

请解释一下什么是 Nginx ?

Nginx , 是一个 Web 服务器和反向代理服务器, 用于 HTTP、HTTPS、SMTP、POP3 和 IMAP 协议。


目前使用的最多的 Web 服务器或者代理服务器, 像淘宝、新浪、网易、迅雷等都在使用。

Nginx 的主要功能如下:

- 作为 http server (代替 Apache , 对 PHP 需要 FastCGI 处理器支持)

FastCGI: Nginx 本身不支持 PHP 等语言, 但是它可以通过 FastCGI 来将请求扔给某些语言或框架处理。

- 反向代理服务器
- 实现负载均衡
- 虚拟主机

 fastcgi 与 cgi 的区别?


1) cgi

web 服务器会根据请求的内容, 然后会 fork 一个新进程来运行外部 c 程序 (或 perl 脚本...), 这个进程会把处理完的数据返回给 web 服务器, 最后 web 服务器把内容发送给用户, 刚才 fork 的进程也随之退出。

如果下次用户还请求改动态脚本, 那么 web 服务器又再次 fork 一个新进程, 周而复始的进行。

2) fastcgi

web 服务器收到一个请求时, 他不会重新 fork 一个进程 (因为这个进程在 web 服务器启动时就开启了, 而且不会退出), web 服务器直接把内容传递给这个进程 (进程间通信, 但 fastcgi 使用了别的方式, tcp 方式通信), 这个进程收到请求后进行处理, 把结果返回给 web 服务器, 最后自己接着等待下一个请求的到来, 而不是退出。

 综上, 差别在于是否重复 fork 进程, 处理请求。

关于 Nginx 和 fastcgi 是如何运行的, 可以看看 [《Nginx + FastCGI运行原理》](#)。

同样, 关于 cgi 可能也会存在 [《php 面试题五之 nginx 如何调用 php 和 php-fpm 的作用和工作原理》](#) 问题。

Nginx 常用命令?


- 启动 nginx 。
- 停止 nginx -s stop 或 nginx -s quit 。
- 重载配置 ./sbin/nginx -s reload (平滑重启) 或 service nginx reload 。
- 重载指定配置文件 nginx -c /usr/local/nginx/conf/nginx.conf 。
- 查看 nginx 版本 nginx -v 。
- 检查配置文件是否正确 nginx -t 。
- 显示帮助信息 nginx -h 。

Nginx 常用配置?

```
worker_processes 8; # 工作进程个数
worker_connections 65535; # 每个工作进程能并发处理 (发起) 的最大连接数 (包含所有连接数)
error_log /data/logs/nginx/error.log; # 错误日志打印地址
access_log /data/logs/nginx/access.log; # 进入日志打印地址
log_format main '$remote_addr'$request' '$status $upstream_addr "$request time"'; # 进入日志格式

## 如果未使用 fastcgi 功能的, 可以无视
fastcgi_connect_timeout=300; # 连接到后端 fastcgi 超时时间
fastcgi_send_timeout=300; # 向 fastcgi 请求超时时间 (这个指定值已经完成两次握手后向fastcgi传送请求的超时时间)
fastcgi_read_timeout=300; # 接收 fastcgi 应答超时时间, 同理也是2次握手后
fastcgi_buffer_size=64k; # 读取 fastcgi 应答第一部分需要多大缓冲区, 该值表示使用1个64kb的缓冲区读取应答第一部分 (应答头), 可以设置为
fastcgi_buffers选项缓冲区大小
fastcgi_buffers 4 64k; # 指定本地需要多少和多大的缓冲区来缓冲fastcgi应答请求, 假设一个php或java脚本所产生页面大小为256kb, 那么会为其分配4个64kb的缓冲来缓存
fastcgi_cache TEST; # 开启fastcgi缓存并为其指定为TEST名称, 降低cpu负载, 防止502错误发生

listen 80; # 监听端口
server_name rrc.test.jiedaibao.com; # 允许域名
root /data/release/rrc/web; # 项目根目录
index index.php index.html index.htm; # 访问根文件
```

 Nginx 日志格式中的 \$time\_local 表示的是什么时候? 请求开始的时间? 请求结束的时间? 其次, 当我们从前到后观察日志中的 \$time\_local 时间时, 有时候会发现时间顺序前后错乱的现象, 请说明原因?

\$time\_local : 在服务器里请求开始写入本地的时间。

因为请求发生时间有前有后, 所以会时间顺序前后错乱。

Nginx 有哪些优点?

- 跨平台、配置简单。
- 非阻塞、高并发连接

处理 2-3 万并发连接数，官方监测能支持 5 万并发。

- 内存消耗小

开启 10 个 Nginx 才占 150M 内存。

- 成本低廉，且开源。
- 稳定性高，宕机的概率非常小。

使用“反向代理服务器”的优点是什么？

反向代理服务器可以隐藏源服务器的存在和特征。它充当互联网云和 Web 服务器之间的中间层。这对于安全方面来说是很好的，特别是当我们使用 Web 托管服务时。

这里，先不考虑负载均衡。

### 🚀 什么是正向代理？

一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。

- 客户端才能使用正向代理。
- 正向代理总结就一句话：代理端代理的是客户端。例如说：🐼 我们使用的翻墙软件，OpenVPN 等等。

### 🚀 什么是反向代理？

反向代理（Reverse Proxy）方式，是指以代理服务器来接受 Internet 上的连接请求，然后将请求，发给内部网络上的服务器并将从服务器上得到的结果返回给 Internet 上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

反向代理总结就一句话：代理端代理的是服务端。

### 🚀 请列举 Nginx 和 Apache 之间的不同点？

Nginx	Apache
<ul style="list-style-type: none"> <li>• Nginx是一个基于事件的web服务器</li> <li>• 所有请求都由一个线程处理</li> <li>• Nginx避免子进程的概念</li> <li>• Nginx类似于速度</li> <li>• Nginx在内存消耗和连接方面比较好</li> <li>• Nginx在负载均衡方面表现较好</li> <li>• 对于PHP来说，Nginx可能更可取，因为它支持PHP</li> <li>• Nginx不支持像IBMi和OpenVMS一样的OS。</li> <li>• Nginx只具有核心功能</li> <li>• Nginx的性能和可伸缩性不依赖于硬件</li> </ul>	<ul style="list-style-type: none"> <li>• Apache是一个基于流程的服务器</li> <li>• 单个线程处理单个请求</li> <li>• Apache是基于子进程的</li> <li>• Apache类似于功率</li> <li>• Apache在内存消耗和连接上并没有提高</li> <li>• 当流量到达进程的极限时，Apache将拒绝新的连接</li> <li>• Apache支持的PHP、Python、Perl和其他语言使用插件，当应用程序基于Python或Ruby时，它非常有用</li> <li>• Apache支持更多的OS</li> <li>• Apache提供了比Nginx更多的功能</li> <li>• Apache依赖于CPU和内存等硬件组件</li> </ul>

- 轻量级，同样起 web 服务，Nginx 比 Apache 占用更少的内存及资源。
- 抗并发，Nginx 处理请求是异步非阻塞的，而 Apache 则是阻塞型的，在高并发下 Nginx 能保持低资源低消耗高性能。
- 最核心的区别在于 Apache 是同步多进程模型，一个连接对应一个进程；Nginx 是异步的，多个连接（万级别）可以对应一个进程。
- Nginx 高度模块化的设计，编写模块相对简单。

### 🚀 LVS、Nginx、HAProxy 有什么区别？

- LVS：是基于四层的转发。
- HAProxy：是基于四层和七层的转发，是专业的代理服务器。
- Nginx：是 WEB 服务器，缓存服务器，又是反向代理服务器，可以做七层的转发。

Nginx 引入 TCP 插件之后，也可以支持四层的转发。

### 🚀 区别

LVS 由于是基于四层的转发所以只能做端口的转发，而基于 URL 的、基于目录的这种转发 LVS 就做不了。

### 🚀 工作选择：

HAProxy 和 Nginx 由于可以做七层的转发，所以 URL 和目录的转发都可以做，在很大并发量的时候我们就要选择 LVS，像中小型公司的话并发量没那么大选择 HAProxy 或者 Nginx 足已。

由于 HAProxy 由是专业的代理服务器配置简单，所以中小型企业推荐使用HAProxy 。

有些使用，使用 HAProxy 还是 Nginx，也和公司运维对哪个技术栈的掌控程度。掌控 OK，选择 Nginx 会更加不错。

另外，LVS + Nginx 和 LVS + HAProxy 也是比较常见的选型组合。

更多更详细的对比，感兴趣的胖友，可以看看《LVS、HAProxy、Nginx 负载均衡的比较分析》。

### 🦅 Squid、Varinsh、Nginx 有什么区别？

三者都实现缓存服务器的作用。所以，本问题所有的视角，都是在作为缓存服务器下来聊。

- 1、Nginx 本来是反向代理/web服务器，用了插件可以做做这个副业(缓存服务器)。

但是本身不支持特性挺多，只能缓存静态文件。

- 2、从这些功能上，Varinsh 和 Squid 是专业的 Cache 服务，而Nginx 这些是第三方模块完成。
- 3、Varnish 本身的技术上优势要高于 Squid，它采用了可视化页面缓存技术。
  - 在内存的利用上，Varnish 比 Squid 具有优势，性能要比 Squid 高。
  - 还有强大的通过 Varnish 管理端口，可以使用正则表达式快速、批量地清除部分缓存
  - Varnish 是内存缓存，速度一流，但是内存缓存也限制了其容量，缓存页面和图片一般是挺好的。
- 4、Squid 的优势在于完整的庞大的 cache 技术资料，和很多的应用生产环境。

### 🚀 工作选择：

要做 cache 服务的话，我们肯定是要选择专业的 cache 服务，优先选择Squid 或者 Varnish。

请解释 Nginx 如何处理 HTTP 请求？

- 首先，Nginx 在启动时，会解析配置文件，得到需要监听的端口与 IP 地址，然后在 Nginx 的 Master 进程里面先初始化好这个监控的 Socket(创建 S socket，设置 addr、reuse 等选项，绑定到指定的 ip 地址端口，再 listen 监听)。
- 然后，再 fork(一个现有进程可以调用 fork 函数创建一个新进程。由 fork 创建的新进程被称为子进程)出多个子进程出来。
- 之后，子进程会竞争 accept 新的连接。此时，客户端就可以向 nginx 发起连接了。当客户端与nginx进行三次握手，与 nginx 建立好一个连接后。此时，某一个子进程会 accept 成功，得到这个建立好的连接的 Socket，然后创建 nginx 对连接的封装，即 ngx\_connection\_t 结构体。
- 接着，设置读写事件处理函数，并添加读写事件来与客户端进行数据的交换。

这里，还是有一些逻辑，继续在「Nginx 是如何实现高并发的？」问题中来看。

- 最后，Nginx 或客户端来主动关掉连接，到此，一个连接就寿终正寝了。

### 🦅 Nginx 是如何实现高并发的？

如果一个 server 采用一个进程(或者线程)负责一个request的方式，那么进程数就是并发数。那么显而易见的，就是会有很多进程在等待中。等什么？最多的应该是等待网络传输。其缺点胖友应该也感觉到了，此处不述。

思考下，Java 的 NIO 和 BIO 的对比哟。

而 Nginx 的异步非阻塞工作方式正是利用了这点等待的时间。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。

Nginx是如何利用的呢，简单来说：同样的 4 个进程，如果采用一个进程负责一个 request 的方式，那么，同时进来 4 个 request 之后，每个进程就负责其中一个，直至会话关闭。期间，如果有第 5 个request进来了。就无法及时反应了，因为 4 个进程都没干完活呢，因此，一般有个调度进程，每当新进来了一个 request，就新开个进程来处理。

回想下，BIO 是不是存在酱紫的问题？嘻嘻。

Nginx 不这样，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 不会这么傻等着，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。

这就是为什么说，Nginx 基于事件模型。

由于 web server 的工作性质决定了每个 request 的大部份生命都是在网络传输中，实际上花费在 server 机器上的时间片不多。这是几个进程就解决高并发的秘密所在。即：

webserver 刚好属于网络 IO 密集型应用，不算是计算密集型。

而正如叔度所说的

异步，非阻塞，使用 epoll，和大量细节处的优化。

也正是 Nginx 之所以然的技术基石。

### 🦅 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)，并发过大会榨干服务器资源。

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。

Netty、Redis 基本采用相同思路。

什么是动态资源、静态资源分离？

动态资源、静态资源分离，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就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路。

动态资源、静态资源分离简单的概括是：动态文件与静态文件的分离。

### 🦋 为什么要做动、静分离？

在我们的软件开发中，有些请求是需要后台处理的（如：`.jsp`、`.do` 等等），有些请求是不需要经过后台处理的（如：`css`、`html`、`jpg`、`js` 等等文件），这些不需要经过后台处理的文件称为静态文件，否则动态文件。

因此我们后台处理忽略静态文件。这会有人又说那我后台忽略静态文件不就完了吗？当然这是可以的，但是这样后台的请求次数就明显增多了。在我们对资源的响应速度有要求的时候，我们应该使用这种动静分离的策略去解决动、静分离将网站静态资源（HTML、JavaScript、CSS、img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问

这里我们将静态资源放到 Nginx 中，动态资源转发到 Tomcat 服务器中去。

🐱 当然，因为现在七牛、阿里云等 CDN 服务已经很成熟，主流的做法，是把静态资源缓存到 CDN 服务中，从而提升访问速度。

- 相比本地的 Nginx 来说，CDN 服务器由于在国内有更多的节点，可以实现用户的就近访问。
- 并且，CDN 服务可以提供更大的带宽，不像我们自己的应用服务，提供的带宽是有限的。

### 🦋 什么叫 CDN 服务？

CDN，即内容分发网络。

其目的是，通过在现有的 Internet 中 增加一层新的网络架构，将网站的内容发布到最接近用户的网络边缘，使用户可就近取得所需的内容，提高用户访问网站的速度。

一般来说，因为现在 CDN 服务比较大众，所以基本所有公司都会使用 CDN 服务。

Nginx 有哪些负载均衡策略？

负载均衡，即是代理服务器将接收的请求均衡的分发到各服务器中。

Nginx 默认提供了 3 种负载均衡策略：

- 1、轮询（默认）`round_robin`

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器 down 掉，能自动剔除。

- 2、IP 哈希 `ip_hash`

每个请求按访问 ip 的 hash 结果分配，这样每个访客固定访问一个后端服务器，可以解决 session 共享的问题。

当然，实际场景下，一般不考虑使用 `ip_hash` 解决 session 共享。

- 3、最少连接 `least_conn`

下一个请求将被分配到活动连接数量最少的服务器

通过 Nginx 插件，我们还可以引入 `fair`、`url_hash` 等负载均衡策略。

另外，我们还可以配置每一个后端节点在负载均衡时的其它配置：

```
weight=1; # (weight 默认为1.weight越大, 负载的权重就越大)
down; # (down 表示单前的server暂时不参与负载)
backup; # (其它所有的非backup机器down或者忙的时候, 请求backup机器)
max_fails=1; # 允许请求失败的次数默认为 1 。当超过最大次数时, 返回 proxy_next_upstream 模块定义的错误
fail_timeout=30; # max_fails 次失败后, 暂停的时间
```

详细的，可以再看看 [《Nginx 官方文档 —— 使用 Nginx 作为 HTTP 负载均衡》](#) 文章。

Nginx 如何实现后端服务的健康检查？

参见 [《Nginx 负载均衡中后端节点服务器健康检查的操作梳理》](#) 文章。

- 方式一，利用 nginx 自带模块 `ngx_http_proxy_module` 和 `ngx_http_upstream_module` 对后端节点做健康检查。
- 方式二，利用 `nginx_upstream_check_module` 模块对后端节点做健康检查。

推荐使用。

Nginx 如何开启压缩？

参见 [《Nginx 开启 gzip 压缩的配置详解和压缩效果对比》](#) 文章。

666. 彩蛋

对于开发来说，Nginx 我们主要掌握会用，所以实际面试场景下，问的也不会太深。

参考与推荐如下文章：

- [《Nginx 面试中最常见的 18 道题 —— 抱佛脚必备》](#)
- [《Nginx 常见面试题》](#)
- [《Nginx 负载均衡策略》](#)
- [《Nginx 多进程模型是如何实现高并发的？》](#)