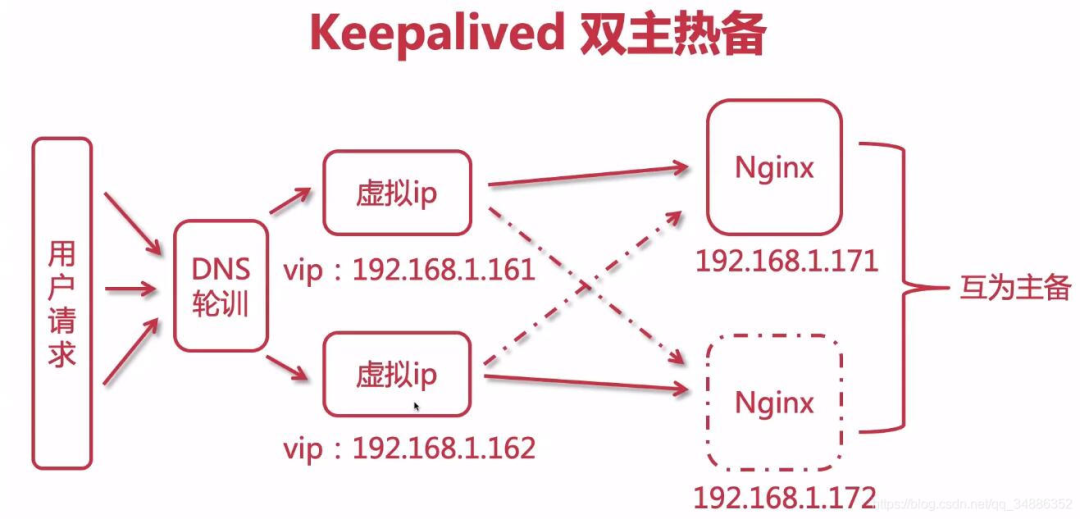
# Nginx+keepalived高可用

高可用方案：



## 安装Keepalive

(1)下载：https://www.keepalived.org/download.html

(2)解压：tar -zxvf keepalived-2.0.18.tar.gz

(3)使用configure命令配置安装目录与核心配置文件所在位置：

./configure --prefix=/usr/local/keepalived --sysconf=/etc

prefix：keepalived安装的位置

sysconf：keepalived核心配置文件所在位置，固定位置，改成其他位置则keepalived启动不了，/var/log/messages中会报错

配置过程中可能会出现警告信息，如下所示：

\*\*\* WARNING - this build will not support IPVS with IPv6. Please install libnl/libnl-3 dev libraries to support IPv6 with IPVS.

# 安装libnl/libnl-3依赖：yum -y install libnl libnl-devel

(4)安装keepalived

make && make install

(5)配置文件 在/etc/keepalived/keepalived.conf

(6)忘记安装配置的目录，则通过如下命令找到：

whereis keepalived

(7)启动keepalived

进入sbin目录

./keepalived

## 配置keepalived

(1)通过命令 vim keepalived.conf 打开配置文件

global\_defs {

# 路由id：当前安装keepalived的节点主机标识符，保证全局唯一

router\_id keep\_171

}

vrrp\_instance VI\_1 {

# 表示状态是MASTER主机还是备用机BACKUP

state MASTER

# 该实例绑定的网卡

interface ens33

# 保证主备节点一致即可

virtual\_router\_id 51

# 权重，master权重一般高于backup，如果有多个，那就是选举，谁的权重高，谁就当选

priority 100

# 主备之间同步检查时间间隔，单位秒

advert\_int 2

# 认证权限密码，防止非法节点进入

authentication {

auth\_type PASS

auth\_pass 1111

}

# 虚拟出来的ip，可以有多个（vip）

virtual\_ipaddress {

192.168.1.161

}

}

附：查看网卡信息命令

ip addr

(2)启动keepalived

(3)查看进程

ps -ef|grep keepalived

(4)查看vip(虚拟ip)

在网卡ens33下，多了一个192.168.1.161，这个就是虚拟ip

(5)把keepalived注册为系统服务

拷贝配置文件

将keepalived目录下etc/init.d/keepalived拷贝到/etc/init.d/下

将keepalived目录下etc/sysconfig/keepalived拷贝到/etc/sysconfig/下

刷新systemctl

systemctl daemon-reload

(6)启动、停止、重启keepalived

#启动:systemctl start keepalived.service

#停止:systemctl stop keepalived.service

#重启:systemctl restart keepalived.service

## 基本服务器宕机的主从切换配置

两台nginx服务器，分别安装keepalived，配置/etc/keepalived/keepalived.conf

(1)主nginx的keepalived

修改主nginx下/etc/keepalived/keepalived.conf文件

! Configuration File for keepalived

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from XXX@XXX.com #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_instance VI\_1 {

state MASTER #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 100 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

virtual\_ipaddress { #设置vip

192.168.17.10 #可以多个虚拟IP，换行即可

}

}

(2)备nginx的keepalived

修改备nginx下/etc/keepalived/keepalived.conf文件

配置备nginx时需要注意：需要修改state为BACKUP , priority比MASTER低，virtual\_router\_id和master的值一致

! Configuration File for keepalived

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from XXX@XXX.com #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_instance VI\_1 {

state BACKUP #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 99 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

virtual\_ipaddress { #设置vip

192.168.17.10 #可以多个虚拟IP，换行即可

}

}

(3) 启动 Keepalived

(4) 访问vip即可访问主机，当主机失效时访问vip就会访问到备机

## 基于nginx进程死掉的主从切换配置

keepalived是通过检测keepalived进程是否存在判断服务器是否宕机，如果keepalived进程在但是nginx进程不在了那么keepalived是不会做主备切换，所以我们需要写个脚本来监控nginx进程是否存在，如果nginx不存在就将keepalived进程杀掉。

在主nginx上需要编写nginx进程检测脚本（check\_nginx.sh），判断nginx进程是否存在，如果nginx不存在就将keepalived进程杀掉，check\_nginx.sh内容如下：

#!/bin/bash

# 如果进程中没有nginx则将keepalived进程kill掉

A=`ps -C nginx --no-header |wc -l` ## 查看是否有 nginx进程 把值赋给变量A

if [ $A -eq 0 ];then ## 如果没有进程值得为 零

service keepalived stop ## 则结束 keepalived 进程

fi

将check\_nginx.sh拷贝至/etc/keepalived下，

注意修改/etc/keepalived/check\_nginx.sh的可执行权限

修改主nginx的keepalived.conf，添加脚本定义检测：

注意下边红色标识地方：

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from miaoruntu@itcast.cn #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_script check\_nginx {

script "/etc/keepalived/check\_nginx.sh" ##监控脚本

interval 2 ##时间间隔，2秒

weight 2 ##权重

}

vrrp\_instance VI\_1 {

state MASTER #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 100 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

track\_script {

check\_nginx #监控脚本

}

virtual\_ipaddress { #设置vip

192.168.101.100 #可以多个虚拟IP，换行即可

}

}

修改后重启keepalived

## keepalived双主热备

(1)配置DNS轮询

在同一个域名下配置两个ip，自行百度

(2)配置第一台主机

global\_defs {

router\_id keep\_171

}

vrrp\_instance VI\_1 {

state MASTER i

nterface ens33

virtual\_router\_id 51

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.1.161

}

}

vrrp\_instance VI\_2 {

state BACKUP

interface ens33

virtual\_router\_id 52

priority 80

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.1.162

}

}

(3)配置第二台主机

global\_defs {

router\_id keep\_172

}

vrrp\_instance VI\_1 {

state BACKUP

interface ens33

virtual\_router\_id 51

priority 80

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.1.161

}

}

vrrp\_instance VI\_2 {

state MASTER

interface ens33

virtual\_router\_id 52

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.1.162

}

}

(4)重启两台Keepalived

systemctl restart keepalived

## keepalived配置nginx自动重启

(1)编写脚本

在/etc/keepalived/下创建脚本check\_nginx\_alive\_or\_not

#!/bin/bash

A=`ps -C nginx --no-header |wc -l`

# 判断nginx是否宕机，如果宕机了，尝试重启

if [ $A -eq 0 ];then

/usr/local/nginx/sbin/nginx

# 等待一小会再次检查nginx，如果没有启动成功，则停止keepalived，使其启动备用机

sleep 3

if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then

killall keepalived

fi

fi

(2)添加运行权限

chmod +x /etc/keepalived/check\_nginx\_alive\_or\_not.sh

(3)配置keepalived监听nginx脚本

vrrp\_script check\_nginx\_alive {

script "/etc/keepalived/check\_nginx\_alive\_or\_not.sh"

interval 2 # 每隔两秒运行上一行脚本

weight 10 # 如果脚本运行失败，则升级权重+10

}

(4)在vrrp\_instance中新增监控的脚本

track\_script {

check\_nginx\_alive # 追踪 nginx 脚本

}

(5)重启Keepalived使得配置文件生效

systemctl restart keepalived

# LVS实现高可用负载均衡

## 为什么要使用LVS+Nginx

lvs基于四层负载均衡，工作效率较Nginx的七层负载更高，使用LVS搭建Nginx集群，可以提高性能

四层负载均衡无法对信息处理，只能通过ip+端口的形式转发，所以需要七成负载进行数据的处理

Nginx接收请求来回，LVS可以只接受不响应

## LVS的三种模式

(1)NAT模式

客户端将请求发往LVS，LVS会选择一台服务器响应请求，服务器将结果返回给LVS，LVS再返回给客户端。

在NAT模式中，服务器的网关必须指向LVS，否则报文无法送达客户端

NAT 技术将请求的报文和响应的报文都需要通过LVS进行地址改写，因此网站访问量比较大的时候负载均衡调度器有比较大的瓶颈，一般要求最多之能 10-20 台节点

NAT 模式支持对 IP 地址和端口进行转换。即用户请求的端口和真实服务器的端口可以不一致

(2)TUN模式

客户端将请求发往LVS，LVS会选择一台服务器响应请求，在客户端与服务器之间建立隧道，返回结果的时候直接由服务器返回响应，不在经过LVS。

TUN模式必须所有的服务器上都绑定VIP的IP地址，所有的服务器都必须有网卡。

TUN模式走隧道运维难度大，并且会直接暴露服务器地址

服务器将应答包直接发给用户。所以，减少了负载均衡器的大量数据流动，负载均衡器不再是系统的瓶颈，就能处理很巨大的请求量，这种方式，一台负载均衡器能够为很多服务器进行分发。而且跑在公网上就能进行不同地域的分发

(3)DR模式

客户端将请求发往LVS，LVS会选择一台服务器响应请求，返回结果的时候通过统一的路由进行返回，不在经过LVS。

和TUN模式一样，LVS只是分发请求，应答包通过单独的路由返回给客户端，与TUN相比这种方式不需要隧道结构，可以兼容大多数的操作系统，同时统一路由可以隐藏真实的物理服务器。DR模式效率更高，但配置更复杂.

所有服务器节点和LVS只能在一个局域网里面。

# 使用教程

## **location 匹配规则**

### URL正则表达式

|  |  |  |
| --- | --- | --- |
| 符合 | 含义 | 案例 |
| . | 匹配换行符之外的任意一个字符 |  |
| ? | 重复出现0次或1次 |  |
| + | 重复出现1次或多次 |  |
| \* | 重复出现0次或多次 |  |
| \d | 匹配数字 |  |
| ^ | 正则表达式开始 |  |
| $ | 正则表达式结束 |  |
| {n} | 重复n次 |  |
| {n,} | 重复n次或更多次 |  |
| [c] | 匹配单个字符c |  |
| [a-z] | 匹配a-z小写字母中的任意一个 |  |
| (a|b|c) | 竖线的意思是或者，表示一般使用小括号括括住，匹配符合a字符 或是b字符 或是c字符的字符串 | 括号()之间匹配的内容，可以通过$1表示匹配第一个括号内的内容，([a-z]{3,}|\d\*) 字符串是abc, 则$1=abc |
| \ | 用于转义特殊字符 |  |

### URL匹配规则

|  |  |  |
| --- | --- | --- |
| 符合 | 含义 | 案例 |
| = | 进行普通字符精确匹配。也就是完全匹配 | 等号类型，不属于正则表达式。优先级最高，一旦匹配成功，则不再查找其他匹配项，停止搜索 |
| ^~ | 前缀匹配。如果匹配成功，则不再匹配其他location | 类型表达式，不属于正则表达式。一旦匹配成功，则不再查找其他匹配项，停止搜索 |
| ~ | 表示执行一个正则匹配，区分大小写 | 正则表达式类型的优先级次之。如果有多个location的正则能匹配的话，则使用正则表达式最长的那个 |
| ~\* | 表示执行一个正则匹配，不区分大小写 | 同~ |
| /xxx/ | 常规字符串路径匹配 | 同前缀匹配 |
| / | 通用匹配，任何请求都会匹配到 | 如果没有匹配到，就匹配通用的 |

### 匹配优先级

* 优先级分成2种类型

类型1： = > ^~ ， 一旦匹配成功，则不再查找其他匹配项，停止搜索

类型2： ~ = ~\* > /xxx/，匹配到之后，还会继续搜索其他其它location，直到找到优先级最高的。或者匹配到类型1的。

类型1 > 类型2

* 优先级顺序整理

(location =) > (location 完整路径) > (location ^~ 路径) > (location ~,~\* 正则顺序) > (location 部分起始路径) > (/)

注意：location的优先级与location配置的位置无关

* 优先级案例

location = / {

# 精确匹配/，主机名后面不能带任何字符串 /

[ configuration A ]

}

location / {

# 匹配所有以 / 开头的请求。

# 但是如果有更长的同类型的表达式，则选择更长的表达式。

# 如果有正则表达式可以匹配，则优先匹配正则表达式。

[ configuration B ]

}

location /documents/ {

# 匹配所有以 /documents/ 开头的请求，匹配符合以后，还要继续往下搜索。

# 但是如果有更长的同类型的表达式，则选择更长的表达式。

# 如果有正则表达式可以匹配，则优先匹配正则表达式。

[ configuration C ]

}

location ^~ /images/ {

# 匹配所有以 /images/ 开头的表达式，如果匹配成功，则停止匹配查找，停止搜索。

# 所以，即便有符合的正则表达式location，也不会被使用

[ configuration D ]

}

location ~\* \.(gif|jpg|jpeg)$ {

# 匹配所有以 gif jpg jpeg结尾的请求。

# 但是 以 /images/开头的请求，将使用 Configuration D，D具有更高的优先级

[ configuration E ]

}

location /images/ {

# 字符匹配到 /images/，还会继续往下搜索

[ configuration F ]

}

location = /test.html {

root /usr/local/var/www/html;

index index.html;

}

## 负载均衡

负载均衡应该是Nginx使用最多的几个配置之一。

下面各种方案适用不同情况下使用，所以可以根据实际情况选择使用哪种策略模式，不过fair和url\_hash需要安装第三方模块才能使用

### 循环(round-robin)

最简单的一种配置方式【Nginx默认的策略】，会将所有的请求依次分配给一个服务器。 比如：第一次访问，http://ip:port/demo返回服务器1，第二次返回服务器2。

http {

upstream www.zx.com {# 这里可以写域名、IP或是随意一个名词

server 127.0.0.1:8001; # 服务器1

server 127.0.0.1:8002; # 服务器2

}

server {

location /demo {# 访问 http://ip:port/demo

proxy\_pass http://www.zx.com;

}

}

}

### 最少连接(least-connected)

Web请求会被转发到连接数最少的服务器上

http {

upstream www.zx.com {

least\_conn; # 表示启动least-connected模式

server 127.0.0.1:8001; # 服务器1

server 127.0.0.1:8002; # 服务器2

}

server {

location /demo {

proxy\_pass http://www.zx.com;

}

}

}

### IP哈希(ip-hash)

IP哈希具有站粘性，会将同一个客户端的请求始终分配给一台服务器，在没有实现服务器间session共享的时候，可以选择使用IP哈希。

http {

upstream www.zx.com {

ip\_hash; # 表示启动ip\_hash模式

server 127.0.0.1:8001; # 服务器1

server 127.0.0.1:8002; # 服务器2

}

server {

location /demo {

proxy\_pass http://www.zx.com;

}

}}

### 权重(weight)

weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。 比如，weight=9 和 1，实际情况是，ngnix的第1~9个请求访问服务器1， 第10个请求访问服务器2。之后循环这个规律

http {

upstream www.zx.com {# 这里可以写域名、IP或是随意一个名词

server 127.0.0.1:8001 weight=9; # 服务器1

server 127.0.0.1:8002 weight=1; # 服务器2

# backup是指热备，只有当8001和8002都宕机的情况下才走8003

server 127.0.0.1:8003 weight=10 backup;

}

server {

location /demo { # 访问 http://ip:port/demo

proxy\_pass http://www.zx.com;

}

}

}

### fair(第三方)

按后端服务器的响应时间来分配请求，响应时间短的优先分配。这个配置是为了更快的给用户响应

模块安装教程：<https://blog.csdn.net/qq_33337927/article/details/97953489>

upstream backend {

fair;

server localhost:8080;

server localhost:8081;

}

### url\_hash(第三方)

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash\_method是使用的hash算法

upstream backend {

hash $request\_uri;

hash\_method crc32;

server localhost:8080;

server localhost:8081;

}

## Nginx代理与反向代理

什么是代理？代理是为网络用户代理了来访问网络的，比如Google agent代理翻墙

什么是反向代理？以[代理服务器](http://baike.baidu.com/view/751.htm" \t "_blank)来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器

区别就是，代理为用户直接访问，反向代理为用户直接访问的代理。

相同的地方是，都是由代理方去访问网络资源，并将数据返回给用户

### 代理URL/域名

# 代理本地

location /account {

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

proxy\_set\_header Host $host:$server\_port;

# 设置用户ip地址，X-Real-IP和 X-Forwarded-For 都是常有的记录真实IP的方式，一般选择一个就可以了

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

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

# 当请求服务器出错去寻找其他服务器

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503;

}

#代理域名，配置方式不同于本地，需要配置跨域

location / {

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Credentials true;

add\_header Access-Control-Allow-Headers "Origin, No-Cache, X-Requested-With, If-Modified-Since, Pragma, Last-Modified, Cache-Control, Expires, Content-Type, X-E4M-With";

proxy\_pass http://zx19890628.gitee.io/;

# 代理服务器最大响应时间

proxy\_read\_timeout 120s;

}

### 代理本地静态文件

expires过期时间，使静态文件可以缓存到Nginx中。 alias 将要开放的目录，结尾必须使用/,用来代替root 之后访问： http://127.0.0.1:8080/img/1.jpg。即可访问F:/壁纸/1.jpg。

location /img/ {

expires 24h;

alias F:/壁纸/;

}

### 代理多个域名

多个域名指向同一个机器。使用Nginx解析到不同的目录

有的时候， 一台机器上需要为多个不同的域名运行不同的站点。

比如： www.siteA.com作为博客，www.siteB.com作为论坛。

第一步: 在DNS解析中设置这两个域名， 设置记录值均为机器的IP。比如下图为某个域名的解析配置



第二步：配置好之后，通过域名都能访问到机器的默认端口。但是机器没法知道某个域名访问的是哪个网站。这时候，你就需要使用虚拟目录了。假设你把博客放在”/home/user/www/blog”下，论坛放在”/home/user/www/forum”下。

假设： Nginx安装目录是在： /usr/local/nginx

1.在nginx上创建一个”vhost”目录。

2.创建创建siteA的配置文件： vi /usr/local/nginx/vhost/vhost\_siteA.conf 。 写入以下信息

server {

listen 80; # 监听端口

server\_name www.siteA.com siteA.com; # 站点域名

root /home/user/www/blog; # 站点根目录

index index.html index.htm index.php; # 默认导航页

location / {

# WordPress固定链接URL重写

if (!-e $request\_filename) {

rewrite (.\*) /index.php;

}

}

# PHP配置

location ~ \.php$ {

fastcgi\_pass unix:/var/run/php5-fpm.sock;

fastcgi\_index index.php;

include fastcgi\_params;

}

}

3.创建创建siteB的配置文件： vi /usr/local/nginx/vhost/vhost\_siteB.conf 。

写入的信息与siteA相同，只是将siteA替换为siteB。

4.打开nginx.conf文件， 打开nginx.conf文件

http {

include /usr/local/nginx/vhost/\*.conf;

}

第三步：重启服务 ： nginx restart

第四步： 分别访问两个域名，对应的站点都可以打开。

### 代理二级域名

# account站点配置

server {

listen 80;

# account为二级域名,二级域名的代理

server\_name account.zx1989.top;

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;

}

}

# 主站点配置

server {

listen 80;

server\_name www.zx1989.top zx1989.top;

location / {

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Credentials true;

add\_header Access-Control-Allow-Headers "Origin, No-Cache, X-Requested-With, If-Modified-Since, Pragma, Last-Modified, Cache-Control, Expires, Content-Type, X-E4M-With";

proxy\_pass http://zx19890628.gitee.io/;

}

}

### 反向代理缓存

upstream [proxyName] {

server 192.168.1.173:8080 weight=1;

server 192.168.1.174:8080 weight=5;

server 192.168.1.175:8080 weight=2;

}

#proxy\_cache\_path 设置缓存保存的目录的位置

#keys\_zone设置共享内以及占用的空间大小

#mas\_size 设置缓存最大空间

#inactive 缓存过期时间，错过此时间自动清理

#use\_temp\_path 关闭临时目录

proxy\_cache\_path /usr/local/nginx/upsteam\_cache keys\_zone=mycache:5m max\_size=1g inactive=8h use\_temp\_path=off;

server {

listem 80;

server\_name www.tomcats.com;

#开启并使用缓存

proxy\_cache mycache;

#针对200和304响应码的缓存过期时间

proxy\_cache\_valid 200 304 8h;

location / {

proxy\_pass http://tomcats;

}

}

## 访问限制

### 禁止使用IP访问

为了避免别人把未备案的域名解析到自己的服务器IP而导致服务器被断网，以及其他的恶意操作。可以设置禁止IP访问。

server { # 配置禁止使用IP访问站点

listen 80 default;# 使用default，表示本机，必须定义

server\_name \_; # 使用下划线，表示本机，必须定义

return 500; # return 500，表示返回响应值。

#也可以这样写，将使用IP访问的重定向到另一个站点 rewrite ^(.\*) http://www.exchangecn.com permanent;

}

server { # 主站点配置

listen 80;

server\_name www.zx1989.top zx1989.top;# 禁止域名访问后，必须对其他server二配置server\_name

}

第一种情况：访问A站定向到B站

server {

server\_name www.exchangecn.net ;

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

}

第二种情况：不是访问A站的全部重定向到指定页面

server {

server\_name www.exchangecn.net;

if ($host != 'exchangecn .net' ) {

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

}

}

### IP黑白名单

第一步：创建一个白名单：

cat /usr/local/nginx/conf/ip.conf

192.168.10.0/24 0;

192.168.12.25 1;

从允许访问的段位从小到大排列,如127.0.0.0/24 下面才能是10.10.0.0/16

24表示子网掩码:255.255.255.0

16表示子网掩码:255.255.0.0

8表示子网掩码:255.0.0.0

第二步：使用geo 获得白名单(http下)

geo $remote\_addr $ip\_whitelist {

default 0;

include /usr/local/nginx/conf/ip.conf;

}

第三步：IP地址在列表中的,则访问代理(server下)

location / {

if ( $ip\_whitelist = 1 ){

proxy\_pass http://127.0.0.1;

break;

}

return 404;

}

第四步：当然也可以作为黑名单，如果在名单内就返回403

if ( $ip\_whitelist = 1 ) {

return 403;

}

对于简单的IP访问控制可以使用如下配置

按照顺序依次检测，直到匹配到第一条规则

location / {

deny 192.168.1.1; # 禁止一个ip

allow 192.168.1.0/24; # 允许一个ip网段

allow 10.1.1.0/16;

allow 2001:0db8::/32;

deny all; # 剩余的全部不允许

}

# 禁止访问某个目录

location ~\* \.(txt|doc)${

root $doc\_root;

deny all;

}

### 防盗链

location / {

valid\_referers none blocked \*.test.com \*.aaa.com ; #设置域名白名单

if($invalid\_referer) { #其余的返回403

return 403;

}

}

测试效果

curl -e "http://www.test.com/" -I -x127.0.0.1:80 'http://www.test.com/static/image/common/logo.png' #返回200正常可以调用

curl -e "http://www.baidu.com/" -I -x127.0.0.1:80 'http://www.test.com/static/image/common/logo.png' 返回403

curl -e指定referer时，要带上http://

### 用户认证

实现基于用户的访问控制，使用basic机制进行用户认证

1.需要安装httpd

2.使用命令Apache htpasswd创建账号密码文件：

htpasswd -bcm /var/user 用户1 密码1

3.修改Nginx配置

location /admin/ {

alias /webapps/app1/data/;

auth\_basic “这里是登录提示语”;

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

}

密码保存文件

name1:password1

name2:password2:comment

name3:password3

### 限流

Nginx的限流模块，是基于漏桶算法实现的，在高并发的场景下非常实用

Nginx限流实战： <https://mp.weixin.qq.com/s/9lK6PRGoAJGZc4WGzu_jVQ>

如何设置能限制某个IP某一时间段的访问次数是一个让人头疼的问题，特别面对恶意的ddos攻击的时候。其中CC攻击（Challenge Collapsar）是DDOS（分布式拒绝服务）的一种，也是一种常见的网站攻击方法，攻击者通过代理服务器或者肉鸡向向受害主机不停地发大量数据包，造成对方服务器资源耗尽，一直到宕机崩溃。 cc攻击一般就是使用有限的ip数对服务器频繁发送数据来达到攻击的目的，nginx可以通过HttpLimitReqModul和HttpLimitZoneModule配置来限制ip在同一时间段的访问次数来防cc攻击。 HttpLimitReqModul用来限制连单位时间内连接数的模块，使用limit\_req\_zone和limit\_req指令配合使用来达到限制。一旦并发连接超过指定数量，就会返回503错误。 HttpLimitConnModul用来限制单个ip的并发连接数，使用limit\_zone和limit\_conn指令 这两个模块的区别前一个是对一段时间内的连接数限制，后者是对同一时刻的连接数限制

1.HttpLimitReqModul 限制某一段时间内同一ip访问数实例

http{

#定义一个名为allips的limit\_req\_zone用来存储session，大小是10M内存，

#以$binary\_remote\_addr 为key,限制平均每秒的请求为20个，

#1M能存储16000个状态，rete的值必须为整数，

#如果限制1秒钟1请求，可以设置成1r/s，即每秒1个请求。

limit\_req\_zone $binary\_remote\_addr zone=allips:10m rate=1r/s;

server{

location {

#限制每ip每秒不超过20个请求，漏桶数burst为5

#brust的意思就是，如果第1秒、2,3,4秒请求为19个，

#第5秒的请求为25个是被允许的。

#但是如果你第1秒就25个请求，第2秒超过20的请求返回503错误。

#nodelay，如果不设置该选项，严格使用平均速率限制请求数，

#第1秒25个请求时，5个请求放到第2秒执行，

#设置nodelay，25个请求将在第1秒执行。

limit\_req zone=allips burst=5 nodelay;

}

}

}

当访问次数大于限制的时候，会出现下面页面

An error occurred.

Sorry, the page you are looking for is currently unavailable.

Please try again later.

If you are the system administrator of this resource then you should check the error log for details.

Faithfully yours, nginx.

2.HttpLimitZoneModule 限制并发连接数实例

limit\_zone只能定义在http作用域，limit\_conn可以定义在http server location作用域

http{

#定义一个名为one的limit\_zone,大小10M内存来存储session，

#以$binary\_remote\_addr 为key

#nginx 1.18以后用limit\_conn\_zone替换了limit\_conn

#且只能放在http作用域

limit\_conn\_zone one $binary\_remote\_addr 10m;

server{

location {

limit\_conn one 20; #连接数限制

#带宽限制,对单个连接限数，如果一个ip两个连接，就是500x2k

limit\_rate 500k;

}

}

}

3.参数详解：

1）limit\_req\_zone定义在http块中，$binary\_remote\_addr 表示保存客户端IP地址的二进制形式。

2）Zone定义IP状态及URL访问频率的共享内存区域。

zone=keyword标识区域的名字，以及冒号后面跟区域大小。16000个IP地址的状态信息约1MB，所以示例中区域可以存储160000个IP地址。

3）Rate定义最大请求速率。示例中速率不能超过每秒100个请求。

4）burst排队大小，nodelay不限制单个请求间的时间。相对于传统的漏桶算法慢吞吞地转发请求的缺陷，Nginx实现了一种漏桶算法的优化版，允许开发者指定快速转发，而且还不影响正常的限流功能。开发者只需要在指定limit\_req的一行中指定burst之后指定另一个参数nodelay，就可以在请求总数没有超过burst指定值的情况下，迅速转发所有请求了。

4.案例

#限制IP访问次数

#定义一个名为allips的limit\_req\_zone用来存储session，大小是10M内存，

#以$binary\_remote\_addr 为key,限制平均每秒的请求为20个，

#1M能存储16000个状态，rete的值必须为整数，

#如果限制两秒钟一个请求，可以设置成30r/m

limit\_req\_zone $binary\_remote\_addr zone=allips:10m rate=20r/s;

1. 基于时间动态限速

nginx内置的一个ssi模块，这个模块有两个时间变量：$date\_local和$date\_gmt，分别对应当前时间和GMT时间

map $date\_local $limit\_rate\_time {

default 4K;

~(00:|01:|02:|03:|04:|05:|06:|07:).\*:.\* 16K;

~(08:|12:|13:|18:).\*:.\* 8K;

~(19:|20:|21:|22:|23:).\*:.\* 16K;

}

limit\_rate $limit\_rate\_time

2. 基于变量动态限速

map $cookie\_type $limit\_rate\_cookie {

vip 512K;

common 8K;

}

limit\_rate $limit\_rate\_cookie

## 日志

nginx的log日志分为access log 和 error log

access log 记录了哪些用户，哪些页面以及用户浏览器、ip和其他的访问信息

error log 则是记录服务器错误日志

### 日志内容

10.1.1.1 - - [22/Aug/2014:16:48:14 +0800] "POST /ajax/MbpRequest.do HTTP/1.1" 200 367 "-" "Dalvik/1.6.0 (Linux; U; Android 4.1.1; ARMM7K Build/JRO03H)" "119.189.56.175" 127.0.0.1:8090 0.022 0.022

10.1.1.1 - - [22/Aug/2014:16:48:19 +0800] "POST /ajax/MbpRequest.do HTTP/1.1" 200 616 "-" "Dalvik/1.6.0 (Linux; U; Android 4.0.4; GT-I9103 Build/IMM76D)" "36.250.89.22" 127.0.0.1:8090 0.036 0.036

1.客户端（用户）IP地址。如：上例中的 10.1.1.1 (内网负载均衡地址)

2.访问时间。如：上例中的 [22/Aug/2014:16:48:19 +0800]

3.访问端口。如：上例中的 127.0.0.1:8080

4.响应时间。如：上例中的 0.022

5.请求时间。如：上例中的 0.022

6.用户地理位置代码（国家代码）。

7.请求的url地址（目标url地址）的host。如：上例中的 /....

8.请求方式（GET或者POST等）。如：上例中的 GET

9.请求url地址（去除host部分）。如：上例中的 /html/test.html

10.请求状态（状态码，200表示成功，404表示页面不存在，301表示永久重定向等，具体状态码可以在网上找相关文章，不再赘述）。如：上例中的 "200"

11.请求页面大小，默认为B（byte）。如：上例中的 2426

12.来源页面，即从哪个页面转到本页，专业名称叫做“referer”。如：上例中的 "http://a.com"

13.用户浏览器语言。如：上例中的 "es-ES,es;q=0.8"

14. 用户浏览器其他信息，浏览器版本、浏览器类型等。如：上例中的 "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.97 Safari/537.11"

### 配置与字段解释

nginx access日志的格式不是一成不变的，是可以自定义的。

在nginx的nginx.conf配置文件找到：log\_format 这里就是日志的格式。内容如下：

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

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

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

'$upstream\_addr $upstream\_response\_time $request\_time ';

access\_log logs/access.log main;

配置access log日志的存储位置及文件，注意：access.log文件是可以按日期进行分割的，方便查看及处理

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

相关说明解释

$remote\_addr 客户端地址 211.28.65.253

$remote\_user 客户端用户名称 --

$time\_local 访问时间和时区 18/Jul/2012:17:00:01 +0800

$time\_iso8601 记录ISO8601标准格式下的本地时间 2022-11-26T20:49:29+08:00

$request 请求的URI和HTTP协议 "GET /article-10000.html HTTP/1.1"

$request\_time 整个请求的总时间 0.205

$request\_length 请求的长度（包括请求行， 请求头和请求正文）。

$http\_host 请求地址，即浏览器中你输入的地址（IP或域名） www.wang.com 192.168.100.100

$status HTTP请求状态 200

$upstream\_status upstream状态 200

$body\_bytes\_sent 发送给客户端资源大小，不包括响应头的大 1547

$bytes\_sent 发送给客户端资源总大小 2000

$http\_referer url跳转来源 https://www.baidu.com/

$http\_user\_agent 用户终端浏览器等信息 Mozilla/5.0 (Windows NT 6.1) Chrome/23.0.1271.97 Safari/537.11

$ssl\_protocol SSL协议版本 TLSv1

$ssl\_cipher 交换数据中的算法 RC4-SHA

$upstream\_addr 后台upstream的地址，即真正提供服务的主机地址 10.10.10.100:80

$upstream\_response\_time 请求过程中，upstream响应时间 0.002

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

$X-Real-IP 记录起始的客户端IP地址和上一层客户端的IP地址

### 修改时间格式

修改原因？因为要获取nginx访问信息，作为开发的数据使用，但是nginx的access.log文件中的默认的时间格式是这样的： [02/Nov/2017:20:48:25 +0800]

而要求的格式类似如下：[2017-11-02 20:52:06]

Nginx修改access.log日志时间格式： <https://www.cnblogs.com/bigberg/p/7774508.html>

# 使用案例

## 文件上传大小限制

http模块下： client\_max\_body\_size 20M;

## 自定义变量

server {

# 定义变量 doc\_root

set $doc\_root /usr/local/var/www;

# 引用变量 $doc\_root

location ~\* \.(gif|jpg|jpeg|png|bmp|ico|swf|css|js)$ {

root $doc\_root/img;

}

location = /50x.html {

root $doc\_root;

}

}

## 设置异常页面

使用error\_page指令，您可以配置NGINX返回自定义页面以及错误代码，替换响应中的其他错误代码，或将浏览器重定向到其他URI。在以下示例中，error\_page指令指定要返回404页面错误代码的页面(/404.html)

server {

listen 80;

server\_name localhost;

# 指的server出现下面状态码的时候，跳转到指定页面

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/local/var/www/;

}

}

## 限制浏览器类型

我们使用user\_agent限制某些浏览器

server {

#禁止useragent为baidu、360和sohu，~\*表示不区分大小写匹配

if ($http\_user\_agent ~\* 'baidu|360|sohu'){

return 403;

}

}

server {

#添加如下内容即可防止爬虫

if ($http\_user\_agent ~\* 'qihoobot|Baiduspider|Googlebot|Googlebot-Mobile|Googlebot-Image|Mediapartners-Google|Adsbot-Google|Feedfetcher-Google|Yahoo! Slurp|Yahoo! Slurp China|YoudaoBot|Sosospider|Sogou spider|Sogou web spider|MSNBot|ia\_archiver|Tomato Bot'){

return 403;

}

}

测试是否有效： curl -A “baidu” -x127.0.0.1:80 www.test.com/forum.php -I 该命令指定百度为user\_agent,返回403

## 访问缓存

server {

listem 80;

server\_name www.tomcats.com;

location / {

proxy\_pass http://tomcats;

expires 10s; #浏览器缓存10秒钟

#expires @22h30m #在晚上10点30的时候过期

#expires -1h #缓存在一小时前时效

#expires epoch #不设置缓存

#expires off #缓存关闭，浏览器自己控制缓存

#expires max #最大过期时间

}

}

## 虚拟主机

可以代理多个小型网站。

方式如下：

1.基于多IP的方式

2.基于多端口的方式

3.基于多域名的方式

### 基于多IP的方式

相当于一台服务器上有多个80端口

1.一个机器添加多个子ip

ifconfig eth0:1 10.0.0.3/24

ip addr add 10.0.0.91/24 dev eth0

2.添加多个网站的配置

server {

listen 10.0.0.9:80;

location / {

root /code/addr1; # 网站1的静态代码

index index.html;

}

}

server {

listen 10.0.0.91:80;

location / {

root /code/addr2; # 网站2的静态代码

index index.html;

}

}

### 基于多端口的方式

这个就简单多了，不同的端口有不同的网站

server {

listen 8001;

location / {

root /code/addr1; # 网站1的静态代码

index index.html;

}

}

server {

listen 8002;

location / {

root /code/addr2; # 网站2的静态代码

index index.html;

}

}

### 基于多域名的方式

server {

listen 80;

server\_name www.zx1.com;

location / {

root /code/addr1; # 网站1的静态代码

index index.html;

}

}

server {

listen 80;

server\_name www.zx2.com;

location / {

root /code/addr2; # 网站2的静态代码

index index.html;

}

}

### 总结

在nginx的虚拟主机类型中，基于域名的虚拟主机应用最为广泛。

1，基于端口和IP的虚拟主机类型，用户体验不好。

2，基于IP类型的虚拟主机，如在公网环境下使用，会产生额外费用。

3，基于域名的虚拟主机，一次付费，用户体验较好。

## 日志切割

### logrotate

在linux上logrotate是一个日志文件管理工具。用于分割日志文件，删除旧的日志文件，并创建新的日志文件，起到“转储”作用，它是默认随linux一起被安装的。

Logrotate是基于CRON来运行的，其脚本是/etc/cron.daily/logrotate

使用方法。在/etc/logrotate.d创建规则

vim /etc/logrotate.d/nginx

/usr/local/nginx/logs/\*.log {

daily #每天切割日志

size +100M # 超过100M时分割，单位K,M,G，优先级高于daily

missingok # 忽略日志丢失，就是如果没有日志文件也不报错

rotate 30 # 日志保留时间 30天

compress # 日志压缩。nocompress 不压缩

delaycompress # 延时压缩

dateext # 日志文件切割时添加日期后缀

notifempty # # 日志为空时不进行切换，默认为ifempty

create 640 nginx nginx #切割好的日志权限

sharedscripts #开始执行脚本

postrotate #标注脚本内容

if [ -f /var/run/nginx.pid ]; then #判断nginx启动

kill -USR1 `cat /usr/local/nginx/logs/nginx.pid` #重新生成一个access.log

fi

endscript #脚本执行完毕

}

手动执行分割

logrotate --force /etc/logrotate.d/nginx

显示详细的信息；而且--debug/-d实际上不会操作具体文件(Dry Run)

logrotate --debug --verbose --force /etc/logrotate.d/nginx

### shell脚本分割

定时任务跑脚本：

0 1 \* \* \* /bin/sh /data1/nginx/logs/nginx\_log.sh

#!/bin/bash

dat=`date +"%Y%m%d" `

mon=`date +"%Y%m"`

echo $dat

nginx\_path="/data/log/nginx/"

mondir=$nginx\_path$mon

/bin/echo $mondir

if [ ! -x "$mondir" ]; then

echo "开始创建日期文件夹"

mkdir "$mondir"

echo "创建日期文件夹结束"

fi

/bin/echo `date +"%Y-%m-%d %H:%M:%S"`

/bin/echo ` ls -al $mondir`

# 声明需要分割的日志文件

logs=("access\_xxx.com" "nginx\_access" "nginx\_error")

for log\_name in ${logs[@]}

do

echo $log\_name

mv $nginx\_path$log\_name".log" $mondir/$log\_name-$dat".log"

done

kill -USR1 `cat /var/run/nginx.pid`

/bin/echo "done "