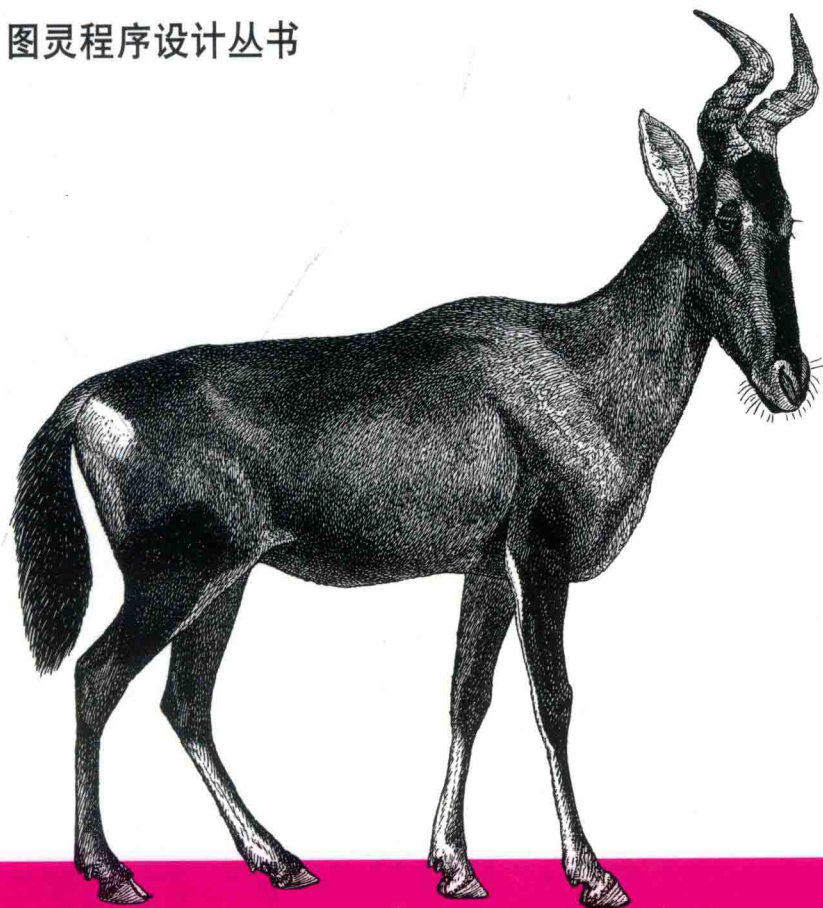


O'REILLY

TURING

图灵程序设计丛书



C++性能优化指南

Optimized C++: Proven Techniques for Heightened Performance

精选编程中频繁使用和能够带来显著性能提升效果的技术

[美] Kurt Guntheroth 著

杨文轩 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS



图灵程序设计丛书

C++性能优化指南

Optimized C++

[美] Kurt Guntheroth 著

杨文轩 译

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

O'Reilly Media, Inc. 授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目 (C I P) 数据

C++性能优化指南 / (美) 柯尔特·甘瑟尔罗斯
(Kurt Guntheroth) 著 ; 杨文轩译. — 北京 : 人民邮
电出版社, 2018.1

(图灵程序设计丛书)

ISBN 978-7-115-47139-0

I. ①C… II. ①柯… ②杨… III. ①C语言—程序设
计—指南 IV. ①TP312.8-62

中国版本图书馆CIP数据核字(2017)第300400号

内 容 提 要

本书是一本 C++ 代码优化指南。作者精选了他在近 30 年编程生涯中最频繁使用的技术和能够带来最大性能提升效果的技术,旨在让读者在提升 C++ 程序的同时,思考优化软件之美。书中主要内容有:代码优化的意义和总原则,与优化相关的计算机硬件背景知识,性能分析方法及工具,优化字符串的使用,算法、动态分配内存、热点语句、查找与排序等等的优化方法。

本书适合所有 C++ 程序员,也可供其他语言的程序员优化代码时作为参考。

-
- ◆ 著 [美] Kurt Guntheroth
 - 译 杨文轩
 - 责任编辑 朱巍
 - 责任印制 彭志环
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
 - 邮编 100164 电子邮件 315@ptpress.com.cn
 - 网址 <http://www.ptpress.com.cn>
 - 大厂聚鑫印刷有限责任公司印刷
 - ◆ 开本: 800×1000 1/16
 - 印张: 19
 - 字数: 449千字 2018年1月第1版
 - 印数: 1—3 500册 2018年1月河北第1次印刷
 - 著作权合同登记号 图字: 01-2017-4807号
-

定价: 89.00元

读者服务热线: (010)51095186转600 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京东工商广登字 20170147 号

版权声明

©2016 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2018. Authorized translation of the English edition, 2018 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2016。

简体中文版由人民邮电出版社出版，2018。英文原版的翻译得到 O'Reilly Media, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

O'Reilly Media, Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O'Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过图书出版、在线服务或者面授课程，每一项 O'Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar 博客有口皆碑。”

——*Wired*

“O'Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O'Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

每一本书在出版时，作者都会感谢其伴侣。我知道这种方式平淡无奇，但我还是要感谢我的妻子 Renee Ostler，是她让本书得以出版。她为我腾出了家里的空间，并让我能够腾出几个月的时间专心写作。尽管她不太了解 C++ 性能优化，但她会一直陪伴我到很晚，问我关于这方面的问题，只是为了表示她的支持。她认为这本书对她很重要，因为它对我很重要。有妻若此，夫复何求。

前言

读者朋友，你好！我的名字叫 Kurt，是一名狂热的编程爱好者。

我编写软件已经超过 35 年了。我从未就职于微软、谷歌、Facebook、苹果或者其他知名公司。但是在这些年里，除了几个短暂的假期外，我每天都在写代码。最近 20 年里，我几乎只编写 C++ 程序，并与其他睿智的开发人员讨论 C++。所以我有资格写一本关于优化 C++ 代码的书。我还发表过许多文章，包括规范、评论、手册、学习笔记以及博客文章 (<http://oldhandsblog.blogspot.com>) 等。有时令我吃惊的是，我共事过的睿智能干的程序员中，只有半数能将两个符合英文语法的句子合并在一起。

我最喜欢的名言之一出自艾萨克·牛顿爵士的一封信。他在信中写道：“我之所以看得更远，是因为我站在巨人的肩上。”现在，我也站在巨人的肩上了，特别是阅读过他们的著作：有优雅的小书，如 Brian Kernighan 和 Dennis Ritchie 合著的《C 程序设计语言》；有充满智慧且走在技术前沿的书籍，如 Scott Meyers 的 *Effective C++* 系列；有充满挑战又能扩展思维的书籍，如 Andrei Alexandrescu 的《C++ 设计新思维》；有科学严谨且讲解准确的书籍，如 Bjarne Stroustrup 和 Margaret Ellis 合著的 *The Annotated C++ Reference Manual*。在我职业生涯的大部分时间，我都从没想过有一天可以自己写一本书。但是突然有一天，我发现我需要写这本书。

那么为什么我要写一本关于 C++ 性能优化的书呢？

在 21 世纪初期，C++ 曾一度受到诟病。C 语言的支持者指出 C++ 程序的性能不如以 C 语言编写的相同代码。拥有巨额营销预算的著名企业吹嘘它们自己的面向对象语言，宣称 C++ 语言难以使用，而它们的工具才是未来。各大高校也决定教授 Java 语言，因为它有一套免费的工具链。由于以上种种原因，大公司投资大笔金钱使用 Java、C# 或是 PHP 来编写网站和操作系统。C++ 看起来正在逐渐衰落。对于任何相信 C++ 语言是强大且有用的工具的人而言，那是一段困难的时期。

就在这时，一件有趣的事情发生了。处理器核心的处理速度停止增长，但是工作负荷在持续加大。于是，那些大公司又开始重新雇用 C++ 程序员去解决它们的扩容问题。用 C++ 从头开始重新编写代码的成本变得比在数据中心消耗的电费要便宜。突然之间，C++ 再度流行起来了。

C++ 与 2016 年年初那些高市场份额的编程语言¹ 相比非常突出的一点是，它为开发人员提供了一连串的可选实现方式，从全自动、自动支持到精准手动控制。C++ 赋予了开发人员掌控性能权衡的力量，这种掌控让性能优化成为可能。

市面上介绍如何优化 C++ 代码的书并不多。其中之一是由 Dov Bulka 与 David Mayhew 精心研究所著，不过现在看来有些过时的《提高 C++ 性能的编程技术》。这两位作者似乎与我有相似的职业经历，也发现了很多相同的优化原则。若想看看其他人怎么看待本书中提出的问题，推荐从《提高 C++ 性能的编程技术》开始。另外，Scott Meyers 等人也广泛地讨论过如何避免使用复制构造函数。

需要掌握的关于代码优化的知识太多了，足够写出 10 本书。在本书中，我精选出了自己工作中频繁使用的技术和能够带来最大性能提升效果的技术进行讲解。奋战在性能优化前线的读者可能会有疑问，为什么本书中没有介绍任何有助于他们解决问题的对策。对于这些疑问，我只能说：“对不起，内容太多，篇幅有限。”

欢迎大家将勘误、评论和最喜欢的优化策略发送至 antelope_book@guntheroth.com。

我热爱软件开发。我享受永不停歇地练习每一种新的循环方式和接口。编写代码是一门科学，也是一门写诗艺术；它是一项冷僻的技术，一种内在的艺术形式，以至于除了少数同行外几乎没有人懂得欣赏它。优雅的函数中蕴藏着美感，被广泛使用的强大的惯用法中蕴藏着智慧。但令人遗憾的是，每一部史诗般的软件诗篇（如 Stepanov 的标准模板库），都对应着 10 000 行单调、庞大、枯燥的代码。

本书的根本目标是帮助读者思考优化软件之美。请记住这一点并践行之。请看得更远一些！

为本书中的代码致歉

虽然我已经编写和优化 C++ 代码超过 20 年了，但是本书中所出现的大多数代码都是特意为本书编写的。就像所有新编写的代码一样，这些代码也会有缺陷。为此，我深表歉意。

多年以来，我一直为 Windows、Linux 和各种嵌入式系统进行开发，而本书中展示的代码则是我在 Windows 上编写的。因此，毫无疑问，本书中的代码和内容都更加偏向于 Windows。那些使用 Windows 操作系统中的 Visual Studio 讲解的优化 C++ 代码的技术，同样也适用于 Linux、Mac OS X 或者其他 C++ 环境。不过，不同优化方式的正确使用时机则取决于编译器和标准库的实现以及代码是在哪种处理器上测试的。优化是一门实验科学。盲目地信任优化建议往往会失望。

我知道在不同的编译器之间、Unix 系统和嵌入式系统之间存在的兼容性是非常难以应对的。如果书中的代码无法在你最喜爱的系统上编译通过，我感到非常抱歉。本书并不会讲解跨系统的兼容性，我个人倾向于展示简单的代码。

我并不喜欢下面这种大括号缩进风格：

```
if (bool_condition) {  
    controlled_statement();  
}
```

注 1：可以从 <http://www.tiobe.com/tiobe-index/> 查询各编程语言的市场份额。——译者注

不过，这种风格的优点是可以在一页中放入更多的代码行，因此，我选择在本书中使用这种风格。

示例代码的使用

可从以下地址下载本书示例代码、解决方案示例等附带资料：guntheroth.com。

本书是要帮你完成工作的，所以书中的示例代码通常可以直接拿去用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN，比如“*Optimized C++* by Kurt Guntheroth(O'Reilly). Copyright 2016 Kurt Guntheroth, 978-1-491-92206-4”。

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

排版约定

本书使用了下列排版约定。

- 黑体
表示新术语。
- 等宽字体 (constant width)
表示程序片段，以及正文中出现的变量、函数名、环境变量、语句和关键字等。

目录

前言	xvii
第 1 章 优化概述	1
1.1 优化是软件开发的一部分	2
1.2 优化是高效的	3
1.3 优化是没有问题的	3
1.4 这儿一纳秒，那儿一纳秒	5
1.5 C++ 代码优化策略总结	5
1.5.1 用好的编译器并用好编译器	6
1.5.2 使用更好的算法	7
1.5.3 使用更好的库	8
1.5.4 减少内存分配和复制	9
1.5.5 移除计算	9
1.5.6 使用更好的数据结构	9
1.5.7 提高并发性	10
1.5.8 优化内存管理	10
1.6 小结	10
第 2 章 影响优化的计算机行为	11
2.1 C++ 所相信的计算机谎言	12
2.2 计算机的真相	12
2.2.1 内存很慢	13
2.2.2 内存访问并非以字节为单位	13
2.2.3 某些内存访问会比其他的更慢	14

2.2.4	内存字分为大端和小端	14
2.2.5	内存容量是有限的	15
2.2.6	指令执行缓慢	16
2.2.7	计算机难以作决定	16
2.2.8	程序执行中的多个流	16
2.2.9	调用操作系统的开销是昂贵的	18
2.3	C++ 也会说谎	18
2.3.1	并非所有语句的性能开销都相同	18
2.3.2	语句并非按顺序执行	18
2.4	小结	19
第 3 章	测量性能	20
3.1	优化思想	21
3.1.1	必须测量性能	21
3.1.2	优化器是王牌猎人	21
3.1.3	90/10 规则	22
3.1.4	阿姆达尔定律	23
3.2	进行实验	24
3.2.1	记实验笔记	26
3.2.2	测量基准性能并设定目标	26
3.2.3	你只能改善你能够测量的	28
3.3	分析程序执行	28
3.4	测量长时间运行的代码	30
3.4.1	一点关于测量时间的知识	30
3.4.2	用计算机测量时间	35
3.4.3	克服测量障碍	41
3.4.4	创建 stopwatch 类	44
3.4.5	使用测试套件测量热点函数	48
3.5	评估代码开销来找出热点代码	48
3.5.1	评估独立的 C++ 语句的开销	49
3.5.2	评估循环的开销	49
3.6	其他找出热点代码的方法	51
3.7	小结	51
第 4 章	优化字符串的使用：案例研究	53
4.1	为什么字符串很麻烦	53
4.1.1	字符串是动态分配的	54

4.1.2	字符串就是值	54
4.1.3	字符串会进行大量复制	55
4.2	第一次尝试优化字符串	56
4.2.1	使用复合赋值操作避免临时字符串	57
4.2.2	通过预留存储空间减少内存的重新分配	57
4.2.3	消除对参数字符串的复制	58
4.2.4	使用迭代器消除指针解引	59
4.2.5	消除对返回的字符串的复制	59
4.2.6	用字符数组代替字符串	60
4.2.7	第一次优化总结	62
4.3	第二次尝试优化字符串	62
4.3.1	使用更好的算法	62
4.3.2	使用更好的编译器	64
4.3.3	使用更好的字符串库	64
4.3.4	使用更好的内存分配器	67
4.4	消除字符串转换	69
4.4.1	将 C 字符串转换为 <code>std::string</code>	69
4.4.2	不同字符集间的转换	70
4.5	小结	70
第 5 章	优化算法	71
5.1	算法的时间开销	72
5.1.1	最优情况、平均情况和最差情况的时间开销	74
5.1.2	摊销时间开销	74
5.1.3	其他开销	75
5.2	优化查找和排序的工具箱	75
5.3	高效查找算法	75
5.3.1	查找算法的时间开销	75
5.3.2	当 n 很小时，所有算法的时间开销都一样	76
5.4	高效排序算法	77
5.4.1	排序算法的时间开销	77
5.4.2	替换在最差情况下性能较差的排序算法	77
5.4.3	利用输入数据集的已知特性	78
5.5	优化模式	78
5.5.1	预计算	79
5.5.2	延迟计算	80
5.5.3	批量处理	80

5.5.4	缓存	80
5.5.5	特化	81
5.5.6	提高处理量	81
5.5.7	提示	81
5.5.8	优化期待路径	82
5.5.9	散列法	82
5.5.10	双重检查	82
5.6	小结	82
第 6 章	优化动态分配内存的变量	83
6.1	C++ 变量回顾	84
6.1.1	变量的存储期	84
6.1.2	变量的所有权	86
6.1.3	值对象与实体对象	86
6.2	C++ 动态变量 API 回顾	88
6.2.1	使用智能指针实现动态变量所有权的自动化	90
6.2.2	动态变量有运行时开销	92
6.3	减少动态变量的使用	92
6.3.1	静态地创建类实例	92
6.3.2	使用静态数据结构	93
6.3.3	使用 <code>std::make_shared</code> 替代 <code>new</code> 表达式	97
6.3.4	不要无谓地共享所有权	97
6.3.5	使用“主指针”拥有动态变量	98
6.4	减少动态变量的重新分配	99
6.4.1	预分配动态变量以防止重新分配	99
6.4.2	在循环外创建动态变量	99
6.5	移除无谓的复制	100
6.5.1	在类定义中禁止不希望发生的复制	101
6.5.2	移除函数调用上的复制	102
6.5.3	移除函数返回上的复制	103
6.5.4	免复制库	105
6.5.5	实现写时复制惯用法	106
6.5.6	切割数据结构	106
6.6	实现移动语义	107
6.6.1	非标准复制语义：痛苦的实现	107
6.6.2	<code>std::swap()</code> ：“穷人”的移动语义	108
6.6.3	共享所有权的实体	109

6.6.4	移动语义的移动部分	109
6.6.5	更新代码以使用移动语义	110
6.6.6	移动语义的微妙之处	111
6.7	扁平数据结构	113
6.8	小结	113
第 7 章	优化热点语句	115
7.1	从循环中移除代码	116
7.1.1	缓存循环结束条件值	117
7.1.2	使用更高效的循环语句	117
7.1.3	用递减替代递增	118
7.1.4	从循环中移除不变性代码	118
7.1.5	从循环中移除无谓的函数调用	119
7.1.6	从循环中移除隐含的函数调用	121
7.1.7	从循环中移除昂贵的、缓慢改变的调用	123
7.1.8	将循环放入函数以减少调用开销	123
7.1.9	不要频繁地进行操作	124
7.1.10	其他优化技巧	126
7.2	从函数中移除代码	126
7.2.1	函数调用的开销	126
7.2.2	简短地声明内联函数	129
7.2.3	在使用之前定义函数	129
7.2.4	移除未使用的多态性	130
7.2.5	放弃不使用的接口	130
7.2.6	用模板在编译时选择实现	133
7.2.7	避免使用 PIMPL 惯用法	134
7.2.8	移除对 DDL 的调用	135
7.2.9	使用静态成员函数取代成员函数	136
7.2.10	将虚析构函数移至基类中	136
7.3	优化表达式	137
7.3.1	简化表达式	137
7.3.2	将常量组合在一起	138
7.3.3	使用更高效的运算符	139
7.3.4	使用整数计算替代浮点型计算	139
7.3.5	双精度类型可能会比浮点型更快	140
7.3.6	用闭形式替代迭代计算	141
7.4	优化控制流程惯用法	142

7.4.1	用 switch 替代 if-else if-else	142
7.4.2	用虚函数替代 switch 或 if	143
7.4.3	使用无开销的异常处理	144
7.5	小结	145
第 8 章	使用更好的库	146
8.1	优化标准库的使用	146
8.1.1	C++ 标准库的哲学	147
8.1.2	使用 C++ 标准库的注意事项	147
8.2	优化现有库	149
8.2.1	改动越少越好	149
8.2.2	添加函数，不要改动功能	150
8.3	设计优化库	150
8.3.1	草率编码后悔多	150
8.3.2	在库的设计上，简约是一种美德	151
8.3.3	不要在库内分配内存	152
8.3.4	若有疑问，以速度为准	152
8.3.5	函数比框架更容易优化	152
8.3.6	扁平继承层次关系	153
8.3.7	扁平调用链	153
8.3.8	扁平分层设计	153
8.3.9	避免动态查找	154
8.3.10	留意“上帝函数”	155
8.4	小结	156
第 9 章	优化查找和排序	157
9.1	使用 std::map 和 std::string 的键值对表	158
9.2	改善查找性能的工具箱	159
9.2.1	进行一次基准测量	160
9.2.2	识别出待优化的活动	160
9.2.3	分解待优化的活动	160
9.2.4	修改或替换算法和数据结构	161
9.2.5	在自定义抽象上应用优化过程	162
9.3	优化 std::map 的查找	163
9.3.1	以固定长度的字符数组作为 std::map 的键	163
9.3.2	以 C 风格的字符串组作为键使用 std::map	164
9.3.3	当键就是值的时候，使用 map 的表亲 std::set	166

9.4	使用 <code><algorithm></code> 头文件优化算法	167
9.4.1	以序列容器作为被查找的键值对表	168
9.4.2	<code>std::find()</code> : 功能如其名, $O(n)$ 时间开销	169
9.4.3	<code>std::binary_search()</code> : 不返回值	169
9.4.4	使用 <code>std::equal_range()</code> 的二分查找	170
9.4.5	使用 <code>std::lower_bound()</code> 的二分查找	170
9.4.6	自己编写二分查找法	171
9.4.7	使用 <code>strcmp()</code> 自己编写二分查找法	172
9.5	优化键值对散列表中的查找	173
9.5.1	使用 <code>std::unordered_map</code> 进行散列	173
9.5.2	对固定长度字符数组的键进行散列	174
9.5.3	以空字符结尾的字符串为键进行散列	175
9.5.4	用自定义的散列表进行散列	176
9.6	斯特潘诺夫的抽象惩罚	177
9.7	使用 C++ 标准库优化排序	178
9.8	小结	179
第 10 章	优化数据结构	181
10.1	理解标准库容器	181
10.1.1	序列容器	182
10.1.2	关联容器	182
10.1.3	测试标准库容器	183
10.2	<code>std::vector</code> 与 <code>std::string</code>	187
10.2.1	重新分配的性能影响	188
10.2.2	<code>std::vector</code> 中的插入与删除	188
10.2.3	遍历 <code>std::vector</code>	190
10.2.4	对 <code>std::vector</code> 排序	191
10.2.5	查找 <code>std::vector</code>	191
10.3	<code>std::deque</code>	191
10.3.1	<code>std::deque</code> 中的插入和删除	193
10.3.2	遍历 <code>std::deque</code>	194
10.3.3	对 <code>std::deque</code> 的排序	194
10.3.4	查找 <code>std::deque</code>	194
10.4	<code>std::list</code>	194
10.4.1	<code>std::list</code> 中的插入和删除	196
10.4.2	遍历 <code>std::list</code> 中	197
10.4.3	对 <code>std::list</code> 排序	197

10.4.4	查找 <code>std::list</code>	197
10.5	<code>std::forward_list</code>	198
10.5.1	<code>std::forward_list</code> 中的插入和删除.....	199
10.5.2	遍历 <code>std::forward_list</code>	199
10.5.3	对 <code>std::forward_list</code> 排序.....	199
10.5.4	查找 <code>std::forward_list</code>	199
10.6	<code>std::map</code> 与 <code>std::multimap</code>	199
10.6.1	<code>std::map</code> 中的插入和删除.....	200
10.6.2	遍历 <code>std::map</code>	202
10.6.3	对 <code>std::map</code> 排序.....	202
10.6.4	查找 <code>std::map</code>	203
10.7	<code>std::set</code> 与 <code>std::multiset</code>	203
10.8	<code>std::unordered_map</code> 与 <code>std::unordered_multimap</code>	204
10.8.1	<code>std::unordered_map</code> 中的插入与删除.....	206
10.8.2	遍历 <code>std::unordered_map</code>	207
10.8.3	查找 <code>std::unordered_map</code>	207
10.9	其他数据结构.....	208
10.10	小结.....	209
第 11 章	优化 I/O.....	210
11.1	读取文件的秘诀.....	210
11.1.1	创建一个吝啬的函数签名.....	211
11.1.2	缩短调用链.....	213
11.1.3	减少重新分配.....	213
11.1.4	更大的吞吐量——使用更大的输入缓冲区.....	215
11.1.5	更大的吞吐量——一次读取一行.....	216
11.1.6	再次缩短函数调用链.....	217
11.1.7	无用的技巧.....	218
11.2	写文件.....	219
11.3	从 <code>std::cin</code> 读取和向 <code>std::cout</code> 中写入.....	220
11.4	小结.....	220
第 12 章	优化并发.....	221
12.1	复习并发.....	222
12.1.1	并发概述.....	222
12.1.2	交叉执行.....	226
12.1.3	顺序一致性.....	226