

34 | Nginx：高性能的Web服务器

2019-08-14 Chrono

《透视HTTP协议》

课程介绍 >



讲述：Chrono

时长 10:23 大小 14.28M



经过前面几大模块的学习，你已经完全掌握了 HTTP 的所有知识，那么接下来请收拾一下行囊，整理一下装备，跟我一起去探索 HTTP 之外的广阔天地。

现在的互联网非常发达，用户越来越多，网速越来越快，HTTPS 的安全加密、HTTP/2 的多路复用等特性都对 Web 服务器提出了非常高的要求。一个好的 Web 服务器必须要具备稳定、快速、易扩展、易维护等特性，才能够让网站“立于不败之地”。

那么，在搭建网站的时候，应该选择什么样的服务器软件呢？

在开头的几讲里我也提到过，Web 服务器就那么几款，目前市面上主流的只有两个：Apache 和 Nginx，两者合计占据了近 90% 的市场份额。

今天我要说的就是其中的 Nginx，它是 Web 服务器的“后起之秀”，虽然比 Apache 小了 10 岁，但增长速度十分迅猛，已经达到了与 Apache“平起平坐”的地位，而在“Top Million”网站

领资料



中更是超过了 Apache，拥有超过 50% 的用户（[🔗参考数据](#)）。



在这里必须要说一下 Nginx 的正确发音，它应该读成“Engine X”，但我个人感觉“X”念起来太“拗口”，还是比较倾向于读做“Engine ks”，这也与 UNIX、Linux 的发音一致。

作为一个 Web 服务器，Nginx 的功能非常完善，完美支持 HTTP/1、HTTPS 和 HTTP/2，而且还在不断进步。当前的主线版本已经发展到了 1.17，正在进行 HTTP/3 的研发，或许一年之后就能在 Nginx 上跑 HTTP/3 了。

Nginx 也是我个人的主要研究领域，我也写过相关的书，按理来说今天的课程应该是“手拿把攥”，但真正动笔的时候还是有些犹豫的：很多要点都已经在书里写过了，这次的专栏如果再重复相同的内容就不免有“骗稿费”的嫌疑，应该有些“不一样的东西”。

所以我决定抛开书本，换个角度，结合 HTTP 协议来讲 Nginx，带你窥视一下 HTTP 处理的内幕，看看 Web 服务器的工作原理。

进程池

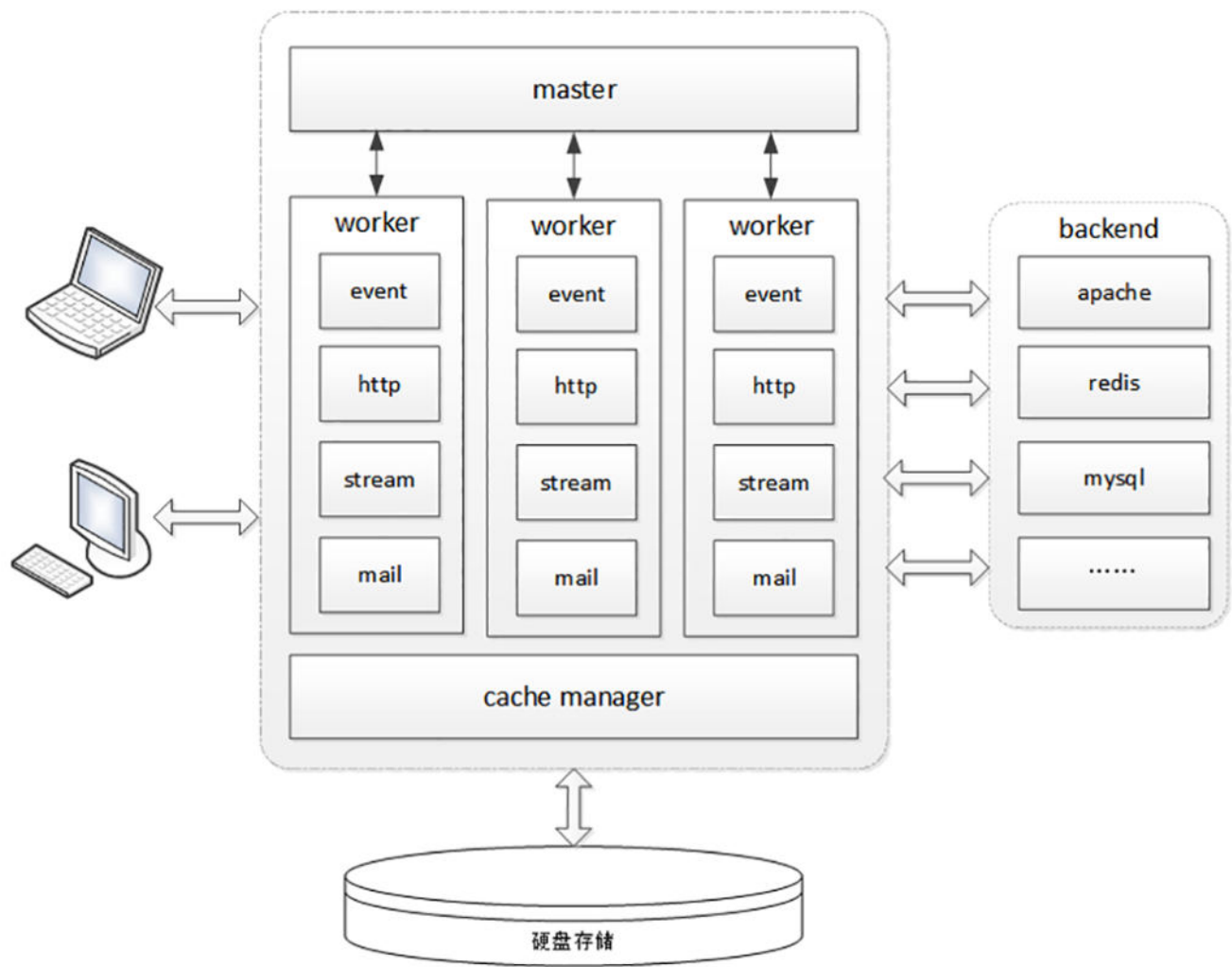
你也许听说过，Nginx 是个“轻量级”的 Web 服务器，那么这个所谓的“轻量级”是什么意思呢？

“轻量级”是相对于“重量级”而言的。“重量级”就是指服务器进程很“重”，占用很多资源，当处理 HTTP 请求时会消耗大量的 CPU 和内存，受到这些资源的限制很难提高性能。

领资料

而 Nginx 作为“轻量级”的服务器，它的 CPU、内存占用都非常少，同样的资源配置下就能够为更多的用户提供服务，其奥秘在于它独特的工作模式。





在 Nginx 之前，Web 服务器的工作模式大多是“Per-Process”或者“Per-Thread”，对每一个请求使用单独的进程或者线程处理。这就存在创建进程或线程的成本，还会有进程、线程“上下文切换”的额外开销。如果请求数量很多，CPU 就会在多个进程、线程之间切换时“疲于奔命”，平白地浪费了计算时间。

Nginx 则完全不同，“一反惯例”地没有使用多线程，而是使用了“**进程池 + 单线程**”的工作模式。

Nginx 在启动的时候会预先创建好固定数量的 worker 进程，在之后的运行过程中不会再 fork 出新进程，这就是进程池，而且可以自动把进程“绑定”到独立的 CPU 上，这样就完全消除了进程创建和切换的成本，能够充分利用多核 CPU 的计算能力。

在进程池之上，还有一个“master”进程，专门用来管理进程池。它的作用有点像是 supervisor（一个用 Python 编写的进程管理工具），用来监控进程，自动恢复发生异常的 worker，保持进程池的稳定和服务能力。



不过 master 进程完全是 Nginx 自行用 C 语言实现的，这就摆脱了外部的依赖，简化了 Nginx 的部署和配置。

I/O 多路复用

如果你用 Java、C 等语言写过程序，一定很熟悉“多线程”的概念，使用多线程能够很容易实现并发处理。

但多线程也有一些缺点，除了刚才说到的“上下文切换”成本，还有编程模型复杂、数据竞争、同步等问题，写出正确、快速的多线程程序并不是一件容易的事情。

所以 Nginx 就选择了单线程的方式，带来的好处就是开发简单，没有互斥锁的成本，减少系统消耗。

那么，疑问也就产生了：为什么单线程的 Nginx，处理能力却能够超越其他多线程的服务器呢？

这要归功于 Nginx 利用了 Linux 内核里的一件“神兵利器”，**I/O 多路复用接口**，“大名鼎鼎”的 `epoll`。

“多路复用”这个词我们已经在之前的 HTTP/2、HTTP/3 里遇到过好几次，如果你理解了那里的“多路复用”，那么面对 Nginx 的 `epoll`“多路复用”也就好办了。

Web 服务器从根本上来说是“I/O 密集型”而不是“CPU 密集型”，处理能力的关键在于网络收发而不是 CPU 计算（这里暂时不考虑 HTTPS 的加解密），而网络 I/O 会因为各式各样的原因不得不等待，比如数据还没到达、对端没有响应、缓冲区满发不出去等等。

这种情形就有点像是 HTTP 里的“队头阻塞”。对于一般的单线程来说 CPU 就会“停下来”，造成浪费。而多线程的解决思路有点类似“并发连接”，虽然有的线程可能阻塞，但由于多个线程并行，总体上看阻塞的情况就不会太严重了。

Nginx 里使用的 `epoll`，就好像是 HTTP/2 里的“多路复用”技术，它把多个 HTTP 请求处理打散成碎片，都“复用”到一个单线程里，不按照先来后到的顺序处理，而是只当连接上真正可读、可写的时候才处理，如果可能发生阻塞就立刻切换出去，处理其他的请求。

领资料



通过这种方式，Nginx 就完全消除了 I/O 阻塞，把 CPU 利用得“满满当当”，又因为网络收发并不会消耗太多 CPU 计算能力，也不需要切换进程、线程，所以整体的 CPU 负载是相当低的。

这里我画了一张 Nginx“I/O 多路复用”的示意图，你可以看到，它的形式与 HTTP/2 的流非常相似，每个请求处理单独来看是分散、阻塞的，但因为都复用到到了一个线程里，所以资源的利用率非常高。



epoll 还有一个特点，大量的连接管理工作都是在操作系统内核里做的，这就减轻了应用程序的负担，所以 Nginx 可以为每个连接只分配很小的内存维护状态，即使有几万、几十万的并发连接也只会消耗几百 M 内存，而其他的 Web 服务器这个时候早就“Memory not enough”了。

多阶段处理

有了“进程池”和“I/O 多路复用”，Nginx 是如何处理 HTTP 请求的呢？

Nginx 在内部也采用的是“化整为零”的思路，把整个 Web 服务器分解成了多个“功能模块”，就好像是乐高积木，可以在配置文件里任意拼接搭建，从而实现了高度的灵活性和扩展性。

Nginx 的 HTTP 处理有四大类模块：

- 1. handler 模块：直接处理 HTTP 请求；
- 2. filter 模块：不直接处理请求，而是加工过滤响应报文；

领资料



3. upstream 模块：实现反向代理功能，转发请求到其他服务器；

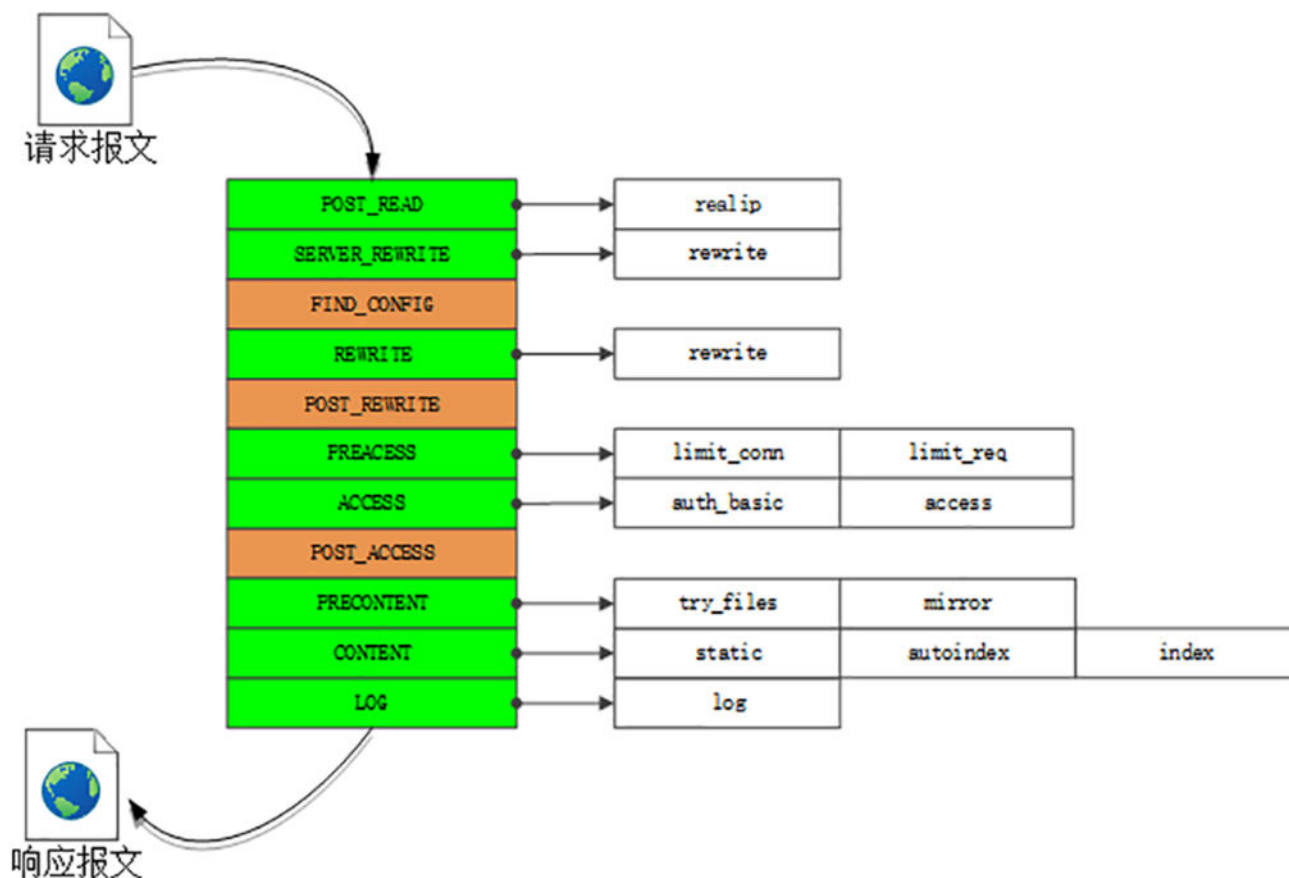
4. balance 模块：实现反向代理时的负载均衡算法。

因为 upstream 模块和 balance 模块实现的是代理功能，Nginx 作为“中间人”，运行机制比较复杂，所以我今天只讲 handler 模块和 filter 模块。

不知道你有没有了解过“设计模式”这方面的知识，其中有一个非常有用的模式叫做“**职责链**”。它就好像是工厂里的流水线，原料从一头流入，线上有许多工人会进行各种加工处理，最后从另一头出来的就是完整的产品。

Nginx 里的 handler 模块和 filter 模块就是按照“职责链”模式设计和组织的，HTTP 请求报文就是“原材料”，各种模块就是工厂里的工人，走完模块构成的“流水线”，出来的就是处理完成的响应报文。

下面的这张图显示了 Nginx 的“流水线”，在 Nginx 里的术语叫“阶段式处理”（Phases），一共有 11 个阶段，每个阶段里又有许多各司其职的模块。



领资料



我简单列几个与我们的课程相关的模块吧：

- charset 模块实现了字符集编码转换；（[🔗 第 15 讲](#)）
- chunked 模块实现了响应数据的分块传输；（[🔗 第 16 讲](#)）
- range 模块实现了范围请求，只返回数据的一部分；（[🔗 第 16 讲](#)）
- rewrite 模块实现了重定向和跳转，还可以使用内置变量自定义跳转的 URI；（[🔗 第 18 讲](#)）
- not_modified 模块检查头字段“if-Modified-Since”和“If-None-Match”，处理条件请求；（[🔗 第 20 讲](#)）
- realip 模块处理“X-Real-IP”“X-Forwarded-For”等字段，获取客户端的真实 IP 地址；（[🔗 第 21 讲](#)）
- ssl 模块实现了 SSL/TLS 协议支持，读取磁盘上的证书和私钥，实现 TLS 握手和 SNI、ALPN 等扩展功能；（[🔗 安全篇](#)）
- http_v2 模块实现了完整的 HTTP/2 协议。（[🔗 飞翔篇](#)）

在这张图里，你还可以看到 limit_conn、limit_req、access、log 等其他模块，它们实现的是限流限速、访问控制、日志等功能，不在 HTTP 协议规定之内，但对于运行在现实世界的 Web 服务器却是必备的。

如果你有 C 语言基础，感兴趣的话可以下载 Nginx 的源码，在代码级别仔细看看 HTTP 的处理过程。

小结

1. Nginx 是一个高性能的 Web 服务器，它非常的轻量级，消耗的 CPU、内存很少；
2. Nginx 采用“master/workers”进程池架构，不使用多线程，消除了进程、线程切换的成本；
3. Nginx 基于 epoll 实现了“I/O 多路复用”，不会阻塞，所以性能很高；
4. Nginx 使用了“职责链”模式，多个模块分工合作，自由组合，以流水线的方式处理 HTTP 请求。

领资料



课下作业

1. 你是怎么理解进程、线程上下文切换时的成本的，为什么 Nginx 要尽量避免？
2. 试着自己描述一下 Nginx 用进程、epoll、模块流水线处理 HTTP 请求的过程。

欢迎你把自己的学习体会写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。



== 课外小贴士 ==

- 01 也有不少的人把 Nginx 读成 “NG ks”，这就错得太多了。
- 02 Nginx 自 1.7.11 开始引入了“多线程”，但只是作为辅助手段，卸载阻塞的磁盘 I/O 操作，主要的 HTTP 请求处理使用的还是单线程里的 epoll。
- 03 如何让 Web 服务器能够高效地处理 10K 以上的并发请求（Concurrent 10K），这就是著名的“C10K 问题”，当然它早已经被 epoll/kqueue 等解决了，现在的新问题是“C10M”。
- 04 Nginx 的“PRECONTENT”阶段在 1.13.3 之前叫“TRY_FILES”，仅供 Nginx 内部使用，用

领资料





叫 `TRY_FILES`，仅供 Nginx 内部使用，用户不可介入。

05 正文里的“流水线”图没有画出 filter 模块所在的位置，它其实是在 CONTENT 阶段的末尾，专门“过滤”响应数据。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 12  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 33 | 我应该迁移到HTTP/2吗？

下一篇 35 | OpenResty：更灵活的Web服务器

领资料



JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



精选留言 (30)

写留言



许童童

2019-08-14

你是怎么理解进程、线程上下文切换时的成本的，为什么 Nginx 要尽量避免？

当从一个任务切换到另一个任务，当前任务的上下文，如堆栈，指令指针等都要保存起来，以便下次任务时恢复，然后再把另一个任务的堆栈加载进来，如果有大量的上下文切换，就会影响性能。

试着自己描述一下 Nginx 用进程、epoll、模块流水线处理 HTTP 请求的过程。

Nginx 启动进程，一个master，多个worker，创建epoll，监听端口，多路复用来管理http请求，http请求到达worker内部，通过模块流水线处理，最后返回http响应。

作者回复: ✓



27

领资料



夏目

2019-12-10

好像高性能的服务都是这样玩的，nginx这个架构类似于netty中的多线程reactor模式，redis则是单线程reactor



作者回复: nginx也是单线程的, 和redis一样自己封装了epoll。单线程的好处是没有race condition, 处理简单。

nginx比redis高明的一点是多进程, 提高了稳定性和并发能力。



11



Leon

2019-08-14

一个线程的时间片没用完就系统调用被系统调度切换出去, 浪费了剩余的时间片, nginx通过epoll和注册回调, 和非阻塞io自己在用户态主动切换上下文, 充分利用了系统分配给进程或者线程的时间片, 所以对系统资源利用很充分

作者回复: great。



9



徐海浪

2019-08-15

多线程就好比一条流水线有多个机械手, 把一件事情中途交给其他线程处理, 要交接处理中间状态信息。

单进程就好比一条流水线只有一个机械手, 切换时间片时暂停状态就可以, 不用交接信息, 减少无用功, 所以效率高。

作者回复: great



6



lesserror

2019-12-26

老师, 以下问题, 麻烦回答一下, 谢谢:

1. 把进程“绑定”到独立的 CPU 上。意思是一个CPU专门负责管理进程嘛?

2. 不过 master 进程完全是 Nginx 自行用 C 语言实现的, 这就摆脱了外部的依赖, 简化了 Nginx 的部署和配置。这句话没理解。

作者回复:

1.unix/linux有个特别的功能, 可以让进程“绑定”在一个cpu上运行, 不会被操作系统调度到其他cpu上跑, 这样就减少了切换的成本, 提高运行效率。不是管理进程的意思。配置指令是“worker_cpu_a

领资料



finity”。

2.在unix上有很多服务管理程序，比如systemd、supervisor，可以实现进程监控、自动重启等。而Nginx的master进程实现了同样的功能，就不需要这样的外部程序来管理进程，保持服务的稳定性。



5



-W.LI-

2019-08-14

老师好!我打算学习nginx，有适合初学者的书推荐么?Java工程师，c全忘了。
线程切换开销:线程切换需要进行系统调用。需要从用户态->内核态->用户态。上下文切换，需要保存寄存器中的信息，以便于完成系统调用后还原现场。会多跑很多指令，出入栈会比寄存器慢很多。相对来说开销就很大了。
nginx和redis一样采用单线程模型。是因为cpu计算不可能是它们瓶颈(所以有些耗cpu资源高的计算不适合放在nginx上做会导致响应时间变长)?进程池+单线程是指，每个worker进程都是单线程是么?

作者回复:

1.Nginx的内容很多，看你想学哪方面了。如果是单纯的运维操作网上的资料有很多，如果是想学Nginx开发和源码就看《Nginx完全开发指南》吧。

2.说的很对，看Nginx源码可以学到很多高性能编程的技巧。

3.Nginx里也可以使用多线程，但需要“魔改”。



4



fakership

2020-08-16

老师，有个问题咨询下

虽然nginx是使用了epoll做了io的多路复用，但对于队头阻塞的话感觉并没有帮助啊，因为还是要等io事件回调后发送http响应报文，所以还是阻塞了下一个请求。

作者回复: 是的，但这完全是两个不相关的事情。

队头阻塞是http/1固有的问题，无论是什么web服务器都无法解决，是对单个客户端而言的。

而Nginx的epoll则是解决了多客户端并发请求的问题，避免一个客户端阻塞其他客户端的处理，可以支持海量客户端访问服务器。



3

领资料





Leon

2019-08-14

切换cpu需要保存线程的上下文，然后再切回去，这是开销

作者回复: √

共 3 条评论 >



2



皮特尔

2020-07-09

Nginx这种异步处理方式叫“协程”吧？

作者回复: 不是。

Nginx是用纯C开发的，里面没有协程的概念，它内部用的是epoll事件机制，reactor并发模式，有ready事件就回调。

OpenResty把lua的协程和epoll事件机制结合在了一起，但两者还是不能混为一谈。

共 3 条评论 >



1



Aaron

2020-06-01

对『进程池 + 单线程』的模式还是不太透彻。

我理解，『单线程』指的是所有 HTTP 请求放在同一个线程里通过『I/O 多路复用』的技术处理，实际就是高度集中（无阻塞）地占用了 CPU（核心）地运算能力。

那么，既然请求是单线程的，那进程池地作用又是什么呢？如果是多进程的，不就又回到进程间上下文切换的消耗问题了吗？

另，Nginx 通过 cpu affinity 将进程绑定到 CPU，假设是单 CPU，将三个 worker 进程绑定到同一个物理 CPU 地意义又在哪呢？

个人认为效率最高的方式，是按照 CPU 的核心数量创建一个『线程池』，将所有请求分配到『线程池』内不同的线程，这样在『I/O 多路复用』的加持下能跑满 CPU 的性能。

作者回复:

1.单线程理解的很对。进程池里的每个进程都是独立的，崩溃不会影响整体服务，如果是多线程，那么线程崩溃进程也就完蛋了。

领资料



2.多进程分散运行在多个cpu上，彼此不干扰，就不会出现进程上下文切换。

3.cpu affinity 是可选的，对于单cpu就没有开启的必要，反而会增加进程切换的成本。

4.刚才说，单进程多线程的缺点就是不够稳定，一个线程出问题，整个进程都受影响。

共 2 条评论 >

👍 1



zero

2020-04-29

老师，您好，我想写博客，我写的博客里面能盗一下您的图么（您的图做的太直观了一看就懂了），我会著名图片的出处😁😁

作者回复: 这个要联系极客时间吧，版权在他们那里。



👍 1



J.Smile

2020-01-20

说一下http2和nginx的多路复用区别和联系：

http2的多路复用：多个请求复用同一个连接并行传输数据，且每个请求抽象为流传输的对象为帧序列。

nginx的IO多路复用：将多个线程的请求打散，汇入同一个线程中传输，epoll监听到事件通道可读或者可写的时候取出或者写入数据，所以nginx的IO多路复用是基于linux内核epoll实现的一种事件监听机制，是NIO非阻塞IO。

作者回复: 说的很好。



👍 1



阿锋

2019-08-14

缓存服务器，是属于正向代理还是反向代理，还是根据情况而定。

作者回复: 正向代理和反向代理是根据它所在的位置来定义的，靠近客户端就是正向，靠近服务器就是反向。

代理与缓存是不相关的，代理可以没有缓存功能。

共 2 条评论 >

👍 1

领资料





忧天小鸡
2021-12-30

这里说的nx的epoll是指模仿epoll的交互逻辑，还是指从epoll的base上做了对tcp的改装？

作者回复: Nginx调用操作系统的epoll接口，来处理tcp事件，本质上epoll和tcp没有直接关系，但tcp会有读写事件，就可以利用epoll来处理。



三千世界
2021-11-29

老师我想问一下，nginx为什么要设计让多个worker进程竞争accept，这样导致惊群问题，还要加锁来解决，反而造成了性能下降。

所以，为什么不让master通过epoll监听有连接可以accept，通过调度，找一个不怎么忙的worker，然后通过管道通知这个worker呢，这样就不会出现惊群问题了

作者回复: accept mutex设计的目的是多worker进程之间负载均衡，避免有的worker处理的连接太多。

初衷是好的，在NGINX初期也确实很有效果，但到了现在，并发越来越多，它的锁成本就显得高了。

目前NGINX不推荐使用accept mutex，而是改用Linux系统内核的reuseport来实现负载均衡。

你说的master监听的方式是很传统的做法，效率更低。

共 2 条评论 >



爱编程的运维
2021-11-05

老师您好，nginx采用IO多路复用技术，使用单线程处理多个IO流数据流是不是也可以多线程+IO多路复用技术？多个线程处理多个IO数据流

作者回复: 当然可以，像envoy，还有NGINX Unit都是多线程+io多路复用。



领资料



功夫熊猫
2021-10-27

线程上下文的切换消耗感觉主要是用户态和内核态不断切换。也就是堆栈，指令指针之类的。



作者回复: 对, CPU要保存当前状态, 再恢复原来的状态, 但当线程多的时候, 累积的成本就很高了。



连长

2021-09-19

Nginx 使用进程池加单线程的工作方式, master进程管理进程池, 利用IO多路复用提供并发性能。epoll连接管理由操作系统处理, 减少应用层操作。

作者回复: great。



dog_brother

2021-05-06

老师, 看了越来越多的框架, 觉得epoll是真的牛, epoll是哪个人/框架首创的呀?

作者回复: 应该不是某个人独立发明的吧, 不然早就知名了, 它也是在之前的select、poll上逐渐改进完善才产生的。



脱缰的野马__

2021-03-20

老师你好, tomcat不主流吗?

作者回复: tomcat应该算是java容器吧, 主要是实现业务, 不是专门的web服务器。



领资料

