# Nginx搭建

官网: http://nginx.org/

## 1.环境准备

物理机:linux-ubuntu14.04，2台

IP:192.168.203.219

IP:192.168.203.220

在上面均需要安装nginx、keepalived。Keepalived提供的VIP:192.168.203.218

## 2.安装nginx

* 在线安装

#apt-get update //更新源,centos:yum update

#apt-get install nginx

#./nginx -v //查看版本

#./nginx -V //查看configure参数

不通的操作系统，安装方式略有区别

* 源码安装(针对个性化需求，需要源码安装)

如果要安装其他模块，需要将源码取下来，将补丁打入源码，在安装。

#下载nginx源码包并解压, nginx-1.4.6.tar.gz以及其他模块pcre-8.32.tar.gz、zlib-1.2.7.tar.gz、nginx\_tcp\_proxy\_module模块,

<http://yaoweibin.github.io/nginx_tcp_proxy_module/README.html>

#cd nginx-1.4.6

# patch -p1 < /path/to/nginx\_tcp\_proxy\_module/tcp.patch #将源码打入nginx

#./configure --prefix=/usr/local/nginx --with-pcre=/tmp/pcre-8.32 --with-zlib=/tmp/zlib-1.2.7 --with-http\_stub\_status\_module --with-http\_ssl\_module --with-http\_realip\_module --add-module=/path/to/nginx\_tcp\_proxy\_module

#make

#make install

## 3.启动、停止、重启nginx服务器

* 在线安装下的命令

#service nginx start //启动

#service nginx stop //停止

#service nginx restart //重启

#service nginx reload //重新加载配置文件，不用重启nginx

* 源码安装下的命令

#cd /usr/local/nginx/sbin

#/usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf //启动加载指定路径的配置文件

#/usr/local/nginx/sbin/nginx -s stop //停止

#/usr/local/nginx/sbin/nginx -s reload //重新加载配置，不用重启服务器)

## 4.根据需求，对配置文件进行修改

### 1.根据域名做请求转发

如在这里根据域名来指向到不同的真实server

#vi /etc/nginx/conf.d/vhost1.conf

*server*

*{*

*listen 10000;*

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

*if ($http\_host ~\* "^(.\*?)\.newtouchx\.com:10001$") {*

*set $app $1;*

*set $domain http://$1.newtouchx.com:10001;*

*}*

*location / {*

*if ($domain ~\* "http://test3.newtouchx.com:10001") {*

*proxy\_pass http://192.168.202.241:8080/$app$request\_uri;*

*}*

*if ($domain ~\* "http://test2.newtouchx.com:10001") {*

*proxy\_pass http://192.168.202.11:8080/$app$request\_uri;*

*}*

*if ($domain ~\* "http://test1.newtouchx.com:10001") {*

*proxy\_pass http://192.168.202.244:8080/$app$request\_uri;*

*}*

*proxy\_pass http://192.168.202.241:8080/default/;*

*proxy\_redirect off;*

*proxy\_set\_header Host $host;*

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

*proxy\_set\_header X-Forwarded-Host $host;*

*proxy\_set\_header X-Forwarded-Server $host;*

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

*}*

### 2.根据应用名做请求转发

*server*

*{*

*listen 10000;*

*server\_name \*.newtouch.cn;*

*location ^~ /temp1/ {*

*proxy\_pass http://www.baidu.com/;*

*proxy\_redirect off;*

*proxy\_set\_header Host $host;*

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

*proxy\_set\_header X-Forwarded-Host $host;*

*proxy\_set\_header X-Forwarded-Server $host;*

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

*}*

*location ^~ /temp2/ {*

*proxy\_pass http://www.sohu.com/;*

*proxy\_redirect off;*

*proxy\_set\_header Host $host;*

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

*proxy\_set\_header X-Forwarded-Host $host;*

*proxy\_set\_header X-Forwarded-Server $host;*

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

*}*

*}*

### 3.根据域名和应用名做请求转发

*server {*

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

*listen 8080;*

*if ($http\_host ~\* "^(.\*?)\.newtouch\.com$") {*

*set $domain http://$1.newtouch.com;*

*}*

*if ($http\_host ~\* "^newtouch.com$") {*

*set $domain http://www.newtouch.com;*

*}*

*location /app1 {*

*if ($domain ~\* "http://www.newtouch.com") {*

*proxy\_pass http://192.168.202.7:81$request\_uri;*

*}*

*if ($domain ~\* "http://step.newtouch.com") {*

*proxy\_pass http://192.168.202.7$request\_uri;*

*}*

*proxy\_pass http://192.168.202.7:81;*

*proxy\_redirect off;*

*proxy\_set\_header Host $host;*

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

*proxy\_set\_header X-Forwarded-Host $host;*

*proxy\_set\_header X-Forwarded-Server $host;*

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

*client\_max\_body\_size 1024m;*

*}*

*location /app2 {*

*if ($domain ~\* "http://www.newtouch.com") {*

*proxy\_pass http://192.168.202.7:81$request\_uri;*

*}*

*if ($domain ~\* "http://step.newtouch.com") {*

*proxy\_pass http://192.168.202.7$request\_uri;*

*}*

*proxy\_pass http://192.168.202.7:81;*

*proxy\_redirect off;*

*proxy\_set\_header Host $host;*

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

*proxy\_set\_header X-Forwarded-Host $host;*

*proxy\_set\_header X-Forwarded-Server $host;*

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

*client\_max\_body\_size 1024m;*

*}*

*}*

### 4.nginx动静分离负载均衡

如在/etc/nginx/sites-enabled/default下的server{}里添加

*#upstream,配置负载均衡的算法和后台真实的服务器*

*upstream dynamic {*

*server localhost:8080 weight=1 max\_fails=2 fail\_timeout=30s;*

*}*

*server {*

*listen 80 default\_server;*

*listen [::]:80 default\_server ipv6only=on;*

*root /usr/front/resources/;*

*index page/item/indexPane.html;*

*#将/下的动态内容请求到动态服务器*

*location / {*

*proxy\_connect\_timeout 3;*

*proxy\_send\_timeout 30;*

*proxy\_read\_timeout 30;*

*proxy\_redirect off;*

*proxy\_set\_header Host $host;*

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

*proxy\_set\_header X-Forwarded-Host $host;*

*proxy\_set\_header X-Forwarded-Server $host;*

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

*proxy\_pass http://dynamic;*

*}*

*#动态内容访问本地磁盘(有可能挂载远程的分布式文件系统)*

*location ~ ^/(lib|css|js|images|fonts|page|test|upload)/.\*$ {*

*root /usr/front/resources/;*

*index page/login/registe.html;*

*expires 3d;*

*}*

*#首页*

*location ~ ^/$ {*

*root /usr/front/resources/;*

*index page/login/registe.html;*

*expires 3d;*

*}*

*}*

### 5.ssl配置

#先到认证中心申请针对某个域名的证书

#vi https.conf，改后reload配置文件

*server {*

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

*listen 443;*

*ssl on;*

*ssl\_certificate /etc/nginx/conf.d/ssl/newtouchall.crt;*

*ssl\_certificate\_key /etc/nginx/conf.d/ssl/newtouch.key;*

*ssl\_session\_timeout 5m;*

*if ($http\_host ~\* "^(.\*?)\.newtouch\.com$") {*

*set $domain https://$1.newtouch.com;*

*}*

*if ($http\_host ~\* "^newtouch.com$") {*

*set $domain https://www.newtouch.com;*

*}*

*location / {*

*if ($domain ~\* "https://www.newtouch.com") {*

*proxy\_pass http://192.168.202.6:81$request\_uri;*

*}*

*if ($domain ~\* "https://step.newtouch.com") {*

*proxy\_pass http://192.168.202.6$request\_uri;*

*} proxy\_pass http://192.168.202.6:81;*

*proxy\_redirect off;*

*proxy\_set\_header Host $host;*

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

*proxy\_set\_header X-Forwarded-Host $host;*

*proxy\_set\_header X-Forwarded-Server $host;*

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

*client\_max\_body\_size 1024m;*

*}*

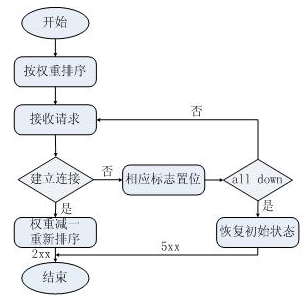
*}*

## 5.nginx算法

* 轮训算法[内置]

默认情况下，每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况

加权轮询的原理很简单，首先我们介绍一下轮询的基本流程。如下是处理一次请求的流程图：



图中有两点需要注意，第一，如果可以把加权轮询算法分为先深搜索和先广搜索，那么nginx采用的是先深搜索算法，即将首先将请求都分给高权重的机器，直到该机器的权值降到了比其他机器低，才开始将请求分给下一个高权重的机器；第二，当所有后端机器都down掉时，nginx会立即将所有机器的标志位清成初始状态，以避免造成所有的机器都处在timeout的状态，从而导致整个前端被夯住。

这里处理请求时，用到了Nginx指令upstream，来实现负载均衡

nginx.conf文件:

*upstream tomcat {*

*server 192.168.1.181:8080;//默认轮询(round robin)算法，加weight按权重排序*

*server 192.168.1.182:8080;*

*}*

*server{*

*listen 80;*

*server\_name 192.168.1.181;*

*location / {*

*proxy\_pass http://tomcat;*

*proxy\_set\_header Host $http\_host;*

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

*proxy\_set\_header REMOTE-HOST $remote\_addr;*

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

*}*

*}*

配置介绍

1. 这里的upstream的name=tomcat,与server中的,location指令中proxy\_pass属性的值相对应(除<http://部分>)
2. 在一个http{…..}指令中可以配置多个upstream 和server 指令来处理多个端口和地址。

Server指令监听地址的端口号默认为：80,请求地址为：<http://localhost/>

1. 在upstream 中server可以监听TCP协议和Unix域套接字混合使用：

*upstream backend {*

*server backend1.example.com weight=5;*

*server 127.0.0.1:8080 max\_fails=3 fail\_timeout=30s;*

*server unix:/tmp/backend3; //这里unix: 为固定模式*

*}*

参数：weight 表示权重，默认值是1,指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。；

max\_fails=number,在一段时间内，允许与服务器之间请求连接失败的次数，超过设置的次数后，Nginx将视该服务器已宕机(这里的一段时间是由fail\_timeout=time属性决定),默认值为1，值设置为0时，将不统计尝试连接的次数；

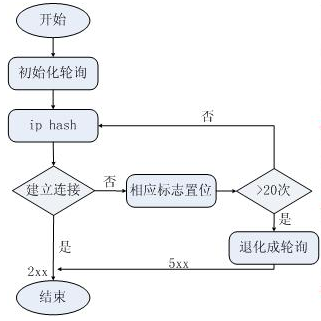
fail\_timeout=time默认值为10秒；

backup该属性表示这个服务器是用来备用的；

down 表示该服务器永远被禁用，通常与指令ip\_hash一起使用。

* IP\_HASH算法[内置]

ip hash是nginx内置的另一个负载均衡的策略，流程和轮询很类似，只是其中的算法和具体的策略有些变化，如下图所示：



从本质上说，ip hash算法是一种变相的轮询算法，如果两个ip的初始hash值恰好相同，那么来自这两个ip的请求将永远落在同一台服务器上，这为均衡性埋下了很深的隐患。

使用ip\_hash指令来实现负载均衡和解决session问题。

每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session 的问题

nginx.conf文件:

*upstream tomcat {*

*ip\_hash;*

*server 192.168.1.181:8080;*

*server 192.168.1.182:8080;*

*}*

*server{*

*listen 80;*

*server\_name 192.168.1.181;*

*location / {*

*proxy\_pass http://tomcat;*

*proxy\_set\_header Host $http\_host;*

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

*proxy\_set\_header REMOTE-HOST $remote\_addr;*

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

*}*

*}*

这里server监听了80端口，地址拦截是冲”/”开始，与上面例子的区别在于，这里的upstream指令中，引用一个指令ip\_hash,只有所有请求这个端口的客户端，都会通过cookie来实现会话的保持。

[备注:如何客户端经常变化，那么ip\_hash算法则不适合，所以ip\_hash适合客户端ip不变化的场合]

**location指令介绍:**

语法规则： location [=|~|~\*|^~] /uri/ { … }

= 开头表示精确匹配

^~ 开头表示uri以某个常规字符串开头，理解为匹配url路径即可。nginx不对url做编码，因此请求为/static/20%/aa，可以被规则^~ /static/ /aa匹配到（注意是空格）。

~ 开头表示区分大小写的正则匹配

~\* 开头表示不区分大小写的正则匹配

!~和!~\*分别为区分大小写不匹配及不区分大小写不匹配的正则

/ 通用匹配，任何请求都会匹配到。

* 黏性session算法[扩展]

第一种:nginx\_upstream\_jvm\_route方式

原理:基于cookie的stickly session(JSESSIONID)的实现

参考网页:<http://code.google.com/p/nginx-upstream-jvm-route/>

Tomcat1 (修改conf/server.xml)

*<Engine name="Catalina" defaultHost="localhost" jvmRoute="a">*

Tomcat2 (修改conf/server.xml)

*<Engine name="Catalina" defaultHost="localhost" jvmRoute="b">*

*nginx.conf文件:*

*upstream tomcat {*

*server 192.168.1.181:8080 srun\_id=a;*

*server 192.168.1.182:8080 srun\_id=b;*

*jvm\_route $cookie\_JSESSIONID reverse;*

*}*

*server{*

*listen 80;*

*server\_name 192.168.1.181;*

*location / {*

*proxy\_pass http://tomcat;*

*proxy\_set\_header Host $http\_host;*

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

*proxy\_set\_header REMOTE-HOST $remote\_addr;*

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

*}*

*}*

在两个tomcat部署两个测试工程，在一个测试的jsp分别打印出session和cookie信息

Tomcat1上的index.jsp

主机A</br>

session:<%out.print(request.getSession()) ;%></br>

cookie:<%out.println(request.getHeader("Cookie")); %>

Tomcat2上的index.jsp

主机B </br>

session:<%out.print(request.getSession()) ;%></br>

cookie:<%out.println(request.getHeader("Cookie")); %>

打开IE,地址栏输入http://192.168.1.181/test/index.jsp

可以看到第一次访问时,cookie为null，nginx接管JSESSIONID的管理，选择后台一台服务器，将其值绑定到JSESSIONID中，下次访问nginx时，从JSESSIONID中取出选择的后台服务器。

当关闭浏览器，重新敲入地址，访问到另外一台机器，一直刷新仍然是那个机器。

说明在一个session周期，访问的后台tomcat是固定的。下次session，可以到另外的tomcat，一旦确定，请求所访问的tomcat则固定。

只能通过JSESSIONID来实现，不能指定另外的key。而JSESSIOINID是会话cookie，当前浏览器关闭，重新开始。

第二种：nginx-sticky-module方式

原理:基于cookie的stickly session(用户自己指定的key，不能用JSESSIONID，默认为route)的实现。由nginx接管route这个cookie的管理，先根据随机算法，选择1台后台服务器,，将该服务器值写入route，下次从route中取出，从而选择调度这台机器

参考网页:http[s://code.google.com/p/nginx-sticky-module/wiki/Documentation](https://code.google.com/p/nginx-sticky-module/wiki/Documenta)

nginx.conf

*upstream tomcat {*

*sticky name=cookieD;*

*server 192.168.1.181:8080;*

*server 192.168.1.182:8080;*

*}*

支持用户自己指定的key，stickly的用法请参考上述网页.

很显然对两种方式，应该采用该方式!JSESSIONID与route互不影响。

* FAIR算法[扩展]

fair策略是扩展策略，其原理是根据后端服务器的响应时间判断负载情况，从中选出负载最轻的机器进行分流。这种策略具有很强的自适应性，但是实际的网络环境往往不是那么简单，因此要慎用。

* 通用HASH算法[扩展]

这两种也是扩展策略，在具体的实现上有些差别，通用hash比较简单，可以以nginx内置的变量为key进行hash，一致性hash采用了nginx内置的一致性hash环，可以支持memcache。

# Keepalived搭建

## 1.安装keepalived

#apt-get install keepalived

## 2.准备keepalived检测nginx的shell脚本

#vi /etc/nginx/conf.d/check\_nginx.sh

*#!/bin/bash*

*#programme:checkscript,whennginx is not startup,thenstartup,ifnginxcan notstartup,then kill all keepalived process*

*#called by keepalived*

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

*if [ $A -eq 0 ];then*

*/etc/init.d/nginx start*

*sleep 3*

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

*killallkeepalived*

*fi*

*fi*

*如果要做得很完美,还应该有个外部脚本,来检测keepalived*

## 3.新建或修改keepalived配置文件

#vi /etc/keepalived/keepalived.conf (安装后没有该文件，需要新建)

*global\_defs {*

*router\_id nginx1#MASTER:nginx1,BACKUP:nginx2*

*}*

*vrrp\_scriptnginx {*

*script "/etc/nginx/conf.d/check\_nginx.sh"*

*interval 2*

*weight 2*

*nopreempt*

*}*

*vrrp\_instance VI\_1{*

*virtual\_router\_id 53#MASTER、BACKUP必须一致且同网段内不能重复,否则检测不到对方#的vrrp心跳包*

*advert\_int 1*

*priority 101 #MASTER > BACKUP*

*state MASTER #MASTER,BACKUP*

*interface p4p1*

*virtual\_ipaddress {*

*192.168.203.218/16 dev p4p1*

*}*

*track\_script {*

*nginx*

*}*

*nopreempt*

*}*

## 4.启动keepalived

#service keepalived start

#service keepalived stop

#service keepalived restart

# Nginx原理

## 1.Nginx架构

<http://tengine.taobao.org/book/chapter_2.html>

<http://blog.csdn.net/yankai0219/article/details/8018275>

## 2.nginx与apache区别

apache:一个请求来到apache,会为这个请求分配1个新的进程(或从进程池找一个空闲的进程),该进程会为当前请求生成1个新的线程,  
随着请求的增多,会生成很多的进程(或线程),在多进程(或多线程)竞争CPU资源的时候,会造成cpu资源的极大浪费,  
,因为在多进程(或多线程)之间工作时，cpu上下文切换会消耗很多的cpu资源。  
  
nginx:nginx采用master主进程和多个worker(与cpu核数一致)进程工作的模式，一个请求发到nginx,nginx-worker进程会竞争这个请求，  
同时一个work进程只有  
一个主线程,为了满足1个主线程服务多个请求,nginx采用了异步非阻塞的事件模型(循环处理多个准备好的事件)

# Keepalived原理

官网: http://www.keepalived.org/

## 1.keepalived原理

**VRRP协议:**

VRRP:Virtual Router Redundancy(虚拟路由冗余协议)简介:在现实的网络环境中，两台需要通信的主机大多数情况下并没有直接的物理连接。对于这样的情况，它们之间路由怎样选择？主机如何选定到达目的主机的下一跳路由,方案通常有两种，A:动态路由协议(RIP、OSPF等)、B:静态路由。但是动态路由维护成本很高，使得静态路由变得十分流行，但是是路由器(或者说默认网关default gateway)却经常成为单点。VRRP的目的就是为了解决静态路由单点故障问题。VRRP通过一竞选(election)来动态的将路由任务交给LAN中虚拟路由器中的某台VRRP路由器。

**Keepalived是基于VRRP协议的完美实现:**

Keepalived是一个基于VRRP协议(Virtual Router Redundancy Protoco:)来实现的WEB服务高可用方案，可以利用其来避免单点故障,一个WEB服务至少会有2台服务器运行Keepalived，一台为主服务器（MASTER），一台为备份服务器（BACKUP），但是对外表现为一个虚拟IP，主服务器会发送特定的消息给备份服务器，当备份服务器收不到这个消息的时候，即主服务器宕机的时候，备份服务器就会接管虚拟IP，继续提供服务，从而保证了高可用性

## 2.同类产品比较以及选择

Keepalived和Heartbeat是两款比较出名的高可用方案，两者各有优缺点，keepalived使用更简单，从安装、配置、使用、维护上keepalived比heartbeat要简单得多，尤其是heartbeat后来拆分成多个子项目后，安装、配置、使用都比较复杂。但是heartbeat功能上更为强大，配套工具更齐全，适合做大型集群管理。协议不同：Keepalived使用VRRP协议进行通信和选举，其目的是为了模拟路由器的双机；而 Heartbeat是基于主机或网络的服务的高可用方式，目的是用户service的双机，另外Heartbeat除了走网络外，还可以通过串口通信，貌似更可靠；使用建议:优先使用Keepalived，当Keepalived不够用的时候才选择Heartbeat