HAProxy

一.安装haproxy

```
# 安装lua
cd /lib
curl -R -O http://www.lua.org/ftp/lua-5.4.4.tar.gz
tar zxf lua-5.4.4.tar.gz
cd lua-5.4.4
make all test
rm -rf ../lua-5.4.4.tar.gz
# 下载压缩包
cd /opt
wget http://www.haproxy.org/download/2.6/src/haproxy-2.6.5.tar.gz
# 解压
tar -xf haproxy-2.6.5.tar.gz
# 编译
make -j PREFIX=/usr/local/haproxy TARGET=linux-glibc USE_PCRE=1 USE_OPENSSL=1 USE_ZLIB=1 USE_SYSTEMD=1 USE_CPU_AFFINITY=1 USE_LUA=1 LUA_INC=/lib/lua-5.4.4/src LUA_LIB=/lib/lua-5.4.4/src
# 移动文件夹
mv haproxy-2.6.5 /usr/local/haproxy
# 创建目录
mkdir -p /usr/local/haproxy/conf/
mkdir -p /usr/local/haproxy/logs/
# 创建用户haproxy
useradd -s /sbin/nologin haproxy
# 赋予权限
chown -R haproxy:haproxy /usr/local/haproxy
chmod -R 666 /usr/local/haproxy/logs/
# 创建配置文件
cat > /usr/local/haproxy/conf/haproxy.cfg << eof
global
# maxconn：设定每个haproxy进程可接受的最大并发连接数，此选项等同于Linux命令行选项"ulimit -n"。
```

```
  maxconn         100000
  chroot          /usr/local/haproxy
  stats socket  /var/lib/haproxy/haproxy.sock mode 600 level admin
# 设置运行haproxy进程的用户id和组id
  uid 1002
  gid 1002
# 设置haproxy进程进入后台运行。这是推荐的运行模式。
  daemon
# 设置haproxy启动时可创建的进程数
  #nbproc 1
  #cpu-map 1 0
  #cpu-map 2 1
  #cpu-map 3 2
  #cpu-map 4 3
# 指定haproxy进程的pid文件。启动进程的用户必须有访问此文件的权限。
  pidfile         /usr/local/haproxy/pid/haproxy.pid
# 全局的日志配置，local0是日志设备，info表示日志级别。其中日志级别有err、warning、info、debug。
  log 127.0.0.1 local3 info

#################################################### 监控模块 ##########################################################################
# 分通过listen关键字定义了一个名为“admin_status”haproxy的监控页面。
listen admin_status
  bind 127.0.0.1:9999
mode http
  log 127.0.0.1 local0 err
# 设置haproxy监控统计页面自动刷新的时间。
  stats refresh 15s
# 设置haproxy监控统计页面的URL路径，可随意指定。例如，指定“stats uri /haproxy-status”，就可以通过http://IP:9188/haproxy-status 查看。
  stats uri /haproxy-status
# 设置登录haproxy统计页面时密码框上的文本提示信息。
  stats realm welcome login\ haproxy
```

```
# 设置登录haproxy统计页面的用户名和密码。用户名和密码通过冒号分割。可为监控页面设置多个用户名和密码，每行一个。
  stats auth admin:admin~!@
# 用来隐藏统计页面上haproxy的版本信息。
  stats hide-version
# 通过设置此选项，可以在监控页面上手工启用或禁用后端真实服务器，仅在haproxy1.4.9以后版本有效。
  stats admin if TRUE

################################################## 时间检测
##########################################################################################
defaults
# 设置haproxy实例默认的运行模式，有tcp、http、health三个.
  mode http
# 设置成功连接到一台服务器的最长等待时间，默认单位是毫秒，但也可以使用其他的时间单位后缀。
  timeout connect 15s
# 设置连接客户端发送数据时最长等待时间，默认单位是毫秒，也可以使用其他的时间单位后缀。
  timeout client 20s
# 设置服务器端回应客户度数据发送的最长等待时间，默认单位是毫秒，也可以使用其他的时间单位后缀。
  timeout server 30s
# 设置对后端服务器的检测超时时间，默认单位是毫秒，也可以使用其他的时间单位后缀。
  timeout check 5s
eof
#创建启动脚本
cat > /usr/lib/systemd/system/haproxy.service <<'eof'
[Unit]
Description=haproxy
After=network.target
[Service]
PIDFile=/usr/local/haproxy/pid/haproxy.pid
Type=forking
ExecStart=/usr/local/haproxy/haproxy -f /usr/local/haproxy/conf/haproxy.cfg
```

```
ExecStop=/usr/bin/kill `/usr/bin/cat /usr/local/haproxy/logs/haproxy.pid`
ExecReload=/usr/local/haproxy/haproxy -f /usr/local/haproxy/conf/haproxy.cfg -st `/usr/bin/cat /usr/local/haproxy/logs/haproxy.pid`
PrivateTmp=true
[Install]
WantedBy=multi-user.target
eof

# 启动haproxy并将haproxy加入开机自启动
systemctl start haproxy
systemctl enable haproxy


##################################### haproxy优化
#####################################################
 maxconn：#最大连接数，根据应用实际情况进行调整，推荐使用10240
 daemon：#守护进程模式，haproxy可以使用非守护进程模式启动，建议使用守护进程模式启动
 nbprod：#负载均衡的并发进程数，建议与当前服务器CPU核数相等或为其2倍
 retries：#重试次数，主要用于对集群节点的检查，如果节点多，且并发量大，设置为2次或3次
 option http-server-close：#主动关闭http请求选项，建议在生产环境中使用此选项
 timeout http-keep-alive：#长连接超时时间，设置长连接超时时间，可以设置为10s
 timeout http-request：#http请求超时时间，建议将此时间设置为5~10s，增加http连接释放速度
 timeout client：#客户端超时时间，如果访问量过大，节点响应慢，可以将此时间设置短一些，建议设置为1min左右

################################### socat工具使用
#####################################################
# socat工具下载
yum install -y socat
```

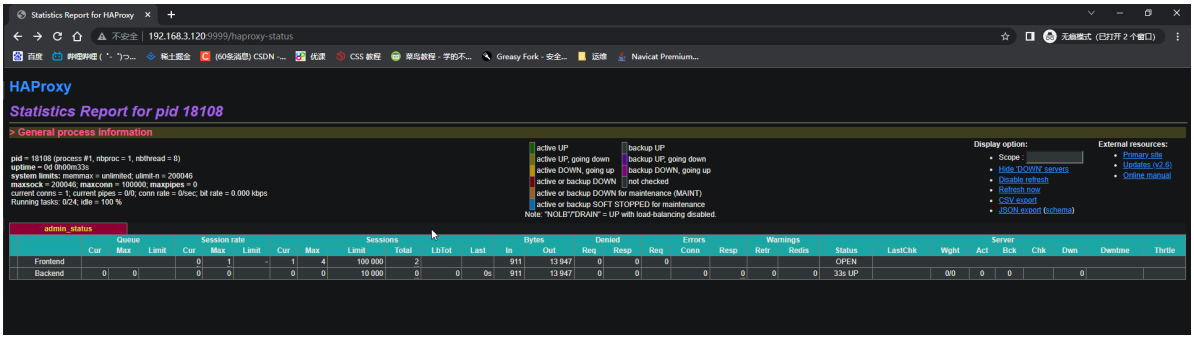
```
#利用管道符把help传给socat，进而标准重输入重定向到sockt文件中。实际为在此socket文件中输入help，获得帮助，只是获得此socket的帮助，不同socket的帮助不同，用法不同，不是所有socket都支持多种用法。
echo "help" | socat stdio
/var/lib/haproxy/haproxy.sock
# 查看后端backend参数信息
echo "show backend" | socat stdio
/var/lib/haproxy/haproxy.sock
echo "show info" | socat stdio
/var/lib/haproxy/haproxy.sock
echo "show servers state" | socat stdio
/var/lib/haproxy/haproxy.sock
# 查询backend的后端服务器权重，未设置，默认是1
echo "get weight nginx_test1/nginx1" | socat stdio
/var/lib/haproxy/haproxy.sock
echo "get weight nginx_test1/nginx2" | socat stdio
/var/lib/haproxy/haproxy.sock
# 修改weight，注意只针对单进程有效，因为socket发送消息，可能发到不同的进程，显示的效果就不一样，因此只适合与单进程。多进程的话除非每个进程都设置相同的值，否则每次就会返回不同的值，因此可以代用多进程配合多个socket文件，进行绑定，就能确保一一对应。
echo "set weight nginx_test1/nginx1 5" | socat stdio
/var/lib/haproxy/haproxy.sock
echo "get weight nginx_test1/nginx1" | socat stdio
/var/lib/haproxy/haproxy.sock
5 (initial 1)
# 将后端服务器禁用，注意只针对单进程有效，多进程需要绑定socket，分别执行
echo "set weight nginx_test1/nginx1 0" | socat stdio
/var/lib/haproxy/haproxy.sock
echo "set weight nginx_test1/nginx2 0" | socat stdio
/var/lib/haproxy/haproxy.sock
# 设置开启两个socket，并进行绑定，然后就能一对一的操作了
[root@superbox2 conf]# vim /etc/haproxy/haproxy.cfg
global
    maxconn 100000
    chroot /apps/haproxy
    #stats socket /var/lib/haproxy/haproxy.sock mode 600 level admin
```

```
    stats socket /var/lib/haproxy/haproxy.sock1 mode 600 level admin process 1
    stats socket /var/lib/haproxy/haproxy.sock2 mode 600 level admin process 2
# 通过对权重都设置为0，实现下线，这样比用交互式命令，直接使用命令，可以方便的脚本实现，这都是临时效果，服务重启这些修改都会丢失。
```

二.haproxy监控页

```
# 配置文件中的以下选项为开启监控页
listen admin_status
  bind 127.0.0.1:9999
mode http
  log 127.0.0.1 local0 err
# 设置haproxy监控统计页面自动刷新的时间。
  stats refresh 15s
# 设置haproxy监控统计页面的URL路径，可随意指定。例如，指定“stats uri /haproxy-status”，就可以通过http://IP:9188/haproxy-status 查看。
  stats uri /haproxy-status
# 设置登录haproxy统计页面时密码框上的文本提示信息。
  stats realm welcome login\ haproxy
# 设置登录haproxy统计页面的用户名和密码。用户名和密码通过冒号分割。可为监控页面设置多个用户名和密码，每行一个。
  stats auth admin:admin~!@
# 用来隐藏统计页面上haproxy的版本信息。
  stats hide-version
# 通过设置此选项，可以在监控页面上手工启用或禁用后端真实服务器，仅在haproxy1.4.9以后版本有效。
  stats admin if TRUE

############################################ 拓展 ##################################################
pid = 3698 (process #2, nbproc = 2, nbthread = 2) #pid为当前pid号，process为当前进程号，nbproc和nbthread为一共多少进程和每个进程多少个线程
uptime = 0d 0h00m08s #启动了多长时间
system limits: memmax = unlimited; ulimit-n = 131124 #系统资源限制：内存/最大打开文件数/
```

```
maxsock = 131124; maxconn = 65536; maxpipes = 0 #最大socket连接数/单进程最大连接数/最大管道数
maxpipes
current conns = 1; current pipes = 0/0; conn rate = 1/sec #当前连接数/当前管道数/当前连接速率
Running tasks: 1/9; idle = 100 % #运行的任务/当前空闲率
active UP: #在线服务器
backup UP: #标记为backup(后端)的服务器
active UP, going down: #监测未通过正在进入down过程
backup UP, going down: #备份服务器正在进入down过程
active DOWN, going up: #down的服务器正在进入up过程
backup DOWN, going up: #备份服务器正在进入up过程
active or backup DOWN: #在线的服务器或者是backup的服务器已经转换成了down状态
not checked: #标记为不监测的服务器
active or backup DOWN for maintenance (MAINT) #active或者backup服务器人为下线的
active or backup SOFT STOPPED for maintenance #active或者backup被人为软下线(人为将weight改成0)
```



backend server信息：

session rate(每秒的连接会话信息):	**Errors(错误统计信息):**
cur:每秒的当前会话数量	Req:错误请求量
max:每秒新的最大会话数量	conn:错误链接量
limit:每秒新的会话限制量	Resp:错误响应量
sessions(会话信息):	Warnings(警告统计信息):
cur:当前会话量	Retr:重新尝试次数
max:最大会话量	Redis:再次发送次数
limit: 限制会话量	
Total:总共会话量	Server(real server信息):
LBTot:选中一台服务器所用的总时间	Status:后端机的状态，包括UP和DOWN
Last：和服务器的持续连接时间	LastChk:持续检查后端服务器的时间
Wght:权重	
Bytes(流量统计):	Act:活动链接数量
In:网络的字节输入总量	Bck:备份的服务器数量
Out:网络的字节输出总量	Chk:心跳检测时间
Dwn:后端服务器连接后都是DOWN的数量	
Denied(拒绝统计信息):	Dwntme:总的downtime时间
Req:拒绝请求量	Thrtle:server 状态
Resp:拒绝回复量	

三.haproxy日志

1）haproxy支持5种日志格式:

1.默认格式：这是非常基本的，很少使用。它只提供关于当前传入连接的非常基本的信息它接受。

2.TCP格式：这是更先进的格式。这种格式在 "option tcplog "时启用。然后haproxy会等待连接结束后再进行记录。

3.HTTP格式，这是最先进的HTTP代理。这种格式当在前端设置了 "选项httplog "时被启用。它提供的信息与它提供了与TCP格式相同的信息，以及一些HTTP特定的字段，如请求，状态代码，以及头文件和cookies的捕获。

4.CLF HTTP格式，等同于HTTP格式，但其字段的排列顺序与HTTP格式相同。字段的排列顺序与CLF格式相同。在这种模式下，所有在这种模式下，所有的定时器、捕捉器、标志等都会在普通字段的末尾出现，每一个字段的顺序都是一样的。在这种模式下，所有的定时器、捕捉器、标志等都在每个字段的末尾出现，与它们在标准HTTP格式中出现的顺序一样，都是普通字段。

5.自定义日志格式，允许你制作你自己的日志行。

默认haproxy的日志是输出到系统的syslog中，查看起来不是非常方便，为了更好的管理haproxy的日志，我们在生产环境中一般单独定义出来。需要将haproxy的info及notice日志分别记录到不同的日志文件中。

日志级别:

> 0：emerg，系统不可用；
> 1：alert，必须马上采取行动的事件；
> 2：crit，关键的事件；
> 3：err，错误事件；
> 4：warning，警告事件；
> 5：notice，普通但重要的事件；
> 6：info，有用的信息；
> 7：debug，调试信息。

```
# 定义日志配置
# 全局的日志配置，local0是日志设备，info表示日志级别。其中日志级别有err、warning、info、debug。
vim /usr/local/haproxy/conf/haproxy.cfg
  log /dev/log local0 notice
  log /dev/log local0 err
  log /dev/log local0 warning
```

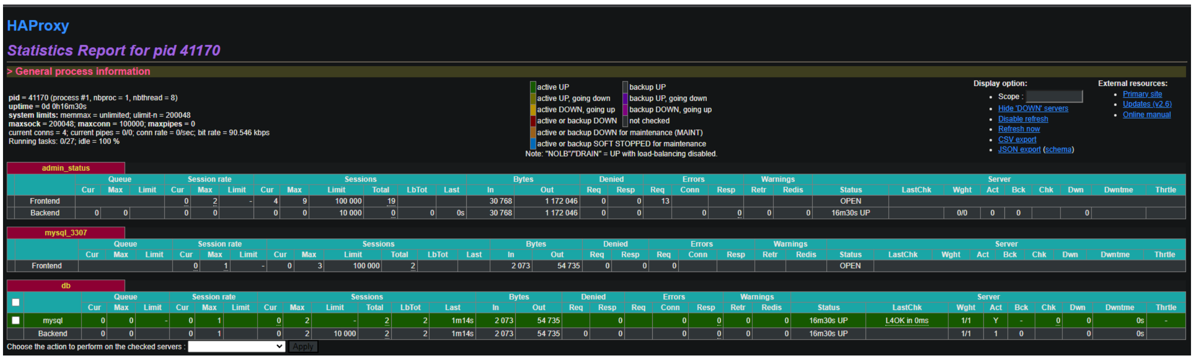
```
  log /dev/log local0 info
  log /dev/log local0 debug
# 打开TCP日志端口
# Provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
# 编辑日志配置文件
local3.* /var/log/haproxy/haproxyerr.log
local4.* /var/log/haproxy/haproxywarning.log
local6.* /var/log/haproxy/haproxyinfo.log
local7.* /var/log/haproxy/haproxydebug.log
# 创建日志文件夹
mkdir /var/log/haproxy
# 重启haproxy服务和日志服务
systemctl restart haproxy.service
systemctl restart rsyslog.service

```

四.haproxy代理

```
`+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  做7层代理(http)1  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+`
listen nginx_test1
  bind 192.168.3.120:80-84,:8080-8090
  mode http
  log  global

  balance  roundrobin
  server   nginx1 192.168.3.94:80      weight 1 check inter 3000 fall 2 rise 5
  server   nginx2 192.168.3.94:8080    weight 1 check inter 3000 fall 2 rise 5
`+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  做7层代理(http)2  +-+-+-+-+-+-+-+-+-++-+-+-++-+-+-+-+-+-+-+-+-+`
# 该写法现在不推荐
```

```
# 前端 frontend
frontend nginx_test1
  bind 192.168.3.120:80-84,:8080-8090
  mode http
  log  global
  use_backend test
# 后端 backend
backend test
  balance  roundrobin
  server   nginx1 192.168.3.94:80      weight 1 check inter 3000 fall 2 rise 5
  server   nginx2 192.168.3.94:8080    weight 1 check inter 3000 fall 2 rise 5
  `+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  4层代理(tcp)代理 MySQL  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+`
# 代理数据库为tcp4层代理
# 前端 frontend
frontend mysql_3307
  bind          :33080
  mode          tcp
# 后端 backend
  use_backend   db
backend db
  mode          tcp
  server        mysql   127.0.0.1:3307  check inter 3000 fall 2 rise 5
```

查看数据库代理是否启动：



HAProxy

Statistics Report for pid 41170

> General process information

pid = 41170 (process #1, nbproc = 1, nbthread = 8)
uptime = 0d 0h16m30s
system limits: memmax = unlimited; ulimit-n = 200048
maxsock = 200048; maxconn = 100000; maxpipes = 0
current conns = 4; current pipes = 0/0; conn rate = 0/sec; bit rate = 90.546 kbps
Running tasks: 0/27; idle = 100 %

active UP | backup UP | active UP, going down | backup UP, going down | active DOWN, going up | backup DOWN, going up | active or backup DOWN | not checked | active or backup DOWN for maintenance (MAINT) | active or backup SOFT STOPPED for maintenance
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Disable refresh
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.6)
- Online manual

admin_status

	Queue Cur	Queue Max	Queue Limit	Session rate Cur	Session rate Max	Session rate Limit	Sessions Cur	Sessions Max	Sessions Limit	Sessions Total	Sessions LbTot	Sessions Last	Bytes In	Bytes Out	Denied Req	Denied Resp	Errors Req	Errors Conn	Errors Resp	Warnings Retr	Warnings Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	2	-	4	9	100 000	19			30 768	1 172 046	0	0	13					OPEN								
Backend	0	0		0	0		0	0	10 000	0	0	0s	30 768	1 172 046	0	0		0	0	0	0	16m30s UP		0/0	0	0		0		

mysql_3307

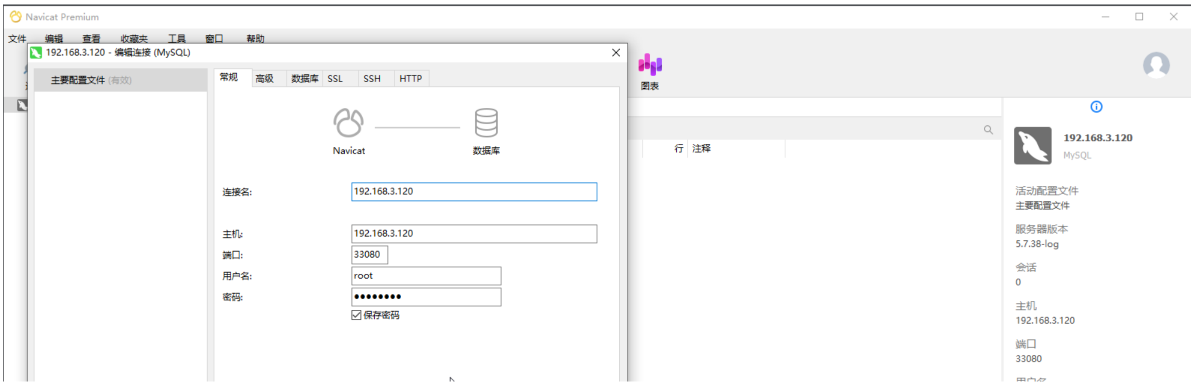
	Queue Cur	Queue Max	Queue Limit	Session rate Cur	Session rate Max	Session rate Limit	Sessions Cur	Sessions Max	Sessions Limit	Sessions Total	Sessions LbTot	Sessions Last	Bytes In	Bytes Out	Denied Req	Denied Resp	Errors Req	Errors Conn	Errors Resp	Warnings Retr	Warnings Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	3	100 000	2			2 073	54 735	0	0	0					OPEN								

db

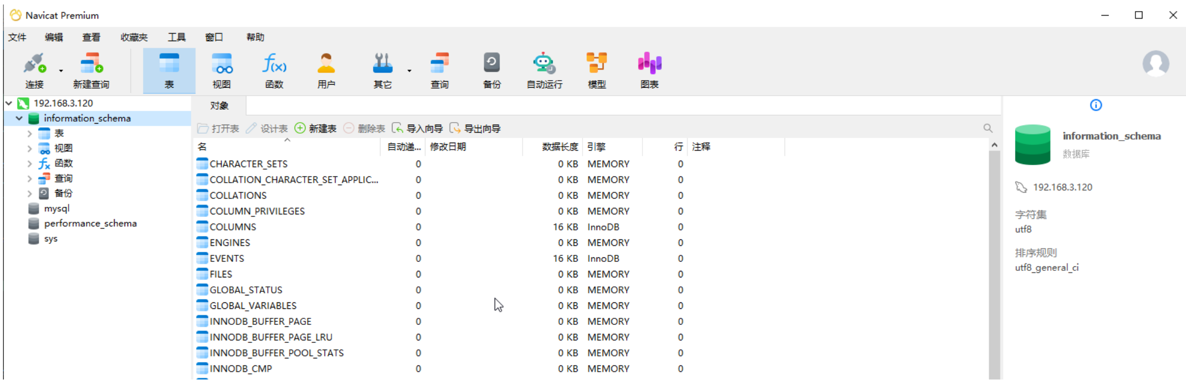
	Queue Cur	Queue Max	Queue Limit	Session rate Cur	Session rate Max	Session rate Limit	Sessions Cur	Sessions Max	Sessions Limit	Sessions Total	Sessions LbTot	Sessions Last	Bytes In	Bytes Out	Denied Req	Denied Resp	Errors Req	Errors Conn	Errors Resp	Warnings Retr	Warnings Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
mysql	0	0	-	0	1		0	2	-	2	2	1m14s	2 073	54 735		0		0	0	0	0	16m30s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	2	10 000	2	2	1m14s	2 073	54 735	0	0		0	0	0	0	16m30s UP		1/1	1	0		0	0s	

Choose the action to perform on the checked servers : Apply

尝试连接数据库：



发现可以成功连接



五.haproxy调度算法

```
`#+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 动态算法1 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+`
# roundrobin算法基于权重进行轮叫调度的算法，在服务器的性能分布比较均匀时，这是一种最公平合理，常用的算法。此算法使用较为频繁
listen nginx_gou
  bind 192.168.3.120:80-84,:8080-8090
  mode http
  log  global
  balance  roundrobin
  server   nginx1 192.168.3.94:80      weight 1 check inter 3000 fall 2 rise 5
  server   nginx2 192.168.3.94:8080    weight 1 check inter 3000 fall 2 rise 5
`#+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 静态算法2 +-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+`
# static-rr算法基于权重进行轮叫调度的算法，不过此算法为静态方法，在运行时调整期服务器权重不会生效,需要重启haproxy服务生效
listen nginx_che
  bind 192.168.3.120:80
  mode http
```

```
  log global
  balance  static-rr
  server   nginx3 192.168.3.94:80     weight 1 check inter 3000 fall 2 rise 5
  server   nginx4 192.168.3.94:8080   weight 1 check inter 3000 fall 2 rise 5
`#+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ first算法 +-+-+-+-+-+-+-+-+-+--+-++-+-+-+-+-+-+-+-+-+-+-+-+
# first算法根据服务器在列表中的位置，自上而下进行调度，但是其只会当第一台服务器的连接数达到上限，新请求才会分配给下一台服务，因此会忽略服务器的权重设置。
listen nginx_che
  bind 192.168.3.120:80
  mode http
  log global
  balance  first
  server   nginx3 192.168.3.94:80     maxconn 2 weight 1 check inter 3000 fall 2 rise 5
  server   nginx4 192.168.3.94:8080   weight 1 check inter 3000 fall 2 rise 5
```

验证first算法:

```
开三个shell窗口，分别在命令行输入：while true;do curl http://192.168.3.120/ ;done
```

shell1:

```
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
```

shell2:

```
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
```

shell3:

```
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
<!doctype html><html lang="zh"><head><title>HTTP状态 404 - 未找到</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;backg
round-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:
none;}</style></head><body><h1>HTTP状态 404 - 未找到</h1><hr class="line" /><p><b>类型</b> 状态报告</p><p><b>描述</b> 源服务器未能找到目标资源的表示或者是不愿公开一个已经存
在的资源表示。</p><hr class="line" /><h3>Apache Tomcat/8.5.82</h3></body></html>Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
<!doctype html><html lang="zh"><head><title>HTTP状态 404 - 未找到</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;backg
round-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:
none;}</style></head><body><h1>HTTP状态 404 - 未找到</h1><hr class="line" /><p><b>类型</b> 状态报告</p><p><b>描述</b> 源服务器未能找到目标资源的表示或者是不愿公开一个已经存
在的资源表示。</p><hr class="line" /><h3>Apache Tomcat/8.5.82</h3></body></html>Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
Welcom to nginxtest1!
<!doctype html><html lang="zh"><head><title>HTTP状态 404 - 未找到</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;backg
round-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:
none;}</style></head><body><h1>HTTP状态 404 - 未找到</h1><hr class="line" /><p><b>类型</b> 状态报告</p><p><b>描述</b> 源服务器未能找到目标资源的表示或者是不愿公开一个已经存
在的资源表示。</p><hr class="line" /><h3>Apache Tomcat/8.5.82</h3></body></html>Welcom to nginxtest1!
<!doctype html><html lang="zh"><head><title>HTTP状态 404 - 未找到</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;backg
round-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:
none;}</style></head><body><h1>HTTP状态 404 - 未找到</h1><hr class="line" /><p><b>类型</b> 状态报告</p><p><b>描述</b> 源服务器未能找到目标资源的表示或者是不愿公开一个已经存
在的资源表示。</p><hr class="line" /><h3>Apache Tomcat/8.5.82</h3></body></html>Welcom to nginxtest1!
<!doctype html><html lang="zh"><head><title>HTTP状态 404 - 未找到</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;backg
round-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:
none;}</style></head><body><h1>HTTP状态 404 - 未找到</h1><hr class="line" /><p><b>类型</b> 状态报告</p><p><b>描述</b> 源服务器未能找到目标资源的表示或者是不愿公开一个已经存
在的资源表示。</p><hr class="line" /><h3>Apache Tomcat/8.5.82</h3></body></html>
```

算法继续:

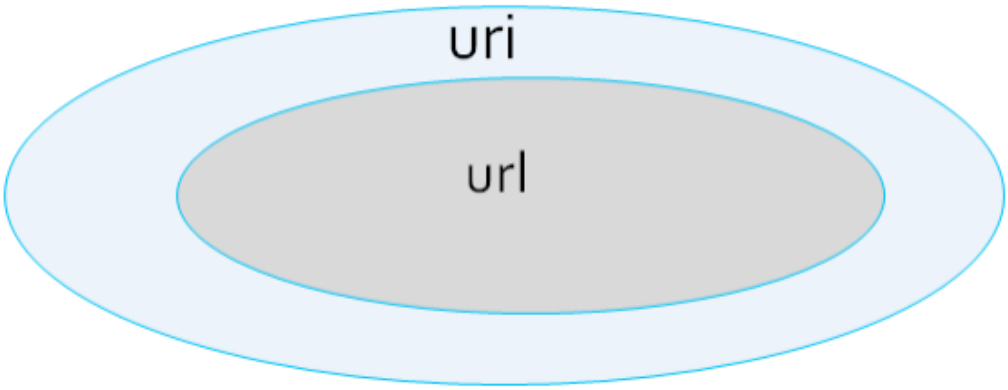
```
`+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ leastconn +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+`
# 数据库负载均衡mysql+haproxy的常用轮询方式,不适用于http。此算法会将新的连接请求转发具有最少连接数目的后端服务器。

listen nginx_test4
  bind 192.168.3.120:80
  mode http
  log  global
  balance  leastconn
  server   nginx7 192.168.3.94:80     weight 1 check inter 3000 fall 2 rise 5
  server   nginx8 192.168.3.94:8080   weight 1 check inter 3000 fall 2 rise 5
```

```
`+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  source  +-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+`
# 基于请求源IP的算法。此算法先对请求的源IP进行hash运算，可以使同一个客户端IP的请求始终被转发到某台特定的后端服务器

listen nginx_test5
  bind 192.168.3.120:80
  mode http
  log  global
  balance  source
  server   nginx9   192.168.3.94:80     weight 1 check inter 3000 fall 2 rise 5
  server   nginx10  192.168.3.94:8080   weight 1 check inter 3000 fall 2 rise 5

`+-+-+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+ 一致性hash +-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+`
# 一致性哈希，该hash是动态的，支持在线调整权重，支持慢启动，优点在于当服务器的总权重发生变化时，对调度结果影响是局部的，不会引起大的变动，hash（o）mod n 。

listen nginx_test6
  bind 192.168.3.120:80
  mode http
  log  global
  hash-type consistent
  server   nginx11   192.168.3.94:80     weight 1 check inter 3000 fall 2 rise 5
  server   nginx12   192.168.3.94:8080   weight 1 check inter 3000 fall 2 rise 5
```

六.haproxy uri

基于对用户请求的uri做hash并将请求转发到后端指定服务器，也可以通过map-based和consistent定义使用取模法还是一致性hash。

```
#URI/URL
http://example.org/absolute/URI/with/absolute/path/to/resource.txt
#URI/URL
ftp://example.org/resource.txt
#URI
/relative/URI/with/absolute/path/to/resource.txt
```



map-base取模法:

基于服务器总权重的hash数组取模，该hash是静态的即不支持在线调整权重，不支持慢启动，其对后端服务器调度均衡，缺点是当服务器的总权重发生变化时，即有服务器上线或下线，都会因权重发生变化而导致调度结果整体改变。 所谓取模运算，就是计算两个数相除之后的余数，10%7=3， 7%4=3，基于权重取模：(2^32-1)%(1+1+2)

uri 取模法配置示例:

基于对用户请求的URI的左半部分或整个uri做hash，再将hash结果对总权重进行取模后，根据最终结果将请求转发到后端指定服务器，适用于后端是缓存服务器场景，默认是静态算法，也可以通过hash-type

指定map-based和consistent，来定义使用取模法还是一致性hash。

意：此算法基于应用层，所以只支持 mode http ，不支持 mode tcp

七.使用socat工具动态调整HAProxy后端服务器权重

1.安装socat

```
yum install -y socat
```

2.查看帮助命令

```
echo "help" | socat stdio /var/lib/haproxy/haproxy.sock
```

3.常用命令

```
disable server : disable a server for maintenance (use 'set server' instead)
enable server  : enable a disabled server (use 'set server' instead)
get weight     : report a server's current weight
set weight     : change a server's weight (deprecated)
show info      : report information about the running process [desc|json|typed]*
```

4.查看haproxy信息

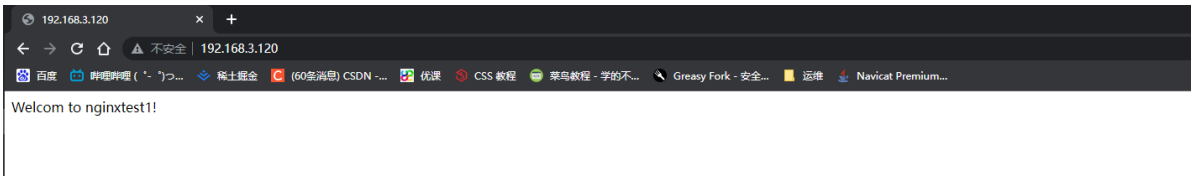
```
echo "show info" | socat stdio /var/lib/haproxy/haproxy.sock
```

5.使用的配置文件如下所示：

```
listen nginx_test1
  bind 192.168.3.120:80-84,:8080-8090
  mode http
  log  global
  balance  roundrobin
  server   nginx1 192.168.3.94:80      weight 1 check inter 3000 fall 2 rise 5
  server   nginx2 192.168.3.94:8080    weight 1 check inter 3000 fall 2 rise 5
# 重启haproxy
systemctl restart haproxy
```

6.浏览器测试

```
# 浏览器输入ip(192.168.3.120)
```



目前是正常轮询的。

7.使用socat查看server权重

```
# 使用命令 echo "get weight "listen"/"server" | socat stdio /haproxy.sock路径 查看server权重
# server1
echo "get weight nginx_test1/nginx1" | socat stdio /var/lib/haproxy/haproxy.sock
# server2
echo "get weight nginx_test1/nginx2" | socat stdio /var/lib/haproxy/haproxy.sock
```

```
[root@superbox2 conf]# echo "get weight nginx_test1/nginx1" | socat stdio /var/lib/haproxy/haproxy.sock
1 (initial 1)

[root@superbox2 conf]# echo "get weight nginx_test1/nginx2" | socat stdio /var/lib/haproxy/haproxy.sock
1 (initial 1)
```

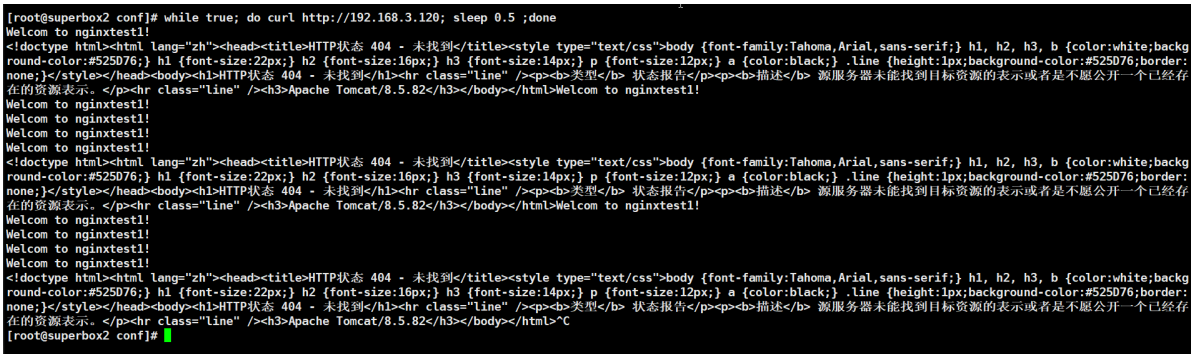
8.使用socat修改server权重

```
# server1
echo "set weight nginx_test1/nginx1 5" | socat stdio /var/lib/haproxy/haproxy.sock
# server2
echo "get weight nginx_test1/nginx2" | socat stdio /var/lib/haproxy/haproxy.sock
```

```
[root@superbox2 conf]# echo "set weight nginx_test1/nginx1 5" | socat stdio /var/lib/haproxy/haproxy.sock

[root@superbox2 conf]# echo "get weight nginx_test1/nginx2" | socat stdio /var/lib/haproxy/haproxy.sock
1 (initial 1)

[root@superbox2 conf]# echo "get weight nginx_test1/nginx1" | socat stdio /var/lib/haproxy/haproxy.sock
5 (initial 1)
```

9.测试修改权重后的访问结果

```
# 使用下面的测试语句测试
while true; do curl http://192.168.3.120; sleep 0.5 ;done
```



八、haproxy十种算法总结

```
静态：
static-rr---------->tcp/http     #做session共享的web集群
first------------->tcp/http     #使用较少

动态：
roundrobin-------->tcp/http
leastconn--------->tcp/http     #数据库
random------------>tcp/http

取决于是否有hash_type consistent，有为动态，默认静态：
source------------>tcp/http     #基于客户端公网IP的会话保持
uri--------------->http         #缓存服务器，CDN服务商，蓝汛、百度、阿里云、腾讯
url_param--------->http
hdr--------------->http         #基于客户端请求报文头部做下一步处理
rdp-cookie-------->tcp          #很少使用
```

```
uri、url_param、hdr只有在mode为http的情况下才能取到值，如果mode为tcp，balance自动更改为roundrobin。
动态与静态最主要区别：可以用socat动态调整权重、支持慢启动功能。
```

九、haproxy基于cookie的会话保持

cookie value：为当前server指定cookie值，实现基于cookie的会话黏性。

COOKIE 语法：

```
cookie <name> [ rewrite | insert | prefix ] [ indirect ] [ nocache ]
              [ postonly ] [ preserve ] [ httponly ] [ secure ]
              [ domain <domain> ]* [ maxidle <idle> ] [ maxlife <life> ]
              [ dynamic ] [ attr <value> ]*

name：cookie 的key名称，用于实现持久连接
insert：如果没有就插入新的cookie
indirect：不会向客户端发送服务器已经处理过请求的cookie信息
nocache：当client和hapoxy之间有缓存时，不缓存cookie
```

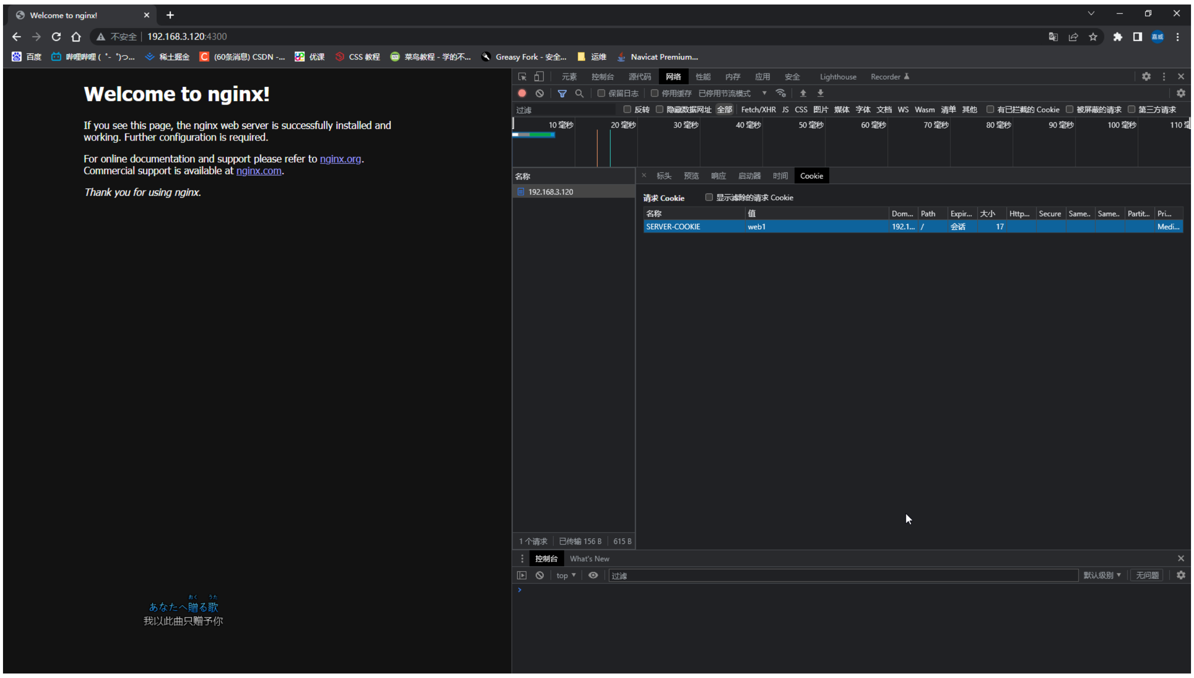
COOKIE 的作用域：

defaults	frontend	listen	backend
yes	no	yes	yes

COOKIE 配置示例：

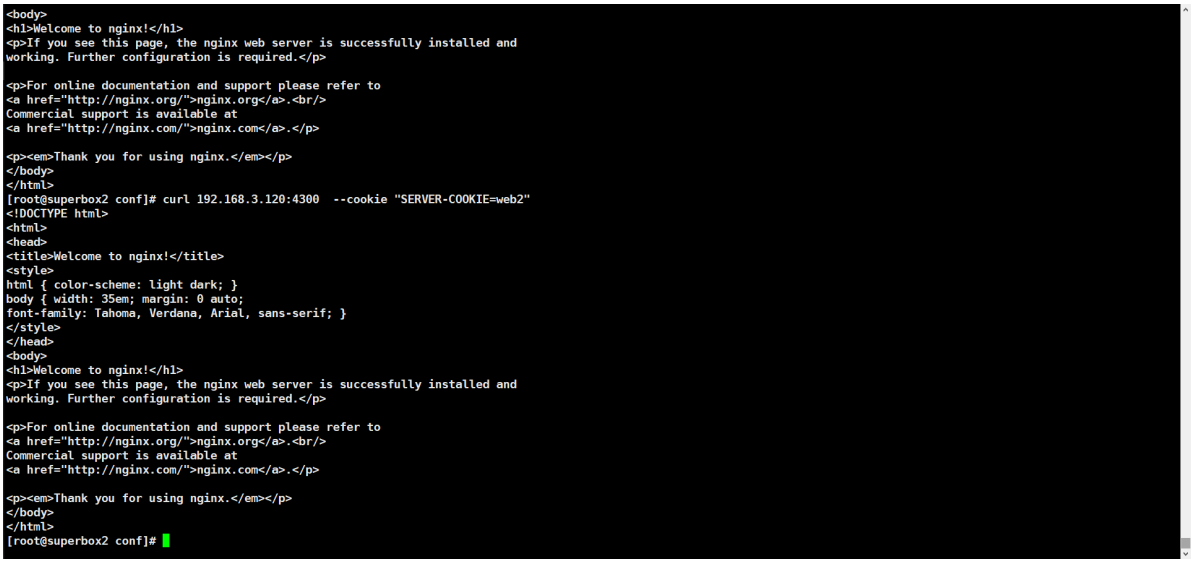
```
#+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ COOKIE配置实例 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
# cookie：为指定的后端服务器设定cookie值，此外指定的值将在请求入站时被检查，第一次为此值挑选的后端服务器将在后续的请求中一直被选中，其目的在于实现持久连接的功能。
listen web_cookie
  bind 192.168.3.120:4300
  mode http
  log  global
  balance roundrobin
  cookie SERVER-COOKIE insert indirect nocache
  server   web1   192.168.3.125:80      cookie web1 check inter 3000 fall 2 rise 5
  server   web2   192.168.3.125:8080    cookie web2 check inter 3000 fall 2 rise 5
```

验证COOKIE信息：



通过命令验证：

```
curl 192.168.3.120  --cookie "SERVER-COOKIE=web1"
curl 192.168.3.120  --cookie "SERVER-COOKIE=web2"
```



```
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@superbox2 conf]# curl 192.168.3.120:4300  --cookie "SERVER-COOKIE=web2"
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@superbox2 conf]#
```

补充知识：

server: 用来定义多台后端真实服务器,不能用于defaults和frontend部分,格式为:server name address:port param*

name：为后端真实服务器指定一个内部名称，随便这下义一个即可。

address：后端真实服务器的iP地址或主机名。

port：指定连接请求发往真实服务器时的目标端口，在未设定时，将使用客户端请求时的同一端口。

param*：为后端服务器设定的一系列参数，可用参数非常多。

check：表示启用对此后端服务器执行健康检查。

inter：设置健康状态检查的时间间隔，单位为毫秒。

rise：设置人故障状态转换至正常状态需要成功检查的次数，如 rise 2：表示2次检查正确就认为此服务器可用。

fall：设置后端服务器从正常状态转换为不可用状态需要检查的次数，如 fall 3表示3 次检查失败就认为此服务器不可用。

cookie：为指定的后端服务器设定cookie值，此外指定的值将在请求入站时被检查，第一次为此值挑选的后端服务器将在后续的请求中一直被选中，其目的在于实现持久连接的功能。

cookie server1：表示web1的serverid为server1。

weigth：设置后端真实服务器的权重，默认为1，最大值为256，设置为0表示不参与负载均衡。

maxconn：设定每个backend中server进程可接受的最大并发连接数，此选项等同于linux命令选项”ulimit -n”。

backup：设置后端真实服务器的备份服器，仅仅在后端所有真实服务器均不可用的情况下才启用。

十、haproxy报文修改

在http模式下，基于实际需求修改客户端的请求报文与响应报文，通过reqadd和reqdel在请求报文添加删除字段，

通过rspadd与rspidel在响应报文中添加与删除字段。

在请求报文尾部添加指定首部：

```
reqadd <string> [{if | unless} <cond>]
```

从请求报文中删除匹配正则表达式的首部：

```
reqdel <search> [{if | unless} <cond>]
reqidel <search> [{if | unless} <cond>]
```

在响应报文尾部添加指定首部：

```
rspadd <string> [{if | unless} <cond>]
```

示例：

rspadd X-Via:\ HAPorxy 从响应报文中删除匹配正则表达式的首部：

```
rspdel <search> [{if | unless} <cond>]
rspidel <search> [{if | unless} <cond>]
```

示例：

```
 rspidel server.* #从响应报文删除server信息
 rspidel X-Powered-By:.* #从响应报文删除X-Powered-By信息
```

十一、haproxy 的压缩、健康检查

1.haproxy压缩

haproxy还支持http协议的压缩机制，常用的算法有gzip和deflate。压缩的功能仅在7层有效，用户在请求时会先将浏览器支持的压缩算法发送给服务器，然后使用相同的压缩算法将数据返回给用户，以节省网络带宽，但是会占用部分CPU性能。

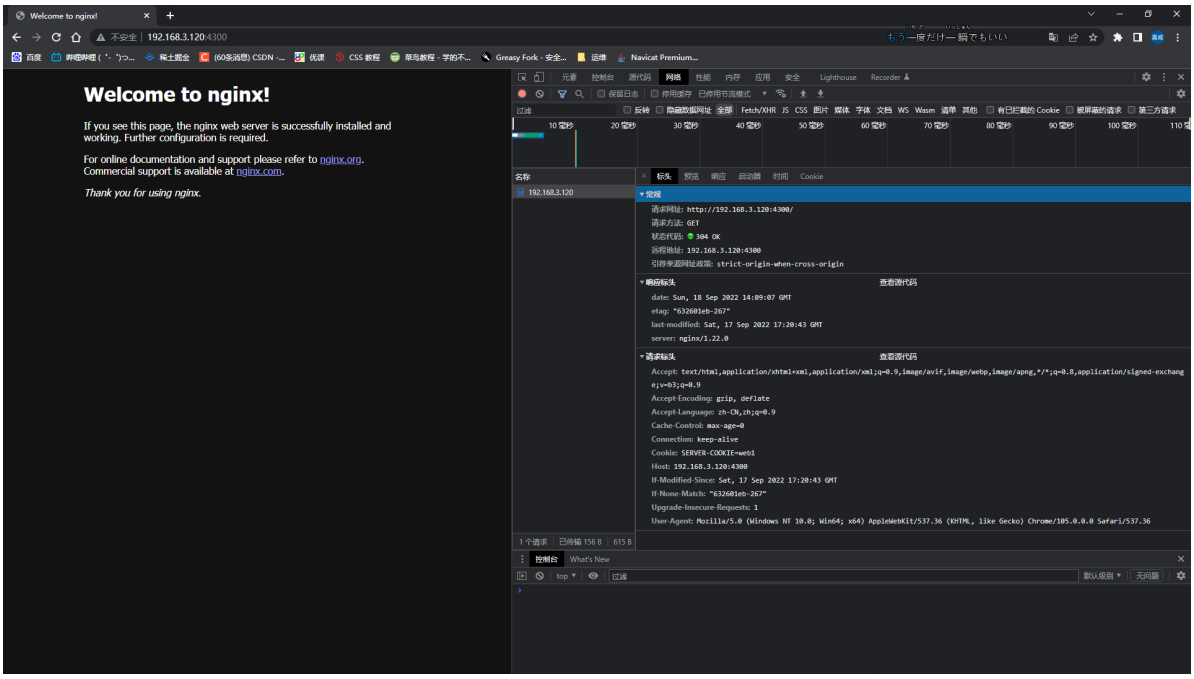
配置选项：

```
compression type #要压缩的文件类型
compression algo    #启用http协议中的压缩机制，常用算法有gzip deflate
dentity #调试使用的压缩方式
gzip #常用的压缩方式，与各浏览器兼容较好
deflate #有些浏览器不支持
raw-deflate #新出的压缩方式
```

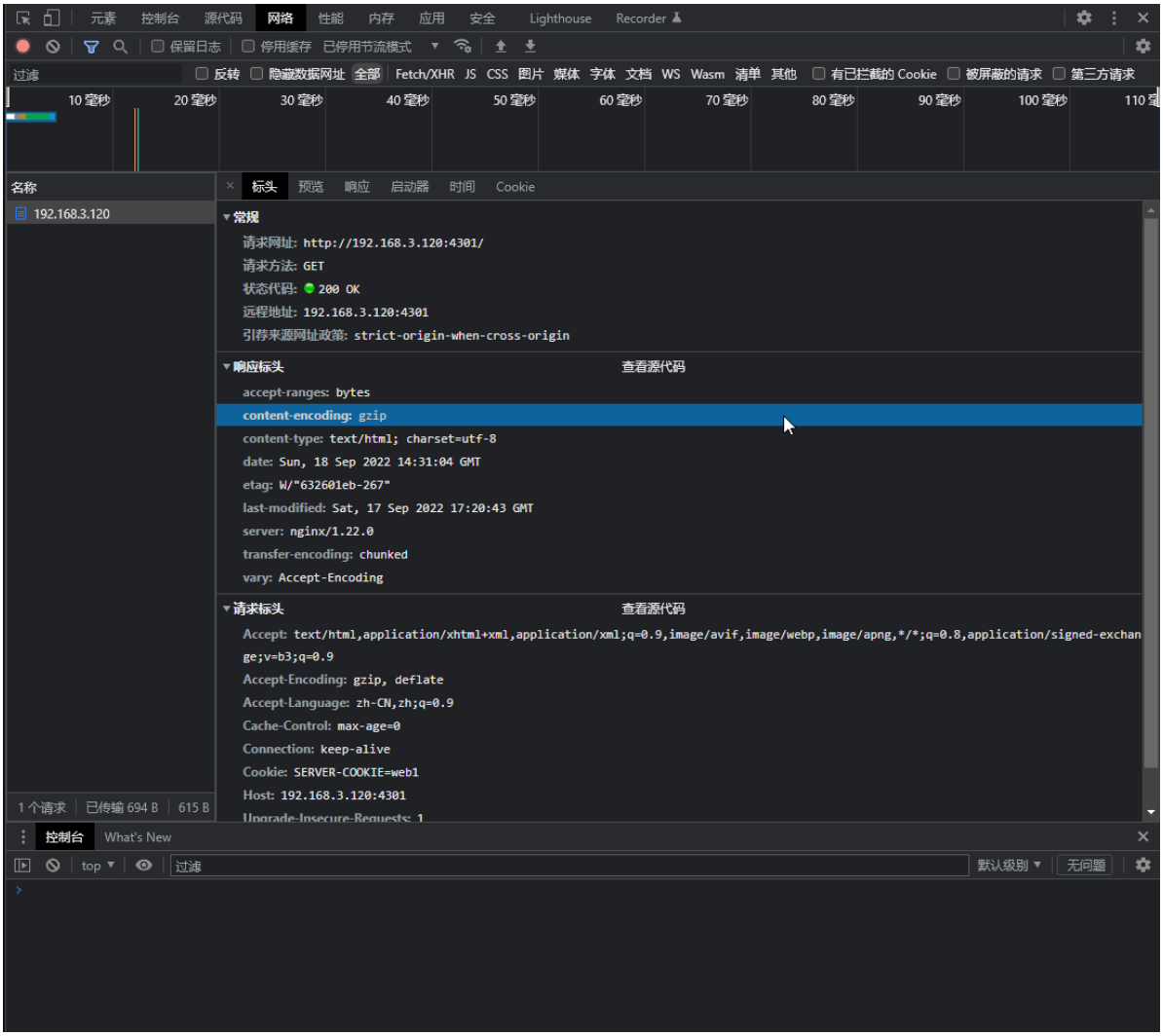
配置示例：

```

```

未设置压缩：



设置压缩后：



2.haproxy健康检查

1. TCP层健康检查

Haproxy的默认的tcp层健康检查机制是利用TCP的三次握手。

1、首先由Haproxy向代理的服务器发起SYN握手协商，默认是与代理的端口建立链接，比如说8080。
2、等待代理服务器确认第一次SYN，并响应ACK，与发起SYN的第二次握手。
3、Haproxy收到确认ACK之后，会向代理服务器发送TCP链接重置的报文，已经确认代理的服务器健康。

配置示例：

```
listen check_tcp
  bind 192.168.3.120:4308
  mode tcp
  log  global
  balance roundrobin
  server web1 192.168.3.125:3306 check
  server web2 192.168.3.125:6379 check
  stick-table type ip size 200k expire 30m
  stick on src

//check代表开启了tcp健康检查
```

2.HTTP层健康检查

1）通过监听端口进行健康检测 。这种检测方式，haproxy只会去检查后端server的端口，并不能保证服务的真正可用。

```
listen http_check_prot
  bind 192.168.3.120:4309
  mode http
  log  global
  balance roundrobin
  cookie SERVER-COOKIE insert indirect nocache
  option  httpchk
  server  web1  192.168.3.125:80      cookie web1 check
  server  web2  192.168.3.125:8080    cookie web2 check
inter 3000 fall 2 rise 5
```

2）通过URI获取进行健康检测 。检测方式，是用过去GET后端server的的web页面，基本上可以代表后端服务的可用性。

```
listen http_check_prot
  bind 192.168.3.120:4309
  mode http
  log  global
  balance roundrobin
  cookie SERVER-COOKIE insert indirect nocache
  option httpchk GET /index.html
  server  web1  192.168.3.125:80      cookie web1 check
  server  web2  192.168.3.125:8080    cookie web2 check
inter 3000 fall 2 rise 5
```

3）通过request获取的头部信息进行匹配进行健康检测 。这种检测方式，则是基于高级，精细的一些监测需求。通过对后端服务访问的头部信息进行匹配检测。

```
listen http_check_prot
  bind 192.168.3.120:4309
  mode http
  log  global
  balance roundrobin
  cookie SERVER-COOKIE insert indirect nocache
  option httpchk HEAD /index.jsp HTTP/1.1\r\nHost:\
www.xxx.com
  server  web1  192.168.3.125:80      cookie web1 check
  server  web2  192.168.3.125:8080    cookie web2 check
inter 3000 fall 2 rise 5
```

十二、ACL

访问控制列表（ACL，Access Control Lists）是一种基于包过滤的访问控制技术，它可以根据设定的条件对经过服
务器传输的数据包进行过滤(条件匹配)，即对接收到的报文进行匹配和过滤，基于请求报文头部中的源地址、源端
口、目标地址、目标端口、请求方法、URL、文件后缀等信息内容进行匹配并执行进一步操作，比如允许其通过或
丢弃。

ACL配置选项：

```
acl      <aclname>    <criterion>      [flags]
[operator]     [<value>]
acl          名称          匹配规范          匹配模式       具体操作符
    操作对象类型
```

ACL-Name：

```
acl    image_service      hdr_dom(host)    -i
img.magedu.com
ACL名称，可以使用大字母A-Z、小写字母a-z、数字0-9、冒号：、点.、中
横线和下划线，并且严格区分大小写，必
须Image_site和image_site完全是两个acl。
```

ACL-criterion（定义ACL匹配规范）：

```
hdr（[<name> [，<occ>]]）：完全匹配字符串,header的指定信息
hdr_beg（[<name> [，<occ>]]）：前缀匹配，header中指定匹配内
容的begin
hdr_end（[<name> [，<occ>]]）：后缀匹配，header中指定匹配内
容end
hdr_dom（[<name> [，<occ>]]）：域匹配，header中的domain
name

hdr_dir（[<name> [，<occ>]]）：路径匹配，header的uri路径
hdr_len（[<name> [，<occ>]]）：长度匹配，header的长度匹配
hdr_reg（[<name> [，<occ>]]）：正则表达式匹配，自定义表达式
(regex)模糊匹配
hdr_sub（[<name> [，<occ>]]）：子串匹配，header中的uri模糊匹
配

dst 目标IP
dst_port 目标PORT

src 源IP
src_port 源PORT

```

```
#示例：
 hdr(<string>) 用于测试请求头部首部指定内容
 hdr_dom(host) 请求的host名称，如 www.magedu.com
 hdr_beg(host) 请求的host开头，如 www. img. video. download. ftp.
 hdr_end(host) 请求的host结尾，如 .com .net .cn
 path_beg 请求的URL开头，如/static、/images、/img、/css
 path_end  请求的URL中资源的结尾，如 .gif .png .css .js .jpg .jpeg
有些功能是类似的，比如以下几个都是匹配用户请求报文中host的开头是不是www：
 acl short_form hdr_beg(host) www.
 acl alternate1 hdr_beg(host) -m beg www.
 acl alternate2 hdr_dom(host) -m beg www.
 acl alternate3 hdr(host) -m beg www.
```

ACL-flags（ACL匹配模式）：

```
    -i 不区分大小写
    -m 使用指定的pattern匹配方法
    -n 不做DNS解析
    -u 禁止acl重名，否则多个同名ACL匹配或关系
```

ACL-operator（ACL 操作符）：

```
整数比较：eq、ge、gt、le、lt
字符比较：
- exact match (-m str) :字符串必须完全匹配模式
- substring match (-m sub) :在提取的字符串中查找模式，如果其中任何一个被发现，ACL将匹配
- prefix match (-m beg) :在提取的字符串首部中查找模式，如果其中任何一个被发现，ACL将匹配
- suffix match (-m end) :将模式与提取字符串的尾部进行比较，如果其中任何一个匹配，则ACL进行匹配
- subdir match (-m dir) :查看提取出来的用斜线分隔（“/”）的字符串，如果其中任何一个匹配，则ACL进行匹配
- domain match (-m dom) :查找提取的用点（“.”）分隔字符串，如果其中任何一个匹配，则ACL进行匹配
```

ACL-value（value的类型）：

The ACL engine can match these types against patterns of the following types :

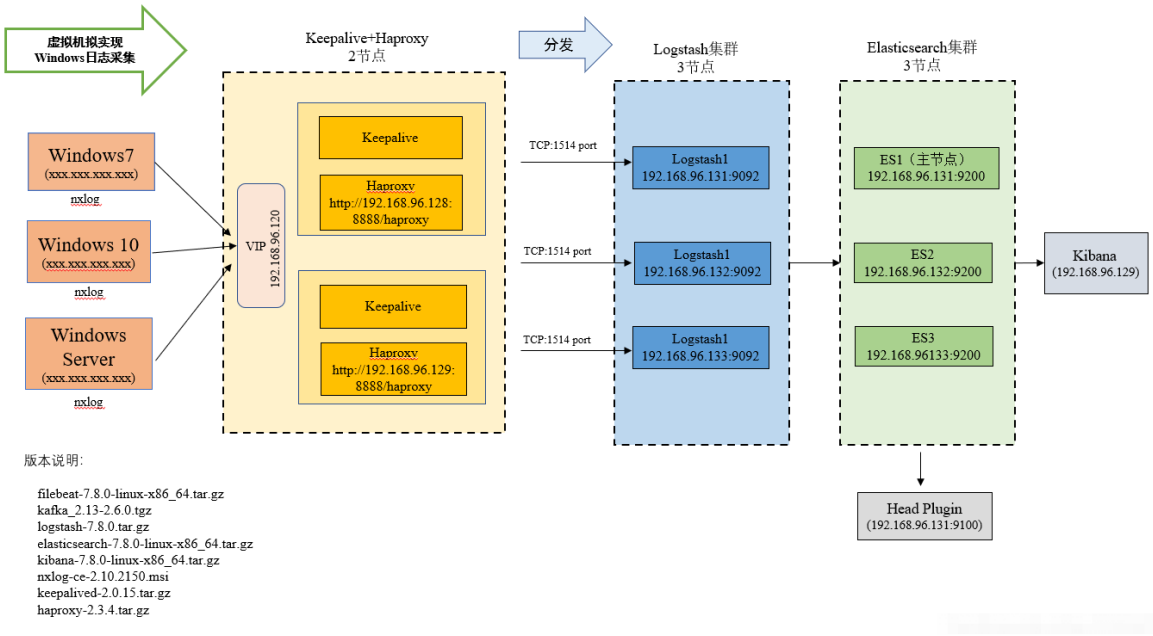
```
- Boolean #布尔值
- integer or integer range #整数或整数范围，比如用于匹配端口范围
- IP address / network #IP地址或IP范围, 192.168.0.1,192.168.0.1/24
- string--> www.magedu.com
 exact –精确比较
 substring–子串
 suffix-后缀比较
 prefix-前缀比较
 subdir-路径， /wp-includes/js/jquery/jquery.js
 domain-域名，www.magedu.com
- regular expression #正则表达式
- hex block #16进制
```

ACL调用方式：

```
- 与：隐式（默认）使用
- 或：使用“or” 或 “||”表示
- 否定：使用“!“ 表示

示例：
 if valid_src valid_port #与关系,A和B都要满足为true
 if invalid_src || invalid_port #或，A或者B满足一个为true
 if ! invalid_src #非，取反，A和B哪个也不满足为true
```

十三、haproxy+Keepalived实现高可用

在单个haproxy负载集群时，如果haproxy挂掉了，那么整个服务应用也不能再继续运行了针对这个问题，引入keepalived对haproxy进行控制管理。



1.配置情况

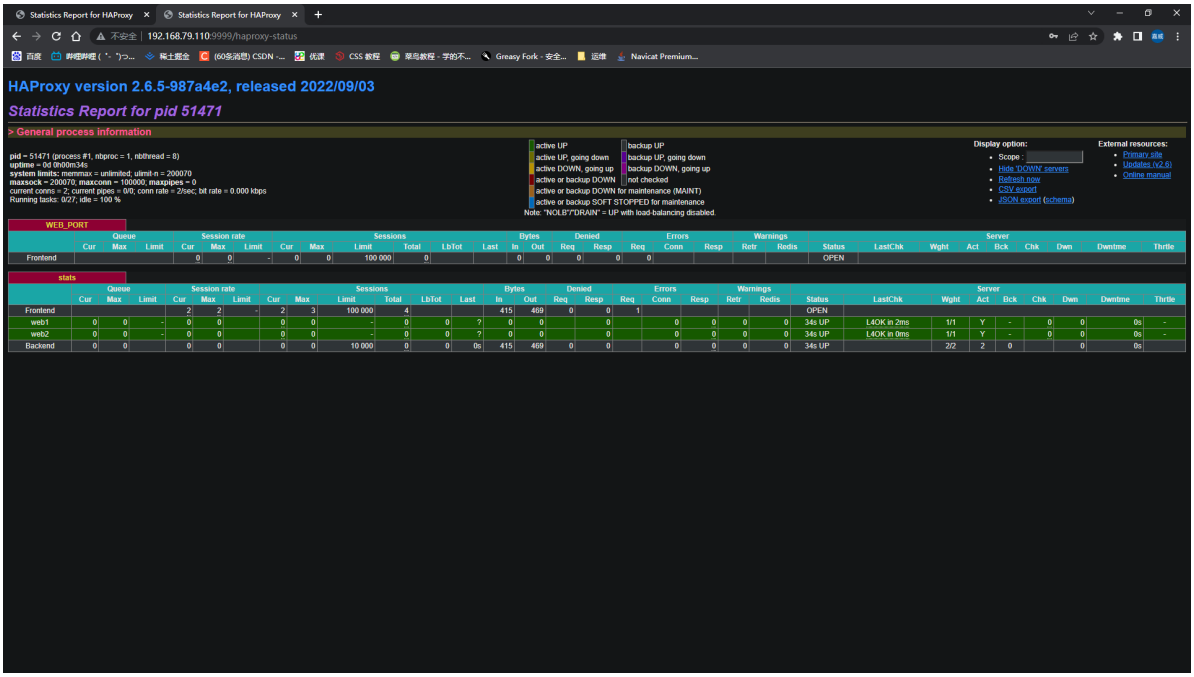
服务器属性	服务器IP	服务器说明
H1搭载 haproxy+keepalived	192.168.79.110	负载服务器+高可用
H2搭载 haproxy+keepalived	192.168.79.120	负载服务器+高可用
VIP	192.168.79.130	

H1、H2服务器安装haproxy，具体安装流程见《一、安装haproxy》，以下省略安装步骤。

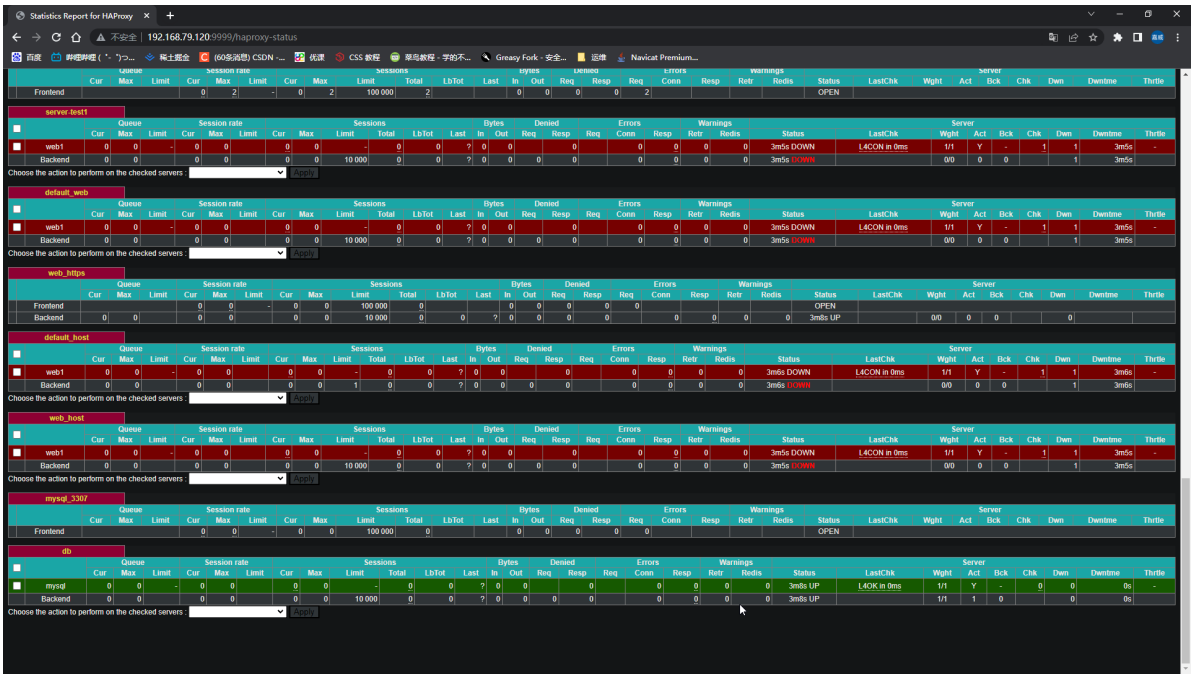
2.网页输入http://localhost:9999/haproxy-status进入监控界面

```
# 初始账号密码为
账号：admin
密码：admin~!@
```

确认H1状态正常：



确认H2状态正常：



2.安装keepalived

3.配置master节点和backup节点。

Master 中的配置大致和 BACKUP 中的相同，不过需要修改 global_defs{} 的 router_id；其次要修改 vrrp_instance VI_1{} 中的state 为 MASTER；修改 mcast_src_ip 为本机 ip, 最后要将 priority 设置为大于 50 的值。注 意Master 和 Backup 中的 virtual_router_id 要保持一致。

下面的是 MASTER节点 的配置：

```
# MASTER节点配置为：
! Configuration File for keepalived

global_defs {
# # 路由ID, 主备的ID不能相同
   router_id KeepalivedA
   script_user root
   enable_script_security
}

vrrp_instance VI_1 {
# Keepalived 的角色。Master 表示主服务器，从服务器设置为 BACKUP
    state MASTER
# 指定监测网卡,通过 ip addr 可查看
    interface ens33
# 主备机应该保持一致
    virtual_router_id 50
# 优先级，MASTER 机器上的优先级要大于这个值
    priority 150
# 设置主备之间的检查时间，单位为 s
    advert_int 1
# 本机 IP
    mcast_src_ip 192.168.79.110
# 定义验证类型和密码
    authentication {
        auth_type PASS
        auth_pass 1234
    }
# VIP地址，可以设置多个
    virtual_ipaddress {
        192.168.79.130
    }
```

```
}
```

下面的是 BACKUP 的配置：

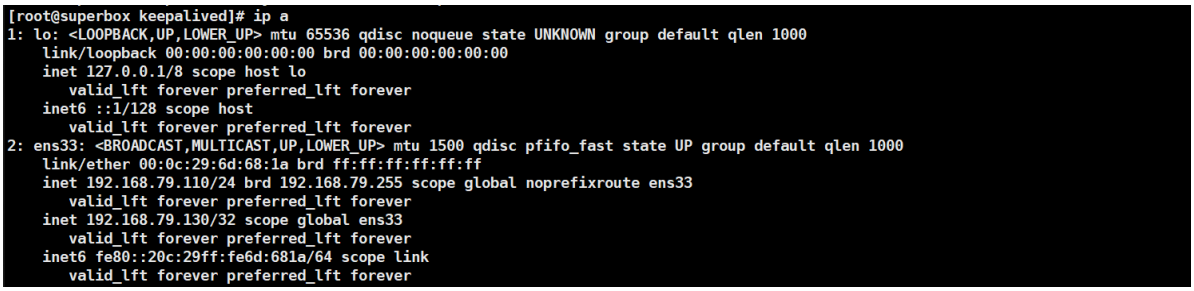
```
! Configuration File for keepalived

global_defs {
# # 路由ID, 主备的ID不能相同
   router_id KeepalivedB
   script_user root
   enable_script_security
}

vrrp_instance VI_1 {
# Keepalived 的角色。Master 表示主服务器，从服务器设置为BACKUP
    state BACKUP
# 指定监测网卡,通过 ip addr 可查看
    interface ens33
# 主备机应该保持一致
    virtual_router_id 50
# 优先级，MASTER 机器上的优先级要大于这个值
    priority 50
# 设置主备之间的检查时间，单位为 s
    advert_int 1
# 本机 IP
    mcast_src_ip 192.168.79.120
# 定义验证类型和密码
    authentication {
        auth_type PASS
        auth_pass 1234
    }
# VIP地址，可以设置多个
    virtual_ipaddress {
        192.168.79.130
    }
}
```

4.启动keepalived

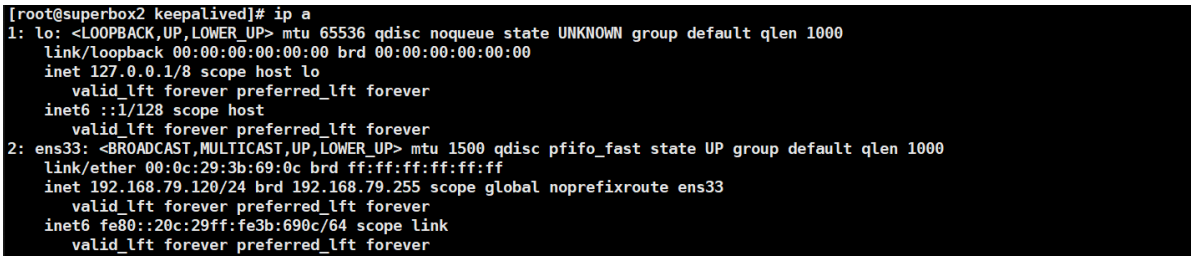
```
# H1
systemctl restart keepalived
# H2
systemctl restart keepalived
```

5.查看keepalived状态

Master节点网卡信息：



BACKUP节点网卡信息：



6.测试keepalived

按原理，VIP 192.168.79.130 默认会绑定在 MASTER 机 192.168.79.110 的网卡上，当关掉其中 MASTER 的 keepalived 服务时，这个虚拟 ip 立马绑定另一台机的网卡上，重新开启 MASTER 的 Keepalived 服务的时候，MASTER 又夺回领导权。

所以尝试停掉master节点的keepalived服务：

```
systemctl stop keepalived
```

查看master网卡信息：

```
[root@superbox keepalived]# systemctl stop keepalived
[root@superbox keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:6d:68:1a brd ff:ff:ff:ff:ff:ff
    inet 192.168.79.110/24 brd 192.168.79.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6d:681a/64 scope link
       valid_lft forever preferred_lft forever
```

可以看到VIP节点信息消失了，再查看backup网卡信息：

```
[root@superbox2 keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:3b:69:0c brd ff:ff:ff:ff:ff:ff
    inet 192.168.79.120/24 brd 192.168.79.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet 192.168.79.130/32 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe3b:690c/64 scope link
       valid_lft forever preferred_lft forever
```

VIP网卡飘到了backup上，接下来重启master节点keepalived服务：

```
[root@superbox keepalived]# systemctl start keepalived
[root@superbox keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:6d:68:1a brd ff:ff:ff:ff:ff:ff
    inet 192.168.79.110/24 brd 192.168.79.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet 192.168.79.130/32 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6d:681a/64 scope link
       valid_lft forever preferred_lft forever
3: ens35: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:6d:68:24 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.100/24 brd 172.16.0.255 scope global noprefixroute ens35
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6d:6824/64 scope link
       valid_lft forever preferred_lft forever
```

VIP网卡又回到了master节点上，而backup节点的VIP网卡消失了：

```
[root@superbox2 keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:3b:69:0c brd ff:ff:ff:ff:ff:ff
    inet 192.168.79.120/24 brd 192.168.79.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe3b:690c/64 scope link
       valid_lft forever preferred_lft forever
```

7.Haproxy 节点检测

为了防止 HAProxy 服务宕掉了，Keepalived 却还在正常工作而没有切换到 BACKUP 上的情况，这里编写一个脚本来检测 Haproxy 服务状态来处理这种情况。当 Haproxy 挂掉之后，脚本尝试重启 Haproxy 服务，如果重启不成功则关闭 Keepalived 服务，Keepalived 切换到 Backup 继续工

作。

编写 haproxy 检测脚本，并加入 Keepalived 中

```
# 脚本路径自行选择
cat > /etc/keepalived/check_haproxy.sh << eof
#!/bin/bash
if [ 1 -eq 0 ];then
    /usr/local/haproxy/haproxy -f
/usr/local/haproxy/conf/haproxy.cfg
fi
sleep 2
if [ 1 -eq 0 ];then
    systemctl stop keepalived.service
fi
eof
##################################### 修改keepalived配置
文件 ##########################################
加入脚本定义部分和脚本执行部分
# 自定义监控脚本
vrrp_script chck_haproxy {
        script "/etc/keepalived/check_haproxy.sh"
        interval 5
# 每两秒检查执行一次
        weight 2
}
track_script {
   chck_haproxy
    }
##################################### 修改后的keepalived
文件 ##########################################
! Configuration File for keepalived

global_defs {
# # 路由ID, 主备的ID不能相同
   router_id KeepalivedB
   script_user root
   enable_script_security
}
# 自定义监控脚本
```

```
vrrp_script chck_haproxy {
        script "/etc/keepalived/check_haproxy.sh"
        interval 5
# 每两秒检查执行一次
        weight 2
}
vrrp_instance VI_1 {
# Keepalived 的角色。Master 表示主服务器，从服务器设置为
BACKUP
    state BACKUP
# 指定监测网卡,通过 ip addr 可查看
    interface ens33
# 主备机应该保持一致
    virtual_router_id 50
# 优先级，MASTER 机器上的优先级要大于这个值
    priority 50
# 设置主备之间的检查时间，单位为 s
    advert_int 1
# 本机 IP
    mcast_src_ip 192.168.79.120
# 定义验证类型和密码
    authentication {
        auth_type PASS
        auth_pass 1234
    }
track_script {
   chck_haproxy
    }
# VIP地址，可以设置多个
    virtual_ipaddress {
        192.168.79.130
    }
}

# 对脚本赋予可执行权限
chmod 755 /etc/keepalived/check_haproxy.sh
# 重启 Keepalived 服务。
systemctl restart keepalived.service
```

测试脚本：

kill 掉 Haproxy 进程后，检查Haproxy 是否会被重新启动，以及如果 Haproxy 如果无法被调起，Keepalived 是否被正常关闭并切换到 Backup 备机。

```
[root@superbox keepalived]# ps -ef | grep haproxy
root      51469      1  0 20:19 ?        00:00:00 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /var/lib/haproxy/haproxy.pid
Haproxy   51471  51469  0 20:19 ?        00:00:05 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /var/lib/haproxy/haproxy.pid
root      59511  59510  0 22:13 ?        00:00:00 /bin/bash /etc/keepalived/check_haproxy.sh
root      59515   2016  0 22:13 pts/0    00:00:00 grep --color=auto haproxy
```

```
1 kill -9 51471
```

```
[root@superbox keepalived]# kill -9 51471
[root@superbox keepalived]# ps -ef | grep haproxy
root      59616   2016  0 22:14 pts/0    00:00:00 grep --color=auto haproxy
[root@superbox keepalived]#
```

等待5秒再查看haproxy是否重启：

```
[root@superbox keepalived]# kill -9 51471
[root@superbox keepalived]# ps -ef | grep haproxy
root      59616   2016  0 22:14 pts/0    00:00:00 grep --color=auto haproxy
[root@superbox keepalived]# ps -ef | grep haproxy
root      59675  59674  0 22:15 ?        00:00:00 /bin/bash /etc/keepalived/check_haproxy.sh
root      59678   2016  0 22:15 pts/0    00:00:00 grep --color=auto haproxy
[root@superbox keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:6d:68:1a brd ff:ff:ff:ff:ff:ff
    inet 192.168.79.110/24 brd 192.168.79.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet 192.168.79.130/32 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6d:681a/64 scope link
       valid_lft forever preferred_lft forever
3: ens35: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:6d:68:24 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.100/24 brd 172.16.0.255 scope global noprefixroute ens35
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6d:6824/64 scope link
       valid_lft forever preferred_lft forever
```

可以看到haproxy自动重启了，VIP也还在H1上面。

十四、keepalived+haproxy 高可用负载均衡器实现对 Logstash 集群的负载