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。

电子工业出版社

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)/poll(2)/epoll(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。)

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，$\mathcal{L}$42 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
{

#ifdef MUDUO_STD_STRING
using std::string;
#else  // !MUDUO_STD_STRING
typedef __gnu_cxx::__sso_string string;
#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

目录