

Linux 多线程 服务端编程

使用 muduo C++ 网络库

陈硕
著

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



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

Linux 多线程 服务端编程

使用 muduo C++ 网络库

陈硕 著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

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。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

Linux 多线程服务端编程:使用 muduo C++网络库 / 陈硕著. —北京:电子工业出版社, 2013.1
ISBN 978-7-121-19282-1

I. ①L… II. ①陈… III. ①Linux 操作系统—程序设计 IV. ①TP316.89

中国版本图书馆 CIP 数据核字(2012)第 304000 号

策划编辑:张春雨

责任编辑:李云静

印 刷:北京丰源印刷厂

装 订:三河市鹏成印业有限公司

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×980 1/16 印张:38.5 字数:801 千字

印 次:2013 年 3 月第 2 次印刷

印 数:3001~6000 册 定价:89.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zits@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

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周期管理、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。）

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，L42 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
15 namespace muduo
16 {
17
18 #ifdef MUDUO_STD_STRING
19 using std::string;
20 #else // !MUDUO_STD_STRING
21 typedef __gnu_cxx::__ssso_string string;
22 #endif
----- 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。

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	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	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

目录

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 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()	101
4.9	多线程与 fork()	102
4.10	多线程与 signal	103
4.11	Linux 新增系统调用的启示	105
第 5 章	高效的多线程日志	107
5.1	功能需求	109
5.2	性能需求	112
5.3	多线程异步日志	114
5.4	其他方案	120
第 2 部分	muduo 网络库	123
第 6 章	muduo 网络库简介	125
6.1	由来	125
6.2	安装	127
6.3	目录结构	129
6.3.1	代码结构	131
6.3.2	例子	134
6.3.3	线程模型	135
6.4	使用教程	136
6.4.1	TCP 网络编程本质论	136
6.4.2	echo 服务的实现	138
6.4.3	七步实现 finger 服务	140
6.5	性能评测	144
6.5.1	muduo 与 Boost.Asio、libevent2 的吞吐量对比	145
6.5.2	击鼓传花：对比 muduo 与 libevent2 的事件处理效率	148
6.5.3	muduo 与 Nginx 的吞吐量对比	153
6.5.4	muduo 与 ZeroMQ 的延迟对比	156
6.6	详解 muduo 多线程模型	157
6.6.1	数独求解服务器	157
6.6.2	常见的并发网络服务程序设计方案	160

第 7 章 muduo 编程示例	177
7.1 五个简单 TCP 示例	178
7.2 文件传输	185
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 LengthHeaderCode	197
7.3.4 服务端的实现	198
7.3.5 客户端的实现	200
7.4 muduo 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 muduo 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 限制服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	240
7.8.1	程序中的时间	240
7.8.2	Linux 时间函数	241
7.8.3	muduo 的定时器接口	242
7.8.4	Boost.Asio Timer 示例	243
7.8.5	Java Netty 示例	245
7.9	测量两台机器的网络延迟和时间差	248
7.10	用 timing wheel 踢掉空闲连接	250
7.10.1	timing wheel 原理	251
7.10.2	代码实现与改进	254
7.11	简单的消息广播服务	257
7.12	“串并转换”连接服务器及其自动化测试	260
7.13	socks4a 代理服务器	264
7.13.1	TCP 中继器	264
7.13.2	socks4a 代理服务器	267
7.13.3	$N:1$ 与 $1:N$ 连接转发	267
7.14	短址服务	267
7.15	与其他库集成	268
7.15.1	UDNS	270
7.15.2	c-ares DNS	272
7.15.3	curl	273
7.15.4	更多	275
第 8 章	muduo 网络库设计与实现	277
8.0	什么都不做的 EventLoop	277
8.1	Reactor 的关键结构	280
8.1.1	Channel class	280
8.1.2	Poller class	283
8.1.3	EventLoop 的改动	287
8.2	TimerQueue 定时器	290
8.2.1	TimerQueue class	290
8.2.2	EventLoop 的改动	292

8.3	EventLoop::runInLoop() 函数	293
8.3.1	提高 TimerQueue 的线程安全性	296
8.3.2	EventLoopThread class	297
8.4	实现 TCP 网络库	299
8.5	TcpServer 接受新连接	303
8.5.1	TcpServer class	304
8.5.2	TcpConnection class	305
8.6	TcpConnection 断开连接	308
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	322
8.10	多线程 TcpServer	324
8.11	Connector	327
8.12	TcpClient	332
8.13	epoll	333
8.14	测试程序一览	336
第 3 部分 工程实践经验谈		337
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	分布式系统中的进程标识	360
9.4.1	错误做法	361
9.4.2	正确做法	362
9.4.3	TCP 协议的启示	363
9.5	构建易于维护的分布式程序	364
9.6	为系统演化做准备	367
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
第 10 章	C++ 编译链接模型精要	391
10.1	C 语言的编译模型及其成因	394
10.1.1	为什么 C 语言需要预处理	395
10.1.2	C 语言的编译模型	398
10.2	C++ 的编译模型	399
10.2.1	单遍编译	399
10.2.2	前向声明	402
10.3	C++ 链接 (linking)	404
10.3.1	函数重载	406
10.3.2	inline 函数	407
10.3.3	模板	409
10.3.4	虚函数	414

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	416
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	423
10.5.2	静态库也好不到哪儿去	424
10.5.3	源码编译是王道	428
第 11 章	反思 C++ 面向对象与虚函数	429
11.1	朴实的 C++ 设计	429
11.2	程序库的二进制兼容性	431
11.2.1	什么是二进制兼容性	432
11.2.2	有哪些情况会破坏库的 ABI	433
11.2.3	哪些做法多半是安全的	435
11.2.4	反面教材: COM	435
11.2.5	解决办法	436
11.3	避免使用虚函数作为库的接口	436
11.3.1	C++ 程序库的作者的生存环境	437
11.3.2	虚函数作为库的接口的两大用途	438
11.3.3	虚函数作为接口的弊端	439
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	443
11.4	动态库接口的推荐做法	443
11.5	以 boost::function 和 boost::bind 取代虚函数	447
11.5.1	基本用途	450
11.5.2	对程序库的影响	451
11.5.3	对面向对象程序设计的影响	453
11.6	iostream 的用途与局限	457
11.6.1	stdio 格式化输入输出的缺点	457
11.6.2	iostream 的设计初衷	461
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	464
11.6.5	iostream 在设计方面的缺点	468
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	480
11.7	值语义与数据抽象	482
11.7.1	什么是值语义	482
11.7.2	值语义与生命期	483
11.7.3	值语义与标准库	488
11.7.4	值语义与 C++ 语言	488
11.7.5	什么是数据抽象	490
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	495
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	509
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	514
12.3.1	语言标准怎么说	515
12.3.2	C/C++ 编译器的表现	516
12.3.3	其他语言的规定	516
12.3.4	脚本语言解释器代码	517
12.3.5	硬件实现	521

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 std::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
第 4 部分	附录	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