A+

HAProxy用法详解 全网最详细中文文档

发表评论

一、HAProxy简介

（1）HAProxy 是一款提供高可用性、负载均衡以及基于TCP（第四层）和HTTP（第七层）应用的代理软件，支持虚拟主机，它是免费、快速并且可靠的一种解决方案。 HAProxy特别适用于那些负载特大的web站点，这些站点通常又需要会话保持或七层处理。HAProxy运行在时下的硬件上，完全可以支持数以万计的 并发连接。并且它的运行模式使得它可以很简单安全的整合进您当前的架构中， 同时可以保护你的web服务器不被暴露到网络上。

（2）HAProxy 实现了一种事件驱动、单一进程模型，此模型支持非常大的并发连接数。多进程或多线程模型受内存限制 、系统调度器限制以及无处不在的锁限制，很少能处理数千并发连接。事件驱动模型因为在有更好的资源和时间管理的用户端(User-Space) 实现所有这些任务，所以没有这些问题。此模型的弊端是，在多核系统上，这些程序通常扩展性较差。这就是为什么他们必须进行优化以 使每个CPU时间片(Cycle)做更多的工作。

（3）HAProxy 支持连接拒绝：因为维护一个连接的打开的开销是很低的，有时我们很需要限制攻击蠕虫（attack bots），也就是说限制它们的连接打开从而限制它们的危害。 这个已经为一个陷于小型DDoS攻击的网站开发了而且已经拯救

了很多站点，这个优点也是其它负载均衡器没有的。

（4）HAProxy 支持全透明代理（已具备硬件防火墙的典型特点）：可以用客户端IP地址或者任何其他地址来连接后端服务器. 这个特性仅在Linux 2.4/2.6内核打了cttproxy补丁后才可以使用. 这个特性也使得为某特殊服务器处理部分流量同时又不修改服务器的地址成为可能。

性能

HAProxy借助于OS上几种常见的技术来实现性能的最大化。

1，单进程、事件驱动模型显著降低了上下文切换的开销及内存占用。

2，O(1)事件检查器(event checker)允许其在高并发连接中对任何连接的任何事件实现即时探测。

3，在任何可用的情况下，单缓冲(single buffering)机制能以不复制任何数据的方式完成读写操作，这会节约大量的CPU时钟周期及内存带宽；

4，借助于Linux 2.6 (>= 2.6.27.19)上的splice()系统调用，HAProxy可以实现零复制转发(Zero-copy forwarding)，在Linux 3.5及以上的OS中还可以实现零复制启动(zero-starting)；

5，内存分配器在固定大小的内存池中可实现即时内存分配，这能够显著减少创建一个会话的时长；

6，树型存储：侧重于使用作者多年前开发的弹性二叉树，实现了以O(log(N))的低开销来保持计时器命令、保持运行队列命令及管理轮询及最少连接队列；

7，优化的HTTP首部分析：优化的首部分析功能避免了在HTTP首部分析过程中重读任何内存区域；

8，精心地降低了昂贵的系统调用，大部分工作都在用户空间完成，如时间读取、缓冲聚合及文件描述符的启用和禁用等；

所有的这些细微之处的优化实现了在中等规模负载之上依然有着相当低的CPU负载，甚至于在非常高的负载场景中，5%的用户空间占用率和95%的系统空间占用率也是非常普遍的现象，这意味着HAProxy进程消耗比系统空间消耗低20倍以上。因此，对OS进行性能调优是非常重要的。即使用户空间的占用率提高一倍，其CPU占用率也仅为10%，这也解释了为何7层处理对性能影响有限这一现象。由此，在高端系统上HAProxy的7层性能可轻易超过硬件负载均衡设备。

在生产环境中，在7层处理上使用HAProxy作为昂贵的高端硬件负载均衡设备故障故障时的紧急解决方案也时长可见。硬件负载均衡设备在“报文”级别处理请求，这在支持跨报文请求(request across multiple packets)有着较高的难度，并且它们不缓冲任何数据，因此有着较长的响应时间。对应地，软件负载均衡设备使用TCP缓冲，可建立极长的请求，且有着较大的响应时间。

HAProxy目前主要有三个版本： 1.3 ， 1.4 ，1.5，CentOS6.6 自带的RPM包为 1.5 的。

二，安装配置HAProxy

以下实验环境均为CentOS6.6 i686平台。

1，安装haproxy

```
[root@LB ~]# yum install -y haproxy #直接使用RPM来安装
[root@LB ~]# rpm -qi haproxy #版本为1.5.4
Name : haproxy Relocations: (not relocatable)
Version : 1.5.4 Vendor: CentOS
Release : 2.el6 Build Date: Thu 23 Jul 2015 04:26:35 PM PDT
Install Date: Sat 29 Aug 2015 06:49:30 PM PDT Build Host: c6b9.bsys.dev.centos.org
Group : System Environment/Daemons Source RPM: haproxy-1.5.4-2.el6.src.rpm
Size : 2542578 License: GPLv2+
Signature : RSA/SHA1, Fri 24 Jul 2015 01:39:18 PM PDT, Key ID 0946fca2c105b9de
Packager : CentOS BuildSystem <http://bugs.centos.org>
URL : http://www.haproxy.org/
Summary : HAProxy is a TCP/HTTP reverse proxy for high availability environments
[root@LB ~]# rpm -ql haproxy
/etc/haproxy
/etc/haproxy/haproxy.cfg ---------->配置文件
/etc/logrotate.d/haproxy
/etc/rc.d/init.d/haproxy
/etc/sysconfig/haproxy
/usr/bin/halog
/usr/bin/iprange
/usr/sbin/haproxy
[root@LB ~]# cd /etc/haproxy/
```

2，详解配置文件

haproxy 的配置文件由两部分组成：全局设定和对代理的设定，共分为五段：global，defaults，frontend，backend，listen。

2.1 配置文件格式

HAProxy的配置处理3类来主要参数来源：

——最优先处理的命令行参数;

——“global”配置段，用于设定全局配置参数；

——proxy相关配置段，如“defaults”、“listen”、“frontend”和“backend”；

2.2 时间格式

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)；
```

2.3 全局配置

* 进程管理及安全相关的参数

– chroot <jail dir>：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；

– daemon：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；

– gid <number>：以指定的GID运行haproxy，建议使用专用于运行haproxy的GID，以免因权限问题带来风险；

– group <group name>：同gid，不过指定的组名；

– log <address> <facility> [max level [min level]]：定义全局的syslog服务器，最多可以定义两个；

– log-send-hostname [<string>]：在syslog信息的首部添加当前主机名，可以为“string”指定的名称，也可以缺省使用当前主机名；

– nbproc <number>：指定启动的haproxy进程的个数，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：

– uid：以指定的UID身份运行haproxy进程；

– ulimit-n：设定每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；Linux默认单进程打开文件数为1024个

– user：同uid，但使用的是用户名；

– stats：用户访问统计数据的接口

– node：定义当前节点的名称，用于HA场景中多haproxy进程共享同一个IP地址时；

– description：当前实例的描述信息；

* 性能调整相关的参数

– maxconn <number>：设定每个haproxy进程所接受的最大并发连接数，其等同于命令行选项“-n”；“ulimit -n”自动计算的结果正是参照此参数设定的；

– maxpipes <number>：haproxy使用pipe完成基于内核的tcp报文重组，此选项则用于设定每进程所允许使用的最大pipe个数；每个pipe会打开两个文件描述符，因此，“ulimit -n”自动计算时会根据需要调大此值；默认为maxconn/4，其通常会显得过大；

– noepoll：在Linux系统上禁用epoll机制；

– nokqueue：在BSE系统上禁用kqueue机制；

– nopoll：禁用poll机制；

– nosepoll：在Linux禁用启发式epoll机制；

– nosplice：禁止在Linux套接字上使用内核tcp重组，这会导致更多的recv/send系统调用；不过，在Linux 2.6.25-28系列的内核上，tcp重组功能有bug存在；

– spread-checks <0..50, in percent>：在haproxy后端有着众多服务器的场景中，在精确的时间间隔后统一对众服务器进行健康状况检查可能会带来意外问题；此选项用于将其检查的时间间隔长度上增加或减小一定的随机时长；

– tune.bufsize <number>：设定buffer的大小，同样的内存条件小，较小的值可以让haproxy有能力接受更多的并发连接，较大的值可以让某些应用程序使用较大的cookie信息；默认为16384，其可以在编译时修改，不过强烈建议使用默认值；

– tune.chksize <number>：设定检查缓冲区的大小，单位为字节；更大的值有助于在较大的页面中完成基于字符串或模式的文本查找，但也会占用更多的系统资源；不建议修改；

– tune.maxaccept <number> ：设定haproxy进程内核调度运行时一次性可以接受的连接的个数，较大的值可以带来较大的吞吐率，默认在单进程模式下为100，多进程模式下为8，设定为-1可以禁止此限制；一般不建议修改；

– tune.maxpollevents <number> ：设定一次系统调用可以处理的事件最大数，默认值取决于OS；其值小于200时可节约带宽，但会略微增大网络延迟，而大于200时会降低延迟，但会稍稍增加网络带宽的占用量；

– tune.maxrewrite <number> ：设定为首部重写或追加而预留的缓冲空间，建议使用1024左右的大小；在需要使用更大的空间时，haproxy会自动增加其值；

– tune.rcvbuf.client <number> ：

– tune.rcvbuf.server <number> ：设定内核套接字中服务端或客户端接收缓冲的大小，单位为字节；强烈推荐使用默认值；

– tune.sndbuf.client ：

– tune.sndbuf.server ：

* Debug相关的参数

```
- debug

- quiet
```

* 超时时长

```
timeout http request ：在客户端建立连接但不请求数据时，关闭客户端连接
timeout queue ：等待最大时长
timeout connect： 定义haproxy将客户端请求转发至后端服务器所等待的超时时长
timeout client：客户端非活动状态的超时时长
timeout server：客户端与服务器端建立连接后，等待服务器端的超时时长，
timeout http-keep-alive ：定义保持连接的超时时长
timeout check：健康状态监测时的超时时间，过短会误判，过长资源消耗
maxconn :每个server最大的连接数

http-server-close ：在使用长连接时，为了避免客户端超时没有关闭长连接，此功能可以使服务器端关闭长连接
redispatch： 在使用基于cookie定向时，一旦后端某一server宕机时，会将会话重新定向至某一上游服务器，必须使用 的选项
```

* 实现访问控制：

```
http-request: 7层过滤
tcp-request content: tcp层过滤，四层过滤
```

2.4 代理

代理相关的配置可以如下配置段中。

```
– defaults <name>
– frontend <name>
– backend <name>
– listen <name>
```

“defaults”段用于为所有其它配置段提供默认参数，这配置默认配置参数可由下一个“defaults”所重新设定。

“frontend”段用于定义一系列监听的套接字，这些套接字可接受客户端请求并与之建立连接。

“backend”段用于定义一系列“后端”服务器，代理将会将对应客户端的请求转发至这些服务器。

“listen”段通过关联“frontend”和“backend”定义了一个完整的代理，通常只对TCP流量有用。

所有代理的名称只能使用大写字母、小写字母、数字、-(中线)、_(下划线)、.(点号)和:(冒号)。此外，ACL名称会区分字母大小写。

三、配置文件中的关键字参考

3.1 balance

```
balance [ ]

balance url_param [check_post []]
```

定义负载均衡算法，可用于“defaults”、“listen”和“backend”。用于在负载均衡场景中挑选一个server，其仅应用于持久信息不可用的条件下或需要将一个连接重新派发至另一个服务器时。支持的算法有：

3.11 roundrobin：基于权重进行轮叫，在服务器的处理时间保持均匀分布时，这是最平衡、最公平的算法。此算法是动态的，这表示其权重可以在运行时进行调整，不过，在设计上，每个后端服务器仅能最多接受4128个连接；并支持慢启动。

3.12 static-rr：基于权重进行轮叫，与roundrobin类似，但是为静态方法，在运行时调整其服务器权重不会生效；不过，其在后端服务器连接数上没有限制；不支持慢启动，在高负荷的情况下，服务器重新上线时会立即被分配大量连接。

3.13 leastconn（WLC）：适用于长连接的会话，新的连接请求被派发至具有最少连接数目的后端服务器；在有着较长时间会话的场景中推荐使用此算法，如LDAP、SQL等，其并不太适用于较短会话的应用层协议，如HTTP；此算法是动态的，

可以在运行时调整其权重；

3.14 source：将请求的源地址进行hash运算，并由后端服务器的权重总数相除后派发至某匹配的服务器；这可以使得同一个客户端IP的请求始终被派发至某特定的服务器；不过，当服务器权重总数发生变化时，如某服务器宕机或添加了新的服务器，许多客户端的请求可能会被派发至与此前请求不同的服务器；常用于负载均衡无cookie功能的基于TCP的协议；其默认为静态，不过也可以使用hash-type修改此特性；

1，对原地址hash，第一次调度时使用WLC

source：IP层，位于同一个NAT服务器背后的多个请求都会定向至同一个upstream server，不利于负载均衡，一般只有不支持使用cookie插入又需要保持会话时使用

cookie：应用层，有更好的负载均衡效果；

2，hash/weight%ip ：除以权重取模

3.15 uri：对URI的左半部分(“问题”标记之前的部分)或整个URI进行hash运算，并由服务器的总权重相除后派发至某匹配的服务器；这可以使得对同一个URI的请求总是被派发至某特定的服务器，除非服务器的权重总数发生了变化；此算法常用于代理缓存或反病毒代理以提高缓存的命中率；需要注意的是，此算法仅应用于HTTP后端服务器场景；其默认为静态算法，不过也可以使用hash-type修改此特性；

3.16 url_param：通过<argument>为URL指定的参数在每个HTTP GET请求中将会被检索；如果找到了指定的参数且其通过等于号“=”被赋予了一个值，那么此值将被执行hash运算并被服务器的总权重相除后派发至某匹配的服务器；此算法可以通过追踪请求中的用户标识进而确保同一个用户ID的请求将被送往同一个特定的服务器，除非服务器的总权重发生了变化；如果某请求中没有出现指定的参数或其没有有效值，则使用轮叫算法对相应请求进行调度；此算法默认为静态的，不过其也可以使用hash-type修改此特性；

3.17 hdr(<name>)：对于每个HTTP请求，通过<name>指定的HTTP首部将会被检索；如果相应的首部没有出现或其没有有效值，则使用轮叫算法对相应请求进行调度；其有一个可选选项“use_domain_only”，可在指定检索类似Host类的首部时仅计算域名部分(比如通过www.feiyu.com来说，仅计算feiyu字符串的hash值)以降低hash算法的运算量；此算法默认为静态的，不过其也可以使用hash-type修改此特性；

3.18 rdp-cookie(name)

，表示根据据cookie(name)来锁定并哈希每一次TCP请求。

3.2 bind

```
bind [<address>]:<port_range> [, …]
bind [<address>]:<port_range> [, …] interface <interface>
```

此指令仅能用于frontend和listen区段，用于定义一个或几个监听的套接字。

<address>：可选选项，其可以为主机名、IPv4地址、IPv6地址或*；省略此选项、将其指定为*或0.0.0.0时，将监听当前系统的所有IPv4地址；<port_range>：可以是一个特定的TCP端口，也可是一个端口范围(如5005-5010)，代理服务器将通过指定的端口来接收客户端请求；需要注意的是，每组监听的套接字<address:port>在同一个实例上只能使用一次，而且小于1024的端口需要有特定权限的用户才能使用，这可能需要通过uid参数来定义；<interface>：指定物理接口的名称，仅能在Linux系统上使用；其不能使用接口别名，而仅能使用物理接口名称，而且只有管理有权限指定绑定的物理接口；

3.3 mode

```
mode { tcp|http|health }
```

设定实例的运行模式或协议。当实现内容交换时，前端和后端必须工作于同一种模式(一般说来都是HTTP模式)，否则将无法启动实例。

tcp：实例运行于纯TCP模式，在客户端和服务器端之间将建立一个全双工的连接，且不会对7层报文做任何类型的检查；通常用于SSL、SSH、SMTP等应用；
http：实例运行于HTTP模式，客户端请求在转发至后端服务器之前将被深度分析，所有不与RFC格式兼容的请求都会被拒绝；此为默认模式；
health：实例工作于health模式，其对入站请求仅响应“OK”信息并关闭连接，且不会记录任何日志信息；此模式将用于响应外部组件的健康状态检查请求；目前来讲，此模式已经废弃，因为tcp或http模式中的monitor关键字可完成类似功能；

3.4 hash-type

```
hash-type <method>
```

定义用于将hash码映射至后端服务器的方法；其不能用于frontend区段；可用方法有map-based和consistent，在大多数场景下推荐使用默认的map-based方法。

map-based：hash表是一个包含了所有在线服务器的静态数组。其hash值将会非常平滑，会将权重考虑在列，但其为静态方法，对在线服务器的权重进行调整将不会生效，这意味着其不支持慢速启动。此外，挑选服务器是根据其在数组中的

位置进行的，因此，当一台服务器宕机或添加了一台新的服务器时，大多数连接将会被重新派发至一个与此前不同的服务器上，对于缓存服务器的工作场景来说，此方法不甚适用。

consistent：“一致性哈希算法”，hash表是一个由各服务器填充而成的树状结构，将服务器散列在hash环上；基于hash键在hash树中查找相应的服务器时，最近的服务器将被选中。此方法是动态的，支持在运行时修改服务器权重，因此兼

容慢速启动的特性。添加一个新的服务器时，仅会对一小部分请求产生影响，因此，尤其适用于后端服务器为cache的场景。不过，此算法不甚平滑，派发至各服务器的请求未必能达到理想的均衡效果，因此，可能需要不时的调整服务器的权

重以获得更好的均衡性。

3.5 log

```
log global
```

```
log <address> <facility> [<level> [<minlevel>]]
```

为每个实例启用事件和流量日志，因此可用于所有区段。每个实例最多可以指定两个log参数，不过，如果使用了“log global”且“global”段已经定了两个log参数时，多余了log参数将被忽略。

global：当前实例的日志系统参数同”global”段中的定义时，将使用此格式；每个实例仅能定义一次“log global”语句，且其没有任何额外参数；

<address>：定义日志发往的位置，其格式之一可以为<IPv4_address:PORT>，其中的port为UDP协议端口，默认为514；格式之二为Unix套接字文件路径，但需要留心chroot应用及用户的读写权限；

<facility>：可以为syslog系统的标准facility之一；

<level>：定义日志级别，即输出信息过滤器，默认为所有信息；指定级别时，所有等于或高于此级别的日志信息将会被发送；

3.6 maxconn

```
maxconn <conns>
```

设定一个前端的最大并发连接数，因此，其不能用于backend区段。对于大型站点来说，可以尽可能提高此值以便让haproxy管理连接队列，从而避免无法应答用户请求。当然，此最大值不能超出“global”段中的定义。此外，需要留心的是，haproxy会为每个连接维持两个缓冲，每个缓冲的大小为8KB，再加上其它的数据，每个连接将大约占用17KB的RAM空间。这意味着经过适当优化后，有着1GB的可用RAM空间时将能维护40000-50000并发连接。

如果为<conns>指定了一个过大值，极端场景下，其最终占据的空间可能会超出当前主机的可用内存，这可能会带来意想不到的结果；因此，将其设定了一个可接受值方为明智决定。其默认为2000。

3.7 default_backend

```
default_backend <backend>
```

在没有匹配的”use_backend”规则时为实例指定使用的默认后端，因此，其不可应用于backend区段。在”frontend”和”backend”之间进行内容交换时，通常使用”use-backend”定义其匹配规则；而没有被规则匹配到的请求将由此参数指定的后端接收。

<backend>：指定使用的后端的名称；

使用案例：

```
use_backend dynamic if url_dyn
use_backend static if url_css url_img extension_img
default_backend dynamic
```

3.8 server

```
server <name> <address>[:port] [param*]
```

为后端声明一个server，因此，不能用于defaults和frontend区段。

<name>：为此服务器指定的内部名称，其将出现在日志及警告信息中；如果设定了”http-send-server-name”，它还将被添加至发往此服务器的请求首部中；

<address>：此服务器的的IPv4地址，也支持使用可解析的主机名，只不过在启动时需要解析主机名至相应的IPv4地址；

[:port]：指定将连接请求所发往的此服务器时的目标端口，其为可选项；未设定时，将使用客户端请求时的同一相端口；

[param*]：为此服务器设定的一系参数；其可用的参数非常多，具体请参考官方文档中的说明，下面仅说明几个常用的参数；

服务器或默认服务器参数：

backup：设定为备用服务器，仅在负载均衡场景中的其它server均不可用于启用此server；

check：启动对此server执行健康状态检查，其可以借助于额外的其它参数完成更精细的设定，如：

inter <delay>：设定健康状态检查的时间间隔，单位为毫秒，默认为2000；也可以使用fastinter和downinter来根据服务器端状态优化此时间延迟；

rise <count>：设定健康状态检查中，某离线的server从离线状态转换至正常状态需要成功检查的次数；

fall <count>：确认server从正常状态转换为不可用状态需要检查的次数；

cookie <value>：为指定server设定cookie值，此处指定的值将在请求入站时被检查，第一次为此值挑选的server将在后续的请求中被选中，其目的在于实现持久连接的功能；

maxconn <maxconn>：指定此服务器接受的最大并发连接数；如果发往此服务器的连接数目高于此处指定的值，其将被放置于请求队列，以等待其它连接被释放；

haproxy 有n个进程，每个支持m个连接，后端有x个服务器，每个最大支持y个连接，则 n*m <= x*y，如果后端服务器支持排队，则n*m <= x*（y+z），z为每个服务器的排队队列

maxqueue <maxqueue>：设定请求队列的最大长度；

observe <mode>：通过观察服务器的通信状况来判定其健康状态，默认为禁用，其支持的类型有“layer4”和“layer7”，“layer7”仅能用于http代理场景；

redir <prefix>：启用重定向功能，将发往此服务器的GET和HEAD请求均以302状态码响应；需要注意的是，在prefix后面不能使用/，且不能使用相对地址，以免造成循环；例如：

```
server srv1 172.16.100.6:80 redir http://imageserver.feiyu.com check
```

weight <weight> ：权重，默认为1，最大值为256，0表示不参与负载均衡（不被调度）；

检查方法：

```
option httpchk
option httpchk
option httpchk
option httpchk ：不能用于frontend段，例如：
backend https_relay
mode tcp
option httpchk OPTIONS * HTTP/1.1\r\nHost:\ www.feiyu.com
server apache1 192.168.1.1:443 check port 80
```

使用案例：

```
server first 172.16.100.7:1080 cookie first check inter 1000
server second 172.16.100.8:1080 cookie second check inter 1000
```

3.9 capture request header

```
capture request header <name> len <length>
```

捕获并记录指定的请求首部最近一次出现时的第一个值，仅能用于“frontend”和“listen”区段。捕获的首部值使用花括号{}括起来后添加进日志中。如果需要捕获多个首部值，它们将以指定的次序出现在日志文件中，并以竖线“|”作为分隔符。不存在的首部记录为空字符串，最常需要捕获的首部包括在虚拟主机环境中使用的“Host”、上传请求首部中的“Content-length”、快速区别真实用户和网络机器人的“User-agent”，以及代理环境中记录真实请求来源的“X-Forward-For”。

<name>：要捕获的首部的名称，此名称不区分字符大小写，但建议与它们出现在首部中的格式相同，比如大写首字母。需要注意的是，记录在日志中的是首部对应的值，而非首部名称。

<length>：指定记录首部值时所记录的精确长度，超出的部分将会被忽略。

可以捕获的请求首部的个数没有限制，但每个捕获最多只能记录64个字符。为了保证同一个frontend中日志格式的统一性，首部捕获仅能在frontend中定义。

3.10 capture response header

```
capture response header <name> len <length>
```

捕获并记录响应首部，其格式和要点同请求首部。

3.11 stats enable

启用基于程序编译时默认设置的统计报告，不能用于“frontend”区段。只要没有另外的其它设定，它们就会使用如下的配置：

```
- stats uri : /haproxy?stats
- stats realm : "HAProxy Statistics"
- stats auth : no authentication
- stats scope : no restriction
```

尽管“stats enable”一条就能够启用统计报告，但还是建议设定其它所有的参数，以免其依赖于默认设定而带来非期后果。下面是一个配置案例。

```
backend public_www
server websrv1 172.16.100.11:80
stats enable
stats hide-version
stats scope .
stats uri /haproxyadmin?stats
stats realm Haproxy\ Statistics
stats auth statsadmin:password
stats auth statsmaster:password
```

3.12 stats hide-version

```
stats hide-version
```

启用统计报告并隐藏HAProxy版本报告，不能用于“frontend”区段。默认情况下，统计页面会显示一些有用信息，包括HAProxy的版本号，然而，向所有人公开HAProxy的精确版本号是非常有风险的，因为它能帮助恶意用户快速定位版本的缺陷和漏洞。尽管“stats hide-version”一条就能够启用统计报告，但还是建议设定其它所有的参数，以免其依赖于默认设定而带来非期后果。具体请参照“stats enable”一节的说明。

3.13 stats realm

```
stats realm <realm>
```

启用统计报告并高精认证领域，不能用于“frontend”区段。haproxy在读取realm时会将其视作一个单词，因此，中间的任何空白字符都必须使用反斜线进行转义。此参数仅在与“stats auth”配置使用时有意义。

<realm>：实现HTTP基本认证时显示在浏览器中的领域名称，用于提示用户输入一个用户名和密码。

尽管“stats realm”一条就能够启用统计报告，但还是建议设定其它所有的参数，以免其依赖于默认设定而带来非期后果。具体请参照“stats enable”一节的说明。

3.14 stats scope

```
stats scope { <name> | "." }
```

启用统计报告并限定报告的区段，不能用于“frontend”区段。当指定此语句时，统计报告将仅显示其列举出区段的报告信息，所有其它区段的信息将被隐藏。如果需要显示多个区段的统计报告，此语句可以定义多次。需要注意的是，区段名称检测仅仅是以字符串比较的方式进行，它不会真检测指定的区段是否真正存在。

<name> ：可以是一个“listen”、“frontend”或“backend”区段的名称，而“.”则表示stats scope语句所定义的当前区段。

尽管“stats scope”一条就能够启用统计报告，但还是建议设定其它所有的参数，以免其依赖于默认设定而带来非期后果。下面是一个配置案例。

```
backend private_monitoring
stats enable
stats uri /haproxyadmin?stats
stats refresh 10s
```

3.15 stats auth

```
stats auth <user>:<passwd>
```

启用带认证的统计报告功能并授权一个用户帐号，其不能用于“frontend”区段。

<user>：授权进行访问的用户名；

<passwd>：此用户的访问密码，明文格式；

此语句将基于默认设定启用统计报告功能，并仅允许其定义的用户访问，其也可以定义多次以授权多个用户帐号。可以结合“stats realm”参数在提示用户认证时给出一个领域说明信息。在使用非法用户访问统计功能时，其将会响应一个“401 Forbidden”页面。其认证方式为HTTP Basic认证，密码传输会以明文方式进行，因此，配置文件中也使用明文方式存储以说明其非保密信息故此不能相同于其它关键性帐号的密码。

尽管“stats auth”一条就能够启用统计报告，但还是建议设定其它所有的参数，以免其依赖于默认设定而带来非期后果。

3.16 stats admin

```
stats admin { if | unless } <cond>
```

在指定的条件满足时启用统计报告页面的管理级别功能，它允许通过web接口启用或禁用服务器，不过，基于安全的角度考虑，统计报告页面应该尽可能为只读的。此外，如果启用了HAProxy的多进程模式，启用此管理级别将有可能导致异常行为。

目前来说，POST请求方法被限制于仅能使用缓冲区减去保留部分之外的空间，因此，服务器列表不能过长，否则，此请求将无法正常工作。因此，建议一次仅调整少数几个服务器。下面是两个案例，第一个限制了仅能在本机打开报告页面时启用管理级别功能，第二个定义了仅允许通过认证的用户使用管理级别功能。

```
backend stats_localhost
stats enable
stats admin if LOCALHOST
backend stats_auth
stats enable
stats auth haproxyadmin:password
stats admin if TRUE
```

3.17 option httplog

```
option httplog [ clf ]
```

启用记录HTTP请求、会话状态和计时器的功能。

clf：使用CLF格式来代替HAProxy默认的HTTP格式，通常在使用仅支持CLF格式的特定日志分析器时才需要使用此格式。

默认情况下，日志输入格式非常简陋，因为其仅包括源地址、目标地址和实例名称，而“option httplog”参数将会使得日志格式变得丰富许多，其通常包括但不限于HTTP请求、连接计时器、会话状态、连接数、捕获的首部及cookie、“frontend”、“backend”及服务器名称，当然也包括源地址和端口号等。

3.18 option logasap

```
option logasap
no option logasap
```

启用或禁用提前将HTTP请求记入日志，不能用于“backend”区段。

默认情况下，HTTP请求是在请求结束时进行记录以便能将其整体传输时长和字节数记入日志，由此，传较大的对象时，其记入日志的时长可能会略有延迟。“option logasap”参数能够在服务器发送complete首部时即时记录日志，只不过，此时将不记录整体传输时长和字节数。此情形下，捕获“Content-Length”响应首部来记录传输的字节数是一个较好选择。下面是一个例子。

```
listen http_proxy 0.0.0.0:80
mode http
option httplog
option logasap
log 172.16.100.9 local2
```

3.19 option forwardfor

```
option forwardfor [ except <network> ] [ header <name> ] [ if-none ]
```

允许在发往服务器的请求首部中插入“X-Forwarded-For”首部。

<network> ：可选参数，当指定时，源地址为匹配至此网络中的请求都禁用此功能。

<name> ：可选参数，可使用一个自定义的首部，如“X-Client”来替代“X-Forwarded-For”。有些独特的web服务器的确需要用于一个独特的首部。

if-none：仅在此首部不存在时才将其添加至请求报文问道中。

HAProxy工作于反向代理模式，其发往服务器的请求中的客户端IP均为HAProxy主机的地址而非真正客户端的地址，这会使得服务器端的日志信息记录不了真正的请求来源，“X-Forwarded-For”首部则可用于解决此问题。HAProxy可以向每个发往服务器的请求上添加此首部，并以客户端IP为其value。

需要注意的是，HAProxy工作于隧道模式，其仅检查每一个连接的第一个请求，因此，仅第一个请求报文被附加此首部。如果想为每一个请求都附加此首部，请确保同时使用了“option httpclose”、“option forceclose”和“option http-server-close”几个option。

下面是一个例子。

```
frontend www
mode http
option forwardfor except 127.0.0.1
```

3.20 errorfile

```
errorfile <code> <file>
```

在用户请求不存在的页面时，返回一个页面文件给客户端而非由haproxy生成的错误代码；可用于所有段中。

<code>：指定对HTTP的哪些状态码返回指定的页面；这里可用的状态码有200、400、403、408、500、502、503和504；

<file>：指定用于响应的页面文件；

例如：

```
errorfile 400 /etc/haproxy/errorpages/400badreq.http
errorfile 403 /etc/haproxy/errorpages/403forbid.http
errorfile 503 /etc/haproxy/errorpages/503sorry.http
```

3.21 errorloc 和 errorloc302

```
errorloc <code> <url>
errorloc302 <code> <url>
```

请求错误时，返回一个HTTP重定向至某URL的信息；可用于所有配置段中。

<code>：指定对HTTP的哪些状态码返回指定的页面；这里可用的状态码有200、400、403、408、500、502、503和504；

<url>：Location首部中指定的页面位置的具体路径，可以是在当前服务器上的页面的相对路径，也可以使用绝对路径；需要注意的是，如果URI自身错误时产生某特定状态码信息的话，有可能会导致循环定向；

需要留意的是，这两个关键字都会返回302状态吗，这将使得客户端使用同样的HTTP方法获取指定的URL，对于非GET法的场景(如POST)来说会产生问题，因为返回客户的URL是不允许使用GET以外的其它方法的。如果的确有这种问题，可以使用errorloc303来返回303状态码给客户端。

3.22 errorloc303

```
errorloc303 <code> <url>
```

请求错误时，返回一个HTTP重定向至某URL的信息给客户端；可用于所有配置段中。

<code>：指定对HTTP的哪些状态码返回指定的页面；这里可用的状态码有400、403、408、500、502、503和504；

<url>：Location首部中指定的页面位置的具体路径，可以是在当前服务器上的页面的相对路径，也可以使用绝对路径；需要注意的是，如果URI自身错误时产生某特定状态码信息的话，有可能会导致循环定向；

例如：

```
backend webserver
server 172.16.100.6 172.16.100.6:80 check maxconn 3000 cookie srv01
server 172.16.100.7 172.16.100.7:80 check maxconn 3000 cookie srv02
errorloc 403 /etc/haproxy/errorpages/sorry.htm
errorloc 503 /etc/haproxy/errorpages/sorry.htm
```

四、ACL

haproxy的ACL用于实现基于请求报文的首部、响应报文的内容或其它的环境状态信息来做出转发决策，这大大增强了其配置弹性。其配置法则通常分为两步，首先去定义ACL，即定义一个测试条件，而后在条件得到满足时执行某特定的动作，如阻止请求或转发至某特定的后端。定义ACL的语法格式如下。

```
acl <aclname> <criterion> [flags] [operator] <value> ...
```

<aclname>：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；

<criterion>：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；

<value>：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；

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)。

5.1 常用的测试标准(criteria)

5.1.1 be_sess_rate

```
be_sess_rate(backend) <integer>
```

用于测试指定的backend上会话创建的速率(即每秒创建的会话数)是否满足指定的条件；常用于在指定backend上的会话速率过高时将用户请求转发至另外的backend，或用于阻止攻击行为。例如：

```
backend dynamic
mode http
acl being_scanned be_sess_rate gt 50
redirect location /error_pages/denied.html if being_scanned
```

5.1.2 fe_sess_rate

```
fe_sess_rate(frontend) <integer>
```

用于测试指定的frontend(或当前frontend)上的会话创建速率是否满足指定的条件；常用于为frontend指定一个合理的会话创建速率的上限以防止服务被滥用。例如下面的例子限定入站邮件速率不能大于50封/秒，所有在此指定范围之外的请求都将被延时50毫秒。

```
frontend mail
bind :25
mode tcp
maxconn 500
acl too_fast fe_sess_rate ge 50
tcp-request inspect-delay 50ms
tcp-request content accept if ! too_fast
tcp-request content accept if WAIT_END
```

5.1.3 hdr <string>

```
hdr(header) <string>
```

用于测试请求报文中的所有首部或指定首部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()。例如下面的例子用于测试首部Connection的值是否为close。

```
hdr(Connection) -i close
```

5.1.4 method <string>

```
method <string>
```

测试HTTP请求报文中使用的方法。

5.1.5 path_beg <string>

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。

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

5.1.6 path_end <string>

用于测试请求的URL是否以<string>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。

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

5.1.7 hdr_beg <string>

用于测试请求报文的指定首部的开头部分是否符合<string>指定的模式。例如，下面的例子用记测试请求是否为提供静态内容的主机img、video、download或ftp。

```
acl host_static hdr_beg(host) -i img. video. download. ftp.
```

5.1.8 hdr_end <string>

用于测试请求报文的指定首部的结尾部分是否符合<string>指定的模式。

五、配置案例

前端调度器IP：192.168.1.210

后端应用服务器IP: 192.168.1.111 和 192.168.1.112

定义独立日志文件

```
[root@node1 haproxy]# vim /etc/rsyslog.conf #为其添加日志功能
# Provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514 ------>启动udp，启动端口后将作为服务器工作
# Provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514 ------>启动tcp监听端口
local2.* /var/log/haproxy.log

[root@node1 haproxy]# service rsyslog restar
[root@LB haproxy]# vim haproxy.cfg
log 127.0.0.1 local2 --------->在global端中添加此行
```

一个最简单的http服务的配置

```
global
log 127.0.0.1 local2
chroot /var/lib/haproxy
pidfile /var/run/haproxy.pid
maxconn 4000
user haproxy
group haproxy
daemon
stats socket /var/lib/haproxy/stats
defaults
mode http
log global
option httplog
option dontlognull
option http-server-close
option forwardfor except 127.0.0.0/8
option redispatch
retries 3
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout http-keep-alive 10s
timeout check 10s
maxconn 3000
frontend webser #webser为名称
option forwardfor
bind *:80
default_backend app
backend app
balance roundrobin #使拥roundrobin 算法
server app1 192.168.1.111:80 check
server app2 192.168.1.112:80 check
```

haproxy统计页面的输出机制

```
frontend webser
log 127.0.0.1 local3
option forwardfor
bind *:80
default_backend app
backend app
cookie node insert nocache
balance roundrobin
server app1 192.168.1.111:80 check cookie node1 intval 2 rise 1 fall 2
server app2 192.168.1.112:80 check cookie node2 intval 2 rise 1 fall 2
server backup 127.0.0.1:8010 check backup
listen statistics
bind *:8009 # 自定义监听端口
stats enable # 启用基于程序编译时默认设置的统计报告
stats auth admin:admin # 统计页面用户名和密码设置
stats uri /admin?stats # 自定义统计页面的URL，默认为/haproxy?stats
stats hide-version # 隐藏统计页面上HAProxy的版本信息
stats refresh 30s # 统计页面自动刷新时间
stats admin if TRUE #如果认证通过就做管理功能，可以管理后端的服务器
stats realm Hapadmin # 统计页面密码框上提示文本，默认为Haproxy\ Statistics
```

动静分离示例：

```
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
acl host_static hdr_beg(host) -i img. imgs. video. videos. ftp. image. download.
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
```

http服务器配置完整示例

```
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
# to have these messages end up in /var/log/haproxy.log you will
# need to:
#
# 1) configure syslog to accept network log events. This is done
# by adding the '-r' option to the SYSLOGD_OPTIONS in
# /etc/sysconfig/syslog
#
# 2) configure local2 events to go to the /var/log/haproxy.log
# file. A line like the following can be added to
# /etc/sysconfig/syslog
#
# local2.* /var/log/haproxy.log
#
log 127.0.0.1 local2
chroot /var/lib/haproxy
pidfile /var/run/haproxy.pid
maxconn 4000
user haproxy
group haproxy
daemon
defaults
mode http
log global
option httplog
option dontlognull
option http-server-close
option forwardfor except 127.0.0.0/8
option redispatch
retries 3
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout http-keep-alive 10s
timeout check 10s
maxconn 30000
listen stats
mode http
bind 0.0.0.0:1080
stats enable
stats hide-version
stats uri /haproxyadmin?stats
stats realm Haproxy\ Statistics
stats auth admin:admin
stats admin if TRUE
frontend http-in
bind *:80
mode http
log global
option httpclose
option logasap #不等待响应结束就记录日志，表示提前记录日志，一般日志会记录响应时长，此不记录响应时长
option dontlognull #不记录空信息
capture request header Host len 20 #记录请求首部的前20个字符
capture request header Referer len 60 #referer跳转引用，就是上一级
default_backend servers
frontend healthcheck
bind :1099 #定义外部检测机制
mode http
option httpclose
option forwardfor
default_backend servers
backend servers
balance roundrobin
server websrv1 192.168.1.111:80 check maxconn 2000
server websrv2 192.168.1.112:80 check maxconn 2000
```

负载均衡MySQL服务的配置示例

```
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
# to have these messages end up in /var/log/haproxy.log you will
# need to:
#
```

```
# 1) configure syslog to accept network log events. This is done
# by adding the '-r' option to the SYSLOGD_OPTIONS in
# /etc/sysconfig/syslog
#
# 2) configure local2 events to go to the /var/log/haproxy.log
# file. A line like the following can be added to
# /etc/sysconfig/syslog
#
# local2.* /var/log/haproxy.log
#
log 127.0.0.1 local2
chroot /var/lib/haproxy
pidfile /var/run/haproxy.pid
maxconn 4000
user haproxy
group haproxy
daemon
defaults
mode tcp
log global
option httplog
option dontlognull
retries 3
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout http-keep-alive 10s
timeout check 10s
maxconn 600
listen stats
mode http
bind 0.0.0.0:1080
stats enable
stats hide-version
stats uri /haproxyadmin?stats
stats realm Haproxy\ Statistics
stats auth admin:admin
stats admin if TRUE
frontend mysql
bind *:3306
mode tcp
log global
default_backend mysqlservers
backend mysqlservers
balance leastconn
server dbsrv1 192.168.1.111:3306 check port 3306 intval 2 rise 1 fall 2 maxconn 300
server dbsrv2 192.168.1.112:3306 check port 3306 intval 2 rise 1 fall 2 maxconn 300
```

感谢“**田飞雨**”投稿分享如此精彩的文章，网址：http://www.tianfeiyu.com

haproxy nginx 代理