深入 C++ 系列

Large-Scale C++ Software Design

大规模 C++ 程序设计

[美] John Lakos 著
李师贤 明仲 曾新红 刘显明 等 译

Addison Wesley

中国电力出版社
www.infopower.com.cn

深入 C++ 系列

深圳大学出版基金资助

Large-Scale C++ Software Design

大规模 C++ 程序设计

[美] John Lakos 著
李师贤 明仲 曾新红 刘显明 等 译

中国电力出版社

Large-Scale C++ Software Design(ISBN 0-201-63362-0)
John Lakos.

图书在版编目（CIP）数据

大规模 C++程序设计 /（美）勒科著；李师贤等译. —北京：中国电力出版社，2003
（深入 C++系列）
ISBN 7-5083-1504-9
Ⅰ.大... Ⅱ.①勒...②李... Ⅲ.C 语言－程序设计 Ⅳ.TP312
中国版本图书馆 CIP 数据核字（2003）第 033472 号

责任编辑：闫宏
丛 书 名：深入C++系列
书　　名：大规模C++程序设计
编　　著：（美）John Lokes
翻　　译：李师贤等
出版发行：中国电力出版社
地址：北京市三里河路6号 邮政编码：100044
电话：（010）88515918 传真：（010）88423191
印　　刷：北京地矿印刷厂
开　　本：787×1092 1/16　　**印　　张**：40　　**字　　数**：750千字
版　　次：2003年9月北京第一版
印　　次：2003年9月第一次印刷
定　　价：72.00 元

前　言

作为 Mentor Graphics 公司 IC 分部的成员，我有幸与许多富有才智的软件工程师一起工作，共同开发了一些非常大型的系统。

早在 1985 年，Mentor Graphics 公司就是最早用 C++开发实际的大型项目的公司之一。那时，无人知道如何做，也没有人预料到项目会出现这样的问题——费用失控、偏离计划、可执行代码庞大、性能低劣以及令人难以置信的昂贵的开发周期，这是不成熟的开发方法不可避免的结果。

许多有价值的经验都是经过痛苦的历程获得的。没有什么书可以帮助指导这种设计过程，也从未有人在这样的规模上尝试使用面向对象设计。

十年后，由于积累了大量有价值的经验，Mentor Graphics 公司已用 C++开发了数个大型软件系统，同时也为其他人在做同样的工作时不用再付出高昂代价开辟了道路。

在我 13 年的 C 语言（后来转为 C++）计算机辅助设计（CAD）软件开发生涯中，我多次看到了这样的情况：提前计划好总能生产出高质量、易维护的软件产品。在 Mentor Graphics 公司，我一直强调要从一开始就确保质量，要把质量作为设计过程中一个必不可少的部分。

1990 年我在 Columbia 大学开设了一门名为“面向对象设计与编程”的研究生课程。从 1991 年起，作为这门课程的老师，我有机会将我们在 Mentor Graphics 公司获得的许多经验传授给学生，这些经验是我们在从事工业化软件的开发过程中取得的。来自数百个研究生和专业程序员的提问和反馈信息，帮助我明确了很多重要的概念。本书是这些经验的总结。据我所知，这是第一本指导开发大型 C++项目的书，也是第一本针对大型 C++项目中出现的质量问题的书。我希望这些信息在读者的工作中有用，就像在我的工作中所起的作用一样。

读者对象

本书是专为有经验的 C++软件开发者、系统设计师、前摄的软件质量保证人员等编写的。本书尤其适合于那些从事大型软件开发工作（如数据库、操作系统、编译程序和框架）的人员阅读。

用 C++开发一个大型的软件系统，需要精通逻辑设计问题，在大多数有关 C++程序设计

的书中都包括了这些问题。若要进行有效的设计，还要求掌握物理设计概念，这些概念虽然与开发的技术方面紧密联系，但是其中有些方面即便是软件开发专家也可能经验很少或者没有经验。

本书提出的多数建议也适用于小型项目。我们开发项目时通常是先从小型项目开始，然后才开发更大更具挑战性的项目。一个特定项目的范围经常会扩展，因而开始时是小项目，后来就变成大项目了。但是，在大型项目中忽略好的策略所造成的直接后果，比在较小型项目中要严重得多。

本书将高层设计概念与特定C++编程细节结合起来，以满足下面两个需求：

（1）一本面向对象设计的书，尤其侧重于C++编程语言的实践方面。

（2）一本C++程序设计的书，描述如何使用C++编程语言来开发非常大型的系统。

请别弄错，这是一本高级别的书。本书既不适用于从头开始学习C++语法，也不适合于用来学习未曾掌握的C++语言细节，这是一本教你如何使用C++的全部功能去开发超大型系统的书。

简言之，如果读者认为自己C++掌握得很好，但想更多地学习如何使用C++语言去有效地开发大型项目，那么这本书就是为你所写的。

正文中的例子

多数人通过例子来学习。通常，我们提供说明现实世界设计的例子，避免那些只说明某一点问题而在设计的其他方面暴露出错误的例子；也避免那些只说明语言的细节但没有其他意义的例子。

除非特别指出，本书正文的所有例子都表示“好的设计”。因此，前几章的例子与整本书所介绍的所有策略一致。这种方法的不足之处是：读者看到示例代码与自己以往所见的代码不一致，但不能准确地知道为什么会这样。作者认为，若能利用书中所有的例子作为参考，就能弥补这个不足。

对于这种策略有两个显著的例外：注释和包前缀（package prefix）。本书正文中的许多例子的注释，为节省篇幅简单地省略了。有些地方虽然有注释，但也是少而精的。但是，这是一处要求读者“按我说的做，而不是按我做的那样做”的地方——至少在本书中如此。笔者在实践中编写接口时会当即非常仔细地添加注释，而不是事后再加注释，这一点读者可以放心。

第二个例外是，书中前面几个例子中的包前缀的使用不一致。在大型项目环境中需要包前缀，但是开始时，包前缀不便使用，需要一段时间适应。笔者选择先不使用注册的包前缀，在第7章正式提出之后再使用，以便能专注于介绍其它重要的基础内容。

当举例说明设计中的功能时，为了文字的简洁，在例子中使用了内联函数。在正文中多次说明了这种情况。由于本书的大部分内容与如何组织的问题（如，什么时候内联）有关，

所以笔者的倾向是在例子中避免使用内联函数。如果一个函数被声明为“inline”，肯定有其正当的理由，而不只是为了表示方便。

用C++开发大型系统是一系列的工程方法的折衷，几乎没有绝对。用“从不”或“总是”这样的词语来陈述是很吸引人的，但这样的词语只能用来对内容进行简单的描述。对于那些我希望将来阅读本书的C++程序员来说，这样总括性的陈述会引起异议——事实确实如此。为了避免陷入这种局面，我会在申明什么是（几乎是）肯定正确的同时，为例外的情况提供脚释或线索。

目前，有多种流行文件扩展名，用于区别C++头文件和C++实现文件。例如：

头文件扩展名：　.h　.hxx　.H　.h++　.hh　.hpp

实现文件扩展名：.c　.cxx　.C　.c++　.cc　.cpp

在所有的例子中，我们都使用.h扩展名来标识C++头文件，使用.c扩展名来标识C++实现文件。在本书中，我们会频繁地称头文件为.h文件，称实现文件为.c文件。最后需要说明的是，本书正文中的所有示例都在SUN SPARC工作站的CFRONT 3.0（SUN版本）上编译过并校验过语法；同时也在HP700系列机的自备C++编译器上进行过上述测试。当然，出现的任何错误都只能由作者负全责。

阅读导航

本书包含很多内容。不是所有的读者都有着相同的知识背景。所以我在第1章中提供了一些基本的（但却是必备的）知识以铺平学习道路。专业的C++程序员可以略过这一部分，或者需要的话简单地参考一下。第2章包含了一些基本的软件设计规则，我希望每一位有经验的开发人员都能够很快地认可。

第0章：引言

概述大型C++软件的开发人员所面临的问题。

第1部分：基础知识

第1章：预备知识

回顾了基本语言信息、一般设计模式以及本书的文体惯例。

第2章：基本规则

介绍了在任何C++项目中都应该遵循的重要设计经验。

后面的内容分为两大部分。前一部分题为“物理设计概念”，介绍了一系列与大型系统物理结构相关的重要论题。这些章节中的内容（第3~7章）集中在编程方面，对许多读者来说将是全新的，并且只精选了适合于大型程序设计的内容。这部分的论述是“自底向上”的，每一章的叙述都承接了前一章的内容。

第2部分：物理设计概念

第3章：组件

介绍一个系统的基础物理构筑模块。

第 4 章：物理层次结构

阐述建立组件层次结构的重要性，这些组件在物理上没有循环依赖，便于测试、维护和重用。

第 5 章：层次化

减少连接时依赖的特殊技术。

第 6 章：绝缘

减少编译时依赖的特殊技术。

第 7 章：包

将上述技术扩展到更大型的系统。

后一部分题为“逻辑设计问题”，研究逻辑设计与物理设计相结合的传统课题。这些章节（第 8~10 章）叙述了如何将一个组件作为一个整体来设计，总结了大量有关合理接口设计的问题，并研究了在大型项目环境中的实现问题。

第 3 部分：逻辑设计问题

第 8 章：构建一个组件

全面设计组件需要考虑的重要问题的总括。

第 9 章：设计一个函数

详细探讨生成一个组件的功能接口的问题。

第 10 章：实现一个对象

针对在大型项目环境中实现对象的若干组织问题。

附录中的主题是整本书的参考。

正文页下注中给出的书目的有关信息可再参阅书后的参考文献。

感谢

如果没有我在 Mentor Graphics 公司的许多同事的共同努力，这本书是不可能出现的。他们对于公司的划时代建设和发展作出了贡献。

首先，我要感谢我的朋友、同事和大学同学 Franklin Klein 对本书所作出的贡献。他实际上审读了本书每一页手稿。Franklin 对许多概念提供了解释——对大多数软件开发人员来说，这些都是新的概念。Franklin 的智慧、学识、社交能力、领会有效交流的细微差别的能力都远远超过我。他对内容修订、叙述顺序和表达风格等方面进行了详细的评审。

在排版期间，数位有天分和献身精神的专业软件人员检查了本书的大部分内容。他们愿意花费宝贵的时间审阅本书，对此我感到很幸运。我要感谢 Brad Appleton、Rick Cohen、Mindy Garber、Matt Greenwood、Amy Katriel、Tom O'Rourke、Ann Sera、Charles Thayer 和 Chris Van Wyk。他们花费大量的精力帮助我尽可能提高本书的价值。我要特别感谢 Rick Eesley，因为

他提供了丰富的评论和实际的建议——尤其是他建议在每一章的结尾作一个小结。

许多专业软件开发人员和质量保证工程师审阅了个别的章节。我要感谢 Samir Agarwal、Jim Anderson、Dave Arnone、Robert Brazile、Tom Cargill、Joe Cicchiello、Brad Cox、Brian Dalio、Shawn Edwards、Gad Gruenstein、William Hopkins、Curt Horkey、Ajay Kamdar、Reid Madsen、Jason Ng、Pete Papamichael、Mahesh Ragavan、Vojislav Stojkovic、Clovis Tondo、Glenn Wikle、Steve Unger 以及 John Vlissides，他们在技术上作出了贡献。我也要感谢 Mentor Graphics 公司的 Lisa Cavaliere-Kaytes 和 Tom Matheson，他们为书中的一些图表提出了宝贵的建议。另外我还要感谢 Eugene Lakos 和 Laura Mengel 所作的贡献。

自从本书首次印刷以来，我要感谢以下读者帮助我排除了一些我要负全责的不可避免的错误：Jamal Khan、Dat Nguyen、Scott Meyers、David Schwartz、Markus Bannert、Sumit Kumar、David Thomas、Wayne Barlow、Brian Althaus、John Szakmeister 以及 Donovan Brown。

如果不是在哥伦比亚大学收到了一封推销信，提供给我一份免费的有关 Rob Murray 的著作的评论拷贝，这本书也许永远也不会出现。因为我只在春季学期才教课，所以在我寄回所附表格的时候，要求将那本书寄到 Mentor Graphics 而不是哥伦比亚。之后不久，我接到了 Pradeepa Siva（她是 Addison-Wesley Corporate & Professional Publishing Group 的）打来的电话，要确定我这个不寻常要求的原因。在使她相信了这个要求的合理性（也许有一些是没有理由的夸大）之后，她说："我想我的老板可能会和你谈谈。"几天之后，我见到了她的老板——出版商。我一直很欣赏该出版社制作的"专业计算机丛书"的卓越品质，正是这种声誉最终使我答应为那套丛书撰写本书。

我非常感谢 Addison-Wesley Corporate & Professional Publishing Group 的成员。出版商 Join Wait，耐心地教给我许多关于人和交流的见识，这些东西将使我终生受益。从阅读大量书籍、与很多专业软件技术人员进行讨论、到站在书店观察潜在读者的购买习惯等方面，John Wait 一直尽力把握行业的脉搏。

以 Marty Rabinowitz 为首的产品人员在各方面都是优秀的。尽管与其他出版商联系的学术界的作者们对我表示担心，但是我仍然高兴地对待 Marty 所提出的在表达作者思想时应做到技术上准确、容易使用以及美学上有魅力的提炼与加工。我特别要感谢 Frances Scanlon，为了排印这本书，她付出了艰苦不懈的努力。

Brian Kernighan，本系列的技术编辑，在风格和实质上都提供了有价值的贡献，并发现了一些印刷错误和没有被其他人发现的不一致性。他的知识的广度和深度，与简洁的写作风格结合起来，对本系列丛书的成功作出了很大的贡献。

最后，因为对其他书中基础逻辑概念和设计原则的引证，我还想感谢本系列中的其他作者。

译者序

C++是当前的主流技术之一。随着我国社会与经济信息化工作的发展和深入，计算机应用水平和层次正在逐步提高，应用软件系统日益复杂化和大型化，我国软件业将面对越来越多的大型软件开发问题。然而，目前国内这一方面的书籍极少，而直接基于 C++大型软件开发的资料尤其缺乏。针对这一情况，我们翻译了《大规模 C++软件设计》一书，以满足读者的需求。

在实施大型和超大型 C++软件工程时，会出现许多意想不到的严重问题。若方法不当，轻则费用失控、进度延迟、执行代码臃肿、低效，重则可能导致开发项目完全失败。作者在总结多年来从事 C++大型工程的经验基础上，提出了物理设计和逻辑设计的一些新概念和新理论，阐明了在从事大型和超大型 C++软件工程时应遵循的一系列物理设计和逻辑设计原则，讨论了设计具有易测试、易维护和可重用等特性的高质量大规模 C++软件产品的方法。在说明这些概念、理论、方法和原则时，既有定性描述又有定量分析，还有许多生动的实例，并且对如何设计复杂系统进行了示范。书中的概念、理论、方法和原则对于大型软件工程的开发具有极强的现实指导意义，有利于提高软件的质量，保证大型项目的成功。作者 John Lakos 曾在 Mentor Graphic 公司（最早用 C++成功开发大型软件的公司之一）长期承担用 C++开发 CAD 软件的工作，书中积聚了他十余年的实践功力。作者从 1990 年起在美国哥伦比亚大学开设名为“面向对象设计与编程”的研究生课程，使本书中提及的方法和实例经受了课堂实践检验，并得到了进一步的总结提炼和理论升华。

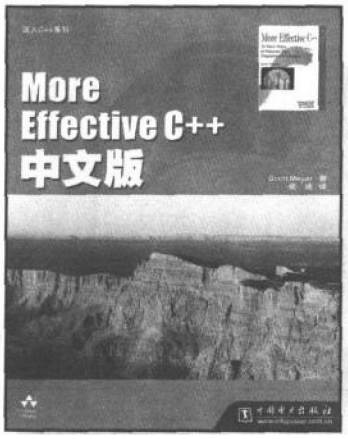
本书虽然主要面向有经验的 C++程序员、系统设计师、软件质量保证人员等中、高级程度的读者，但书中对于 C++面向对象程序设计的重要概念、设计模式与良好的编程风格的精彩再现，以及对于在任何 C++项目中都应该遵循的重要设计原则和指导方针的深刻揭示，无疑亦可以使初级程度的读者大受裨益。即使是针对大规模软件设计的一些理念和方法，亦无妨为中、小型软件开发所参考或借用。也许正是由于上述缘故，本书尽管定位于大规模软件设计这一高端问题，但并未因此而曲高和寡，其英文版问世至今，读者甚众，短短数年，重印已达九次之多。

参加本书翻译的有李师贤、明仲、曾新红、刘显明、王志刚、李皓、李智、梅晓勇、李

宏新、杜云梅、邱璟、孙恒、徐晶、舒忠梅、章远和熊春玲等。李师贤对全书进行了译校和审定。明仲初译了前言及偶数章节。曾新红初译了奇数章节。限于译校者水平所限，译文中的谬误之处在所难免，恳请读者批评指正。

译　者

深入 C++ 系列



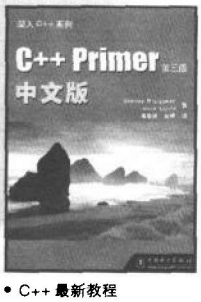
侯捷译著

- 影响深远的经典著作
- 35 条专家经验
- 条条精彩

- 最具权威性的 STL 书籍
- 侯捷评语：“另辟捷径”

经典著作　不容错过　循序渐进　可收宏效

- C++ 最佳入门书籍
- 快速掌握 C++ 的全新方法
- 单刀直入 C++ 核心部分

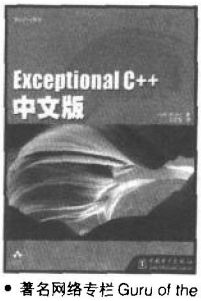


- C++ 最新教程
- C++ 最佳参考书
- 名著名译
- 相得益彰

- comp.lang.c++ 精华荟萃
- C++ 程序员进阶必备

- 斯坦福大学教授的呕心之作
- 与《Art of Computer Programming》齐名的算法巨著

- 经典权威的 C++ 进阶图书
- C++ 设计思想的精彩汇总
- 原汁原味的理论接触



- 著名网络专栏 Guru of the Week 的精华荟萃
- 47 条专家经验
- 条条精彩

目　录

第2部分 物理设计概念

第 3 部分 逻辑设计问题

0 引言

开发出好的 C++程序并非易事。当项目很大时，用 C++开发高可靠、易维护的软件则更加困难，并且将引出许多新概念。就像建筑师建造摩天大楼时不能使用从建造单个家庭房屋取得的经验一样，许多从 C++小型项目中学到的技术和方法也不能简单地用于大型项目的开发。

本书讨论的是如何设计非常大型的、高质量的软件系统，也是专门为有经验的软件开发者而编写的。本书可以指导他们构造出易维护、易测试的软件结构。本书不是一本有关程序设计的理论方法的书，而是引导开发者走向成功的完全的、实际的指导。它是精通 C++的程序员开发大型多节点系统的多年经验的总结。我们将示范如何设计复杂系统。这些复杂系统需要数百个程序员参加，有数千个类并且其源代码可能有数百万行。

本章将讨论在使用 C++开发大型项目时会遇到的一些问题，并为在前几章中我们必须做的基础工作提供环境知识。引言中用到的几个术语还没有定义，大多数术语应根据上下文理解。在后面的章节中，会给出这些术语的更精确的定义。真正的高潮将出现在第 5 章，我们将应用特殊的技术来减少 C++系统中的耦合（即相互依赖的程度）。

0.1 从 C 到 C++

在控制大型系统的复杂性方面，人们已充分认识了面向对象风范的潜在优点。在写作本书时，每 7 到 11 个月，C++程序员的数量就翻了一番[①]。在有经验的 C++程序员的手中，C++是人类发挥技能和工程才干的强有力的工具。然而，如果认为在大型项目中，只要使用 C++就能确保成功，那就完全错了。

① **stroustrup94**，7.1 节，163~164 页。

C++并不只是 C 的扩充：C++支持一种全新的风范。由于面向对象风范比对应的面向过程风范需要更多的努力和悟性，所以其名声并不好。C++比 C 更难掌握，而且有无数情形令程序员陷入困境。常常会出现这样的情况，即程序员会忽略一些严重的错误，等到他们发现了想要弥补这些错误，并使项目仍能满足进度要求时，却已经太晚了。即使相当小的失误，例如不加区分地使用虚函数或通过值来传递用户自定义类型，在完全正确的 C++程序中，也可能导致其运行速度比完成同样功能的 C 程序慢十倍。

许多程序员在开始接触 C++时，会经历这样的过程，在这个过程中，编程效率大大降低甚至陷入停顿，因为似乎要探索无数的设计方案。在此过程中，传统的程序设计员会深感不安，因为他们想要竭力掌握“面向对象”概念。

尽管对于最有经验的专业 C 程序员来说，C++的规模和复杂度在开始时都有点难以承受，但一个能干的 C 程序员要写出一个小的但不是微不足道的 C++程序，并不需要花太长的时间。然而，这种用 C++创建小型程序的未经训练的技术，来完成大型项目是完全不能胜任的。也就是说，C++技术的不成熟应用不适合大型项目，不谙此道所造成的后果甚多。

0.2 用 C++开发大型项目

就像 C 程序一样，一个很差的 C++程序可能会非常难以理解和维护。如果接口没有完全封装，则很难对其实现进行调整或加强。不良的封装会妨碍重用，也可能导致可测试性方面的优势完全消失。

与主流观点相反，从根本上说，最普通形式的面向对象程序要比对应的面向过程的程序更难测试和校验[①]。通过虚函数改变内部行为的能力可能使类不变式（class invariant）无效，而对于程序的正确性来说，类不变式是必要的。此外，一个面向对象系统的控制流路径的潜在数量可能十分巨大。

幸运的是，我们没有必要写这样霸道的（也是不好测试的）面向对象程序。可靠性可以通过限制只使用面向对象风范中的一个更容易测试的子集来获得。

当程序变得更大时，一种本质不同的力量会起作用。下面的小节会介绍一些我们可能会遇到的问题的具体例子。

0.2.1 循环依赖

作为一个软件专业人员，可能面临过这样一个局面：第一次看一个软件系统，似乎找不到一个合理的起点或者自身有完整意义的系统片段。不能理解或不能单独使用系统的任何一

① **perry**，13~19 页。