一. haproxy 配置文件详解

haproxy 配置分为五部分，分别如下:

1 global: (全局配置主要用于设定义全局参数，属于进程级的配置，通常和操作系统配置有关)

2 default : (配置默认参数，这些参数可以被用到 frontend，backend，Listen 组件)

在此部分中设置的参数值，默认会自动引用到下面的 frontend、backend、listen 部分中，因引，某些参数属于公用的配置，只需要在 defaults 部分添加一次即可。而如果 frontend、backend、listen 部分也配置了与 defaults 部分一样的参数，Defaults 部分参数对应的值自动被覆盖。

3 frontend： (接收请求的前端虚拟节点，Frontend 可以更加规则直接指定具体使用后端的 backend)

frontend 是在 haproxy 1.3 版本以后才引入的一个组件，同时引入的还有 backend 组件。通过引入这些组件，在很大程度上简化了 haproxy 配置文件的复杂性。forntend 可以根据 ACL 规则直接指定要使用的后端 backend。

4 backend : (后端服务集群的配置，真实服务器，一个 Backend 对应一个或者多个实体服务器)

在 HAProxy1.3 版本之前，HAProxy 的所有配置选项都在这个部分中设置。为了保持兼容性，haproxy 新的版本依然保留了 listen 组件配置式。两种配置方式任选一中。

5 Listen : (Fronted 和 backend 的组合体) 比如 haproxy 实例状态监控部分配置

代理配置段:

Backend:后端服务器组的定义

Frontend:定义面向客户的监听的地址和端口,以及关联的后端的服务器组

Listen:组合的方式直接定义 frontend 及相关的 backend

Defaults:默认的配置

关于 haproxy 时间格式配置说明

一些包含了值的参数表示时间，如超时时长。这些值一般以毫秒为单位，但也可以使用其它的时间单位后缀。

us: 微秒(microseconds)，即 1/1000000 秒；

ms: 毫秒(milliseconds)，即 1/1000 秒；

s: 秒(seconds)；

m: 分钟(minutes)；

h：小时(hours)；

d: 天(days)；

1.1 haproxy global 部分配置说明

通常主要定义全局配置主要用于设定义全局参数，属于进程级的配置，通常和操作系统配置有关

global

log 127.0.0.1 local3　　#定义 haproxy 日志输出设置,全局的日志配置，local0 是日志输出设置，info 表示日志级别（err，waning，notice，info，debug）；

log 127.0.0.1 local3 notice #定义 haproxy 日志级别

#ulimit -n 82000　　#设置每个进程的可用的最大文件描述符 设定每个进程所能够打开的最大文件描述符数目，默认情况下其会自动进行计算，因此不推荐修改此选项。

```
maxconn 20480              #默认最大连接数 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 命令选项”ulimit -n”
chroot /usr/local/haproxy  #chroot 运行路径 修改 haproxy 的工作目录至指定的目录并在放弃权限之前执行 chroot()操作,可以提升 haproxy 的安全级别，不过需要注意的是要确保指定的目录为空目录且任何用户均不能有写权限；
user haproxy               #运行 haproxy 的用户
group haproxy              #运行 haproxy 的用户组
daemon                     #以后台形式运行 harpoxy 让 haproxy 以守护进程的方式工作于后台，其等同于“-D”选项的功能，当然，也可以在命令行中以“-db”选项将其禁用；
nbproc 1                   #设置进程数量 指定启动的 haproxy 进程个数，只能用于守护进程模式的 haproxy；默认只启动一个进程，鉴于调试困难等多方面的原因，一般只在单进程仅能打开少数文件描述符的场景中才使用多进程模式；
pidfile /usr/local/haproxy/run/haproxy.pid  #haproxy 进程 PID 文件
#debug                     #haproxy 调试级别，建议只在开启单进程的时候调试
#quiet                     #减少常规的日志信息输出
stats socket /var/lib/haproxy/stats  #义统计信息保存位置。
```

如要设置 haproxy 的日志内容，可参考以下配置

```
capture request header Host len 40
capture request header Content-Length len 10
capture request header Referer len 200
capture response header Server len 40
capture response header Content-Length len 10
capture response header Cache-Control len 8
```

1.2 haproxy defaults 部分配置说明

用于设置配置默认参数，这些参数可以被用到 frontend，backend，listen 组件；
此部分中设置的参数值，默认会自动引用到下面的 frontend、backend、listen 部分中，因引，某些参数属于公用的配置，只需要在 defaults 部分添加一次即可。而如果 frontend、backend、listen 部分也配置了与 defaults 部分一样的参 数，Defaults 部分参数对应的值自动被覆盖

```
defaults
log   global       #引入 global 定义的日志格式
mode   http        #所处理的类别(7 层代理 http，4 层代理 tcp)
maxconn 50000      #最大连接数
option  httplog    #日志类别为 http 日志格式
option  httpclose  #每次请求完毕后主动关闭 http 通道
option  originalto  #HAProxy 上配置此选项，这样 HAProxy 会把自身 ip 地址添加到"X-Original-To"字段发送给后端服务器。
option  dontlognull  #不记录健康检查日志信息
option  forwardfor  #如果后端服务器需要获得客户端的真实 ip，需要配置的参数， 可以从 http header 中获取客户端的 IP
option  nolinger    #强制 HAProxy 在关闭连接时不执行完整的四路挥手过程，而是会立即发送 RST（复位）分节来终止连接，默认 linger。
retries 3          #单个请求失败后重试的最大次数
```

```
option redispatch    #serverID 对应的服务器挂掉后，强制定向到其他健康的服务器, 当使用了 cookie 时，haproxy 将会将其请求的后端服务器的 serverID 插入到 cookie 中，以保证会话的 SESSION 持久性；而此时，如果后端的服务器宕掉了,但是客户端的 cookie 是不会刷新的，如果设置此参数，将会将客户的请求强制定向到另外一个后端 server 上，以保证服务的正常---》
stats refresh 30      #设置统计页面刷新时间间隔
option abortonclose    #当服务器负载很高的时候，自动结束掉当前队列处理比较久的连接
balance roundrobin     #设置默认负载均衡方式，轮询方式
#contimeout 5000       #设置连接超时时间
#clitimeout 50000      #设置客户端超时时间
#srvtimeout 50000      #设置服务器超时时间
timeout http-request   10s  #默认 http 请求超时时间
timeout queue          1m   #默认队列超时时间
timeout connect        10s  #haproxy 和后端服务器建立连接的默认超时时间
timeout client         1m   #haproxy 和客户端的默认超时时间
timeout server         1m   #后端服务器处理请求的默认超时时间
timeout http-keep-alive 10s  #默认持久连接超时时间
timeout check          10s  #设置心跳检查超时时间
```

mode http

设置 haproxy 的运行模式，有三种｛http|tcp|health｝。

tcp 模式：在此模式下，客户端和服务器端之前将建立一个全双工的连接，不会对七层报文做任何检查，默认为 tcp 模式，经常用于 SSL、SSH、SMTP 等应用。

http 模式：在此模式下，客户端请求在转发至后端服务器之前将会被深度分板，所有不与 RFC 格式兼容的请求都会被拒绝。

health：已基本不用了。

log global

设置日志继承全局配置段的设置。

option httplog

表示开始打开记录 http 请求的日志功能。

option dontlognull

如果产生了一个空连接，那这个空连接的日志将不会记录，主要指的是健康检查类日志信息，这类日志只有简单的回应而没有实际内容，可以提高系统性能。

option http-server-close

打开 http 协议中服务器端关闭功能，使得支持长连接，使得会话可以被重用，使得每一个日志记录都会被记录。

option forwardfor except 127.0.0.0/8

如果上游服务器上的应用程序想记录客户端的真实 IP 地址，haproxy 会把客户端的 IP 信息发送给上游服务器，在 HTTP 请求中添加”X-Forwarded-For”字段,但当是 haproxy 自身的健康检测机制去访问上游服务器时是不应该把这样的访问日志记录到日志中的，所以用 except 来排除 127.0.0.0，即 haproxy 身。 如果有多层 haproxy，则 option forwardfor if-none 可以只在 header 不存在 X-Forwarded-For 的时候去设置这个值

option redispatch

当与上游服务器的会话失败(服务器故障或其他原因)时，把会话重新分发到其他健康的服务器上,当原来故障的服务器恢复时，会话又被定向到已恢复的服务器上。还可以

用”retries”关键字来设定在判定会话失败时的尝试连接的次数。

retries 3

向上游服务器尝试连接的最大次数，超过此值就认为后端服务器不可用。

option abortonclose

当 haproxy 负载很高时，自动结束掉当前队列处理比较久的链接。

timout http-request 10s

客户端发送 http 请求的超时时间。

timeout queue 1m

当上游服务器在高负载响应 haproxy 时，会把 haproxy 发送来的请求放进一个队列中，timeout queue 定义放入这个队列的超时时间。

timeout connect 5s

haproxy 与后端服务器连接超时时间，如果在同一个局域网可设置较小的时间。

timeout client 1m

定义客户端与 haproxy 连接后，数据传输完毕，不再有数据传输，即非活动连接的超时时间。

timeout server 1m

定义 haproxy 与上游服务器非活动连接的超时时间。

timeout http-keep-alive 10s

haproxy 处理完一个请求，在同一个连接上等待下一个请求到来的最大时间，时间较短时可以尽快释放出资源，节约资源。

timeout check 10s

健康检测的时间的最大超时时间。

maxconn 3000

最大并发连接数。

contimeout 5000

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，新版本的 haproxy 使用 timeout connect 替代，该参数向后兼容。

clitimeout 3000

设置连接客户端发送数据时的成功连接最长等待时间，默认单位是毫秒，新版本 haproxy 使用 timeout client 替代。该参数向后兼容。

srvtimeout 3000

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，新版本 haproxy 使用 timeout server 替代。该参数向后兼容。

balance roundrobin

设置负载算法为：轮询算法 rr

balance :用来定义负载均衡算法

1.roundrobin：基于权重进行的轮叫算法，在服务器的性能分布经较均匀时这是一种最公平的，最合量的算法。

2.static-rr：也是基于权重时行轮叫的算法，不过此算法为静态方法，在运行时调整其服务权重不会生效。

3.source：是基于请求源 IP 的算法，此算法对请求的源 IP 时行 hash 运算，然后将结果与后端服务器的权理总数相除后转发至某台匹配的后端服务器，这种方法可以使用一个客户端 IP 的请求始终转发到特定的后端服务器。

4.leastconn：此算法会将新的连接请求转发到具有最少连接数目的后端服务器。在会话时

间较长的场景中推荐使用此算法。例如数据库负载均衡等。此算法不适合会话较短的环境，如基于 http 的应用。

5.uri：此算法会对部分或整个 URI 进行 hash 运算，再经过与服务器的总权重要除，最后转发到某台匹配的后端服务器上。

6.uri_param：此算法会椐据 URL 路径中的参数时行转发，这样可以保证在后端真实服务器数量不变时，同一个用户的请求始终分发到同一台机器上。

7.hdr：此算法根据 httpd 头时行转发，如果指定的 httpd 头名称不存在，则使用 roundrobin 算法进行策略转发。

8.rdp-cookie(name)：示根据据 cookie(name)来锁定并哈希每一次 TCP 请求。

1.3 haproxy frontend 部分配置说明

frontend 是在 haproxy 1.3 版本以后才引入的一个组件，同时引入的还有 backend 组件。通过引入这些组件，在很大程度上简化了 haproxy 配置文件的复杂性。frontend 根据任意 HTTP 请求头内容做 ACL 规则匹配,然后把请求定向到相关的 backend

配置 DEMO：

```
frontend  main *:5000
   acl url_static       path_beg      -i /static /images /javascript /stylesheets
   acl url_static       path_end      -i .jpg .gif .png .css .js

   use_backend static          if url_static
   default_backend             app
```

配置参数详解：

```
bind 0.0.0.0:80         #设置监听端口，即 haproxy 提供的 web 服务端口，和 lvs 的 vip 类似
mode http               #http 的 7 层模式
log global              #应用全局的日志设置
option httplog          #启用 http 的 log
option httpclose        #每次请求完毕后主动关闭 http 通道，HAproxy 不支持 keep-alive 模式
option forwardfor       #如果后端服务器需要获得客户端的真实 IP 需要配置此参数，将可以从 HttpHeader 中获得客户端 IP
http-request add-header X-Client-Info "IP=%ci:Port=%cp" #在响应中添加客户端 IP 和端口的自定义头

acl url_static       path_beg       -i /static /images /javascript /stylesheets   #定义路径匹配规则
acl url_static       path_end       -i .jpg .gif .png .css .js                    #定义后缀匹配规则

use_backend static          if url_static                                #定义 acl 规则的对应的 backend
default_backend wwwpool   #设置请求默认转发的后端服务池，
```

frontend http_80_in

定义一个名为 http_80_in 的 frontend。

bind 0.0.0.0:80

定义 haproxy 前端部分监听的端口。

mode http

定义为 http 模式。

log global

继承 global 中 log 的定义。

option forwardfor

使后端 server 获取到客户端的真实 IP。

acl url_static　　path_beg　　-i /static /images /javascript /stylesheets

定义路径匹配规则

acl url_static　　path_end　　-i .jpg .gif .png .css .js

定义后缀匹配规则

acl 匹配方式列表：

path - 完全匹配 URL 路径
path_end - 匹配 URL 路径的结束部分
path_dir - 如果 URL 路径包含指定的目录名，则匹配成功
path_reg - 使用正则表达式匹配 URL 路径
path_sub - 匹配 URL 路径中包含的子字符串
hdr - 匹配 HTTP 请求头的值
hdr_beg - 匹配 HTTP 请求头的开始部分
hdr_end - 匹配 HTTP 请求头的结束部分
hdr_dom - 匹配请求头中的域名部分
url - 对整个 URL 进行匹配
url_beg - 匹配 URL 的开始部分
cookie - 根据 Cookie 的内容进行匹配
src - 匹配客户端的源 IP 地址
dst - 匹配目标服务器的 IP 地址
method - 匹配 HTTP 请求方法，如 GET、POST 等
ssl_fc - 检查是否为 SSL 连接。
query - 匹配 URL 中的查询字符串

use_backend static　　if url_static

定义 acl 规则的对应的 backend

ACL 用法详解：

haproxy ACL 具有很强大的功能，能够定义三到七层的规则。ACL 的作用，就是为了匹配一些特别的请求，然后对其进行修改或者分发到不同的服务器组中

1、haproxy acl 定义

格式： acl [flags] [operator] []

：ACL 名称，区分字符大小写，且其只能包含大小写字母、数字、-(连接线)、_(下划线)、.(点号)和:(冒号)；haproxy 中，acl 可以重名，这可以把多个测试条件定义为一个共同的 acl；

：测试标准，即对什么信息发起测试；测试方式可以由[flags]指定的标志进行调整；而有些测试标准也可以需要为其在之前指定一个操作符[operator]；

：[flags]，目前 haproxy 的 acl 支持的标志位有 3 个：

-i：不区分中模式字符的大小写；

-f：从指定的文件中加载模式；

--：标志符的强制结束标记，在模式中的字符串像标记符时使用；

：acl 测试条件支持的值有以下四类：

整数或整数范围：如 1024:65535 表示从 1024 至 65535；仅支持使用正整数(如果出现类似小数的标识，其为通常为版本测试)，且支持使用的操作符有 5 个，分别为 eq、ge、gt、le 和 lt；

字符串：支持使用“-i”以忽略字符大小写，支持使用“\”进行转义；如果在模式首部出现了-i，可以在其之前使用“–”标志位；

正则表达式：其机制类同字符串匹配；

IP 地址及网络地址；

注意:同一个 acl 中可以指定多个测试条件，这些测试条件需要由逻辑操作符指定其关系。条件间的组合测试关系有三种：“与”(默认即为与操作)、“或”(使用“||”操作符)以及“非”(使用“!”操作符)。

常用的测试标准(criteria)

(1) 基于七层协议（http）规则 acl 测试标准

method

method 测试 HTTP 请求报文中请求类型。

例如： 利用 method 来实现前段读写分离：

```
acl read method GET
acl read method HEAD
acl write method PUT
acl write method POST
use_backend imgservers if read
use_backend uploadservers if write
```

path_beg ||url_beg

用于测试请求的 URL 是否以指定的模式开头;下面的例子用于测试 URL 是否以 /static、/images、/javascript 或/stylesheets 头。

例如：利用 path_beg 实现以/static /images 开头的请求转交到 static server 上：

```
acl url_static path_beg -i /static /images
use_backend static if url_static
```

path_end || url_reg

用于测试请求的 URL 是否以指定的模式结尾。例如，下面的例子用户测试 URL 是否以 jpg、gif、png、css 或 js 结尾

例如：利用 path_end 实现以.jpg .gif .png .css 等结尾的格式的请求转交到 static server 上：

```
acl url_static path_end -i .jpg .gif .png .css .js .html
use_backend static if url_static
```

hdr_beg

用于测试请求报文的指定首部的开头部分是否符合指定的模式。

```
acl is_lvs3 hdr_beg(host) -i lvs3.test.net:8080
```

use_backend lvs3 if is_lvs3
#表示当 request header 中的 host 前缀部分匹配到 lvs.test.net.:8080 则将请求转给后端 backend 定义的 is_lvs3 上

hdr_end
用于测试请求报文的指定首部的结尾部分是否符合指定的模式
acl is_lvs3 hdr_end(host) -i lvs3.test.net:8080
use_backend lvs3 if is_lvs2
#表示当 request header 中的 host 后缀部分匹配到 lvs3.test.net.:8080 则将请求转给后端 backend 定义的 is_lvs2 上

hdr
用于测试请求报文中的所有首部或指定首部信息是否满足指定的条件；指定首部时， 其名称不区分大小写， 且在括号中“（）”不能有任何多余的空白字符。测试服务器端的响应报文时可以使用 shdr()。
例如 当用户访问 172.16.1.100 时，重定向到 http://www.afwing.com
acl dstipaddr hdr(Host) 172.16.1.100
redirect location http://www.afwing.com if dstipaddr

(2) 基于四层协议（ip）规则 acl 测试标准
dst 和 dst_port
定义访问地址为目标地址或目标端口的 acl 规则

src 和 src_port
定义访问地址为源地址或源端口的 acl 规则
例如： 允许 10.0.0.0/24 的用户访问，其他用户将禁止
acl net10 src 10.0.0.0/24
tcp-request content accept if net10
tcp-request content reject
tcp-request content accept [{if | unless}]
Accept a connection if/unless a content inspection condition is matched

2、haproxy acl 引用

当定义好了 ACL 后我们可以利用“use_backend” 参数来引用 acl 规则，如果需要引用多个 acl 时，只需要依次在后面添加相关 acl ，也可以 匹配多个 acl，如下示例：
1. 正常写法：
use_backend static if url_static
2. 或者写法：
use_backend backend1 if aclA || aclB
use_backend backend1 if aclA || !aclC
注意上面“||”也可以使用 or 来表示
3. 非（不符合）写法：
use_backend backend1 if aclA !aclB

use_backend backend1 if aclA !aclB !aclC
4. 与（and）写法：
use_backend backend1 if aclA !aclB
use_backend backend1 if aclA !aclB !aclC

3、haproxy 常见应用实例:

haproxy 利用 acl 实现页面动静分离;

```
frontend webservs
   bind *:80
   acl url_static path_beg -i /static /images /javascript /stylesheets
   acl url_static path_end -i .jpg .gif .png .css .js .html
   acl url_php path_end -i .php

   use_backend static if url_static or host_static
   use_backend dynamic if url_php
   default_backend dynamic

backend static
   balance roundrobin
   server node1 192.168.1.111:80 check maxconn 3000

backend dynamic
   balance roundrobin
   server node2 192.168.1.112:80 check maxconn 1000
```

1.4 haproxy backend 部分配置说明

用来定义后端服务集群的配置，真实服务器，一个 Backend 对应一个或者多个实体服务器

配置 DEMO：

```
backend app
   balance     roundrobin
   server  app1 127.0.0.1:5001 check
   server  app2 127.0.0.1:5002 check
   server  app3 127.0.0.1:5003 check
   server  app4 127.0.0.1:5004 check
```

配置参数详解：

```
backend wwwpool              #定义 wwwpool 服务器组
   mode http                 #http 的 7 层模式
   option redispatch
   option abortonclose
   balance source            #负载均衡的方式，源哈希算法
```

cookie SERVERID　　#允许插入 serverid 到 cookie 中，serverid 后面可以定义
option httpchk GET /test.html #心跳检测
#option httpchk：此选项表示启用 HTTP 的服务状态检测功能。
HAProxy 作为一个专业的负载均衡器，并且它支持对 backend 部分指定的后端服务节点的 健康检查，以保证在后端的 backend 中某个节点不能服务时，
把从 frontend 端进来的客户端请求分配至 backend 中其他健康节点上，从而保证 整体服务的可用性。
#option httpchk 用法：
#option httpchk GET /index.html
#method：表示 HTTP 请求的方式，常用的有 OPTIONS、GET、HEAD 几种方式。
一般健康检查可以采用 HEAD 方式进行，而不是采用 GET 方式，这是因为 HEAD 方式没有数据返回，仅检查 Response 的 HEAD 是不是状态码 200。因此，相对于 GET，HEAD 方式更快，更简单。
#uri：表示要检测的 URL 地址，通过执行此 URL，可以获取后端服务器的运行状态，在正常情况下返回状态码 200，返回其他状态码均为异常状态。
#version：指定心跳检测时的 HTTP 的版本号。
server web1 10.1.1.2:80 cookie 2 weight 3 check inter 2000 rise 2 fall 3 maxconn 8 #用来定义多台后端真实服务器,不能用于 defaults 和 frontend 部分
#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。
#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。
#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：设置从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。
#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务器，仅仅在后端所有真实服务器均不可用的情况下才启用。

1.5 haproxy listen 部分配置说明

常常用于状态页面监控，以及后端 server 检查，是 Fronted 和 backend 的组合体
如下为 haproxy 访问状态监控页面配置
listen admin_status #Frontend 和 Backend 的组合体,监控组的名称，按需自定义名称

监控状态页示例：

访问地址 xxxx:8888/admin?stats

```
listen main *:8888
  mode http                 #http 的 7 层模式
  log 127.0.0.1 local3 err      #错误日志记录
  stats refresh 5s            #每隔 5 秒自动刷新监控页面
  stats uri /admin?stats        #监控页面的 url 访问路径
  stats realm itnihao\ welcome   #监控页面的提示信息
  stats auth admin:admin        #监控页面的用户和密码 admin,可以设置多个用户名
  stats auth admin1:admin1      #监控页面的用户和密码 admin1
  stats hide-version           #隐藏统计页面上的 HAproxy 版本信息
  stats admin if TRUE          #手工启用/禁用,后端服务器(haproxy-1.4.9 以后版本)
```

listen 常规示例：

```
listen  appli4-backup 0.0.0.0:10004
      option  httpchk /index.html
      option  persist
      balance roundrobin
      server  inst1 192.168.114.56:80 check inter 2000 fall 3
      server  inst2 192.168.114.56:81 check inter 2000 fall 3 backup

listen  ssl-relay 0.0.0.0:8443
      option  ssl-hello-chk
      balance source
      server  inst1 192.168.110.56:443 check inter 2000 fall 3
      server  inst2 192.168.110.57:443 check inter 2000 fall 3
      server  back1 192.168.120.58:443 backup

listen  appli5-backup 0.0.0.0:10005
      option  httpchk *
      balance roundrobin
      cookie  SERVERID insert indirect nocache
      server  inst1 192.168.114.56:80 cookie server01 check inter 2000 fall 3
      server  inst2 192.168.114.56:81 cookie server02 check inter 2000 fall 3
      server  inst3 192.168.114.57:80 backup check inter 2000 fall 3
      capture cookie ASPSESSION len 32
      srvtimeout     20000

      option  httpclose             # disable keep-alive
      option  checkcache             # block response if set-cookie & cacheable

      rspidel ^Set-cookie:\ IP=      # do not let this cookie tell our internal IP address

      errorloc       502     http://192.168.114.58/error502.html
```

```
    errorfile     503    /etc/haproxy/errors/503.http
```

总结:代理配置段主要有以下几部分配置块:1. Frontend:定义面向客户的监听的地址和端口,以及关联的后端的服务器组 ； 2. Backend:后端服务器组的定义；3. Listen:组合的方式直接定义 frontend 及相关的 backend;4. Defaults:默认的配置。其中 listen 配置块可以直接使用 frontend 和 backend 中任意一项参数配置， 比如 acl 语法配置以及 server 语法配置参数。

上一小节的从 haproxy 的配置文件我们知道 haproxy 相关参数基本介绍，但是在实际生产环境中，往往需要根据相关规则做请求匹配跳转，这时就需要用到 Frontend；Backend 这两个配置段，再结合 Frontend 的 acl 匹配规则来做相应的客户端请求跳转；另外，可能还会遇到客户端访问出错，这时就需要利用错误重定向功能，将其定义到相关友情提示的错误页面上，这样更加人性化。

二. haproxy 错误重定向用法

格式为: errorfile 错误代码 code 错误代码文件路径

如下为常用的优化客户端访问出现错误时错误提示页面示例

errorfile 403 /etc/haproxy/errorfiles/403.http

errorfile 500 /etc/haproxy/errorfiles/500.http

errorfile 502 /etc/haproxy/errorfiles/502.http

errorfile 503 /etc/haproxy/errorfiles/503.http

errorfile 504 /etc/haproxy/errorfiles/504.http

```
frontend  main *:80
   acl url_static       path_beg       -i /static /images /javascript /stylesheets
   acl url_static       path_end       -i .jpg .gif .png .css .js
   use_backend static          if url_static
   default_backend             app
   errorfile       503       /usr/share/haproxy/503.http
```

自定义页面 DEMO

header 设置部分直接复制下面内容或者 haproxy 默认的错误文件页，否则格式会出错。文件放和 haproxy 的错误文件页放在同一目录

```
HTTP/1.0 503 Service Unavailable
Cache-Control: no-cache
Connection: close
Content-Type: text/html

<html><body><h1>503 Service Unavailable</h1>
No server is sfsdfsfavailable to handle this request.
</body></html>
```

三. haproxy 日志配置

vim /etc/rsyslog.conf，打开下面几项

```
$ModLoad imudp
$UDPServerRun 514

local2.*    /var/log/haproxy.log   #local2 必须与 haproxy 配置中的 local2 一致
```

然后 systemctl restart rsyslog 重启 rsyslog

自定义日志格式：

```
capture cookie <name> len <length>       #捕获请求和响应报文中的 cookie 并记录日志
capture request header <name> len <length>  #捕获请求报文中指定的首部内容和长度并记录日志
capture response header <name> len <length> #捕获响应报文中指定的内容和长度首部并记录日志
```

示例：

```
capture request header Host len 256
capture request header User-Agent len 512
capture request header Referer len 15
```

四. haproxy 内置变量列表

客户端信息

%ci - 客户端 IP 地址。

%cp - 客户端端口号。

%{varname}i - 请求头中的内容，如%{X-Forwarded-For}i 用于获取代理链中的客户端 IP。

服务器信息

%{+Q}s - 后端服务器的地址。

%S - 当前使用的服务器名称。

%{+Q}<server_name> - 特定服务器的地址。

%{server.name}: 当前处理请求的服务器名称。

%{backend.name}: 当前请求所属的后端名称。

请求信息

%r - 完整的 HTTP 请求行。

%U - 请求的 URL 路径。

%{method}r - 请求方法，如 GET、POST。

%{host}i - 请求头中的 Host 字段。

响应信息

%>s - 响应状态码。

%ST - 响应时间（毫秒）。

%B - 发送的字节数，不包括头部。

%b - 发送的字节数，包括头部。

时间与日期

%t - 记录日志的时间戳。

%{format}t - 自定义时间格式，如%{Y-m-d}t。

会话与连接状态

%C - 当前连接的会话 ID。

%L - 连接是否是长连接（HTTP/1.1 Keep-Alive）。
%{date}t: 自定义日期格式的时间戳。

请求与响应细节
%r: 完整的 HTTP 请求行。
%{status}t: 响应状态码。
%b: 发送给客户端的字节数。
%U: 请求的 URL 路径。
%{query}r: 请求 URL 中的查询字符串。

其他
%Tq - 客户端等待处理的时间。
%Tw - 等待服务器响应的时间。
%TT - 总响应时间，从接收到请求到发送完响应的时间。

会话和连接状态:
%{session.id}: 当前会话的 ID（如果使用了会话管理）。
%{conn.id}: 当前连接的唯一 ID。

SSL/TLS 相关:
%{ssl_c_s_dn}i: 客户端证书的 DN（Distinguished Name）。
%{ssl_fc_sni}i: 客户端请求的 SSL 服务器名称指示（SNI）。

自定义变量:
HAproxy 也支持通过脚本或特定配置生成自定义变量，但这些不是直接内置的。

五. haproxy 监控页

1. 监控状态页示例

访问地址 xxxx:8888/admin?stats

```
listen main *:8888
  mode http                 #http 的 7 层模式
  log 127.0.0.1 local3 err     #错误日志记录
  stats refresh 5s          #每隔 5 秒自动刷新监控页面
  stats uri /admin?stats       #监控页面的 url 访问路径
  stats realm itnihao\ welcome  #监控页面的提示信息
  stats auth admin:admin       #监控页面的用户和密码 admin,可以设置多个用户名
  stats auth admin1:admin1     #监控页面的用户和密码 admin1
  stats hide-version          #隐藏统计页面上的 HAproxy 版本信息
  stats admin if TRUE         #手工启用/禁用,后端服务器(haproxy-1.4.9 以后版本)
```

2. 监控页参数列表

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
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 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)

3. 监控参数详解

3. 1 Frontend statistics

session rate(每秒的连接会话信息)：
cur:当前每秒建立的会话数
max:每秒建立的最大会话数；
limit:前端每秒将接受的最大会话数，由 rate-limit sessions 设置设置。如果超过此限制，则其他连接将在套接字的 backlog 中保持待处理（在系统缓冲区中）。

sessions(会话信息)：
cur:当前保持的会话量
max:同一时间保持的最大会话量
limit: 限制的会话量上限
可在 global、frontend、listen、defaults 段通过 maxconn 设置，表示和客户端（即 frontend）的最大连接并发数。
其中 global 段的值是硬限制，frontend、listen、defaults 段的 maxconn 值不能超过 global 段的值。
Total:从 HAProxy 启动到现在，总共累计会话量
Cum. connections （累积连接数） 自上次重新加载负载均衡器以来建立的累计连接数。
Cum. sessions （累积会话数）自上次重新加载以来建立的累积会话数（端到端连接）。
Cum. HTTP requests （累积 HTTP 请求数）自上次重新加载以来的累计 HTTP 请求数。
HTTP 1xx responses
HTTP 2xx responses
Compressed 2xx

HTTP 3xx responses
HTTP 4xx responses
HTTP 5xx responses
Other responses 其他响应收到其他指标未涵盖的响应的 HTTP 请求总数。
Intercepted requests 拦截的请求 输入的请求总数
Cache lookups 缓存查找 检查资源缓存的总次数。
Cache hits 缓存命中数 返回缓存资源的总次数。
Failed hdr rewrites 失败的 hdr 重写由于缓冲区中没有足够的空间而无法添加或设置 HTTP 标头的次数。尝试增加 tune.maxrewrite 或 tune.bufsize。
tune.maxrewrite or tune.bufsize.

Bytes(流量统计)：
In:网络的字节输入总量
Out:网络的字节输出总量

Denied(拒绝统计信息)：
Req:拒绝请求量
--http-request deny 停止进一步评估规则并立即拒绝请求，返回 HTTP 403 Forbidden 错误码。
--http-request reject 关闭连接而不发送任何响应。
--http-request silent-drop 立刻停止处理客户端请求，但保持连接打开一段时间以便客户端重试。
--http-request tarpit 阻塞请求，使客户端等待直到超时。
--http-response silent-drop 放弃生成的响应，不发送给客户端。
--tcp-request connection silent-drop 在连接建立前丢弃数据包。
--tcp-request content reject 创建会话后关闭连接，但不在 HTTP 解析阶段之前发送任何响应。
--tcp-request content silent-drop 创建会话后丢弃数据包，不发送任何响应。
Resp:拒绝回复量

Errors(错误统计信息)：
Req:错误请求量
可能的原因包括：

--客户端提前终止
--来自客户端的读取错误
--客户端超时
--客户端关闭了连接
--客户端发送了格式错误的请求
--该请求被 Tar 坑坑洼洼

Server(real server 信息)：
status
--当 Status 为 OPEN 时，前端运行正常并准备好接收流量。您可以通过执行 disable

frontend Runtime API 命令来禁用前端，该前端会将其状态更改为 STOP。下面是调用该命令的示例：

--$ echo "disable frontend www" | \ sudo socat stdio tcp4-connect:127.0.0.1:9999

--Use the enable frontend command, which has the same syntax, to change the frontend's status back to OPEN.

3. 2 Backend Statistics

Queue: 表示 HAProxy 中当前排队等待处理的连接数：

Max：从 HAProxy 启动到现在，最大的队列请求数；

Limit：队列中等待处理的最大请求阈值（只能用于 server 字段）。若队列中等待请求的数量超过该阈值，则下一个请求将被定向到其他服务器。默认为 0，表示没有限制；

session rate(每秒的连接会话信息)：

cur:当前每秒与服务器建立的会话数

max:每秒与服务器建立的最大会话数；

sessions(会话信息)：

cur: 与服务器的活动连接数。

max: 与给定服务器同时建立的最多连接

limit: 服务器允许的最大连接数，由服务器行上的 maxconn 参数设置。

--后端行显示 Limit 的 fullconn 值，或者如果未设置，则使用以下公式：路由到此后端的前端的 Sessions Limit 值之和除以 10。

--因此，如果您有两个前端，每个前端的 Sessions Limit 为 1000，则它们的总和将为 2000。将其除以 10 得到此后端的限制值 200。

Total:与服务器的累计连接数（包括未成功建立连接的次数，也就是所有尝试连接的次数，包括成功的和因为各种原因未成功的）

（此值会大于 LBTot，因为后端负载过高的时候，分配给后端的一次请求可能会多次建立连接，tcp 建立连接超时的情况，每新建一次 TCP 连接都会计入 Total 次数）

Cum. sessions 与此服务器建立的累计连接数

New connections 与此服务器建立新连接而不是重用现有连接的次数

Reused connections（重复使用的连接） 重复使用现有连接的次数。

Cum. HTTP responses （累积 HTTP 响应数）从此服务器收到的 HTTP 响应的累积数。

HTTP 1xx responses

HTTP 2xx responses

HTTP 3xx responses

HTTP 4xx responses

HTTP 5xx responses

Other responses

Failed hdr rewrites 失败的 hdr 重写由于缓冲区中没有足够的空间而无法添加或设置 HTTP 标头的次数。尝试增加 tune.maxrewrite 或 tune.bufsize。

tune.maxrewrite or tune.bufsize.

Queue time 连接在等待到服务器的连接槽时保持排队状态的时间（以毫秒为单位），是最近 1024 个成功连接的平均值。

Connect time 成功连接到服务器所花费的时间（以毫秒为单位），是最近 1024 次成功连

接的平均值。
Response time 响应时间 最近 1024 次成功连接的平均服务器响应时间（以毫秒为单位）。
Total time 总时间 最近 1024 个成功连接的平均总会话时间（以毫秒为单位）。
LBTot: 显示选择给定服务器来为请求提供服务的总次数。这可能是由于正常的负载平衡，也可能是由于从失败的服务器重新分派。
Last：自上次收到连接以来的时间。

Bytes(流量统计)：
In:网络的字节输入总量
Out:网络的字节输出总量

Denied(拒绝统计信息)：
Req:拒绝请求量
http-request deny 停止进一步评估规则并立即拒绝请求，返回 HTTP 403 Forbidden 错误码
http-request reject 关闭连接而不发送任何响应。
http-request silent-drop 立刻停止处理客户端请求，但保持连接打开一段时间以便客户端重试。
http-request tarpit 阻塞请求，使客户端等待直到超时。
http-response silent-drop 放弃生成的响应，不发送给客户端。
tcp-request content reject 创建会话后关闭连接，但不在 HTTP 解析阶段之前发送任何响应。
tcp-request content silent-drop 创建会话后丢弃数据包，不发送任何响应。
tcp-response content silent-drop 是唯一一个会影响“Resp”列的指令。每当有响应被这条指令放弃时，“Resp”列都会增加。
Resp:拒绝回复量，由于后端中存在以下任何配置指令而被拒绝的任何给定服务器的响应数：
http-response deny
tcp-response content reject

Errors(错误统计信息)：
conn:错误链接量，在建立连接阶段发送错误的次数
Resp:获取响应时遇到的错误。
--当负载均衡器成功建立了与服务器的连接，但随后因某些原因无法接收预期的响应时，此计数会增加。
--特殊操作如 tcp-response content close 或者服务器主动关闭连接也会导致此类错误。

Warnings(警告统计信息)：
Retr:retries 重试连接的总次数；
Redis:redispatches 重新分配次数（当 real server 挂掉后，强制定向到其他健康的服务器）；

Server(real server 信息)：
Status:后端机的状态，包括 UP 和 DOWN

UP The server is reporting as healthy.

DOWN The server is reporting as unhealthy and unable to receive requests.

NOLB You've added http-check disable-on-404 to the backend, and the configured URL for health checks has returned an HTTP 404 response.

MAINT The server has been disabled or put into maintenance mode. 此操作会影响服务器宕机次数 Dwn（+1）、宕机时间 Dwntme。

DRAIN The server has been put into drain mode. 不增加宕机次数 Dwn 和宕机时间 Dwntme。

no check

LastChk:持续检查后端服务器的时间
Wght:权重
Act:活动后端的数量
Bck:备份的服务器数量
Chk:心跳检测失败的次数
Dwn:后端服务器从 UP 转为 DOWN 的次数
Dwntme:总的 downtime 时间(haproxy 启动以来累计的 downtime 时间))
Thrtle:当前分配请求的百分比（和正常分配数量相除）

六、haproxy ACL 配置

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。 http://cbonte.github.io/haproxy-dconv/2.0/configuration.html#7

1、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

匹配内容：请求头、请求方式、请求 URL、请求参数、请求内容、source ip、source port、dest ip、dest port
```

1.1 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。
```

1.2 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.

1.3 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

1.4 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

1.5 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

2、ACL 调用方式

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

3、ACL 的实例

3.1 基于 host 匹配

```
listen web_host
 bind 192.168.32.204:80
 mode http
 balance roundrobin
 log global
 option httplog
 acl web_host hdr_dom(host) www.magedu.net
 use_backend magedu_host if web_host
 default_backend default_web

backend magedu_host
 mode http
```

```
 server web1 10.0.0.201:80 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.2 基于源 IP 或子网调度访问

```
将指定的源地址调度至指定的 web 服务器组。
frontend  main *:5000
 mode http
 balance roundrobin
 log global
 option httplog
 acl ip_range_test src 10.0.0.0/16 192.168.32.201
 use_backend magedu_host if ip_range_test
 default_backend default_web

backend magedu_host
mode http
server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
mode http
server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.3 基于源地址的访问控制

```
拒绝指定 IP 或者 IP 范围访问
frontend  main *:5000
 mode http
 balance roundrobin
 log global
 option httplog
 acl block_test src 192.168.32.203 192.168.0.0/24
 block if block_test
 default_backend default_web

backend magedu_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
```

```
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.4 匹配浏览器类型

```
匹配客户端浏览器，将不同类型的浏览器调动至不同的服务器组
frontend  main *:5000
 mode http
 balance roundrobin
 log global
 option httplog
 acl web_host hdr_dom(host) www.magedu.net
 use_backend magedu_host if web_host
 acl redirect_test hdr(User-Agent) -m sub -i "Mozilla/5.0 (Windows NT 6.1; WOW64;Trident/7.0;
rv:11.0) like Gecko"
 redirect prefix http://192.168.32.201 if redirect_test
 #redirect location http://192.168.32.201 if redirect_test  直接跳转不带后缀
 default_backend default_web

backend magedu_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.5 基于文件后缀名实现动静分离

```
frontend  main *:5000
 mode http
 balance roundrobin
 log global
 option httplog
 acl php_server path_end -i .php
 use_backend php_server_host if php_server
 acl image_server path_end -i .jpg .png .jpeg .gif
 use_backend image_server_host if image_server
 default_backend default_web

backend php_server_host
 mode http
 server web1 10.0.0.203 check inter 2000 fall 3 rise 5

backend image_server_host
```

```
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.6 匹配访问路径实现动静分离

```
frontend  main *:5000
 mode http
 balance roundrobin
 log global
 option httplog
 acl static_path path_beg -i /static /images /javascript
 use_backend static_path_host if static_path
 default_backend default_web

backend static_path_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.7 根据请求参数匹配

```
frontend  main *:5000
 mode http
 balance roundrobin
 log global
 option httplog
 acl static_path urlp(username) -m str test
 use_backend static_path_host if static_path
 default_backend default_web

backend static_path_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.8 根据请求方式匹配

```
#示例 1：
frontend  main *:5000
 mode http
 balance roundrobin
 log global
 option httplog
 acl static_path path_end -i .php
 use_backend static_path_host if METH_POST static_path
 default_backend default_web

backend static_path_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5

#示例 2：
frontend  main *:5000
 acl static_path method POST
 use_backend static_path_host if static_path
 default_backend default_web

backend static_path_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.9 根据是否 https 请求匹配

```
frontend  main *:5000
 acl https ssl_fc
 use_backend static_path_host if https
 default_backend default_web

backend static_path_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
```

```
server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.10 实现 443 端口复用

tls1.2_ticket_auth

```
global

defaults

frontend ssl
 mode tcp
 bind *:443
 tcp-request inspect-delay 5s
 tcp-request content accept if { req.ssl_hello_type 1 }

 #acl ssh_payload payload(0,7) -m bin 5353482d322e30
 acl nginx_app req_ssl_sni -i a1.google.xyz
 acl canddy_app req_ssl_sni -i a2.google.xyz

 use_backend nginx if nginx_app
 use_backend canddy if canddy_app

backend canddy
 #mode tcp
 server canddy 127.0.0.1:5443

backend nginx
 #mode tcp
 option ssl-hello-chk
 server webserver 127.0.0.1:4443
```

3.10 HTTP 访问控制

```
listen web_host
 bind 192.168.32.204:80
 mode http
 balance roundrobin
 log global
 option httplog
 acl static_path path_beg -i /static /images /javascript
 use_backend static_path_host if static_path
 acl badguy_deny src 192.168.32.200
 http-request deny if badguy_deny
 http-request allow
 default_backend default_web
```

```
backend static_path_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.11 预定义 ACL 使用

预定义 ACL

```
ACL name              Equivalent to                    Usage
FALSE                 always_false                     never match
HTTP                  req_proto_http                   match if protocol is valid HTTP
HTTP_1.0              req_ver 1.0                      match HTTP version 1.0
HTTP_1.1              req_ver 1.1                      match HTTP version 1.1
HTTP_CONTENT          hdr_val(content-length) gt 0     match an existing content-length
HTTP_URL_ABS          url_reg ^[^/:]*://               match absolute URL with scheme
HTTP_URL_SLASH        url_beg /                        match URL beginning with "/"
HTTP_URL_STAR         url *                            match URL equal to "*"
LOCALHOST             src 127.0.0.1/8                  match connection from local host
METH_CONNECT          method CONNECT                   match HTTP CONNECT method
METH_DELETE           method DELETE                    match HTTP DELETE method
METH_GET              method GET HEAD                  match HTTP GET or HEAD method
METH_HEAD             method HEAD                      match HTTP HEAD method
METH_OPTIONS          method OPTIONS                   match HTTP OPTIONS method
METH_POST             method POST                      match HTTP POST method
METH_PUT              method PUT                       match HTTP PUT method
METH_TRACE            method TRACE                     match HTTP TRACE method
RDP_COOKIE            req_rdp_cookie_cnt gt 0          match presence of an RDP cookie
REQ_CONTENT           req_len gt 0                     match data in the request buffer
TRUE                  always_true                      always match
WAIT_END              wait_end                         wait for end of content analysis
```

预定义 ACL 使用

```
listen web_host
 bind 192.168.32.204:80
 mode http
 balance roundrobin
 log global
 option httplog
 acl static_path path_beg -i /static /images /javascript
 use_backend static_path_host if HTTP_1.1 TRUE static_path
 default_backend default_web
```

```
backend php_server_host
 mode http
 server web1 192.168.32.203 check inter 2000 fall 3 rise 5

backend static_path_host
 mode http
 server web1 10.0.0.201 check inter 2000 fall 3 rise 5

backend default_web
 mode http
 server web1 10.0.0.202:80 check inter 2000 fall 3 rise 5
```

3.12 常见的控制类型

```
#常见的控制类型
设置或修改请求头
作用：设置或修改请求中的头信息。
示例：
set-header X-Forwarded-For %[src]

添加请求头
作用：在请求中添加新的头信息。
示例：
add-header X-Client-IP %[src]

删除请求头
作用：从请求中删除指定的头信息。
示例：
del-header X-Cache

重定向请求
作用：将请求重定向到另一个 URL。
示例：
redirect location http://example.com/newpath if { some_condition }

跟踪会话状态
作用：跟踪和管理会话状态。
示例：
track-sc0 src table my_table

设置变量
作用：在请求处理过程中设置变量。
示例：
set-var(txn.is_admin) 1 if { path_beg /admin }
```

删除变量
作用：删除请求处理过程中的变量。
示例：
unset-var(txn.is_admin) if { path_beg /public }

设置防火墙标记
作用：设置或修改防火墙标记。
示例：
set-mark 0x100 if { src 192.168.1.0/24 }

设置日志级别
作用：调整请求的日志记录级别。
示例：
set-log-level warning if { path_beg /error }

捕获请求信息
作用：捕获请求中的某些信息并记录到日志中。
示例：
capture req.uri len 128

替换请求头中的值
作用：替换请求头中的某个值。
示例：
replace-value hdr(X-Real-IP) %[src] if { hdr(X-Real-IP) -m sub proxy }

替换整个请求头
作用：替换整个请求头。
示例：
replace-header Host example.com if { hdr(host) -m sub old.example.com }

允许请求通过
作用：允许请求通过，常用于访问控制。
示例：
allow if { src 192.168.1.0/24 }

将请求放入陷阱
作用：将请求放入陷阱，延迟响应。
示例：
tarpit if { src 192.168.1.0/24 }

修改请求路径
作用：改变请求的路径。
示例：

```
set-path /newpath%[path] if { path_beg /oldpath }
```

修改请求 URI
作用：改变请求的 URI。
示例：

```
set-uri /newuri%[uri] if { uri_beg /olduri }
```

修改请求查询字符串
作用：改变请求中的查询字符串。
示例：

```
set-query param1=value1&param2=value2 if { query -m sub oldparam }
```

重定向请求到指定前缀
作用：将请求重定向到指定的前缀。
示例：

```
redirect prefix http://example.com/newpath code 301 if { path_beg /oldpath }
```

重定向请求到指定 URL
作用：将请求重定向到指定的 URL。
示例：

```
redirect location http://example.com/newpage code 301 if { path_beg /oldpage }
```

重定向请求到指定协议
作用：将请求重定向到指定的协议（HTTP 或 HTTPS）。
示例：

```
redirect scheme https if { ssl_fc }
```

重定向请求并设置状态码
作用：将请求重定向并设置特定的 HTTP 状态码。
示例：

```
redirect status 404 if { path_beg /notfound }
```

拒绝请求
作用：拒绝请求，常用于访问控制。
示例：

```
reject if { src 192.168.1.0/24 }
```

发送客户端名称
作用：发送客户端的名称信息。
示例：

```
send-name if { src 192.168.1.0/24 }
```

发送客户端状态
作用：发送客户端的状态信息。

示例：
send-state if { src 192.168.1.0/24 }

发送 PROXY 协议头部
作用：发送 PROXY 协议头部，用于传递客户端的真实 IP 地址。
示例：
send-proxy if { src 192.168.1.0/24 }

发送 PROXY v2 协议头部
作用：发送 PROXY v2 协议头部，用于传递更详细的客户端信息。
示例：
send-proxy-v2 if { src 192.168.1.0/24 }

使用映射文件设置变量
作用：根据映射文件设置变量。
示例：
set-map(txn.user_role) hdr(User-Role) /etc/haproxy/user_roles.map

使用映射文件设置变量并处理结尾
作用：根据映射文件设置变量，并处理结尾。
示例：
set-map-end(txn.user_role) hdr(User-Role) /etc/haproxy/user_roles.map

使用映射文件设置变量并处理子串
作用：根据映射文件设置变量，并处理子串。
示例：
set-map-sub(txn.user_role) hdr(User-Role) /etc/haproxy/user_roles.map

3.12 鉴权方式

```
userlist basic-auth-list
 group is-regular-user
 group is-admin

 user admin  password 0y]wNEj0j                                groups is-admin
 user michal password $5$gZZsvtRWI$9JIU8pfHLG8BtYW5tceAKD1oNAjjfL5e4LwUfAW1sqA
groups is-regular-user
 user milosz password $1$R29iAdV/$1QUKx8eo6e5pcMIEgaZwt0                groups is-regular-
user
 user guest  insecure-password guestpassword

backend app
   acl is_admin http_auth(basic-auth-list)

   http-request auth realm "Admin Area" if !is_admin
```

```
http-request deny if !is_admin
balance     roundrobin
server  app1 192.168.1.166:80 check
```

七、报文、请求头修改

1：报文修改

```
http-response { allow | deny | add-header | set-nice | capture id | redirect | set-header | del-header | replace-header | replace-value | set-status [reason ] | set-log-level | set-mark | set-tos | add-acl() | del-acl() | del-map() | set-map() | set-var() | unset-var() | { track-sc0 | track-sc1 | track-sc2 } [table] | sc-inc-gpc0() | sc-set-gpt0() | silent-drop | } [ { if | unless } ]

http-request { allow | tarpit | auth [realm ] | redirect | deny [deny_status ] | add-header | set-header | capture [ len | id ] | del-header | set-nice | set-log-level | replace-header | replace-value | set-method | set-path | set-query | set-uri | set-tos | set-mark | add-acl() | del-acl() | del-map() | set-map() | set-var() | unset-var() | { track-sc0 | track-sc1 | track-sc2 } [table] | sc-inc-gpc0() | sc-set-gpt0() | silent-drop | } [ { if | unless } ]


tcp-response content' expects 'accept', 'close', 'reject', 'send-spoe-group', 'sc-inc-gpc0(*)', 'sc-inc-gpc1(*)', 'sc-set-gpt0(*)', 'set-var(*)', 'unset-var(*)', 'silent-drop' in backend 'static' (got 'add').
```

2：作用域

Scope	Description
proc	haproxy 进程所有周期内
sess	tcp session 周期内
txn	整个 http 请求周期内
req	the variable is available during the HTTP request phase only
res	the variable is available during the HTTP response phase only

```
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

3：请求头设置

```
frontend www
 bind :80
 http-request set-var(txn.useragent) req.hdr(user-agent)  #这里把请求参数存储到变量中供下面指令使用，因为 http-response 无法获取到请求参数
 http-response add-header X-Message str("My message") if { var(txn.useragent) -i -m sub Chrome }
```

```
global

 # Set a variable named "my_string" to a string value
 set-var proc.my_string str("some string value")

 # Set a variable named "my_num_var" to an integer value
 set-var proc.my_num_var int(123)


frontend www
 bind :80
 http-response set-header X-MyString %[var(proc.my_string)]
 http-response set-header X-MyNumber %[var(proc.my_num_var)]
```

添加请求头
#在响应中添加客户端 IP 和端口的自定义头
http-request add-header X-Client-Info "IP=%ci:Port=%cp"
http-request set-var(txn.http_version) req.ver
http-response add-header Via "%[var(txn.http_version)] %[hostname]"

设置请求头
#对需要重写的请求进行 PHPSESSID cookie 的添加或修改

http-request set-cookie PHPSESSID=%[var(sess.uniqid)] if needs_rewrite

设置请求方法

http-request set-method POST

删除请求头

http-request del-header X-Forwarded-For

替换请求头

acl h_xff_exists req.hdr(X-Forwarded-For) -m found
http-request replace-header X-Forwarded-For (.*) %[src],\1 if h_xff_exists #这里\1 用于捕获括号中的值，即添加新值到捕获的值中

替换 path 中的指定路径

http-request replace-path ^/api(/?.*) \1 #这里替换了 url 中的/api 路径部分
http-request replace-path ^(/.*)/api(/?.*) \1\2 #这里替换了 url 中的/api 路径部分

部分替换请求头

```
frontend www
 bind :80
 http-request replace-value Host (.*):.*  \1  #括号外是替换掉的部分，括号里面的保留
 use_backend webservers
```

替换查询字符串

```
frontend www
 bind :80
 http-request set-query %[query,regsub(%3D,=,g)]
 use_backend webservers
```

重排查询参数顺序

```
frontend www
 bind :80
 http-request set-query user=%[urlp(user)]&group=%[urlp(group)]
 use_backend webservers
```

重写 URL

```
backend b_squid
 acl https ssl_fc
 http-request set-uri https://%[req.hdr(Host)]%[path]?%[query] if https
 http-request set-uri http://%[req.hdr(Host)]%[path]?%[query] unless https
```

设置路径

acl p_ext_jpg path_end -i .jpg
acl p_folder_images path_beg -i /images/

http-request set-path /images/%[path] if !p_folder_images p_ext_jpg

禁用缓存

http-response set-header Cache-Control "no-store, no-cache, must-revalidate, post-check=0, pre-check=0"
http-response set-header Pragma "no-cache"
http-response set-header Expires "Wed, 17 Aug 2005 00:00:00 GMT"

4：临时变量

```
# 设置变量：set-var(变量名) 变量值
# 读取变量：%[var(变量名)]
frontend www
  bind :80
  http-request set-var(txn.mypath) path
  http-response add-header X-Path %[var(txn.mypath)]
```

5：环境变量

```
# 设置环境变量：setenv 变量名 变量值
# 读取环境变量：$变量名
global
  setenv IP           192.168.56.10
  presetenv IP          192.168.56.10   #如果变量存在则不会覆盖
  setenv PORT         8080
  setenv USER         foo
  setenv PASS         bar
  setenv DEFAULT_BACKEND be_app
  setenv LOG_ADDRESS    192.168.56.5
  setenv LOG_LEVEL      notice

defaults
  # Set syslog logging
  log "$LOG_ADDRESS" local0 "$LOG_LEVEL"

userlist credentials
  # Set a Basic auth username and password
  user "$USER" insecure-password "$PASS"

frontend fe_main
  # Set the IP address and port using variables
  bind "$IP":"$PORT"
```

```
  # enforce Basic authentication
  http-request auth unless { http_auth(credentials) }

  default_backend "$DEFAULT_BACKEND"
```

八、返回码、返回页、返回信息

json 格式返回页

```
HTTP/1.1 503 Service UnavailableCache-Control: no-cacheConnection: closeContent-Type: application/json{ "errors" : [ { "status" : "503", "title" : "Service unavailable", "detail" : "No server is available to handle this request." } ] }

http-errors json
   errorfile 404 /etc/haproxy/errors/404-json.http
   errorfile 503 /etc/haproxy/errors/503-json.http

frontend api
   bind :8080
   default_backend apiservers
   errorfiles json
   http-response return status 404 default-errorfiles if { status 404 }
```

html 格式返回页

```
HTTP/1.1 404 Not Found
Cache-Control: no-cache
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8" />
      <title>404 Not Found</title>
   </head>
   <body>
      <main>
         <h1>404 Not Found</h1>

         This is my custom 404 Not Found page!
      </main>
   </body>
</html>
```

```
errorfile 404 /etc/haproxy/errors/404.http

frontend site1
  bind :80
  default_backend webservers
  errorfiles myerrors
  http-response return status 404 default-errorfiles if { status 404 }
```

返回 json 和 html 格式

```
frontend site1
  bind :80
  default_backend webservers
  http-request return status 200 content-type text/html file /etc/haproxy/errors/maintenance.html
  http-request return status 200 content-type application/json file /etc/haproxy/errors/maintenance.html if url_static
```

九、capture 捕获请求信息输出到日志

```
http-request capture url,lower len 10
http-request capture path,upper len 10
http-request capture method,upper len 10
http-request capture src,upper len 10
http-request capture urlp(test) len 10

capture cookie ASPSESSION len 32
capture response header Content-length len 9
capture response header Location len 15
capture request header X-Forwarded-For len 15
capture request header Host len 15
capture request header Yixiao len 15
```

十、请求重定向和路由

1：http 重定向到 https

```
frontend myfrontend
 mode http
 bind :80
 bind :443 ssl crt /site.pem

 # Redirect HTTP to HTTPS
 http-request redirect scheme https unless { ssl_fc }
```

```
default_backend web_servers
```

2：重定向 url

url 地址栏也改变

```
 http-request redirect prefix http://192.168.32.201 if redirect_test    #带后缀跳转
 http-request redirect location http://192.168.32.201 if redirect_test   #直接跳转不带后缀
----------------------------------------------------------------------------------------------


#path 里添加指定前缀
frontend www
 bind :80
 acl begins_with_api path_beg /api/v2/
 http-request redirect code 301 prefix /api/v2 unless begins_with_api
 #http-request redirect refix /api/v2 unless begins_with_api  #不改变返回码
 default_backend webservers

#path 里删除指定路径
/etc/haproxy/path_redirect.map 内容
^/api/(.*)$    /\1
http-request redirect location %[capture.req.uri,map(/etc/haproxy/path_redirect.map)] code 301 if { path_beg /api/ }
```

3：重写 url

不改变地址栏 url 地址，更改发送到后端的 url 地址

```
替换 path 中的指定路径
http-request replace-path ^/api(/?.*) \1        #这里删除了 url 开头的/api 路径部分
http-request replace-path ^(/.*)/api(/?.*) \1\2   #这里删除了 url 中的/api 路径部分
http-request replace-path ^(/?.*) /api\1        #这里在原 url 开头部分添加/api
http-request replace-path ^(/api?.*/).*.php$ \1   #这里删除了 url 最后一个/后面的 1.php 路径部分
http-request replace-path ^/api(/?.*/).*.php$ \1   #这里删除了 url 开头的/api 和最后一个/后面的 1.php 路径部分
```

十一、时间格式

```
us：微秒。1us = 1/1000000s
ms：毫秒。1ms = 1/1000s。默认值
s：秒。1s = 1000ms
m：分钟。1m = 60s = 6000ms
h：小时。1h = 60m = 3600s = 3600000ms
d：天。1d = 24h = 1440m = 86400s = 86400000ms
```

十二、SSL 配置

```
ca-base <dir>指定一个获取 SSL CA 证书和 CRLs 的默认目录。
```

ca-file 和 crl-file 两个指令如果设置为相对路径，则以此目录为基础。
ca-file 和 crl-file 如果设置为绝对路径，则 ca-base 会被忽略。

十三、进程绑定特定 CPU

cpu-map <"all"|"odd"|"even"|process_num> <cpu-set>...
将一个进程绑定到指定的 CPU 集上，此设置仅在 Linux 2.6 或以上版本才有效。进行此设置意味着进程只允许在某一 CPU 集上运行。
第一个参数指定将要被绑定的进程数，必须是 1~32 或 1~64 中的数字（取决于你的机器是 32 还是 64 位的），此时 nbproc 上的 PID 将被忽略。设置为 all 表示所有进程，设置为 odd 表示奇数的进程，even 表示偶数的进程。这和使用 bing-process 指令是一个意思。
第二个参数指定 CPU 集。每个 CPU 集都有自己的唯一编号，范围在 0~31 或 0~63，由两个这样的数字加上一个“-”连接符组成。你可以指定多个 CPU 或范围用以允许进程对其做绑定。
你可以编写多个 cpu-map 指令，但只有最后一个指令会生效，上面的都会被覆盖。

十四、stick-table

1：参数解析

type ip|integer|string：使用什么类型的 key 作为客户端标识符。可以是客户端的源 IP，可以是一个整数 ID 值，也可以是一段从请求报文或响应报文中匹配出来的字符串。
size：表中允许的最大 stickiness 记录数量。单位使用 k、m 和 g 表示，分别表示 1024、2^20 和 2^30 条记录。
expire：stickiness 记录的过期时长。 us：微秒。ms：默认值 s。s:秒。m：分钟。h：小时。d：天。
nopurge：默认情况下，当表满后，如果还有新的 stickiness 记录要插入进来，haproxy 会自动将一部分老旧的 stickiness 记录 flush 掉，以释放空间存储新纪录。指定 nopurge 后，将不进行 flush，只能通过记录过期来释放表空间，因此该选项必须配合 expire 选项同时使用。
peers：指定要将 stick table 中的记录 replication 到对端 haproxy 节点。
store：指定要存储在 stick table 中的额外状态统计数据。其中代表后端服务器的标识符 server ID(即 key/value 的 value 部分)会自动插入，无需显式指定。

2：限流选项

server_id : 一个整数，request 被分配到的 server ID
gpc0 : 通用计数器
conn_cnt : connection 计数器，记录了匹配当前 entry 的，从一个客户端处接收到的连接的绝对数量，这个数量并不意味着被 accepted 的连接数量，单纯就是收到的数量。
conn_cur : 当前 connection 数，当一新 connection 匹配指定的 entry 的时候，这个数值增加，当 connection 结束的时候，这个数值减少。通过这个值可以了解一个 entry 上任意时间点的准确的连接数。
conn_rate() : connection 的连接频率 ，指定时间范围内(毫秒为单位)建立 connection 的频率。
sess_cnt : session 计数器。记录了匹配当前 entry 的，从一个客户端处接收到的 session 的绝

对数量。一个 session 指的是一个已经被 layer 4 规则接受的 connection。
sess_rate() : session 的连接频率，指定时间范围内(毫秒为单位)建立 session 的频率。
http_req_cnt : HTTP 请求计数器。 记录了匹配当前 entry 的，从一个客户端接受到的 HTTP 请求的绝对数量。 无论这个请求是合法还是非法。当开启了客户端 keep-alive 的时候，这个数值与 sess_cnt 计数器的值就会不同。
http_req_rate() : HTTP 的请求频率，指定时间范围内(毫秒为单位)收到的 HTTP 请求的频率。 无论这个请求是合法还是非法。当开启了客户端 keep-alive 的时候，这个数值与 sess_rate() 数器的值就会不同。
http_err_cnt : HTTP 错误计数器。 记录了匹配这个 entry 的 HTTP 错误的绝对数量，包含: 无效的、被截断的请求 被拒绝的或封堵的请求 认证失败 4xx 错误
http_err_rate() : HTTP 的请求错误频率，指定时间范围内(毫秒为单位)匹配的 entry 产生的 HTTP 错误的频率。
bytes_in_cnt : 一个匹配 entry 的客户端发往服务器的字节数。headers 也包含在统计中，通常用于图片或者 video 服务器限制上传文件。
bytes_in_rate() : 收到字节频率计数器，指定时间范围内(毫秒为单位)收到的字节数的频率，通常用于防止用户上传太快上传太多内容。这个计数器不建议使用，因为会有不公平情况，建议使用 byte_in_cnt。
bytes_out_cnt : 服务器发往客户端的字节数。headers 也包含在统计中，通常用于防止机器人爬站。
bytes_out_rate() : 发送字节频率计数器，指定时间范围内(毫秒为单位)服务器发送给客户端的字节数的频率。通常用于防止用户下载太快太多内容。这个计数器也不建议使用，因为会有不公平情况，建议使用 byte_out_cnt。

```
frontend www
 bind :80
 stick-table type ip size 1m expire 10s store http_req_rate(10s),conn_rate(10s),bytes_in_rate(10s)
```

http_req_cnt : HTTP 请求计数器
记录了匹配当前 entry 的，从一个客户端接受到的 HTTP 请求的绝对数量。无论这个请求是合法还是非法。

http_req_rate(10s):
这个参数记录了每个 IP 地址在最近 10 秒内的 HTTP 请求速率。主要用于 7 层负载

conn_rate(10s):
表示每个 IP 地址在最近 10 秒内的连接建立速率。这个数量并不意味着被 accepted 的连接数量，单纯就是收到的数量.主要用于 4 层负载

conn_cnt : Connection 计数器
记录了匹配当前 entry 的，从一个客户端处接收到的连接的绝对数量。这个数量并不意味着被 accepted 的连接数量，单纯就是收到的数量

conn_cur
表示每个 IP 地址在当前已经建立的连接数量（完成 3 次握手）。主要用于 4 层负载

sess_cnt : Session 计数器
记录了匹配当前 entry 的，从一个客户端处接收到的 session 的绝对数量。如果开启了 keepalived，则 session 和请求数不同。
一个 session 指的是一个已经被 layer 4 规则接受的 connection。

sess_rate() : session 的连接频率 (takes 12 bytes).
这个值统计指定时间范围内(毫秒为单位)进来的 session 的频率。
这个数值可以通过 ACL 匹配一些规则。

http_err_cnt : HTTP 错误计数器
记录了匹配这个 entry 的 HTTP 错误的绝对数量，包含:无效的、被截断的请求被拒绝的或封堵的请求认证失败 4xx 错误

http_err_rate() : HTTP 的请求错误频率 (takes 12 bytes).
这个值统计指定时间范围内(毫秒为单位)匹配的 entry 产生的 HTTP 错误的频率。

bytes_in_cnt : 一个匹配 entry 的客户端发往服务器的字节数。
Headers 也包含在统计中，通常用于图片或者 video 服务器限制上传文件。

bytes_in_rate() : 收到字节频率计数器(takes 12 bytes).
这个值统计指定时间范围内(毫秒为单位)收到的字节数的频率。通常用于防止用户上传太快上传太多内容。

bytes_out_cnt : 服务器发往客户端的字节数。
Headers 也包含在统计中，通常用于防止机器人爬站。

bytes_out_rate() : 发送字节频率计数器(takes 12 bytes).
这个值统计指定时间范围内(毫秒为单位)服务器发送给客户端的字节数的频率。通常用于防止用户下载太快太多内容。

gpc0
通用计数器，手动增加计数，主要用于统计 acl 规则触发次数

3：限流示例

http-request: 这个指令告诉 HAProxy 在处理 HTTP 请求时执行特定的操作。
track-sc0: 这部分指令指示 HAProxy 跟踪客户端请求，并更新一个称为“sticky counter”的内部计数器。这里的“sc0”是三种粘滞性计数器之一，用于记录与特定客户端相关的请求次数或会话。使用 sc0、sc1、sc2 等可以区分不同的跟踪目的或策略。
src: 这个关键字指定了用于跟踪的键，即客户端的源 IP 地址。这意味着 HAProxy 会基于请求的源 IP 来识别客户端，并确保来自同一 IP 地址的请求被定向到同一后端服务器，从而保持会话的连续性。
http-request track-sc0 src 的作用是确保来自同一客户端 IP 的所有 HTTP 请求都能路由到同一个后端服务器，

```
#同一 ip 10s 内请求数量不能超过 10 个，不管请求是否成功
frontend www
 bind :80
 stick-table  type ip  size 1m  expire 10s  store http_req_rate(10s)
 http-request track-sc0 src
 http-request deny if { sc_http_req_rate(0) gt 10 }
---------------------------------------------------------------------------------------------------------------

frontend
 stick-table type ip size 100k expire 30s store conn_rate(3s) #connection rate
 stick-table type ip size 1m expire 10s store gpc0,http_req_rate(10s) #http req rate
 stick-table type ip size 100k expire 30s store conn_cur #concurrent connections
 tcp-request connection track-sc1 src
 tcp-request connection reject if { sc1_conn_rate ge 20 }
 tcp-request connection reject if { src_get_gpc0(front) gt 50 }
 tcp-request connection reject if { sc1_conn_cur ge 50 }

backend
 acl abuse src_http_req_rate(front) ge 10
 acl flag_abuser src_inc_gpc0(front)
 tcp-request content reject if abuse flag_abuser
---------------------------------------------------------------------------------------------------------------

#10s 内 tcp 连接建立的数量不能超过 50 个，同时同一 ip 当前已经建立的连接数量不能超过 50 个
#该方式主要用于 4 层负载
frontend
 stick-table type ip size 100k expire 30s store conn_rate(10s) #connection rate
 stick-table type ip size 100k expire 30s store conn_cur #concurrent connections
 tcp-request connection track-sc1 src
 tcp-request connection reject if { sc1_conn_rate ge 50 }  #同一 ip 10S 内建立连接的数量不能超过 50 个
 tcp-request connection reject if { sc1_conn_cur ge 50 }  #同一 ip 当前已经建立连接的数量不能超过 50 个，防止长连接大量堆积的情况
---------------------------------------------------------------------------------------------------------------

#10s 内发送到同一后端的请求数量不能超过 10 个，不管请求是否成功
frontend www
 bind :80
 stick-table  type string  size 1m  expire 10s  store http_req_rate(10s)
 use_backend apiservers if { path_beg /api/ }
 default_backend webservers

backend webservers
```

```
 server s1 192.168.50.20:80
 http-request track-sc0 be_name table www
 http-request deny if { sc_http_req_rate(0) gt 10 }

backend apiservers
 server s1 192.168.50.21:80
 http-request track-sc0 be_name table www
 http-request deny if { sc_http_req_rate(0) gt 10 }
```

```
#通用计数器 gpc 使用示例
#sc-inc-gpc0 负责触发增加计数器, gpc1_rate(10s)控制单位时间内 gpc 增加的数量，gpc 控制通用计数器总量
frontend main
 bind :80
 stick-table  type ip  size 1m  expire 10s  store gpc0,gpc1,gpc0_rate(10s),gpc1_rate(10s)
 http-request track-sc0 src
 http-request sc-inc-gpc0(0) if { req.hdr(Host) example.com }
 http-request sc-inc-gpc1(0) if { url_param(example) test }
 http-request deny if { sc_gpc0_rate(0) gt 10 }   #限制请求频率
 http-request deny if { src_get_gpc0(main) gt 5 } #限制请求总数,main 是 frontend 名称
 tcp-request content reject if { src_get_gpc0 gt 2 } #限制请求频率
 tcp-request connection reject if { src_get_gpc0(main) gt 20 #限制请求频率, 该指令不允许在backend 中使用
```

4：重新加载保留 stick-table 数据

```
#peer  local 中的 local 是 hostname 值, 如果要和 hostname 不一样， 需要 global 部分添加 localpeer local
global
 localpeer local

peers mycluster
 peer local 127.0.0.1:10000

frontend  main
 bind *:5000
   stick-table  type ip  size 100k  expire 24h  store http_req_cnt peers mycluster
   http-request track-sc0 src
```

```
global
 localpeer local

peers mycluster
```

```
 peer local 127.0.0.1:10000  #local 是 hostname,如果要和 hostname 不一样，需要 global 部分添加  localpeer local
 table sticktable1 type ip  size 100k  expire 24h  store http_req_cnt

frontend  main
 bind *:5000
   stick-table  type ip  size 100k  expire 24h  store http_req_cnt peers mycluster
   http-request track-sc0 src table mycluster/sticktable1
```

5：节点之间同步 stick-table 数据

```
peers mycluster
 peer loadbalancer1 192.168.1.10:10000   #loadbalancer1 是 hostname
 peer loadbalancer2 192.168.1.11:10000   #loadbalancer1 是 hostname
 table sticktable1 type ip  size 1m  expire 10s  store http_req_rate(10s)

frontend www
 bind :80
 http-request track-sc0 src table mycluster/sticktable1
 http-request deny if { sc_http_req_rate(0,mycluster/sticktable1) gt 10 }
----------------------------------------------------------------------------------------------------------------

#多 table 示例
peers mycluster
 peer loadbalancer1 192.168.1.10:10000
 peer loadbalancer2 192.168.1.11:10000
 table sticktable1  type ip      size 1m  expire 10s  store http_req_rate(10s)
 table sticktable2  type string  size 1m  expire 10s  store http_req_rate(10s)
 table sticktable3  type string  size 1m  expire 10s  store http_req_rate(10s)

frontend www
 bind :80

 # key is source IP address
 http-request track-sc0 src  table mycluster/sticktable1

 # key is backend name
 http-request track-sc1 be_name table mycluster/sticktable2

 # key is Host header
 http-request track-sc2 req.hdr(Host) mycluster/table sticktable3

 http-request deny if { sc_http_req_rate(0,mysticktable) gt 10 }
----------------------------------------------------------------------------------------------------------------
```

```
backend
 stick-table type ip  size 1m  expire 10s  store http_req_rate(10s) peers mycluster
```

十五、hard-stop-after

```
hard-stop-after <time>
```

设置一个强行清除 soft-stop 动作的超时时间。参数<time>默认为毫秒，指的是当 HAProxy 运行实例收到 SIGUSR1 信号而执行 soft-stop 后的最大超时时间。
设置该设置项可以确保在实例因执行 soft-stop 却被一些长连接（比如 TCP 模式下有一个超长的超时时间设置）阻止退出时依然能退出。TCP 或 HTTP 模式均有效。

十六、DeviceAtlas

十七、作用域

proc: 这个作用域表示变量在整个 HAProxy 进程的生命周期中都是可用的。这意味着从 HAProxy 启动到关闭，这个变量都可以被访问。它适用于那些需要跨多个请求或会话保持状态的高级用例。
sess: 作用于客户端的整个 TCP 会话期间。从连接建立到断开，这个变量都可被使用。这在实现会话保持或跟踪特定会话的状态时非常有用，比如记录会话的开始时间或处理特定会话的计数。
txn: 交易（Transaction）作用域限制了变量的可用性，仅在处理单个 HTTP 请求和响应的完整交互过程中。这意味着从接收到 HTTP 请求到发送完成响应的这段时间内，变量是有效的。这适用于需要在请求处理的开始和结束之间保持状态的情况，如根据请求内容动态调整响应。
req: 只在 HTTP 请求阶段可用，即从 HAProxy 接收到客户端请求开始，到开始处理响应之前。这个作用域的变量非常适合用于基于请求头或请求内容来做出决策，例如路径重写、条件路由等。
res
: 仅在 HTTP 响应阶段可用，即从后端服务器开始发送响应到响应完全构建并准备发送给客户端的阶段。这适用于需要根据响应内容来执行某些操作的场景，比如根据响应状态码来决定是否重试请求或记录特定日志。

十八、邮件配置

```
defaults
# email settings
 email-alert mailers smtp_servers    #邮件服务器配置
 email-alert from 1466873053@qq.com   #发送人邮箱
 email-alert to yixiaogo@hotmail.com
 #发送目标邮箱
 email-alert level notice       #日志级别
 email-alert myhostname ecs-cca1  #hostname

mailers smtp_servers
 mailer smtp1 127.0.0.1:25
```

```
mailer smtp2 xxxx:25 #可以配置多少邮件服务器，启动负载均衡的作用
```

十九、返回后端组状态

主要用于返回后端服务器状态给监控软件使用

```
frontend www
 bind :80
 monitor-uri /checkstatus                    #该页面返回 200 状态码
 monitor fail if { nbsrv(webservers) eq 0 }      #如果后端没有活跃的服务器，则 monitor-uri 配置的地址直接返回 503 状态码
 default_backend webservers

backend webservers
 balance roundrobin
 server server1 10.0.1.3:80 check
 server server2 10.0.1.4:80 check
```

二十、prometheus

主要用于返回后端服务器状态给监控软件使用

```
frontend prometheus
 bind :8405
 mode http
 http-request use-service prometheus-exporter
 no log
---------------------------------------------------------------------------

frontend prometheus
 bind *:8405
 mode http
 http-request use-service prometheus-exporter if { path /metrics }
 no log
```

二十一、auth 认证

```
userlist mycredentials
 user joe   insecure-password joespassword
 user alice insecure-password alicespassword
 user mark  insecure-password markspassword

frontend www
 bind :80
 bind :443 ssl crt /site.pem
 http-request redirect scheme https unless { ssl_fc }
 http-request auth unless { http_auth(mycredentials) }
 default_backend webservers
```

```
---------------------------------------------------------------------------------------------------------
#安装 mkpasswd 命令
$ sudo apt install whois

#通过 sha256 加密 mypassword123，获得如下加密后的密码
$ mkpasswd -m sha-256 mypassword123
$5$s6Subz0X7FSX2zON$r94OtF6gOfWlGmySwvn3pDFIAHbIpe6mWneueqtBOm/

# 上面获取的加密后的密码配置到 haproxy 配置文件中
userlist mycredentials user joe password $5$s6Subz0X7FSX2zON$r94OtF6gOfWlGmySwvn3pDFIAHbIpe6mWneueqtBOm/
```

```
userlist basic-auth-list
 group is-regular-user
 group is-admin



 user admin password $5$gZZsvtRWI$9JIU8pfHLG8BtYW5tceAKD1oNAjjfL5e4LwUfAW1sqA groups is-admin
 user milosz password $1$R29iAdV/$1QUKx8eo6e5pcMIEgaZwt0 groups is-regular-user
 user guest  insecure-password guestpassword

backend app
   http-request auth unless { http_auth(mycredentials) }
   balance     roundrobin
   server  app1 192.168.1.166:80 check
```

二十二、HTTPS 证书

1：创建证书

```
# 1：创建证书目录
$ mkdir /certs && cd /certs

# 2：创建根证书-ca.crt 证书和私钥-privateCA.pem
openssl req \
 -newkey rsa:2048 \
 -nodes \
 -x509 \
 -days 3650 \
 -keyout privateCA.pem \
 -out ca.crt
```

```
# 3：创建 client.csr 和 clientKey.pem，用于证书签名请求
 openssl req \
 -newkey rsa:2048 \
 -nodes \
 -days 3650 \
 -subj "/CN=exampleUser/O=exampleOrganization" \
 -keyout clientKey.pem \
 -out client.csr

# 4：配置 client-cert-extensions.cnf 文件
vim client-cert-extensions.cnf
----------------------------------------
basicConstraints = CA:FALSE
keyUsage = digitalSignature
extendedKeyUsage = clientAuth
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
----------------------------------------

# 5：根据 client.csr、client.crt、ca.crt、privateCA.pem、client-cert-extensions.cnf 生成最终的 client.crt 证书
openssl x509 \
 -req \
 -in client.csr \
 -out client.crt \
 -CA ca.crt \
 -CAkey privateCA.pem \
 -CAcreateserial \
 -days 3650 \
 -extfile client-cert-extensions.cnf

# 6：根据 clientKey.pem  client.crt 生成 site.pem
 cat clientKey.pem  client.crt  > /certs/site.pem
```

tips：简单创建证书和私钥

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout server.key -out server.crt

#生成一个 haproxy 使用的证书文件
cat server.key server.crt > haproxy.pem
```

2：配置 haproxy.cfg

```
frontend  main
```

```
 bind :443 ssl crt site.pem
 use_backend       static

backend static

 balance     roundrobin
 server      static1 192.168.1.251:80 check
 server      static2 192.168.1.225:80 check
 server      static1 192.168.1.166:443 check ssl verify none  #后端如果要配置为 443 端口，需要添加 ssl verify none，默认是发送 http 请求
```

3：验证客户端证书

```
#服务端 haproxy 配置
frontend  main

 bind :443 ssl crt /root/site.pem  verify required ca-file /root/ca.pem  #这里的 ca.pem 是客户端 haproxy site.pem 对应的根证书和根私钥
 use_backend       static
-------------------------------------------------------------------------------------------------------------------------------
-



#客户端 haproxy 配置
global

 ssl-server-verify none

frontend  main
 bind :443 ssl crt /root/site.pem #这里的 site.pem 是客户端证书和私钥
 use_backend       static

backend static
 server   static 192.168.1.208:443 ssl crt /certs/load-balancer-client.crt #这里的 load-balancer-client.crt 是客户端证书和私钥
```

4：crt-list、crt-store

crt-store 需要 HAproxy 3.0 及以上版本支持

```
#crt
frontend www
   bind :443 ssl crt /etc/haproxy/site1.pem
-----------------------------------------------------------------------------

#crt-list
```

```
crt-list.txt
www.example.com /etc/haproxy/certs/www.example.com.pem
example.com /etc/haproxy/certs/example.com.pem
www.example.net /etc/haproxy/certs/www.example.net.pem
example.net /etc/haproxy/certs/example.net.pem

frontend www
 bind :443 ssl crt-list /etc/haproxy/certs/crt-list.tx
 default_backend webservers
-------------------------------------------------------------------------

#crt-store
crt-store web
 crt-base /etc/haproxy/ssl/certs/
 key-base /etc/haproxy/ssl/private/
 load crt "site1.crt"
 load crt "site2.crt" key "site2.key"

frontend main
 bind *:443 ssl crt "@web/site1.crt" crt "@web/site2.crt"
```

在此示例中：

名为 web 的 crt-store 部分指示负载均衡器将使用的 TLS 证书。
crt-base 指令指定用于搜索 TLS 证书文件的目录。在此示例中，文件位于 /etc/haproxy/ssl/certs。
key-base 指令指定用于搜索密钥文件的目录。在此示例中，文件位于 /etc/haproxy/ssl/private。
每个 load 指令都指示一个证书文件和与之相关的选项。
load 指令的 alias 参数允许我们稍后使用该名称引用此证书。

通过在 bind 指令的 crt 参数按名称引用此 crt-store 定义，在前端中使用该定义。
名称的格式为 @<crt-store name>/<certificate name 或 alias>，或者在本例中为 @web/site2.crt

5：设置最低 TLS 版本

```
global
 ssl-default-bind-options ssl-min-ver TLSv1.2  #仅允许 TLS 1.2 或更高版本
```

6：默认证书配置

HAProxy 3.0 及以上版本支持

```
frontend  main
```

```
 bind :443 ssl default-crt /certs/default.pem.ecdsa /certs/default.pem.rsa crt /certs/  #配置默认证书
 #我们将默认证书指定为 /certs/default.pem.ecdsa 和 /certs/default.pem.rsa。
 负载均衡器将首先尝试使用 ECDSA 证书，如果客户端不支持，则使用 RSA 证书。
 我们将证书的目录设置为 /certs/。负载均衡器将根据 SNI 从此目录中选择适当的证书。
 如果找不到匹配的证书，它将使用默认证书。
 请注意，仅当您没有使用 strict-sni 选项时，负载均衡器才会使用默认证书。

 use_backend       static

backend static

 balance     roundrobin
 server      static1 192.168.1.251:80 check
 server      static2 192.168.1.225:80 check
 server      static1 192.168.1.166:443 check ssl verify none  #后端如果要配置为 443 端口，需要添加 ssl verify none，默认是发送 http 请求



设置默认证书的其他方法，负载均衡器将使用目录中的第一个证书。如果要以这种方式加载默认证书，请确保第一个证书文件是多证书捆绑包（PEM 格式）。
如果您使用的是 crt-list，请用星号标记您的证书，以指示负载均衡器应将其视为默认证书。例如：
default.pem.rsa *
default.pem.ecdsa *
```

7：重定向 HTTP to HTTPS

```
# 如果是 http 协议则重定向到 https 协议
frontend www
 mode http
 bind :80
 bind :443 ssl crt /certs/site.pem
 http-request redirect scheme https unless { ssl_fc }
 default_backend webservers
```

8：ssl 证书域名验证

```
backend webservers
   server web1 10.0.0.5:443 ssl verify required ca-file /myca.pem sni req.hdr(Host)
   server web2 10.0.0.6:443 ssl verify required ca-file /myca.pem sni req.hdr(Host)


--------------------------------------------------------------------------------
web1 和 web2：这两个服务器分别位于 10.0.0.5:443 和 10.0.0.6:443 上。
ssl verify required：要求验证服务器提供的 SSL 证书的有效性。
```

ca-file /myca.pem：指定 CA 证书文件的位置，用于验证服务器证书。
sni req.hdr(Host)：使用客户端请求中的 Host 头作为 SNI 字段的内容。

sni req.hdr(Host) 的作用是从客户端请求的 Host 头中提取服务器名称，并将其作为 SNI 字段传递给后端服务器。
这种做法对于处理多域名、多证书的场景特别有用，能够确保每个域名都能获得正确的 SSL 证书，从而提升整体的安全性和性能。

9：防止中间人攻击

```
http-response set-header Strict-Transport-Security
http-response set-header Strict-Transport-Security "max-age=16000000; includeSubDomains; preload;"
#
```

max-age 必需。设置在用户至少访问过一次网站后，浏览器应记住规则的时间（以秒为单位）。下次用户客户端访问您的域时，将再次缓存 HSTS 策略。
includeSubDomains 可选。告知浏览器它应该在规则中包含您的所有子域。
preload 可选。将您的网站提交到 HSTS 预加载服务，该服务是浏览器将使用 HTTPS 自动连接到的网站。preload 选项需要 includeSubDomains。
启用预载意味着您可以向特定的服务提交自己的网站地址，以便将其加入到各大主流浏览器内置的信任库中去
这样一来，即便是在首次访问某个新安装或者从未见过的网站时，只要它是已知的预装载名单上的成员之一，浏览器就会直接默认使用 HTTPS 与其建立联系，无需等待接收到来自服务器的第一条消息后再做出判断切换至加密通道的操作。
需要注意的是，要使 preloaded 生效，首先必须已经启用了 includeSubDomains；其次还需要满足一系列条件才能成功申请进入预加载清单，比如整个网站都需要完全符合严格的 SSL/TLS 配置要求等等。

10：OCSP stapling

HAProxy 3 及以上版本才支持

```
global
 ocsp-update.mode on
 ocsp-update.mindelay 300  #同一 CSP 响应的自动更新之间的最小间隔
 ocsp-update.maxdelay 3600  #同一 CSP 响应的自动更新之间的最大间隔

crt-store web
 crt-base /etc/hapee-3.0/certs/
 load crt mysite.pem ocsp-update on ocsp mysite.ocsp
```

保存 OCSP 响应到文件中

```
#执行命令
$ openssl ocsp \
 -issuer issuer.pem \
 -cert mysite.pem \
 -url http://ocsp.issuer.com \
```

```
 -host ocsp.issuer.com:80 \
 -respout mysite.ocsp

#修改 crt-list.txt 文件
/etc/hapee-3.0/certs/mysite.pem [alpn h2 ocsp-update on ocsp /etc/hapee-3.0/certs/mysite.ocsp]
```

运行一个 OCSP 响应服务器

1、我们依赖于 openssl 命令行，要检查其是否已安装，请运行：

```
openssl version
output：
OpenSSL 3.0.11 19 Sep 2023 (Library: OpenSSL 3.0.11 19 Sep 2023)
```

2、此服务器将充当 CA，生成 TLS 证书、私钥、CRL 和 CSR 以进行测试。创建用于存储这些不同类型文件的目录结构：

```
sudo mkdir -p /exampleCA/rootCA/{certs,newcerts,crl,private,csr}
echo 1000 | sudo tee /exampleCA/rootCA/serial
echo 0100 | sudo tee /exampleCA/rootCA/crlnumber
sudo touch /exampleCA/rootCA/index.txt
sudo touch /exampleCA/openssl-root.cnf
```

3、以 root 身份编辑文件

编辑 /exampleCA/openssl-root.cnf 并添加以下内容，该内容定义了我们需要的 OpenSSL 设置。

更改行 authorityInfoAccess = OCSP;URI：http：//192.168.56.39：8888 使用 OCSP 响应程序服务器的 IP 地址。这将嵌入到每个 TLS 服务器证书中，作为进行 OCSP 查询的调用地址。

```
openssl-root.cnf
[ ca ]                                  # The default CA section
default_ca = CA_default                         # The default CA name
[ CA_default ]                                # Default settings for the CA
dir             = /exampleCA/rootCA             # CA directory
certs           = $dir/certs                  # Certificates directory
crl_dir         = $dir/crl                    # CRL directory
new_certs_dir     = $dir/newcerts                 # New certificates directory
database        = $dir/index.txt                # Certificate index file
serial          = $dir/serial                 # Serial number file
RANDFILE        = $dir/private/.rand              # Random number file
private_key      = $dir/private/rootCA.key          # Root CA private key
certificate      = $dir/certs/rootCA.crt           # Root CA
certificatecrl            = $dir/crl/crl.pem              # Root CA CRL
crlnumber       = $dir/crlnumber                # Root CA CRL
numbercrl_extensions    = crl_ext                     # CRL
extensionsdefault_crl_days  = 30                        # Default CRL validity days
default_md       = sha256                     # Default message digest
preserve        = no                         # Preserve existing extensions
email_in_dn      = no                          # Exclude email from the DN
name_opt        = ca_default                    # Formatting options for names
```

```
cert_opt        = ca_default                    # Certificate output options
policy          = policy_strict                 # Certificate policy
unique_subject    = no                          # Allow multiple certs with the same DN

[ policy_strict ]                               # Policy for stricter validation
countryName           = match                   # Must match the issuer's country
stateOrProvinceName     = match                 # Must match the issuer's state
organizationName       = match                  # Must match the issuer's organization
organizationalUnitName  = optional              # Organizational unit is optional
commonName            = supplied                # Must provide a common name
emailAddress          = optional                # Email address is optional

[ req ]                                         # Request settings
default_bits       = 2048                       # Default key size
distinguished_name  = req_distinguished_name    # Default DN template
string_mask        = utf8only                   # UTF-8 encoding
default_md         = sha256                     # Default message digest
prompt            = no                          # Non-interactive mode

[ req_distinguished_name ]                      # Template for the DN in the CSR
countryName                 = Country Name (2 letter code)
stateOrProvinceName           = State or Province Name (full name)
localityName                = Locality Name (city)
0.organizationName            = Organization Name (company)
organizationalUnitName        = Organizational Unit Name (section)
commonName                  = Common Name (your domain)
emailAddress                = Email Address

[ v3_ca ]                                       # Root CA certificate exten
sionssubjectKeyIdentifier = hash                # Subject key identifier
authorityKeyIdentifier = keyid:always,issuer    # Authority key identifier
basicConstraints = critical, CA:true            # Basic constraints for a CA
keyUsage = critical, keyCertSign, cRLSign       # Key usage for a CA

[ crl_ext ]                                     # CRL extensions
authorityKeyIdentifier = keyid:always,issuer    # Authority key identifier

[ v3_intermediate_ca ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ v3_OCSP ]
```

```
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = OCSPSigning

[ server_cert ]
basicConstraints = CA:false
nsCertType = server
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
authorityInfoAccess = OCSP;URI:http://192.168.56.39:8888
```

4、为 CA 生成根证书和私有密钥：

```
sudo openssl genrsa -out /exampleCA/rootCA/private/rootCA.key 4096
sudo openssl req \
-config /exampleCA/openssl-root.cnf \
-extensions v3_ca \
-new \
-x509 \
-days 7300 \
-sha256 \
-subj "/C=US/ST=Ohio/L=Columbus/O=Example Corp/OU=IT Department/CN=Root CA" \
-key /exampleCA/rootCA/private/rootCA.key \
-out /exampleCA/rootCA/certs/rootCA.crt
```

5、将此根证书添加到服务器的 CA 证书列表中，使其受信任。

```
sudo mkdir /usr/local/share/ca-certificates/extra
sudo cp /exampleCA/rootCA/certs/rootCA.crt /usr/local/share/ca-certificates/extra/
sudo update-ca-certificates
output：
Updating certificates in /etc/ssl/certs...1 added, 0 removed; done.Running hooks in /etc/ca-certificates/update.d...done.
```

6、创建用于对 OCSP 响应进行签名的私钥和 CSR。这将提示您输入密码。

```
sudo openssl req \
-config /exampleCA/openssl-root.cnf \
-new \
-sha256 \
-subj "/C=US/ST=Ohio/L=Columbus/O=Example Corp/OU=IT Department/CN=ocsp.example.com" \
-keyout /exampleCA/rootCA/private/ocsp.example.com.key \
-out /exampleCA/rootCA/csr/ocsp.example.com.csr
```

7、生成名为 ocsp.example.com.crt 从 CSR 中获取。这将提示您对证书进行签名并将其提交到证书数据库。

```
sudo openssl ca \
-config /exampleCA/openssl-root.cnf \
```

```
-extensions v3_OCSP \
-days 375 \
-notext \
-md sha256 \
-in /exampleCA/rootCA/csr/ocsp.example.com.csr \
-out /exampleCA/rootCA/certs/ocsp.example.com.crt
```

output：

```
Sign the certificate? [y/n]:y1 out of 1 certificate requests certified, commit? [y/n]yWrite out database with 1 new entriesDatabase updated
```

8、启动 OCSP 服务器。这将提示您输入密码。

```
sudo openssl ocsp \
-port 8888 \
-index /exampleCA/rootCA/index.txt \
-rsigner /exampleCA/rootCA/certs/ocsp.example.com.crt \
-rkey /exampleCA/rootCA/private/ocsp.example.com.key \
-CA /exampleCA/rootCA/certs/rootCA.crt \
-text \  -nmin 60
```

output：

```
ACCEPT 0.0.0.0:8888 PID=3241Enter pass phrase for /exampleCA/rootCA/private/ocsp.example.com.key:ocsp: waiting for OCSP client connections...
```

9、另一个终端窗口中，创建一个名为 www.example.com.crt 用于负载均衡器。这将提示您对证书进行签名并将其提交到证书数据库。

```
sudo openssl req \
-config /exampleCA/openssl-root.cnf \
-newkey rsa:2048 \
-nodes \
-subj "/C=US/ST=Ohio/L=Columbus/O=Example Corp/OU=IT Department/CN=www.example.com" \
-keyout /exampleCA/rootCA/private/www.example.com.key \
-out /exampleCA/rootCA/csr/www.example.com.csr

sudo openssl ca \
-config /exampleCA/openssl-root.cnf \
-extensions server_cert \
-days 3650 \
-notext \
-md sha256 \
-in /exampleCA/rootCA/csr/www.example.com.csr \
-out /exampleCA/rootCA/certs/www.example.com.crt
```

output：

```
Sign the certificate? [y/n]:y1 out of 1 certificate requests certified, commit? [y/n]yWrite out database with 1 new entriesDatabase updated
```

10、此时 1.At，您可以使用此证书在其侦听 IP 地址查询 OCSP 响应程序服务器。它应该返回一个 good status。

```
openssl ocsp \
-issuer /exampleCA/rootCA/certs/rootCA.crt \
-cert /exampleCA/rootCA/certs/www.example.com.crt \
-url http://192.168.56.39:8888 \  -resp_text
```

output：

OCSP Response Data: OCSP Response Status: successful (0x0) Response Type: Basic OCSP Response Version: 1 (0x0) Responder Id: C = US, ST = Ohio, O = Example Corp, OU = IT Department, CN = ocsp.example.com Produced At: Dec 17 15:07:20 2024 GMT Responses: Certificate ID: Hash Algorithm: sha1 Issuer Name Hash: FB1F13EE9B62FF5CE28500BCCB28584007D61524 Issuer Key Hash: 200DF0756FA74CAFB9421A42A6C1C965532FC369 Serial Number: 1001 Cert Status: good This Update: Dec 17 15:07:20 2024 GMT Next Update: Dec 17 15:12:20 2024 GMT

11、复制 www.example.com.crt 证书、www.example.com.key 私有密钥和 CA 根证书合并到单个 PEM 文件中，以便在 QA 负载均衡器上使用。

```
sudo cat /exampleCA/rootCA/certs/www.example.com.crt > ~/www.example.com.pem
sudo cat /exampleCA/rootCA/private/www.example.com.key >> ~/www.example.com.pem
sudo cat /exampleCA/rootCA/certs/rootCA.crt >> ~/www.example.com.pem
```

12、将这些文件复制到 QA 负载均衡器上的主目录：

~/www.example.com.pem

/exampleCA/rootCA/certs/rootCA.crt

13、在 QA 负载均衡器上，将根证书添加到服务器的 CA 证书列表中，使其受信任。

```
sudo mkdir /usr/local/share/ca-certificates/extrasudo cp ~/rootCA.crt /usr/local/share/ca-certificates/extra/sudo update-ca-certificates
```

output：

```
Updating certificates in /etc/ssl/certs...1 added, 0 removed; done.Running hooks in /etc/ca-certificates/update.d...done.
```

14、复制 www.example.com.pem 文件复制到 certs 目录。例如：

```
sudo mkdir -p /etc/haproxy/certs
sudo cp ~/www.example.com.pem /etc/haproxy/certs/
```

15、更新您的 crt-list 文件使用此证书并启用 OCSP 装订。将此目录作为 crt-list.txt 保存到 certs 目录。

```
crt-list.txt/etc/haproxy/certs/www.example.com.pem [ocsp-update on]
```

16、更新您的前端以使用 crt-list 中。

```
frontend www
 bind :443 ssl crt-list /etc/haproxy/certs/crt-list.txt
 default_backend webservers
```

17、重新加载负载均衡器配置。

```
sudo systemctl reload haproxy
```

18、此时 1.At，您的负载均衡器日志应显示 OCSP 更新已成功完成。

检查日志文件 /var/log/haproxy.log 中是否有 OCSP 更新消息。

```
haproxy.log
<OCSP-UPDATE> /etc/haproxy/certs/www.example.com.pem 1 "Update successful" 0 1
```

19、To 映射域 www.example.com 负载均衡器的 IP 地址，编辑服务器的 /etc/hosts 文件：

```
hosts
```

```
127.0.0.1 www.example.com
```

20、测试负载均衡器是否正在装订 OCSP 响应。输出应显示 OCSP 响应数据。

```
openssl s_client -connect www.example.com:443 -servername www.example.com -status </dev/null
```

output：

```
CONNECTED(00000003)depth=1 C = US, ST = Ohio, L = Columbus, O = Example Corp, OU = IT Department, CN = Root CAverify error:num=19:self-signed certificate in certificate chainverify return:1depth=1 C = US, ST = Ohio, L = Columbus, O = Example Corp, OU = IT Department, CN = Root CAverify return:1depth=0 C = US, ST = Ohio, O = Example Corp, OU = IT Department, CN = www.example.comverify return:1OCSP response: ==========OCSP Response Data: OCSP Response Status: successful (0x0) Response Type: Basic OCSP Response Version: 1 (0x0) Responder Id: C = US, ST = Ohio, O = Example Corp, OU = IT Department, CN = ocsp.example.com Produced At: Dec 18 15:37:31 2024 GMT Responses: Certificate ID: Hash Algorithm: sha1 Issuer Name Hash: FB1F13EE9B62FF5CE28500BCCB28584007D61524 Issuer Key Hash: 200DF0756FA74CAFB9421A42A6C1C965532FC369 Serial Number: 1001 Cert Status: good This Update: Dec 18 15:37:31 2024 GMT Next Update: Dec 18 16:37:31 2024 GMT
```

二十三、客户端 ip 保持

1：TCP 负载 IP 透传

负载均衡 haproxy 配置

```
listen web_prot_http_nodes
bind 192.168.7.101:80
mode tcp
balance roundrobin
server web1 blogs.studylinux.net:80 send-proxy check inter 3000 fall 3 rise 5
```

后端服务器 Nginx 配置

```
server {
listen 80 proxy_protocol;
server_name blogs.studylinux.net;
```

PROXY 协议允许 Nginx 和 Nginx Plus 接受来自代理服务器和负载平衡器的客户端连接信息，比如 HAproxy 和 Amazon Elastic Load Balancer (ELB) 。

通过 PROXY 协议，Nginx 可以从 HTTP，SSL，HTTP / 2，SPDY，WebSocket 和 TCP 中获取到源 IP 地址。获取到客户端的源 IP 地址，可以在为网页指定语言、设置 IP 黑名单或只是简单的日志和统计分析。

通过 PROXY 协议传输的数据是客户端的 IP 地址、代理服务器的 IP 地址和所有的端口号。

2：HTTP 负载 IP 透传

负载均衡 haproxy 配置

```
defaults
option forwardfor
```

```
或者：
option forwardfor header X-Forwarded-xxx
#option forwardfor 则后端服务器 web 格式为 X-Forwarded-For
#自定义传递 IP 参数,后端 web 服务器写 X-Forwarded-xxx，
```

3：web 服务器日志格式配置

```
apache
LogFormat "%{X-Forwarded-For}i %a %l %u %t "%r" %>s %b "%{Referer}i" "%{User-Agent}i"" combined

tomcat
pattern='%{X-Forwarded-For}i %l %T %t "%r" %s %b "%{User-Agent}i"'/>

nginx
log_format main '"$http_x_forwarded_For" - $remote_user [$time_local] "$request"
```

二十四、DNS 解析

1：使用域名进行后端转发

```
resolvers mynameservers
 nameserver ns1 192.168.2.10:53
 nameserver ns2 192.168.3.10:53

 # Maximum size of a DNS answer allowed, in bytes
 accepted_payload_size 512

 # Whether to add nameservers found in /etc/resolv.conf
 parse-resolv-conf

 # How long to "hold" a backend server's up/down status depending on the name resolution status.
 # For example, if an NXDOMAIN response is returned, keep the backend server in its current state (up) for
 # at least another 30 seconds before marking it as down due to DNS not having a record for it.
 hold valid    10s
 hold other    30s
 hold refused  30s
 hold nx       30s
 hold timeout  30s
 hold obsolete 30s

 # How many times to retry a query
 resolve_retries 3
```

```
 # How long to wait between retries when no valid response has been received
 timeout retry 1s

 # How long to wait for a successful resolution
 timeout resolve 1s

backend webservers
 server s1 hostname1.example.com:80   check  resolvers mynameservers
 server s2 hostname2.example.com:8080 check  resolvers mynameservers
 server s3 hostname3.example.com:8080 check  resolvers mynameservers
 server s1 hostname1.example.com:80 check resolvers mynameservers init-addr last,libc,none  #域名解析失败不会影响 HAproxy 启动
```

2：DNS 服务发现

```
#DNS 自动发现
resolvers
 mydns  nameserver dns1 192.168.50.30:53
 accepted_payload_size 8192

backend webservers
 balance roundrobin
 server-template web 5 myservice.example.local:80 check resolvers mydns init-addr none
--------------------------------------------------------------------------------------------------------------------------------
----

#DNS 服务自动发现
resolvers
 mydns  nameserver dns1 192.168.50.30:53  accepted_payload_size 8192

backend webservers
 balance roundrobin
 server-template web 5 _myservice._tcp.example.local resolvers mydns check init-addr none
```

Tips：SRV DNS 返回记录/ nbvcx`

```
_service is the standard network service name (taken from /etc/services) or a port number
_proto is the standard protocol name (tcp or udp)
name is the name of the service, i.e. the name used in the query
TTL is the validity period for the response (the load balancer ignores this field because it maintains its own expiry data defined in the configuration)
class is the DNS class (IN)
SRV is the DNS record type (SRV)
priority is the priority of the target host. Lower value = higher preference (the load balancer ignores this field but may use it later to indicate active / backup state)
```

weight is the relative weight in case of records with the same priority. Higher number = higher preference
port is the port where the service is configured
target is the hostname of the machine providing the service, ending in a dot

二十五、Load balancing

1：FastCGI

haproxy 配置：

```
fcgi-app php-fpm
 log-stderr global
 option keep-conn
 docroot /www
 index index.php
 path-info ^(/.+\.php)(/.*)?$

backend app
 mode http
 filter fcgi-app php-fpm
 use-fcgi-app php-fpm
 balance     roundrobin

 server      static1 192.168.1.166:9000 proto fcgi
```

修改后端 php-fpm 配置文件：

```
listen = 0.0.0.0:9000

同时注释掉 listen.allowed_clients = 127.0.0.1 这行
```

2：GRPC

```
frontend grpc_service
 mode http
 bind :3001 proto h2
 default_backend grpc_servers

backend grpc_servers
 mode http
 balance roundrobin
 server s1 192.168.0.10:3000 check proto h2
 server s2 192.168.0.10:3000 check proto h2
```

3：HTTP2

```
backend servers
  mode http
  server s1 192.168.0.10:443 ssl alpn h2,http/1.1
```

```
 server s2 192.168.0.11:443 ssl alpn h2,http/1.1   #如果客户端和服务端都支持 http2 则用HTTP2，否则降级到 http1.1
```

二十六、HTTP3

HAProxy Enterprise 2.8r1 以上版本支持 http3

```
frontend example
 bind :80

 # Enable HTTPS
 bind :443  ssl crt ssl.pem

 # enables HTTP/3 over QUIC
 bind quic4@:443 ssl crt ssl.pem alpn h3

 # Redirects to HTTPS
 http-request redirect scheme https unless { ssl_fc }

 # 'Alt-Svc' header invites client to switch to the QUIC protocol
 # Max age (ma) is set to 15 minutes (900 seconds), but
 # can be increased once verified working as expected
 http-response set-header alt-svc "h3=\":443\";ma=900;"

 default_backend webservers
```

二十七、压缩

```
backend web_servers
 filter compression
 compression algo gzip                                                    #压缩算法
 compression type text/css text/html text/javascript application/javascript text/plain text/xml application/json  #压缩的类型
 compression offload                                                      #压缩交由 haproxy 统一处理，后端不处理
```

二十八、启动时执行命令

```
program echo
 command echo "Hello, World!"      #command 指定执行的 shell 命令
 user hapee-lb
 group hapee
 no option start-on-reload         #reload haproxy 时不执行 command 命令
```

二十九、Syslog 远程服务器负载

HAProxy 2.3 及以上版本支持

1：转发其他网络设备的 Syslog 日志

```
log-forward syslog
 # 监听 udp514 端口等待接收 udp 日志
 dgram-bind 0.0.0.0:514

 # 监听 tcp514 端口等待接收 tcp 日志
 bind 0.0.0.0:514

 # 通过 UDP 转发 Syslog 消息
 log <your_server_ip_address>:514 local0

 # 通过 TCP 转发 Syslog 消息
 log ring@logbuffer local0

 # 将传入消息转换为标准化的 Syslog 协议
 log ring@logbuffer format rfc5424 local0

ring logbuffer
 description "buffer for logs"
 format rfc5424
 maxlen 1500
 size 65536
 timeout connect 10s
 timeout server 20s

 # 通过 TCP 往外发消息
 server logserver <your_syslog_server_ip_address>:514
```

2：发送 HAproxy 日志

```
global
 #log 192.168.1.100 local0 UDP 协议发送
 log ring@logbuffer local0  #TCP 协议发送

defaults
 log global

ring logbuffer
 description "buffer for logs"
 format rfc5424
 maxlen 1500
 size 65536
 timeout connect 10s
 timeout server 20s

 # Sends outgoing messages via TCP
```

```
server logserver <your_syslog_server_ip_address>:514
```