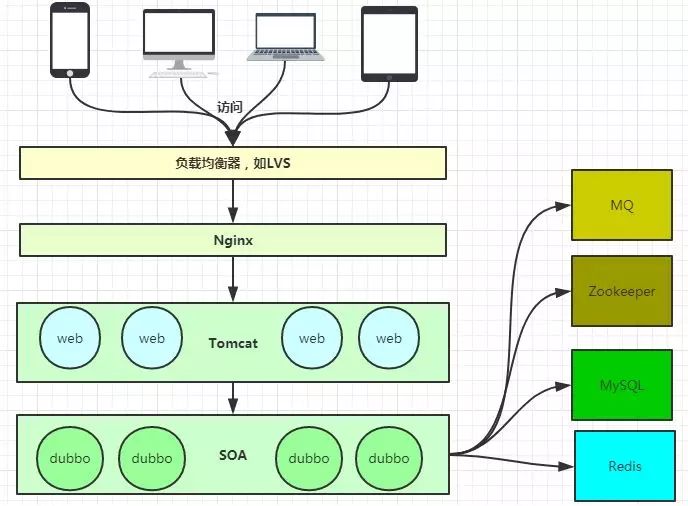
# nginx常用知识总结

Nginx是一款**轻量级的Web服务器、反向代理服务器**，由于它的内存占用少，启动极快，高并发能力强，在互联网项目中广泛应用。



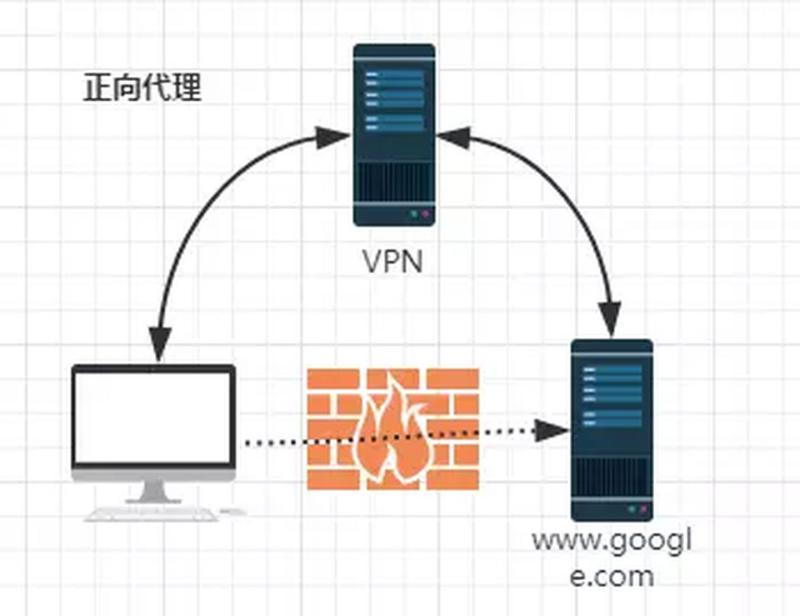
架构图

上图基本上说明了当下流行的技术架构，其中Nginx有点入口网关的味道。

## **反向代理服务器？**

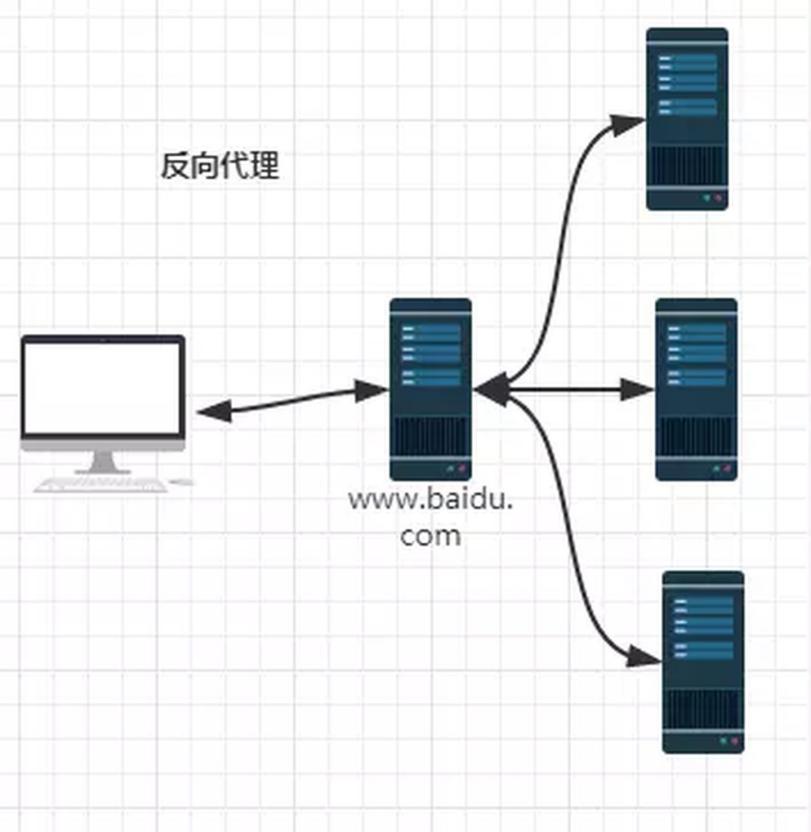
经常听人说到一些术语，如反向代理，那么什么是反向代理，什么又是正向代理呢？

**正向代理：**



正向代理示意图

**反向代理：**



反向代理示意图

由于防火墙的原因，**我们并不能直接访问谷歌，那么我们可以借助VPN来实现，这就是一个简单的正向代理的例子**。这里你能够发现，正向代理“代理”的是客户端，而且客户端是知道目标的，而目标是不知道客户端是通过VPN访问的。

当我们在外网访问百度的时候，其实会进行一个转发，代理到内网去，这就是所谓的反向代理，**即反向代理“代理”的是服务器端，而且这一个过程对于客户端而言是透明的，即客户端不需要知道内网去了哪里处理，反正只知道访问了百度就行。**

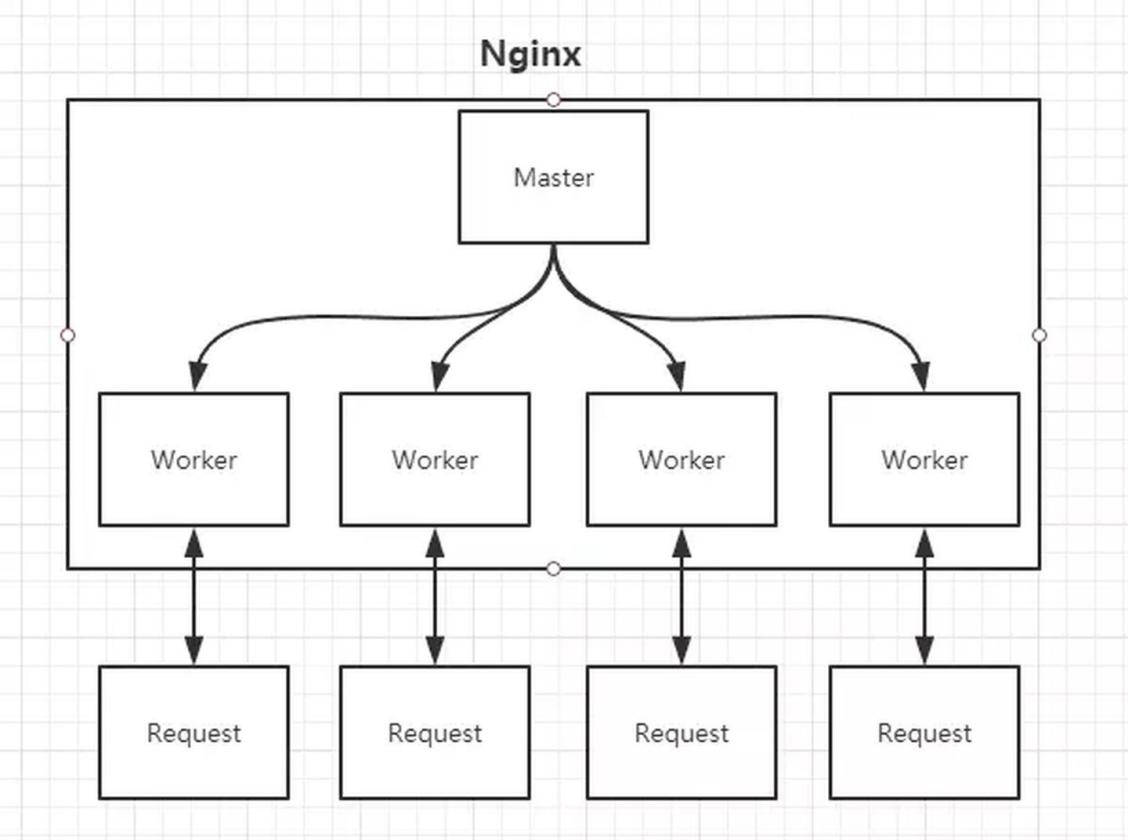
##### 反向代理的作用

1. 保障应用服务器的安全（增加一层代理，可以屏蔽危险攻击，更方便的控制权限）
2. 实现负载均衡
3. 实现跨域（号称是最简单的跨域方式）

## **Nginx的Master-Worker模式**

nginx进程

启动Nginx后，其实就是在80端口启动了Socket服务进行监听，如图所示，**Nginx涉及Master进程和Worker进程**。



Master-Worker模式

Master进程的作用是？

**读取并验证配置文件nginx.conf；管理worker进程；**

Worker进程的作用是？

**每一个Worker进程都维护一个线程（避免线程切换），处理连接和请求；注意Worker进程的个数由配置文件决定，一般和CPU个数相关（有利于进程切换），配置几个就有几个Worker进程。**

## **思考：Nginx如何做到热部署？**

所谓热部署，就是配置文件nginx.conf修改后，不需要重启Nginx，不需要中断请求，就能让配置文件生效！

通过上文我们已经知道worker进程负责处理具体的请求，为达到热部署的效果，nginx的方案是：

修改配置文件nginx.conf后，重新生成新的worker进程，当然会以新的配置进行处理请求，而且新的请求必须都交给新的worker进程，至于老的worker进程，等把那些以前的请求处理完毕后，kill掉即可。

## **思考：Nginx如何做到高并发下的高效处理？**

上文已经提及Nginx的worker进程个数与CPU绑定、worker进程内部包含一个线程高效回环处理请求，这的确有助于效率，但这是不够的。

**作为专业的程序员，我们可以开一下脑洞：BIO/NIO/AIO、异步/同步、阻塞/非阻塞...**

要同时处理那么多的请求，要知道，有的请求需要发生IO，可能需要很长时间，如果等着它，就会拖慢worker的处理速度。

**Nginx采用了Linux的epoll模型，epoll模型基于事件驱动机制，它可以监控多个事件是否准备完毕，如果OK，那么放入epoll队列中，这个过程是异步的。worker只需要从epoll队列循环处理即可。**

## **思考：Nginx挂了怎么办？**

Nginx既然作为入口网关，很重要，如果出现单点问题，显然是不可接受的。

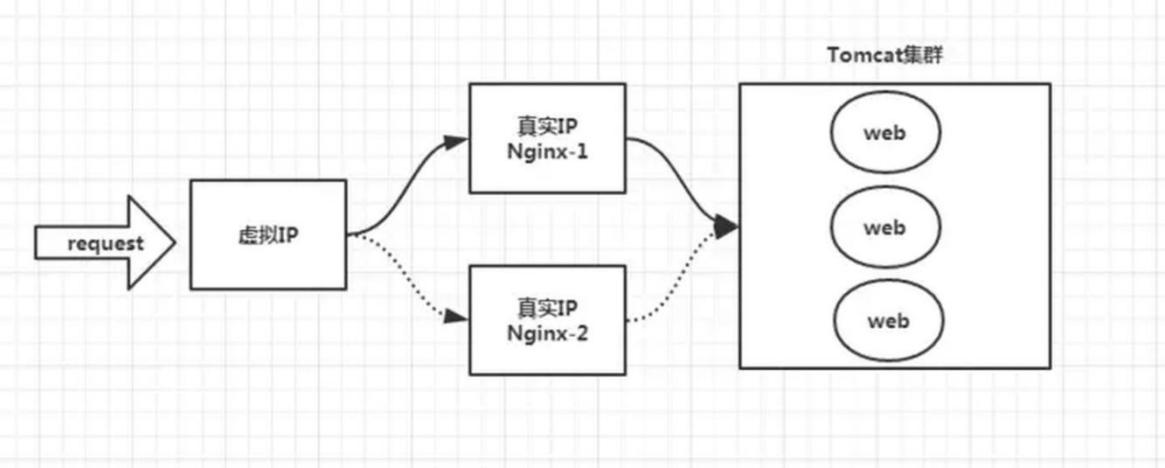
答案是：**Keepalived+Nginx实现高可用**。

**Keepalived是一个高可用解决方案**，主要是用来防止服务器单点发生故障，可以通过和Nginx配合来实现Web服务的高可用。（其实，Keepalived不仅仅可以和Nginx配合，还可以和很多其他服务配合）

Keepalived+Nginx实现高可用的思路：

第一：请求不要直接打到Nginx上，应该先通过Keepalived（这就是所谓虚拟IP，VIP）

第二：Keepalived应该能监控Nginx的生命状态（提供一个用户自定义的脚本，定期检查Nginx进程状态，进行权重变化,，从而实现Nginx故障切换）



Keepalived+Nginx

## **我们的主战场：nginx.conf**

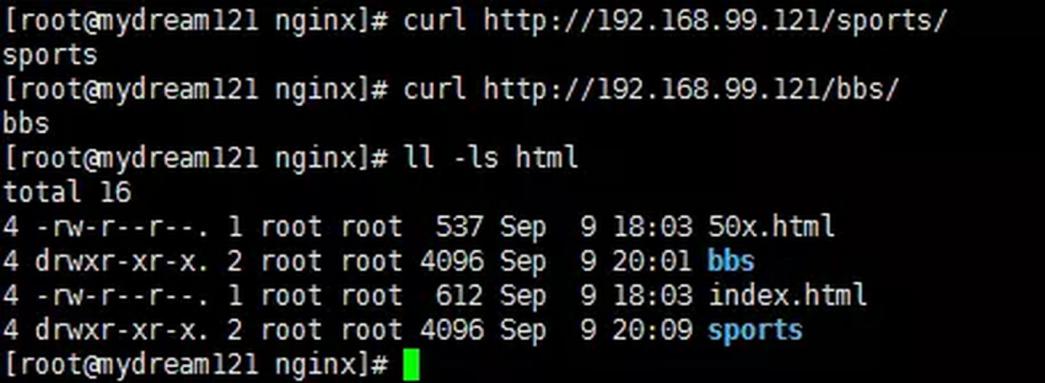
很多时候，在开发、测试环境下，我们都得自己去配置Nginx，**就是去配置nginx.conf**。

nginx.conf是典型的分段配置文件，下面我们来分析下。

## **虚拟主机**



http的server端



访问结果

其实这是把Nginx作为web server来处理静态资源。

第一：location可以进行正则匹配，应该注意正则的几种形式以及优先级。

第二：Nginx能够提高速度的其中一个特性就是：动静分离，就是把静态资源放到Nginx上，由Nginx管理，动态请求转发给后端。

**第三：我们可以在Nginx下把静态资源、日志文件归属到不同域名下（也即是目录），这样方便管理维护。**

**第四：Nginx可以进行IP访问控制，有些电商平台，就可以在Nginx这一层，做一下处理，内置一个黑名单模块，那么就不必等请求通过Nginx达到后端在进行拦截，而是直接在Nginx这一层就处理掉。**

## **反向代理【proxy\_pass】**

所谓反向代理，很简单，其实就是在location这一段配置中的root替换成**proxy\_pass**即可。root说明是静态资源，可以由Nginx进行返回；而proxy\_pass说明是动态请求，需要进行转发，比如代理到Tomcat上。

反向代理，上面已经说了，过程是透明的，比如说request -> Nginx -> Tomcat，那么对于Tomcat而言，请求的IP地址就是Nginx的地址，而非真实的request地址，这一点需要注意。不过好在Nginx不仅仅可以反向代理请求，还可以由用户**自定义设置HTTP HEADER**。

## **负载均衡【upstream】**

上面的反向代理中，我们通过proxy\_pass来指定Tomcat的地址，很显然我们只能指定一台Tomcat地址，那么我们如果想指定多台来达到负载均衡呢？

第一，通过**upstream**来定义一组Tomcat，并指定负载策略（IPHASH、加权论调、最少连接），健康检查策略（Nginx可以监控这一组Tomcat的状态）等。

第二，将proxy\_pass替换成upstream指定的值即可。

**负载均衡可能带来的问题？**

负载均衡所带来的明显的问题是，一个请求，可以到A server，也可以到B server，这完全不受我们的控制，当然这也不是什么问题，只是我们得注意的是：**用户状态的保存问题，如Session会话信息，不能再保存到服务器上，而应该保存到某个公共集群中，比如redis集群。**

## **缓存**

缓存，是Nginx提供的，可以加快访问速度的机制，说白了，在配置上就是一个开启，同时指定目录，让缓存可以存储到磁盘上。具体配置，大家可以参考Nginx官方文档，这里就不在展开了。

例1：静态资源和反向代理

**server** {

listen 8080;

server\_name localhost;

location / {

root html; *# Nginx默认值* index index.html index.htm;

} *# 静态化配置，所有静态请求都转发给 nginx 处理，存放目录为 my-project*

location ~ .\*\.(html|htm|gif|jpg|jpeg|bmp|png|ico|js|css)$ {

root /usr/local/var/www/my-project; *# 静态请求所代理到的根目录*

} *# 动态请求匹配到path为'node'的就转发到8002端口处理*

location /node/ {

proxy\_pass http://localhost:8002; *# 充当服务代理*

}

}

访问静态资源 nginx 服务器会返回 my-project 里面的文件，如获取 index.html： <http://localhost:8080/index.html>

而访问/node/，就会请求到8002端口去。

例2：负载均衡

*# 负载均衡：设置domain*

upstream domain {

server localhost:8000;

server localhost:8001;

}

**server** {

listen 8080;

server\_name localhost;

location / {

*# root html; # Nginx默认值*

*# index index.html index.htm;*

proxy\_pass http://domain; *# 负载均衡配置，请求会被平均分配到8000和8001端口* proxy\_set\_header Host $host:$server\_port;

}

}

例3：过滤

server {

listen 8080;

server\_name localhost;

location / {

# IP访问限制（只允许IP是 10.81.1.11 的机器才能访问）

allow 10.81.1.11;

deny all;

root html;

index index.html index.htm;

}

}

当你的项目没有灰度环境，又想在功能上线后先让测试同事试用一下的时候，就需要设置访问的白名单了。

例4：图片防盗链

server {

listen 8080;

server\_name localhost;

location / {

root /usr/local/var/www/my-project; # 设置为个人项目的根目录路径 index index.html index.htm;

}

# 图片防盗链

location ~\* \.(gif|jpg|jpeg|png|bmp|swf)$ {

valid\_referers none blocked 192.168.0.103;#只允许本机IP外链引用

if ($invalid\_referer){

return 403;

}

}

}

不允许别的网站以外链的方式进行引用图片，其他域名下的请求都会被403禁止访问。

### 例5：解决跨域

跨域是前端经常会遇到的问题，解决的方式有很多，例如：jsonp、node.js中转、CORS等。

但是使用 Nginx 来跨域简单明了，主要用到的是 Nginx 的反向代理原理。

server {

listen 8080;

server\_name localhost;

location / {

# 跨域代理设置

proxy\_pass http://www.proxy.com;

# 要实现跨域的域名

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Methods 'GET, POST, OPTIONS'; add\_header Access-Control-Allow-Headers 'DNT,X-Mx-ReqToken,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Authorization';

}

}

思路就是在反向代理请求过程中，增加可以跨域访问的请求头。

例6：适配PC端和移动端

server {

listen 8080;

server\_name localhost;

location / {

# 适配移动端/PC端配置

set $type "pc";

if ($http\_user\_agent ~\* (mobile|nokia|iphone|ipad|android|samsung|htc|blackberry)) {

set $type "mobile";

}

root /usr/local/var/www/my-project/$type;

# 根据设备类型选择设定根目录文件夹名（pc/mobile）

index index.html index.htm;

}

}

主要思路也是通过判断浏览器的 useragent 来取对应的静态页面资源。如果我们的前端应用部署后是有服务的怎么办呢？那就不取对应的静态页面，而是 proxy\_pass 代理到对应的页面路由上即可。

例7：开启gzip

http {

# 配置gzip压缩

gzip on;

gzip\_min\_length 1000; # 设定压缩的临界点

gzip\_comp\_level 3; # 压缩级别

gzip\_types text/plain application/xml; # 要压缩的文件类别

}

能看到响应头带上了Content-Encoding: gzip 标识，就说明你配置的gzip压缩功能起效果了。

例8：合并请求

web前端性能优化中很重要的一条就是减少 http 请求数。而通过淘宝开发的 nginx-http-concat 第三方模块，我们可以实现多请求的合并。

配置好 Nginx 的合并请求功能后，前端就可以用一种特殊的 url 请求规则（例如example.com/??1.js,2.js,3.js ）向 Nginx 发起请求。此时，Nginx 会将前端想要的多个资源请求合并成一个请求返回给前端，极大的减少了网络请求时间的开销。

server {

......

# 新增一个 location，static 为静态资源目录

location /static/ {

concat on; # 是否打开资源合并开关

concat\_types application/javascript; # 允许合并的资源类型

concat\_unique off; # 是否允许合并不同类型的资源

concat\_max\_files 20; # 允许合并的最大资源数目

}

}