Linux 多线程服务端编程 使用 muduo C++ 网络库

陈硕 (giantchen@gmail.com) 最后更新 2012-09-30

封面文案

示范在多核时代采用现代 C++ 编写 多线程 TCP 网络服务器的正规做法

内容简介

本书主要讲述采用现代 C++ 在 x86-64 Linux 上编写多线程 TCP 网络服务程序的主流常规技术,重点讲解一种适应性较强的多线程服务器的编程模型,即 one loop per thread。这是在 Linux 下以 native 语言编写用户态高性能网络程序最成熟的模式,掌握之后可顺利地开发各类常见的服务端网络应用程序。本书以 muduo 网络库为例,讲解这种编程模型的使用方法及注意事项。

本书的宗旨是贵精不贵多。掌握两种基本的同步原语就可以满足各种多线程同步的功能需求,还能写出更易用的同步设施。掌握一种进程间通信方式和一种多线程网络编程模型就足以应对日常开发任务,编写运行于公司内网环境的分布式服务系统。

作者简介

陈硕,北京师范大学硕士,擅长 C++ 多线程网络编程和实时分布式系统架构。曾在摩根士丹利 IT 部门工作 5 年,从事实时外汇交易系统开发。现在在美国加州硅谷某互联网大公司工作,从事大规模分布式系统的可靠性工程。编写了开源 C++ 网络库 muduo,参与翻译了《代码大全(第 2 版)》和《C++ 编程规范(繁体版)》,整理了《C++ Primer(第 4 版)(评注版)》,并曾多次在各地技术大会演讲。

封底文案

看完了 W. Richard Stevens 的传世经典《UNIX 网络编程》,能照着例子用 Sockets API 编写 echo 服务,却仍然对稍微复杂一点的网络编程任务感到无从下手?学习网络编程有哪些好的练手项目?书中示例代码把业务逻辑和 Sockets 调用混在一起,似乎不利于将来扩展?网络编程中遇到一些具体问题该怎么办?例如:

- 程序在本机测试正常,放到网络上运行就 经常出现数据收不全的情况?
- TCP 协议真的有所谓的"粘包问题"吗? 该如何设计消息帧的协议?又该如何编码 实现分包才不会掉到陷阱里?
- 带外数据 (OOB)、信号驱动 IO 这些高级 特性到底有没有用?
- 网络消息格式该怎么设计? 发送 C struct 会有对齐方面的问题吗? 对方不用 C/C++ 怎么通信? 将来服务端软件升级,需要在 消息中增加一个字段,现有的客户端就必须强制升级?
- 要处理成千上万的并发连接,似乎《UNIX 网络编程》介绍的传统 fork()模型应付不过来,该用哪种并发模型呢?试试select(2)/pol1(2)/epol1(4)这种 IO 复用模型吧,又感觉非阻塞 IO 陷阱重重,怎么程序的 CPU 使用率一直是 100%?
 - 要不改用现成的 libevent 网络库吧,怎么 查询一下数据库就把其他连接上的请求给 耽误了?
- 再用个线程池吧。万一发回响应的时候对方已经断开连接了怎么办?会不会串话?

读过《UNIX 环境高级编程》,想用多线程来发挥多核 CPU 的性能潜力,但对程序该用哪种多线程模型感到一头雾水?有没有值得推荐的适用面广的多线程 IO 模型?互斥器、条件变量、读写锁、信号量这些底层同步原语哪些该用哪些不该用?有没有更高级的同步设施能简化开发? 《UNIX 网络编程(第2卷)》介绍的那些琳琅满目的进程间通信(IPC)机制到底用哪个才能兼顾开发效率与可伸缩性?

网络编程和多线程编程的基础打得差不多,开始实际做项目了,更多问题扑面而来:

- 网上听人说服务端开发要做到 7 × 24 运行,为了防止内存碎片连动态内存分配都不能用,那岂不是连 C++ STL 也一并禁用了? 硬件的可靠性高到值得去这么做吗?
- 传闻服务端开发主要通过日志来查错,那 么日志里该写些什么?日志是写给谁看的? 怎样写日志才不会影响性能?
- 分布式系统跟单机多进程到底有什么本质 区别?心跳协议为什么是必需的,该如何 实现?
 - C++ 的大型工程该如何管理?库的接口如何设计才能保证升级的时候不破坏二进制兼容性?有没有更适合大规模分布式系统的部署方案?

这本《Linux 多线程服务端编程:使用 muduo C++ 网络库》中,作者凭借多年的工程实践经验试图解答以上疑问。当然,内容还远不止这些……

前言

本书主要讲述采用现代 C++ 在 x86-64 Linux 上编写多线程 TCP 网络服务程序的主流常规技术,这也是我对过去 5 年编写生产环境下的多线程服务端程序的经验总结。本书重点讲解多线程网络服务器的一种 IO 模型,即 one loop per thread。这是一种适应性较强的模型,也是 Linux 下以 native 语言编写用户态高性能网络程序最成熟的模式,掌握之后可顺利地开发各类常见的服务端网络应用程序。本书以 muduo 网络库为例,讲解这种编程模型的使用方法及注意事项。

muduo 是一个基于非阻塞 IO 和事件驱动的现代 C++ 网络库,原生支持 one loop per thread 这种 IO 模型。muduo 适合开发 Linux 下的面向业务的多线程服务端网络应用程序,其中"面向业务的网络编程"的定义见附录 A。"现代 C++"指的不是 C++11 新标准,而是 2005 年 TR1 发布之后的 C++ 语言和库。与传统 C++ 相比,现代 C++ 的变化主要有两方面:资源管理(见第 1 章)与事件回调(见第 449 页)。

本书不是多线程编程教程,也不是网络编程教程,更不是 C++ 教程。读者应该已经大致读过《UNIX 环境高级编程》、《UNIX 网络编程》、《C++ Primer》或与之内容相近的书籍。本书不谈 C++11,因为目前(2012年)主流的 Linux 服务端发行版的g++ 版本都还停留在 4.4, C++11 进入实用尚需一段时日。

本书适用的硬件环境是主流 x86-64 服务器,多路多核 CPU、几十 GB 内存、千 兆以太网互联。除了第5章讲诊断日志之外,本书不涉及文件 IO。

本书分为四大部分,第1部分 "C++多线程系统编程"考察多线程下的对象生命期管理、线程同步方法、多线程与 C++的结合、高效的多线程日志等。第2部分"muduo 网络库"介绍使用现成的非阻塞网络库编写网络应用程序的方法,以及muduo 的设计与实现。第3部分"工程实践经验谈"介绍分布式系统的工程化开发方法和 C++在工程实践中的功能特性取舍。第4部分"附录"分享网络编程和 C++语言的学习经验。

本书的宗旨是贵精不贵多。掌握两种基本的同步原语就可以满足各种多线程同步的功能需求,还能写出更易用的同步设施。掌握一种进程间通信方式和一种多线程网络编程模型就足以应对日常开发任务,编写运行于公司内网环境的分布式服务系统。(本书不涉及分布式存储系统,也不涉及 UDP。)

iv

术语与排版范例

本书大量使用英文术语,甚至有少量英文引文。设计模式的名字一律用英文,例如 Observer、Reactor、Singleton。在中文术语不够突出时,也会使用英文,例如 class、heap、event loop、STL algorithm 等。注意几个中文 C++ 术语: 对象 **实体** (instance)、函数重载**决议** (resolution)、模板**具现化** (instantiation)、**覆写** (override) 虚函数、**提领** (dereference) 指针。本书中的英语可数名词一般不用复数形式,例如两个 class,6个 syscall;但有时会用 (s) 强调中文名词是复数。fd 是文件描述符(file descriptor)的缩写。 "CPU 数目"一般指的是核(core)的数目。容量单位 kB、MB、GB 表示的字节数分别为 10^3 、 10^6 、 10^9 ,在特别强调准确数值时,会分别用 KiB、MiB、GiB 表示 2^{10} 、 2^{20} 、 2^{30} 字节。用诸如 §11.5 表示本书第 11.5 节,C42 表示上下文中出现的第 42 行代码。[JCP]、[CC2e] 等是参考文献,见书末清单。

一般术语用普通罗马字体,如 mutex、socket; C++ 关键字用无衬线字体,如 class、this、mutable; 函数名和 class 名用等宽字体,如 fork(2)、muduo::EventLoop, 其中 fork(2) 表示系统函数 fork() 的文档位于 manpage 第 2 节,可以通过 man 2 fork 命令查看。如果函数名或类名过长,可能会折行,行末有连字号 "-",如 EventLoop-ThreadPool。文件路径和 URL 采用窄字体,例如 muduo/base/Date.h、http://chenshuo.com。用中文楷体表示引述别人的话。

代码

本书的示例代码以开源项目的形式发布在 GitHub 上,地址是 http://github.com/chenshuo/recipes/ 和 http://github.com/chenshuo/muduo/。本书配套页面提供全部源代码打包下载,正文中出现的类似 recipes/thread 的路径是压缩包内的相对路径,读者不难找到其对应的 GitHub URL。本书引用代码的形式如下,左侧数字是文件的行号,右侧的"muduo/base/Types.h"是文件路径 1。例如下面这几行代码是 muduo::string 的 typedef。

```
muduo/base/Types.h

namespace muduo

for {

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h

muduo/base/Types.h
```

¹ 在第 6、7 两章的 muduo 示例代码中,路径 muduo/examples/XXX 会简写为 examples/XXX。此外,第 8 章会把 recipes/reactor/XXX 简写为 reactor/XXX。

前言

本书假定读者熟悉 diff -u 命令的输出格式,用于表示代码的改动。

本书正文中出现的代码有时为了照顾排版而略有改写,例如改变缩进规则,去掉单行条件语句前后的花括号等。就编程风格而论、应以电子版代码为准。

联系方式

邮箱: giantchen@gmail.com

主页: http://chenshuo.com/book (正文和脚注中出现的 URL 可从这里找到。)

微博: http://weibo.com/giantchen 博客: http://blog.csdn.net/Solstice

代码: http://github.com/chenshuo

陈硕 中国·香港

内容一览

第	1部分	C++ 多线程系统编程	1
	第1章	线程安全的对象生命期管理	3
	第2章	线程同步精要	31
	第3章	多线程服务器的适用场合与常用编程模型	59
	第4章	C++ 多线程系统编程精要	83
	第5章	高效的多线程日志	107
第	2 部分	muduo 网络库	123
	第6章	muduo 网络库简介	125
	第7章	muduo 编程示例	177
	第8章	muduo 网络库设计与实现	277
第	3 部分	工程实践经验谈 3	337
	第9章	分布式系统工程实践	339
	第 10 章	C++ 编译链接模型精要	391
	第 11 章	反思 C++ 面向对象与虚函数	429
	第 12 章	C++ 经验谈	501
第	4 部分	附录 5	559
	附录 A	谈一谈网络编程学习经验	561
	附录 B	从 《C++ Primer (第 4 版)》 入手学习 C++	579
	附录C	关于 Boost 的看法	591
	附录 D	关于 TCP 并发连接的几个思考题与试验	593
	参考文献		599

第1部	分 C++ 多线程系统编程	1
第1章	线程安全的对象生命期管理	3
1.1	当析构函数遇到多线程	3
	1.1.1 线程安全的定义	4
	1.1.2 MutexLock与 MutexLockGuard	4
	1.1.3 一个线程安全的 Counter 示例	4
1.2	对象的创建很简单	5
1.3	销毁太难	7
	1.3.1 mutex 不是办法	7
	1.3.2 作为数据成员的 mutex 不能保护析构	8
1.4	线程安全的 Observer 有多难	8
1.5	原始指针有何不妥	11
1.6	神器 shared_ptr/weak_ptr	13
1.7	插曲:系统地避免各种指针错误	14
1.8	应用到 Observer 上	16
1.9	再论 shared_ptr 的线程安全	17
1.10	shared_ptr 技术与陷阱	19
1.11	对象池	21
	$1.11.1 {\tt enable_shared_from_this} \; . \; . \; . \; . \; . \; . \; . \; . \; . \; $	23
	1.11.2 弱回调	24
1.12	替代方案	26
1.13	心得与小结	26
1.14	Observer 之谬	28
第2章	线程同步精要	31
2.1	互斥器 (mutex)	32

	2.1.1 只使用非递归的 mutex	33
	2.1.2 死锁	35
2.2	条件变量(condition variable)	40
2.3	不要用读写锁和信号量	43
2.4	封装 MutexLock、MutexLockGuard、Condition	44
2.5	线程安全的 Singleton 实现	48
2.6	sleep(3) 不是同步原语	50
2.7	归纳与总结	51
2.8	借 shared_ptr 实现 copy-on-write	52
第 3 章	多线程服务器的适用场合与常用编程模型	59
3.1	进程与线程	59
3.2	单线程服务器的常用编程模型	61
3.3	多线程服务器的常用编程模型	62
	3.3.1 one loop per thread	62
	3.3.2 线程池	63
	3.3.3 推荐模式	64
3.4	进程间通信只用 TCP	65
3.5	多线程服务器的适用场合	67
	3.5.1 必须用单线程的场合	69
	3.5.2 单线程程序的优缺点	70
	3.5.3 适用多线程程序的场景	71
3.6	"多线程服务器的适用场合"例释与答疑	74
第4章	C++ 多线程系统编程精要	83
4.1	基本线程原语的选用	84
4.2	C/C++ 系统库的线程安全性	85
4.3	Linux 上的线程标识	89
4.4	线程的创建与销毁的守则	91
	$4.4.1$ pthread_cancel 与 C++	94
	4.4.2 exit(3) 在 C++ 中不是线程安全的	94
4.5	善用thread 关键字	96
4.6	多线程与 IO	98

4.7	用 RAII 包装文件描述符	99
4.8	RAII与 fork()1	.01
4.9	多线程与 fork()	02
4.10	多线程与 signal	03
4.11	Linux 新增系统调用的启示	05
第5章	高效的多线程日志 1	07
5.1	功能需求	09
5.2	性能需求	12
5.3	多线程异步日志	14
5.4	其他方案	20
佐 。 並		
弗 2 部	分 muduo 网络库 12	23
第6章		25
6.1	由来	
6.2	安装	27
6.3	目录结构	29
	6.3.1 代码结构	31
	6.3.2 例子	34
	6.3.3 线程模型	35
6.4	使用教程1	36
	6.4.1 TCP 网络编程本质论	36
	6.4.2 echo 服务的实现	38
	6.4.3 七步实现 finger 服务	40
6.5	性能评测 1	44
	6.5.1 muduo与 Boost.Asio、libevent2的吞吐量对比1	45
	6.5.2 击鼓传花:对比 muduo 与 libevent2 的事件处理效率 1	48
	6.5.3 muduo 与 Nginx 的吞吐量对比	53
	6.5.4 muduo 与 ZeroMQ 的延迟对比	56
6.6	详解 muduo 多线程模型	.57
	6.6.1 数独求解服务器	.57
	6.6.2 常见的并发网络服务程序设计方案 1	60

第7章	mudu	ıo 编程示例	177		
7.1	五个简	筍单 TCP 示例 	178		
7.2	文件传输				
7.3	Boost.	.Asio 的聊天服务器	194		
	7.3.1	TCP 分包	194		
	7.3.2	消息格式	195		
	7.3.3	编解码器 LengthHeaderCodec	197		
	7.3.4	服务端的实现	198		
	7.3.5	客户端的实现	200		
7.4	mudu	ıo Buffer 类的设计与使用	204		
	7.4.1	muduo 的 IO 模型	204		
	7.4.2	为什么 non-blocking 网络编程中应用层 buffer 是必需的	205		
	7.4.3	Buffer 的功能需求	207		
	7.4.4	Buffer 的数据结构	209		
	7.4.5	Buffer 的操作	211		
	7.4.6	其他设计方案	217		
	7.4.7	性能是不是问题	218		
7.5	一种自	自动反射消息类型的 Google Protobuf 网络传输方案	220		
	7.5.1	网络编程中使用 Protobuf 的两个先决条件	220		
	7.5.2	根据 type name 反射自动创建 Message 对象	221		
	7.5.3	Protobuf 传输格式	226		
7.6	在 mu	ıduo 中实现 Protobuf 编解码器与消息分发器	228		
	7.6.1	什么是编解码器 (codec)	229		
	7.6.2	实现 ProtobufCodec	232		
	7.6.3	消息分发器 (dispatcher) 有什么用	232		
	7.6.4	ProtobufCodec 与 ProtobufDispatcher 的综合运用	233		
	7.6.5	ProtobufDispatcher 的两种实现	234		
	7.6.6	ProtobufCodec 和 ProtobufDispatcher 有何意义	236		
7.7	限制朋	B务器的最大并发连接数	237		
	7.7.1	为什么要限制并发连接数	237		
	7.7.2	在 muduo 中限制并发连接数	238		

7.8	定时器		
7.0	7.8.1	程序中的时间	
	7.8.2	Linux 时间函数	
	7.8.3	muduo 的定时器接口	
	7.8.4	Boost.Asio Timer 示例	
	7.8.5	Java Netty 示例	
7.9		java Netty 小例	
7.10		ing wheel 踢掉空闲连接	
	7.10.1	timing wheel 原理	
	7.10.2	代码实现与改进	
7.11		1消息广播服务	
7.12		专换"连接服务器及其自动化测试	
7.13	socks4	a 代理服务器	
	7.13.1	TCP 中继器	
	7.13.2	socks4a 代理服务器	
	7.13.3	N:1与1:N连接转发	
7.14	短址服	/务	
7.15	与其他	」库集成	
	7.15.1	UDNS	
	7.15.2	c-ares DNS	
	7.15.3	curl	
	7.15.4	更多	
第8章	mudu	o 网络库设计与实现 277	
8.0		3不做的 EventLoop	
8.1			
0.1	8.1.1	Channel class	
	8.1.2	Poller class	
	8.1.3	EventLoop 的改动	
8.2		Dueue 定时器	
0.4	8.2.1		
		TimerQueue class	
	8.2.2	EventLoop 的改动	

xii

0.3	Function with an O 应数	202	
8.3	EventLoop::runInLoop() 函数		
	8.3.1 提高 TimerQueue 的线程安全性		
	8.3.2 EventLoopThread class		
8.4	实现 TCP 网络库		
8.5	TcpServer 接受新连接		
	8.5.1 TcpServer class		
	8.5.2 TcpConnection class		
8.6	TcpConnection 断开连接		
8.7	Buffer 读取数据	313	
	8.7.1 TcpConnection 使用 Buffer 作为输入缓冲	314	
	8.7.2 Buffer::readFd()	315	
8.8	TcpConnection 发送数据	316	
8.9	完善 TcpConnection	320	
	8.9.1 SIGPIPE	321	
	8.9.2 TCP No Delay 和 TCP keepalive	321	
	$8.9.3$ WriteCompleteCallback 和 HighWaterMarkCallback $\dots \dots$	322	
8.10	多线程 TcpServer	324	
8.11	Connector		
8.12	TcpClient	332	
8.13	epoll	333	
8.14	测试程序一览	336	
<i>66 - 4</i> 0	7. () —— 45 —— 45 —— 15 —— 15 —— 16 —— 17 —— 18		
第3部	3分 工程实践经验谈 3	37	
第9章	分布式系统工程实践	339	
9.1	我们在技术浪潮中的位置	341	
	9.1.1 分布式系统的本质困难	343	
	9.1.2 分布式系统是个险恶的问题	344	
9.2	分布式系统的可靠性浅说	349	
	9.2.1 分布式系统的软件不要求 7 × 24 可靠	352	
	9.2.2 "能随时重启进程"作为程序设计目标	354	
9.3	分布式系统中心跳协议的设计	356	

9.4	公布式	、 系统中的进程标识	260
9.4			
	9.4.1	错误做法	
	9.4.2	正确做法	
	9.4.3	TCP 协议的启示	
9.5		5于维护的分布式程序	
9.6	为系统	演化做准备	
	9.6.1	可扩展的消息格式	368
	9.6.2	反面教材: ICE 的消息打包格式	369
9.7	分布式	程序的自动化回归测试	370
	9.7.1	单元测试的能与不能	370
	9.7.2	分布式系统测试的要点	373
	9.7.3	分布式系统的抽象观点	374
	9.7.4	一种自动化的回归测试方案	375
	9.7.5	其他用处	379
9.8	分布式	系统部署、监控与进程管理的几重境界	380
	9.8.1	境界 1: 全手工操作	382
	9.8.2	境界 2: 使用零散的自动化脚本和第三方组件	383
	9.8.3	境界 3: 自制机群管理系统, 集中化配置	386
	9.8.4	境界 4: 机群管理与 naming service 结合	389
笋 10 音	Стт	编译链接模型精要	391
		的编译模型及其成因	
10.1	10.1.1	为什么 C 语言需要预处理	
		C 语言的编译模型	
10.2		り編译模型	
10.2		「編件侯至	
	10.2.1		
10.0		前向声明	
10.3		维接(linking)	
		函数重载	
		inline 函数	
		模板	
	10.3.4	虚函数	414

xiv 目录

10.4	工程项	[目中头文件的使用规则415
	10.4.1	头文件的害处
	10.4.2	头文件的使用规则417
10.5	工程项	[目中库文件的组织原则 418
	10.5.1	动态库是有害的
	10.5.2	静态库也好不到哪儿去
	10.5.3	源码编译是王道
第 11 章	反思	C++ 面向对象与虚函数 429
11.1	朴实的	I C++ 设计
11.2	程序库	的二进制兼容性
	11.2.1	什么是二进制兼容性
	11.2.2	有哪些情况会破坏库的 ABI
	11.2.3	哪些做法多半是安全的
	11.2.4	反面教材: COM 435
	11.2.5	解决办法
11.3	避免使	:用虚函数作为库的接口
	11.3.1	C++ 程序库的作者的生存环境
	11.3.2	虚函数作为库的接口的两大用途
	11.3.3	虚函数作为接口的弊端
	11.3.4	假如 Linux 系统调用以 COM 接口方式实现 442
	11.3.5	Java 是如何应对的
11.4	动态库	接口的推荐做法
11.5	以 boo	st::function 和 boost::bind 取代虚函数 447
	11.5.1	基本用途
	11.5.2	对程序库的影响
	11.5.3	对面向对象程序设计的影响453
11.6	iostrea	ım 的用途与局限
	11.6.1	stdio 格式化输入输出的缺点
	11.6.2	iostream 的设计初衷
	11.6.3	iostream 与标准库其他组件的交互 463

	11.6.4	iostream 在使用方面的缺点
	11.6.5	iostream 在设计方面的缺点
	11.6.6	一个 300 行的 memory buffer output stream 476
	11.6.7	现实的 C++ 程序如何做文件 IO
11.7	值语义	.与数据抽象
	11.7.1	什么是值语义482
	11.7.2	值语义与生命期
	11.7.3	值语义与标准库
	11.7.4	值语义与 C++ 语言
	11.7.5	什么是数据抽象
	11.7.6	数据抽象所需的语言设施493
	11.7.7	数据抽象的例子
第 12 章	C++ :	经验谈 501
12.1	用异或	来交换变量是错误的501
	12.1.1	编译器会分别生成什么代码503
	12.1.2	为什么短的代码不一定快505
12.2	不要重	:载全局::operator new()507
	12.2.1	内存管理的基本要求507
	12.2.2	重载::operator new() 的理由508
	12.2.3	::operator new() 的两种重载方式 508
	12.2.4	现实的开发环境
	12.2.5	重载::operator new()的困境 510
	12.2.6	解决办法: 替换 malloc()512
	12.2.7	为单独的 class 重载::operator new() 有问题吗 513
	12.2.8	有必要自行定制内存分配器吗 513
12.3	带符号	整数的除法与余数
	12.3.1	语言标准怎么说
	12.3.2	C/C++ 编译器的表现 516
	12.3.3	其他语言的规定
	12.3.4	脚本语言解释器代码517
	12.3.5	硬件实现

xvi

12.4	在单元	测试中 mock 系统调用	522
	12.4.1	系统函数的依赖注入	522
	12.4.2	链接期垫片 (link seam)	524
12.5	慎用匿	名 namespace	526
	12.5.1	C语言的 static 关键字的两种用法	526
	12.5.2	C++ 语言的 static 关键字的四种用法	526
	12.5.3	匿名 namespace 的不利之处	527
	12.5.4	替代办法	529
12.6	采用有	利于版本管理的代码格式	529
	12.6.1	对 diff 友好的代码格式	530
	12.6.2	对 grep 友好的代码风格	537
	12.6.3	一切为了效率	538
12.7	再探 s	td::string	539
	12.7.1	直接拷贝 (eager copy)	540
	12.7.2	写时复制(copy-on-write)	542
	12.7.3	短字符串优化 (SSO)	543
12.8	用 STL	algorithm 轻松解决几道算法面试题	546
	12.8.1	用 next_permutation() 生成排列与组合	546
	12.8.2	用 unique() 去除连续重复空白	548
	12.8.3	用 {make,push,pop}_heap() 实现多路归并	549
	12.8.4	用 partition() 实现"重排数组,让奇数位于偶数前面"	553
	12.8.5	用 lower_bound() 查找 IP 地址所属的城市	554
<u> </u>	/\ m	4 3 .	 0
第4部	分削	引求	559
附录 A	谈一谈	网络编程学习经验	561
附录 B	从《C-	++ Primer(第 4 版)》入手学习 C++	579
附录 C	关于 B	oost 的看法	591
附录 D	关于 T	CP 并发连接的几个思考题与试验	593
参考文献	参考文献 599		