Nginx Interview Problems And Answers

[](https://s4.51cto.com/oss/201906/11/cbf620d171a8ca343cf53f6281a875af.jpg-wh_651x-s_244382094.jpg)

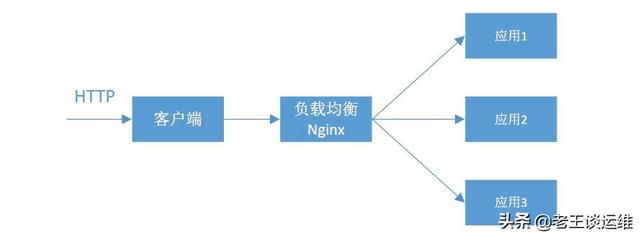
一、Nginx负载均衡的算法怎么实现的?Nginx 有哪些负载均衡策略?Nginx为什么要做动静分离?

1、问题剖析：主要是看对Nginx的基本原理是否熟悉，需要能够根据实际业务场景选择合适的负载均衡(转发)策略。其次，在网站建设上，了解 Nginx 的优劣势，能够找到弥补其不足的方法，使得企业的业务达到\*\*\*访问状态。

2、问题剖析

（1）Nginx负载均衡的算法怎么实现的?策略有哪些?

负载均衡是 Nginx常用的一个功能，当一台服务器在单位时间内的访问量越大时，服务器压力就越大，大到超过自身承受能力时，服务器就会崩溃。为了避免服务器崩溃，大家会通过负载均衡的方式来分担服务器压力。将对台服务器组成一个集群，当用户访问时，先访问到一个转发服务器，再由转发服务器将访问分发到压力更小的服务器。



Nginx负载均衡实现的策略有以下五种：

a.轮询(默认)

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某个服务器宕机，能自动剔除故障系统。

upstream backserver {

 server 192.168.0.12;

 server 192.168.0.13;

}

b.权重 weight

weight的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下。其次是为在主从的情况下设置不同的权值，达到合理有效的地利用主机资源。

upstream backserver {

 server 192.168.0.12 weight=2;

 server 192.168.0.13 weight=8;

}

权重越高，在被访问的概率越大，如上例，分别是20%，80%。

c.ip\_hash( IP绑定)

每个请求按访问IP的哈希结果分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的session共享问题

upstream backserver {

 ip\_hash;

 server 192.168.0.12:88;

 server 192.168.0.13:80;

}

e.fair(第三方插件)

必须安装upstream\_fair模块。

对比 weight、ip\_hash更加智能的负载均衡算法，fair算法可以根据页面大小和加载时间长短智能地进行负载均衡，响应时间短的优先分配。

upstream backserver {

 server server1;

 server server2;

 fair;

}

哪个服务器的响应速度快，就将请求分配到那个服务器上。

f.url\_hash(第三方插件)

必须安装Nginx的hash软件包

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率。

upstream backserver {

 server squid1:3128;

 server squid2:3128;

 hash $request\_uri;

 hash\_method crc32;

}

2. 为什么要做动静分离?

Nginx是当下最热的Web容器，网站优化的重要点在于静态化网站，网站静态化的关键点则是是动静分离，动静分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们则根据静态资源的特点将其做缓存操作。

让静态的资源只走静态资源服务器，动态的走动态的服务器

Nginx的静态处理能力很强，但是动态处理能力不足，因此，在企业中常用动静分离技术。

[](https://s5.51cto.com/oss/201906/11/5cf3a97ef1cc797329daab2ddba9eab0.jpg)

对于静态资源比如图片，js，css等文件，我们则在反向代理服务器nginx中进行缓存。这样浏览器在请求一个静态资源时，代理服务器nginx就可以直接处理，无需将请求转发给后端服务器tomcat。

若用户请求的动态文件，比如servlet,jsp则转发给Tomcat服务器处理，从而实现动静分离。这也是反向代理服务器的一个重要的作用。

**面试题：**

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?**

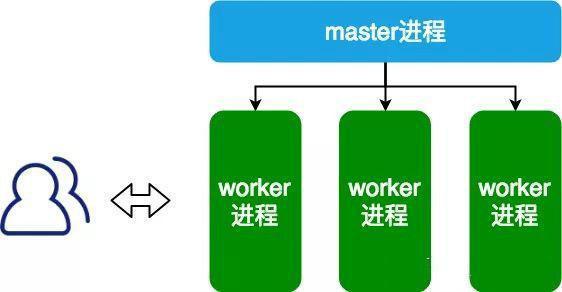
异步，非阻塞，使用了epoll 和大量的底层代码优化。

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