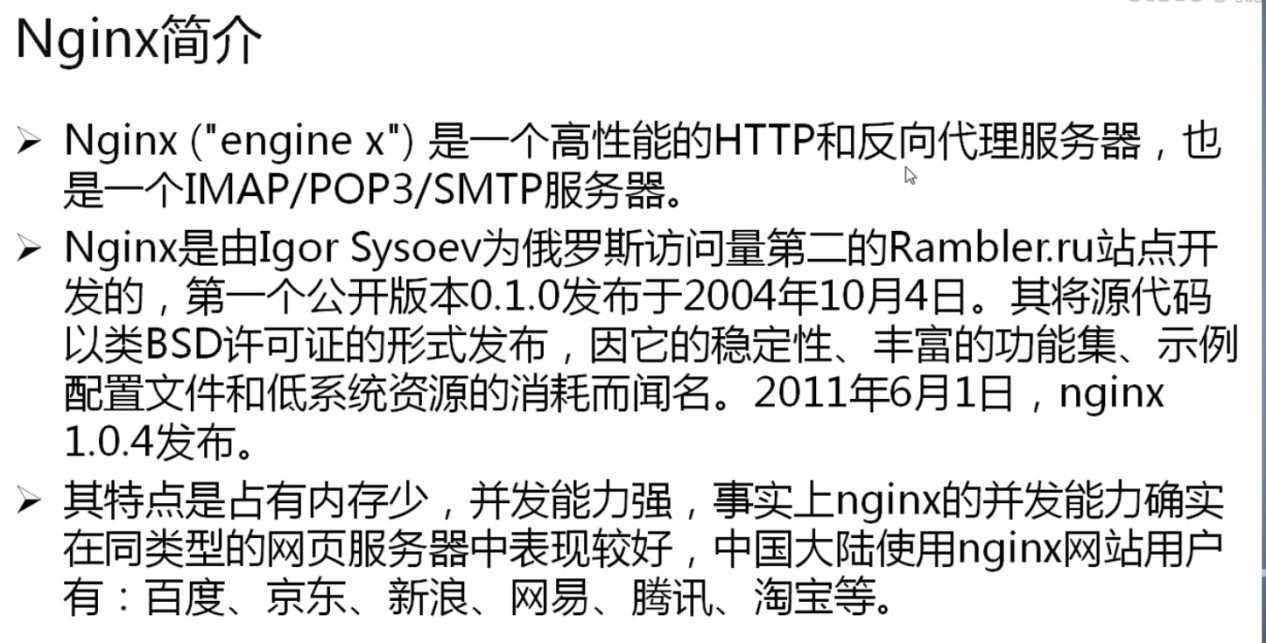
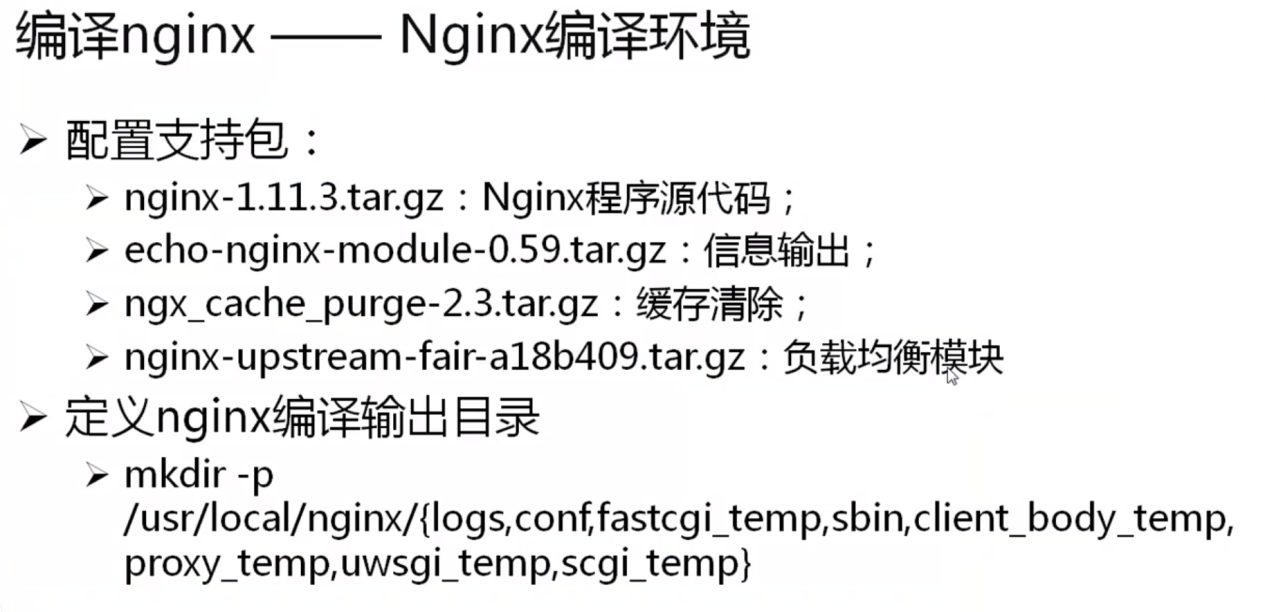
# Nginx

1. nginx 简介





1. 将所有的源代码程序包一起上传到Linux服务器
2. 将所有的开发包解压到指定目录：

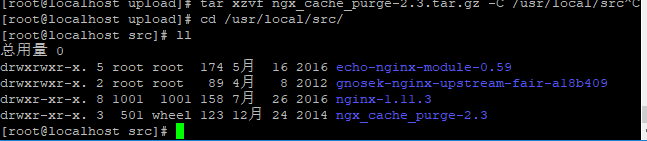
tar zxvf echo-nginx-module-0.59.tar.gz -C /usr/local/src/

tar xzvf nginx-1.11.3.tar.gz -C /usr/local/src/

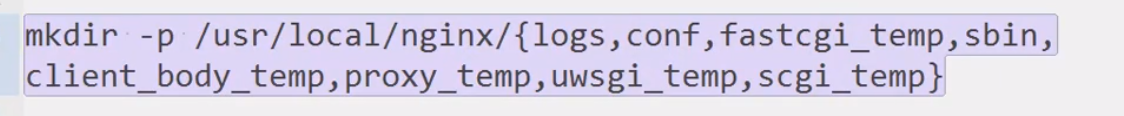
tar xzvf nginx-upstream-fair-a18b409.tar.gz -C /usr/local/src/

tar xzvf ngx\_cache\_purge-2.3.tar.gz -C /usr/local/src/

1. 进入到指定目录，查看指定目录是否存在



1. 默认情况下，nginx源代码会保存到源代码目录中，这样操作不方便管理，建议单独建立一个文件目录，保存nginx相关配置，假设目录名称：、usr/local/nginx;这里面需要一堆的子文件夹，



mkdir -p /usr/local/nginx/{logs,conf,fastcgi\_temp,sbin,client\_body\_temp,proxy\_temp,uwsgi\_temp,scgi\_temp}

1. 执行如下命令进行编译操作：

|-首先配置相关配置项

进入cd /usr/local/src/nginx-1.11.3,执行以下命令：

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

--sbin-path=/usr/local/nginx/sbin/ \

--with-http\_ssl\_module \

--conf-path=/usr/local/nginx/conf/nginx.conf \

--pid-path=/usr/local/nginx/logs/nginx.pid \

--error-log-path=/usr/local/nginx/logs/error.log \

--http-log-path=/usr/local/nginx/logs/access.log \

--http-fastcgi-temp-path=/usr/local/nginx/fastcgi\_temp \

--http-client-body-temp-path=/usr/local/nginx/client\_body\_temp \

--http-proxy-temp-path=/usr/local/nginx/proxy\_temp \

--http-uwsgi-temp-path=/usr/local/nginx/uwsgi\_temp \

--http-scgi-temp-path=/usr/local/nginx/scgi\_temp \

--add-module=/usr/local/src/echo-nginx-module-0.59 \

--add-module=/usr/local/src/gnosek-nginx-upstream-fair-a18b409 \

--add-module=/usr/local/src/ngx\_cache\_purge-2.3

以上编译过程中，可能会出现：

(1)、checking for C compiler ... not found

执行以下命令即可：

yum -y install gcc gcc-c++ autoconf automake make

(2)、./configure: error: the HTTP rewrite module requires the PCRE library.

执行以下命令：

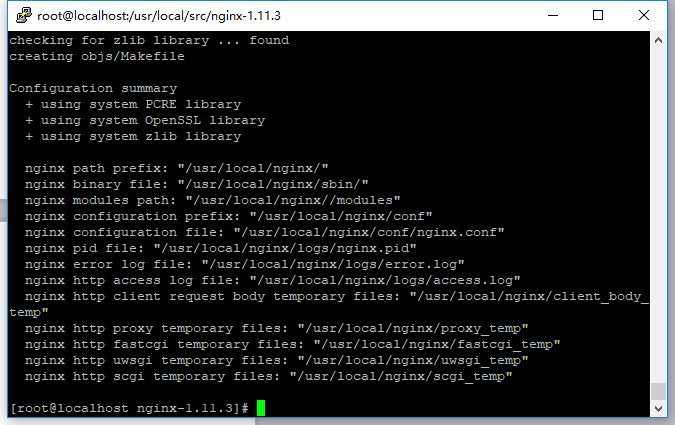
yum -y install pcre-devel

(3)、./configure: error: SSL modules require the OpenSSL library.

执行以下命令：

yum -y install openssl openssl-devel

Ok,在重新执行nginx的编译操作：



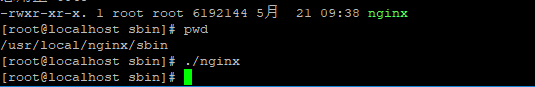
|-执行编译操作

make

|-执行nginx的安装操作：

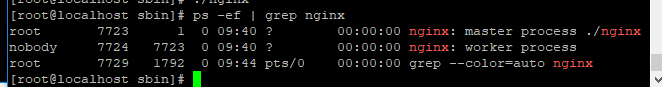
make install

1. 启动nginx

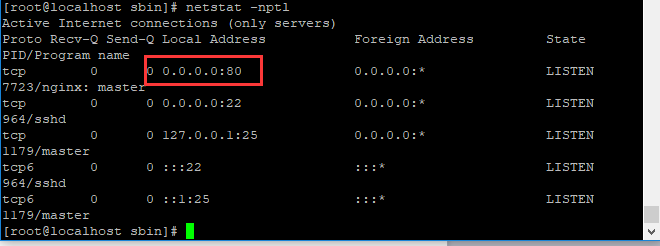


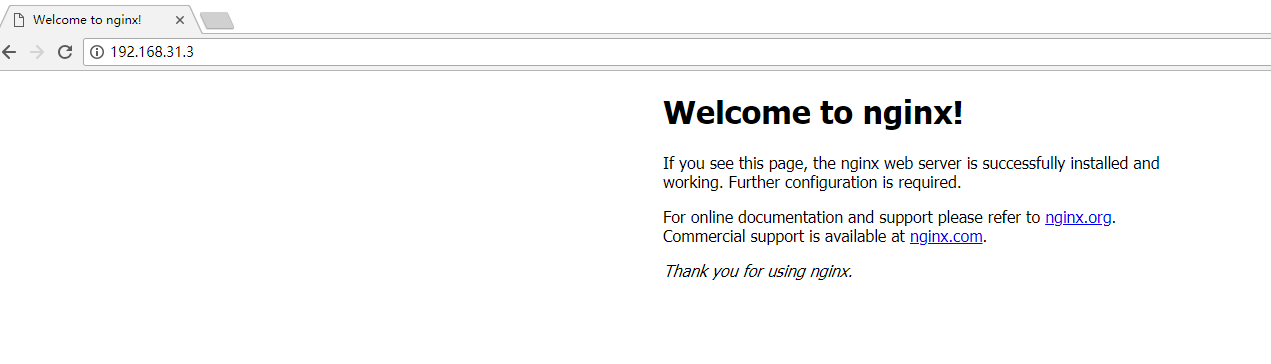
查看是否有nginx进程：

ps -ef | grep nginx



默认情况下，nginx将占用80端口：





总结：nginx开发的过程中需要绑定多个模块，所以在配置之前一定要进行相应的配置。

Nginx的所有配置项：/usr/local/nginx/conf/nginx.conf

## 配置nginx

Nginx可以将多个web容器整合到一起使用。

### nginx与Tomcat整合

|  |  |  |  |
| --- | --- | --- | --- |
| No | 主机 | IP | 描述 |
| 1 | xxx | 192.168.31.3 | nginx所在的服务器 |
| 2 | Xxx | 192.168.31.5 | Tomcat所在的服务器 |

本次实现的操作模式就是间接的使用nginx访问Tomcat。

1. 在【Tomcat-server-01】上配置一个虚拟目录：

* 建立一个Tomcat工作目录：mkdir -p /usr/data/tomcat/myweb
* 将Tomcat下的Root/WEB-INF,拷贝到此目录中：

cp -r /data/software/apache-tomcat-8.5.31/webapps/ROOT/WEB-INF/ /data/myweb/

* 编写一个简单的jsp文件，放到Tomcat的工作目录下，
* 修改Tomcat的配置文件，追加虚拟目录；

<Context path="/你设置的路径" docBase="你的web应用路径" />，即：

<Context path="/yangmi" docBase="/data/myweb" />

* 启动Tomcat服务：

/usr/local/tomcat/bin/catalian.sh start或./startup.sh

查看Tomcat是否启动jps或ps -ef| grep tomcat

* 打开浏览器访问：

1. 【nginx-server-01】通过nginx要求代理当前的Tomcat服务；

* 修改nginx配置文件（nginx.conf）：

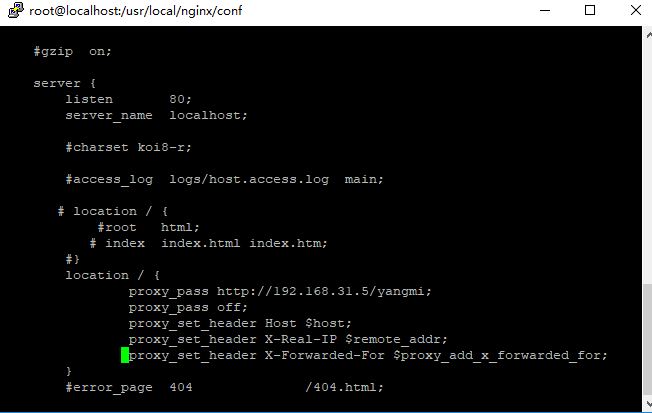
proxy\_pass http://192.168.31.5:8080/;

proxy\_redirect off;

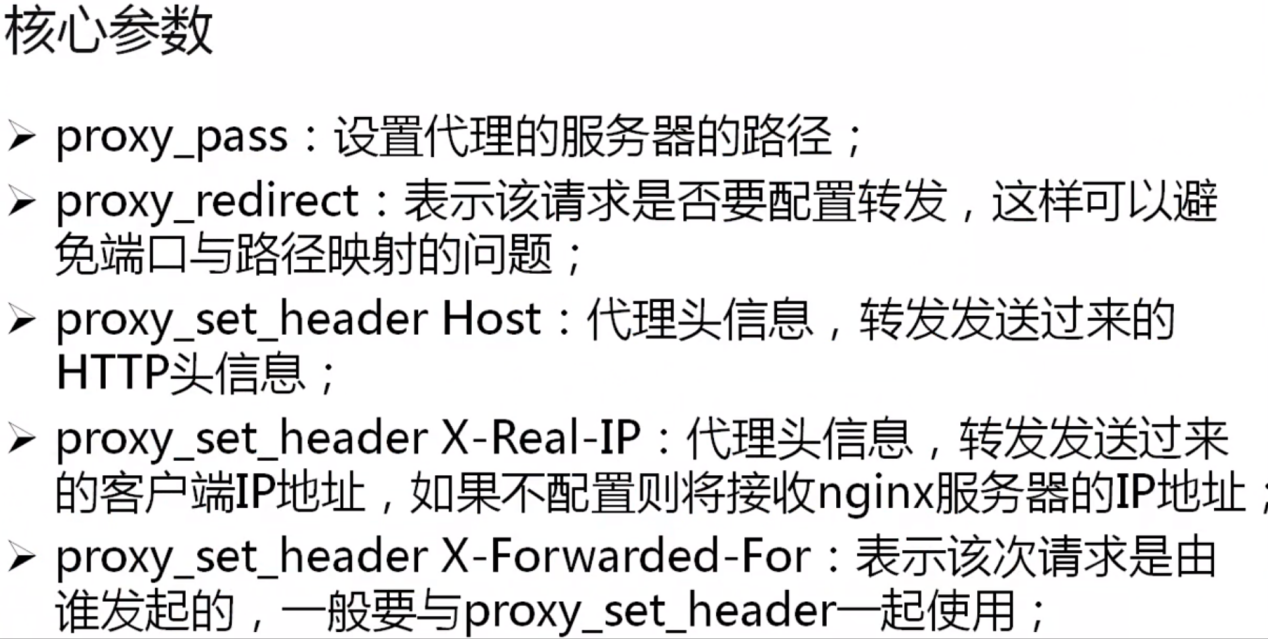
proxy\_set\_header Host $host;

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

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



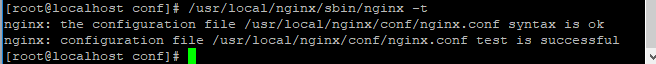
可参看官网的例子：http://www.nginx.cn/doc/



1. 【nginx-server-01】现在的nginx已启动，如果想加载新的配置，最好先检测下，而后在重新加载；

* 检测配置是否正确：

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



以上检测只是告诉用户没有语法错误。现在nginx已启动着，可以动态加载配置：

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

此时就可以实现nginx反向代理Tomcat。

### 负载均衡

负载均衡的本质是一个nginx可以代理多个Tomcat服务器，形成一个Tomcat集群存在。也就是说集群中的任何一台主机都将可能成为nginx服务主机。

1. 【tomcat-server-\*】修改主机名称：vi/etc/hostname
2. 【tomcat-server-\*】修改各自的index.jsp文件内容，追加各自的服务器名称

<%

InetAddress addr = InetAddress.getLocalHost();

out.println("主机地址："+addr.getHostAddress());

out.println("主机名："+addr.getHostName());

%>

1. 【tomcat-server-\*】两台服务器，启动各自的Tomcat
2. 【nginx-server-01】修改配置文件，追加负载均衡配置。

* 追加参与负载均衡的所有主机项，与server配置项组平行

upstream myproject {

server 192.168.31.5:8080;

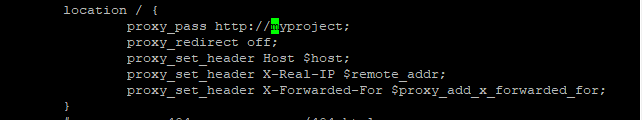
#server 192.168.31.5:8081;

server 192.168.31.7:8080;

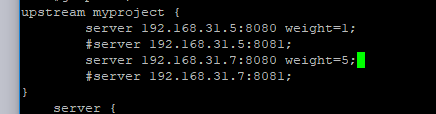
#server 192.168.31.7:8081;

}

* 修改代理配置项，让其跳转到“myproject”上：

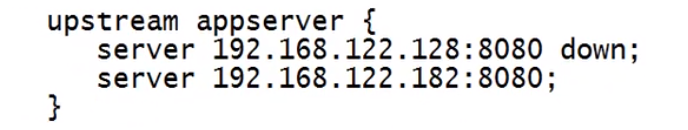


1. 【nginx-server-01】检测配置，然后加载配置。
2. 【nginx-server-01】增加权重配置，如果参与集群运算的Tomcat主机，性能不同，那么可以配置不同的权重，修改nginx配置文件：

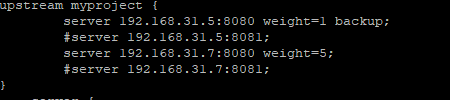


1. 【nginx-server-01】还可以配置手工的宕机或备份主机：

* down:表示该主机不参与服务；

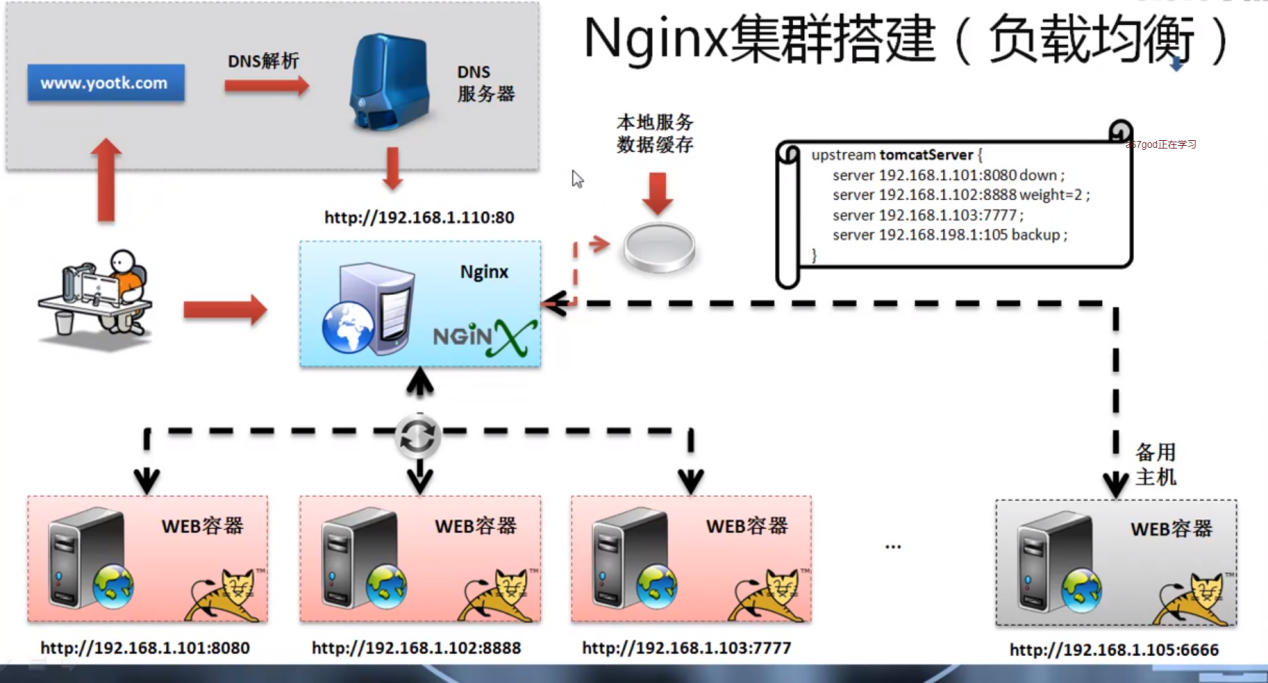


* Backup:表示该主机正常访问中不使用，如果所有的主机性能不够的时候自动启动该配置项；



### 缓存配置

如果只是做了简单的代理控制，静态的文件信息，都要反复传递，那么一定会造成网络资源的浪费，最好的解决方法是将一些几乎不会改变的信息，直接缓存到nginx服务器上。对于缓存还必须提供有清除机制。



1. 【nginx-server-01】上建立缓存的数据目录：

mkdir -p /usr/local/nginx/{temp,cache}

2、修改【nginx-server-01】上nginx的配置文件，

1. 设置代理的临时保存目录：

proxy\_temp\_path /usr/local/nginx/temp ;

1. 配置所有的缓存路径，其中“cache\_one”就是缓存中使用的名字，在服务器的配置上会使用到这个名字：

proxy\_cache\_path /usr/data/nginx/cache levels=1:2 keys\_zone=cache\_one:200m inactive=1d max\_size=30g;

1. 用来控制同时传输到客户端的buffer的数量：

proxy\_busy\_buffers\_size 128k;

1. 启用缓存:

proxy\_cache cache\_one;

1. 进行清空：

location ~ /purge(/.\*) {

allow 127.0.0.1;

allow 192.168.122.0/24(24表示匹配192.168.122网段16表示匹配192.168网段);

allow all;

proxy\_cache\_purge cache\_one $host$1$is\_args$args;

}

1. 所有要缓存的数据：

location ~ .\*\.(gif|jpg|png|htm|html|css|js|flv|ico|swf)(.\*) {

proxy\_pass http://myproject;

proxy\_redirect off;

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

proxy\_cache cache\_one;

proxy\_cache\_key $host$uri$is\_args$args;

proxy\_cache\_valid 200 302 1h;

proxy\_cache\_valid 301 1d;

proxy\_cache\_valid any 1m;

expires 30d;

}

范例：nginx缓存配置：

#user nobody;

worker\_processes 1;

#error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#pid logs/nginx.pid;

events {

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

proxy\_temp\_path /usr/local/nginx/temp ;

proxy\_cache\_path /usr/local/nginx/cache levels=1:2 keys\_zone=cache\_one:200m inactive=1d max\_size=30g;

proxy\_connect\_timeout 5;

proxy\_read\_timeout 60;

proxy\_send\_timeout 5;

proxy\_buffer\_size 16k;

proxy\_buffers 4 64k;

proxy\_busy\_buffers\_size 128k;

proxy\_temp\_file\_write\_size 128k;

gzip on;

gzip\_min\_length 1k;

gzip\_buffers 4 8k;

gzip\_http\_version 1.1;

gzip\_types text/plain application/x-JavaScript text/css application/xml;

gzip\_disable "MSIE [1-6]\.";

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

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

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

#access\_log logs/access.log main;

sendfile off;

#tcp\_nopush on;

#keepalive\_timeout 0;

keepalive\_timeout 65;

#gzip on;

upstream myproject {

server 192.168.31.5:8080 weight=1;

#server 192.168.31.5:8081;

server 192.168.31.7:8080 weight=1;

#server 192.168.31.7:8081;

}

server {

listen 80;

server\_name localhost;

error\_page 403 http://192.168.31.5:8080/403.jsp;

error\_page 404 http://192.168.31.5:8080/404.jsp;

#charset koi8-r;

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

location / {

# root html;

# index index.html index.htm;

proxy\_pass http://myproject ;

proxy\_redirect off;

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

#proxy\_cache cache\_one;

#proxy\_cache\_key $host$uri$is\_args$args;

#proxy\_cache\_valid 200 302 1h;

#proxy\_cache\_valid 301 1d;

}

location ~ /purge(/.\*) {

allow 127.0.0.1;

allow 192.168.31.0/24;

allow all;

proxy\_cache\_purge cache\_one $host$1$is\_args$args;

}

#注意，此规则必须放于purge之后，因为放之前就会优先匹配到此规则，会在清除缓存的时候报404错误。

location ~ .\*\.(gif|jpg|png|htm|html|css|js|flv|ico|swf)(.\*) {

proxy\_pass http://myproject ;

proxy\_redirect off;

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

proxy\_cache cache\_one;

proxy\_cache\_key $host$uri$is\_args$args;

proxy\_cache\_valid 200 302 1h;

proxy\_cache\_valid 301 1d;

proxy\_cache\_valid any 1m;

expires 30d;

}

#error\_page 404 /404.html;

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

#

proxy\_cache\_key $host$uri$is\_args$args;

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

location = /50x.html {

root html;

}

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

#

#location ~ \.php$ {

# proxy\_pass http://127.0.0.1;

#}

# 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;

# fastcgi\_index index.php;

# fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;

# include fastcgi\_params;

#}

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

# concurs with nginx's one

#

#location ~ /\.ht {

# deny all;

#}

}

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

#

#server {

# listen 8000;

# listen somename:8080;

# server\_name somename alias another.alias;

# location / {

# root html;

# index index.html index.htm;

# }

#}

# HTTPS server

#

#server {

# listen 443 ssl;

# server\_name localhost;

# ssl\_certificate cert.pem;

# ssl\_certificate\_key cert.key;

# ssl\_session\_cache shared:SSL:1m;

# ssl\_session\_timeout 5m;

# ssl\_ciphers HIGH:!aNULL:!MD5;

# ssl\_prefer\_server\_ciphers on;

# location / {

# root html;

# index index.html index.htm;

# }

#}

}

范例：清空缓存：

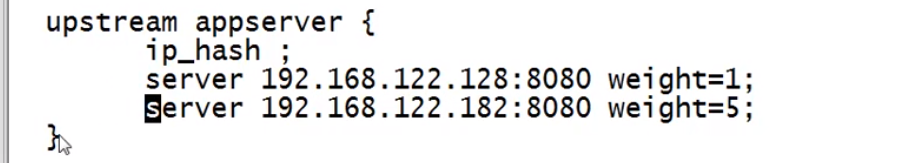
http://192.168.31.3/purge/images/ads-pic-01.jpg

## 集群session管理

### ip\_hash

在几成主机配置的时候，如果使用了ip\_hash的配置，那么就表示该用户只能在一台主机上操作，比如访问到的是TomcatA上，那么就被绑定在了tomcatA上，这样就解决了session问题。

范例：在配置文件上直接修改：



### redis管理（序列化session）

考虑到用户session可能跨越多个Tomcat保存，最好的做法是每一个Tomcat的session不再由自己管理，而是保存到一个公共的空间，所有Tomcat的session将通过此公共空间进行读取。可以通过Redis来实现，这一操作，与nginx没有关系了，只需要在所有的Tomcat上修改即可。

1. 下载redis与Tomcat管理session的开发包：

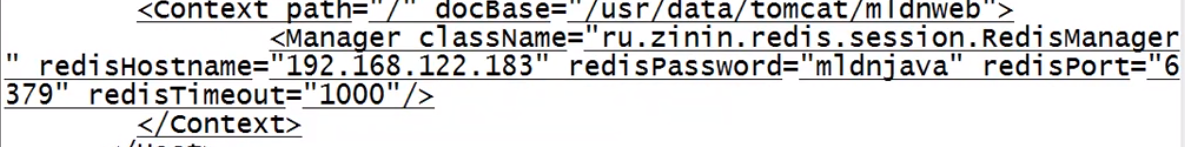


1. 将开发包拷贝到Tomcat/liib安装目录下
2. 【tomcat-server-01】修改server.xml文件，追加session管理机制：

<Context path="/yangmi" docBase="/data/myweb">

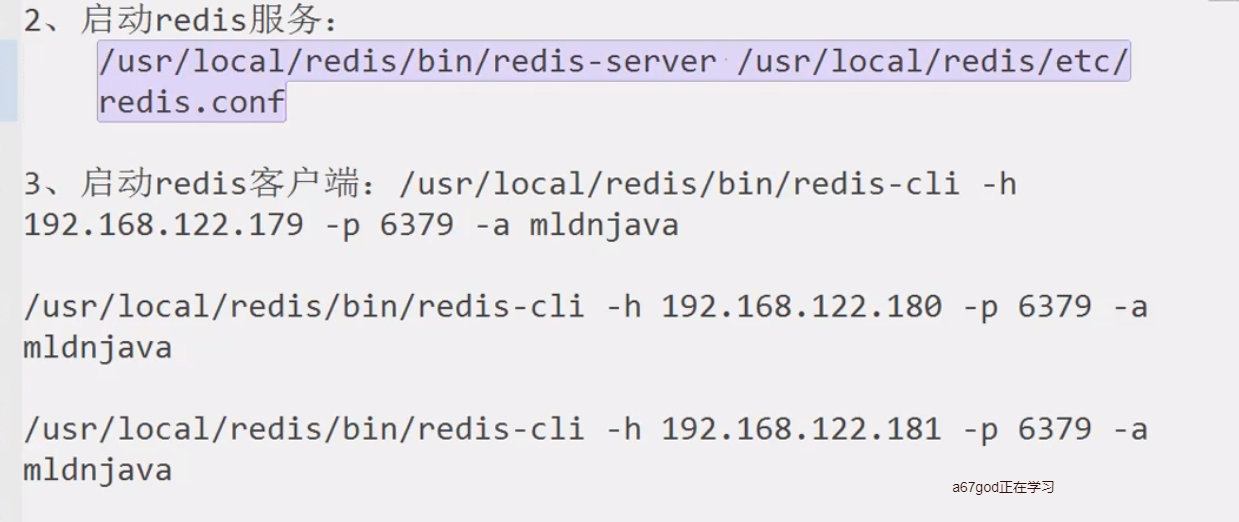
<Manager className="ru.zinin.redis.session.RedisManager" redisHostname="192.168.31.4" redisPassword="12345" redisPort=''6379" redisTimeout="3000"/>

</Context>



1. 将此配置信息发送给其他tomcat主机（不建议这么做，因为每一个tomcat的配置都可能不一样，这样会造成覆盖），将server.xml文件发送给servertomcat-02上的tomcat。

* 启动tomcat进程测试配置是否正确
* 启动Redis服务器

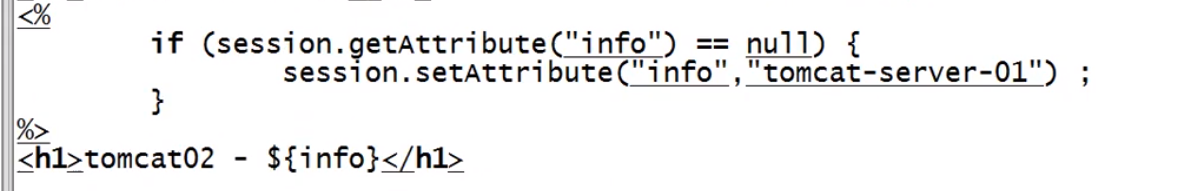


* 将server.xml发送给其他服务器：

scp /data/software/apache-tomcat-8.5.31/conf/server.xml 192.168.31.7:/data/software/apache-tomcat-8.5.31/conf/

1. 启动所有tomcat服务

此时所有session相关的数据都写入到了Redis中，设置的session属性一定可以直接使用 。



在测试的过程中发现，远程redis拒绝连接，这种情况下redis配置需要注释掉本地ip绑定，这样就允许所有ip访问了。这个问题困扰了我一天。

## 基于HA机制的Nginx配置实现

### Nginx问题分析

为避免单nginx单主机宕机，需要有另外的备用机，这种方案称为高可用（HA）.

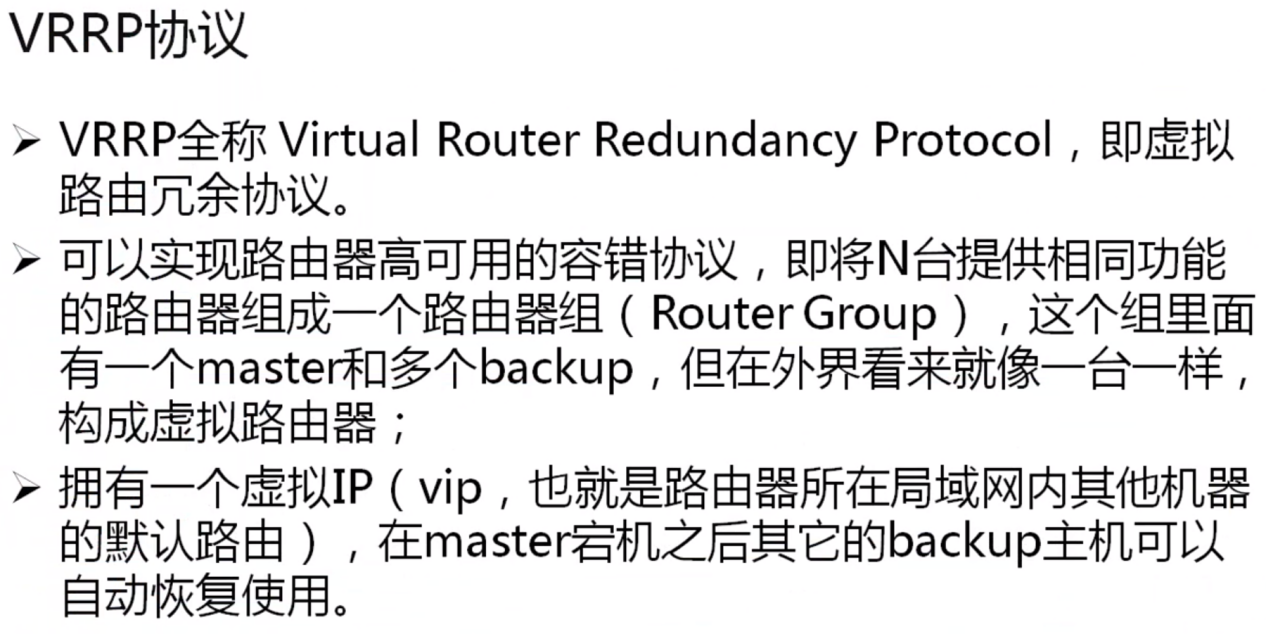
要使用Nginx进行HA配置，那么必须两个主机上的nginx配置相同，并且不应该同时进行服务，如果在工作的主机出现宕机，那么另外一台主机应该在很短的时间内恢复使用。

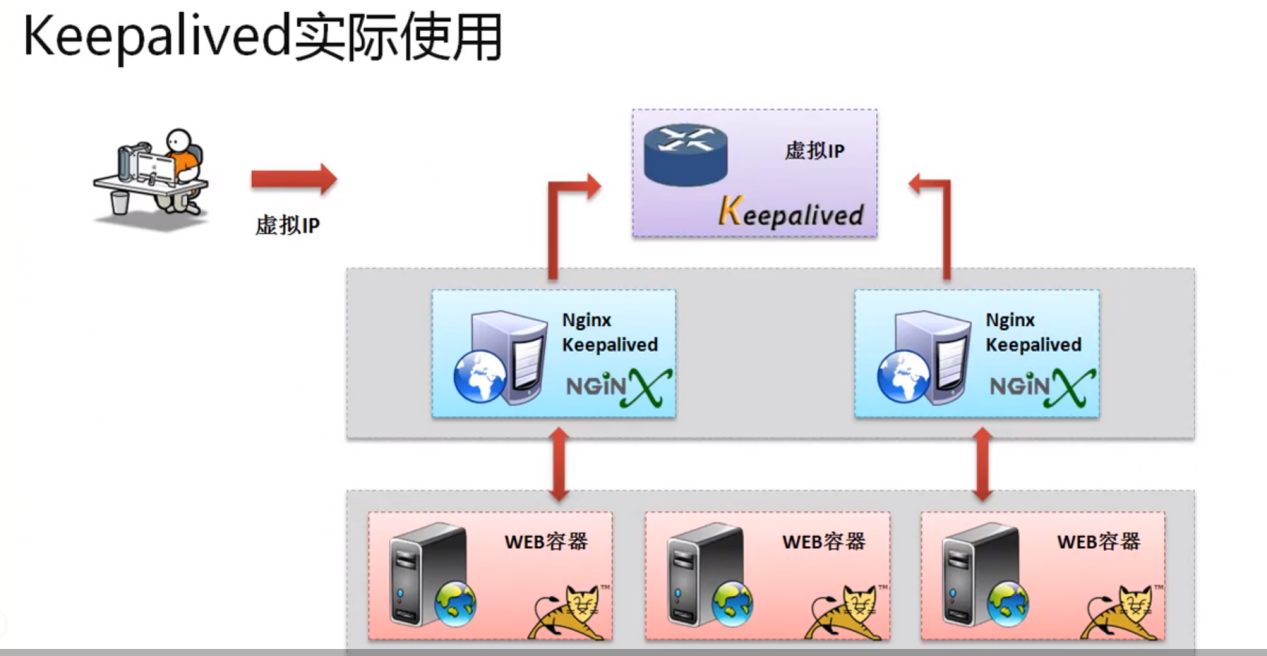
此时需要使用一个keepalive的组件，才可以实现这种HA的机制。

### Keepalive简介

Keepalive是一个免费的开源组件，其可以实现高可用的机制，很多第三方的整合组件，都需要keepalive组件进行协调整合处理。其最大的功能：采用一个虚拟路由算法，虚拟出一个新的IP地址，而后利用IP地址，代理多个真实服务器。



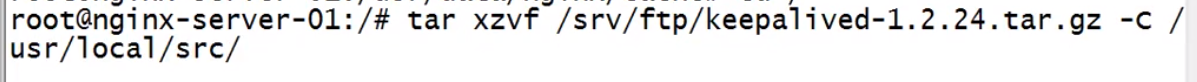




### HA机制Nginx实现

首先keepalive是一个二进制的源代码的版本，需要按照要求进行相应的配置。

1. 将组件上传到Linux服务器，进行解压缩：

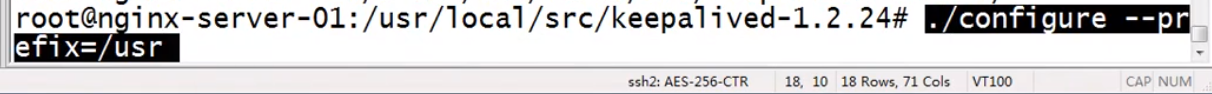


1. 进入keepalive的保存目录：



1. 配置keepalive的保存输出路径：

* 编译配置：

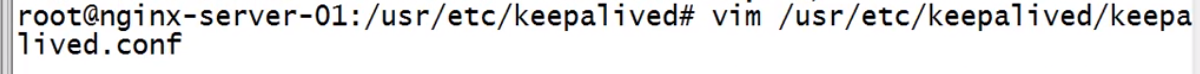


* 编译项目：make
* 程序安装：make install

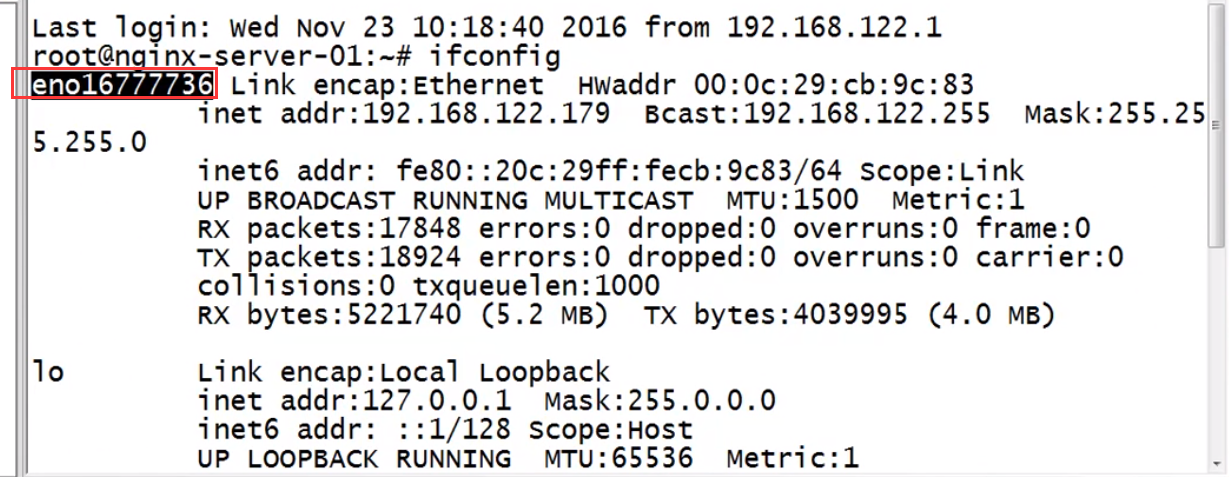
此时如果操作成功后，会在“/usr/ect/”中出现相关的配置项，以及“/usr/sbin”下也会出现可作为服务启动文件；

1. 对keepalive进行VRRP的配置：

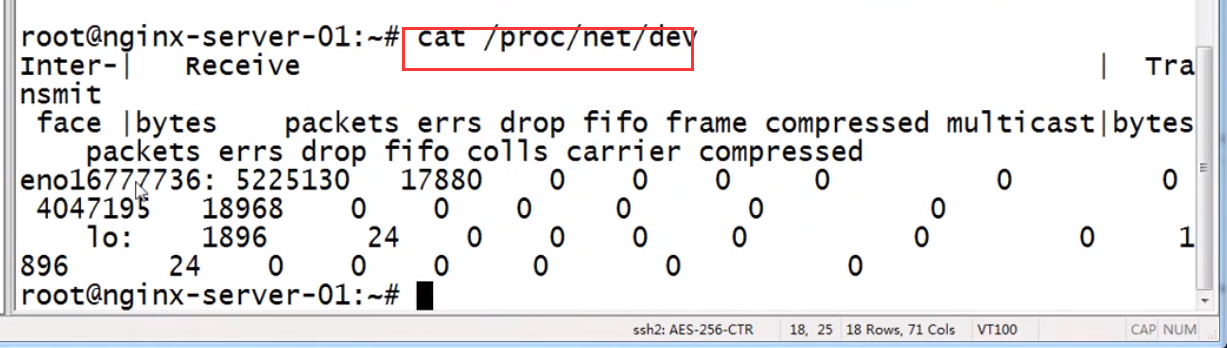
* 编辑文件：



* 需要取得网卡的名字：



或者是



* 有以下几个核心配置项：

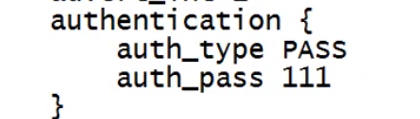
|-定义要进行虚拟IP设置的网卡名称：



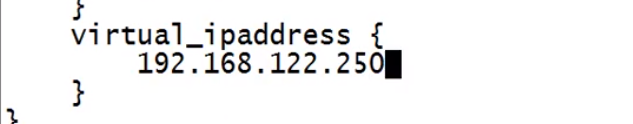
|-每一个拥有keepalive进程的优先级(值越大，级别越高):



|-进行keepalive的内部认证处理：



|-设置虚拟的IP地址，可以有多个

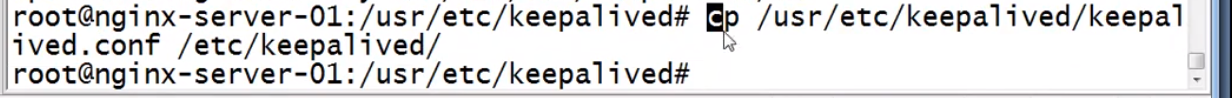


1. 将配置文件拷贝到“/etc/keepalived”目录下:

* 建立相应的目录：

mkdir -p /etc/keepalived

* 拷贝配置文件：



1. 启动服务观察是否可以正常使用

以下是centos 7以前的启动方法，centos7改为systemctl start keepalived.service.

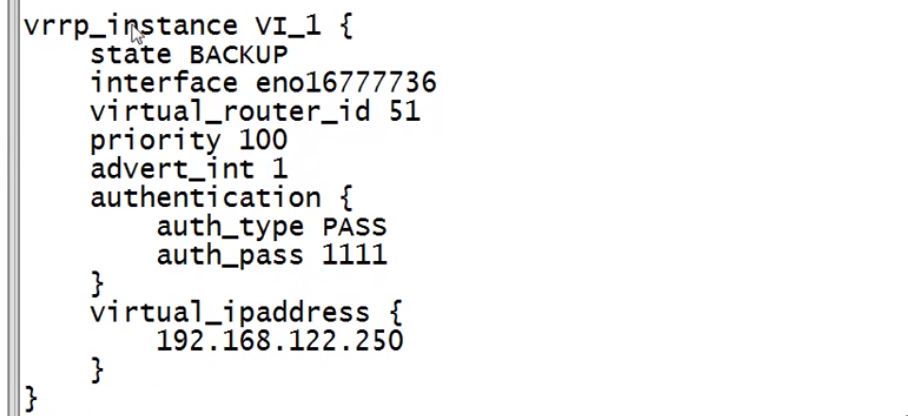


* 查看当前状态：



* 停止服务命令:service keepalived stop

1. 关闭当前的虚拟机，将其将其复制（克隆）为另外一个nginx主机；
2. 【nginx-server-02】需要将状态修改为BACKUP(最初为master)，确认ip地址是否与之前的配置相同（配置的备用主机IP）:



1. 此时不要直接使用nginx的地址访问服务器，通过虚拟地址访问（记得启动）：

测试关闭一台主机；

完整配置如下：

! Configuration File for keepalived

global\_defs {

notification\_email {

mldnqa@163.com

}

notification\_email\_from root

smtp\_server 127.0.0.1

smtp\_connect\_timeout 30

router\_id LVS\_DEVEL

vrrp\_skip\_check\_adv\_addr

vrrp\_strict

vrrp\_garp\_interval 0

vrrp\_gna\_interval 0

}

vrrp\_instance VI\_1 {

state BACKUP

interface eno16777736

virtual\_router\_id 51

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.122.250

}

}

virtual\_server 192.168.122.250 80 {

delay\_loop 6

lb\_algo rr

lb\_kind NAT

persistence\_timeout 50

protocol TCP

sorry\_server 192.168.122.168 80

real\_server 192.168.122.166 80 {

weight 1

HTTP\_GET {

url {

path /

status\_code 200

}

connect\_timeout 3

nb\_get\_retry 3

delay\_before\_retry 3

}

}

real\_server 192.168.122.168 80 {

weight 1

HTTP\_GET {

url {

path /

status\_code 200

}

connect\_timeout 3

nb\_get\_retry 3

delay\_before\_retry 3

}

}

}

问题记录：

1. In file included from vrrp\_daemon.c:37:0:

./../include/vrrp\_iproute.h:32:24: 致命错误：linux/mpls.h：没有那个文件或目录

#include <linux/mpls.h>

^

编译中断。

make[2]: \*\*\* [vrrp\_daemon.o] 错误 1

make[2]: 离开目录“/root/soft/keepalived-1.3.2/keepalived/vrrp”

make[1]: \*\*\* [all-recursive] 错误 1

make[1]: 离开目录“/root/soft/keepalived-1.3.2/keepalived”

make: \*\*\* [all-recursive] 错误 1

要是遇到以上类似的错误，建议使用最近几个版本的keepalive，或者是用yum安装下载。

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

解决：yum -y install libnl libnl-devel

3、configure: error: libnfnetlink headers missing

解决： yum install -y libnfnetlink-devel

1. 启动keepalived服务报错：Redirecting to /bin/systemctl start keepalived.service

解决：重启服务不再通过 service 操作，而是通过 systemctl 操作。

systemctl start keepalived.service

1. 5月 22 21:17:11 localhost.localdomain Keepalived\_vrrp[19042]: iptables command not available - can't filter IPv4 VIP address destinations；

解决：yum upgrade，然后重启服务器。

1. 启动防火墙失败：Failed to start firewalld.service: Unit not found.

解决：yum install firewalld

查看状态，看电脑上是否已经安装 firewalld

#systemctl status firewalld

#yum install firewalld  //安装firewalld 防火墙

#systemctl start firewalld.service   //开启防火墙

# systemctl stop firewalld.service   //关闭防火墙

# systemctl enable firewalld.service  //设置开机自动启动

# systemctl disable firewalld.service   //设置关闭开机制动启动

**注意一定要开放端口或关闭防火墙**