Nginx 是如何实现并发的?为什么 Nginx 不使用多线程?Nginx常见的优化手段有哪些?502错误可能原因有哪些?

**面试官心理分析**

主要是看应聘人员的对NGINX的基本原理是否熟悉，因为大多数运维人员多多少少都懂点NGINX，但是真正其明白原理的可能少之又少。明白其原理，才能做优化，否则只能照样搬样，出了问题也无从下手。

懂皮毛的人，一般会做个 Web Server，搭建一个 Web 站点;初级运维可能搞个 HTTPS 、配置一个反向代理; 中级运维定义个 upstream、写个正则判断;老鸟做个性能优化、写个ACL，还有可能改改源码(小编表示没有改源码的能力)。

**面试题剖析**

**1. Nginx 是如何实现高并发的?**

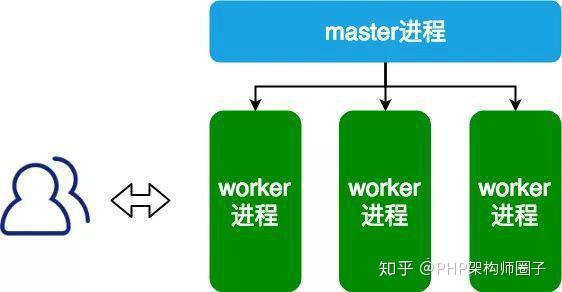
异步，非阻塞，使用了epoll 和大量的底层代码优化。

如果一个server采用一个进程负责一个request的方式，那么进程数就是并发数。正常情况下，会有很多进程一直在等待中。

而nginx采用一个master进程，多个woker进程的模式。

* master进程主要负责收集、分发请求。每当一个请求过来时，master就拉起一个worker进程负责处理这个请求。
* 同时master进程也负责监控woker的状态，保证高可靠性
* woker进程一般设置为跟cpu核心数一致。nginx的woker进程在同一时间可以处理的请求数只受内存限制，可以处理多个请求。

Nginx 的异步非阻塞工作方式正把当中的等待时间利用起来了。在需要等待的时候，这些进程就空闲出来待命了，因此表现为少数几个进程就解决了大量的并发问题。



每进来一个request，会有一个worker进程去处理。但不是全程的处理，处理到什么程度呢?处理到可能发生阻塞的地方，比如向上游(后端)服务器转发request，并等待请求返回。那么，这个处理的worker很聪明，他会在发送完请求后，注册一个事件：“如果upstream返回了，告诉我一声，我再接着干”。于是他就休息去了。此时，如果再有request 进来，他就可以很快再按这种方式处理。而一旦上游服务器返回了，就会触发这个事件，worker才会来接手，这个request才会接着往下走。

**2. 为什么 Nginx 不使用多线程?**

Apache: 创建多个进程或线程，而每个进程或线程都会为其分配 cpu 和内存(线程要比进程小的多，所以worker支持比perfork高的并发)，并发过大会耗光服务器资源。

Nginx: 采用单线程来异步非阻塞处理请求(管理员可以配置Nginx主进程的工作进程的数量)(epoll)，不会为每个请求分配cpu和内存资源，节省了大量资源，同时也减少了大量的CPU的上下文切换。所以才使得Nginx支持更高的并发。

**3. Nginx常见的优化配置有哪些?**



(1) 调整worker\_processes

指Nginx要生成的worker数量,\*\*\*实践是每个CPU运行1个工作进程。

了解系统中的CPU核心数，输入

1. $ grep processor / proc / cpuinfo | wc -l

(2) \*\*\*化worker\_connections

Nginx Web服务器可以同时提供服务的客户端数。与worker\_processes结合使用时，获得每秒可以服务的\*\*\*客户端数

\*\*\*客户端数/秒=工作进程\*工作者连接数

为了\*\*\*化Nginx的全部潜力，应将工作者连接设置为核心一次可以运行的允许的\*\*\*进程数1024。

(3) 启用Gzip压缩

压缩文件大小，减少了客户端http的传输带宽，因此提高了页面加载速度

建议的gzip配置示例如下:( 在http部分内)



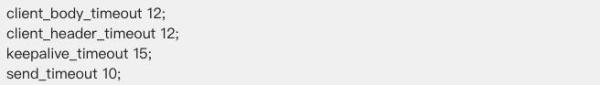
(4) 为静态文件启用缓存

为静态文件启用缓存，以减少带宽并提高性能，可以添加下面的命令，限定计算机缓存网页的静态文件：

1. location ~\* .(jpg|jpeg|png|gif|ico|css|js)$ {
2. expires 365d;
3. }

(5) Timeouts

keepalive连接减少了打开和关闭连接所需的CPU和网络开销，获得\*\*\*性能需要调整的变量可参考：



(6) 禁用access\_logs

访问日志记录，它记录每个nginx请求，因此消耗了大量CPU资源，从而降低了nginx性能。

完全禁用访问日志记录

1. access\_log off;

如果必须具有访问日志记录，则启用访问日志缓冲

1. access\_log /var/log/nginx/access.log主缓冲区= 16k

**4. 502报错可能原因有哪些?**



(1) FastCGI进程是否已经启动

(2) FastCGI worker进程数是否不够

(3) FastCGI执行时间过长

(4) FastCGI Buffer不够

nginx和apache一样，有前端缓冲限制，可以调整缓冲参数

1. fastcgi\_buffer\_size 32k;
2. fastcgi\_buffers 8 32k;

(5) Proxy Buffer不够

如果你用了Proxying，调整

1. proxy\_buffer\_size 16k;
2. proxy\_buffers 4 16k;

(6) php脚本执行时间过长

将php-fpm.conf的

1. <value name="request\_terminate\_timeout">0s</value>

0s改成一个时间