https://www.w3cschool.cn/nginx/nginx-d1aw28wa.html



Nginx是一款[轻量级](https://baike.baidu.com/item/%E8%BD%BB%E9%87%8F%E7%BA%A7/10002835" \t "_blank)的[Web](https://baike.baidu.com/item/Web/150564) 服务器/[反向代理](https://baike.baidu.com/item/%E5%8F%8D%E5%90%91%E4%BB%A3%E7%90%86/7793488)服务器及[电子邮件](https://baike.baidu.com/item/%E7%94%B5%E5%AD%90%E9%82%AE%E4%BB%B6/111106)（IMAP/POP3）代理服务器，并在一个BSD-like 协议下发行。其特点是占有内存少，[并发](https://baike.baidu.com/item/%E5%B9%B6%E5%8F%91/11024806)能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，

### 服务器

Nginx作为[负载均衡服务器](https://baike.baidu.com/item/%E8%B4%9F%E8%BD%BD%E5%9D%87%E8%A1%A1%E6%9C%8D%E5%8A%A1%E5%99%A8" \t "_blank)：Nginx 既可以在内部直接支持 Rails 和 PHP 程序对外进行服务，也可以支持作为 HTTP[代理服务器](https://baike.baidu.com/item/%E4%BB%A3%E7%90%86%E6%9C%8D%E5%8A%A1%E5%99%A8)对外进行服务。Nginx采用C进行编写，不论是系统资源开销还是CPU使用效率都比 Perlbal 要好很多。

处理静态文件，索引文件以及自动索引;打开文件描述符缓冲。

无缓存的反向代理加速，简单的负载均衡和容错。

**代码**

Nginx代码完全用[C语言](https://baike.baidu.com/item/C%E8%AF%AD%E8%A8%80" \t "_blank)从头写成，已经移植到许多体系结构和[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F/192)，包括：[Linux](https://baike.baidu.com/item/Linux)、[FreeBSD](https://baike.baidu.com/item/FreeBSD" \t "_blank)、[Solaris](https://baike.baidu.com/item/Solaris/3517" \t "_blank)、[Mac OS X](https://baike.baidu.com/item/Mac%20OS%20X" \t "_blank)、[AIX](https://baike.baidu.com/item/AIX" \t "_blank)以及[Microsoft Windows](https://baike.baidu.com/item/Microsoft%20Windows" \t "_blank)。

**代理服务器**

作为邮件[代理服务器](https://baike.baidu.com/item/%E4%BB%A3%E7%90%86%E6%9C%8D%E5%8A%A1%E5%99%A8)：Nginx 同时也是一个非常优秀的邮件[代理服务器](https://baike.baidu.com/item/%E4%BB%A3%E7%90%86%E6%9C%8D%E5%8A%A1%E5%99%A8" \t "_blank)（最早开发这个产品的目的之一也是作为邮件代理服务器），Last.fm 描述了成功并且美妙的使用经验

Nginx使用什么算法来实现[负载均衡](https://baike.baidu.com/item/%E8%B4%9F%E8%BD%BD%E5%9D%87%E8%A1%A1" \t "_blank)它能实现基于连接数的负载均衡吗?

Nginx使用简单的轮巡算法，所以无法做基本链接计数的负载均衡。这个可能会在将来的版本中有所改变。

### ######Nginx配置文件nginx.conf中文详解#####

#定义Nginx运行的用户和用户组

user www www;

#nginx进程数，建议设置为等于CPU总核心数。

worker\_processes 8;

#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]

error\_log /usr/local/nginx/logs/error.log info;

#进程pid文件

pid /usr/local/nginx/logs/nginx.pid;

#指定进程可以打开的最大描述符：数目

#工作模式与连接数上限

#这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。

#现在在linux 2.6内核下开启文件打开数为65535，worker\_rlimit\_nofile就相应应该填写65535。

#这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。

worker\_rlimit\_nofile 65535;

events

{

#参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型

#是Linux 2.6以上版本内核中的高性能网络I/O模型，linux建议epoll，如果跑在FreeBSD上面，就用kqueue模型。

#补充说明：

#与apache相类，nginx针对不同的操作系统，有不同的事件模型

#A）标准事件模型

#Select、poll属于标准事件模型，如果当前系统不存在更有效的方法，nginx会选择select或poll

#B）高效事件模型

#Kqueue：使用于FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X.使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。

#Epoll：使用于Linux内核2.6版本及以后的系统。

#/dev/poll：使用于Solaris 7 11/99+，HP/UX 11.22+ (eventport)，IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。

#Eventport：使用于Solaris 10。 为了防止出现内核崩溃的问题， 有必要安装安全补丁。

use epoll;

#单个进程最大连接数（最大连接数=连接数\*进程数）

#根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把cpu跑到100%就行。每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为。

worker\_connections 65535;

#keepalive超时时间。

keepalive\_timeout 60;

#客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。

#分页大小可以用命令getconf PAGESIZE 取得。

#[root@web001 ~]# getconf PAGESIZE

#4096

#但也有client\_header\_buffer\_size超过4k的情况，但是client\_header\_buffer\_size该值必须设置为“系统分页大小”的整倍数。

client\_header\_buffer\_size 4k;

#这个将为打开文件指定缓存，默认是没有启用的，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没被请求后删除缓存。

open\_file\_cache max=65535 inactive=60s;

#这个是指多长时间检查一次缓存的有效信息。

#语法:open\_file\_cache\_valid time 默认值:open\_file\_cache\_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open\_file\_cache中缓存项目的有效信息.

open\_file\_cache\_valid 80s;

#open\_file\_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，它将被移除。

#语法:open\_file\_cache\_min\_uses number 默认值:open\_file\_cache\_min\_uses 1 使用字段:http, server, location 这个指令指定了在open\_file\_cache指令无效的参数中一定的时间范围内可以使用的最小文件数,如果使用更大的值,文件描述符在cache中总是打开状态.

open\_file\_cache\_min\_uses 1;

#语法:open\_file\_cache\_errors on | off 默认值:open\_file\_cache\_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误.

open\_file\_cache\_errors on;

}

#设定http服务器，利用它的反向代理功能提供负载均衡支持

**http**

{

#文件扩展名与文件类型映射表

include mime.types;

#默认文件类型

default\_type application/octet-stream;

#默认编码

#charset utf-8;

#服务器名字的hash表大小

#保存服务器名字的hash表是由指令server\_names\_hash\_max\_size 和server\_names\_hash\_bucket\_size所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.

server\_names\_hash\_bucket\_size 128;

#客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。

client\_header\_buffer\_size 32k;

#客户请求头缓冲大小。nginx默认会用client\_header\_buffer\_size这个buffer来读取header值，如果header过大，它会使用large\_client\_header\_buffers来读取。

large\_client\_header\_buffers 4 64k;

#设定通过nginx上传文件的大小

client\_max\_body\_size 8m;

#开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。

#sendfile指令指定 nginx 是否调用sendfile 函数（zero copy 方式）来输出文件，对于普通应用，必须设为on。如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络IO处理速度，降低系统uptime。

sendfile on;

#开启目录列表访问，合适下载服务器，默认关闭。

autoindex on;

#此选项允许或禁止使用socke的TCP\_CORK的选项，此选项仅在使用sendfile的时候使用

tcp\_nopush on;

tcp\_nodelay on;

#长连接超时时间，单位是秒

keepalive\_timeout 120;

#FastCGI相关参数是为了改善网站的性能：减少资源占用，提高访问速度。下面参数看字面意思都能理解。

fastcgi\_connect\_timeout 300;

fastcgi\_send\_timeout 300;

fastcgi\_read\_timeout 300;

fastcgi\_buffer\_size 64k;

fastcgi\_buffers 4 64k;

fastcgi\_busy\_buffers\_size 128k;

fastcgi\_temp\_file\_write\_size 128k;

#gzip模块设置

gzip on; #开启gzip压缩输出

gzip\_min\_length 1k; #最小压缩文件大小

gzip\_buffers 4 16k; #压缩缓冲区

gzip\_http\_version 1.0; #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）

gzip\_comp\_level 2; #压缩等级

gzip\_types text/plain application/x-javascript text/css application/xml; #压缩类型，默认就已经包含textml，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。

gzip\_vary on;

#开启限制IP连接数的时候需要使用

#limit\_zone crawler $binary\_remote\_addr 10m;

#负载均衡配置

upstream jh.w3cschool.cn {

#upstream的负载均衡，weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。

server 192.168.80.121:80 weight=3;

server 192.168.80.122:80 weight=2;

server 192.168.80.123:80 weight=3;

#nginx的upstream目前支持4种方式的分配

#1、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，如果后端服务器down掉，能自动剔除。

#2、weight

#指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

#例如：

#upstream bakend {

# server 192.168.0.14 weight=10;

# server 192.168.0.15 weight=10;

#}

#2、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的问题。

#例如：

#upstream bakend {

# ip\_hash;

# server 192.168.0.14:88;

# server 192.168.0.15:80;

#}

#3、fair（第三方）

#按后端服务器的响应时间来分配请求，响应时间短的优先分配。

#upstream backend {

# server server1;

# server server2;

# fair;

#}

#4、url\_hash（第三方）

#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，后端服务器为缓存时比较有效。

#例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash\_method是使用的hash算法

#upstream backend {

# server squid1:3128;

# server squid2:3128;

# hash $request\_uri;

# hash\_method crc32;

#}

#tips:

#upstream bakend{#定义负载均衡设备的Ip及设备状态}{

# ip\_hash;

# server 127.0.0.1:9090 down;

# server 127.0.0.1:8080 weight=2;

# server 127.0.0.1:6060;

# server 127.0.0.1:7070 backup;

#}

#在需要使用负载均衡的server中增加 proxy\_pass http://bakend/;

#每个设备的状态设置为:

#1.down表示单前的server暂时不参与负载

#2.weight为weight越大，负载的权重就越大。

#3.max\_fails：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

#4.fail\_timeout:max\_fails次失败后，暂停的时间。

#5.backup： 其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

#nginx支持同时设置多组的负载均衡，用来给不用的server来使用。

#client\_body\_in\_file\_only设置为On 可以讲client post过来的数据记录到文件中用来做debug

#client\_body\_temp\_path设置记录文件的目录 可以设置最多3层目录

#location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡

}

**#虚拟主机的配置**

server

{

#监听端口

listen 80;

#域名可以有多个，用空格隔开

server\_name www.w3cschool.cn w3cschool.cn;

index index.html index.htm index.php;

root /data/www/w3cschool;

#对\*\*\*\*\*\*进行负载均衡

location ~ .\*.(php|php5)?$

{

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

include fastcgi.conf;

}

#图片缓存时间设置

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

{

expires 10d;

}

#JS和CSS缓存时间设置

location ~ .\*.(js|css)?$

{

expires 1h;

}

#日志格式设定

#$remote\_addr与$http\_x\_forwarded\_for用以记录客户端的ip地址；

#$remote\_user：用来记录客户端用户名称；

#$time\_local： 用来记录访问时间与时区；

#$request： 用来记录请求的url与http协议；

#$status： 用来记录请求状态；成功是200，

#$body\_bytes\_sent ：记录发送给客户端文件主体内容大小；

#$http\_referer：用来记录从那个页面链接访问过来的；

#$http\_user\_agent：记录客户浏览器的相关信息；

#通常web服务器放在反向代理的后面，这样就不能获取到客户的IP地址了，通过$remote\_add拿到的IP地址是反向代理服务器的iP地址。反向代理服务器在转发请求的http头信息中，可以增加x\_forwarded\_for信息，用以记录原有客户端的IP地址和原来客户端的请求的服务器地址。

log\_format access '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" $http\_x\_forwarded\_for';

#定义本虚拟主机的访问日志

access\_log /usr/local/nginx/logs/host.access.log main;

access\_log /usr/local/nginx/logs/host.access.404.log log404;

**#对 "/" 启用反向代理**

location / {

proxy\_pass http://127.0.0.1:88;

proxy\_redirect off;

proxy\_set\_header X-Real-IP $remote\_addr;

#后端的Web服务器可以通过X-Forwarded-For获取用户真实IP

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

#以下是一些反向代理的配置，可选。

proxy\_set\_header Host $host;

#允许客户端请求的最大单文件字节数

client\_max\_body\_size 10m;

#缓冲区代理缓冲用户端请求的最大字节数，

#如果把它设置为比较大的数值，例如256k，那么，无论使用firefox还是IE浏览器，来提交任意小于256k的图片，都很正常。如果注释该指令，使用默认的client\_body\_buffer\_size设置，也就是操作系统页面大小的两倍，8k或者16k，问题就出现了。

#无论使用firefox4.0还是IE8.0，提交一个比较大，200k左右的图片，都返回500 Internal Server Error错误

client\_body\_buffer\_size 128k;

#表示使nginx阻止HTTP应答代码为400或者更高的应答。

proxy\_intercept\_errors on;

#后端服务器连接的超时时间\_发起握手等候响应超时时间

#nginx跟后端服务器连接超时时间(代理连接超时)

proxy\_connect\_timeout 90;

#后端服务器数据回传时间(代理发送超时)

#后端服务器数据回传时间\_就是在规定时间之内后端服务器必须传完所有的数据

proxy\_send\_timeout 90;

#连接成功后，后端服务器响应时间(代理接收超时)

#连接成功后\_等候后端服务器响应时间\_其实已经进入后端的排队之中等候处理（也可以说是后端服务器处理请求的时间）

proxy\_read\_timeout 90;

#设置代理服务器（nginx）保存用户头信息的缓冲区大小

#设置从被代理服务器读取的第一部分应答的缓冲区大小，通常情况下这部分应答中包含一个小的应答头，默认情况下这个值的大小为指令proxy\_buffers中指定的一个缓冲区的大小，不过可以将其设置为更小

proxy\_buffer\_size 4k;

#proxy\_buffers缓冲区，网页平均在32k以下的设置

#设置用于读取应答（来自被代理服务器）的缓冲区数目和大小，默认情况也为分页大小，根据操作系统的不同可能是4k或者8k

proxy\_buffers 4 32k;

#高负荷下缓冲大小（proxy\_buffers\*2）

proxy\_busy\_buffers\_size 64k;

#设置在写入proxy\_temp\_path时数据的大小，预防一个工作进程在传递文件时阻塞太长

#设定缓存文件夹大小，大于这个值，将从upstream服务器传

proxy\_temp\_file\_write\_size 64k;

}

#设定查看Nginx状态的地址

location /NginxStatus {

stub\_status on;

access\_log on;

auth\_basic "NginxStatus";

auth\_basic\_user\_file confpasswd;

#htpasswd文件的内容可以用apache提供的htpasswd工具来产生。

}

#本地动静分离反向代理配置

#所有jsp的页面均交由tomcat或resin处理

location ~ .(jsp|jspx|do)?$ {

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_pass http://127.0.0.1:8080;

}

#所有静态文件由nginx直接读取不经过tomcat或resin

location ~ .\*.(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|

pdf|xls|mp3|wma)$

{

expires 15d;

}

location ~ .\*.(js|css)?$

{

expires 1h;

}

}

}

######Nginx配置文件nginx.conf中文详解#####