



下载APP



# 罗剑锋说C语言 | 为什么NGINX是C编程的经典范本？

2022-01-10 Chrono

《深入C语言和程序运行原理》

课程介绍 &gt;

**讲述：Chrono**

时长 15:13 大小 13.95M



你好，我是罗剑锋，不过我更喜欢你称呼我的网名：Chrono。

很高兴受邀来到这个专栏做一期分享。先来简单地做个自我介绍：我是一个有差不多 20 年工作经验的编程“老兵”，出版过两本书《C++ 11/14 高级编程》《Boost 程序库完全开发指南》，也在极客时间上写过专栏《透视 HTTP 协议》和《C++ 实战笔记》。

领资料

可以看到，我主要的研究方向是 C++。不过，因为 C++ 和 C “一脉相承”，而且我在大学的时候初学的编程语言就是 C，所以，C 语言对于我来说，也是一门有着深厚感情的语言。



最近这几年，我的研发重心逐渐转移到了 Linux 系统编程和 NGINX 深度定制上，而这两者都是用 C 语言实现的，所以 C 语言又重新占据了我不少的工作时间。相比 C++ 来说，

C 更加简单纯粹，没有那么多复杂深奥的语法规则，写起来也就更加轻松自如一些。

有很多人在学习 C 语言的时候都有一种感慨：C 语言的语法、语义、库函数都很精悍干练，把这些东西全部弄懂并不需要花费太多力气。但想要再进一步，用它写出高效、实用的程序，这其中就有一道很大的“鸿沟”需要跨越，经常遇到的情况是面对一个问题不知道如何下手。

**我认为，要学好 C 编程，掌握基本的语言特性只是迈出了第一步。**因为比起其他编程语言，C 更接近系统底层，所以还需要了解计算机原理、操作系统等知识，并且把它们在 C 语言里“打通”“融成一体”，这样才算是真正学会了 C 语言编程。

那么，学习 C 语言编程有没有什么好方法呢？

除了阅读经典著作和实际开发编码之外，我觉得还有一种很有效的方式：**钻研优秀的开源项目**。通过学习那些经过“千锤百炼”的一行行源码，践行鲁迅先生的“拿来主义”，把开源项目的精华部分转变为自己的知识储备。也就是那句老话：“他山之石，可以攻玉。”

所以，今天借着这个机会，我就来聊聊我个人认为的 C 语言编程的经典范本，NGINX，并向你展示用 NGINX 学习 C 语言编程的正确打开方式。

## 什么是 NGINX？

正式讲方法之前，我们需要先来了解一下什么是 NGINX。有后端开发、网络应用背景的同学应该都知道 NGINX，它是一个高性能、高稳定、功能齐备的 Web 服务器。

NGINX 具有运行效率高、资源占用低、支持海量并发、运维友好等特点，适应了互联网“爆炸式”发展的大潮。所以，自从 2004 年公开发布 0.1.0 版以来，NGINX 的市场占有率就一路攀升，当然，相应的就是竞争对手 Apache httpd、Microsoft IIS 份额的下跌。

到今年，也就是 2021 年的 5 月份，W3Techs 网站的统计数据表明，NGINX 不仅在前一千、前一百名，而且是在所有的网站中，使用率都超过了传统的 Apache，总计的站点数量超过了 4 亿个。这也就意味着，NGINX 取代了已经存在 27 年的 Apache httpd，正式成为全球最受欢迎的 Web 服务器。

然而，NGINX 的用途还远远不止于 Web 服务。由于它核心的框架机制非常灵活、易于扩展，在多年的发展过程中，官方团队和广大志愿者又为它添加了反向代理、负载均衡等能力。并且，由此进一步衍生出了内容缓存、API 网关、安全防护、协议适配等许许多多的额外特性。这让 NGINX 成长为了一个全能的网络服务器软件。

从上面的介绍中，我们可以看到，NGINX 经过了全球用户和各种实际场景的验证，获得了极大的成功，说它是世界顶级的开源项目之一也丝毫不为过。**而它，正是用标准的 ANSI C 语言开发实现的。**

很自然地，我们会感到好奇：为什么仅仅使用最基本的 C 语言，NGINX 就能够编写出性能如此强劲、功能如此丰富的服务器应用呢？里面究竟有哪些奥秘呢？

如果你能挖掘出这些蕴含在 NGINX 源码之中的“奥秘”和“宝藏”，让它为己所用，无疑会很好地提升自己的 C 语言编程“功力”。这对于我们当前的具体工作，乃至今后的职场发展，都是非常有价值的。

## 我们能从 NGINX 中学到什么？

作为服务器领域里的“全能选手”，NGINX 源码里包含的内容非常丰富，上至配置文件的解析、各种协议的转换、限流限速、访问控制，下至端口监听、信号处理、多进程 / 多线程、epoll 调用，诸如此类，不一而足。可以说，在 Linux 环境里大部分的应用开发问题，都可以在 NGINX 里找到对应或者类似的解决方案。

而且，除了三个例外，这些功能全都是由 NGINX 从零开始编码实现的，具有高度的独立性。这三个例外的功能是数据压缩、正则表达式和加密解密，它们是由 NGINX 之外的开源项目 zlib、PCRE、OpenSSL 来完成的。而 zlib、PCRE、OpenSSL 这三个库，也是用 C 语言开发的久负盛誉的开源库，NGINX 是为了避免“重复造轮子”，这也情有可原。

由于 NGINX 里可研究的地方实在太多，下面我就挑出两个比较有代表性的知识点，给你简单地介绍一下。

### 第一个点，是 NGINX 的跨平台兼容能力。

我们都知道，操作系统的世界里不只有主流的 Windows、Linux，还有 macOS、FreeBSD 等等，而且每个操作系统还有不同的版本区别。

Java、Python 等语言有虚拟机，完全屏蔽了这些差异，而 C 程序更接近底层硬件，通常要直接调用系统函数编写代码，这就让代码的跨平台兼容成了一个大问题。

而 NGINX 却很好地实现了多系统平台的支持功能，能够在 Windows、Linux、macOS、FreeBSD、Solaris 等许多操作系统上运行，并且还兼容 GCC、Clang、Intel C 等不同的编译器和更下层的 x86、arm、SPARC 等硬件。那么，它是怎么做到的呢？

其实原理也很简单，就是引入“中间层”。具体手法是使用宏、条件编译还有包装函数，在代码层面把不同的系统底层调用封装起来。这样，上层使用的时候看到的就是一致的接口，不用再关心如何处理系统差异的“杂事”。

比如，对于 UNIX 系统里的非阻塞错误码（errno），NGINX 就使用条件编译，统一定义成宏 NGX\_EAGAIN，从而消除了 HP-UX 与 Linux、FreeBSD 等其他系统的差异：

[复制代码](#)

```
1 #if (__hpux__)
2 #define NGX_EAGAIN      EWOULDBLOCK
3 #else
4 #define NGX_EAGAIN      EAGAIN
5 #endif
```


又比如，对于常用的函数 memcpy，NGINX 先是使用宏做了一层包装，然后再针对 Intel C 编译器做特别优化，最终使用的函数实际上是 ngx\_copy：

[复制代码](#)

```
1 #define ngx_memcpy(dst, src, n) (void) memcpy(dst, src, n)
2 #define ngx_cpymem(dst, src, n) (((u_char *) memcpy(dst, src, n)) + (n))
3
4 #if ( __INTEL_COMPILER >= 800 )
5 static ngx_inline u_char *
6 ngx_copy(u_char *dst, u_char *src, size_t len);
7 #else
8 #define ngx_copy ngx_cpymem
9 #endif
```

NGINX 还把许多操作系统独有的功能，分别定义在不同的头文件里，像 ngx\_linux\_config.h、ngx\_darwin\_config.h。然后，检测编译时的操作系统，再使用条件

编译的方式包含进来，实现了针对不同操作系统的定制化：

 复制代码

```
1 #if (NGX_FREEBSD)
2 #include <ngx_freebsd_config.h>
3
4 #elif (NGX_LINUX)
5 #include <ngx_linux_config.h>
6
7 #elif (NGX_SOLARIS)
8 #include <ngx_solaris_config.h>
9
10 #elif (NGX_DARWIN)
11 #include <ngx_darwin_config.h>
12
13 #elif (NGX_WIN32)
14 #include <ngx_win32_config.h>
15
16 #else /* POSIX */
17 #include <ngx_posix_config.h>
18
19 #endif
```


所以，研究了 NGINX 源码之后，我们就可以学会跨平台、兼容多系统这个对于 C 语言来说非常重要的技巧。

## 下面我们来看第二个点，NGINX 的内存管理能力。

在 C 语言里，使用动态内存的标准函数是 malloc 和 free，但反复地调用它们分配和释放操作效率很低，而且容易导致内存碎片，影响系统稳定。

为了解决这个问题，NGINX 构造了两种内存池，块式内存池 ngx\_pool 和页式内存池 ngx\_slab\_pool。原理是预先向系统申请较大的一块内存，之后自己在里面按需切分使用，使用完毕后再一次性释放。这样，就减少了系统调用的次数，也消除了内存碎片。

块式内存池 ngx\_pool 多用在请求处理这种内存使用量不确定、生命周期短的场景，它的数据结构简单摘录如下：

 复制代码

```
1 typedef struct ngx_pool_s ngx_pool; // ngx_palloc.h
```

```
2 struct ngx_pool_s {
3     ngx_pool_data_t    d;    // 描述本内存池节点的信息；
4     size_t              max;  // 可分配的最大块；
5     ngx_pool            *current; // 当前使用的内存池节点；
6     ngx_chain_t         *chain; // 为 chain 做的优化，空闲缓冲区链表；
7     ngx_pool_large_t    *large; // 大块的内存，串成链表；
8     ngx_pool_cleanup_t  *cleanup; //清理链表头指针；
9 };
10
```

ngx\_pool 实际上是一个内存块链表，使用指针串联起多块动态分配的内存。小片的内存可以在块里直接移动指针分配，大块的内存就直接调用 malloc 分配，这就兼顾了不同的内存需求，非常灵活。

页式内存池 ngx\_slab\_pool 多用在进程间的共享内存、生命周期较长的场景。由于共享内存通常容量固定，不能动态增长，所以就需要“精打细算”，管理的难度比 ngx\_pool 高很多。

ngx\_slab\_pool 的数据结构摘录如下：

[复制代码](#)

```
1 typedef struct {
2     size_t      min_size; // 最小分配数量，通常是 8 字节；
3     size_t      min_shift; // 最小左移，通常是 3，即 2^3=8；
4     ngx_slab_page_t *pages; // 页数组；
5     ngx_slab_page_t *last; // 页链表指针，最后一页；
6     ngx_slab_page_t free; // 空闲页链表头节点；
7     ngx_slab_stat_t *stats; // 统计信息数组；
8     ngx_uint_t      pfree; // 空闲页数量；
9     u_char          *start; // 共享内存的开始地址；
10    u_char          *end; // 共享内存的末尾地址；
11    void             *addr; // 内存的起始地址；
12 } ngx_slab_pool;
```

ngx\_slab\_pool 把内存划分成 4K 大小的 page，每个 page 又可以再划分成更小的 8 字节、16 字节、32 字节的 slab。

然后，NGINX 把这些 page 串成链表，就形成了一整片连续的内存。分配大块内存的时候，就从链表里摘下多个 page，释放的时候再重新接入链表；而分配小块内存的时候就取一个 page，使用 bitmap 的方式在 page 里切割小片的 slab 分配。



可以看到，NGINX 有着优秀的内存管理能力，能够应用于多种应用场景。我们完全可以把这些源码引入到自己的项目里，从而提升内存的使用效率。

## 如何阅读 NGINX 的源码？

看到 NGINX 源码里的这些编程技巧和架构设计，是不是觉得很值得借鉴？我猜你已经有种跃跃欲试的感觉了，那现在就动手下载源码并阅读吧。你可以从 [🔗 NGINX 官网](#)上下载源码压缩包，或者从 GitHub [🔗 这个网址](#)上获取 NGINX 的源码。

不过我必须要提醒你：虽然 NGINX 的源码写得很规范，但阅读起来并不轻松。

拿当前的稳定版 NGINX 1.20 来说，源码大约有 15 万行（不包含空行）。虽然它的体量跟其他开源项目比起来算是中等规模，但想要弄懂其中错综复杂的数据结构和 workflows，也是一个很大的挑战。

所以，如果我们真的要把 NGINX 的源码读懂、读透，还是需要做些准备工作的。最好的方式就是**分而治之，有的放矢**。

**首先，我们需要大概了解 NGINX 源码的组成结构。**

获取 NGINX 源码后，可以看到它里面有很多文件，目录结构我列在了下面：

```
1 nginx
2 |— auto
3 |— conf
4 |— contrib
5 |— html
6 |— man
7 |— src
```

[📄 复制代码](#)

其中的 auto、conf、html，是预编译和配置示例等辅助文件，我们不需要太过关心。真正的 C 源码在 src 目录，这里又细分出几个子目录：

```
1 nginx/src
2 |— core
```

[📄 复制代码](#)

```
3 |— event
4 |   |— modules
5 |— http
6 |   |— modules
7 |   |— v2
8 |— mail
9 |— misc
10|— os
11|   |— unix
12|— stream
```

这些子目录的命名都很明确，可以说是一目了然。比如，core 就是 NGINX 的核心框架功能，event 就是事件驱动机制，http 就是 HTTP 协议请求处理，os 就是具体的操作系统接口封装。

**然后，你就可以有针对性地去查看这些目录里的源码了。**我建议你只看自己感兴趣，或者当前急需了解的功能，避免不必要的时间和精力浪费。

接下来，在阅读 NGINX 源码的时候，我们也需要**提前了解一下 NGINX 的编码风格，知道它的一些开发约定**。这样，就可以比较容易地进入状态，减少理解 NGINX 源码的障碍。

这里，我简单列出一些 NGINX 源码的特点，你可以参考：

类型名、函数名使用前缀 “ngx\_” ，宏使用前缀 “NGX\_” ；

枚举类型使用后缀 “\_e” ；

函数指针类型使用后缀 “\_pt” ；

结构体（struct）使用后缀 “\_s” ，同名的 “\_t” 后缀是它的等价形式；

为了节约内存，一些变量使用了“位域”特性（bit field）；

大量使用了 void\* 指针，通过类型强制转换，实现了其他语言里的类、泛型的功能。

## 小结

好了，今天的分享就到这里吧，最后做一个简单的小结：

1. 学习 C 语言编程，可以选择阅读经典著作、工作开发实践和钻研开源项目三种方式；



2. NGINX 集成了 Web 服务器、反向代理、负载均衡等能力，是一个优秀的 C 语言开源项目；
3. NGINX 源码里蕴含了非常精妙的编程技巧和设计思想，非常值得仔细研究；
4. NGINX 源码规模很大，要有选择、有目标地去阅读，这样才能事半功倍。

## 课外小贴士

1. W3Techs 的统计报告可参考 [这个链接](#)。报告中有一个有趣的地方：排名第三的 CloudFlare Server 也是基于 NGINX 的。如果把它也算作是 NGINX 的话，那么 NGINX 的总份额就遥遥领先了，是绝对的“霸主”。
2. 在 DockerHub 网站上，NGINX 也是最受欢迎的项目之一，官方镜像的下载量已经超过了 10 亿次，与 MySQL、Redis、Python、Go 等流行技术是同一数量级。
3. GitHub 上的 NGINX 源码实际上只是一个同步镜像，而真正的源码位置在 [这里](#)，是用一个比较“另类”的版本控制软件 Mercurial 管理的。
4. NGINX 公司在 2019 年被 F5 networks 公司收购，之后就正式进入了中国市场，陆续推出了中文官网 [nginx.org.cn](#) 和 [nginx-cn.net](#)，分别面向社区用户和商业用户。网站里有很多技术干货，而且都翻译成了中文，值得一读。
5. 如果觉得直接看源码有困难，那么可以参考 GitHub 上的一些源码注释项目，比如我的：[https://github.com/chronolaw/annotated\\_nginx](https://github.com/chronolaw/annotated_nginx)。另外，我还有一本书《NGINX 完全开发指南》，里面有对 NGINX 数据结构、运行机制的详细解说，还配有图例，结合着源码注释，相信可以帮助你更好地学习。

分享给需要的人，Ta 订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 1     提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇    LMOS说C语言 | 用面向对象的思想开发C语言程序

## 更多课程推荐

## 陈天 · Rust 编程第一课

实战驱动，快速上手 Rust

陈天

Tubi TV 研发副总裁



涨价倒计时 🕒

今日订阅 **¥89**，1月12日涨价至**¥199**

## 精选留言 (1)

💬 写留言



=

2022-01-10

感谢罗老师分享的C语言进阶学习方向。

老师提到的Nginx跨系统平台的C语言实现让我联想起之前在专栏中看过的C语言标准IO，它也屏蔽了低级IO以及具体的操作系统内核提供的系统调用函数。

我对内存管理比较感兴趣，在接下来要学习一下Nginx中的内存管理。感谢罗老师指出了一条前行的道路。

展开 ▾

