Nginx 配合 haproxy 的模式的扩展

一、Proxy protocol 简单说明：

通过代理转发的 tcp 连接一般都会涉及到源 tcp 连接参数丢失的情况（参数包括：源地址，源端口）。在 http 协议中，其中 Forwarded-For 以及 X-Original-To 字段被分别用来表示源地址和目标地址。

然而这种实现需要线路上的所有中继都实现 http 的全栈解析，因为 Forwarded-For 在 http 数据包内部。

而这里引入的 proxy protocol，只需要发送数据时在 http 数据包的前面加入数据头，而在接收的时候解析掉数据头就可以了，更为简单。

简单的带有 proxy protocol 头部的 tcp 数据包如下图所示：

```
PROXY TCP4 192.168.37.154 192.168.37.160 59003 807
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: 192.168.37.160:807
If-Modified-Since: Wed, 28 May 2014 01:55:59 GMT
If-None-Match: "5385422f-264"
X-Forwarded-For: 192.168.37.154
```

可以看到第一行就是简易的头部，数据顺序是源请求地址，目标地址，源端口，目标端口。

Proxy-protocol 服务器使用 getsockname() and getpeername() 函数来分别获取需要的信息插入数据头中，数据包括：

1. 地址类型(AFAF_INET for IPv4, AF_INET6 for IPv6, AF_UNIX)
2. socket 类型 (SOCK_STREAM for TCP, SOCK_DGRAM for UDP)
3. layer 3 的源地址和目标地址
4. layer 4 的源端口和目标端口

由于为了方便调试，proxy protocol 的第一个版本使用，人类可读的形式，保存地址信息，这样利于调试，但是在 ipv6 的地址解析时会有消耗资源更多的劣势。所以 version 2 改为了不可读的形式。

二、环境和配置描述

Window 7 host 主机：

Ip：192.168.37.154

Ubuntu 虚拟机：

Ip：192.168.37.163

服务：

前端 haproxy

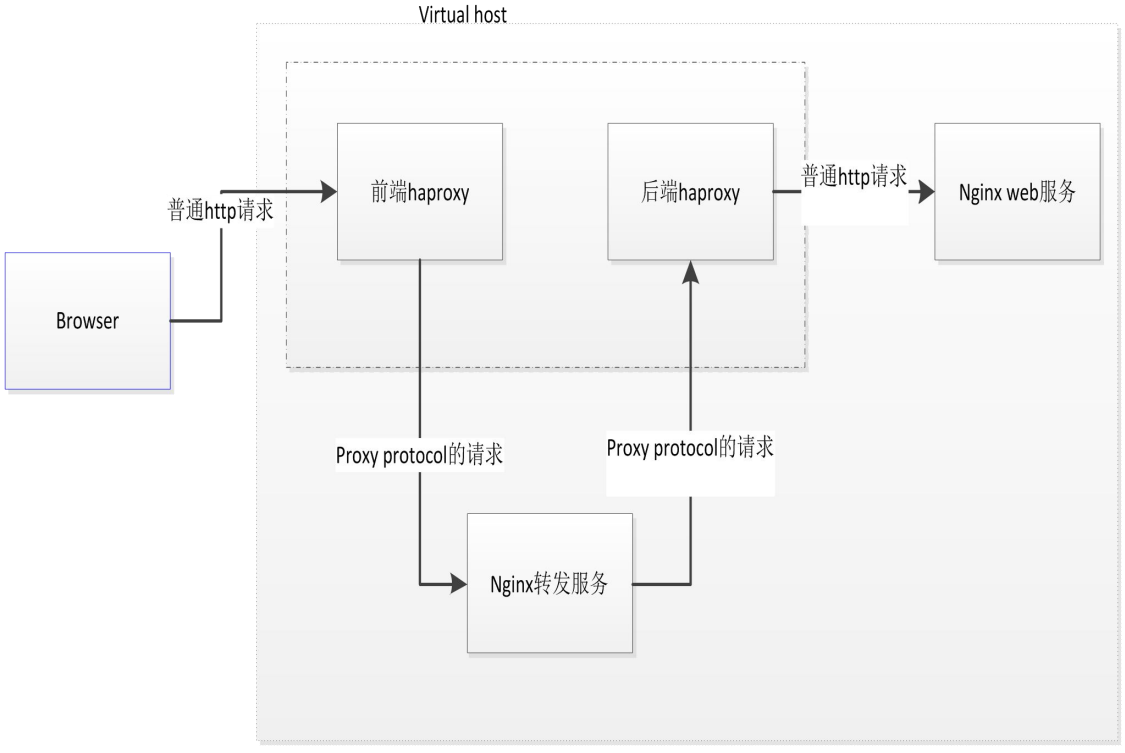
后端 haproxy

Nginx 转发服务

Nginx web 服务

Win7 机器上装有 linux 的虚拟机。虚拟机运行有 web 服务。在虚拟机中运行两个 haproxy 的转发服务，以及两个 nginx 服务。前端 Haproxy 服务用于接收 host 机器上浏览器对虚拟机 web 服务的请求，并将接收到的请求处理后（在 http 头部前加入 proxy protocol 的头），输出到 nginx 转发服务上，由 nginx 解析后，再转发到后端 haproxy 服务上。此时转发到后端 haproxy 上的请求，是 proxy protocol 的形式，最终由后端 haproxy 处理后，输出普通的 http 请求到运行 web 服务的 nginx 服务上。

请求转发的路线入下图所示：



其中前端的 haproxy 负责把前端的请求进行负载均衡的分发，同时可以把需要进行规则过滤的请求转发给 nginx 进行过滤匹配。所以需要 proxy protocol 来提供源地址信息给 nginx 进行过滤规则的匹配。Nginx 匹配完之后，可以再转发给 haproxy 进行统一配置的分发。

其中两个 haproxy 服务运行在一个进程上面，其配置文件内容为：

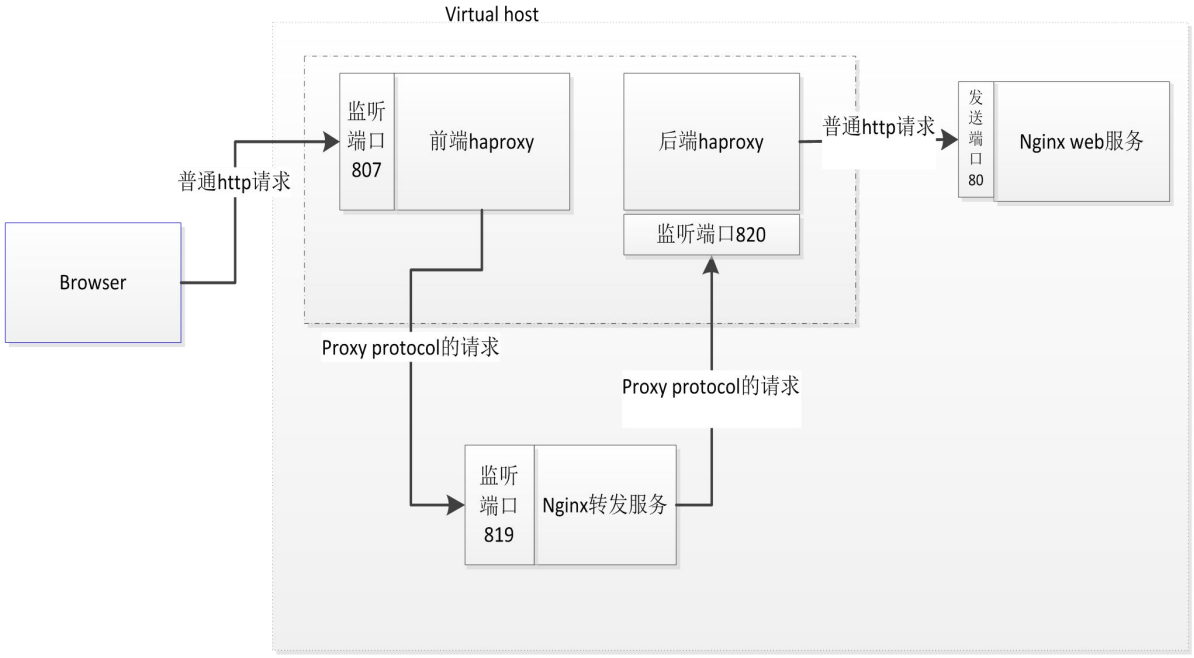
```
global
daemon
maxconn 256
log 127.0.0.1    local3 info    # linux syslog  服务

defaults
mode http                        #http  模式
timeout connect 5000ms
timeout client 50000ms
timeout server 50000ms
#source 0.0.0.0    usesrc clientip

frontend http-in            #前端 haproxy  监听在端口 807 上面，接收浏览器的请求
      bind 0.0.0.0:807
      mode http
      option httplog
      log global
      stats enable
      option http-pretend-keepalive
      option forwardfor
      use_backend proxy-out     #表示请求转发到  相应的 server 上面

backend proxy-out
      server http_proxy 127.0.0.1:819    check send-proxy    #将请求转发给 nginx 转发服务

frontend proxy-in
      bind 127.0.0.1:820 accept-proxy       #接收来自 nginx 转发服务的 proxy 协议的情切
      option httplog
      log global
      mode http
      stats enable
      option http-pretend-keepalive
      option forwardfor
      use_backend http-out

backend http-out
      server proxy_http 127.0.0.1:80    #将请求转发给 web 服务
```

如上配置后 haproxy 就开始提供服务了。

接下来是 nginx 转发服务的配置

```
    upstream backend {
        server 127.0.0.1:820;
    }

    server {
        listen *:819    proxy_protocol;
        location /{
            proxy_pass http://backend;
            proxy_set_header Host    $proxy_protocol_dst;
            send_proxy_protocol on;
            proxy_connect_timeout 10s;
            proxy_read_timeout 10s;
        }
    }
```

```
server {
        listen          80;
        server_name    _;

        #charset koi8-r;

        #access_log    logs/host.access.log    main;

        location / {
            root    html;
            index    index.html index.htm;
        }
        #error_page    404                    /404.html;

        # redirect server error pages to the static page /50x.html
        #
        error_page    500 502 503 504    /50x.html;
        location = /50x.html {
            root    html;
        }
}
```



三、代码更改

Nginx1.7 支持 proxy protocol：

Nginx 已经支持 proxy protocol 的解析，在 listen 字段后添加 proxy_protocol 即代表，该服务接收到的请求都要使用解析 proxy protocol 的数据头部。

解析过程是在 ngx_http_wait_request_handler 中处理请求时，用 ngx_proxy_protocol_parse 函数进行解析。由于原 ngx_proxy_protocol_parse 函数只解析了 proxy protocol 中的源请求字段，为解析源端口，目标地址，目标端口。所以我加入了解析后面几个字段的代码。并把信息保存在请求相关的 ngx_connection_t 结构中。并在 ngx_connection_t 结构中添加了相应的字段。

ngx_connection_t 结构中添加的字段有：

```
struct ngx_connection_s {
....
    ngx_str_t               proxy_protocol_addr;
    ngx_str_t               proxy_protocol_addr_port; //源端口
    ngx_str_t               proxy_protocol_dst;  //  目标地址
    ngx_str_t               proxy_protocol_dst_port; //目标端口
    int                      proxy_protocol_flag;     //proxy  标志
....
}
```

ngx_proxy_protocol_parse 中添加代码段后是：

```
ngx_proxy_protocol_parse(ngx_connection_t *c, u_char *buf, u_char *last) //buf  是数据包
```

的数据

```
    {
    ...
    c->proxy_protocol_addr.data = ngx_pnalloc(c->pool, len);

        if (c->proxy_protocol_addr.data == NULL) {
            return NULL;
        }

        ngx_memcpy(c->proxy_protocol_addr.data, addr, len);
        c->proxy_protocol_addr.len = len;

        if ( (t = (u_char *)memchr(p, ' ', s)) == NULL ) {
            goto invalid;
        }
        len = t-p;
        c->proxy_protocol_dst.data = ngx_pnalloc(c->pool, len);

        if (c->proxy_protocol_dst.data == NULL) {
            return NULL;
        }
        ngx_memcpy(c->proxy_protocol_dst.data,p,len);
        c->proxy_protocol_dst.len = len;
        p += (len+1);
        s -= (len+1);

        if ( (t = (u_char *)memchr(p, ' ', s)) == NULL ) {
            goto invalid;
        }
        len = t-p;
        c->proxy_protocol_addr_port.data = ngx_pnalloc(c->pool, len);

        if (c->proxy_protocol_addr_port.data == NULL) {
            return NULL;
        }
        ngx_memcpy(c->proxy_protocol_addr_port.data,p,len); //保存源端口
        c->proxy_protocol_addr_port.len = len;

    p += (len+1);
        s -= (len+1);

        if ( (t = (u_char *)memchr(p, CR, s)) == NULL ) {
            goto invalid;
        }
```

```
        len = t-p;
        c->proxy_protocol_dst_port.data = ngx_pnalloc(c->pool, len);
        if (c->proxy_protocol_dst_port.data == NULL) {
            return NULL;
        }
        ngx_memcpy(c->proxy_protocol_dst_port.data,p,len); //  保存目标端口
        c->proxy_protocol_dst_port.len = len;
        p += len;
        c->proxy_protocol_flag = 1;   //设置 flag
    ...
    }
```

解析的过程支持以后，需要添加的发送的过程。

由于使用 proxy_pass 模块进行请求的转发，所以在这个命令的基础上在添加一个选项来表示，是否进行 proxy protocol 的转发。

```
location /{
    proxy_pass http://backend;
    proxy_set_header Host  $proxy_protocol_dst;
    send_proxy_protocol on;
    proxy_connect_timeout 10s;
    proxy_read_timeout 10s;
}
```

在 http_proxy 模块的代码中添加这条命令：

```
{ ngx_string("send_proxy_protocol"),

    NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
    ngx_conf_set_flag_slot,
    NGX_HTTP_LOC_CONF_OFFSET,
    offsetof(ngx_http_proxy_loc_conf_t, upstream.send_proxy_protocol),
    NULL },
```

同时在配置变量 ngx_http_proxy_loc_conf_t 的中的 upstream 结构中添加 flag 变量 send_proxy_protocol。

由于 http 的请求最终都会调用 upstream 进行发送，并且 upstream 模块更加便于扩展，所以在 upstream 中添加一个函数，在生成 http 请求包之前，插入 proxy protocol 数据头。

在 ngx_http_upstream_connect 函数中通过之前的 flag 变量判断是否是 proxy protocol 如果是的话就调用 ngx_http_upstream_send_proxy_protocol 进行 proxy potocol 的头部的生成。

ngx_http_upstream_send_proxy_protocol 通过函数 ngx_output_chain 在 upstream 的输出中附加上 proxy protocol 的 header。

Deader 信息来自于两部分，如果 connection 里面之前的一端发送的是 proxy 头部的 http 请求，则从 connection 变量里的字符串形式保存的数据里面取信息，如果之前的一端是普通的 http，则从 sockaddr 变量转化出字符串形式的地址信息来生成 header。

同时添加了几个可以在配置文件里面使用的变量：

```
{ ngx_string("proxy_protocol_addr"), NULL,
      ngx_http_variable_proxy_protocol_addr, 0, 0, 0 },

{ ngx_string("proxy_protocol_addr_port"), NULL,
      ngx_http_variable_proxy_protocol_addr_port, 0, 0, 0 },

{ ngx_string("proxy_protocol_dst"), NULL,
      ngx_http_variable_proxy_protocol_dst, 0, 0, 0 },

{ ngx_string("proxy_protocol_dst_port"), NULL,
      ngx_http_variable_proxy_protocol_dst_port, 0, 0, 0 },
```

数据通信情况描述:

数据包抓包效果如下所示：

浏览器发起的请求：

```
6 1.771130000 192.168.37.154        192.168.37.160        HTTP        388 GET / HTTP/1.1

GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: 192.168.37.160:807
If-Modified-Since: Wed, 28 May 2014 01:55:59 GMT
If-None-Match: "5385422f-264"
Connection: Keep-Alive
```

发到 819 号端口的请求（加入 proxy protocol 头部）

```
▶Frame 19: 461 bytes on wire (3688 bits), 461 bytes captured (3688 bits) on interface 1
▶Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▼Transmission Control Protocol, Src Port: 47837 (47837), Dst Port: 819 (819), Seq: 1, Ack: 1, Len: 395
```

```
Stream Content
PROXY TCP4 192.168.37.154 192.168.37.160 59498 807
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: 192.168.37.160:807
If-Modified-Since: Wed, 28 May 2014 01:55:59 GMT
If-None-Match: "5385422f-264"
X-Forwarded-For: 192.168.37.154
```

经由 nginx 发到 820 端口的数据

```
PROXY TCP4 192.168.37.154 192.168.37.160 59532 807
GET / HTTP/1.0
Host: 192.168.37.160:807
Connection: close
Accept: text/html, application/xhtml+xml, */*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
If-Modified-Since: Wed, 28 May 2014 01:55:59 GMT
If-None-Match: "5385422f-264"
X-Forwarded-For: 192.168.37.154
```

经 nginx 处理后 http 内部数据顺序有所改变

最后是 nginx 发送网 web 服务器的数据包

```
GET / HTTP/1.0
Host: 192.168.37.160:807
Accept: text/html, application/xhtml+xml, */*
Accept-Language: zh-CN
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
If-Modified-Since: Wed, 28 May 2014 01:55:59 GMT
If-None-Match: "5385422f-264"
X-Forwarded-For: 192.168.37.154
X-Forwarded-For: 192.168.37.154
Connection: keep-alive
```

对 ipv6 的情况的配置：

Haproxy 配置文件为：

```
frontend http-in_v6
        bind :::807
        mode http
        option httplog
        log global
        stats enable
        option http-pretend-keepalive
        option forwardfor
        use_backend proxy-out_v6

backend proxy-out_v6
        server http_proxy ::1:819 check send-proxy

frontend proxy-in_v6
        bind :::820 accept-proxy
        option httplog
        log global
        mode http
        stats enable
        option http-pretend-keepalive
        option forwardfor
        use_backend http-out_v6

backend http-out_v6
        server proxy_http ::1:80
```

Nginx 配置文件为：

```
        upstream backend {
                server [::1]:820;
                server 127.0.0.1:820;
        }
        server {
                listen *:819    proxy_protocol;
                listen [::]:819    proxy_protocol;

                location /{
                        proxy_pass http://backend;
                        proxy_set_header Host    $proxy_protocol_dst;
                        send_proxy_protocol on;
                        proxy_connect_timeout 10s;
                        proxy_read_timeout 10s;
                }
        }
```

宿主机配置 ipv6 地址：2004::192:168:37:161/64 ，虚拟机配置 ipv6 地址：2004::192:168:37:162/64

在虚拟机里抓包后。

宿主机地址栏中输入 http://[2004::192:168:37:162]:807 向虚拟机发起请求。请求内容为

```
GET / HTTP/1.1
Host: [2004::192:168:37:162]:807
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

请求到达前端 haproxy 后，转发到 nginx 的数据包为：

```
PROXY TCP6 2004::192:168:37:161 2004::192:168:37:162 50113 807
GET / HTTP/1.1
Host: [2004::192:168:37:162]:807
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
If-Modified-Since: Tue, 03 Jun 2014 04:22:39 GMT
If-None-Match: "538d4d8f-14a"
Cache-Control: max-age=0
X-Forwarded-For: 2004::192:168:37:161
```

数据被交给 nginx 进行解析，解析后再加上 proxy protocol 头部，发送给后端的 haproxy：

```
PROXY TCP6 2004::192:168:37:161 2004::192:168:37:162 50113 807
GET / HTTP/1.0
Host: 2004::192:168:37:162
Connection: close
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
If-Modified-Since: Tue, 03 Jun 2014 04:22:39 GMT
If-None-Match: "538d4d8f-14a"
Cache-Control: max-age=0
X-Forwarded-For: 2004::192:168:37:161
```

可以看出上面的 host 字段发生了变化，由于之前配置中的 proxy_set_header Host $proxy_protocol_dst;

后端 haproxy 去除 proxy protocol 的数据头之后发送给 nginx web 引擎。

```
GET / HTTP/1.1
Host: [2004::192:168:37:162]:807
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
If-Modified-Since: Tue, 03 Jun 2014 04:22:39 GMT
If-None-Match: "538d4d8f-14a"
Cache-Control: max-age=0
```

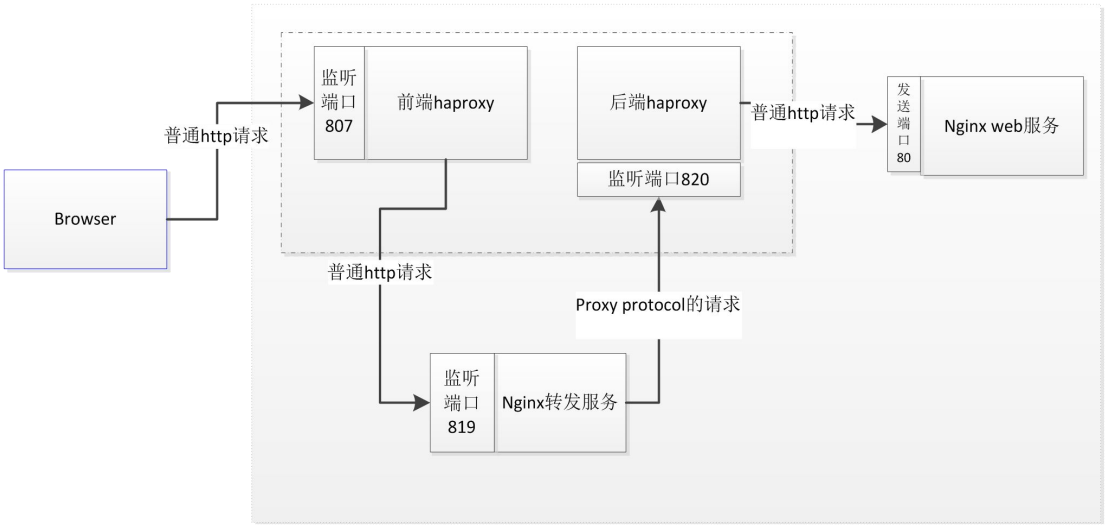
使用浏览器可以正常浏览网页。

四、其他模式

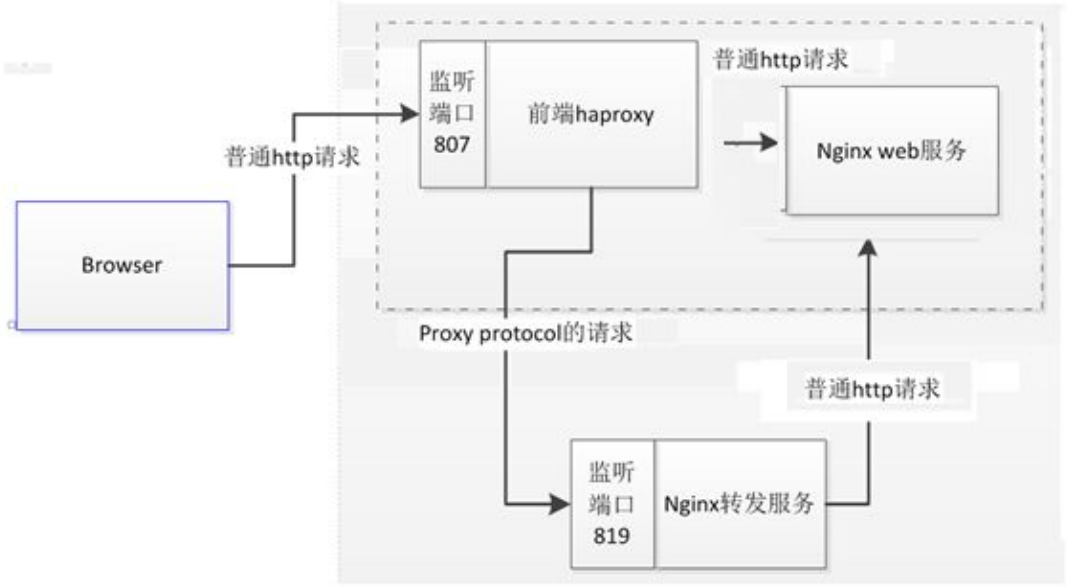
1.

前端的 haproxy 发送普通的 http 请求，nginx 来发送 proxy protocol 的 http。通过在

配置文件中打开和关闭相应的开关就可以实现



Nginx 在解析 proxy protocol后再转发的数据直接转发到 web 服务器上，不再进行 proxy protocol 的封装



2. Unix domain socket 的配置实验

Unix domain socket 或者 ipc socket(inter-process communication socket) 是一种在系统内部的两个进程之间进行端点通信的 socket。

While similar in functionality to named pipes, Unix domain sockets may be created as connection mode (SOCK_STREAM or SOCK_SEQPACKET) or as connectionless (SOCK_DGRAM), while pipes are streams only.

相对有名管道来说功能一样，然而 unix domain socket 有维持连接模式(SOCK_STREAM or SOCK_SEQPACKET) 以及无连接状态模式 (SOCK_DGRAM)的分别，

管道只有流一种模式。

Unix domain sockets 使用文件系统来标识他们的地址。他们以文件节点的形式被进程引用。这使得两个进程可以打开同样的 socket 来通信。然而，通信实际上全部发生在系统的内核层。

除了发送数据，通过 domain socket 进程还可发送文件描述符，用函数 sendmsg() 和 recvmsg()。

使用 domain socket 模式，haproxy 的配置需要做出调整：

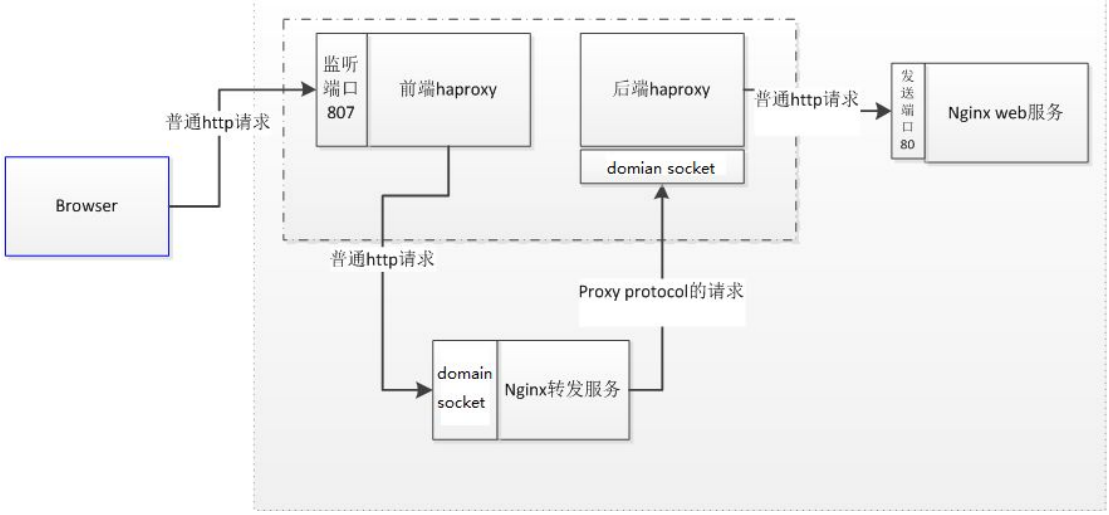
```
frontend http-in
        bind *:807
        mode http
        option httplog
        log global
        stats enable
        option http-pretend-keepalive
        option forwardfor
        use_backend proxy-out

backend proxy-out
        server    socket    /var/run/nginx.sock check send-proxy

frontend proxy-in
        bind /var/run/haproxy.sock user root mode 600 accept-proxy
        option httplog
        log global
        mode http
        stats enable
        option http-pretend-keepalive
        option forwardfor
        use_backend http-out

backend http-out
        server proxy_http 127.0.0.1:80
```

上面的 proxy-out 使用 domain socket 方式进行输出，输出到 nginx 的监听端。Proxy-in 端又用于接收 nginx 的转发。

Nginx 的配置调整如下：

```
upstream backend {
        server [::1]:820;
        server 127.0.0.1:820;
        server unix://var/run/haproxy.sock;
}
server {
        listen *:819    proxy_protocol;
        listen [::]:819    proxy_protocol;
        listen unix://var/run/nginx.sock proxy_protocol;
        location /{
                proxy_pass http://backend;
                proxy_set_header Host    $proxy_protocol_dst;
                send_proxy_protocol on;
                proxy_connect_timeout 10s;
                proxy_read_timeout 10s;
        }
}
```

示意图如下：



经测试无误，浏览器可正常浏览网页。

2.proxy protocol version2

为了在解析 ipv6 地址时更为高效，版本二没有再使用字符串表示地址，而是直接的网络字节序表示。

版本二的描述如下：

数据开头由协议的签名组成，为 12 个字节：

\x0D \x0A \x0D \x0A \x00 \x0D \x0A \x51 \x55 \x49 \x54 \x0A

第13个字节，包含了版本和命令，高位的四个位代表版本，必须是\x02。低位的四个位的意义为：

- \x0 : 本地 : 链接用于无延迟的情况，链接的两端是发送者和接收者。

- \x1 : 代理: 链接建立在节点的中间线路上。用于表示链接原先的端点。接收者必须使用协议里的数据来获取原先的地址。
- 其他 : 其他的类型必须被忽略。

第14个字节包含了链接的类型。高四位包含了address family, 低四位包含了协议类型。

Address family类型有：
- 0x0 : AF_UNSPEC : 用于发送未知的协议类型，发送方必须使用这种类型当发送local 命令的时候。
- 0x1 : AF_INET : ipv4的地址类型。

- 0x2 : AF_INET6 : ipv6的地址类型.地址是16个字节的网络字节序。

- 0x3 : AF_UNIX : domain unix socket. 地址长度是108个字节。

低四位表示了协议类型
- 0x0 : UNSPEC : 未指定。

- 0x1 : STREAM : SOCK_STREAM protocol

- 0x2 : DGRAM : SOCK_DGRAM protocol

- other 其他为非法，会被忽略。

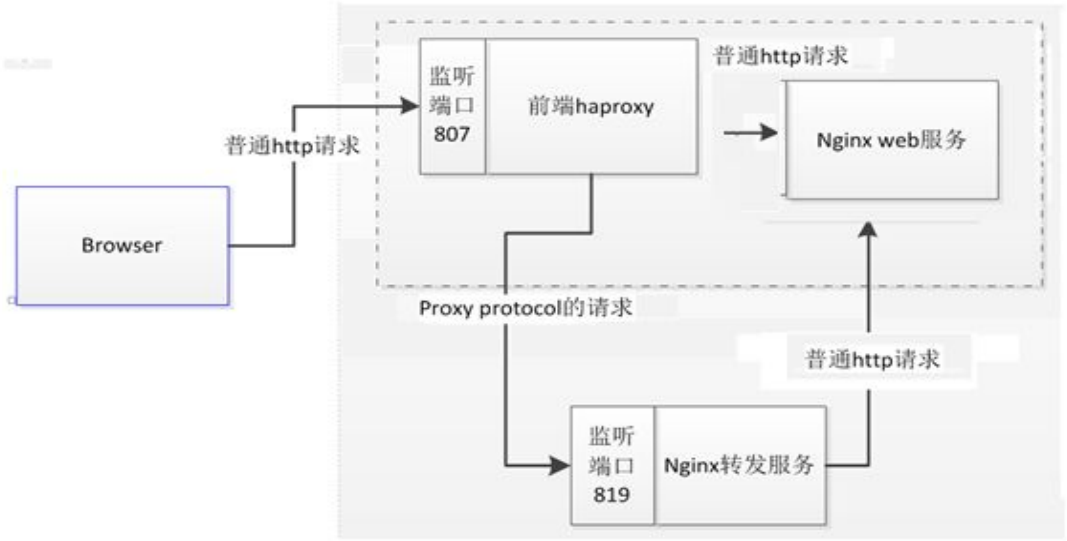
第15和16个字节是后面地址数据的字节数。

从第17个字节开始，是网络字节序的地址信息。
- Layer3 的网络字节序的源地址
- Layer3 的网络字节序的目标地址
- Layer4 的网络字节序的源端口
- Layer4 的网络字节序的目标端口

实验环境描述：

Haproxy版本为 1.5-dev25-2705a61，Nginx版本为：1.7.1。

通信模式选用下图，其中的proxy protocol为version2.



haproxy配置文件应修改：

```
global
daemon
maxconn 256
log 127.0.0.1    local3 info
stats socket /var/run/haproxy.sock mode 600 level admin
stats timeout 2m

defaults
mode http
timeout connect 5000ms
timeout client 50000ms
timeout server 50000ms
#source 0.0.0.0    usesrc clientip

frontend http-in
        bind *:807
        mode http
        option httplog
        log global
        stats enable
        option http-pretend-keepalive
        option forwardfor
        use_backend proxy-out

backend proxy-out
        server http_proxy 127.0.0.1:819 check send-proxy-v2
```

Nginx配置应修改部分：

```
upstream backend {
    server 127.0.0.1:80;
}
server {
    listen *:819    proxy_protocol_v2;
    location /{
        proxy_pass http://backend;
        proxy_set_header Host    $proxy_protocol_dst;
        proxy_connect_timeout 10s;
        proxy_read_timeout 10s;
    }
}
```

以上的通信在虚拟机上抓包为：

浏览器的请求：

```
GET / HTTP/1.1
Host: 192.168.37.167:807
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
If-Modified-Since: Tue, 03 Jun 2014 04:22:39 GMT
If-None-Match: "538d4d8f-14a"
Cache-Control: max-age=0
```

经由haproxy处理后变为proxy protocol version2的数据形式：

```
.
QUIT
!.....%...%..A.'GET / HTTP/1.1
Host: 192.168.37.167:807
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
If-Modified-Since: Tue, 03 Jun 2014 04:22:39 GMT
If-None-Match: "538d4d8f-14a"
Cache-Control: max-age=0
X-Forwarded-For: 192.168.37.154
```

```
e5 98 0d 0a 0d 0a 00 0d  0a 51 55 49 54 0a 21 11   ..⋯⋯ .QUIT.!.
00 0c c0 a8 25 9a c0 a8  25 a7 e0 41 03 27 47 45   ....%... %..A.'GE
54 20 2f 20 48 54 54 50  2f 31 2e 31 0d 0a 48 6f   T / HTTP /1.1..Ho
73 74 3a 20 31 39 32 2e  31 36 38 2e 33 37 2e 31   st: 192. 168.37.1
36 37 3a 38 30 37 0d 0a  55 73 65 72 2d 41 67 65   67:807.. User-Age
6e 74 3a 20 4d 6f 7a 69  6c 6c 61 2f 35 2e 30 20   nt: Mozi lla/5.0
28 57 69 6e 64 6f 77 73  20 4e 54 20 36 2e 31 3b   (Windows  NT 6.1;
20 57 4f 57 36 34 3b 20  72 76 3a 32 39 2e 30 29    WOW64;  rv:29.0)
20 47 65 63 6b 6f 2f 32  30 31 30 30 31 30 31 20    Gecko/2 0100101
46 69 72 65 66 6f 78 2f  32 39 2e 30 0d 0a 41 63   Firefox/ 29.0..Ac
63 65 70 74 3a 20 74 65  78 74 2f 68 74 6d 6c 2c   cept: te xt/html,
```

该份数据再转发给nginx处理之后输出正常的http数据包到web服务器：

```
GET / HTTP/1.0
Connection: close
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
If-Modified-Since: Tue, 03 Jun 2014 04:22:39 GMT
If-None-Match: "538d4d8f-14a"
Cache-Control: max-age=0
X-Forwarded-For: 192.168.37.154
```

与原始的请求相比缺少了host字段，因为配置文件中设置了proxy protocol v1才有的数据，可以更正为其他字段。浏览器浏览无误。

代码修改解释：

在配置文件中添加bind 部分的proxy_protocol_v2命令。

添加之后把之前代码中的proxy_protocol 标志增加一个位，数值为2时代表version2。

```
struct ngx_http_addr_conf_s {
    /* the default server configuration for this address:port */
    ngx_http_core_srv_conf_t  *default_server;

    ngx_http_virtual_names_t  *virtual_names;

#if (NGX_HTTP_SSL)
    unsigned                   ssl:1;
#endif
#if (NGX_HTTP_SPDY)
    unsigned                   spdy:1;
#endif
    unsigned                   proxy_protocol:2;
};
```

在version1的基础上，当为version2的时候调用version2相应的函数进行处理。

```
u_char *
ngx_proxy_protocol_parse(ngx_connection_t *c, u_char *buf, u_char *last, unsigned version )
{

    if (version == 1)
    {
        return ngx_proxy_protocol_parse_v1(c,buf,last);
    }else if (version == 2)
    {
        return ngx_proxy_protocol_parse_v2(c,buf,last);
    }
    return NULL;
}
```

在 return ngx_proxy_protocol_parse_v2(c,buf,last) 函数中。
先进行 proxy protocol version 2 的签名的匹配：
u_char sig[12] = { 0x0D, 0x0A, 0x0D, 0x0A, 0x00, 0x0D, 0x0A, 0x51, 0x55, 0x49, 0x54, 0x0A};
int flag = memcmp(buf,sig,12);
同时添加数据结构：

```
struct proxy_hdr_v2 {
    uint8_t sig[12];    /* hex 0D 0A 0D 0A 00 0D 0A 51 55 49 54 0A */
    uint8_t ver;           /* protocol version and command */
    uint8_t fam;           /* protocol family and address */
    uint16_t len;         /* number of following bytes part of the header */
};


union proxy_addr {
    struct {              /* for TCP/UDP over IPv4, len = 12 */
        uint32_t src_addr;
        uint32_t dst_addr;
        uint16_t src_port;
        uint16_t dst_port;
    } ipv4_addr;
    struct {              /* for TCP/UDP over IPv6, len = 36 */
        uint8_t    src_addr[16];
        uint8_t    dst_addr[16];
        uint16_t src_port;
        uint16_t dst_port;
    } ipv6_addr;
    struct {              /* for AF_UNIX sockets, len = 216 */
        uint8_t src_addr[108];
        uint8_t dst_addr[108];
    } unix_addr;
};


struct proxy_addr_v2{
    union proxy_addr addr;
    uint8_t ver;
    uint8_t fam;
    uint16_t addr_len;
};
```

匹配签名成功之后，解析出地址的长度，之后的地址保存将会十分容易，因为数据已经是网络字节序的编码了，所以调用 memcpy 函数直接拷贝到上面的结构中就可以。

代码段如下：

```
memcpy(&header,buf,sizeof( header ));//拷贝地址之前的数据

header.len = sw16(header.len);//big-endian

len = sizeof(header) + header.len;
                                        //拷贝地址
memcpy(&c->proxy_v2_addr.addr,buf+sizeof(header),header.len);
c->proxy_v2_addr.fam = header.fam;
c->proxy_v2_addr.ver = header.ver;
c->proxy_v2_addr.addr_len = header.len;
```

最后返回普通的 http 协议开始的字节的地址即可。
尚未完成的工作：提供可以在配置文件中使用的变量。