。所以从这个

角度上看，学习C++11与C++98在思想上是一脉相承的，程序员可以用较小的代价对C++的知识进行更新换代。而在现实中，只要修改少量已有代码（甚至不修改），就可以使用C++11编译器对旧有代码进行升级编译而获得新标准带来的好处，这也非常具有实用性。因此，从很多方面来看，C++程序员都应该乐于升级换代已有的知识，而学习及使用C++11也正是大势所趋。

在本书开始编写的时候，C++11标准刚刚发布一年，而本书出版的时候，C++11也只不过才诞生了两年。这一两年，各个编译器厂商或者组织都将支持C++11新特性作为了一项重要工作。不过由于C++11的语言特性非常的多，因此本书在接近完成时，依然没有一款编译器支持C++11所有的新特性。但从从业者的角度看，C++11迟早会普及，也迟早会成为C++程序员的首选，因此即使现阶段编译器对C++新特性的支持还不充分，但还是有必要在这个时机推出一本全面介绍C++11新特性的中文图书。希望通过这样的图书，使得更多的中国程序员能够最快地了解C++11新语言标准的方方面面，并且使用最新的C++11编译器来从各方面提升自己编写的C++程序。

读者对象

本书针对的对象是已经学习过C++，并想进一步学习、了解C++11的程序员。这里我们假定读者已经具备了基本的C++编程知识，并掌握了一定的C++编程技巧（对于C++的初学者来说，本书阅读起来会有一定的难度）。通过本书，读者可以全面而详细地了解C++11对C++进行的改造。无论是试图进行更加精细的面向对象程序编写，或是更加容易地进行泛型编程，或是更加轻松地改造使用程序库等，读者都会发现C++11提供了更好的支持。

本书作者和书籍支持

本书的作者都是编译器行业的从业者，主要来自于IBM XL编译器中国开发团队。IBM XL编译器中国开发团队创立于2010年，拥有编译器前端、后端、性能分析、测试等各方面的人员，工作职责涵盖了IBM XL C/C++及IBM XL Fortran编译器的开发、测试、发布等与编译器产品相关的方方面面。虽然团队成立时间不长，成员却都拥有比较丰富的编译器开发经验，对C++11的新特性也有较好的理解。此外，IBM北美编译器团队成员Michael（他是C++标准委员会的成员）也参加了本书的编写工作。在书籍的编写上，Michael为本书拟定了提纲、确定了章节主题，并直接编写了本书的首章。其余作者则分别对C++11各种新特性进行了详细研究讨论，并完成了书稿其余各章的撰写工作。在书稿完成后，除了请Michael为本书的部分章节进行了审阅并提出修改意见外，我们

又邀请了IBM中国信息开发部及IBM北京编译器团队的一些成员对本书进行了详细的审阅。虽然在书籍的策划、编写、审阅上我们群策群力，尽了最大的努力，以保证书稿质量，不过由于C++11标准发布时间不长，理解上的偏差在所难免，因此本书也可能在特性描述中存在一些不尽如人意或者错误的地方，希望读者、同行等一一为我们指出纠正。我们也会通过博客（http://ibm.co/HKOGCx）、微博（www.weibo.com/ibmcompiler）发布与本书相关的所有信息，并与本书读者共同讨论、进步。

如何阅读本书

读者在书籍阅读中可能会发现，本书的一些章节对C++基础知识要求较高，而某些特性很可能很难应用于自己的编程实践。这样的情况应该并不少见，但这并不是这门语言缺乏亲和力，或是读者缺失了背景知识，这诚然是由于C++的高成熟度导致的。在C++11中，不少新特性都会局限于一些应用场景，比如说库的编写，而编写库却通常不是每个程序员必须的任务。为了避免这样的状况，本书第1章对C++11的语言新特性进行了分类，因此读者可以选择按需阅读，对不想了解的部分予以略过。一些本书的使用约定，读者也可以在第1章中找到。

致谢

在这里我们要对IBM中国信息开发部的陈晶（作者之一）、卢昉、付琳，以及IBM北京编译器团队的冯威、许小羽、王颖对本书书稿详尽细致的审阅表示感谢，同时也对他们专业的工作素养表示由衷的钦佩。此外，我们也要感谢IBM XL编译器中国开发团队的舒蓓、张嗣元两位经理在本书编写过程中给予的大力支持。而IBM图书社区的刘慎峰及华章图书的杨福川编辑的辛勤工作则保证了本书的顺利出版，在这里我们也要对他们以及负责初审工作的孙海亮编辑说声谢谢。此外，我们还要感谢各位作者的家人在书籍编写过程中给予作者的体谅与支持。最后要感谢的是本书的读者，感谢你们对本书的支持，希望通过这本书，我们能够一起进入C++编程的新时代。

IBM XL编译器中国开发团队

第1章　新标准的诞生

从最初的代号C++0x到最终的名称C++11，C++的第二个真正意义上的标准姗姗来迟。可以想象，这个迟来的标准必定遭遇了许多的困难，而C++标准委员会应对这些困难的种种策略，则构成新的C++语言基因，我们可以从新的C++11标准中逐一体会。而客观上，这些基因也决定了C++11新特性的应用范畴。在本章中，我们会从设计思维和应用范畴两个维度对所有的C++11新特性进行分类，并依据这种分类对一些特性进行简单的介绍，从而一览C++11的全景。

1.1　曙光：C++11标准的诞生

1.1.1　C++11/C++0x（以及C11/C1x）——新标准诞生

2011年11月，在印第安纳州布卢明顿市，“八月印第安纳大学会议”（August Indiana University Meeting）缓缓落下帷幕。这次会议的结束，意味着长久以来以C++0x为代号的C++11标准终于被C++标准委员会批准通过。至此，C++新标准尘埃落定。从C++98标准通过的时间开始计算，C++标准委员会，即WG21，已经为新标准工作了11年多的时间。对于一个编程语言标准而言，11年显然是个非常长的时间。其间我们目睹了面向对象编程的盛极，也见证了泛型编程的风起云涌，还见证了C++后各种新的流行编程语言的诞生。不过在新世纪第二个10年的伊始，C++的标准终于二次来袭。

事实上，在2003年WG21曾经提交了一份技术勘误表（Technical Corrigendum，简称TC1）。这次修订使得C++03这个名字已经取代了C++98成为C++11之前的最新C++标准名称。不过由于TC1主要是对C++98标准中的漏洞进行修复，核心语言规则部分则没有改动，因此，人们还是习惯地把两个标准合称为C++98/03标准。

注意　在本书中，但凡是C++98和C++03标准没有差异时，我们都会沿用C++98/03这样的俗称，或者直接简写为C++98。如果涉及TC1中所提出的微小区别，我们会使用C++98和C++03来分别指代两种C++标准。

C++11是一种新语言的开端。虽然设计C++11的目的是为了要取代C++98/03，不过相比于C++03标准，C++11则带来了数量可观的变化，这包括了约140个新特性，以及对C++03标准中约600个缺陷的修正。因此，从这个角度看来C++11更像是从C++98/03中孕育出的一种新语言。正如当年C++98/03为C++引入了如异常处理、模板等许多让人耳目一新的新特性一样，C++11也通过大量新特性的引入，让C++的面貌焕然一新。这些全新的特性以及相应的全新的概念，都是我们要在本书中详细描述的。

1.1.2　什么是C++11/C++0x

C++0x是WG21计划取代C++98/03的新标准代号。这个代号还是在2003年的时候取的。当时委员会乐观地估计，新标准会在21世纪的第一个10年内完成。从当时看毕竟还有6年的时间，确实无论如何也该好了。不过2010新年钟声敲响的时候，WG21内部却还在为一些诸如哪些特性该放弃，哪些特性该被削减的议题而争论。于是所有人只好接受这个令人沮丧的事实：新标准没能准时发布。好在委员会成员保持着乐观的情绪，还常常相互开玩笑说，x不是一个0到9的十进制数，而应该是一个十六进制数，我们还可以有A、B、C、D、E、F。虽然这是个玩笑，但也有点认真的意思，如果需要，WG21会

再使用“额外”的6年，在2015年之前完成标准。不过众所周知的，WG21“只”再花了两年时间就完成了C++11标准。

注意　C语言标准委员会（C committee）WG14也几乎在同时开始致力于取代C99标准。不过相比于WG21，WG14对标准完成的预期更加现实。因为他们使用的代号是C1x，这样新的C标准完成的最后期限将是2019年。事实上WG14并没用那么长时间，他们最终在2011年通过了提案，也就是C11标准。

从表1-1中可以看到C++从诞生到最新通过的C++11标准的编年史。

表 1-1　C++ 发展编年史

日　期	事　件
1990 年	*The Annotated C++ Reference Manual*，M.A.Ellis 和 B.Stroustrup 著。主要描述了 C++ 核心语言，没有涉及库
1998 年	第一个国际化的 C++ 语言标准：IOS/IEC 15882:1998。包括了对核心语言及 STL、locale、iostream、numeric、string 等诸多特性的描述
2003 年	第二个国际化的 C++ 语言标准：IOS/IEC 15882:2003。核心语言及库与 C++98 保持了一致，但包含了 TC1（Technical Corrigendum 1，技术勘误表 1）。自此，C++03 取代了 C++98
2005 年	TR1（Technical Report 1，技术报告 1）：IOS/IEC TR 19768:2005。核心语言不变。TR1 作为标准的非规范出版物，其包含了 14 个可能进入新标准的新程序库
2007 年 9 月	SC22 注册（特性）表决。通过了 C++0x 中核心特性
2008 年 9 月	SC22 委员会草案（Committee Draft, CD）表决。基本上所有 C++0x 的核心特性都完成了，新的 C++0x 标准草稿包括了 13 个源自 TR1 的库及 70 个库特性，修正了约 300 个库缺陷。此外，新标准草案还包括了 70 多个语言特性及约 300 个语言缺陷的修正
2010 年 3 月	SC22 最终委员会草案（Final Committee Draft, FCD）表决。所有核心特性都已经完成，处理了各国代表的评议

（续）

日　期	事　件
2011 年 11 月	JTC1 C++11 最终国际化标准草案（Final Draft International Standard, FDIS）发布，即 IOS/IEC 15882:2011。新标准在核心语言部分和标准库部分都进行了很大的改进，这包括 TR1 的大部分内容。但整体的改进还是与先前的 C++ 标准兼容的
2012 年 2 月	在 ANSI 和 ISO 商店可以以低于原定价的价格买到 C++11 标准

注意　语言标准的发布通常有两种——规范的（Normative）及不规范的（Non-normative）。前者表示内容通过了批准（ratified），因此是正式的标准，而后者则不是。不过不规范的发布通常是有积极意义的，比方说TR1，它就是不规范的标准，但是后来很多TR1的内容都成为了C++11标准的一部分。

图1-1比较了两个语言标准委员会（WG21，WG14）制定新标准的工作进程，其中一些重要时间点都标注了出来。

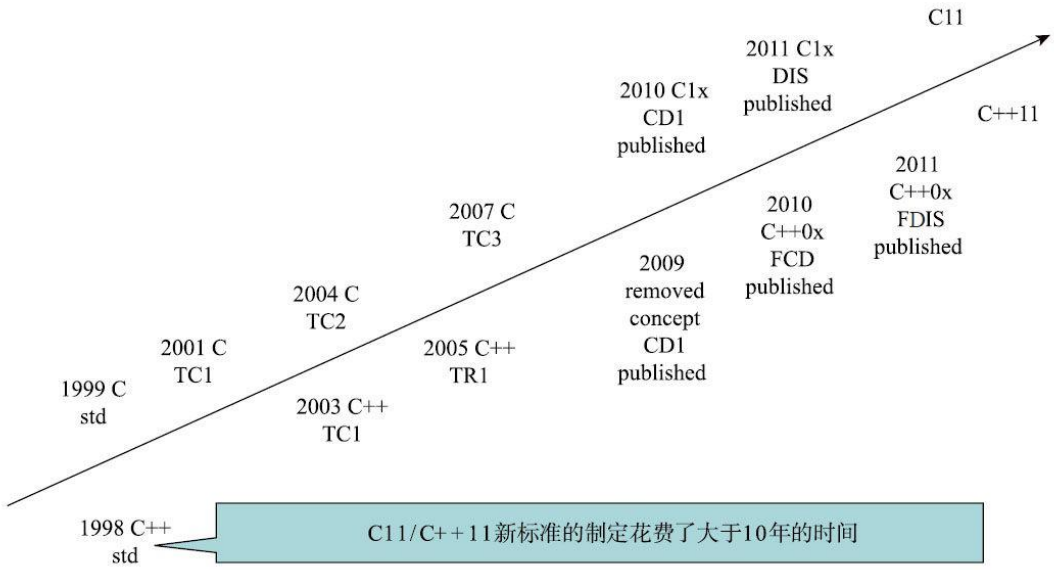


图　1-1　WG21和WG14制定新语言标准的工作进程

1.1.3　新C++语言的设计目标

如果读者已经学习过C++98/03，就可以发现C++98/03的设计目标如下：

- ❑ 比C语言更适合系统编程（且与C语言兼容）。
- ❑ 支持数据抽象。
- ❑ 支持面向对象编程。
- ❑ 支持泛型编程。

这些特点使得面向对象编程和泛型编程在过去的10～20年内成为编程界的明星。不过从那时开始，C++的发展就不仅仅是靠学者的远见前瞻去推动的，有时也会借由一些“奇缘”而演进。比方说，C++模板就是这样一个“奇缘”。它使得C++近乎成为了一种函数式编程语言，而且还使得C++程序员拥有了模板元编程的能力。但是凡事有两面，C++98/03中的一些较为激进的特性，比如说动态异常处理、输出模板，现在回顾起来则是不太需要的。当然，这是由于我们有了“后见之明”，或者由于这些特性在新情况下不再适用，又或者它们影响了C++11的新特性的设计。因此一部分这样的特性已经被C++11弃用了。在附录B中我们会一一列出这些弃用的特性，并分析其被弃用的原因。

而C++11的整体设计目标如下：

- 使得C++成为更好的适用于系统开发及库开发的语言。
- 使得C++成为更易于教学的语言（语法更加一致化和简单化）。
- 保证语言的稳定性，以及和C++03及C语言的兼容性。

我们可以分别解释一下。

首先，使C++成为更好的适用于系统开发及库开发的语言，意味着标准并不只是注重为某些特定的领域提供专业化功能，比如专门为Windows开发提供设计，或者专门为数值计算提供设计。标准希望的是使C++能够对各种系统的编程都作出贡献。

其次，使得C++更易于教学，则意味着C++11修复了许多令程序员不安的语言“毒瘤”。这样一来，C++语法显得更加一致化，新手使用起来也更容易上手，而且有了更好的语法保障。其实语言复杂也有复杂的好处，比如ROOTS、DEALII等一些复杂科学运算的算法，它们的作者非常喜爱泛型编程带来的灵活性，于是C++语言最复杂的部分正好满足了他们的需求。但是在这个世界上，新手总是远多于专家。而即使是专家，也常常只是精通自己的领域。因此语言不应该复杂到影响人们的学习。本书作者之一也是WG21中的一员，从结果上看，无论读者怎么看待C++11，委员会大多数人都认同C++11达成了易于教学这个目标（即使其中还存在着些看似严重的小缺陷）。

最后，则是语言的稳定性。经验告诉我们，伟大的编程语言能够长期存活下来的原因还是因为语言的设计突出了实用性。事实上，在标准制定过程中，委员会承担了很多压力，这些压力源自于大家对加入更多语言特性的期盼——每一个人都希望将其他编程语言中自己喜欢的特性加入到新的C++中。对于这些热烈而有些许盲目的期盼，委员会成员在Bjarne Stroustrup教授的引导下，选择了不断将许多无关的特性排除在外。其目的是防止C++成为一个千头万绪的但功能互不关联的语言。而如同现在看到的那样，C++11并非大而无序，相反地，许多特性可以良好地协作，进而达到“1+1＞2”的效果。可以说，有了这些努力，今天的读者才能够使用稳定而强大的C++11，而不用担心语言本身存在着混乱状况甚至是冲突。

值得一提的是，虽然在取舍新语言特性方面标准委员会曾面临过巨大压力，但与此同时，标准委员会却没有收集到足够丰富的库的新特性。作为一种通用型语言，C++是否是成功，通常会依赖于不同领域中C++的使用情况，比如科学计算、游戏、制造业、网络编程等。在C++11通过的标准库中，服务于各个领域的新特性确实还是太少了。因此很有可能在下一个版本的C++标准制定中，如何标准化地使用库将成为热门话题，标准委员会也准备好了接受来自这方面的压力。

1.2　今时今日的C++

1.2.1　C++的江湖地位

如今C++依旧位列通用编程语言三甲，不过似乎没有以前那么流行了。事实上，编程语言排名通常非常难以衡量。比如，某位教授或学生用了C++来教授课程应该被计算在内吗？在新的联合攻击战斗机（Joint Strike Fighter，JSF-35）的航空电子设备中使用了C++编程应该计算在内吗？又或者C++被用于一款流行的智能手机操作系统的编程中算不算呢？再或者是C++被用于编写最流行的在线付费搜索引擎，或用于构建一款热门的第一人称射击游戏的引擎，或用于构建最热门的社交网络的代码库，这些都该计算在内吗？事实上，据我们所知，以上种种都使用了C++编程。而且在构建致力于沟通软硬件的系统编程中，C++也常常是必不可少的。甚至，C++还常用于设计和编写编程语言。因此我们可以认为，编程语言价值的衡量标准应该包括数量、新颖性、质量，以及以上种种，都应该纳入“考核”。这样一来，结论就很明显了：C++无处不在。

1.2.2　C++11语言变化的领域

如果谁说C++11只是对C++语言做了大幅度的改进，那么他很可能就错过了C++11精彩的地方。事实上，读罢本书后，读者只需要看一眼代码，就可以说出代码究竟是C++98/03的，还是C++11的。C++11为程序员创造了很多更有效、更便捷的代码编写方式，程序员可以用简短的代码来完成C++98/03中同样的功能，简单到你惊呼“天哪，怎么能这么简单”。从一些简单的数据统计上看，比起C++98/03，C++11大大缩短了代码编写量，依情况最多可以将代码缩短30%～80%。

那么C++11相对于C++98/03有哪些显著的增强呢？事实上，这包括以下几点：

❑ 通过内存模型、线程、原子操作等来支持本地并行编程（Native Concurrency）。

❑ 通过统一初始化表达式、auto、declytype、移动语义等来统一对泛型编程的支持。

❑ 通过constexpr、POD（概念）等更好地支持系统编程。

❑ 通过内联命名空间、继承构造函数和右值引用等，以更好地支持库的构建。

表1-2列出了C++11批准通过的，且本书将要涉及的语言特性。这是一张相当长的表，而且一个个陌生的词汇足以让新手不知所措。不过现在还不是了解它们的时候。

但看过这张表，读者至少会有这样一种感觉：C++11的确像是一门新的语言。如果我们将C++98/03标准中的特性和C++11放到一起，C++11则像是个恐怖的“编程语言范型联盟”。利用它不仅仅可以写出面向对象语言的代码，也可以写出过程式编程语言代码、泛型编程语言代码、函数式编程语言代码、元编程编程语言代码，或者其他。多范型的支持使得C++11语言的“硬能力”几乎在编程语言中“无出其右”。

表 1-2　C++11 主要的新语言特性（中英文对照）

中文翻译	英文名称	备　注
__cplusplus 宏	__cplusplus macro	
对齐支持	alignment support	
通用属性	general attribute	
原子操作	atomic operation	
auto 类型推导（初始化类型推导）	auto (type deduction from 18nitialize)	
C99 特性	C99	
强类型枚举	enum class (scoped and strongly typed enums)	
复制及再抛出异常	copy and rethrow exception	本书未讲解
常量表达式	constexpr	
decltype	decltype	
函数的默认模板参数	default template parameters for function	
显式默认和删除的函数（默认的控制）	defaulted and deleted functions (control of defaults)	
委托构造函数	delegating constructors	
并行动态初始化和析构	Dynamic Initialization and Destruction with Concurrency	本书未讲解
显式转换操作符	explicit conversion operators	
扩展的 friend 语法	extended friend syntax	
扩展的整型	extended integer types	
外部模板	extern templates	
一般化的 SFINAE 规则	generalized SFINAE rules	
统一的初始化语法和语义	Uniform initialization syntax and semantics	
非受限联合体	unrestricted union	
用户定义的字面量	user-defined literals	
变长模板	variadic templates	
类成员初始化	in-class member initializers	
继承构造函数	inherited constructors	
初始化列表	initializer lists	
lambda 函数	lambda	
局部类型用作模板参数	local classes as template arguments	
long long 整型	long long integers	
内存模型	memory model	
移动语义（参见右值引用）	move semantics（see rvalue references）	
内联名字空间	Inline namespace	

（续）

中文翻译	英文名称	备　注
防止类型收窄	Preventing narrowing	
指针空值	nullptr	
POD	POD（plain old data）	
基于范围的 for 语句	range-based for statement	
原生字符串字面量	raw string literals	
右值引用	rvalue reference	
静态断言	static assertions	
追踪返回类型语法	trailing return type syntax	
模板别名	template alias	
线程本地的存储	thread-local storage	
Unicode	Unicode	

而从另一个角度看，编程中程序员往往需要将实物、流程、概念等进行抽象描述。但通常情况下，程序员需要抽象出的不仅仅是对象，还有一些其他的概念，比如类型、类型的类型、算法，甚至是资源的生命周期，这些实际上都是C++语言可以描述的。在C++11中，这些抽象概念常常被实现在库中，其使用将比在C++98/03中更加方便，更加好用。从这个角度上讲，C++11则是一种所谓的“轻量级抽象编程语言”（Lightweigh t Abstraction Programming Language）。其好处就是程序员可以将程序设计的重点更多地放在设计、实现，以及各种抽象概念的运用上。

总的来说，灵活的静态类型、小的抽象概念、绝佳的时间与空间运行性能，以及与硬件紧密结合工作的能力都是C++11突出的亮点。而反观C++98/03，其最强大的能力则可能是体现在能够构建软件基础架构，或构建资源受限及资源不受限的项目上。因此，C++11也是C++在编程语言领域上一次“泛化”与进步。

要实现表1-2中的各种特性，需要编译器完成大量的工作。对于大多数编译器供应商来说，只能分阶段地发布若干个编译版本，逐步支持所有特性（罗马从来就不是一天建成的，对吧）。大多数编译器已经开始了对C++11特性的支持。有3款编译器甚至从2008年前就开始支持C++11了：IBM的XL C/C++编译器从版本10.1开始。GNU的GCC编译器从版本4.3开始，英特尔编译器从版本10.1开始。而微软则从Visual Studio 2010开始。最近，苹果的clang/llvm编译器也从2010年的版本2.8开始支持C++11新特性，并且急速追赶其他编译器供应商。在本书附录C中，读者可以找到现在情况下各种编译器对C++11的支持情况。