低版本不支持此方法

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

限制总请求数

针对的对象是单个 ip
这个限制是 7 层的，也就是已经完成 3 次握手后，发送的 http 请求数量

```
frontend main
   bind *:5000
   stick-table  type ip  size 100k  expire 24h  store http_req_cnt
   http-request track-sc0 src
   http-request deny deny_status 429 if { sc_http_req_cnt(0) gt 1 }
```

限制指定时间内请求频率

针对的对象是单个 ip
这个限制是 7 层的，也就是已经完成 3 次握手后，发送的 http 请求数量

```
frontend main
   bind *:5000
   stick-table  type ip  size 100k  expire 24h  store http_req_rate(10s)
   http-request track-sc0 src
   http-request deny deny_status 429 if { sc_http_req_rate(0) gt 1 }
```

限制指定 ip 的带宽大小

HAProxy 2.7 以上才支持此功能

```
frontend myfrontend
 mode http
 bind :80
 filter bwlim-out mylimit default-limit 625000 default-period 1s

frontend myfrontend
 mode http
 bind :80
 filter bwlim-out mylimit default-limit 625000 default-period 1s
 http-response set-bandwidth-limit mylimit

frontend
 myfrontend
 mode http
 bind :80
 filter bwlim-in mylimit default-limit 625000 default-period 1s
```

```
frontend
 myfrontend
 mode http
 bind :80
 filter bwlim-in mylimit default-limit 625000 default-period 1s
 http-request set-bandwidth-limit mylimit
```

```
peers mypeers
 peer hapee 127.0.0.1:10000
 table downloadrate type integer size 1m expire 3600s store bytes_out_rate(1s)
 table uploadrate type integer size 1m expire 3600s store bytes_in_rate(1s)

frontend  fe_main
 bind :80
 filter bwlim-out mydownloadlimit limit 625000 key src table mypeers/downloadrate
 filter bwlim-in myuploadlimit limit 625000 key src table mypeers/uploadrate
 http-response set-bandwidth-limit mydownloadlimit
 http-request set-bandwidth-limit myuploadlimit

backend webservers
 server web1 192.168.56.6:80 check maxconn 30
 server web2 192.168.56.7:80 check maxconn 30
 filter bwlim-out mydownloadlimit limit 625000 key be_id table mypeers/downloadrate
 filter bwlim-in myuploadlimit limit 625000 key be_id table mypeers/uploadrate
 http-response set-bandwidth-limit mydownloadlimit
 http-request set-bandwidth-limit myuploadlimit
```

指定 URL 限流

1、创建文件 rates.map，并添加如下值：

```
/urla  10
/urlb  20
/urlc  30
```

2、更新 frontend 配置以包括 stick-table 和 http-request track 指令，如下所示：

```
frontend website
 bind :80
 stick-table  type binary  len 20  size 100k  expire 10s  store http_req_rate(10s)
 http-request track-sc0 base32+src
 http-request set-var(req.rate_limit)  path,map_beg(/rates.map,20)
 http-request set-var(req.request_rate)  base32+src,table_http_req_rate()
 acl rate_abuse var(req.rate_limit),sub(req.request_rate) lt 0
 http-request deny deny_status 429 if rate_abuse
 default_backend servers
```

通过请求参数限流

请求示例：http://yourwebsite.com/api/v1/does_a_thing?token=abcd1234

```
frontend website
 bind :80
 stick-table type string size 100k expire 24h store http_req_rate(24h)
 acl has_token url_param(token) -m found
 acl exceeds_limit url_param(token),table_http_req_rate() gt 1000
 http-request track-sc0 url_param(token) unless exceeds_limit
 http-request deny deny_status 429 if !has_token or exceeds_limit
```

熔断

1：通过 observe 指定熔断

```
#http 请求错误次数超过多少次后自动下线，并在连续检测多少次成功后重新上线服务器
backend myservice
 default-server maxconn 30  check  observe layer7  error-limit 50  on-error mark-down  inter 1s rise 30  slowstart 20s
 # maxconn 是最大连接数，可以防止 ddos 攻击，error-limit 是错误 http 请求次数，rise 是连续检查健康指定次数后上线服务器，slowstart 是慢启动，在指定时间内逐渐恢复流量
 server s1 192.168.0.10:80
 server s2 192.168.0.11:80
```

2：通过 stick-table 表熔断

```
# 如果 10S 内 HTTP 5xx 错误的数量占比总请求数量 比率 超过 50%，则拒绝新请求
backend myservice
 #定义一个粘性表，存储请求速率、grpc0 计数、grpc0 计数速率、grpc1 计数
 stick-table type string  size 1  expire 30s  store http_req_rate(10s),gpc0,gpc0_rate(10s),gpc1

 # Is the circuit open (no traffic can flow)?
 acl circuit_open be_name,table_gpc1 gt 0

 # Reject request if circuit is open
 http-request deny deny_status 503 if circuit_open

 # Begin tracking requests
 http-request track-sc0 be_name

 # Count HTTP 5xx server errors
 http-response sc-inc-gpc0(0) if { status ge 500 }

 # Store the HTTP request rate and error rate in variables
 http-response set-var(res.req_rate) sc_http_req_rate(0)
 http-response set-var(res.err_rate) sc_gpc0_rate(0)
```

```
 # Check if error rate is greater than 50% using some math
 http-response sc-inc-gpc1(0) if { int(100),mul(res.err_rate),div(res.req_rate) gt 50 }

 server s1 192.168.0.10:80 check
 server s2 192.168.0.11:80 check
```

过载保护

```
frontend website
 maxconn 20000
 bind :80
 default_backend web_servers

frontend database
 maxconn 20000
 bind :3306
 default_backend database_servers

frontend api
 maxconn 20000
 bind :8080
 default_backend api_servers
---------------------------------------------------------------------

backend web_servers
 balance roundrobin
 default-server maxconn 30
 server s1 192.168.0.10:80
 server s2 192.168.0.11:80
 server s3 192.168.0.12:80
---------------------------------------------------------------------

backend web_servers
 balance roundrobin
 timeout queue 30s
 server s1 192.168.0.10:80 maxconn 30
 server s2 192.168.0.11:80 maxconn 40
 server s3 192.168.0.12:80 maxconn 50
---------------------------------------------------------------------

#指定优先级，数字越小优先级越高，优先级高的会优先处理，会排在队列的更靠前位置
backend web_servers
 balance roundrobin
 acl is_checkout path_beg /checkout/
```

```
 http-request set-priority-class int(1) if is_checkout
 http-request set-priority-class int(2) if !is_checkout
 timeout queue 30s
 server s1 192.168.0.10:80 maxconn 30
 server s2 192.168.0.11:80 maxconn 30
 server s3 192.168.0.12:80 maxconn 30
```

封禁 IP

```
frontend  main
 bind :80
 acl block_test src 78.153.0.0/16
 http-request tarpit if block_test   #阻塞请求，使客户端等待直到超时，可以用 timeout tarpit 10s 设置阻塞时间
 http-request tarpit deny_status 429 if block_test
 #http-request reject if block_test  #拒绝且不发送任何响应
 #http-request deny if block_test    #拒绝并返回 HTTP 403 Forbidden 错误码
```

```
--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。在创建会话并初始化 HTTP 解析器后关闭连接而不响应。
--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。
#您可以静默丢弃客户端的 HTTP 请求，该请求会立即断开连接，而不会通知客户端连接已关闭。
这意味着负载均衡器会释放用于此连接的所有资源。客户端通常需要超时才能释放其连接端。
请注意，静默丢弃会影响负载均衡器和客户端之间的任何有状态防火墙或代理，因为它们通常会保留连接，而不知道它已断开连接。
--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 connection reject
#在创建会话之前，最早的时间点关闭连接而不响应。这些请求不会显示在您的日志中。
拒绝响应会立即关闭连接，而不发送响应。客户端的浏览器将显示错误消息 The connection was reset。使用 tcp-request content reject 或 tcp-request connection reject 来丢弃您知道不想连接的连接
--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，这些请求仍会显示在您的日志中。
--tcp-request content silent-drop      创建会话后丢弃数据包，不发送任何响应。
```