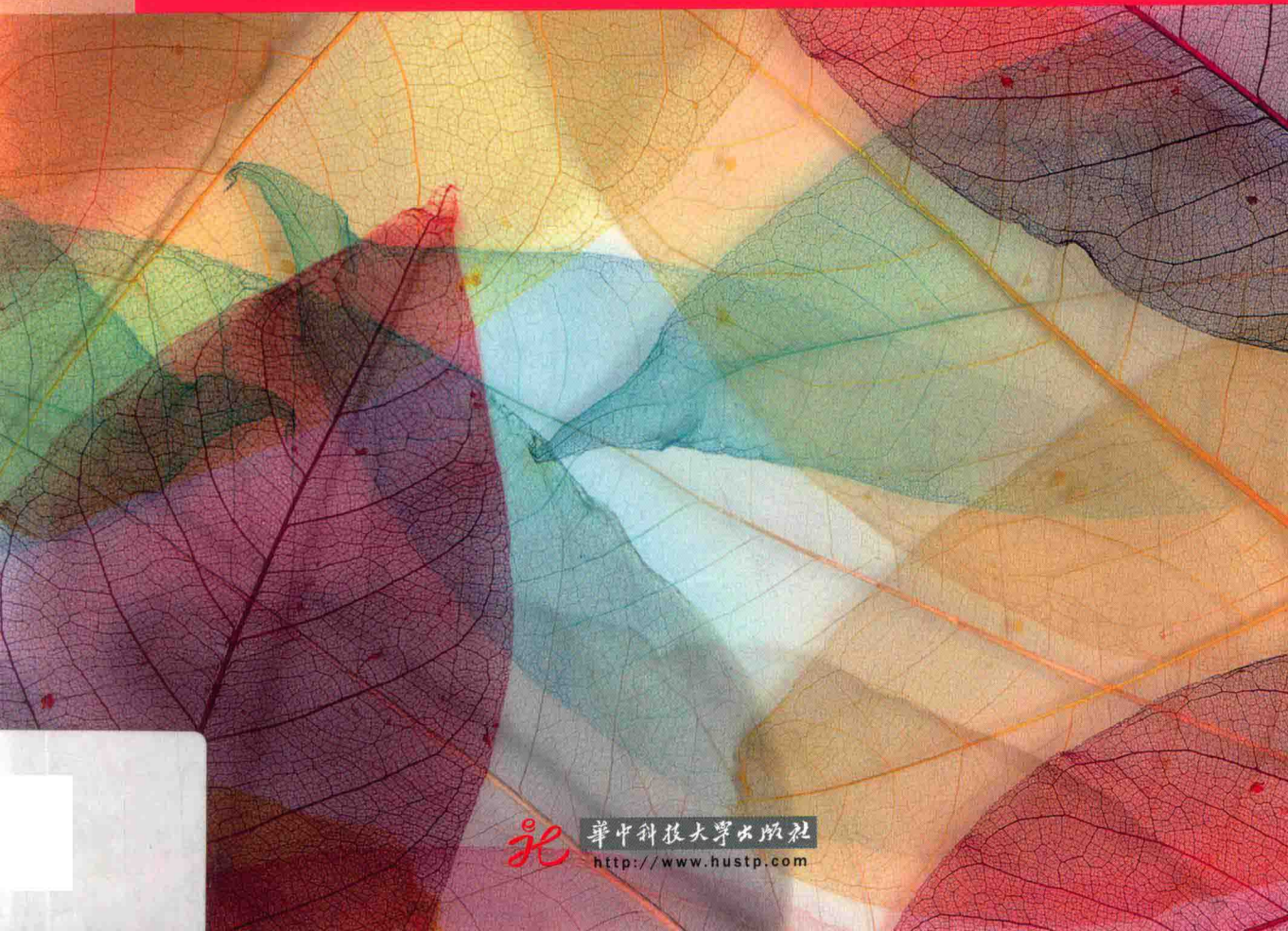
A Deep Dive into C++ Semantics and its Performance

剖析C++语义，从底层操作看C++ 的实现方法

C++语义和性能分析

杨 镰 著



華中科技大學出版社
http://www.hustp.com

C++语义和性能分析

杨 镰 著

华中科技大学出版社
中国·武汉

内 容 简 介

本书从C++的发展历史、类型系统、语义以及性能的视角，给读者展现了C++语言的基本理念和发展主线。一方面用抽象的理论框架，比如集合论、有限自动机、类型和类型系统等诠释了C++的理论方面；另一方面又从语义着手，从底层操作剖析了C++的实现方法。使读者既能站得高、看得远，又能把握C++的实质，从而全面掌握C++语言。

本书作者有20多年的C++开发经验，在微软总部工作期间，用C++参与开发过包括Windows XP系统在内的许多知名商用软件系统，在C++语言上有着深厚的功底。在本书作者看来，理解C++的精髓和灵魂，必须从C++类型系统着手，这样才能写出性能优异而又易于维护的系统。另外，从C++11到C++17的最新构造，都没有脱离C++类型系统的主体思想。熟读本书可以作为学习C++17的前奏。

图书在版编目(CIP)数据

C++语义和性能分析/杨镰著. —武汉：华中科技大学出版社，2017.11
ISBN 978-7-5680-3329-9

Ⅰ. ①C… Ⅱ. ①杨… Ⅲ. ①C++语言-程序设计 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2017)第211447号

C++语义和性能分析　　杨镰　著
C++ Yuyi he Xingneng Fenxi

策划编辑：徐晓琦　　责任校对：张　琳
责任编辑：陈元玉　　责任监印：周治超
出版发行：华中科技大学出版社(中国·武汉)　　电话：(027)81321913
武汉市东湖新技术开发区华工科技园　　邮编：430223
录　　排：华中科技大学惠友文印中心
印　　刷：湖北新华印务有限公司
开　　本：787mm×960mm　1/16
印　　张：14　　字　　数：287千字
版　　次：2017年11月第1版第1次印刷　　定　　价：43.80元

本书献给

我的父母杨奉军教授和杨烽老师，你们给了我身体和智慧。

我的太太 Ella，你给了我爱和宽容。

我的儿子杨安东和杨安仁，你们给了我欢乐和希望。

从C++的发展历史、类型系统、语义及性能的视角来重新认识C++语言。

序

我认为这是一本很特别的书，特别之处在于这是由一个有多年计算机开发经验的工程师带着对生活和哲学的思考写出的专业书，具有科学和人文融合的味道；同时作者又兼具中国和美国的计算机高等教育及工业界工作的背景，使此书又有了中西文化融合的味道。

作者说："本书一方面试图用一些抽象的理论框架，比如集合论、有限自动机、类型和类型系统等来诠释 C++语言的理论方面。一方面又从语义着手，从底层操作来看 C++语言的实现方法。目的是使读者既能站得高、看得远，又能把握 C++语言的实质，从而全面掌握 C++语言。"

我结合自己在大学教授 C++语言的有限经验和做算法理论研究的体会，就此谈点读后感，试作此书的序言。

1. 从语言表达的角度诠释 C++的理论方面

作者从 Smalltalk 和 C++切入，看到了一个充满哲理性的表达工具，帮助作者清晰地表达自己的逻辑思维和创造性。而我也因 Smalltalk 的机缘，从最初对语言本身的抽象性和多态性的不解，以及对面向对象与传统的面向过程程序设计的关系的困惑，到逐步开始思考 OOP 与认知的深层关系，这对我最终走上算法理论的研究有很大的促进作用。

我接触 Smalltalk 缘于我的硕士论文课题：设计一种多风格语言 LIPS——Lisp+Prolog+Smalltalk。感恩我的硕士论文导师张运桢老师，在当时学术界还不太开放的情况下，张老师勇敢地走出了国门，到美国的雪城大学进修，带回了一个使用 C 语言实现的 Lisp 系统——XLisp，当时的课题工作就是在此基础上完成的。

后来我去了法国，完成博士论文后进入大学教授 C++语言，但是后来感觉有些教不下去了，主要原因是不理解为什么当用 C++语言编写稍微复杂一点的程序时就很困难，特别是涉及模板和继承机制的使用时更是如此。如今读了本书有所感悟，作者说："它（C++）是植根于 C 以及 VNA 系统的，因此，一个对 VNA 不清楚的程序员，是不可能掌握 C++语言的。C++也并非一种数学上有深厚根基的语言，很难像 Lisp 语言那样用 Lambda 表达式和高阶函数等概念表达 C++。这其实是 C++程序不容易编写好的深层原因。"

2. 从机器的角度诠释 C++的实际方面

我认为此书与一般介绍 C++书籍的最大区别在于，作者是从机器的角度诠释 C++的实际方面。

程序语言作为形式语言，有语法与语义之别。以符号学的观点，语法指符号与符号之间的关系，语义指符号与指称之间的关系。一般介绍 C++的书籍大多将 C++作为一种程序语言，从 C++的程序结构出发，介绍各种语句的语法，结合各种例子，通过语法理解其语义。然而本书别具风格，使用“语义操作”（semantic operation）概念，从机器的角度诠释 C++的实际方面。语义操作就是通过下层操作实现上层语义，强调“操作”实现语义，而不是强调通过操作得到的“语义”（结果）。因此，语义操作可以表示机器实现对数学形式的“转换”，也可以是低层语言对高层语言的“理解”，从而将语法与语义有机结合起来。

图灵机作为计算和思维的工具，把时序-存储空间的概念作为计算的基本元素引入计算数学。从语义操作的角度诠释 C++的实际方面，有助于读者从算法和图灵机的角度理解 C++，使读者既能站得高、看得远，又能把握 C++的实质，从而全面掌握 C++语言。

总之，作者对机器、程序、程序语言和语言理论的内在关系看得比较清楚，能够在这么多的层次之间跨越很不容易，这使本书具有一定的哲学上的广度和逻辑上的一致性。相信此书会给有心的中国 IT 人士以启发！

柳 渝

于法国儒尔凡尔纳大学

2017 年 6 月

前　言

我与 C++

1988 年，我在美国波特兰州立大学（PSU）攻读计算机科学硕士学位时，第一次接触到 C++语言。当时，我的导师是波兰裔学者，名叫博卡斯基（Marek Perkowski，http://www.pdx.edu/profile/marek-perkowski）。他研究机器人和设计自动化（design automation）理论，喜欢“人工智能”领域里颇受青睐的 Lisp 程序设计语言，因此他也把对 Lisp 这种优美、简洁语言的欣赏传递给了他的研究生们。我也开始使用 Lisp 语言，常常能感受到这种语言的数学美和惊人的抽象能力。

在 PSU 那种松散的学术氛围中，不同的声音是一件平常的事情：博士候选人大卫·史密斯在一次讨论时说：

OOP 方法才是今后的方向；Functional 语言（如 Lisp）只能待在象牙塔里；数风流语言，还要看 C++语言。

这大概是我第一次听说 C++语言，我还将信将疑。不过，不久之后发生的一件事使我终身难忘，也证实了他的观点。在计算机系举办的一次“软件五角棋”大赛中，我使用 Lisp 语言编写了一个五角棋程序，自认为它非常优美、智慧且简洁，所以信心百倍。可是，第一轮就惨败给了我的对手。而对手的程序是使用 C++语言编写的！惨败的原因是，相比对手的程序，我的程序反应速度实在很慢，所以失分于超时。

这件事加深了我对 C++语言的印象，我决定尝试使用 C++语言和 OOP 方法！于是在 1988 年的冬季，我一学期同时选了 Smalltalk 语言和 C++语言两门课程。

从 Smalltalk 语言中，我认识到 OOP 方法的精髓。同时我也认识到 Smalltalk 语言在性能和实用性上的局限性；从 C++语言中，我体会到 OOP 方法中沉淀的哲学思想与实际应用的结合，以及程序设计的艺术性和工艺性。在学习 C++语言之前，我对于计算机科学的兴趣完全停留在表面。学习了 C++语言之后，我惊喜地发现：C++语言让我这个不喜欢烦琐、复杂系统的计算机科学新手看到了一个充满哲理性的表达工具，能够帮助我清晰地表达自己的逻辑思维和

创造性。另一方面，C++的实用性和强大的性能保障，也让我最终选择了用它来做毕业论文项目。

离开学校后不久，我就进入微软公司总部的微软研究院工作，从事 Windows NT 的性能研究。去微软公司面试的时间是 1995 年 8 月 17 日，距离 Windows 95 的轰动性发布只有一周时间。记得面试的一个 C++问题是论述“多态性（polymorphism）”是否意味着“动态绑定。”我的面试官告诉我：“C++没有动态绑定，只有静态绑定，而 COM（component object model）里才有真正的动态绑定。”我十分认真、激动地争辩道：“C++的 polymorphism 就是动态绑定。”两人从会议室争论到咖啡厅，然后又争论到坐落在美丽的“华盛顿湖畔”的一家意大利餐厅。晚上，当人事部门宣布面试结果时，我以为一定没戏，后悔自己太认真，不知道天高地厚。没想到我被那位面试官所在的团队——微软公司总部的微软研究院“软件性能研究中心（PPRC）”录用，而且我们一直是合作良好的同事。直到今天，我还认为我对了，他错了。但在微软公司当时的氛围里，“对”“错”并不是最重要的，重要的是“激情”和“执着”。而正是这种精神和文化，将 20 世纪 90 年代的微软和世界区分开来，让软件世界的“微软时代”降临。我也有幸加入到这个世界一流公司的研发队伍中，在几个重要的产品组里，贡献了自己的青春，实现了读研时的理想。20 世纪 90 年代，微软公司是充满活力和朝气的崛起的巨人。

记得有个朋友的儿子在美国东部一所大学读书，一次去他家过圣诞节，吃饭时他问我：“你怎么就能进微软，而我的许多朋友都进不去呢？”其实，我的真实答案是两个字：“运气”。但是我当时的回答是：“你必须每天花至少 20 分钟时间，读一些具体的计算机科学著作；然后至少花 10 分钟时间，想一些抽象的问题，比如宇宙大爆炸理论的逻辑错误和量子叠加的荒谬性等，只要和技术无关就好。”因为我一直以为：如果你对哲学没有兴趣，你不可能成为一个优秀的程序员；而做一个平庸的程序员太累了，不值得！

加入微软公司后，我有机会参与到多个大型的 C++项目的开发中。从 Windows NT 性能测试工具到 Windows Vista 的安全性，以及后来的 Windows 2008 服务器，C++都在实战中给我惊喜，但有时它也会给我带来噩梦。然而，我体会最深的，就是几乎每天我都会庆幸自己又学到了新的技能和思想，这种新东西可能是技术上的，也可能是方法上的，这种体会持续存在于我在微软公司工作的 5000 多个日日夜夜。

我发现，C++并不是一门简单、平庸的语言。正因为它的定位，它是植根于 C 以及 VNA 系统的，因此一个对 VNA 不清楚的程序员，是不可能掌握 C++语言的。C++也并非一门在数

学上有深厚根基的语言，很难像 Lisp 语言那样用 Lambda 表达式和高阶函数等概念表达 C++。这其实是 C++程序不容易编写好的深层原因。

在微软公司工作 10 多年，我接触过许多 C++资深程序员，比如《深度探索 C++对象模型》的作者 Stan Lippman。我是作为 VC++组的架构师，通过“发现”VC++在实现 C++ SPEC 方面的一些错误而结识了他，并数次就 C++的历史、哲学和设计模板问题与他交流。

然而，我接触更多的是 C++新手，比如我带领的多个美国常春藤名校的实习生，现在也早已成为各部门的高级主管。即使在这样一个世界一流的软件公司里，他们中间一些人对 C++语言的精神，尤其是它的语义层面上的规则，仍存在诸多的误解。一些通俗但并不准确的观点在业内长期流行：比如 C++的后门很多，C++的性能较 C 有较大差距等。造成这些误解，一方面是由于缺乏系统性的教育和简明易懂的著作，另一方面是 C++语言本身缺乏严格的数学、逻辑框架。

C++ 的现状

C++从 1979 年的开始构思到本书的写作（2016 年）之时，已经有 37 年的历史。有趣的是，C++的历史和中国改革开放的历史非常吻合，都是从 20 世纪 70 年代末开始谋划和实施的。今天，计算机硬件的能力和 20 世纪 80 年代的相比，已经不是一个数量级了，同时，计算机软件的格局，较之 20 世纪 80 年代，也发生了惊人的变化。今天的云计算、分布式计算和大数据计算的百花齐放，也预示着真正的软件革命已经悄悄来临。

和 C++语言同时代的程序设计语言，如 Basic、Cobol、Pascal 等，基本上已经进入程序设计历史博物馆。而在今天的系统程序设计、并行计算、服务程序、机器学习和嵌入式设备领域，C++语言仍然占有一席重要地位。这主要是因为它在性能上仍称雄于拥有诸多语言新秀的程序设计语言之林。

C++也面临着自身的问题和竞争对手的强烈挑战，其用户群在 Java 和 JavaScript 的压力下有缩小的趋势。除了特殊领域要求特殊的程序设计语言这一现实外，C++语言的复杂性和学习难度也会喧宾夺主地掩盖其强大的设计理念和语言功能。

从程序设计的发展来看，正是由于 C++语言的强大和复杂这对矛盾，才催生了后来的 Java、JavaScript 和 C#等“类 C++语言”。这些语言的共性就是简化了 C++语言中的复杂性，这样做

虽然从某种程度上牺牲了性能和灵活性，但是推动了程序设计在普通程序员中的技能熟练度和普及度。而这些新生代计算机语言，对 21 世纪开启的互联网应用、移动应用、云计算等软件工程新领域做出了巨大贡献。

那么是不是可以说 C++语言已经不适应今天的程序设计需求了呢？C++语言是不是已经发展到尽头了呢？

C++语言在这样强大的竞争环境里，也在 1998 年的标准化（ISO/IEC 14882:1998）之后，10 多年没有新版本出现。直到 2011 年，C++11 的推出又给了这门“古老”的语言以年轻的面孔。C++11 不是一个权宜之计的“面部拉皮”，而是继续沿着 C++先哲们的既定路线，在新的程序设计环境下的革新。C++11 在语法、功能和程序设计新范式的支持上，突破了自己的局限，让 C++这门强大的语言更显得生机盎然。

在这个背景下，我们欣慰地见证了许多新一代程序员对 C++语言兴趣的增长，这些新生代程序员的激情也激发了我撰写本书的热情。

本书不是一本 C++的入门读物，更不想穷尽 C++的特征和编程技巧。我们试图从 C++的历史和哲学入手，诠释它的一些重要设计决定，以及随之而来的操作语义和性能后果。本书也并非一本 OOP 方法论的著作，也不涉及 C++设计范式，而是尝试着从哲理及其语义这两个一高一低的层面来理解 OOP，理解 C++对 OOP 的独特贡献。

最近，在参考文献[8]中，Robert Hundt 通过对 loop recognition 图论算法的实现，比较了 C++、Java、Scala 和 GO 语言的性能后指出：

在性能上，C++语言遥遥领先对手。但是，它也需要十分细致的调试工作，这其中的一些工作十分老练、精致，非一般程序员所能掌握。

从以上分析可知，C++语言在高性能计算的领域里仍独占鳌头。其简单明了的结论是：C++语言不但没有过时，而且会在 GPU、大数据、人工智能等领域发挥更大的作用。

许多 C++新手没有受过严格的训练，也不了解 C++语言的语义和实质，这是妨碍 C++语言应用更上一层楼的主要因素。因此，本书的一个主要任务就是帮助读者从 C++语言的历史、哲学和语义层面上深入理解 C++，它的重点是从 C++的类型系统入手，告诉你 C++的精髓和进化路径，而不是教你熟悉它的语法规则。

水到渠成之后，本书会在最后一章分析 C++语言的性能特征，以及一些常用的性能提升途

径给出的一些可以复制的方案。希望这种分析能够提升 C++ “一般”程序员的实际工作能力，突破 Robert Hundt 在上面提到的一般程序员所面临的瓶颈，并且令其尽快从“一般”上升为“专家”。

书中多数问题的论证和结论都是通过实例来阐述的。大多数实例都是笔者自己编写，并且在自己的 Dell Precision M6700 Laptop 上运行、测试过的。这些实例都是用 VC++ 2012 版本编写的。如果需要本书中实例的源代码，可以通过华中科技大学出版社与笔者联系。

杨镰

2017 年 2 月

常用专业术语

在本书的开端，先用一定的篇幅把书中常用的专业术语、词汇整理出来，以便于用户查找。专业术语、词汇的熟练应用和深刻理解是了解主题的第一步，也是重要的一步。许多术语，如OOP（object oriented programming，面向对象编程）、VNA（Von Neumann Architecture，冯·诺依曼架构）等，对于计算机专业人士应该是信手拈来、不用解释的。但是，由于国内作者们往往喜欢用中文来替代原有的英文术语，这是一个“坏习惯”：它在一定程度上妨碍了国内计算机专业人士与国际接轨。所以，本书会频繁使用原本的英文术语。为了让对这些词汇陌生的读者方便查找，笔者把它们罗列在本书的最前面。

VNA（Von Neumann Architecture）——冯·诺依曼架构

VNA 是本书中用得很多的一个词，它代表计算机硬件体系结构上的冯·诺依曼架构，是匈牙利裔美国科学家冯·诺依曼在研究第一代电子数字计算机（EDVAC）时提出的。目前的计算机硬件体系基本上还是遵循 VNA 的。

VNA 是图灵理论计算机模型的一种物理实现。图 0-1 清楚地展示了 VNA 的主要构件。VNA 的特殊贡献在于程序和数据都存放于 main memory（主存储器）中的设计思想。虽然所有的现代计算机都遵循这一设计思想，并且这一思想在图灵理论计算机模型中也被表达出来，但是要真正实现这一设计思想仍需巨大努力。因为在 VNA 之前，程序和数据是两个不同的概念，前者是不能改变的，而后者是一个 volatile（易挥发性的）概念。VNA 所带来的灵活性，才真正让现代计算机系统的发展成为可能。

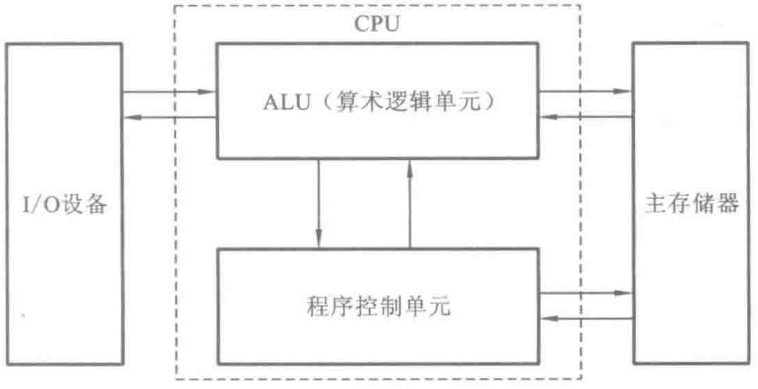


图 0-1　VNA 的一般结构

值得提出的是，VNA 是实现图灵机的一种“工程方法”。如果从概念上来说，则 VNA 是图灵机的一类。由于图灵机的抽象性和数学符号的晦涩性，VNA 更容易被计算机工程师认同、理解。

现代计算机硬件技术虽然已经取得了实质性的发展（根据摩尔定律），但是 VNA 最关键的一点是存储程序（stored-program）并没有变化。

在讨论面向对象编程时，不能忘记作为 20 世纪数字计算机基础的 VNA，因为它的特性决定了计算机软件的特性。OOP 的诞生是 VNA 上软件设计的需求，而 OOP 的性能和效率也是通过单机 VNA 系统的上下文来探讨的。

BS（Bjarne Stroustrup）—— C++ 的创始人

Bjarne Stroustrup 是 C++ 的创始人。本书因为时常谈及 C++ 的历史，所以 BS 在书中经常出现，既是为了“省事”，也是对 Bjarne Stroustrup 的一种“尊敬式的调侃”，所以千万不要把本书的 BS 当成美语中的口头禅（Bull Shit）!

OOP（object oriented programming）—— 面向对象编程

虽然本书不是一本 OOP 专著，但是任何关于 C++ 的书都不可能不涉及 OOP。如果还没听说 OOP，估计阅读本书也会很困难，那么建议先从入门书籍入手。

object —— 对象

笔者一直对将 object 翻译成“对象”持有保留意见，而对于 object 这样一个重要的概念，一定会在 C++ 著作中反复出现，所以本书未用中文术语“对象”，而是坚持用 object 本身。

class —— 类

类是 C++语言的核心，也是 UDT 的主要组成部分，class 通常是不用翻译的。

FP（functional programming）—— 函数式编程

比较 OOP 与 FP 的差别，是当前程序设计领域里的热门话题。FP 在分布式计算上的成功（比如 Spark 上的 Scala 语言），使得许多程序员开始重视这种程序设计方法学。作为概念上的互补，本书中会经常谈到 FP。

UDT（user defined types）—— 用户自定义类型

这个术语虽然不像 OOP 和 VNA 那么高大上，但是在本书中会反复地用到它，而且出现的频率相当高，这是因为 C++特别强调 UDT（比如 struct、class 等）和 FDT 的区别。本书也会在不同的主题上讨论 UDT 和 FDT 的语义差异。

FDT（fundamental data types）—— 基本数据类型

基本数据类型又称原生数据类型（native data types），也有人将其称为“built-in types”，本书沿用 BS 在参考文献[1]里的定义，将语言自带的数据类型（整数、浮点数、布尔值等）称为 FDT。

REF（reference）—— 引用

reference 是指针的一种比较安全的和实用的“语法糖”。一般翻译为“引用”，但是“引用”是一个动词，而 reference 是一个名词，所以翻译并不精确。与其另行翻译，不如使用原文，所以本书倾向于使用 REF 来简洁地代表 reference。

MI（multiple inheritance）—— 多重继承

MI 是 C++ 中 UDT 设计的一种常见模式。本书会讨论它的性能和语义。

GC（garbage collection）—— 垃圾回收

虽然本书不涉及环保理论，但 GC 却出现了许多次。新技术的涌现使得人类产生垃圾的能力与日俱增，从太空到大洋，从河流到山谷，当然也不能排除虚拟世界。在一个计算机程序的生命周期里，会有大量的内存垃圾产生，如何有效地控制和回收它们是一个持久的课题。在 OOP 世界里，回收使用后的 object，是 GC 的主要使命。

runtime —— 运行时

在描述 C++ 这类编译语言的文献中，runtime 效率、runtime 状态等十分频繁地被使用。区分什么是 compile-time（编译时）语义和 runtime 语义，在本书中也是讨论的重点。

side effect —— 副作用

side effect 是指一条计算机程序指令除了达到所预期的目的以外，有时还会产生预想不到的后果，即副作用。side effect 是发生在一个特定的 context 之内的，所以它是一个相对的概念。在 OOP 与 FP 的辩论中，side effect 通常会被认为是 OOP 的弱点。然而，换一个角度，side effect

在一定的 context 中则是程序设计中状态改变的方法，也是 OOP 达到计算目的的手段。

context —— 上下文（或者称为运行环境、操作环境、影响域、作用域等）

context 似乎包含多重寓意，然而它们在本质上都是近似的，甚至是相同的，它指的是一种操作所处的特定环境和作用域。context 在 OOP 中尤其重要，这是因为许多 OOP 的概念（如 static、mutable、side effect 等）都是在特定的 context 里定义的。

template —— 模板

模板编程是 C++ 的重要特征，我们没有理由不认识 template 这个对 C++ 程序员极为重要的术语。

size ——（对象的）字节长度

这里，size 是指一个 object 在内存中所占空间的字节长度，虽然也可以简单地称为“长度”，但是笔者认为这个词太常用了，最好不翻译。

type inference —— 类型推论

这是现代编译理论和类型理论的一个重要概念。

casting —— 转型

这个术语是 C++和 C 程序员经常使用的，无需翻译。

目录