

C++

并发编程实战

C++
Concurrency
IN ACTION

[美] Anthony Williams 著
译

**翻译灾难
须读英文原著**

C++ 并发编程实战

具有多核的多处理器现已成为标配。C++ 语言的 C++11 版本为多线程应用程序提供了强大的支持，你需要掌握其原理、技巧以及新的并发语言特性，才能独领风骚。

本书帮助你循序渐进地学习用 C++11 编写健壮且优雅的多线程应用程序。你将学习线程内存模型、新的线程支持库，以及基础的线程启动和同步功能。与此同时，你还将学到如何解决并发应用程序中的棘手问题。

本书具有以下特色：

- 针对 C++
- 针对多核
- 用于学习

翻译灾难

须读英文原著

Anthony Williams 拥有十余年的 C++ 经验，并且是 BSI C++ 专家组的成员。

“有思想、有深度的指南，从专业人士那儿来的第一手资料。”

——Neil Horlock, Credit Suisse

“简化了 C++ 多线程的黑魔法”。

——Rick Wagner, Red Hat

“读这本书让我头痛，但痛定思痛”。

——Joshua Heyer, Ingersoll Rand

“作者展示了如何将并发变为现实。”

——Roger Orr, OR/2 Limited



人民邮电出版社-信息技术分社
<http://weibo.com/ptpitbooks>

MANNING

美术编辑：董志桢

分类建议：计算机 / 程序设计 / Web 设计
人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-38732-5



9 787115 387325 >

ISBN 978-7-115-38732-5

定价：89.00 元

内容提要

本书是一本基于 C++11 新标准的并发和多线程编程深度指南。内容包括 `std::thread`、`std::mutex`、`std::future` 和 `std::async` 等基础类的使用，内存模型和原子操作、基于锁和无锁数据结构的构建，以及并行算法和线程管理，最后还介绍了多线程代码的测试。本书的附录部分还对 C++11 新语言特性中与多线程相关的项目进行了简要的介绍，并提供了 C++11 线程库的完整参考。

本书适合于需要深入了解 C++多线程开发的读者，以及使用 C++进行各类软件开发的开发人员、测试人员阅读。使用第三方线程库的读者，也可以从本书后面的章节中了解到相关的指引和技巧。同时，本书

翻译灾难
须读英文原著

序

我是在离开大学后的第一份工作中遇到多线程编程的概念的。我们当时在编写一个数据处理应用程序，它需要用传入的数据记录来填充数据库。虽然数据很多，但每个数据都是独立的，并且在它被插入数据库前需要大量的处理。为了充分利用 10-CPU UltraSPARC 的能力，我们在多线程中运行代码，让每个线程处理它们自己的一组输入数据。在这个过程中，我们用 C++ 编写代码，使用 POSIX 线程，并且犯了相当多的错误。尽管多线程对我们而言都是全新的，但我们最终还是完成了。正是在这个项目中的工作，我第一次了解了 C++ 标准委员会和全新发布的 C++ 标准。

我对多线程和并发有前所未有的兴趣。虽然别人认为它困难、复杂，是问题之源，但我却视它为强大的工具，因为它可以允许你利用现有的硬件让代码运行得更快。随后我了解到它即便是在单核硬件上也能提高应用程序的响应和性能，通过使用多线程来隐藏诸如 I/O 这样耗费时间的操作延迟。我还了解了它怎样在 OS 级别工作以及 Intel CPU 如何处理任务切换。

同时，我对 C++ 的兴趣给我带来了与 ACCU 以及 BSI 的 C++ 标准专家组以及 Boost 接触的机会。我凭兴趣跟进了 Boost 线程库的初期开发，当它被最初的开发者抛弃时，我便趁机介入了。我从此成为了 Boost 线程库最主要的开发者和维护者。

当 C++ 标准委员会的工作从修正现有标准的缺陷转移到编写下一代标准（希望在 2009 年之前完成故命名为 C++0x，现在正式为 C++11，因为最终发布于 2011 年）的提案后，我更多地参与到 BSI 并且开始起草我自己的提案。多线程刚被明确地提上议事日程，我就全力以赴地投入进去，并且撰写或共同撰写了许多与多线程和并发相关的提案，它们组成了新标准的一部分。我感到很荣幸，可以有这个机会用这种方式组合我的计算机相关的两大兴趣——C++ 和多线程。

这本书借鉴了我在 C++ 和多线程上的全部经验，其目标是教会其他 C++ 开发者如何安全并有效地使用 C++11 线程库。我也希望能顺带传授一些我在这方面的热情。

译者简介

翻译灾难
须读英文原著

致 *Kim*、*Hugh* 和 *Erin*。

致谢

我要首先对我的妻子 Kim 说一声“谢谢”，感谢在我写这本书的时候她给我的爱和支持。写书占用了我最近 4 年很多的空闲时间，没有她的耐心、支持和理解，我不可能做到。

其次，我想感谢 Manning 的团队，包括总编 Marjan Bace、副总编 Michael Stephens、开发编辑 Cynthia Kane、编审 Karen Tegtmeier、文字编辑 Linda Recktenwald、校对 Katie Tennant 和制作经理 Mary Piergies。他们使得这本书成功面世，没有他们的努力，你现在就读不到这本书了。

我还想感谢 C++ 标准委员会的其他成员，他们为多线程工具上撰写了文件。正是 Andrei Alexandrescu、Pete Becker、Bob Blainer、Hans Boehm、Beman Dawes、Lawrence Crowl、Peter Dimov、Jeff Garland、Kevlin Henney、Howard Hinnant、Ben Hutchings、Jan Kristofferson、Doug Lea、Paul McKenney、Nick McLaren、Clark Nelson、Bill Pugh、Raul Silvera、Herb Sutter、Detlef Vollmann 和 Michael Wong，以及所有在文件上批注的人们，他们在委员会会议上进行了讨论，还有其他人的帮助才形成了 C++11 中的多线程和并发支持。

最后，我还想感谢下面的人：Dr. Jamie Allsop、Peter Dimov、Howard Hinnant、Rick Molloy、Jonathan Wakely 和 Dr. Russel Winder，他们的建议极大地改进了这本书。另外特别感谢 Russel 的详细审阅，还有技术校对 Jonathan，在编辑出版过程中极其用心地检查了终稿中所有内容中的错误（当然所有的错误都是由我造成的）。此外我想感谢我的审稿专家组：Ryan Stephens、Neil Horlock、John Taylor Jr.、Ezra Jivan、Joshua Heyer、Keith S. Kim、Michele Galli、Mike Tian-Jian Jiang、David Strong、Roger Orr、Wagner Rick、Mike Buksas 和 Bas Vodde。还要谢谢 MEAP 版的读者，他们花费时间来指出错误或是强调了需要阐明的部分。

前言

这本书是对新 C++ 标准中的并发和多线程工具的深度指南，内容包括从 `std::thread`、`std::mutex` 和 `std::async` 的基本用法，到复杂的原子操作与内存模型。

路线图

前 4 章介绍了由类库提供的各种类库工具以及他们如何使用。

第 5 章涵盖了内存模型和原子操作的低阶基础，包括原子操作怎样在其他代码上强制实行排序约束，并标志着导言章节的结束。

第 6 章和第 7 章开始涵盖高阶的论题，包括一些如何使用基础工具来构造更复杂的数据结构的示例——第 6 章中基于锁的数据结构，以及第 7 章中无锁的数据结构。

第 8 章继续高阶论题，包括如何设计多线程代码的指南，涵盖了影响性能的论点，以及各种并行算法的示范实现。

第 9 章涵盖了线程管理——线程池、工作队列和中断操作。

第 10 章包括了测试和调试——bug 的类型，定位它们的技巧，如何测试它们，等等。

附件包含了对由新标准引入的与多线程相关的一些新语言工具的简要介绍，第 4 章中提到的消息传递库的具体实现，以及 C++11 线程库的完整参考。

谁应该阅读本书

如果你打算用 C++ 编写多线程代码，你就应该阅读本书。如果你正要使用 C++ 标准库中新的多线程工具，这本书是必备的指南。如果你正使用替代的线程库，后面几章中的指引和技巧应该也是有用的。

假设你对 C++ 已经有了很好的了解，但对新的语言特性却不甚熟悉，这些在附录 A 中也能找到答案。假定你之前没有多线程编程的知识和经验，那就更应该阅读本书。

如何使用本书

如果你以前从未写过多线程代码，我建议你按顺序从头到尾阅读本书，可以跳过第 5 章中的细节部分，但第 7 章大量依赖第 5 章中的材料，所以如果你跳过了第 5 章，你一定要阅览第 7 章，除非你曾读过。

如果你之前未曾使用过 C++11 语言工具，在你开始确定准备快速开始书中例子之前最好浏览一下附录 A。新语言工具的使用凸显在文字之中，然而，当你遇到了之前没有见过的东西时，总是可以翻看附录的。

如果你在其他环境中拥有大量编写多线程代码的经验，开始的几章可能让你值得浏览一遍，以便你可以看看你了解的工具怎样映射到新 C++ 标准中。如果你打算用原子变量做一些低阶的工作，第 5 章就是必需的。为了确认你熟悉多线程 C++ 中类似异常安全的东西，值得阅览一下第 8 章。如果你在脑海中有特定的任务，索引和目录可以帮助你快速找到相关的章节。

一旦你打算促进 C++ 线程库的使用，附录 D 应该仍然有用，比如查询每个类和函数调用的细节。你可能会想一次又一次地翻回主章节，来刷新你对某一概念的使用或者看一看示例代码。

代码约定和下载

所有代码清单和正文中的源代码，出现等宽字体 (`like this`)，是为了便于从常规文本中区分出来。代码注解伴随着很多清单，指出重要的概念。在有些情况下，数字符号往往指向清单后面的注释。

本书中所有的工作示例源代码可以在出版社网址下载，www.manning.com/CPlusPlusConcurrencyinAction。

软件需求

为了直截了当地使用本书中的代码，你需要一个最新的 C++ 编译器，它支持示例中使用的新的 C++11 语言特性（参见附录 A），并且你需要 C++ 标准线程库的副本。

在撰写本书的时候，g++ 是我所知道的唯一带有标准线程库实现的编译器，尽管 Microsoft Visual Studio 2011 预览版也包括了实现。线程库的 g++ 实现最早是在 g++ 4.3 中引入了基本形式，并且在后来的版本中进行了扩展。g++ 4.3 也引入了一些新 C++11 语言特性的初次支持；对新语言特性更多的支持在各个后续版本中。详情参见 g++ C++11 状态页面¹。

Microsoft Visual Studio 2010 提供了一些新 C++11 语言特性，例如右值引用和 lambda

¹ GNU Compiler Collection C++0x/C++11 状态页面，<http://gcc.gnu.org/projects/cxx0x.html>。

函数，但并不带有线程库的实现。

作者的公司 Just Soft Solutions Ltd，出售为 Microsoft Visual Studio 2005、Microsoft Visual Studio 2008、Microsoft Visual Studio 2010 和 g++ 的各种版本¹设计的 C++11 标准线程库的完整实现。这些实现已被用来测试本书中的示例。

Boost 线程库²提供了基于 C++ 标准线程库提案的 API，可以移植到许多平台。本书中的大部分示例可以通过审慎地将 `std::` 替换成 `boost::` 进行修改，并使用适当的 `#include` 指令，来与 Boost 线程库一起工作。在 Boost 线程库中有少数工具不受支持（比如 `std::async`）或者拥有不同的名称（比如 `boost::unique_future`）。

作者在线

购买 C++ Concurrency in Action 包括了免费访问由 Manning 出版社运营的私有网络论坛，在这里你可以对本书做出评论，提问技术问题，并且从作者和其他用户那里得到帮助。如果要访问此论坛并订阅它，可以访问网址 www.manning.com/CPlusPlusConcurrencyinAction。该页面提供了论坛的使用指南以及论坛上的行为规则。

Manning 对我们读者的承诺是提供一个场所，在这里每个读者之间以及读者和作者之间可以进行有意义的对话。就作者而言并没有承诺任何规定的参与量，作者对本书论坛的贡献只是义务的（且无偿的）。我们建议你试着向作者提问一些有挑战性的问题，以免他失去兴趣！

作者在线论坛以及过往讨论的归档，在本书在印期间都可以从出版社网站进行访问。

¹ C++ 标准线程库的 `just::thread` 实现，<http://www.stdthread.co.uk>。

² Boost C++ 库集合，<http://www.boost.org>。

印刷资源

Cargill, Tom, “Exception Handling: A False Sense of Security,” in *C++ Report* 6, no. 9, (November-December 1994). Also available at http://www.informit.com/content/images/020163371x/supplements/Exception_Handling_Article.html.

Hoare, C.A.R., *Communicating Sequential Processes* (Prentice Hall International, 1985), ISBN 0131532898. Also available at <http://www.usingcsp.com/cspbook.pdf>.

Michael, Maged M., “Safe Memory Reclamation for Dynamic Lock-Free Objects Using Atomic Reads and Writes” in *PODC’02: Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing* (2002), ISBN 1-58113-485-1.

———. U.S. Patent and Trademark Office application 20040107227, “Method for efficient implementation of dynamic lock-free data structures with safe memory reclamation.”

Sutter, Herb, *Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions* (Addison Wesley Professional, 1999), ISBN 0-201-61562-2.

———. “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software,” in *Dr. Dobbs’s Journal* 30, no. 3 (March 2005). Also available at <http://www.gotw.ca/publications/concurrency-ddj.htm>.

在线资源

Atomic Ptr Plus Project Home, <http://atomic-ptr-plus.sourceforge.net/>.

Boost C++ library collection, <http://www.boost.org>.

C++0x/C++11 Support in GCC, <http://gcc.gnu.org/projects/cxx0x.html>.

C++11—The Recently Approved New ISO C++ Standard, <http://www.research.att.com/~bs/C++0xFAQ.html>.

Erlang Programming Language, <http://www.erlang.org/>.

GNU General Public License, <http://www.gnu.org/licenses/gpl.html>.

Haskell Programming Language, <http://www.haskell.org/>.

IBM Statement of Non-Assertion of Named Patents Against OSS, <http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf>.

Intel Building Blocks for Open Source, <http://threadingbuildingblocks.org/>.

The just::thread Implementation of the C++ Standard Thread Library, <http://www.stdthread.co.uk>.

Message Passing Interface Forum, <http://www.mpi-forum.org/>.

Multithreading API for C++0X—A Layered Approach, C++ Standards Committee Paper N2094, <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n2094.html>.

OpenMP, <http://www.openmp.org/>.

SETI@Home, <http://setiathome.ssl.berkeley.edu/>.

翻译灾难 须读英文原著

简要目录

第 1 章 你好，C++并发世界	1	第 9 章 高级线程管理	258
第 2 章 管理线程	13	第 10 章 多线程应用的测试与调试	285
第 3 章 在线程间共享数据	31	附录 A C++11 部分语言特性简明 参考	299
第 4 章 同步并发操作	63	附录 B 并发类库简要对比	324
第 5 章 C++内存模型和原子类型操作	97	附录 C 消息传递框架与完整的 ATM 示例	326
第 6 章 设计基于锁的并发数据结构	140	附录 D C++线程类库参考	344
第 7 章 设计无锁的并发数据结构	170		
第 8 章 设计并发代码	213		

目录

1

第1章 你好, C++并发世界 1

- 1.1 什么是并发 2
 - 1.1.1 计算机系统上的并发 2
 - 1.1.2 并发的途径 3
- 1.2 为什么使用并发 5
 - 1.2.1 为了划分关注点而使用并发 5
 - 1.2.2 为了性能而使用并发 6
 - 1.2.3 什么时候不使用并发 7
- 1.3 在C++中使用并发和多线程 8
 - 1.3.1 C++多线程历程 8
 - 1.3.2 新标准中的并发支持 9
 - 1.3.3 C++线程库的效率 9
 - 1.3.4 平台相关的工具 10
- 1.4 开始入门 11
- 1.5 小结 12

2

第2章 管理线程 13

- 2.1 基本线程管理 13
 - 2.1.1 启动线程 14
 - 2.1.2 等待线程完成 16
 - 2.1.3 在异常环境下的等待 17
 - 2.1.4 在后台运行线程 19
- 2.2 传递参数给线程函数 20
- 2.3 转移线程的所有权 23
- 2.4 在运行时选择线程数量 26

2.5 标识线程 28

2.6 小结 29

3

第3章 在线程间共享数据 31

- 3.1 线程之间共享数据的问题 32
 - 3.1.1 竞争条件 33
 - 3.1.2 避免有问题的竞争条件 34
- 3.2 用互斥元保护共享数据 35
 - 3.2.1 使用C++中的互斥元 35
 - 3.2.2 为保护共享数据精心组织代码 36
 - 3.2.3 发现接口中固有的竞争条件 38
 - 3.2.4 死锁: 问题和解决方案 44
 - 3.2.5 避免死锁的进一步指南 46
 - 3.2.6 用std::unique_lock灵活锁定 51
 - 3.2.7 在作用域之间转移锁的所有权 52
 - 3.2.8 锁定在恰当的粒度 54
- 3.3 用于共享数据保护的替代工具 56
 - 3.3.1 在初始化时保护共享数据 56
 - 3.3.2 保护很少更新的数据结构 59

3.3.3 递归锁 61

3.4 小结 62

4

第4章 同步并发操作 63

4.1 等待事件或其他条件 63

4.1.1 用条件变量等待条件 65

4.1.2 使用条件变量建立一个
线程安全队列 67

4.2 使用 future 等待一次性
事件 71

4.2.1 从后台任务中返回值 72

4.2.2 将任务与 future 相关联 74

4.2.3 生成(std::)promise 77

4.2.4 为 future 保存异常 79

4.2.5 等待自多个线程 80

4.3 有时间限制的等待 82

4.3.1 时钟 83

4.3.2 时间段 84

4.3.3 时间点 85

4.3.4 接受超时的函数 86

4.4 使用操作同步来简化
代码 88

4.4.1 带有 future 的函数式
编程 88

4.4.2 具有消息传递的同步
操作 92

4.5 小结 96

5

第5章 C++内存模型和原子 类型上操作 97

5.1 内存模型基础 98

5.1.1 对象和内存位置 98

5.1.2 对象、内存位置以及
并发 99

5.1.3 修改顺序 100

5.2 C++中的原子操作及
类型 100

5.2.1 标准原子类型 101

5.2.2 std::atomic_flag 上的
操作 103

5.2.3 基于 std::atomic<bool>的
操作 105

5.2.4 std::atomic<T*>上的操作:
指针算术运算 107

5.2.5 标准原子整型的
操作 108

5.2.6 std::atomic<>初级类

模板 109

5.2.7 原子操作的自由函数 111

5.3 同步操作和强制

顺序 112

5.3.1 synchronizes-with
关系 114

5.3.2 happens-before 关系 114

5.3.3 原子操作的内存
顺序 116

5.3.4 释放序列和
synchronizes-with 133

5.3.5 屏障 135

5.3.6 用原子操作排序非原子
操作 137

5.4 小结 138

6

第6章 设计基于锁的并发 数据结构 140

6.1 为并发设计的含义是
什么 141

6.2 基于锁的并发数据
结构 142

6.2.1 使用锁的线程
安全栈 142

6.2.2 使用锁和条件变量的线程
安全队列 145

6.2.3 使用细粒度锁和条件变量
的线程安全队列 149

6.3 设计更复杂的基于锁的
数据结构 160

6.3.1 编写一个使用锁的线程
安全查找表 160

6.3.2 编写一个使用锁的线程
安全链表 165

6.4 小结 169

7

第7章 设计无锁的并发数据 结构 170

7.1 定义和结果 171

7.1.1 非阻塞数据结构的
类型 171

7.1.2 无锁数据结构 172

7.1.3 无等待的数据结构 172

7.1.4 无锁数据结构的优点与
缺点 172

7.2 无锁数据结构的例子	173	8.3.2 其他数据结构中的数据访问方式	228
7.2.1 编写不用锁的线程安全栈	174	8.4 为并发设计时的额外考虑	230
7.2.2 停止恼人的泄漏: 在无锁数据结构中管理内存	178	8.4.1 并行算法中的异常安全	230
7.2.3 用风险指针检测不能被回收的结点	182	8.4.2 可扩展性和阿姆达尔定律	237
7.2.4 使用引用计数检测结点	189	8.4.3 用多线程隐藏延迟	238
7.2.5 将内存模型应用至无锁栈	194	8.4.4 用并发提高响应性	239
7.2.6 编写不用锁的线程安全队列	198	8.5 在实践中设计并发代码	241
7.3 编写无锁数据结构的准则	209	8.5.1 std::for_each 的并行实现	241
7.3.1 准则: 使用 std::memory_order_seq_cst 作为原型	210	8.5.2 std::find 的并行实现	243
7.3.2 准则: 使用无锁内存回收模式	210	8.5.3 std::partial_sum 的并行实现	248
7.3.3 准则: 当心 ABA 问题	210	8.6 总结	256
7.3.4 准则: 识别忙于等待的循环以及辅助其他线程	211		
7.4 小结	211		
第 8 章 设计并发代码	213	第 9 章 高级线程管理	258
8.1 在线程间划分工作的技术	214	9.1 线程池	259
8.1.1 处理开始前在线程间划分数据	214	9.1.1 最简单的线程池	259
8.1.2 递归地划分数据	215	9.1.2 等待提交给线程池的任务	261
8.1.3 以任务类型划分工作	219	9.1.3 等待其他任务的任務	265
8.2 影响并发代码性能的因素	222	9.1.4 避免工作队列上的竞争	267
8.2.1 有多少个处理器	222	9.1.5 工作窃取	269
8.2.2 数据竞争和乒乓缓存	223	9.2 中断线程	273
8.2.3 假共享	225	9.2.1 启动和中断另一个线程	274
8.2.4 数据应该多紧密	225	9.2.2 检测一个线程是否被中断	275
8.2.5 过度订阅和过多的任务切换	226	9.2.3 中断等待条件变量	276
8.3 为多线程性能设计数据结构	226	9.2.4 中断在 std::condition_variable_any 上的等待	279
8.3.1 为复杂操作划分数组元素	227	9.2.5 中断其他阻塞调用	281
		9.2.6 处理中断	281
		9.2.7 在应用退出时中断后台任务	282
		9.3 总结	284
		第 10 章 多线程应用的测试与调试	285
		10.1 并发相关错误的	

类型	285
10.1.1 不必要的阻塞	286
10.1.2 竞争条件	286
10.2 定位并发相关的错误的技巧	288
10.2.1 审阅代码以定位潜在的错误	288
10.2.2 通过测试定位并发相关的错误	290
10.2.3 可测试性设计	291
10.2.4 多线程测试技术	292
10.2.5 构建多线程的测试代码	295
10.2.6 测试多线程代码的性能	297
10.3 总结	298

附录 A C++11 部分语言特性 简明参考	299
附录 B 并发类库简要对比	324
附录 C 消息传递框架与完整的 ATM 示例	326
附录 D C++线程类库 参考	344

**翻译灾难
须读英文原著**