普通的，监听端口是80的，域名是这个域名的请求，location后面配置的是/ 也就是所有的请求都会到下面这个locatin里，走的前端的资源是root这个路径里面的，index就是首页的名称。

Nginx作为反向代理的时候一个用户会占用两个连接，一个是client到nignx，一个是nginx到java服务器。

server {   
listen 80;   
server\_name a.test.com;   
  
location / {   
index index.html;   
root /home/www/host\_a/;   
}   
}

这个是一个新的，index和root可以弄到location外面。

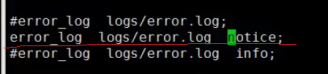
server {   
listen 80;   
server\_name b.test.com;   
  
index index.html;   
root /home/www/b/;   
  
#禁止对self目录的访问  
location /(self)/ {   
deny all;   
}   
}

可以指定一个新的location 设置对这个server路径的一些控制，这里就是设置了对这个路径不能访问 deny all。

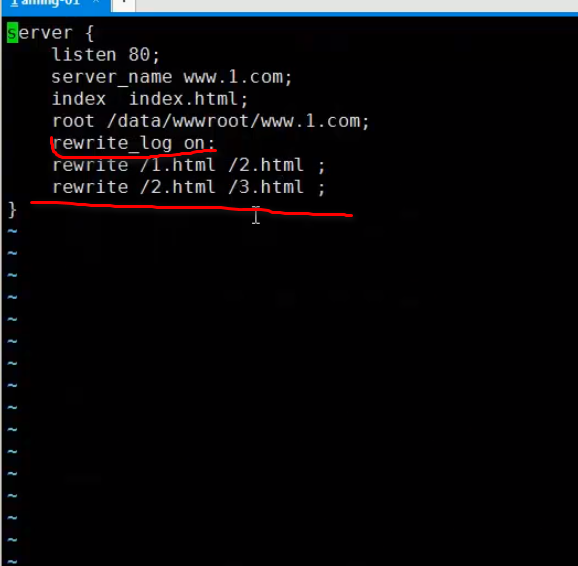
这个是在一个server里面 通过多个location，指定对不同路径的访问进入到不同的资源，

但是还是同一个server。

server {  
listen 80;  
  
#使用正则格式，这里表示接受任何ip  
server\_name ~^\d+\.\d+\.\d+\.\d+$;  
  
index index.html index.htm;  
root /home/lg/www/;  
  
location /share {  
root /home/lg/Downloads;  
}  
  
location ^~ /Videos {  
root /home/lg/;  
autoindex on;  
autoindex\_exact\_size on;  
autoindex\_localtime on;  
allow all;  
}  
  
location ^~ /html5 {  
root /home/lg/workspace/nodejs/;  
index index.html index.htm;  
}  
  
location = /404.html {  
root /usr/share/nginx/html;  
}  
}

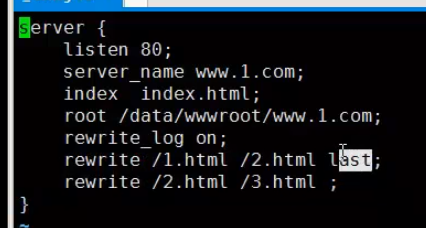


在nginx.conf的error\_LOG里面就可以对nginx的日志进行配置是否开启。



上面就是在一个server里配置了一个rewrite，rewrite的log配置为了on 上面的配置的结果就是你访问1.html会转到2.html最终会转到3.html。

Rewrite就是转发的意思,在nginx里配置的转发



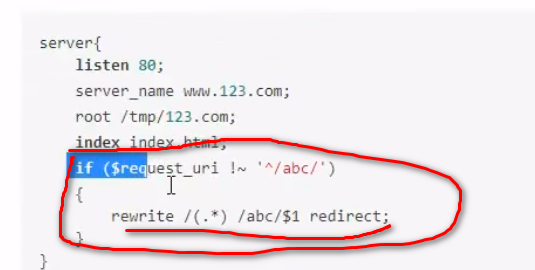
如果你在这里加上了一个last或者break就只会跳到2，不会由2跳到3了，如果只有一个location是这样的，如果是有多个location，每个location都有rewrite语句的话就是遇到break就彻底不执行了，其他location也不会执行，如果遇到的是last，本location内停了却又会从小到下把所有的location的rewrite走一遍。

还有个if和return 这咋看起来和java的代码似的。

就是if 可以判断url的内容啊 请求方式啊 浏览器信息啊，也就是请求头里的东西来做操作 if里面可以写rewrite转发或者return直接满足if条件了返回404 500啥的。



上面就是if rewrite和return的一些例子。

、

这个东西表示的是rewrite也是可以匹配转换的，也就是正则重定向，把url符合这个正则的转换到另一个地方。

Nginx的变量



就是一些http的一些参数，可以在ngnix的内置的变量里直接拿到，$host就拿到了主机名，可以在if语句里或者return的内容里使用。

上面的意思就是nginx作为一款服务器软件,也是可以像mysql那样获得到http的头啊,url啥的,并可以根据这些信息弄出if return啥的操作.

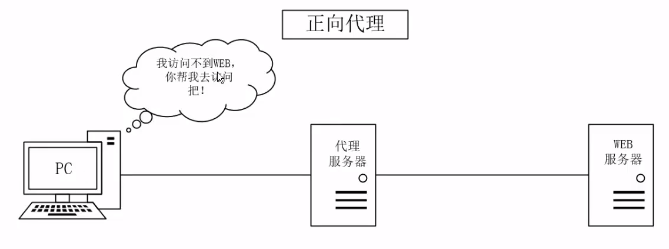


配置locatin也是可以有通配符啥的，而且如果有匹配到多个location是有优先级的。

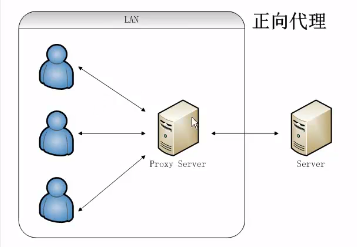
负载均衡和反向代理是一个意思。

Ng也有正向代理功能

所谓正向代理就是



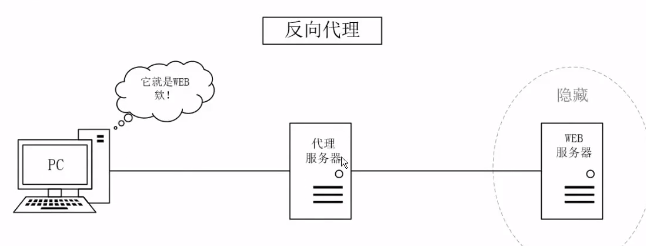
用户知道自己访问的是代理服务器，由于用户连接不到web就委托代理服务器去访问，



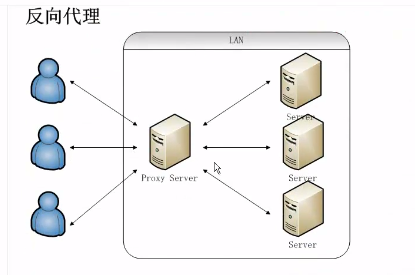
正向代理的服务器是和用户是一体的，他代理了用户。

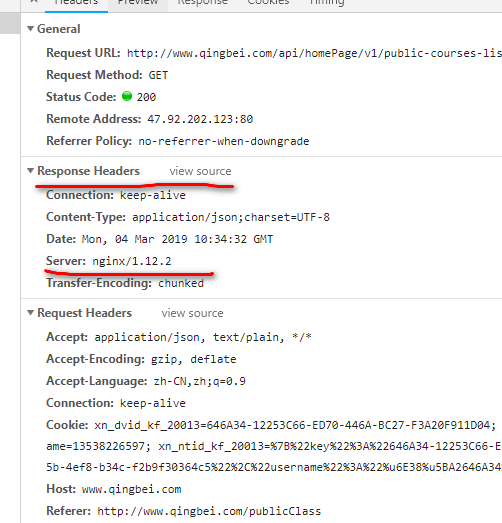
反向代理

在用户眼里反向代理服务器就是真的服务器用户是感受不到web服务器的



反向代理服务器一般都和web服务器在一个内网中，反向代理服务器是代理了web服务器，而正向代理服务器是代理了用户。

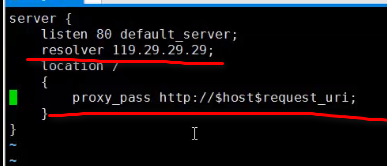




可见responseheader里面有个server指示了服务端用的服务器是啥。

Server配置里面的

Listen 80 default的意思就是如果所有的server都匹配不到这个域名的话就会走这个server。



这个是正向代理，比如一个机房只有一个机器连着外网，别的机器都没有连，大家想连外网就通过这个机器，这个机器安装一个nginx做为正向代理，其他电脑访问这个服务器，这个服务器在吧这个服务代理到外面，然后把返回的内容给其他电脑。

配置如上图，default server的作用明白了吧，哪个resolver配置的是个dns服务器。

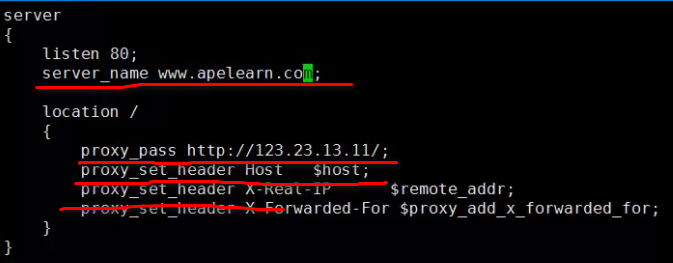
Location表示所有的路径都会转发的下面的这个 $hots$request\_uri是nginx自带的全局变量。

比如b的域名是www.apelearm.com 外界访问b，b里装了nginx，然后b把请求代理到c，c的ip是123.23.13.11 配置就上面的配置，servername指定了我代理的是这个域名，下面的location里的proxy\_pass指定了要代理到源代码存在的哪个服务器。

但是 这样的话就会涉及到两个http消息，一个是client发给代理的还有一个是代理发给

，由于是不同的两个节点发的，http的一些头肯定会不一样，看下面的配置

Proxy\_set\_header，就是代理这里根据client的信息设置一些头，第一个设置host，就是设置client想要访问的域名，因为你的proxy\_pass那里只有ip嘛，proxy这里要把域名也设置上，真正的java服务器才能知道client想要的域名是这个。







Client和代理和最终服务器之间的location proxy\_pass弄过后的url对应关系如下。

上面哪个proxy\_set\_header host会让最终的java进程收到的请求的url是www.aliqingbei.com/xxx 是这样的带域名的，如果不弄的话就不会带域名。

上面的例子还有两个proxy\_set\_header realip啥的，有这两个的话最终的java服务器可以看到client的ip而不是只能看到proxy服务器的ip。

nginx的proxy\_redirect指定修改被代理**[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/u010391029/article/details/_blank)**返回的响应头中的location头域跟refresh头域数值。

server {

       listen       80;

       server\_name  www.boke.com;

       location / {

            proxy\_pass http://192.168.1.154:8080;

       }

 }

这样配置的话如果服务器让用户重定向到服务器的另一个url，比如重定向到

<http://www.boke.com/women> 但是由于tomcat发的重定向请求在浏览器里会暴露tomcat的信息。

HTTP/1.1 301 Moved Permanently

Server: nginx

Date: Thu, 24 Dec 2015 12:02:00 GMT

Content-Type: text/html; charset=iso-8859-1

Connection: keep-alive

Location: http://192.168.1.154:8080/wuman/

暴露了

怎么办呢，我们可以这样。

server {

       listen       80;

       server\_name  www.boke.com;

       location / {

            proxy\_pass http://192.168.1.154:8080;

proxy\_redirect http://192.168.1.154:8080/wuman/  http://www.boke.com/wuman/;

       }

加个这样的配置就好了。

如果使用“default”参数，将根据location和proxy\_pass参数的设置来决定。   
例如下列两个配置等效：   
  
location /one/ {  proxy\_pass       http://upstream:port/two/;  proxy\_redirect   default;} location /one/ {  proxy\_pass       http://upstream:port/two/;  proxy\_redirect   http://upstream:port/two/   /one/;}

好像又给重定向回来了。

Nginx也是有buff的,就是resp并不会马上发回去,而是先存着,等够量了再发回去,但是这样呢,交互性不是很好哈

注意nginx的buff缓存是java服务器返回的resp，而且是每个req都是有buff的。

Nignx\_buffer ，即缓冲区，它在 Nginx 上发挥的作用就是 启用一个缓冲区，先在这个缓冲区内进行存储，再把数据发送出去 。和在线观看视频有点类似，先把视频文件缓冲一部分到本地再开始播放。

若没有 buffer，数据将会直接从 Nginx 传输到客户端。假设如果客户端的加载速度足够快，你可以直接把 buffer 关掉，让数据尽可能快地到达客户端。

而使用 buffer，Nginx 将会临时存储后端 response 到缓冲区，然后慢慢把数据发送到客户端。启用 buffer 的好处在于可以把数据一次性地发送给目标，相较于即时传输可以节约出这部分带宽。

顺带一提，Nginx 全局配置中的 tcp\_nopush 的作用就是 数据包会累计到一定大小之后才会发送 。而 tcp\_nodelay 是尽快发送数据，所以若你启用了 buffer，建议关闭 tcp\_nodelay。

这使得一台Nginx可以同时作为“负载均衡服务器”与“Web缓存服务器”来使用

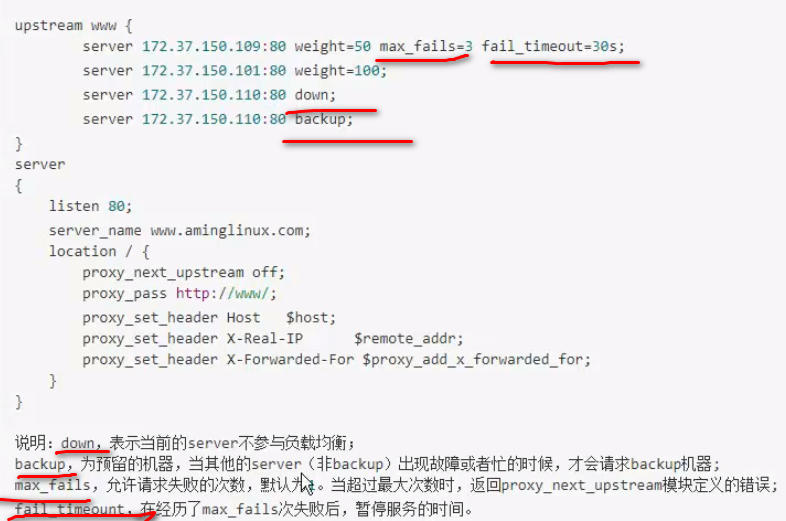
Nginx也可以开启缓存，对内容resp进行缓存，如果你频繁访问的话就会走缓存。



配置负载均衡也就是把消息代理到多个服务器，但是server还是只配置一个，locaiton还是和以前一样，但是location里的proxy\_pass的内容会关联到一个upstream，这个upstream里会配置多个server的ip。



带了权重，想要保证同一个用户的请求发到相同的服务器只需要在upstream加上ip\_hash这几个字就可以了。也就是把用户ip和哪个服务器绑定了，解决了session共享问题。

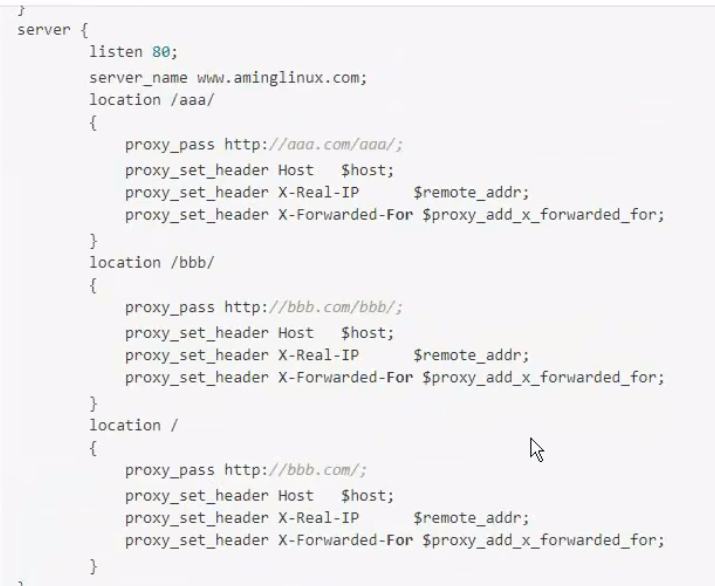


proxy\_next\_upstream http\_500 | http\_502 | http\_503 | http\_504 |http\_404;

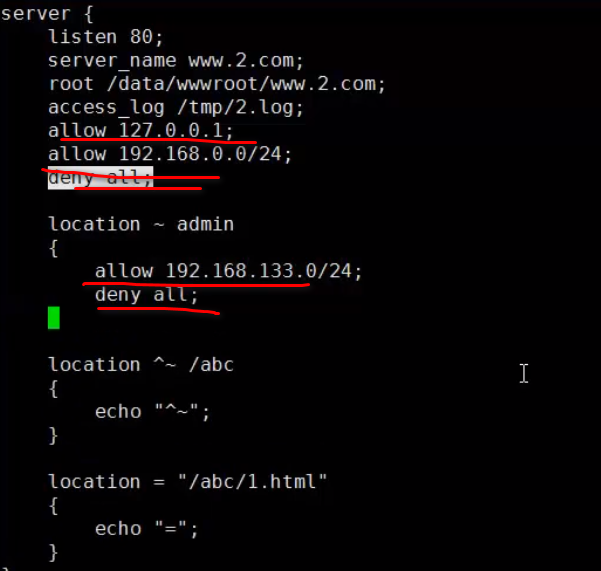
当其中一台返回错误码404,500...等错误时，可以分配到下一台服务器程序继续处理，提高平台访问成功率，多可运用于前台程序负载，设置。proxy\_next\_upstream的作用就是定义了发送了什么错误是让下一个服务器来重新弄下，也可以关了，避免消息一直发。



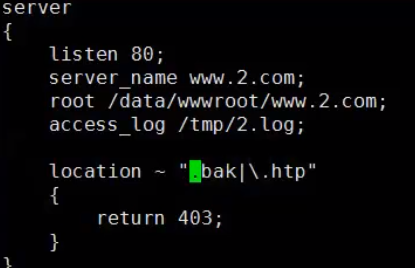
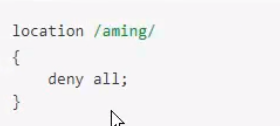
这个就是一个server对应多个location，每个location对应不同的路径，每个location关联一个upstream。



在server和location可以对单独ip或者网段啥的用allow和deny命令限制访问。

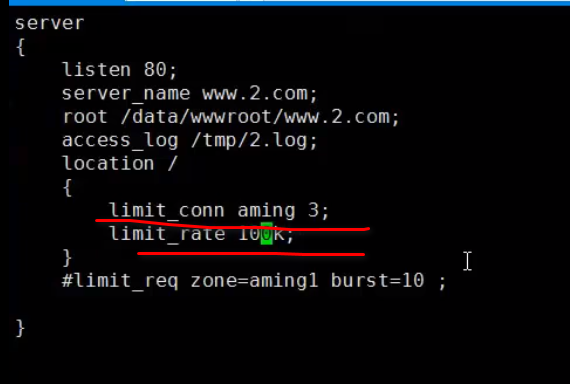


基于location的访问控制。



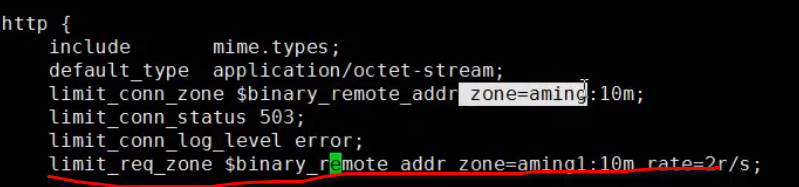
反正nginx里面有很多的全局变量，documenturi啊request\_url啊,user\_agent啊，http\_referer啊，我们可以根据这些东东做访问控制。

很多配置都是既可以配置再location里也可以配置再server里。



上图的配置可以放在server也可以弄在location里，一个是限制连接数，一个是限制下载速度，比如文件下载，如果你限制成10k，那么无论你多块的网速在这里下载一个文件也是最多10k的速度。

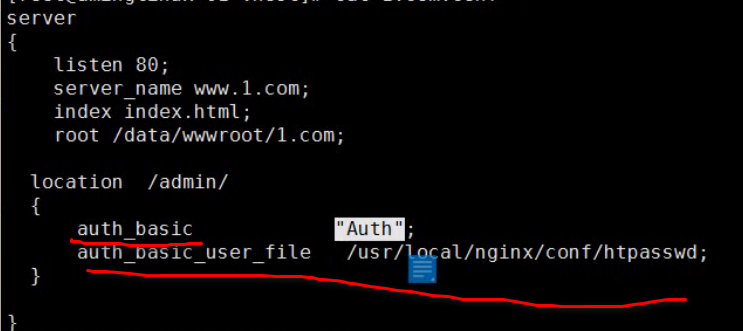
这里的连接数是tcp连接



也可以对请求进行限制

2019-03-09_201709

Burst指的是同时处理多少请求，这里是500，如果同时来了600个那么就会有100个报错503，因为后面是nodelay，如果没有nodelay，那么来了600个就会有100个在后面等待而不是报错。

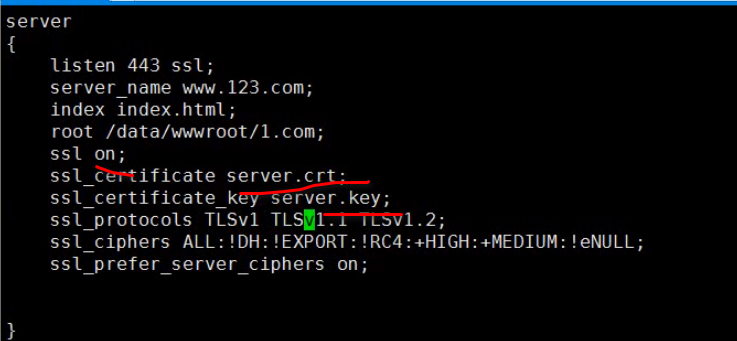


Nginx自带一个认证功能，可以让你对一个地方设置认证机制，比如这里我们是对这个url的location设置了认证，第二个参数是你的账号和密码文件的路径，这个文件需要httppassword工具生成，配置完成后，你再访问这个url的时候就会弹出下图的东西。



CA是个机构，用来签发CA证书的。

我们见到了ssl都是单向ssl，就是只有服务端有证书，客户端验证证书，双向的就是客户端和服务端都要验证对方的证书。



通知的sever.crt和server.key是俩文件，路径要指定正确才可以的。

https的端口是443，http的是80，要想都支持就弄两个server就行。

网页访问nginx的https在nginx配置就可以了。

需要配置.crt和.key的路径。



### Nignx负载均衡健康检测

被动检测

http {  
 upstream onmpw {  
 server 192.168.144.128;  
 server 192.168.144.132 max\_fails=3 fail\_timeout=30s;  
 server 192.168.144.131 max\_fails=2;  
 }  
 server {  
 listen 80;  
 location / {  
 proxy\_pass http://onmpw;  
 132结尾的ip,意思就是当失败次数到达3次后,30s内不会再向这个服务器转发请求了.

131结尾的ip的意思就是失败次数到了2次后,默认10s后不会再向这个服务器转发请求.

主动检测

由nginx定期的向每台应用服务器发送特殊的请求,来监测应用服务器是否可以正常访问.这种方式称为主动监测.需要指出的是,主动检测指令health\_check命令只有在商业版本的nginx才提供.

为了实现主动监测这种方式，我们需要在Nginx负载均衡的配置文件中加入health\_check指令。除此之外，我们还需要在设置应用服务器信息的组里加入zone指令。

http {  
 upstream onmpw {  
 zone onmpw 64k;  
 server 192.168.144.128;  
 server 192.168.144.132;  
 server 192.168.144.131;  
 }  
 server {  
 listen 80;  
 location / {  
 proxy\_pass http://onmpw;  
 health\_check;  
 }  
 }  
 }

在这里我们设置了一组应用服务器。通过一个单一的location，将所有的请求都分发到这组应用服务器上。在这种情况下，每隔5s Nginx Plus就会向每一台应用服务器发送’/’请求。任何一台应用服务器连接错误或者响应超时亦或者是被代理的服务器响应了一个状态码2xx或者是3xx，health\_check机制就会认为是失败的。对于任何一台应用服务器，如果health\_check失败，则就会被认为是不稳定的。那么Nginx Plus就不再向这台应用服务器分发访问请求。

zone指令定义了一块儿内存空间。这块儿空间存储在各个工作进程中共享的运行环境的状态和应用服务器组的配置信息。这块儿空间应该根据实际情况尽量申请的大一些，要保证能存下这些信息。

下面我们再看这样的一个例子

例三

location / {  
 proxy\_pass http://onmpw;  
 health\_check interval=10 fails=3 passes=2;  
 }

在上面的例三中，interval=10表示两次进行health\_check的间隔为10s，如果不设置默认两次的间隔是5s。fails=3表示一台应用服务器如果请求失败次数达到3次，则该应用服务器被认为不能访问。最后是passes=2表示，被认定为不能访问的服务器需要再次进行两次health\_check 以后才会再次被认为是可以正常访问的。

在health\_check中，我们可以指定请求的url。

例四

location / {  
 proxy\_pass http://onmpw;  
 health\_check uri=/some/path;  
 }

对于onmpw组中的第一台应用服务器128来说，一次health check请求的url是http://192.168.144.128/some/path。

上面两种监测方式是普遍被使用的，希望本文对大家有所帮助。