nginx 常用模块整理

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 ：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

ngx_http_rewrite_module模块
常用的几个指令
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，因此，隐含有循环机制；[flag]所表示的标志位用于控制此循环机制；

[flag]：

last：如果规则有很多条。这里重写完成一次之后就会重新开始匹配规则，直至最后一条。也就是说。如果规则写的不好很容易造成死循环，不停的重写规则。
break：重写完成之后不再从头再次匹配规则。直接跳出循环
redirect：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；
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；
示例比如我们要将http请求重定向到https请求。我们可以在http的server里面这样写里面这样写

```
 rewrite (.*)$ https://www.ice.com$1 break;
1
```

具体可以看官方文档
http://nginx.org/en/docs/http/ngx_http_rewrite_module.html#return