## Nginx配置解析 (/etc/nginx/nginx.conf)

user nginx ; #用户

worker\_processes 8; #工作进程，根据硬件调整，大于等于cpu核数

error\_log logs/nginx\_error.log crit; #错误日志

pid logs/nginx.pid; #pid放置的位置

worker\_rlimit\_nofile 204800; #指定进程可以打开的最大描述符 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。

events {

use epoll; #使用epoll的I/O 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

worker\_connections 204800; #工作进程的最大连接数量，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\_processes\*worker\_connections

keepalive\_timeout 60;

keepalive超时时间。

client\_header\_buffer\_size 4k;

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。

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

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\_valid 80s;

这个是指多长时间检查一次缓存的有效信息。

open\_file\_cache\_min\_uses 1;

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。

}

#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

{

include mime.types;

#设定mime类型,类型由mime.type文件定义

default\_type application/octet-stream;

log\_format main '$host $status [$time\_local] $remote\_addr [$time\_local] $request\_uri '

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

'$bytes\_sent $request\_time $sent\_http\_x\_cache\_hit';

log\_format log404 '$status [$time\_local] $remote\_addr $host$request\_uri $sent\_http\_location';

$remote\_addr与$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\_s ent ：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：记录客户毒啊浏览器的相关信息；

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；

access\_log /dev/null;

#用了log\_format指令设置了日志格式之后，需要用access\_log指令指定日志文件的存放路径；

# access\_log /usr/local/nginx/logs/access\_log main;

server\_names\_hash\_bucket\_size 128;

#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.

client\_header\_buffer\_size 4k;

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。

large\_client\_header\_buffers 8 128k;

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  
如果设置过小HTTP头/Cookie过大 会报400 错误nginx 400 bad request  
求行如果超过buffer，就会报HTTP 414错误(URI Too Long)  
nginx接受最长的HTTP头部大小必须比其中一个buffer大，否则就会报400的

HTTP错误(Bad Request)。

**open\_file\_cache max 102400**

使用字段:http, server, location 这个指令指定缓存是否启用,如果启用,将记录文件以下信息: ·打开的文件描述符,大小信息和修改时间. ·存在的目录信息. ·在搜索文件过程中的错误信息 --没有这个文件,无法正确读取,参考open\_file\_cache\_errors指令选项:  
·max -指定缓存的最大数目,如果缓存溢出,最长使用过的文件(LRU)将被移除  
例: open\_file\_cache max=1000 inactive=20s; open\_file\_cache\_valid 30s; open\_file\_cache\_min\_uses 2; open\_file\_cache\_errors on;

**open\_file\_cache\_errors**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\_min\_uses**

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\_valid**

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.

client\_max\_body\_size 300m;

设定通过**nginx**上传文件的大小

sendfile on;

#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。

tcp\_nopush 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

proxy\_connect\_timeout 90;   
#后端服务器连接的超时时间\_发起握手等候响应超时时间

proxy\_read\_timeout 180;

#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\_send\_timeout 180;

#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\_buffer\_size 256k;

#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\_buffers 4 256k;

#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\_busy\_buffers\_size 256k;

proxy\_temp\_file\_write\_size 256k;

#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

proxy\_temp\_path /data0/proxy\_temp\_dir;

#proxy\_temp\_path和proxy\_cache\_path指定的路径必须在同一分区  
proxy\_cache\_path /data0/proxy\_cache\_dir levels=1:2 keys\_zone=cache\_one:200m inactive=1d max\_size=30g;  
#设置内存缓存空间大小为200MB，1天没有被访问的内容自动清除，硬盘缓存空间大小为30GB。

keepalive\_timeout 120;

keepalive超时时间。

tcp\_nodelay on;

client\_body\_buffer\_size 512k;  
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

proxy\_intercept\_errors on;

表示使nginx阻止HTTP应答代码为400或者更高的应答。

upstream img\_relay {

server 127.0.0.1:8027;

server 127.0.0.1:8028;

server 127.0.0.1:8029;

hash $request\_uri;

}

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默认为1.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

server

#配置虚拟机

{

listen 80;

#配置监听端口

server\_name image.\*\*\*.com;

#配置访问域名

location ~\* \.(mp3|exe)$ {

#对以“mp3或exe”结尾的地址进行负载均衡

proxy\_pass http://img\_relay$request\_uri;

#设置被代理服务器的端口或套接字，以及URL

proxy\_set\_header Host $host;

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

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

#以上三行，目的是将代理服务器收到的用户的信息传到真实服务器上

}

location /face {

if ($http\_user\_agent ~\* "xnp") {

rewrite ^(.\*)$ http://211.151.188.190:8080/face.jpg redirect;

}

proxy\_pass http://img\_relay$request\_uri;

proxy\_set\_header Host $host;

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

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

error\_page 404 502 = @fetch;

}

location @fetch {

access\_log /data/logs/face.log log404;

#设定本服务器的访问日志

rewrite ^(.\*)$ http://211.151.188.190:8080/face.jpg redirect;

}

location /image {

if ($http\_user\_agent ~\* "xnp") {

rewrite ^(.\*)$ http://211.151.188.190:8080/face.jpg redirect;

}

proxy\_pass http://img\_relay$request\_uri;

proxy\_set\_header Host $host;

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

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

error\_page 404 502 = @fetch;

}

location @fetch {

access\_log /data/logs/image.log log404;

rewrite ^(.\*)$ http://211.151.188.190:8080/face.jpg redirect;

}

}

server

{

listen 80;

server\_name \*.\*\*\*.com \*.\*\*\*.cn;

location ~\* \.(mp3|exe)$ {

proxy\_pass http://img\_relay$request\_uri;

proxy\_set\_header Host $host;

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

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

}

location / {

if ($http\_user\_agent ~\* "xnp") {

rewrite ^(.\*)$ http://i1.\*\*\*img.com/help/noimg.gif redirect;

}

proxy\_pass http://img\_relay$request\_uri;

proxy\_set\_header Host $host;

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

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

#error\_page 404 http://i1.\*\*\*img.com/help/noimg.gif;

error\_page 404 502 = @fetch;

}

location @fetch {

access\_log /data/logs/baijiaqi.log log404;

rewrite ^(.\*)$ http://i1.\*\*\*img.com/help/noimg.gif redirect;

}

#access\_log off;

}

server

{

listen 80;

server\_name \*.\*\*\*img.com;

location ~\* \.(mp3|exe)$ {

proxy\_pass http://img\_relay$request\_uri;

proxy\_set\_header Host $host;

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

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

}

location / {

if ($http\_user\_agent ~\* "xnp") {

rewrite ^(.\*)$ http://i1.\*\*\*img.com/help/noimg.gif;

}

proxy\_pass http://img\_relay$request\_uri;

proxy\_set\_header Host $host;

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

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

#error\_page 404 http://i1.\*\*\*img.com/help/noimg.gif;

error\_page 404 = @fetch;

}

#access\_log off;

location @fetch {

access\_log /data/logs/baijiaqi.log log404;

rewrite ^(.\*)$ http://i1.\*\*\*img.com/help/noimg.gif redirect;

}

}

server

{

listen 8080;

server\_name ngx-ha.\*\*\*img.com;

location / {

stub\_status on;

access\_log off;

}

}

server {

listen 80;

server\_name imgsrc1.\*\*\*.net;

root html;

}

server {

listen 80;

server\_name \*\*\*.com w.\*\*\*.com;

# access\_log /usr/local/nginx/logs/access\_log main;

location / {

rewrite ^(.\*)$ http://www.\*\*\*.com/ ;

}

}

server {

listen 80;

server\_name \*\*\*\*\*\*\*.com w.\*\*\*\*\*\*\*.com;

# access\_log /usr/local/nginx/logs/access\_log main;

location / {

rewrite ^(.\*)$ http://www.\*\*\*\*\*\*\*.com/;

}

}

server {

listen 80;

server\_name \*\*\*\*\*\*.com;

# access\_log /usr/local/nginx/logs/access\_log main;

location / {

rewrite ^(.\*)$ http://www.\*\*\*\*\*\*.com/;

}

}

location /NginxStatus {  
stub\_status on;  
access\_log on;  
auth\_basic "NginxStatus";  
auth\_basic\_user\_file conf/htpasswd;  
}

#设定查看Nginx状态的地址

location ~ /\.ht {  
deny all;  
}

#禁止访问.htxxx文件

}

注释：变量

Ngx\_http\_core\_module模块支持内置变量，他们的名字和apache的内置变量是一致的。

首先是说明客户请求title中的行，例如$http\_user\_agent,$http\_cookie等等。

此外还有其它的一些变量

$args此变量与请求行中的参数相等

$content\_length等于请求行的“Content\_Length”的值。

$content\_type等同与请求头部的”Content\_Type”的值

$document\_root等同于当前请求的root指令指定的值

$document\_uri与$uri一样

$host与请求头部中“Host”行指定的值或是request到达的server的名字（没有Host行）一样

$limit\_rate允许限制的连接速率

$request\_method等同于request的method，通常是“GET”或“POST”

$remote\_addr客户端ip

$remote\_port客户端port

$remote\_user等同于用户名，由ngx\_http\_auth\_basic\_module认证

$request\_filename当前请求的文件的路径名，由root或alias和URI request组合而成

$request\_body\_file

$request\_uri含有参数的完整的初始URI

$query\_string与$args一样

$sheeme http模式（http,https）尽在要求是评估例如

Rewrite ^(.+)$ $sheme://example.com$; Redirect;

$server\_protocol等同于request的协议，使用“HTTP/或“HTTP/

$server\_addr request到达的server的ip，一般获得此变量的值的目的是进行系统调用。为了避免系统调用，有必要在listen指令中指明ip，并使用bind参数。

$server\_name请求到达的服务器名

$server\_port请求到达的服务器的端口号

$uri等同于当前request中的URI，可不同于初始值，例如内部重定向时或使用index