一、yum安装nginx

1、vim /etc/yum.repos.d/nginx.repo

[nginx]

name=nginx repo

baseurl=http://nginx.org/packages/centos/$releasever/$basearch/

gpgcheck=0

enabled=1

2、yum install -y nginx //安装

3、systemctl start nginx //启动nginx

二、源码安装

wget http://nginx.org/download/nginx-1.14.0.tar.gz //下载

tar zxvf nginx-1.14.0.tar.gz //解压

./configure --prefix=/usr/local/nginx //编译

安装相关包：yum -y install gcc-c++ yum install -y pcre-devel yum install -y zlib-devel

make && make install

ls /usr/local/nginx/ //安装的目录

ls /usr/local/nginx/html/ //默认页面

/usr/local/nginx/sbin/nginx -V //查看版本

/usr/local/nginx/sbin/nginx //启动服务

/usr/local/nginx/sbin/nginx -t //检测配置文件

/usr/local/nginx/sbin/nginx -s reload //重新加载配置

vim /etc/ini.t/nginx //nginx启动脚本

#!/bin/bash

# chkconfig: - 30 21

# description: http service.

# Source Function Library

. /etc/init.d/functions

# Nginx Settings

NGINX\_SBIN="/usr/local/nginx/sbin/nginx"

NGINX\_CONF="/usr/local/nginx/conf/nginx.conf"

NGINX\_PID="/usr/local/nginx/logs/nginx.pid"

RETVAL=0

prog="Nginx"

start()

{

echo -n $"Starting $prog: "

mkdir -p /dev/shm/nginx\_temp

daemon $NGINX\_SBIN -c $NGINX\_CONF

RETVAL=$?

echo

return $RETVAL

}

stop()

{

echo -n $"Stopping $prog: "

killproc -p $NGINX\_PID $NGINX\_SBIN -TERM

rm -rf /dev/shm/nginx\_temp

RETVAL=$?

echo

return $RETVAL

}

reload()

{

echo -n $"Reloading $prog: "

killproc -p $NGINX\_PID $NGINX\_SBIN -HUP

RETVAL=$?

echo

return $RETVAL

}

restart()

{

stop

start

}

configtest()

{

$NGINX\_SBIN -c $NGINX\_CONF -t

return 0

}

case "$1" in

start)

start

;;

stop)

stop

;;

reload)

reload

;;

restart)

restart

;;

configtest)

configtest

;;

\*)

echo $"Usage: $0 {start|stop|reload|restart|configtest}"

RETVAL=1

esac

exit $RETVAL

chkconfig --add nginx //加入服务列表

chkconfig nginx on //开机启动

三、nginx的配置文件

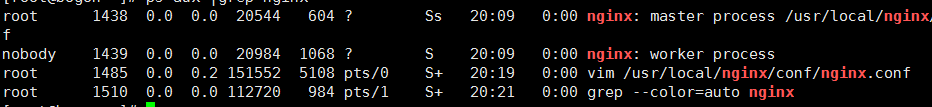
1、/usr/local/nginx/conf/nginx.conf //源码安装的配置文件

2、/etc/nginx/ //yum 安装的配置文件目录

/usr/local/nginx/conf/nginx.conf 解释

#user nobody; //主进程（master）用户是root,worker进程用户是nobody

worker\_processes 1; //worker进程启动的数量，这里只有1个，几核CPU就跳成几



3、/usr/local/nginx/logs/ //日志存放的地方

4、全局配置(user、worker\_processes、error\_log、pid)就是配置文件里面 events上面部分

**备注：**

**nginx.conf全局配置**

**user nobody;**

定义运行nginx服务的用户,还可以加上组,如 **user** **nobody** nobody;

**worker\_processes 1;**

定义nginx子进程数量，即提供服务的进程数量，该数值建议和服务cpu核数保持一致。

除了可以定义数字外，还可以定义为**auto**，表示让系统自动调整。

**error\_log logs/error.log;**

定义错误日志的路径，可以是相对路径（相对prefix路径的），也可以是绝对路径。

该配置可以在此处定义，也可以定义到http、**server**、location里

**error\_log logs/error.log notice;**

定义错误日志路径以及日志级别.

错误日志级别：常见的错误日志级别有[debug|**info**|**notice**|**warn**|**error**|**crit**|**alert**|**emerg**]，级别越高记录的信息越少。

如果不定义默认是crit

**pid logs/nginx.pid;**

定义nginx进程**pid**文件所在路径，可以是相对路径，也可以是绝对路径。

**worker\_rlimit\_nofile 100000;**

定义nginx最多打开文件数限制。如果没设置的话，这个值为操作系统（ulimit -n）的限制保持一致。

把这个值设高，nginx就不会有“too many open files”问题了。

5、events(网络连接相关，worker\_connections)

备注：

### events配置部分

#### worker\_connections 1024;

定义每个work\_process同时开启的最大连接数，即允许最多只能有这么多连接。如果worker\_processes的值为2，那么最大连接数就是2048

#### accept\_mutex on;

当某一个时刻只有一个网络连接请求服务器时，服务器上有多个睡眠的进程会被同时叫醒，这样会损耗一定的服务器性能。

Nginx中的accept\_mutex设置为**on**，将会对多个Nginx进程（worker processer）接收连接时进行序列化，防止多个进程争抢资源。

默认就是**on**。

#### multi\_accept on;

nginx worker processer可以做到同时接收多个新到达的网络连接，前提是把该参数设置为**on**。

默认为off，即每个worker process一次只能接收一个新到达的网络连接。

#### use epoll;

Nginx服务器提供了多个事件驱动器模型来处理网络消息。

其支持的类型有：select、poll、kqueue、epoll、rtsing、/dev/poll以及eventport。

\* select：只能在Windows下使用，这个事件模型不建议在高负载的系统使用

\* poll:Nginx默认首选，但不是在所有系统下都可用

\* kqueue:这种方式在FreeBSD 4.1+, OpenBSD2.9+, NetBSD 2.0, 和 MacOS X系统中是最高效的

\* epoll: 这种方式是在Linux 2.6+内核中最高效的方式

\* rtsig:实时信号，可用在Linux 2.2.19的内核中，但不适用在高流量的系统中

\* /dev/poll: Solaris 7 11/99+,HP/UX 11.22+, IRIX 6.5.15+, and Tru64 UNIX 5.1A+操作系统最高效的方式

\* eventport: Solaris 10最高效的方式

6、http（最重要的部分，大部分功能都放这里）

### http配置部分

##### 官方文档 <http://nginx.org/en/docs/>

##### 参考链接： <https://segmentfault.com/a/1190000012672431>

##### 参考链接： <https://segmentfault.com/a/1190000002797601>

##### 参考链接：http的header <https://kb.cnblogs.com/page/92320/>

#### MIME-Type

**include mime.types;** *//cat conf/mime.types*

定义nginx能识别的网络资源媒体类型（如，文本、html、js、css、流媒体等）

default\_type application/octet-stream;

定义默认的type，如果不定义改行，默认为text/plain.

#### log\_format

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

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

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

其中main为日志格式的名字，后面的为nginx的内部变量组成的一串字符串。

#### access\_log logs/access.log main;

定义日志的路径以及采用的日志格式，该参数可以在**server**配置块中定义。

#### sendfile on;

是否调用sendfile函数传输文件，默认为off，使用sendfile函数传输，可以减少user mode和kernel mode的切换，从而提升服务器性能。

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。

#### sendfile\_max\_chunk 128k; //sendfile设置成on,这里才生效

该参数限定Nginx worker process每次调用sendfile()函数传输数据的最大值，默认值为0，如果设置为0则无限制。

#### tcp\_nopush on;

当tcp\_nopush设置为**on**时，会调用tcp\_cork方法进行数据传输。

使用该方法会产生这样的效果：当应用程序产生数据时，内核不会立马封装包，而是当数据量积累到一定量时才会封装，然后传输。这样有助于解决网络堵塞问题。

默认值为**on**。举例：快递员收快递、发快递，包裹累积到一定量才会发，节省运输成本。

#### keepalive\_timeout 65 60;

该参数有两个值，第一个值设置nginx服务器与客户端会话结束后仍旧保持连接的最长时间，单位是秒，默认为75s。

第二个值可以省略，它是针对客户端的浏览器来设置的，可以通过curl -I看到header信息中有一项Keep-Alive: **timeout**=60，如果不设置就没有这一项。

第二个数值设置后，浏览器就会根据这个数值决定何时主动关闭连接，Nginx服务器就不操心了。但有的浏览器并不认可该参数。

#### send\_timeout

这个超时时间是发送响应的超时时间，即Nginx服务器向客户端发送了数据包，但客户端一直没有去接收这个数据包。

如果某个连接超过send\_timeout定义的超时时间，那么Nginx将会关闭这个连接。

#### client\_max\_body\_size 10m;

浏览器在发送含有较大HTTP包体的请求时，其头部会有一个Content-**Length**字段，client\_max\_body\_size是用来限制Content-**Length**所示值的大小的。

这个限制包体的配置不用等Nginx接收完所有的HTTP包体，就可以告诉用户请求过大不被接受。会返回413状态码。

例如，用户试图上传一个1GB的文件，Nginx在收完包头后，发现Content-**Length**超过client\_max\_body\_size定义的值，

就直接发送413(Request Entity Too **Large**)响应给客户端。

#### gzip on；

是否开启gzip压缩。

#### gzip\_min\_length 1k;

设置允许压缩的页面最小字节数，页面字节数从header头得content-length中进行获取。默认值是20。建议设置成大于1k的字节数，小于1k可能会越压越大。

#### gzip\_buffers 4 16k;

设置系统获取几个单位的buffer用于存储gzip的压缩结果数据流。4 16k代表分配4个16k的buffer。

#### gzip\_http\_version 1.1;

用于识别 **http** 协议的版本，早期的浏览器不支持 Gzip 压缩，用户会看到乱码，所以为了支持前期版本加上了这个选项。

如果你用了Nginx反向代理并期望也启用Gzip压缩的话，由于末端通信是**http**/1.1，故请设置为 1.1。

#### gzip\_comp\_level 6;

gzip压缩比，1压缩比最小处理速度最快，9压缩比最大但处理速度最慢(传输快但比较消耗cpu)

### gzip\_types mime-type ... ;

匹配mime类型进行压缩，无论是否指定,”text/html”类型总是会被压缩的。

在conf/mime.conf里查看对应的type。

示例：gzip\_types text/plain application/x-javascript text/css text/html application/xml;

#### gzip\_proxied any;

Nginx作为反向代理的时候启用，决定开启或者关闭后端服务器返回的结果是否压缩，匹配的前提是后端服务器必须要返回包含”Via”的 **header**头。

以下为可用的值：

off - 关闭所有的代理结果数据的压缩

expired - 启用压缩，如果**header**头中包含 "Expires" 头信息

no-cache - 启用压缩，如果**header**头中包含 "Cache-Control:no-cache" 头信息

no-store - 启用压缩，如果**header**头中包含 "Cache-Control:no-store" 头信息

**private** - 启用压缩，如果**header**头中包含 "Cache-Control:private" 头信息

no\_last\_modified - 启用压缩,如果**header**头中不包含 "Last-Modified" 头信息

no\_etag - 启用压缩 ,如果**header**头中不包含 "ETag" 头信息

auth - 启用压缩 , 如果**header**头中包含 "Authorization" 头信息

any - 无条件启用压缩

#### gzip\_vary on；

和http头有关系，会在响应头加个 Vary: **Accept**-Encoding ，可以让前端的缓存服务器缓存经过gzip压缩的页面，例如，用Squid缓存经过Nginx压缩的数据。

7、server（虚拟主机相关）

### nginx.conf server部分配置

**server**{} 包含在**http**{}内部，每一个**server**{}都是一个虚拟主机（站点）。

#### 以下为nginx.conf配置文件中server{}部分的内容。

server {

listen 80; *//监听端口为80，可以自定义其他端口，也可以加上IP地址，如，listen 127.0.0.1:8080;*

server\_name localhost; *//定义网站域名，可以写多个，用空格分隔。*

**#charset koi8-r; *//定义网站的字符集，一般不设置，而是在网页代码中设置。***

**#access\_log logs/host.access.log main; *//定义访问日志，可以针对每一个server（即每一个站点）设置它们自己的访问日志。***

**##在server{}里有很多location配置段**

location / {

root html; *//定义网站根目录，目录可以是相对路径也可以是绝对路径。*

index index.html index.htm; *//定义站点的默认页。*

}

**#error\_page 404 /404.html; *//定义404页面***

**# redirect server error pages to the static page /50x.html**

**#**

error\_page 500 502 503 504 /50x.html; *//当状态码为500、502、503、504时，则访问50x.html*

location = /50x.html {

root html; *//定义50x.html所在路径*

}

**# proxy the PHP scripts to Apache listening on 127.0.0.1:80**

**#**

**#定义访问php脚本时，将会执行本location{}部分指令**

**#location ~ \.php$ {**

**# proxy\_pass http:*//127.0.0.1; //proxy\_pass后面指定要访问的url链接，用proxy\_pass实现代理。***

**#}**

**# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000**

**#**

**#location ~ \.php$ {**

**# root html;**

**# fastcgi\_pass 127.0.0.1:9000; *//定义FastCGI服务器监听端口与地址，支持两种形式，1 IP:Port， 2 unix:/path/to/sockt***

**# fastcgi\_index index.php;**

**# fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name; *//定义SCRIPT\_FILENAME变量，后面的路径/scripts为上面的root指定的目录***

**# include fastcgi\_params; //引用prefix/conf/fastcgi\_params文件，该文件定义了fastcgi相关的变量**

**#}**

**# deny access to .htaccess files, if Apache's document root**

**# concurs with nginx's one**

**#**

**#location ~ /\.ht { *//访问的url中，以/.ht开头的，如，www.example.com/.htaccess，会被拒绝，返回403状态码。***

**# deny all; *//这里的all指的是所有的请求。***

**#}**

}

**# another virtual host using mix of IP-, name-, and port-based configuration**

**#**

**#server {**

**# listen 8000; *//监听8000端口***

**# listen somename:8080; *//指定ip:port***

**# server\_name somename alias another.alias; *//指定多个server\_name***

**# location / {**

**# root html;**

**# index index.html index.htm;**

**# }**

**#}**

**# HTTPS server**

**#**

**#server {**

**# listen 443 ssl; *//监听443端口，即ssl***

**# server\_name localhost;**

**### 以下为ssl相关配置**

**# ssl\_certificate cert.pem; *//指定pem文件路径***

**# ssl\_certificate\_key cert.key; *//指定key文件路径***

**# ssl\_session\_cache shared:SSL:1m; *//指定session cache大小***

**# ssl\_session\_timeout 5m; *//指定session超时时间***

**# ssl\_protocols TLSv1 TLSv1.1 TLSv1.2; *//指定ssl协议***

**# ssl\_ciphers HIGH:!aNULL:!MD5; *//指定ssl算法***

**# ssl\_prefer\_server\_ciphers on; *//优先采取服务器算法***

**# location / {**

**# root html;**

**# index index.html index.htm;**

**# }**

**#}**

8、location（server里面）

**nginx架构分析**

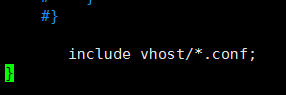
**nginx虚拟主机配置**

1、cd /usr/local/nginx/conf/ //进入该目录编辑主配置文件

2、vim nginx.conf //编辑主配置文件

禁掉http里面的server部分，在最后添加虚拟主机配置

include vhost/\*.conf;



vhost/ //是相对与主配置文件目录来的，所有在/usr/local/nginx/conf/ 目录里面创建vhost目录

3、cd /usr/local/nginx/conf

4、mkdir vhost

5、cd vhost

6、vim www.1.conf

server {

listen 80;

server\_name www.1.com;

root /data/wwwroot/www.1.com; //网站根目录

}

vim www.2.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

}

7、创建网站根目录

mkdir -p /data/wwwroot/www.1.com

mkdir /data/wwwroot/www.2.com

8、cd /data/wwwroot/www.1.com

vim index.html

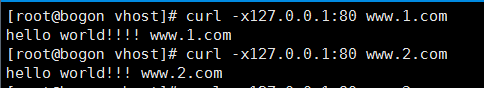
hello world!!!! www.1.com

9、cd /data/wwwroot/www.2.com

vim index.html

hello world!!!! www.2.com

10、测试



系统并未指定www.3.com,但默认会访问到第一个站点，及第一个站点为默认虚拟主机



12、定义默认虚拟主机/usr/local/nginx/conf/vhost

vim default.conf

server {

listen 80 default\_server; //default\_server 指定为默认虚拟主机

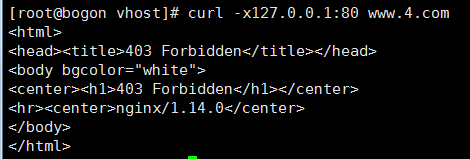
deny all;

}

/usr/local/nginx/sbin/nginx -t //检查配置文件有无错误

/usr/local/nginx/sbin/nginx -s reload

再次测试那些未指定的域名，就直接403了



13、vim www.1.conf

server {

listen 80;

server\_name www.1.com;

index 1.html; //指定了index（索引页）为 1.html

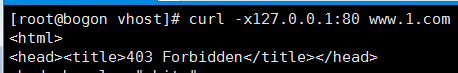
root /data/wwwroot/www.1.com;

}

/usr/local/nginx/sbin/nginx -t //检查配置文件有无错误

/usr/local/nginx/sbin/nginx -s reload

curl -x127.0.0.1:80 www.1.com



curl -x127.0.0.1:80 www.1.com/index.html



14、在/data/wwwroot/www.1.com 编辑1.html

vim 1.html

wo s 111

curl -x127.0.0.1:80 www.1.com



15、cd /usr/local/nginx/conf/vhost

vim aminglinux.com.conf

server {

listen 80;

server\_name \*.aminglinux.com; //泛解析

root /data/wwwroot/aminglinux.com;

index index.html;

}

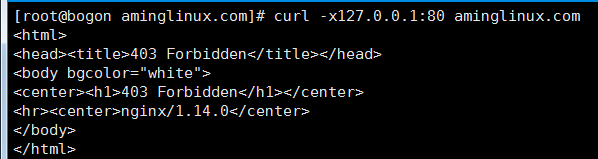
16、mkdir /data/wwwroot/aminglinux.com

vim index.html

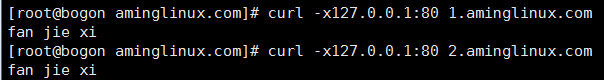


curl -x127.0.0.1:80 aminglinux.com

curl -x127.0.0.1:80 aminglinux.com/index.html //也是一样的



和定义的server\_name有关系



vim aminglinux.com.conf

server {

listen 80;

server\_name aminglinux.com \*.aminglinux.com; //泛解析

root /data/wwwroot/aminglinux.com;

index index.html;

}

/usr/local/nginx/sbin/nginx -t //检查配置文件有无错误

/usr/local/nginx/sbin/nginx -s reload



16访问不同的端口，但域名相同

cd /usr/local/nginx/conf/vhost/

cp www.1.com.conf www.1.com2.conf

vim www.1.com2.conf

server {

listen 8080; //端口不同

server\_name www.1.com; //域名相同

index index.html;

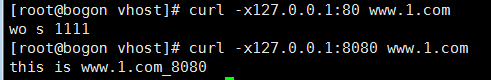
root /data/wwwroot/www.1.com\_8080;

}

mkdir /data/wwwroot/www.1.com\_8080

vim /data/wwwroot/www.1.com\_8080/index.html

/usr/local/nginx/sbin/nginx -s reload



**nginx的rewrite配置**

#### if指令

格式：**if** (条件判断) { 具体的rewrite规则 }

##### 条件举例

条件判断语句由Nginx内置变量、逻辑判断符号和目标字符串三部分组成。

其中，内置变量是Nginx固定的非自定义的变量，如，$request\_method, $request\_uri等。

逻辑判断符号，有=, !=, ~, ~\*（匹配不区分大小写）, !~, !~\*

!表示相反的意思，~为匹配符号，它右侧为正则表达式，区分大小写，而~\*为不区分大小写匹配。

目标字符串可以是正则表达式，通常不用加引号，但表达式中有特殊符号时，比如空格、花括号、分号等，需要用单引号引起来。

##### 示例1

**if** ($request\_method = **POST**) *//当请求的方法为POST时，直接返回405状态码*

{

**return** 405; *//在该示例中并未用到rewrite规则，if中支持用return指令。*

}

##### 示例2

**if** ($http\_user\_agent ~ MSIE) *//user\_agent带有MSIE字符的请求，直接返回403状态码*

{

**return** 403*;*

}

如果想同时限制多个user\_agent，还可以写成这样

**if** ($http\_user\_agent ~ "MSIE|firefox|spider")

{

**return** 403*;*

}

##### 示例3

**if**(!-f $request\_filename) //当请求的文件不存在，将会执行下面的rewrite规则

{

rewrite 语句;

}

##### 示例4

**if**($request\_uri ~\* 'gid=\d{9,12}/') //\d表示数字，{9,12}表示数字出现的次数是9到12次，如gid=123456789/就是符合条件的。

{

rewrite 语句;

}

cd /usr/local/nginx/conf/vhost

1、vim www.1.com.conf

server {

listen 80;

server\_name www.1.com;

index 1.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

rewrite /1.html /2.html;

rewrite /2.html /3.html; //这2条访问1.html会跳到3.html;1-->2-->3

}

2、vim nginx.conf



3、cd /data/wwwroot/www.1.com

vim 1.html

111

vim 2.html

222

vim 3.html

3333

4、重启nginx服务

5、curl -x127.0.0.1:80 www.1.com/1.html



6、vim www.1.com.conf

server {

listen 80;

server\_name www.1.com;

index 1.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

rewrite /1.html /2.html break;

rewrite /2.html /3.html; //这条访问1.html会跳到2.html;1-->2结束

}

7、重启服务

8、curl -x127.0.0.1:80 www.1.com/1.html



9、vim www.1.com.conf

server {

listen 80;

server\_name www.1.com;

index 1.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

rewrite /1.html /2.html last;

rewrite /2.html /3.html; //这条访问1.html会跳到2.html;1-->2结束

}

10、重启服务

11、curl -x127.0.0.1:80 www.1.com/1.html



12、vim www.1.com.conf

server {

listen 80;

server\_name www.1.com;

index 1.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

location / {

rewrite /1.html /2.html; //1

rewrite /2.html /3.html; //2

}

location /2.html

{

rewrite /2.html /a.html;

}

location /3.html //3

{

rewrite /3.html /b.html; //4

}

}

1-->2-->3-->4

匹配备注：当请求/1.html，最终将会访问/b.html，连续执行**location** **/下的两次rewrite**，跳转到了/3.html，然后又匹配**location** **/3**.html

13、vim www.1.com.conf

server {

listen 80;

server\_name www.1.com;

index 1.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

location / {

rewrite /1.html /2.html break; //1

rewrite /2.html /3.html;

}

location /2.html

{

rewrite /2.html /a.html;

}

location /3.html

{

rewrite /3.html /b.html;

}

}

就执行1，break后面的不执行。

当请求/1.html，最终会访问/2.html

在location{}内部，遇到break，本location{}内以及后面的所有location{}内的所有指令都不再执行。

14、vim www.1.com.conf

server {

listen 80;

server\_name www.1.com;

index 1.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

location / {

rewrite /1.html /2.html last; //1

rewrite /2.html /3.html;

}

location /2.html //2

{

rewrite /2.html /a.html; //3

}

location /3.html

{

rewrite /3.html /b.html;

}

}

请求/1.html，最终会访问/a.html

在location{}内部，遇到last，本location{}内后续指令不再执行，而重写后的url再次从头开始，从头到尾匹配一遍规则。

https://coding.net/u/aminglinux/p/nginx/git/blob/master/rewrite/break.md

**nginx中的return用法**

1、vim www.1.com.conf //匹配if 里面的返回405

server {

listen 80;

server\_name www.1.com;

index 1.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

if ( $request\_uri ~ "\.htpasswd|\.bak" )

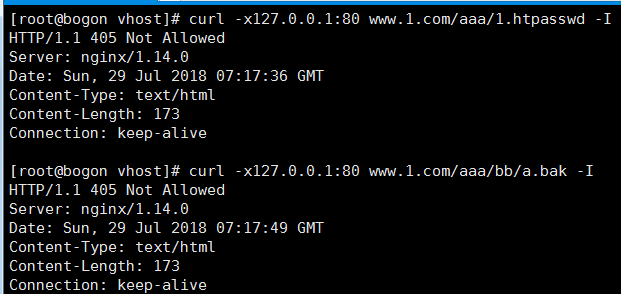
{

return 405;

rewrite /(.\*) /aaa.txt; #本行不执行

}

}



#### nginx的return指令

该指令一般用于对请求的客户端直接返回响应状态码。在该作用域内**return**后面的所有nginx配置都是无效的。

可以使用在**server**、location以及**if**配置中。

除了支持跟状态码，还可以跟字符串或者url链接。

Copy

##### 直接返回状态码

示例1：

server{

**listen** 80;

server\_name www.aming.com;

return 403;

rewrite /(.\*) /abc/$1; *//该行配置不会被执行。*

}

示例2：

server {

.....

if ($request\_uri ~ "\.htpasswd|\.bak")

{

return 404;

rewrite /(.\*) /aaa.txt; *//该行配置不会被执行。*

}

*//如果下面还有其他配置，会被执行。*

.....

}

##### 返回字符串

示例3：

server{

listen 80;

server\_name www.aming.com;

return 200 "hello";

}

说明：如果要想返回字符串，必须要加上状态码，否则会报错。

还可以支持json数据

示例4：

**location** **^~ /aming** {

default\_type application/json ;

return 200 '{"name":"aming","id":"100"}';

}

也支持写一个变量

示例5：

**location** **/test** {

return 200 "$host $request\_uri";

}

##### 返回url

示例6：

server{

listen 80;

server\_name www.aming.com;

**return** http:*//www.aminglinux.com/123.html;*

rewrite /(.\*) /abc/$1; //该行配置不会被执行。

}

注意：**return**后面的url必须是以http:*//或者https://开头的。*

##### 生成场景实战

背景：网站被黑了，凡是在百度点击到本网站的请求，全部都跳转到了一个赌博网站。

通过nginx解决：

**if** ($http\_referer ~ 'baidu.com')

{

return 200 "<html><script>window.location.href='//$host$request\_uri';</script></html>";

}

如果写成：

return http://$host$request\_uri; 在浏览器中会提示“重定向的次数过多”。

server {

listen 80;

server\_name www.1.com;

index index.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

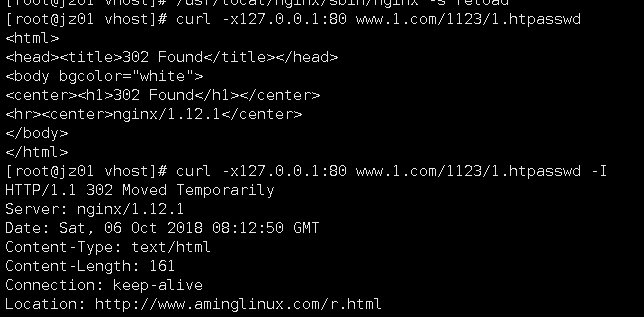
if ( $request\_uri ~ "\.htpasswd|\.bak" )

{

return http://www.aminglinux.com/r.html;

}

}



server {

listen 80;

server\_name www.1.com;

index index.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

if ( $request\_uri ~ "\.htpasswd|\.bak" )

{

return 301 http://www.aminglinux.com/r.html;

}

}



server {

listen 80;

server\_name www.1.com;

index index.html;

root /data/wwwroot/www.1.com;

rewrite\_log on;

if ( $request\_uri ~ "\.htpasswd|\.bak" )

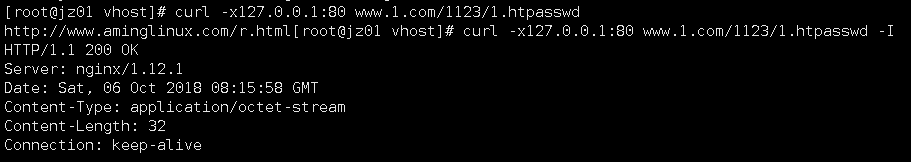
{

return 200 http://www.aminglinux.com/r.html;

}

}

//状态码加200是不会跳转的，只返回字符串



**rewrite规则语法**

1、cd /usr/local/nginx/conf/vhost

vim www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

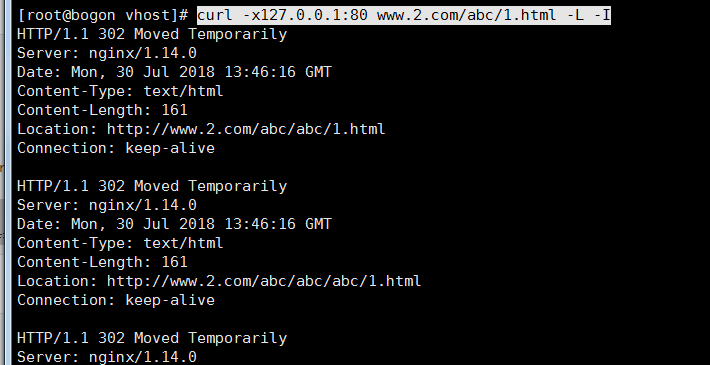
rewrite /(.\*) /abc/$1 redirect; $1是前面的“（.\*）”

}

说明：本例中的rewrite规则有问题，会造连续循环，最终会失败，解决该问题有两个方案。

关于循环次数，经测试发现，curl 会循环50次，chrome会循环80次，IE会循环120次，firefox会循环20次。

curl -x127.0.0.1:80 www.2.com/abc/1.html -L -I

****

2、解决上面的循环问题

cd /usr/local/nginx/conf/vhost

vim www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

rewrite /(.\*) /abc/$1 break;

}

3、解决循环问题方法2

vim www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

if ($request\_uri !~ '^/abc/') //只要不匹配www.2.com/abc/……

{

rewrite /(.\*) /abc/$1 redirect; //就跳转

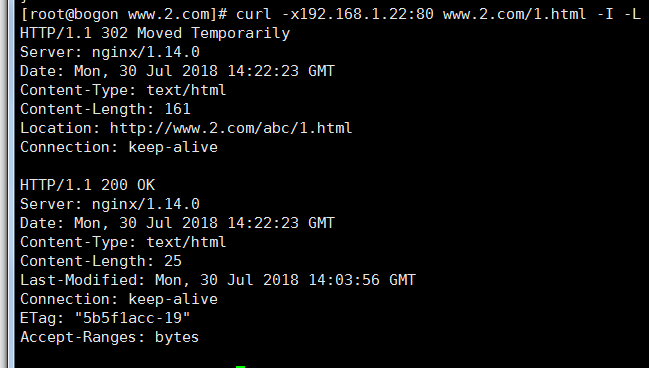
}

}

如果你访问的是www.2.com/1.html会跳转到www.2.com/abc/1.html

request\_uri:除开域名以外后面的东西

www.2.com/abc/1.html // /abc/1.html为 request\_uri



#### rewrite规则

格式：rewrite regex replacement [flag]

\* rewrite配置可以在server、location以及**if**配置段内生效

\* regex是用于匹配URI的正则表达式，其不会匹配到$host（域名）

\* replacement是目标跳转的URI，可以以http://或者https://开头，也可以省略掉$host，直接写$request\_uri部分（即请求的链接）

\* flag，用来设置rewrite对URI的处理行为，其中有**break**、last、rediect、permanent，其中**break**和last在前面已经介绍过，

rediect和permanent的区别在于，前者为临时重定向(302)，而后者是永久重定向(301)，对于用户通过浏览器访问，这两者的效果是一致的。

但是，对于搜索引擎蜘蛛爬虫来说就有区别了，使用301更有利于SEO。所以，建议replacemnet是以http://或者https://开头的flag使用permanent。

##### 示例1

location / {

rewrite /(.\*) http://www.aming.com/$1 permanent;

}

说明：.\*为正则表达式，用()括起来，在后面的URI中可以调用它，第一次出现的()用$1调用，第二次出现的()用$2调用，以此类推。；.\*代表后面的$1

##### 示例2

location / {

rewrite /.\* http://www.aming.com$request\_uri permanent;

}

说明：在replacement中，支持变量，这里的$request\_uri就是客户端请求的链接

**nginx全局变量**

https://coding.net/u/aminglinux/p/nginx/git/blob/master/rewrite/variable.md

1、vim www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

return 200 "$args";

2、测试什么是$args，就是？后面的东西

curl -x127.0.0.1:80 'www.2.com/abc/1.php?a=1&b=2'



https://coding.net/u/aminglinux/p/nginx/git/blob/master/rewrite/variable.md

**rewrite实战**

https://coding.net/u/aminglinux/p/nginx/git/blob/master/rewrite/example.md

1、重定向

vim www.2.com.conf

server {

listen 80;

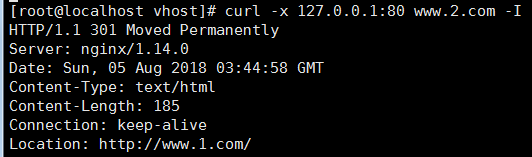
server\_name www.2.com;

root /data/wwwroot/www.2.com;

rewrite /(.\*) http://www.1.com/$1 permanent;

access\_log /tmp/2.log;

}



2、多域名跳转

server {

listen 80;

server\_name www.2.com 2.com;

root /data/wwwroot/www.2.com;

if ($host != www.2.com)

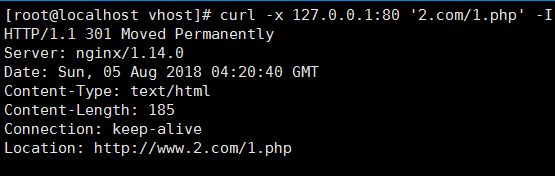
{

rewrite /(.\*) http://www.2.com/$1 permanent;

}

access\_log /tmp/2.log;

}



3、http跳的https

[root@localhost vhost]# cat www.2.com.conf

server {

listen 80;

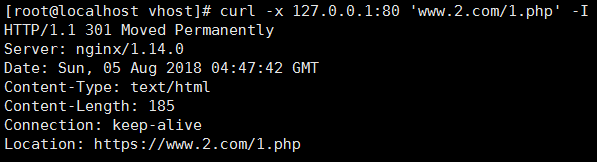
server\_name www.2.com;

root /data/wwwroot/www.2.com;

rewrite /(.\*) https://www.2.com/$1 permanent;

access\_log /tmp/2.log;

}



4、域名访问二级目录

cat www.2.com.conf

server {

listen 80;

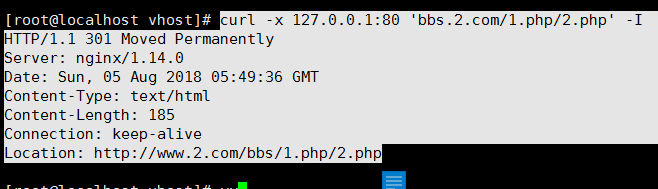
server\_name bbs.2.com;

root /data/wwwroot/www.2.com;

rewrite /(.\*) http://www.2.com/bbs/$1 permanent;

access\_log /tmp/2.log;

}



5、静态请求分离

[root@localhost vhost]# cat www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

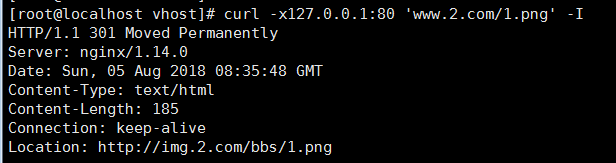
if ($uri ~\* (jpg|gif|jepg|png|css|xml|js|bmp|swf|mp3)$)

{

rewrite /(.\*) http://img.2.com/bbs/$1 permanent;

}

}



6、防盗链

vim www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location ~\* ^.+.(jpg|jpeg|gif|css|png|js|rar|zip|flv)$

{

valid\_referers none blocked server\_names \*.2.com \*.aming.com aminglinux.com aming.com;

if ($invalid\_referer) //黑名单refer

{

return 403;

}

}

}

valid\_referers:白名单referer none//空refere, blocked指的是referer头部的值被防火墙或者代理服务器删除或者伪装的情况，该情况下，referer头部的值不以http://或者https://开头



7、cat /usr/local/nginx/conf/vhost/www.2.com.conf //多条件判断

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location ~\* ^.+.(jpg|jpeg|gif|css|png|js|rar|zip|flv)$

{

valid\_referers server\_names \*.2.com \*.aming.com aminglinux.com aming.com;

if ($invalid\_referer)

{

return 403;

}

}

set $a 0;

if ($request\_uri !~ "^/abc/")

{

set $a "${a}1"; //如果满足上面的if条件这里的$a就是01，不满足为0

}

if ($http\_user\_agent ~ 'IE|chrome')

{

set $a "${a}2"; //如果第一个if满足、第二个if也满足，$a为012

}

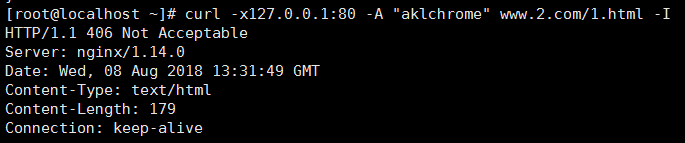
if ($a = "012")

{

return 406;

}

}



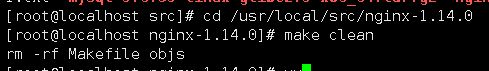
**nginx的location配置**

1、安装第三方模块echo-nginx-module

git clone https://github.com/openresty/echo-nginx-module.git

2、cd /usr/local/src/nginx-1.14.0 //进入之前的源码包

make clean



3、/usr/local/nginx/sbin/nginx -V //查看nginx源码编译参数

4、./configure --prefix=/usr/local/nginx --add-module=/usr/local/src/echo-nginx-module //重新编译之前的参数，并且加上

--add后面的东西

5、make && make install

6、/etc/init.d/nginx restart

7、cat /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location /abc/

{

echo hello;

}

}

8、/usr/local/nginx/sbin/nginx -t //检查配置文件有无错误

/usr/local/nginx/sbin/nginx -s reload

这个只能匹配到 www.2.com/abc/xxxx,



10、cat /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location /abc/

{

echo "/";

}

location ~\* abc //这个只要匹配到“abc”关键字就可以

{

echo "~\*";

}

}

~\* 的优先级高于 / 所以匹配



11、cat /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location ~ /abc/

{

echo "hello";

}

}

这个是可以匹配到www.2.com/xxx/abc/,主要匹配 “/abc/”



12、cat /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location ~ abc

{

echo "~";

}

location ~\* abc

{

echo "~\*";

}

}

“~” 和 “~\*”的优先级相同，但是谁在前面就匹配谁



13、cat /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location ~\* abc

{

echo "~\*";

}

location ^~ /abc

{

echo "^~";

}

}

"^~"的优先级高于"~\*"



14、cat /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location ^~ /abc

{

echo "^~";

}

location = "/abc/1.html"

{

echo "=";

}

}



**nginx正向代理**

1、正常百度没有解析到本地



2、cd /usr/local/nginx/conf/vhost/zp.conf

mv default.conf default.conf.1

vim zp.conf

server {

listen 80 default\_server;

resolver 119.29.29.29;

location /

{

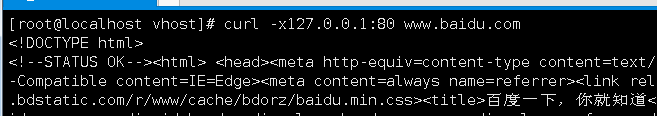
proxy\_pass http://$host$request\_uri;

}

}

/usr/local/nginx/sbin/nginx -t //检查配置文件有无错误

/usr/local/nginx/sbin/nginx -s reload



**Nginx反向代理配置**

1、cd /usr/local/nginx/conf/vhost

vim fp.conf

server

{

listen 80;

server\_name www.test.com;

location /

{

proxy\_pass http://127.0.0.1:8080/; //该IP为代理那个服务器的IP,就是www.test.com的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;

}

}

2、vim test.com.conf

server

{

listen 8080 default\_server;

server\_name www.test.com;

root /data/wwwroot/test.com;

index index.html;

}

3、mkdir /data/wwwroot/test.com

4、cd !$

vim index.html

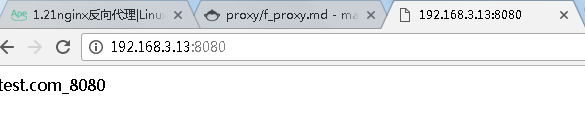
cat index.html

test.com\_8080

5、测试



192.168.3.13是虚拟机的IP地址



6、cat default\_8080.com.conf

server

{

listen 8080 default\_server;

root /data/wwwroot/test.com;

index index.html;

location /

{

echo "8080 default";

}

}

7、cat test.com.conf //更改第2点的结果

server

{

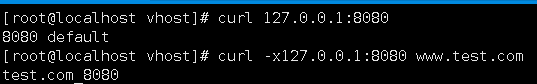
listen 8080;

server\_name www.test.com;

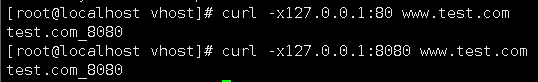
root /data/wwwroot/test.com;

index index.html;

}



访问80和8080一样

****

vim fp.conf

server

{

listen 80;

server\_name www.test.com;

location /

{

proxy\_pass http://127.0.0.1:8080/; //如果这里写的是不同于“www.test.com”的域名，就不要加下面的header 信息，但是要做域名解析

# proxy\_set\_header Host $host;

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

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

}

}

更改配置后需要重新加载服务



**nginx反向代理-proxy\_pass**

1、cat fp.conf

server

{

listen 80;

server\_name www.test.com;

location /aming/

{

proxy\_pass http://127.0.0.1:8080/;

# proxy\_set\_header Host $host;

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

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

}

}

2、cat default\_8080.com.conf

server

{

listen 8080 default\_server;

root /data/wwwroot/test.com;

index index.html;

location /aming/

{

echo "8080 default aming";

}

location /linux/

{

echo "8080 default linux";

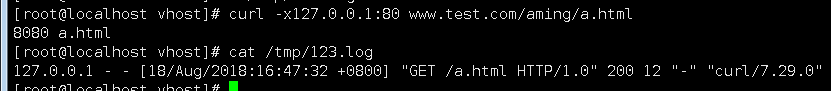
}

access\_log /tmp/123.log;

}

3、cat /data/wwwroot/test.com/a.html

8080 a.html



4、将fp.conf

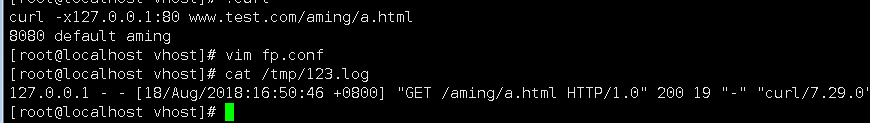
proxy\_pass http://127.0.0.1:8080/; 更改为

proxy\_pass http://127.0.0.1:8080;

5、重启服务

/usr/local/nginx/sbin/nginx -t

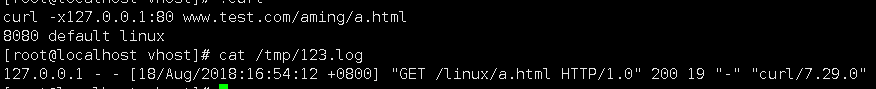
/usr/local/nginx/sbin/nginx -s reload



6、将fp.conf

proxy\_pass http://127.0.0.1:8080/; 更改为

proxy\_pass http://127.0.0.1:8080/linux/;



7、将fp.conf

proxy\_pass http://127.0.0.1:8080/; 更改为

将fp.conf

proxy\_pass http://127.0.0.1:8080/linux;



#### nginx的proxy\_buffering和proxy\_cache

两个都是nginx代理中内存设置相关的参数。

Copy

##### proxy\_buffering设置

proxy\_buffering主要是实现被代理服务器的数据和客户端的请求异步。

为了方便理解，我们定义三个角色，A为客户端，**B为代理服务器，C为被代理服务器。**

当proxy\_buffering开启，A发起请求到**B，B再到C，C反馈的数据先到B的buffer上，**

然后**B会根据proxy\_busy\_buffer\_size来决定什么时候开始把数据传输给A。在此过程中，如果所有的buffer被写满，**

数据将会写入到temp\_file中。

相反，如果proxy\_buffering关闭，C反馈的数据实时地通过**B传输给A。**

###### 以下配置，都是针对每一个http请求的。

1. proxy\_buffering on;

该参数设置是否开启proxy的buffer功能，参数的值为on或者off。

如果这个设置为off，那么proxy\_buffers和proxy\_busy\_buffers\_size这两个指令将会失效。

但是无论proxy\_buffering是否开启，proxy\_buffer\_size都是生效的

2. proxy\_buffer\_size 4k;

该参数用来设置一个特殊的buffer大小的。

从被代理服务器（C）上获取到的第一部分响应数据内容到代理服务器（B）上，通常是header，就存到了这个buffer中。

如果该参数设置太小，会出现502错误码，这是因为这部分buffer不够存储header信息。建议设置为4k。

3. proxy\_buffers 8 4k;

这个参数设置存储被代理服务器上的数据所占用的buffer的个数和每个buffer的大小。

所有buffer的大小为这两个数字的乘积。

4. proxy\_busy\_buffer\_size 16k;

在所有的buffer里，我们需要规定一部分buffer把自己存的数据传给A，这部分buffer就叫做busy\_buffer。

proxy\_busy\_buffer\_size参数用来设置处于busy状态的buffer有多大。

对于B上buffer里的数据何时传输给A，我个人的理解是这样的：

1）如果完整数据大小小于busy\_buffer大小，当数据传输完成后，马上传给A；

2）如果完整数据大小不少于busy\_buffer大小，则装满busy\_buffer后，马上传给A；

5. proxy\_temp\_path

语法：proxy\_temp\_path path [level1 level2 level3]

定义proxy的临时文件存在目录以及目录的层级。

例：proxy\_temp\_path /usr/local/nginx/proxy\_temp 1 2;

其中/usr/local/nginx/proxy\_temp为临时文件所在目录，1表示层级1的目录名为一个数字(0-9),2表示层级2目录名为2个数字(00-99)

6. proxy\_max\_temp\_file\_size

设置临时文件的总大小，例如 proxy\_max\_temp\_file\_size 100M;

7. proxy\_temp\_file\_wirte\_size

设置同时写入临时文件的数据量的总大小。通常设置为8k或者16k。

https://coding.net/u/aminglinux/p/nginx/git/blob/master/proxy/bu\_ca.md

#### Nginx的负载均衡配置

Nginx通过upstream和proxy\_pass实现了负载均衡。本质上也是Nginx的反向代理功能，只不过后端的**server**为多个。

##### 案例一（简单的轮询）

upstream www {

server 172.37**.150.109**:80*;*

server 172.37**.150.101**:80*;*

server 172.37**.150.110**:80*;*

}

server {

listen 80*;*

server\_name www.aminglinux.com*;*

location / {

proxy\_pass http://www/*;*

proxy\_set\_header Host $host*;*

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

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

}

}

说明：当被代理的机器有多台时，需要使用upstream来定义一个服务器组，

其中www名字可以自定义，在后面的proxy\_pass那里引用。

这样nginx会将请求均衡地轮询发送给www组内的三台服务器。

Copy

##### 案例二（带权重轮询+ip\_hash算法）

upstream www {

server 172.37**.150.109**:80 weight=50*;*

server 172.37**.150.101**:80 weight=100*;*

server 172.37**.150.110**:80 weight=50*;*

ip\_hash*;*

}

server {

listen 80*;*

server\_name www.aminglinux.com*;*

location / {

proxy\_pass http://www/*;*

proxy\_set\_header Host $host*;*

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

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

}

}

说明：可以给www组内的三台机器配置权重，权重越高，则分配到的请求越多。

ip\_hash为nginx负载均衡算法，原理很简单，它根据请求所属的客户端IP计算得到一个数值，然后把请求发往该数值对应的后端。

所以同一个客户端的请求，都会发往同一台后端，除非该后端不可用了。ip\_hash能够达到保持会话的效果。

##### 案例三（upstream其他配置）

upstream www {

**server** 172.37.150.109:80 weight=50 max\_fails=3 fail\_timeout=30s;

**server** 172.37.150.101:80 weight=100;

**server** 172.37.150.110:80 down;

**server** 172.37.150.110:80 backup;

}

**server**

{

listen 80;

server\_name www.aminglinux.com;

location / {

proxy\_next\_upstream off;

proxy\_pass http:*//www/;*

proxy\_set\_header Host $host;

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

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

}

}

说明：down，表示当前的**server**不参与负载均衡；

backup，为预留的机器，当其他的**server**（非backup）出现故障或者忙的时候，才会请求backup机器;

max\_fails，允许请求失败的次数，默认为1。当失败次数达到该值，就认为该机器down掉了。 失败的指标是由proxy\_next\_upstream模块定义，其中404状态码不认为是失败。

fail\_timeount，定义失败的超时时间，也就是说在该时间段内达到max\_fails，才算真正的失败。默认是10秒。

proxy\_next\_upstream，通过后端服务器返回的响应状态码，表示服务器死活，可以灵活控制后端机器是否加入分发列表。

语法: proxy\_next\_upstream error | timeout | invalid\_header | http\_500 | http\_502 | http\_503 | http\_504 |http\_404 | off ...;

默认值: proxy\_next\_upstream error timeout

error **# 和后端服务器建立连接时，或者向后端服务器发送请求时，或者从后端服务器接收响应头时，出现错误**

timeout **# 和后端服务器建立连接时，或者向后端服务器发送请求时，或者从后端服务器接收响应头时，出现超时**

invalid\_header **# 后端服务器返回空响应或者非法响应头**

http\_500 **# 后端服务器返回的响应状态码为500**

http\_502 **# 后端服务器返回的响应状态码为502**

http\_503 **# 后端服务器返回的响应状态码为503**

http\_504 **# 后端服务器返回的响应状态码为504**

http\_404 **# 后端服务器返回的响应状态码为404**

off **# 停止将请求发送给下一台后端服务器**

##### 案例四（根据不同的uri）

upstream aa.com {

server 192.168.0.121;

server 192.168.0.122;

}

upstream bb.com {

server 192.168.0.123;

server 192.168.0.124;

}

server {

listen 80;

server\_name www.aminglinux.com;

location ~ aa.php

{

proxy\_pass http:*//aa.com/;*

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 ~ bb.php

{

proxy\_pass http:*//bb.com/;*

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 /

{

proxy\_pass http:*//bb.com/;*

proxy\_set\_header Host $host;

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

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

}

}

说明：请求aa.php的，会到aa.com组，请求bb.php的会到bb.com，其他请求全部到bb.com。

##### 案例五（根据不同的目录）

upstream aaa.com

{

server 192.168.111.6;

}

upstream bbb.com

{

server 192.168.111.20;

}

**server** {

listen 80;

server\_name www.aminglinux.com;

location /aaa/

{

proxy\_pass http://aaa.com/aaa/;

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 /bbb/

{

proxy\_pass http://bbb.com/bbb/;

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 /

{

proxy\_pass http://bbb.com/;

proxy\_set\_header Host $host;

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

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

}

}

#### Nginx访问控制 —— deny\_allow

Nginx的deny和allow指令是由ngx\_http\_access\_module模块提供，Nginx安装默认内置了该模块。

除非在安装时有指定 *--without-http\_access\_module。*

Copy

##### 语法

语法：allow/deny address | **CIDR** | **unix**: | **all**

它表示，允许/拒绝某个ip或者一个ip段访问.如果指定unix:,那将允许socket的访问。

注意：unix在1.5.1中新加入的功能。

在nginx中，allow和deny的规则是按顺序执行的。

##### 示例

示例1：

location /

{

allow 192.168.0.0/24;

allow 127.0.0.1;

deny all;

}

说明：这段配置值允许192.168.0.0/24网段和127.0.0.1的请求，其他来源IP全部拒绝。

示例2：

location ~ "admin"

{

allow 110.21.33.121;

deny all

}

说明：访问的uri中包含admin的请求，只允许110.21.33.121这个IP的请求。

#### 基于location的访问控制

在生产环境中，我们会对某些特殊的请求进行限制，比如对网站的后台进行限制访问。

这就用到了**location**配置。

Copy

##### 示例1

**location** **/aming**/

{

**deny** all;

}

说明：针对/aming/目录，全部禁止访问，这里的**deny** all可以改为return 403.

##### 示例2

location ~ ".bak|\.ht"

{

return 403;

}

说明：访问的uri中包含.bak字样的或者包含.ht的直接返回403状态码。

测试链接举例：

1. www.aminglinux.com/123.bak

2. www.aminglinux.com/aming/123/.htalskdjf

##### 示例3

location ~ (data|**cache**|tmp|image|attachment).\*\.php$

{

deny all;

}

说明：请求的uri中包含data、**cache**、tmp、image、attachment并且以.php结尾的，全部禁止访问。

测试链接举例：

1. www.aminglinux.com/aming/**cache**/1.php

2. www.aminglinux.com/image/123.phps

3. www.aminglinux.com/aming/datas/1.php

#### Nginx基于$document\_uri的访问控制

这就用到了变量$document\_uri，根据前面所学内容，该变量等价于$uri，其实也等价于location匹配。

##### 示例1

if ($document\_uri ~ "/admin/")

{

return 403;

}

说明：当请求的uri中包含/admin/时，直接返回403.

if结构中不支持使用allow和deny。

测试链接：

1. www.aminglinux.com/123/admin/1.html 匹配

2. www.aminglinux.com/admin123/1.html 不匹配

3. www.aminglinux.com/admin.php 不匹配

##### 示例2

if ($document\_uri = /admin.php)

{

return 403;

}

说明：请求的uri为/admin.php时返回403状态码。

测试链接：

1. www.aminglinux.com/admin.php 匹配

2. www.aminglinux.com/123/admin.php 不匹配

##### 示例3

**if** ($document\_uri ~ '/data/|/cache/.\*\.php$')

{

**return** 403;

}

说明：请求的uri包含data或者**cache**目录，并且是php时，返回403状态码。

测试链接：

1. www.aminglinux.com/data/123.php 匹配

2. www.aminglinux.com/cache1/123.php 不匹配

#### nginx基于$request\_uri访问控制

$request\_uri比$docuemnt\_uri多了请求的参数。

主要是针对请求的uri中的参数进行控制。

##### 示例

if ($request\_uri ~ "gid=\d{9,12}")

{

return 403;

}

说明：\d{9,12}是正则表达式，表示9到12个数字，例如gid=1234567890就符号要求。

测试链接：

1. www.aminglinux.com/index.php?gid=1234567890&pid=111 匹配

2. www.aminglinux.com/gid=123 不匹配

背景知识：

曾经有一个客户的网站cc攻击，对方发起太多类似这样的请求：/read-123405150-1-1.html

实际上，这样的请求并不是正常的请求，网站会抛出一个页面，提示帖子不存在。

所以，可以直接针对这样的请求，return 403状态码。

#### Nginx基于$user\_agent的访问控制

user\_agent大家并不陌生，可以简单理解成浏览器标识，包括一些蜘蛛爬虫都可以通过user\_agent来辨识。

通过观察访问日志，可以发现一些搜索引擎的蜘蛛对网站访问特别频繁，它们并不友好。

为了减少服务器的压力，其实可以把除主流搜索引擎蜘蛛外的其他蜘蛛爬虫全部封掉。

另外，一些**cc**攻击，我们也可以通过观察它们的user\_agent找到规律。

##### 示例

if ($user\_agent ~ 'YisouSpider|MJ12bot/v1.4.2|YoudaoBot|Tomato')

{

return 403;

}

说明：user\_agent包含以上关键词的请求，全部返回403状态码。

测试：

1. curl -A "123YisouSpider1.0"

2. curl -A "MJ12bot/v1.4.1"

#### Nginx基于$http\_referer的访问控制

在前面讲解rewrite时，曾经用过该变量，当时实现了防盗链功能。

其实基于该变量，我们也可以做一些特殊的需求。

##### 示例

背景：网站被黑挂马，搜索引擎收录的网页是有问题的，当通过搜索引擎点击到网站时，却显示一个博彩网站。

由于查找木马需要时间，不能马上解决，为了不影响用户体验，可以针对此类请求做一个特殊操作。

比如，可以把从百度访问的链接直接返回404状态码，或者返回一段html代码。

**if** ($http\_referer ~ 'baidu.com')

{

return 404;

}

或者

**if** ($http\_referer ~ 'baidu.com')

{

return 200 "<html><script>window.location.href='//$host$request\_uri';</script></html>";

}

**nginx访问控制-限速**

vim /usr/local/nginx/conf/nginx.conf //加入下面的三行

httpd{

limit\_conn\_zone $binary\_remote\_addr zone=aming:10m;

limit\_conn\_status 503; //超过并发数就显示503

limit\_conn\_log\_level error;

...

}

vim /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

location /

{

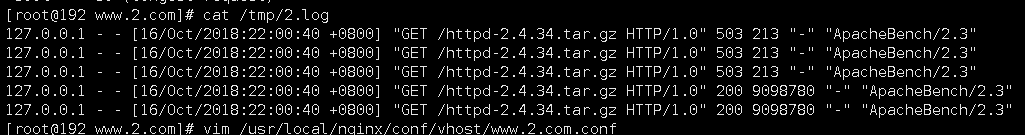
limit\_conn aming 2; //并发数为2

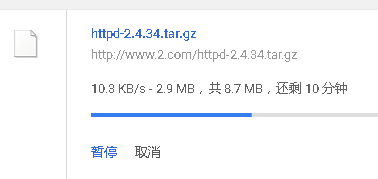
limit\_rate 10k; //下载速度为10K/S

}

}

ab -n 5 -c 5 http://www.2.com/httpd-2.4.34.tar.gz





vim /usr/local/nginx/conf/nginx.conf

http

{

include mime.types;

default\_type application/octet-stream;

limit\_conn\_zone $binary\_remote\_addr zone=aming:10m;

limit\_conn\_status 503;

limit\_conn\_log\_level error;

limit\_req\_zone $binary\_remote\_addr zone=aming1:10m rate=2r/s;

vim /usr/local/nginx/conf/vhost/www.2.com.conf

server {

listen 80;

server\_name www.2.com;

root /data/wwwroot/www.2.com;

access\_log /tmp/2.log;

#location /

#{

# #limit\_conn aming 2;

# limit\_rate 100k;

#}

limit\_req zone=aming1 burst=500 nodelay; //brust同时支持500个请求，nodelay,响应无需等待

}

https://coding.net/u/aminglinux/p/nginx/git/blob/master/access/limit.md

#### Nginx的用户认证

当访问一些私密资源时，最好配置用户认证，增加安全性。

##### 步骤和示例

* 安装httpd
* yum **install** -y httpd
* 使用htpasswd生产密码文件
* htpasswd -c /usr/local/nginx/conf/htpasswd aming
* 配置nginx用户认证
* **location** **/admin**/
* {
* auth\_basic "Auth";
* auth\_basic\_user\_file /usr/local/nginx/conf/htpasswd;
* }
* 测试

curl -uaming:passwd www.aminglinux.com/admin/1.html

#### Nginx的错误日志

Nginx错误日志平时不用太关注，但是一旦出了问题，就需要借助错误日志来判断问题所在。

配置参数格式：error\_log /path/**to**/log level;

##### Nginx错误日志级别

常见的错误日志级别有debug | **info** | **notice** | **warn** | **error** | **crit** | **alert** | **emerg**

级别越高记录的信息越少，如果不定义，默认级别为error.

它可以配置在main、http、server、location段里。

如果在配置文件中定义了两个error\_log，在同一个配置段里的话会产生冲突，所以同一个段里只允许配置一个error\_log。

但是，在不同的配置段中出现是没问题的。

Copy

##### Nginx错误日志示例

error\_log /var/log/nginx/error.log crit;

如果要想彻底关闭error\_log，需要这样配置

error\_log /dev/null;

备注：同时在主配置文件和虚拟主机配置文件中定义错误日志，只会在虚拟主机中显示错误日志

**Nginx访问日志**

##### 访问日志格式示例

示例1

log\_format combined\_realip '$remote\_addr $http\_x\_forwarded\_for [$time\_local]'

'$host "$request\_uri" $status'

'"$http\_referer" "$http\_user\_agent"'*;*

示例2

log\_format main '$remote\_addr [$time\_local] '

'$host "$request\_uri" $status "$request"'

'"$http\_referer" "$http\_user\_agent" "$request\_time"'*;*

若不配置log\_format或者不在access\_log配置中指定log\_format，则默认格式为：

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

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

'"$http\_user\_agent"*;*

|  |  |
| --- | --- |
| $time\_local | 通用日志格式下的本地时间；（服务器时间） |
| $remote\_addr | 客户端（用户）IP地址 |
| $status | 请求状态码，如200，404，301，302等 |
| $body\_bytes\_sent | 发送给客户端的字节数，不包括响应头的大小 |
| $bytes\_sent | 发送给客户端的总字节数 |
| $request\_length | 请求的长度（包括请求行，请求头和请求正文） |
| $request\_time | 请求处理时间，单位为秒，小数的形式 |
| $upstream\_addr | 集群轮询地址 |

$upstream\_response\_time

指从Nginx向后端（php-cgi)建立连接开始到接受完数据 然后关闭连接为止的时间

|  |  |
| --- | --- |
| $remote\_user | 用来记录客户端用户名称 |

$request 请求方式（GET或者POST等）+URL（包含$request\_method,$host,$request\_uri）

$http\_user\_agent 用户浏览器标识

|  |  |
| --- | --- |
| $http\_host | 请求的url地址（目标url地址）的host |
| $host | 等同于$http\_host |

$http\_referer

来源页面，即从哪个页面转到本页，如果直接在浏览器输入网址来访问，则referer为空

$uri

请求中的当前URI(不带请求参数，参数位于$args)，不同于浏览器传递的$request\_uri的值，它可以通过内部重定向，或者使用index指令进行修改。

|  |  |
| --- | --- |
| $document\_uri | 等同于$uri |
| $request\_uri | 比$uri多了参数，即$uri+$args |
| $http\_x\_forwarded\_for | 如果使用了代理，这个参数会记录代理服务器的ip和客户端的ip |

#### Nginx访问日志过滤

一个网站，会包含很多元素，尤其是有大量的图片、**js、css等静态元素。**

这样的请求其实可以不用记录日志。

Copy

##### 配置示例

location ~\* ^.+\.(gif|**jpg**|**png**|**css**|**js**)$

{

access\_log off;

}

或

location ~\* ^.+\.(gif|**jpg**|**png**|**css**|**js**)$

{

access\_log /dev/null;

}

##### shell脚本切割Nginx日志

切割脚本内容：

**#!/bin/bash**

logdir=/var/log/nginx //定义日志路径

prefix=`date -d "-1 day" +%y%m%d` //定义切割后的日志前缀

cd $logdir

**for** f **in** `ls \*.log`

**do**

mv $f $f-$prefix //把日志改名

**done**

/bin/kill -USR1 $(cat /usr/local/nginx/logs/nginx.pid 2>/dev/null) 2>/dev/null //生成新的日志

bzip2 \*$prefix //压缩日志

find . -type f -mtime +180 |xargs /bin/rm -f //删除超过180天的老日志

##### 系统日志切割机制

在/etc/logrotate.d/下创建nginx文件，内容为：

/**data**/logs/\*log {

**daily**

**rotate** 30

**missingok**

**notifempty**

**compress**

**sharedscripts**

**postrotate**

/**bin**/**kill** -**USR1** $(**cat** /**usr**/**local**/**nginx**/**logs**/**nginx**.**pid** 2>/**dev**/**null**) 2>/**dev**/**null** || :

**endscript**

}

说明：

1 nginx日志在/**data**/logs/目录下面，日志名字以log结尾

2 daily表示每天切割

3 rotate 30表示日志保留30天

4 missingok表示忽略错误

5 notifempty表示如果日志为空，不切割

6 compress表示压缩

7 sharedscripts和endscript中间可以引用系统的命令

8 postrotate表示当切割之后要执行的命令

**Nginx+Tomcat架构**

配置文件示例

**server**

{

listen 80;

server\_name www.aminglinux.com;

location ~\* "\.(jpg|png|jepg|js|css|xml|bmp|swf|gif|html)$"

{

root /data/wwwroot/aminglinux/;

access\_log off;

expire 7d;

}

**location** /

{

proxy\_pass http://127.0.0.1:8080/;

proxy\_set\_header Host $host;

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

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

}

}

说明：

1 首先，把各种静态文件的请求分离出来，单独由nginx处理。

2 其他请求直接代理8080端口，即tomcat服务。