**一、一般来说nginx 配置文件中对优化比较有作用的为以下几项：**

**1. worker\_processes 8;**

nginx 进程数，建议按照cpu 数目来指定，一般为它的倍数 (如,2个四核的cpu计为8)。

**2. worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;**

为每个进程分配cpu，上例中将8 个进程分配到8 个cpu，当然可以写多个，或者将一  
个进程分配到多个cpu。

**3. worker\_rlimit\_nofile 65535;**

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。

查看linux系统文件描述符的方法：

[root@web001 ~]# sysctl -a | grep fs.file

fs.file-max = 789972

fs.file-nr = 510 0 789972

**4. 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。

)

**5. worker\_connections 65535;**

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。

**6. keepalive\_timeout 60;**

keepalive 超时时间。

**7. 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。**

**8. 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。

**9. open\_file\_cache\_valid 80s;**

这个是指多长时间检查一次缓存的有效信息。

**10. 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。

**二、关于内核参数的优化：**

**net.ipv4.tcp\_max\_tw\_buckets = 6000**

timewait 的数量，默认是180000。

**net.ipv4.ip\_local\_port\_range = 1024 65000**

允许系统打开的端口范围。

**net.ipv4.tcp\_tw\_recycle = 1**

启用timewait 快速回收。

**net.ipv4.tcp\_tw\_reuse = 1**

开启重用。允许将TIME-WAIT sockets 重新用于新的TCP 连接。

**net.ipv4.tcp\_syncookies = 1**

开启SYN Cookies，当出现SYN 等待队列溢出时，启用cookies 来处理。

**net.core.somaxconn = 262144**

web 应用中listen 函数的backlog 默认会给我们内核参数的net.core.somaxconn 限制到128，而nginx 定义的NGX\_LISTEN\_BACKLOG 默认为511，所以有必要调整这个值。

**net.core.netdev\_max\_backlog = 262144**

每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。

**net.ipv4.tcp\_max\_orphans = 262144**

系统中最多有多少个TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤儿连接将即刻被复位并打印出警告信息。这个限制仅仅是为了防止简单的DoS 攻击，不能过分依靠它或者人为地减小这个值，更应该增加这个值(如果增加了内存之后)。

**net.ipv4.tcp\_max\_syn\_backlog = 262144**

记录的那些尚未收到客户端确认信息的连接请求的最大值。对于有128M 内存的系统而言，缺省值是1024，小内存的系统则是128。

**net.ipv4.tcp\_timestamps = 0**

时间戳可以避免序列号的卷绕。一个1Gbps 的链路肯定会遇到以前用过的序列号。时间戳能够让内核接受这种“异常”的数据包。这里需要将其关掉。

**net.ipv4.tcp\_synack\_retries = 1**

为了打开对端的连接，内核需要发送一个SYN 并附带一个回应前面一个SYN 的ACK。也就是所谓三次握手中的第二次握手。这个设置决定了内核放弃连接之前发送SYN+ACK 包的数量。

**net.ipv4.tcp\_syn\_retries = 1**

在内核放弃建立连接之前发送SYN 包的数量。

**net.ipv4.tcp\_fin\_timeout = 1**

如 果套接字由本端要求关闭，这个参数决定了它保持在FIN-WAIT-2 状态的时间。对端可以出错并永远不关闭连接，甚至意外当机。缺省值是60 秒。2.2 内核的通常值是180 秒，3你可以按这个设置，但要记住的是，即使你的机器是一个轻载的WEB 服务器，也有因为大量的死套接字而内存溢出的风险，FIN- WAIT-2 的危险性比FIN-WAIT-1 要小，因为它最多只能吃掉1.5K 内存，但是它们的生存期长些。

**net.ipv4.tcp\_keepalive\_time = 30**

当keepalive 起用的时候，TCP 发送keepalive 消息的频度。缺省是2 小时。

**三、下面贴一个完整的内核优化设置:**

**vi /etc/sysctl.conf** CentOS5.5中可以将所有内容清空直接替换为如下内容:

net.ipv4.ip\_forward = 0  
net.ipv4.conf.default.rp\_filter = 1  
net.ipv4.conf.default.accept\_source\_route = 0  
kernel.sysrq = 0  
kernel.core\_uses\_pid = 1  
net.ipv4.tcp\_syncookies = 1  
kernel.msgmnb = 65536  
kernel.msgmax = 65536  
kernel.shmmax = 68719476736  
kernel.shmall = 4294967296  
net.ipv4.tcp\_max\_tw\_buckets = 6000  
net.ipv4.tcp\_sack = 1  
net.ipv4.tcp\_window\_scaling = 1  
net.ipv4.tcp\_rmem = 4096 87380 4194304  
net.ipv4.tcp\_wmem = 4096 16384 4194304  
net.core.wmem\_default = 8388608  
net.core.rmem\_default = 8388608  
net.core.rmem\_max = 16777216  
net.core.wmem\_max = 16777216  
net.core.netdev\_max\_backlog = 262144  
net.core.somaxconn = 262144  
net.ipv4.tcp\_max\_orphans = 3276800  
net.ipv4.tcp\_max\_syn\_backlog = 262144  
net.ipv4.tcp\_timestamps = 0  
net.ipv4.tcp\_synack\_retries = 1  
net.ipv4.tcp\_syn\_retries = 1  
net.ipv4.tcp\_tw\_recycle = 1  
net.ipv4.tcp\_tw\_reuse = 1  
net.ipv4.tcp\_mem = 94500000 915000000 927000000  
net.ipv4.tcp\_fin\_timeout = 1  
net.ipv4.tcp\_keepalive\_time = 30  
net.ipv4.ip\_local\_port\_range = 1024 65000

使配置立即生效可使用如下命令：  
**/sbin/sysctl -p**

**四、下面是关于系统连接数的优化**

**linux 默认值 open files 和 max user processes 为 1024**

#ulimit -n

1024

#ulimit Cu

1024

**问题描述：** 说明 server 只允许同时打开 1024 个文件，处理 1024 个用户进程

使用ulimit -a 可以查看当前系统的所有限制值，使用ulimit -n 可以查看当前的最大打开文件数。

新装的linux 默认只有1024 ，当作负载较大的服务器时，很容易遇到error: too many open files 。因此，需要将其改大。

**解决方法：**

使用 ulimit Cn 65535 可即时修改，但重启后就无效了。（注ulimit -SHn 65535 等效 ulimit -n 65535 ，-S 指soft ，-H 指hard)

有如下三种修改方式：

1. 在/etc/rc.local 中增加一行 ulimit -SHn 65535  
2. 在/etc/profile 中增加一行 ulimit -SHn 65535  
3. 在**/etc/security/limits.conf** 最后增加：

**\* soft nofile 65535  
\* hard nofile 65535  
\* soft nproc 65535  
\* hard nproc 65535**

具体使用哪种，**在 CentOS 中使用第1 种方式无效果，使用第3 种方式有效果**，而在Debian 中使用第2 种有效果

# ulimit -n

65535

# ulimit -u

65535

备注：ulimit 命令本身就有分软硬设置，加-H 就是硬，加-S 就是软默认显示的是软限制

soft 限制指的是当前系统生效的设置值。 hard 限制值可以被普通用户降低。但是不能增加。 soft 限制不能设置的比 hard 限制更高。 只有 root 用户才能够增加 hard 限制值。

**五、下面是一个简单的nginx 配置文件：**

user www www;  
worker\_processes 8;  
worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00010000 00100000  
01000000;  
error\_log /www/log/nginx\_error.log crit;  
pid /usr/local/nginx/nginx.pid;  
worker\_rlimit\_nofile 204800;  
events  
{  
use epoll;  
worker\_connections 204800;  
}  
http  
{  
include mime.types;  
default\_type application/octet-stream;  
charset utf-8;  
server\_names\_hash\_bucket\_size 128;  
client\_header\_buffer\_size 2k;  
large\_client\_header\_buffers 4 4k;  
client\_max\_body\_size 8m;  
sendfile on;  
tcp\_nopush on;  
keepalive\_timeout 60;  
fastcgi\_cache\_path /usr/local/nginx/fastcgi\_cache levels=1:2  
keys\_zone=TEST:10m  
inactive=5m;  
fastcgi\_connect\_timeout 300;  
fastcgi\_send\_timeout 300;  
fastcgi\_read\_timeout 300;  
fastcgi\_buffer\_size 4k;  
fastcgi\_buffers 8 4k;  
fastcgi\_busy\_buffers\_size 8k;  
fastcgi\_temp\_file\_write\_size 8k;  
fastcgi\_cache TEST;  
fastcgi\_cache\_valid 200 302 1h;  
fastcgi\_cache\_valid 301 1d;  
fastcgi\_cache\_valid any 1m;  
fastcgi\_cache\_min\_uses 1;  
fastcgi\_cache\_use\_stale error timeout invalid\_header http\_500;  
open\_file\_cache max=204800 inactive=20s;  
open\_file\_cache\_min\_uses 1;  
open\_file\_cache\_valid 30s;  
tcp\_nodelay on;  
gzip on;  
gzip\_min\_length 1k;  
gzip\_buffers 4 16k;  
gzip\_http\_version 1.0;  
gzip\_comp\_level 2;  
gzip\_types text/plain application/x-javascript text/css application/xml;  
gzip\_vary on;  
server  
{  
listen 8080;  
server\_name backup.aiju.com;  
index index.php index.htm;  
root /www/html/;  
location /status  
{  
stub\_status on;  
}  
location ~ .\*/.(php|php5)?$  
{  
fastcgi\_pass 127.0.0.1:9000;  
fastcgi\_index index.php;  
include fcgi.conf;  
}  
location ~ .\*/.(gif|jpg|jpeg|png|bmp|swf|js|css)$  
{  
expires 30d;  
}  
log\_format access ‘$remote\_addr — $remote\_user [$time\_local] “$request” ‘  
‘$status $body\_bytes\_sent “$http\_referer” ‘  
‘”$http\_user\_agent” $http\_x\_forwarded\_for’;  
access\_log /www/log/access.log access;  
}  
}

**六、关于FastCGI 的几个指令：**

fastcgi\_cache\_path /usr/local/nginx/fastcgi\_cache levels=1:2 keys\_zone=TEST:10minactive=5m;

这个指令为FastCGI 缓存指定一个路径，目录结构等级，关键字区域存储时间和非活动删除时间。

**fastcgi\_connect\_timeout 300;**

指定连接到后端FastCGI 的超时时间。

**fastcgi\_send\_timeout 300;**

向FastCGI 传送请求的超时时间，这个值是指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 应答的超时时间，这个值是指已经完成两次握手后接收FastCGI 应答的超时时间。

**fastcgi\_buffer\_size 4k;**

指定读取FastCGI 应答第一部分需要用多大的缓冲区，一般第一部分应答不会超过1k，由于页面大小为4k，所以这里设置为4k。

**fastcgi\_buffers 8 4k;**

指定本地需要用多少和多大的缓冲区来缓冲FastCGI 的应答。

**fastcgi\_busy\_buffers\_size 8k;**

这个指令我也不知道是做什么用，只知道默认值是fastcgi\_buffers 的两倍。

**fastcgi\_temp\_file\_write\_size 8k;**

在写入fastcgi\_temp\_path 时将用多大的数据块，默认值是fastcgi\_buffers 的两倍。

**fastcgi\_cache TEST**

开启FastCGI 缓存并且为其制定一个名称。个人感觉开启缓存非常有用，可以有效降低CPU 负载，并且防止502 错误。

**fastcgi\_cache\_valid 200 302 1h;  
fastcgi\_cache\_valid 301 1d;  
fastcgi\_cache\_valid any 1m;**

为指定的应答代码指定缓存时间，如上例中将200，302 应答缓存一小时，301 应答缓存1 天，其他为1 分钟。

**fastcgi\_cache\_min\_uses 1;**

缓存在fastcgi\_cache\_path 指令inactive 参数值时间内的最少使用次数，如上例，如果在5 分钟内某文件1 次也没有被使用，那么这个文件将被移除。

**fastcgi\_cache\_use\_stale error timeout invalid\_header http\_500;**

不知道这个参数的作用，猜想应该是让nginx 知道哪些类型的缓存是没用的。以上为nginx 中FastCGI 相关参数，另外，FastCGI 自身也有一些配置需要进行优化，如果你使用php-fpm 来管理FastCGI，可以修改配置文件中的以下值：

**<value name=”max\_children”>60</value>**

同时处理的并发请求数，即它将开启最多60 个子线程来处理并发连接。

**<value name=”rlimit\_files”>102400</value>**

最多打开文件数。

**<value name=”max\_requests”>204800</value>**

每个进程在重置之前能够执行的最多请求数。

Nginx常用模块

1. 性能相关配置

worker\_processes number | auto；

worker进程的数量；通常应该为当前主机的cpu的物理核心数。多于8个的话建议写8，超过8个性能不会提升，稳定性降低。

worker\_cpu\_affinity auto [cpumask] #将work进程绑定在固定cpu上提高缓存命中率

例：

worker\_cpu\_affinity 0001 0010 0100 1000;

worker\_cpu\_affinity 0101 1010;

worker\_priority number

指定worker进程的nice值，设定worker进程优先级： [-20,20]

worker\_rlimit\_nofile number

worker进程所能够打开的文件数量上限,默认较小，生产中需要调大如65535。系统资源通过配置修改/etc/security/limits.conf 例：root soft nofile 65535，或命令修改ulimit -n，修改后需重启服务或系统生效。

2. 时间驱动events相关的配置

worker\_connections number

每个worker进程所能够打开的最大并发连接数数量，如10240

总最大并发数： worker\_processes \* worker\_connections

use method

指明并发连接请求的处理方法,默认自动选择最优方法不用调整

如：use epoll;

accept\_mutex on | off 互斥；

处理新的连接请求的方法； on指由各个worker轮流处理新请求

， Off指每个新请求的到达都会通知(唤醒)所有的worker进程，但

只有一个进程可获得连接，造成“惊群”，影响性能，默认on

3. http核心模块相关配置ngx\_http\_core\_module

3.1web服务模板

server { ... }

配置一个虚拟主机

server {

listen address[:PORT]|PORT;

server\_name SERVER\_NAME;

root /PATH/TO/DOCUMENT\_ROOT;

}

注意：

(1) 基于port；

listen PORT; 指令监听在不同的端口

(2) 基于ip的虚拟主机

listen IP:PORT; IP 地址不同

(3) 基于hostname

server\_name fqdn; 指令指向不同的主机名

3.2套接字相关配置

listen address[:port] [default\_server] [ssl] [http2 | spdy] [backlog=number] [rcvbuf=size] [sndbuf=size]

default\_server 设定为默认虚拟主机

ssl 限制仅能够通过ssl连接提供服务

backlog=number 超过并发连接数后，新请求进入后援队列的长度

rcvbuf=size 接收缓冲区大小

sndbuf=size 发送缓冲区大小

3.3 server\_name

server\_name name ...;

支持\*通配任意长度的任意字符

server\_name \*.magedu.com www.magedu.\*

支持~起始的字符做正则表达式模式匹配，性能原因慎用

server\_name ~^www\d+\.magedu\.com$ #\d 表示 [0-9]

匹配优先级机制从高到低：

(1) 首先是字符串精确匹配 如： www.magedu.com

(2) 左侧\*通配符 如： \*.magedu.com

(3) 右侧\*通配符 如： www.magedu.\*

(4) 正则表达式 如： ~^.\*\.magedu\.com$

(5) default\_server

3.4 延迟发送选项

tcp\_nodelay on | off;

tcp\_nopush on | off;

在keepalived模式下的连接是否启用TCP\_NODELAY选项。

tcp\_nopush必须在sendfile 为on时才有效，当为off时，延迟发送，合并多个请求后再发送

默认On时，不延迟发送

可用于： http, server, location

3.5 sendfile

sendfile on | off;

是否启用sendfile功能，在内核中封装报文直接发送。如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处理速度降低系统负载，如图片显示不正常把这个改为off。

默认Off

3.6 隐藏版本信息

server\_tokens on | off | build | string

是否在响应报文的Server首部显示nginx版本

3.7 location匹配

location [ = | ~ | ~\* | ^~ ] uri { ... }

location @name { ... }

在一个server中location配置段可存在多个，用于实现从uri到文件系统的路径映射； ngnix会根据

用户请求的URI来检查定义的所有location，并找出一个最佳匹配，而后应用其配置

示例：

server {...

server\_name www.magedu.com;

location /images/ {

root /data/imgs/;

}

}

http://www.magedu.com/images/logo.jpg

--> /data/imgs/images/logo.jpg

=：对URI做精确匹配；

^~：对URI的最左边部分做匹配检查，不区分字符大小写

~：对URI做正则表达式模式匹配，区分字符大小写

~\*：对URI做正则表达式模式匹配，不区分字符大小写

不带符号：匹配起始于此uri的所有的uri

匹配优先级从高到低：

=, ^~, ～/～\*, 不带符号

3.7 路径别名alias path

示例：

http://www.magedu.com/bbs/index.php

location /bbs/ {

alias /web/forum/;

} --> /web/forum/index.html

location /bbs/ {

root /web/forum/;

} --> /web/forum/bbs/index.html

注意： location中使用root指令和alias指令的意义不同

(a) root，相当于追加在root目录后面

(b) alias，相当于对location中的url进行替换

3.8 错误页面显示

error\_page code ... [=[response]] uri;

模块： ngx\_http\_core\_module

定义错误页， 以指定的响应状态码进行响应

可用位置： http, server, location, if in location

error\_page 404 /404.html

error\_page 404 =200 /404.html #防止404页面被劫持

3.9 长连接相关配置

keepalive\_timeout timeout [header\_timeout];

设定保持连接超时时长， 0表示禁止长连接， 默认为75s

keepalive\_requests number;

在一次长连接上所允许请求的资源的最大数量，默认为100

keepalive\_disable none | browser ...

对哪种浏览器禁用长连接

send\_timeout time;

向客户端发送响应报文的超时时长，此处是指两次写操作之间的间隔时长，而非

整个响应过程的传输时长

3.10 请求报文缓存

client\_body\_buffer\_size size;

用于接收每个客户端请求报文的body部分的缓冲区大小；默认为16k；超出此大小时，

其将被暂存到磁盘上的由client\_body\_temp\_path指令所定义的位置

client\_body\_temp\_path path [level1 [level2 [level3]]];

设定用于存储客户端请求报文的body部分的临时存储路径及子目录结构和数量

目录名为16进制的数字；

client\_body\_temp\_path /var/tmp/client\_body 1 2 2

1 1级目录占1位16进制，即2^4=16个目录 0-f

2 2级目录占2位16进制，即2^8=256个目录 00-ff

2 3级目录占2位16进制， 即2^8=256个目录 00-ff

3.11 对客户端进行限制相关配置

limit\_rate rate;

限制响应给客户端的传输速率，单位是bytes/second 默认值0表示无限制

limit\_except method ... { ... }，仅用于location

限制客户端使用除了指定的请求方法之外的其它方法

method:GET, HEAD, POST, PUT, DELETE，MKCOL, COPY, MOVE, OPTIONS, PROPFIND,

PROPPATCH, LOCK, UNLOCK, PATCH

例：

limit\_except GET {

allow 192.168.1.0/24;

deny all;

}

除了GET和HEAD 之外其它方法仅允许192.168.1.0/24网段主机使用

4. 访问控制模块ngx\_http\_access\_module

实现基于ip的访问控制功能

allow address | CIDR | unix: | all;

deny address | CIDR | unix: | all;

http, server, location, limit\_except

自上而下检查，一旦匹配，将生效，条件严格的置前

示例：

location / {

deny 192.168.1.1;

allow 192.168.1.0/24;

allow 10.1.1.0/16;

allow 2001:0db8::/32;

deny all;

}

5. 用户认证模块ngx\_http\_auth\_basic\_module

实现基于用户的访问控制，使用basic机制进行用户认证

auth\_basic string | off;

auth\_basic\_user\_file file;

location /admin/ {

auth\_basic "Admin Area";

auth\_basic\_user\_file /etc/nginx/.ngxpasswd;

}

用户口令：

1、明文文本：格式name:password:comment

2、加密文本：由htpasswd命令实现 httpd-tools所提供

htpasswd [-c第一次创建时使用] [-D删除用户] passwdfile username

6. 状态查看模块ngx\_http\_stub\_status\_module

用于输出nginx的基本状态信息

Active connections:当前状态，活动状态的连接数

accepts：统计总值，已经接受的客户端请求的总数

handled：统计总值，已经处理完成的客户端请求的总数

requests：统计总值，客户端发来的总的请求数

Reading：当前状态，正在读取客户端请求报文首部的连接的连接数

Writing：当前状态，正在向客户端发送响应报文过程中的连接数

Waiting：当前状态，正在等待客户端发出请求的空闲连接数

示例：

location /status {

stub\_status;

allow 172.16.0.0/16;

deny all;

}

7. 日志记录模块ngx\_http\_log\_module

1、 log\_format name string ...;

string可以使用nginx核心模块及其它模块内嵌的变量

2、 access\_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]];

access\_log off;

访问日志文件路径，格式及相关的缓冲的配置

buffer=size

flush=time

示例

log\_format compression '$remote\_addr-$remote\_user [$time\_local] '

'"$request" $status $bytes\_sent '

'"$http\_referer" "$http\_user\_agent" "$gzip\_ratio"';

access\_log /spool/logs/nginx-access.log compression buffer=32k;

json格式日志示例;log\_format json '{"@timestamp":"$time\_iso8601",'

'"client\_ip":"$remote\_addr",'

'"size":$body\_bytes\_sent,'

'"responsetime":$request\_time,'

'"upstreamtime":"$upstream\_response\_time",'

'"upstreamhost":"$upstream\_addr",'

'"http\_host":"$host",'

'"method":"$request\_method",'

'"request\_uri":"$request\_uri",'

'"xff":"$http\_x\_forwarded\_for",'

'"referrer":"$http\_referer",'

'"agent":"$http\_user\_agent",'

'"status":"$status"}';

3、 open\_log\_file\_cache max=N [inactive=time] [min\_uses=N] [valid=time];

open\_log\_file\_cache off;

缓存各日志文件相关的元数据信息

max：缓存的最大文件描述符数量

min\_uses：在inactive指定的时长内访问大于等于此值方可被当作活动项

inactive：非活动时长

valid：验正缓存中各缓存项是否为活动项的时间间隔

例: open\_log\_file\_cache max=1000 inactive=20s valid=1m;

8. 压缩相关选项ngx\_http\_gzip\_module

1、gzip on | off; #启用或禁用gzip压缩

2、gzip\_comp\_level level; #压缩比由低到高： 1 到 9 默认： 1

3、gzip\_disable regex ...; #匹配到客户端浏览器不执行压缩

4、gzip\_min\_length length; #启用压缩功能的响应报文大小阈值

5、gzip\_http\_version 1.0 | 1.1; #设定启用压缩功能时，协议的最小版本 默认： 1.1

6、gzip\_buffers number size;

支持实现压缩功能时缓冲区数量及每个缓存区的大小

默认： 32 4k 或 16 8k

7、gzip\_types mime-type ...;

指明仅对哪些类型的资源执行压缩操作；即压缩过滤器

默认包含有text/html，不用显示指定，否则出错

8、gzip\_vary on | off;

如果启用压缩，是否在响应报文首部插入“Vary: AcceptEncoding

9、 gzip\_proxied off | expired | no-cache | no-store |

private | no\_last\_modified | no\_etag | auth | any ...;

nginx对于代理服务器请求的响应报文，在何种条件下启

用压缩功能

off：对被代理的请求不启用压缩

expired,no-cache, no-store， private：对代理服务器

请求的响应报文首部Cache-Control值任何一个，启用压缩功能

示例：

gzip on;

gzip\_comp\_level 6;

gzip\_http\_version 1.1;

gzip\_vary on;

gzip\_min\_length 1024;

gzip\_buffers 16 8k;

gzip\_proxied any;

gzip\_disable "MSIE[1-6]\.(?!.\*SV1)";

gzip\_types text/xml text/plain text/css application/javascript application/xml application/json;

9. https模块ngx\_http\_ssl\_module模块：

1、 ssl on | off;

为指定虚拟机启用HTTPS protocol， 建议用listen指令代替

2、 ssl\_certificate file;

当前虚拟主机使用PEM格式的证书文件

3、 ssl\_certificate\_key file;

当前虚拟主机上与其证书匹配的私钥文件

4、 ssl\_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];

支持ssl协议版本，默认为后三个

5、 ssl\_session\_cache off | none | [builtin[:size]]

[shared:name:size];

builtin[:size]：使用OpenSSL内建缓存，为每worker进程私有

[shared:name:size]：在各worker之间使用一个共享的缓存

6、 ssl\_session\_timeout time;

客户端连接可以复用ssl session cache中缓存的ssl参数的有

效时长，默认5m

示例：

server {

listen 443 ssl;

server\_name www.magedu.com;

root /vhosts/ssl/htdocs;

ssl on;

ssl\_certificate /etc/nginx/ssl/nginx.crt;

ssl\_certificate\_key /etc/nginx/ssl/nginx.key;

ssl\_session\_cache shared:sslcache:20m;

ssl\_session\_timeout 10m;

}

10. 重定向模块ngx\_http\_rewrite\_module：

1、rewrite regex replacement [flag]

将用户请求的URI基于regex所描述的模式进行检查，匹配到时将其替换为replacement指定的新的URI

注意：如果在同一级配置块中存在多个rewrite规则，那么会自下而下逐个检查；被某条件规则替换

完成后，会重新一轮的替换检查

隐含有循环机制,但不超过10次；如果超过，提示500响应码， [flag]所表示的标志位用于控制此循环

机制

如果replacement是以http://或https://开头，则替换结果会直接以重向返回给客户端

[flag]：

last：重写完成后停止对当前URI在当前location中后续

的其它重写操作，而后对新的URI启动新一轮重写检查；提前重

启新一轮循环

break：重写完成后停止对当前URI在当前location中后

续的其它重写操作，而后直接跳转至重写规则配置块之后的其它

配置；结束循环，建议在location中使用

redirect：临时重定向，重写完成后以临时重定向方式直

接返回重写后生成的新URI给客户端，由客户端重新发起请求；

不能以http://或https://开头，使用相对路径，状态码： 302

permanent:重写完成后以永久重定向方式直接返回重写

后生成的新URI给客户端，由客户端重新发起请求，状态码：301

例：

rewrite ^/zz/(.\*\.html)$ /zhengzhou/$1 break;

rewrite ^/zz/(.\*\.html)$ https://www.dianping/zhengzhou/$1 permanent;

2、 return

return code [text];

return code URL;

return URL;

停止处理，并返回给客户端指定的响应码

3、 rewrite\_log on | off;

是否开启重写日志, 发送至error\_log（notice level）

4、 set $variable value;

用户自定义变量

注意：变量定义和调用都要以$开头

5、 if (condition) { ... }

引入新的上下文,条件满足时，执行配置块中的配置指令； server, location

condition：

比较操作符：

== 相同

!= 不同

~：模式匹配，区分字符大小写

~\*：模式匹配，不区分字符大小写

!~：模式不匹配，区分字符大小写

!~\*：模式不匹配，不区分字符大小写

文件及目录存在性判断：

-e, !-e 存在（包括文件，目录，软链接）

-f, !-f 文件

-d, !-d 目录

-x, !-x 执行

浏览器分流示例：

if ($http\_user\_agent ~ Chrom) {

rewrite ^(.\*)$ /chrome/$1 break;

}

if ($http\_user\_agent ~ MSIE) {

rewrite ^(.\*)$ /IE/$1 break;

}

11. 引用模块ngx\_http\_referer\_module

valid\_referers none|blocked|server\_names|string ...;

定义referer首部的合法可用值，不能匹配的将是非法值,用于防盗链，

none：请求报文首部没有referer首部,比如直接在浏览器打开一个图片

blocked：请求报文有referer首部，但无有效值，伪装的头部信息。

server\_names：参数，其可以有值作为主机名或主机名模式

arbitrary\_string：任意字符串，但可使用\*作通配符

regular expression：被指定的正则表达式模式匹配到的字符

串,要使用~开头，例如： ~.\*\.magedu\.com

示例：

location ~\*^.+\.(jpg|gif|png|swf|flv|wma|wmv|asf|mp3|mmf|zip|rar)$ {

valid\_referers none blocked server\_names \*.magedu.com

\*.mageedu.com magedu.\* mageedu.\* ~\.magedu\.;

if ($invalid\_referer) {

return 403;

break；

}

access\_log off;

}

12. 反向代理模块ngx\_http\_proxy\_module

12.1 proxy\_pass URL;  
Context:location, if in location, limit\_except

注意： proxy\_pass后面的路径不带uri时，其会将location的uri传递给后端主机

server {

...

server\_name HOSTNAME;

location /uri/ {

proxy\_pass http://host[:port];

}

...

}

上面示例： http://HOSTNAME/uri --> http://host/uri

http://host[:port]/ 意味着： http://HOSTNAME/uri --> http://host/

注意：如果location定义其uri时使用了正则表达式的模式，则proxy\_pass之后必须不能使用uri;

用户请求时传递的uri将直接附加代理到的服务的之后

server {

...

server\_name HOSTNAME;

location ~|~\* /uri/ {

proxy\_pass http://host; 不能加/

}

...

}

http://HOSTNAME/uri/ --> http://host/uri/

12.2 proxy\_set\_header field value;  
设定发往后端主机的请求报文的请求首部的值  
Context: http, server, location

后端记录日志记录真实请求服务器IP

proxy\_set\_header Host $host；

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

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

标准格式如下：

X-Forwarded-For: client1, proxy1, proxy2

如后端是Apache服务器应更改日志格式：

%h -----> %{X-Real-IP}i

12.3 proxy\_cache\_path;

定义可用于proxy功能的缓存； Context:http

proxy\_cache\_path path [levels=levels] [use\_temp\_path=on|off]

keys\_zone=name:size [inactive=time] [max\_size=size]

[manager\_files=number] [manager\_sleep=time]

[manager\_threshold=time] [loader\_files=number] [loader\_sleep=time]

[loader\_threshold=time] [purger=on|off] [purger\_files=number]

[purger\_sleep=time] [purger\_threshold=time];

例：proxy\_cache\_path /data/nginx/cache（属主要为nginx） levels=1:2 keys\_zone=nginxcache:20m inactive=2m

12.4 调用缓存

proxy\_cache zone | off; 默认off

指明调用的缓存，或关闭缓存机制； Context: http,server, location

12.5

proxy\_cache\_key string;

缓存中用于“键”的内容

默认值： proxy\_cache\_key $scheme$proxy\_host$request\_uri;

12.6

proxy\_cache\_valid [code ...] time;

定义对特定响应码的响应内容的缓存时长

定义在http{...}中

示例:

proxy\_cache\_valid 200 302 10m;

proxy\_cache\_valid 404 1m;

示例：

在http配置定义缓存信

proxy\_cache\_path /var/cache/nginx/proxy\_cache

levels=1:1:1 keys\_zone=proxycache:20m

inactive=120s max\_size=1g;

调用缓存功能，需要定义在相应的配置段，如server{...}；

proxy\_cache proxycache;

proxy\_cache\_key $request\_uri;

proxy\_cache\_valid 200 302 301 1h;

proxy\_cache\_valid any 1m;

12.7

proxy\_cache\_use\_stale;

proxy\_cache\_use\_stale error | timeout |

invalid\_header | updating | http\_500 | http\_502 |

http\_503 | http\_504 | http\_403 | http\_404 | off ...

在被代理的后端服务器出现哪种情况下，可以直接使用过

期的缓存响应客户端

12.8

proxy\_cache\_methods GET | HEAD | POST ...;

对哪些客户端请求方法对应的响应进行缓存， GET和HEAD方法总是被缓存

12.9

proxy\_hide\_header field;

By default, nginx does not pass the header fields

“Date”, “Server”, “X-Pad”, and “X-Accel-...” from the

response of a proxied server to a client. 用于隐藏后端服

务器特定的响应首部

12.10

proxy\_connect\_timeout time;

定义与后端服务器建立连接的超时时长，如超时会出现502错误，默认为60s，一般不建议超出75s

12.11

proxy\_send\_timeout time;

把请求发送给后端服务器的超时时长；默认为60s

12.12

proxy\_read\_timeout time;

等待后端服务器发送响应报文的超时时长， 默认为60s

13. 首部信息

add\_header name value [always];

添加自定义首部

add\_header X-Via $server\_addr;

add\_header X-Cache $upstream\_cache\_status;

add\_header X-Accel $server\_name;

add\_trailer name value [always];

添加自定义响应信息的尾部

14. hph 相关模块ngx\_http\_fastcgi\_module

14.1

fastcgi\_pass address;

address为后端的fastcgi server的地址

可用位置： location, if in location

14.2

fastcgi\_index name;

fastcgi默认的主页资源

示例： fastcgi\_index index.php;

14.3

fastcgi\_param parameter value [if\_not\_empty];

设置传递给 FastCGI服务器的参数值，可以是文本，变

量或组合

示例1：

1）在后端服务器先配置fpm server和mariadb-server

2）在前端nginx服务上做以下配置：

location ~\* \.php$ {

fastcgi\_pass 后端fpm服务器IP:9000;

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME

/usr/share/nginx/html$fastcgi\_script\_name;

include fastcgi.conf;

…

}

示例2：

通过/pm\_status和/ping来获取fpm server状态信息（真实服务器端php-fpm配置文件中将这两项

注释掉）

location ~\* ^/(status|ping)$ {

include fastcgi\_params;

fastcgi\_pass 后端fpm服务器IP:9000;

fastcgi\_param SCRIPT\_FILENAME $fastcgi\_script\_name;

include fastcgi.conf;

}

14.4 fastcgi 缓存相关

fastcgi\_cache\_path path [levels=levels] [use\_temp\_path=on|off]

keys\_zone=name:size [inactive=time] [max\_size=size]

[manager\_files=number] [manager\_sleep=time] [manager\_threshold=time]

[loader\_files=number] [loader\_sleep=time] [loader\_threshold=time]

[purger=on|off] [purger\_files=number] [purger\_sleep=time]

[purger\_threshold=time];

定义fastcgi的缓存；

path 缓存位置为磁盘上的文件系统

max\_size=size

磁盘path路径中用于缓存数据的缓存空间上限

levels=levels：缓存目录的层级数量，以及每一级的目录数量

levels=ONE:TWO:THREE

示例： leves=1:2:2

keys\_zone=name:size

k/v映射的内存空间的名称及大小

inactive=time

非活动时长

14.5

fastcgi\_cache zone | off;

调用指定的缓存空间来缓存数据

可用位置： http, server, location

14.6

fastcgi\_cache\_key string;

定义用作缓存项的key的字符串

示例： fastcgi\_cache\_key $request\_rui;

14.7

fastcgi\_cache\_methods GET | HEAD | POST ...;

为哪些请求方法使用缓存

14.8

fastcgi\_cache\_min\_uses number;

缓存空间中的缓存项在inactive定义的非活动时间内至少要被访问到

此处所指定的次数方可被认作活动项

14.9

fastcgi\_keep\_conn on | off;

收到后端服务器响应后， fastcgi服务器是否关闭连接，建议启用长连接

14.10

fastcgi\_cache\_valid [code ...] time;

不同的响应码各自的缓存时长

示例：

http {

fastcgi\_cache\_path /var/cache/nginx/fcgi\_cache

levels=1:2:1 keys\_zone=fcgicache:20m inactive=120s;

...

server {

location ~\* \.php$ {

...

fastcgi\_cache fcgicache;

fastcgi\_cache\_key $request\_uri;

fastcgi\_cache\_valid 200 302 10m;

fastcgi\_cache\_valid 301 1h;

fastcgi\_cache\_valid any 1m;

...

}

}

15. 代理模块ngx\_http\_upstream\_module模块

用于将多个服务器定义成服务器组，而由proxy\_pass,fastcgi\_pass等指令进行引用

15.1

upstream name { ... }

定义后端服务器组，会引入一个新的上下文

默认调度算法是wrr

Context: http

upstream httpdsrvs {

server ...

server...

...

}

15.2

server address [parameters];

在upstream上下文中server成员，以及相关的参数； Context:upstream

address的表示格式：

unix:/PATH/TO/SOME\_SOCK\_FILE

IP[:PORT]

HOSTNAME[:PORT]

parameters：

weight=number 权重，默认为1

max\_conns 连接后端报务器最大并发活动连接数， 1.11.5后支持

max\_fails=number 失败尝试最大次数；超出此处指定的次数时

server将被标记为不可用,默认为1

fail\_timeout=time 后端服务器标记为不可用状态的连接超时时

长，默认10s

backup 将服务器标记为“备用”，即所有服务器均不可用时才启用

down 标记为“不可用”，配合ip\_hash使用，实现灰度发布

15.3

ip\_hash 源地址hash调度方法

15.4

least\_conn 最少连接调度算法，当server拥有不同的权重时其为wlc，当所有后端主机连接数相同时，

则使用wrr，适用于长连接

15.5

hash key [consistent] 基于指定的key的hash表来实

现对请求的调度，此处的key可以直接文本、变量或二者组合

作用：将请求分类，同一类请求将发往同一个upstream

server，使用consistent参数， 将使用ketama一致性hash算法，

适用于后端是Cache服务器（如varnish）时使用

hash $request\_uri consistent;

hash $remote\_addr;

15.6

keepalive 连接数N;

为每个worker进程保留的空闲的长连接数量,可节约nginx

端口，并减少连接管理的消耗

15.7

health\_check [parameters];

健康状态检测机制；只能用于location上下文

常用参数：

interval=time检测的频率，默认为5秒

fails=number：判定服务器不可用的失败检测次数；默认为1次

passes=number：判定服务器可用的失败检测次数；默认为1次

uri=uri：做健康状态检测测试的目标uri；默认为/

match=NAME：健康状态检测的结果评估调用此处指定的match配置块

注意：仅对nginx plus有效

15.8

match name { ... }

对backend server做健康状态检测时，定义其结果判断机制；

只能用于http上下文

常用的参数：

status code[ code ...]: 期望的响应状态码

header HEADER[operator value]：期望存在响应首

部，也可对期望的响应首部的值基于比较操作符和值进行比较

body：期望响应报文的主体部分应该有的内容

注意：仅对nginx plus有效

16. ngx\_stream\_core\_module模块

模拟反代基于tcp或udp的服务连接，即工作于传输层的反代或调度器

stream { ... }

定义stream相关的服务； Context:main

stream {

upstream telnetsrvs {

server 192.168.22.2:23;

server 192.168.22.3:23;

least\_conn;

}

server {

listen 10.1.0.6:23;

proxy\_pass telnetsrvs;

}

}

listen address:port [ssl] [udp] [proxy\_protocol]

[backlog=number] [bind] [ipv6only=on|off] [reuseport]

[so\_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];

17. ngx\_stream\_proxy\_module模块

可实现代理基于TCP， UDP (1.9.13), UNIX-domain

sockets的数据流

1 proxy\_pass address;

指定后端服务器地址

2 proxy\_timeout timeout;

无数据传输时，保持连接状态的超时时长

默认为10m

3 proxy\_connect\_timeout time;

设置nginx与被代理的服务器尝试建立连接的超时时长

默认为60s

示例：

stream {

upstream telnetsrvs {

server 192.168.10.130:23;

server 192.168.10.131:23;

hash $remote\_addr consistent;

}

server {

listen 172.16.100.10:2323;

proxy\_pass telnetsrvs;

proxy\_timeout 60s;

proxy\_connect\_timeout 10s;

}

}

17.linux对于nginx做的内核优化(/etc/sysctl.conf)

fs.file-max = 999999

net.ipv4.ip\_forward = 0

net.ipv4.conf.default.rp\_filter = 1

net.ipv4.conf.default.accept\_source\_route = 0

kernel.sysrq = 0

kernel.core\_uses\_pid = 1

net.ipv4.tcp\_syncookies = 1

kernel.msgmnb = 65536

kernel.msgmax = 65536

kernel.shmmax = 68719476736

kernel.shmall = 4294967296

net.ipv4.tcp\_max\_tw\_buckets = 6000

net.ipv4.tcp\_sack = 1

net.ipv4.tcp\_window\_scaling = 1

net.ipv4.tcp\_rmem = 10240 87380 12582912

net.ipv4.tcp\_wmem = 10240 87380 12582912

net.core.wmem\_default = 8388608

net.core.rmem\_default = 8388608

net.core.rmem\_max = 16777216

net.core.wmem\_max = 16777216

net.core.netdev\_max\_backlog = 262144

net.core.somaxconn = 40960

net.ipv4.tcp\_max\_orphans = 3276800

net.ipv4.tcp\_max\_syn\_backlog = 262144

net.ipv4.tcp\_timestamps = 0

net.ipv4.tcp\_synack\_retries = 1

net.ipv4.tcp\_syn\_retries = 1

net.ipv4.tcp\_tw\_recycle = 1

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_mem = 94500000 915000000 927000000

net.ipv4.tcp\_fin\_timeout = 1

net.ipv4.tcp\_keepalive\_time = 30

net.ipv4.ip\_local\_port\_range = 1024 65000

执行sysctl -p使内核修改生效