### 代理服务器操作手册

**前言：**

1.蓝色字体表示命令行命令，正式执行时不要复制前面的#号，#号只是提示应该使用root权限操作

2.绿色字体表示注释，有时注释太多就不用绿色表示了

3.注意：本文档的所有操作请先在在测环境进行实践，请不要直接在真实的服务器中操作！

**版权声明**：

本文档以开源的形式发布，所有条款如下：

1. 无担保：作者不保证文档内容的准确无误，亦不承担由于使用此文档所导致的任何后果

2. 自由使用：任何人可以出于任何目的而自由地 阅读/链接/打印/转载/引用/再创作 此文档，无需任何附加条件

若您 阅读/链接/打印/转载/引用/再创作 本文档，则说明接受以上2个条款。

作者：李茂福

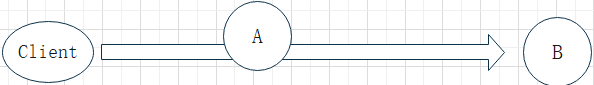
时间：2022-05-01

**章0、代理\_端口转发\_NAT\_等概念**

NAT网络地址转换：这个概念最早用于路由器和防火墙等实体网络设备上，表示路由器/防火墙设备将收到的某数据包的ip地址/传输层端口号进行修改，再走路由转发到目标主机。

NAT最初只是指network address的转换，即ip地址，后来也可以修改传输层的端口号，涉及到端口号的修改的操作也叫作PAT（port address转换），但我们仍习惯于把这二者的修改统称为NAT。 NAT是分2个方向的，对源ip:port的修改叫作sNAT（源地址转换），对目标ip:port的修改叫作dNAT（目的地址转换）

**①sNAT （源NAT）**



\*sNAT是指客户端发来的数据报文是给B的，但在路由上经过了A设备，

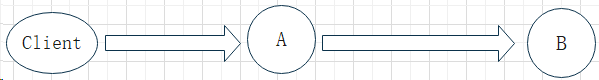
\*然后A对这个流量的源ip:port做了修改，改成了A的ip:port，并且A记录了这个转换的映射关系，

\*B收到数据后，响应的报文是发回给A，因为从B那里看，数据的源ip是A

\*A收到B发回的响应报文后，再根据 之前保存的映射关系，把响应报文的目的ip:port改为Client

\*最后响应报文走路由层 发到Client

**②dNAT （目的NAT、端口映射）**



\*dNAT如果强调对 端口的修改，也可叫作 **端口映射**

\*dNAT是指客户端发来的数据报文是给A的，但A把目的ip:port改为了B的，并记录这个转换的映射关系

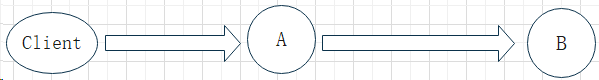
\*B收到数据后，响应的报文是发给Client的，因为之前A并没有修改报文的源ip:port，所以从B那里看，报文就是从Client发来的。

\*B发回的响应报文经路由层转发，如果再经过A时，A再根据之前保存的映射关系，把响应报文的源ip:port改回A自己的

\*响应报文再走路由层发到Client

\*\*\*注意，如果B发回的响应报文经路由转发后，不经过A设备，则源ip:port得不到修改，这样即使Client收到了响应报文，Client也不认这个报文，因为在tcp连接里，要求源ip:port和目的ip:port对得上，不然就不是同一个连接了。

**③端口转发 （port-forward）**



\*端口转发是指客户端发来的数据报文是给A的，但A把源ip:port改为A的，同时也把目的ip:port改为B的，并记录这个映射关系

\*B收到数据后，响应的报文是发给A的，因为从B那里看，收到的报文源ip:port是A的

\*A收到响应报文后，再根据映射关系，修改响应报文的源ip:port为自己，目的ip:port为Client，

\*响应报文再走路由层发到Client

\*\*\*端口转发结合了sNat和dNat，确保了B的响应报文一定发回给A，这样就能根据映射关系修改响应报文的ip:port，使Client收到正确的报文

**先来个小结吧**：

以上三种情况（sNAT、dNAT、端口转发）都只是在路由层面上对数据包的ip:port进行修改，并维护一个映射关系表，进行数据修改的设备A 本身并不处理tcp连接的问题，关于tcp等相关的事宜是由真正的Client和目标B来处理。A设备上是要求开启路由转发的功能。A的操作系统里的普通进程并不和这些转发的数据打交道，比如转发的如果是tcp层的数据，则在A的系统里用netstat命令看不到tcp连接情况

以上三种情况一般分别用于以下场景：

sNAT：用于家庭用户的出口网关，因为随着ipv4资源的耗尽，以及网民数量的增加，不可 能给每个用户分配一个公网ip，所以给用户分配的是内网ip，然后内网数据发往公网时， 要经过出口网关，这时出口网关就是上面例子中的A设备，把源ip:port进行修改，改 为A的公网ip:port，使得数据能在公网中传输。

dNAT：处于内网的用户可能有一些服务器，要给公网上的用户提供服务，但他们没有公网 ip，或者只有少数几个公网ip，所以得充分利用这些公网ip资源，对目标ip:port进行 修改，映射到内网的某服务器的某端口上，也就是访问同一外网ip的不同port，映射 到不同的内网服务器上。

**端口转发**:适用于隐藏客户端或服务端的真实ip，使用某个中介进行转发，以突破各方面对 ip的限制。

根据报文的源ip:port及目标ip:port的修改情况 可分为3种情况：

1.**只修改源ip:port**，即sNAT源地址转换，其中ip:port可以只改ip，也必须改ip

响应报文一定原路返回

2.**只修改目标ip:port**，即dNAT目的地址转换，其中ip:port可以只改其一，必改其一

目标的响应报文不一定原路返回，所以Client那边会得不到正确的响应，因为报文如果

不经过中间的A（修改报文ip:port的始作俑者），则返回时的报文ip:port得不到修改，

所以client认为此响应报文不是正确的，所以要确保响应报文在路由上能经过A

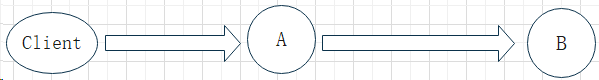
3.**修改源ip:port以及修改目标ip:port**，即端口转发，源中的ip:port可以只改ip，也必改ip

目标ip:port可以只改其一，即改ip或port都行，必改其一

响应报文一定原路返回

**④端口代理 （port-proxy）**

端口代理的用途同端口转发，但实现原理不同



\*Client访问A的某TCP端口，建立了tcp连接，然后A再和B建立tcp连接，这2个连接也 是做了映射关系的记录的

\*Client向A发送的tcp载荷数据，由A再发给B

\*B响应的数据是发给A的，A收到后再把响应数据发给Client

和端口转发不同的是：

\*端口代理时A转发的只有载荷数据，不包括tcp的控制消息，比如建立连接，维持连接和 关闭连接的 相应控制报文。即在A上要监听某个tcp端口，要向系统的tcp/ip协议 栈请求端口资源，收到client的报文后，并不修改其源/目的ip:port，而是另外再去 和B建立tcp连接。

\*client和A之间的tcp连接 并不影响 A和B之间的tcp连接。即使Client和A的连接中断 了，A和B之间也有可能仍在连接。

\*也就是说A上面要有运行某个普通的进程去监听要转发的端口，当此端口有客户端连接时， 再启用某本地端口和B建立一个tcp连接。

当有多个客户端和A的监听的端口连接时，A就要和B建立多个tcp连接

\*与端口转发相比，端口代理的开销较大，对A的系统本身的资源有一定影响

**Linux**上一般是在防火墙进程里启用端口转发（port-forward），系统本身不用监听要转发的 tcp端口，不处理tcp连接的事情，用netstat命令看不到监听端口，系统的防火墙也不 用再另外允许此端口入站，因为它只是过路包，不入站

**Windows**上一般是在系统里使用svchost.exe进行端口代理（portProxy），系统要监听 要代 理的tcp端口，处理tcp连接的相关事宜，用netstat命令可见监听的端口，系统的防火 墙要另外指明允许此端口入站

**http(s)/socks** **代理**（一般指浏览器/web服务器的代理），分2种方向:

**⑤正向代理：**

\*Client访问的目标不管是谁（只要符合走代理的规则），就统统把流量发给代理服务器，让 代理帮Client去访问目标服务器，

\*Client到代理之间的（http(s)/tcp/udp）流量是封装在代理协议层之上的，也就是说套了一 层代理协议的壳，

\*Client知道自己是把流量发给了代理服务器

\*代理服务器收到流量后，再解开这个壳，得到（http(s)/tcp/udp）流量，再根据此流量里的 相关信息（如域名）去找目标服务器的ip，最后代理再把流量发给目标服务器。

\*目标服务器响应的报文是发给代理服务器，

\*代理收到响应报文后再根据映射关系，把响应报文里的数据发给Client

**⑥反向代理：**

\*Client就正常访问目标服务器B，Client并不知道自己访问的是代理，

\*结果B是一个反向代理服务器（比如Nginx）它把收到的client发来的流量再进行某些修改 （修改Http的报头字段，当然也可不修改），再把数据发给后面的真实服务器

\*后面的真实服务器的响应报文是发给反向代理，

\*反向代理收到响应报文后再根据映射关系，修改某些字段，

\*最后把响应报文里的数据发给Client

关于http/https/socks的代理，代理服务器A不仅要处理tcp的相关事宜，还可能要处理高层协议的事情，比如http层的报文过滤，报头修改等，当然有时域名的解析也是由代理服务器去做的。

**小结**：

以上几种情况，凡是带有“代理”2个字的，都是要由代理服务器/进程（即A设备）去处理tcp连接相关的事情，client和A的tcp连接建立后，A再和目标B建立连接，然后将这2个连接做个映射关系，在这2个连接里传输载荷数据，当然，这个载荷数据也可能会被A修改某些字段。若某个连接断开了，A会尽快断开另一连接。

**章一、HaProxy（反向代理https,tcp）**

HAproxy软件常应用于**反向代理**场景。仅支持http(s)和tcp，无缓存，不支持udp。

可直接yum安装haproxy软件包

# yum install haproxy

# systemctl enable haproxy

# systemctl start haproxy

★HAProxy的主配置文件为/etc/haproxy/haproxy.cfg ，默认内容如下：

global

log 127.0.0.1 local2 #定义haproxy日志输出设置，1~7

chroot /var/lib/haproxy #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

pidfile /var/run/haproxy.pid

maxconn 4000 #最大连接数

user haproxy

group haproxy

daemon #让haproxy以守护进程的方式工作于后台

stats socket /var/lib/haproxy/stats #定义统计信息保存位置

nbproc 1 #设置启动服务时可创建的进程数，应小于等于cpu核数

defaults

mode http #代理类型(7层代理http，4层代理tcp)

log global #引入global定义的日志格式

option httplog #日志类别为http日志格式

option dontlognull #如果产生了一个空连接，则此连接的日志将不会记录

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 #haproxy会把客户端的IP信息发送给后端服务器，在HTTP请求中添加X-Forwarded-For字段

option redispatch #当与后端服务器的会话失败时，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

retries 3 #3次连接失败就认为服务器不可用

timeout http-request 10s #超时设置

timeout queue 1m

timeout connect 10s #haproxy与后端服务器连接超时时间

timeout client 1m #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 #haproxy与后端服务器非活动连接的超时时间

timeout http-keep-alive 10s

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

**#以下为默认的一个示例，正式使用时可删除**

frontend main \*:5000 #定义一个名为main的前端监听器，监听所有ip的5000端口

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 #匹配到上面的规则就走名为static的后端

default\_backend app #默认反向到名为app的后端

backend static #名为static的一个后端配置

balance roundrobin #后端服务器的负载均衡模式

server static 127.0.0.1:4331 check #server用来定义多台后端真实服务器,不能用于defaults和frontend部分,格式为 server name address:port param\*

backend app #名为app的一个后端配置

balance roundrobin #后端服务器的负载均衡模式

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

#

**★其他参数说明：**

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**

|  |  |
| --- | --- |
| **roundrobin** | **基于权重进行的轮询**算法，在服务器的性能分布经较均匀时这是一种最公平的算法 |
| 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 |
| **source** | 是**基于请求源IP**的算法，此算法对请求的源IP进行hash运算，然后将结果与后端服务器的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 |
| **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 |
| 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相除，最后转发到某台匹配的后端服务器上 |
| uri\_param | 椐据URL路径中的参数进行转发，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 |
| hdr | 根据http头进行转发，如果指定的http头名称不存在，则使用roundrobin算法进行策略转发 |
| rdp-cookie(name) | 根据cookie(name)来锁定并哈希每一次TCP请求 |

**backend的server的参数：**

server static 127.0.0.1:4331 check #这个check就是其中一个参数，其他参数如下

|  |  |
| --- | --- |
| 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 | 设置将故障状态转为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为不可用状态需要检查的次数，如 fall 3 表示3次检查失败就认为此服务器不可用 |
| cookie xx | 为指定的后端服务器设定cookie值，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 |
| 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 | 设定每个后端server进程可接受的最大并发连接数 |
| backup | 设置后端真实服务器的备份服器，**仅仅**在后端所有真实服务器均不可用的情况下才启用 |

**应用示例1: https**

frontend https\_testxx\_com

bind \*:443 ssl crt /etc/testxx.com.pem #这个pem文件含crt及key 2个文件

mode http

option httpclose

option forwardfor

acl is\_xxx\_com hdr\_beg(host) xxx.com #匹配域名

use\_backend web\_ser\_testxx\_com if is\_xxx\_com #匹配上的流量使用指定的后端

default\_backend web\_ser\_testxx\_com #没匹配的默认走这个后端

backend web\_ser\_testxx\_com

mode http

balance source

option httpchk /index.html

#option后面可指定httpchk,smtpchk,mysql-check,pgsql-check,ssl-hello-chk

cookie SERVERID insert indirect nocache

server s1 192.168.1.11:80 check cookie s1 weight 2 inter3000 maxconn 1000

server s2 192.168.1.12:80 check cookie s2 weight 1 inter 3000 rise 2 fall 3

**应用示例2: tcp**

frontend tcp\_8443

bind \*:8443,\*:7443,\*:9999 #前端可监听多个端口

mode tcp

default\_backend xx\_serv\_8443\_to\_443

backend xx\_serv\_8443\_to\_443

mode tcp

balance source

server s1 192.168.1.11:443 weight 1 check inter 2000 rise 2 fall 3

server s2 192.168.1.12:443 weight 1 check inter 2000 rise 2 fall 3

**监控页面：**

listen admin\_stats

stats enable

bind \*:8080 #监听的ip及端口号

mode http

option httplog

log global

maxconn 10

stats refresh 10s #统计页面自动刷新时间

stats uri /admin #访问的uri http://ip:8080/admin

#stats realm haproxy #haproxy字符串是用户在登录监控页面时提示的信息

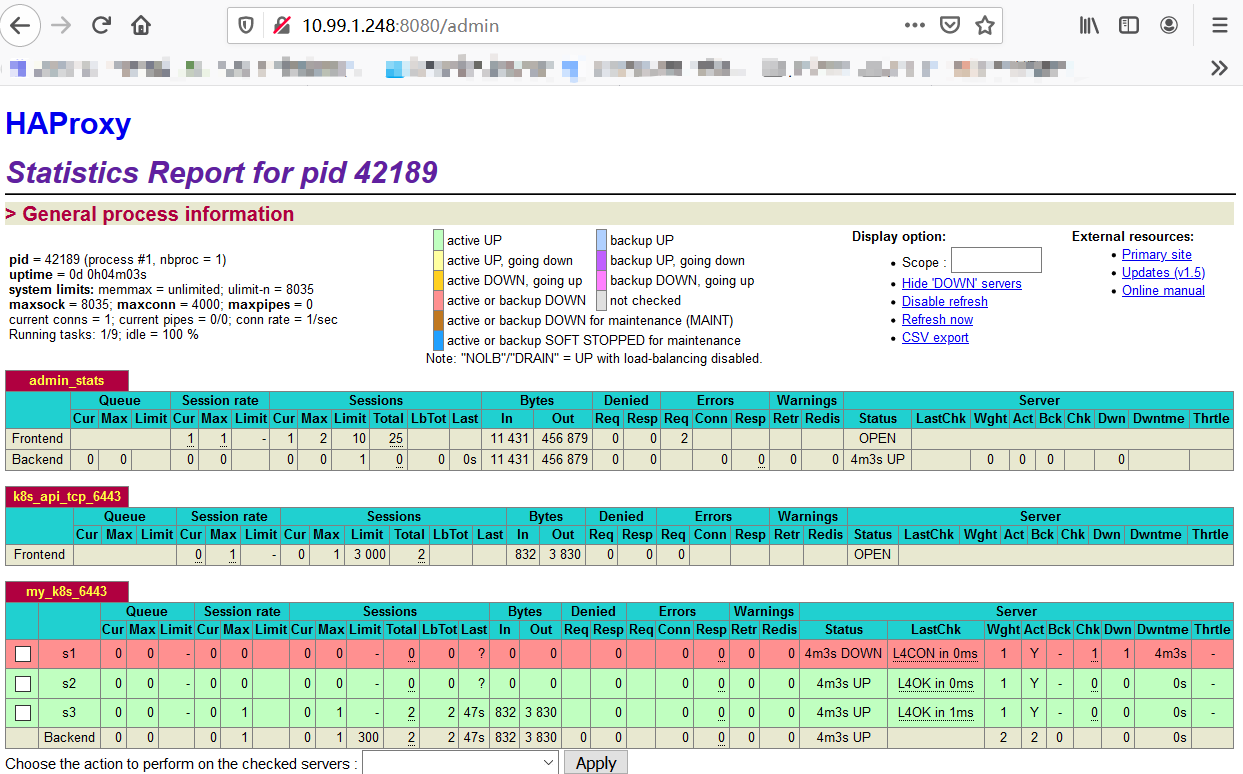
stats realm Global\ haproxy #字符串是在监控页面上提示的信息

stats auth admin:passwdxx #认证用户名和密码，可有多行设置多个用户

stats hide-version #隐藏HAProxy的版本号

stats admin if TRUE #如果认证成功了，可通过webui管理节点

#



**章二、keepalived（vrrp, lvs）**

keepalived一般可以做vrrp功能，在多台服务器上虚拟出一个ip来，当然，它也可以做负载均衡服务器（LVS是Linux Virtual Server的简称，常指负载均衡服务）

# yum install keepalived

# systemctl enable keepalived

默认配置文件为/etc/keepalived/keepalived.conf ，内容如下：

global\_defs **{**

notification\_email { #故障发生时给下面的邮箱发邮件通知

acassen@firewall.loc

sysadmin@firewall.loc

}

notification\_email\_from Alexandre.Cassen@firewall.loc #通知邮件从这个邮箱发出

smtp\_server 192.168.200.1

smtp\_connect\_timeout 30

router\_id LVS\_DEVEL

vrrp\_skip\_check\_adv\_addr

vrrp\_strict

vrrp\_garp\_interval 0

vrrp\_gna\_interval 0

**}**

vrrp\_instance VI\_1 **{** **#VRRP实例，创建VIP**，名为VI\_1

state MASTER #角色，可为MASTER，BACKUP

interface eth0 #绑定的网口，从这个网口发vrrp报文，要允许224.0.0.18入站

virtual\_router\_id 51 #vrid全局唯一，同一个个vrrp服务器组的vrid是一样的

priority 100 #默认100，越高越优先

advert\_int 1 #默认hello时间1秒

authentication { #认证，可选

auth\_type PASS #认证方式为密码

auth\_pass 1111 #认证密码只取前8个字符

}

virtual\_ipaddress {

192.168.200.18 #宣告的vip

}

vrrp\_script chk\_serv { #健康检查

script /path/to/check.sh #一句指令或者一个脚本文件，需返回0(成功)或非0(失败)，keepalived以此为依据判断其监控的服务状态

interval 1 #健康检查周期，秒

weight -10 #如果script中的指令执行失败，那么将优先级减少10

}

notify\_master /path/to\_master.sh #notify\_master表示切换为主机时执行的脚本

notify\_backup /path/to\_backup.sh #notify\_backup表示切换为备机时的脚本

notify\_fault /path/fault.sh #notify\_fault表示出错时执行的脚本

notify /path/notify.sh #notify表示任何一状态切换时都会调用该脚本，且在以上三个脚本执行完成之后进行调用

smtp\_alert #开启邮件通知（用全局区域的邮件设置来发通知）

nopreempt #非抢占模式，假如当前是低优先级的服务器有了vip，高优先级的服务器不会接管它

preempt\_delay 300 #高优先级服务器延迟接管资源（VIP/Route信息等），前提是没有nopreempt选项

**}**

virtual\_server 10.99.1.248 6443 **{** **#LVS负载均衡**

delay\_loop 5 #延迟轮询时间，秒

lb\_algo sh #负载均衡算法，可指定：sh, wrr, rr, lc

lb\_kind DR #常用DR，NAT，FNAT

persistence\_timeout 60 #会话保持时间，秒

protocol TCP #支持UDP

real\_server 10.99.1.51 6443 { #后端服务器，可有多组

weight 1 #权重，默认1，0为失效

TCP\_CHECK {

connect\_timeout 10 #连接超时

nb\_get\_retry 3 #重连次数

delay\_before\_retry 3 #重连间隔

connect\_port 6443 #健康检查的端口

}

}

**}**

virtual\_server 192.168.200.100 443 **{** **#LVS负载均衡**

delay\_loop 6

lb\_algo rr

lb\_kind NAT

persistence\_timeout 50

protocol TCP

real\_server 192.168.201.100 443 { #后端服务器，可有多组

weight 1

SSL\_GET { # HTTP\_GET 或 SSL\_GET

url { #检查url，可以指定多个

path /

digest ff20ad2481f97b1754ef3e12ecd3a9cc #用genhash算出的摘要信息

status\_code 200 #检查的http状态码，HTTP\_GET有效

}

connect\_timeout 3 #超时时长

nb\_get\_retry 3 #重试次数

delay\_before\_retry 3 #下次重试的时间延迟

}

}

**}**

#

**LVS原理讲解**

**LVS的4种工作模式：**

|  |  |
| --- | --- |
| NAT | dNAT，修改报文的目标ip为后端的ip，后端可有多台；后端响应的报文不一定经过lvs服务器了（也可修改报文的目标port）源ip:port不变 |
| DR | 封装新的MAC地址，**仅修改报文的目标mac地址**为后端服务器的mac，要求后端服务器与当前lvs服务器处于同一物理网络(vlan)，且后端服务器上要绑定有报文的目标ip，响应报文不一定经过lvs服务器。为了不让后端服务器的lo上的vip响应arp，/etc/sysctl.conf得配置以下2行：  net.ipv4.conf.lo.arp\_ignore = 1 ，net.ipv4.conf.all.arp\_ignore = 1 |
| TUN | 在原请求IP报文之外新加一个IP首部（源ip为lvs服务器的，目的ip为后端服务器的ip），后端收到报文后发现是自己的IP地址，就会将报文接受下来，拆除最外层的IP后，会发现里面还有一层IP首部，而且目标地址是自己的lo接口VIP，然后后端服务器开始处理此请求，处理完成后，通过lo接口发出响应。此时的源IP地址为VIP，目标IP为client ip |
| FullNAT | 修改请求报文的源和目标IP |

**LVS调度算法：**

|  |  |
| --- | --- |
| rr | RoundRobin 均衡轮询 |
| wrr | 加权轮询，调度器可以给后端主机指定权重 |
| lc | 最少连接 |
| wlc | 加权最少连接 |
| lblc | 基于局部性最少连接，调度算法是针对目标IP地址的负载均衡，目前主要用于Cache集群系统。该算法根据请求的目标IP地址找出该目标IP地址最近使用的服务器，若该服务器 是可用的且没有超载，将请求发送到该服务器；若服务器不存在，或者该服务器超载且有服务器处于一半的工作负载，则用"最少链接"的原则选出一个可用的服务 器，将请求发送到该服务器。 |
| lblcr | 带复制的基于局部性最少连接，它与LBLC算法的不同之处是它要维护从一个 目标IP地址到一组服务器的映射，而LBLC算法维护从一个目标IP地址到一台服务器的映射。该算法根据请求的目标IP地址找出该目标IP地址对应的服务 器组，按"最小连接"原则从服务器组中选出一台服务器，若服务器没有超载，将请求发送到该服务器，若服务器超载；则按"最小连接"原则从这个集群中选出一 台服务器，将该服务器加入到服务器组中，将请求发送到该服务器。同时，当该服务器组有一段时间没有被修改，将最忙的服务器从服务器组中删除，以降低复制的程度。 |
| dh | 目标地址哈希，使用场景是正向代理中的负载均衡 |
| sh | 源IP地址hash，实现session sticky |
| sed | 最短延迟调度（Shortest Expected Delay ） |
| nq | 永不排队/最少队列调度（Never Queue）无需队列，如果有台 realserver的连接数＝0就直接分配过去，不需要再进行sed运算，保证不会有一个主机很空间。在SED基础上无论+几，第二次一定给下一个，保证不会有一个主机不会很空闲着，不考虑非活动连接（如dns/udp），才用NQ。SED要考虑活动状态连接，如httpd等处于保持状态的服务就需要考虑非活动连接给服务器的压力。 |

**LVS常见术语：**

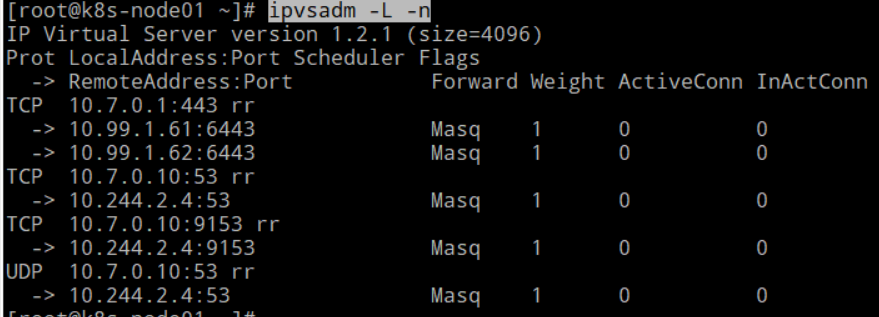
|  |  |
| --- | --- |
| ipvsadm | 用户空间的命令行工具，用于管理集群服务及集群服务上的RS等 |
| IPVS | IP Virtual Server工作于内核上的netfilter INPUT HOOK之上的程序，可根据用户定义的集群实现请求转发 |
| VS | Virtual Server ，虚拟服务 |
| Director/Balancer | LVS负载均衡器、分发器 |
| RS | Real Server 后端服务器 |
| CIP | Client IP 客户端IP |
| DIP | Director IP 负载均衡器IP |
| RIP | Real Server IP 后端服务器IP |

**章三、ipvsadm**

ipvsadm是一个命令行工具，用于管理LVS的策略规则

# yum install ipvsadm

# ipvsadm -L -n # -L(--list)查看lvs策略，-n表示以数字形式显示端口号



# ipvsadm --save -n # --save(-S)保存策略到标准输出

# ipvsadm --save -n > ipvs.save #保存到文件

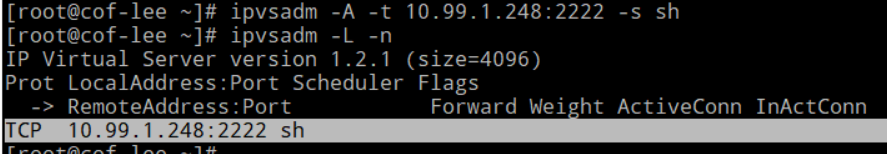
# ipvsadm --clear #删除所有策略，同-C

# ipvsadm --restore < ipvs.save #从标准输入恢复策略，同-R

# ipvsadm -A -t 10.99.1.248:2222 -s sh #添加一个集群服务，-A同--add-service

-t同--tcp-service表示TCP协议，后面紧跟本地ip:port，-u(--udp-service)表示UDP协议

-s同--scheduler表示调度算法，默认为wlc，可指定为rr, wrr, lc, wlc, sh, dh, sed, nq, lblc, lblcr



# ipvsadm -E -t 10.99.1.248:2222 -s rr #修改集群服务的调度算法为rr (-E同--edit-service)

# ipvsadm -D -t 10.99.1.248:2222 #删除一个集群服务 （-D同--delete-service）

# ipvsadm -a -t 10.99.1.248:2222 -r 10.99.1.61:22 -m -w 1 #-r(--real-server)往集群服务中添加一个后端真实服务器

# ipvsadm -a -t 10.99.1.248:2222 -r 10.99.1.62:22 -m -w 1 #-m同--masquerading（dNAT模式），-w同--weight权重

其他lvs负载均衡模式： --gatewaying(-g)表示DR模式，默认，--ipip(-i)表示TUN模式

# ipvsadm -e -t 10.99.1.248:2222 -r 10.99.1.61:22 -m -w 2 #修改后端真实服务器的权重

# ipvsadm -d -t 10.99.1.248:2222 -r 10.99.1.61:22 #删除集群服务中的一个后端服务器

（-e同--edit-server，-d同--delete-server）

**章四、nginx（反向代理https,tcp,udp）**

Centos7自带的yum repo仓库没有nginx安装包，所以要添加nginx的yum repo源文件，

先去官网看看repo的文件名称

http://nginx.org/packages/centos/7/noarch/RPMS/ 这里找nginx-release-centos-7-xxx.rpm的文件，并复制链接，再在centos7系统里安装此rpm包

# rpm -ivh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx.noarch.rpm

然后在 /etc/yum.repos.d/目录下多了一个nginx.repo仓库源文件

如果安装不了这个rpm，我们也可以直接创建nginx.repo文件，内容如下：

[nginx-stable]

name=nginx-stable-repo

baseurl=http://nginx.org/packages/centos/7/$basearch/

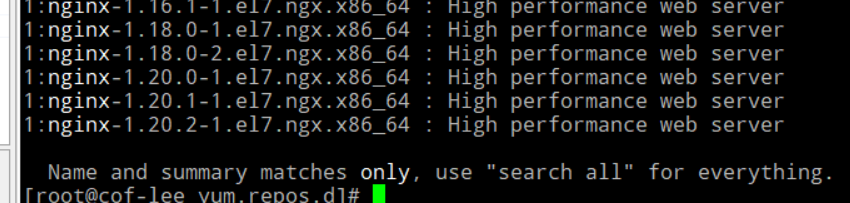
gpgcheck=1

enable=1

gpgkey=https://nginx.org/keys/nginx\_signing.key

保存

# yum search nginx --showduplicates



# yum install nginx #默认安装最新版本，可指定需要的版本

# yum install nginx-1.20.2-1.el7.ngx.x86\_64 #安装指定版本的nginx

# nginx -v #查看nginx版本

# nginx -t #检查默认配置是否有语法错误

★使用yum安装的nginx，建议只使用systemctl命令去管理nginx服务

程序路径：/usr/sbin/nginx 主配置文件：/etc/nginx/nginx.conf

配置讲解：

# cat /etc/nginx/nginx.conf

user nginx;

worker\_processes auto;

error\_log /var/log/nginx/error.log notice;

pid /var/run/nginx.pid;

events {

worker\_connections 1024; #最大连接数

}

http **{**

include /etc/nginx/mime.types; #文件扩展名与文件类型映射表

default\_type application/octet-stream; #默认文件类型，字节流

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

# 定义一个名为main的日志格式

access\_log /var/log/nginx/access.log main; #访问日志文件及使用的日志格式

sendfile on; #允许sendfile方式传输文件，默认为off

#tcp\_nopush on;

keepalive\_timeout 65; #连接超时时间，默认为75s

#gzip on;

include /etc/nginx/conf.d/\*.conf; #加载此目录下的其他\*.conf配置文件

**}**

★**它会加载/etc/nginx/conf.d/default.conf配置，内容如下**：

# cat default.conf | grep -v '#' #里面是一个示例站点

server **{** #server这段是加载到 http {} 模块里的

listen 80; #监听80端口

server\_name localhost; #http的server\_name

location / { #针对 / 这个路径的配置

root /usr/share/nginx/html; #指定此url所在根目录

index index.html index.htm; #默认index页面文件

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html { #针对/50x.html这个url的配置

root /usr/share/nginx/html;

}

**}**

#

**nginx做TCP反向代理示例：**

stream **{** #该模块与 http{}同级，不在http{}里

upstream tcp\_xx\_serv {

#tcp不能指定调度算法，默认weight，若后端有故障可主动移除

server 10.99.1.61:22 weight=1 max\_fails=2 fail\_timeout=8; #后端服务器

server 10.99.1.62:22 weight=1 max\_fails=2 fail\_timeout=8;

} #6秒内共失败2次则此后端不可达（每次失败超时是由

server { # proxy\_connect\_timeout决定的）

listen 2222; #监听本服务器上的2222/tcp

proxy\_pass tcp\_xx\_serv; #反向代理到名为tcp\_xx\_serv的upstream

proxy\_connect\_timeout 4; #默认60，最大75秒，本代理与后端服务的连接超时

}

**}**

#

**nginx做UDP反向代理示例：**

stream { #该模块与 http{}同级，不在http{}里

upstream udp\_xx\_serv {

#udp不能指定调度算法，默认weight，若后端有故障也不能主动移除

server 10.99.1.61:522; #后端服务器

server 10.99.1.62:522;

}

server {

listen 5222 udp; #监听本服务器上的5222/udp

proxy\_pass udp\_xx\_serv; #反向代理到名为udp\_xx\_serv的upstream

}

}

**nginx做https反向代理示例：**

upstream lb\_xxx { #位于http{}模块里

ip\_hash; #调度算法，其他有的 weight, fair, url\_hash（需加载相应模块支持）

server 10.99.1.61:480 weight=1 max\_fails=2 fail\_timeout=8; #后端服务器

server 10.99.1.61:480 weight=1 max\_fails=2 fail\_timeout=8;

}

server **{** #位于http{}模块里

listen 8822 ssl; #监听8822 ssl端口，防火墙开放8822/tcp端口

server\_name vtest.xxx.com; #域名

ssl\_certificate /etc/test/vtest.xxx.com.crt; #证书文件

ssl\_certificate\_key /etc/test/vtest.xxx.com.key; #key文件

ssl\_session\_cache shared:SSL:10m;

ssl\_session\_timeout 10m; #客户端可重用ssl会话的时间，默认5分钟

keepalive\_timeout 75 75; #保持http连接的超时（第2个值75可选，表示返回给客户的http报头添加keepalive字段，这2个值可不一致，一般客户端浏览器不受第2个值的影响）决定服务器是否关闭http连接的是第一个值

client\_header\_timeout 60; #2次GET/POST请求时间间隔不能超过此时间，秒

client\_body\_timeout 60; #否则返回request time out (408)

send\_timeout 60; #客户端与本服务器在60秒内至少读一次

ssl\_ciphers ECDHE-RSA-AES128-GCM-SHA256:HIGH:!aNULL:!MD5;

ssl\_prefer\_server\_ciphers on;

location /xxx { #将此/xxx 路径做反向代理

proxy\_redirect off;

proxy\_pass https://lb\_xxx; #指定后端upstream，协议可为http,https

proxy\_http\_version 1.1;

proxy\_connect\_timeout 4; #默认60，最大75秒，本代理与后端服务的连接超时

proxy\_read\_timeout 60; #本服务与后端在指定时间内至少读一次，秒

proxy\_send\_timeout 60; #本服务与后端在指定时间内至少发一次

proxy\_ssl\_trusted\_certificate /etc/test/back-server-ca.crt; #验证后端证书

proxy\_ssl\_verify on;

proxy\_ssl\_verify\_depth 1;

proxy\_ssl\_session\_resuse on;

proxy\_ssl\_server\_name on;

proxy\_set\_header Host $host; #如果前后端域名相同则设置为原报文的host

#proxy\_set\_header Host *server.xxx.com*; #前后端域名不同则需改host字段

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_intercept\_errors on;

}

location / {

root /test; #服务器的/test目录作为 vtest.xxx.com站点的根目录

index index.html;

}

**}**

#

**章五、httpd（反向代理http）**

# yum install httpd

# systemctl enable httpd

★httpd的配置文件不允许在配置同行末尾加#去注释，只可换行注释！

# cat /etc/httpd/conf/httpd.conf | grep -v '^#' #查看默认配置文件

# ServerTokens OS #告诉客户端本服务器的版本和操作系统

ServerRoot "/etc/httpd" #服务根目录

Listen 80 # Listen ip:port 省略ip时表示监听所有ip

Include conf.modules.d/\*.conf #加载模块配置文件 /etc/httpd/conf.modules.d/\*.conf

User apache

Group apache

ServerAdmin root@localhost #系统管理员邮箱，当出问题时会通知此邮箱

<Directory /> # Directory后面的路径是绝对路径，标签对 里面指定httpd程序可以在这个路径下进行的操作权限

AllowOverride none

Require all denied

</Directory>

**DocumentRoot "/var/www/html"** #默认站点的根目录

<Directory "/var/www"> #标签对 里面指定httpd可以在此目录下进行的操作权限

AllowOverride None

Require all granted #给上面指定的路径授权，授权后才能访问

</Directory>

<Directory "/var/www/html"> #标签对 里面指定httpd可以在此目录下进行的操作权限

Options Indexes FollowSymLinks #2个option， Indexes表示如果在此目录下找不到首页文件，就显示整个目录下的文件外；FollowSymLinks表示让链接文件可以生效，链接如果跳出了此目录，也生效

AllowOverride None #AllowOverride 允许覆盖参数，ALL表示全部的权限均可以覆盖；None表示不可覆盖，让.htaccess文件失效；Limits表示允许用户利用Allow,Deny,Order管理可浏览的权限

Require all granted

</Directory>

# Order allow,deny 表示开放所有，拒绝特定

# Order deny,allow 表示拒绝所有，开放特定

# 如：允许所有人访问，除了192.168.1.1（不让它访问）

# Order allow,deny

# allow from all

# deny 192.168.1.1

<IfModule dir\_module>

DirectoryIndex index.html #网站默认的首页文件名/var/www/html/index.html

</IfModule>

<Files ".ht\*">

Require all denied #.htaccess文件默认被禁止访问

</Files>

ErrorLog "logs/error\_log" #/etc/httpd/logs/ 链接到/var/log/httpd/

LogLevel warn

<IfModule log\_config\_module>

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined

LogFormat "%h %l %u %t \"%r\" %>s %b" common

<IfModule logio\_module>

# You need to enable mod\_logio.c to use %I and %O

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio

</IfModule>

CustomLog "logs/access\_log" combined

</IfModule>

<IfModule alias\_module>

ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"

</IfModule>

<Directory "/var/www/cgi-bin">

AllowOverride None

Options None

Require all granted

</Directory>

<IfModule mime\_module>

TypesConfig /etc/mime.types

AddType application/x-compress .Z

AddType application/x-gzip .gz .tgz

AddType text/html .shtml

AddOutputFilter INCLUDES .shtml

</IfModule>

**AddDefaultCharset UTF-8 #默认字符集**

<IfModule mime\_magic\_module>

MIMEMagicFile conf/magic

</IfModule>

EnableSendfile on

IncludeOptional conf.d/\*.conf #/etc/httpd/conf.d/\*.conf

# Include和IncludeOptional功能相同，都可以包括其它配置文件；但是当无匹配文件时，Include会报错，IncludeOptional会忽略错误

#

# ServerSignature off #错误页不对外显示服务器名字、Apache版本等信息

#要求用户密码验证才能访问的目录

<Directory "/mydir">

Options None

AllowOverride None

AuthType Basic

AuthName "String"

AuthUserFile "/path/to/.htpass" #apache用户对此文件要有读权限

Require user userName1 userName2

</Directory>

#要求用户密码验证或来自192.168.1的客户端都能访问的目录

<Directory "/mydir">

Require valid-user

Allow from 192.168.1

Satisfy Any #Any表示以上条件有一个满足就行，ALL表示以上条件都满足才行

</Directory>

# htpasswd /path/to/.htpass userName1 #往密码文件里添加用户密码

# httpd -t #检查默认配置是否有语法错误

★httpd支持在一台物理主机上实现多个网站（vhost虚拟主机）

**网站的唯一标识：**

IP相同，但端口不同

IP不同，但端口均为默认端口

FQDN不同, IP和端口都相同

**多虚拟主机有三种实现方案：**

基于ip：为每个虚拟主机准备至少一个ip地址

基于port：为每个虚拟主机使用至少一个独立的port

基于FQDN：为每个虚拟主机使用至少一个FQDN，根据客户端请求报文中首部 Host字段值去匹配

**创建虚拟主机**，在/etc/httpd/conf.d/路径下创建单独的vhost-xx.conf文件

Listen 89 #这个属于全局配置，监听\*:89端口

<virtualhost \*:89>

ServerName xxx.com #当监听同端口号的vhost有多个时，根据FQDN区分

documentroot "/data/web89/" #此网站的根目录

CustomLog logs/web89\_access.log combined

<directory "/data/web89"> #根目录需要授权才可访问

require all granted

</directory>

</virtualhost>

#

**httpd反向代理http**

############################### 示例1

<virtualhost \*:89>

ServerName xxx.com

documentroot "/data/web89/"

ProxyPass "/xx" "http://back-server.com/" #客户端访问/xx的uri时，反向代理到后面的地址

ProxyPassReverse "/xx" "http://back-server.com/"

#在没有重定向的情况下，ProxyPassReverse是可以省略的，否则一般情况下应该将其设置为和ProxyPass相同。ProxyPassReverse作用是防止重定向时客户端无法正确访问

CustomLog logs/web89\_access.log combined

<directory "/data/web89">

require all granted

</directory>

</virtualhost>

############################### 示例2

<virtualhost \*:89>

ServerName xxx.com

documentroot "/data/web89/"

<Proxy balancer://backxx>

BalancerMember http://back-serv1:8080 timeout=2 #默认权重1

BalancerMember http://back-serv2:8080 timeout=2 loadfactor=3 #权重为3倍

ProxySet lbmethod=bytraffic #负载均衡算法

</Proxy>

ProxyPass "/xx/" "balancer://backxx/" #路径 /xx/末尾有/斜杠，则浏览器输入时也得带/，否则不会反向代理转发到后端服务器

ProxyPassReverse "/xx/" "balancer://backxx/"

CustomLog logs/web89\_access.log combined

<directory "/data/web89">

require all granted

</directory>

</virtualhost>

############################### 示例3

Listen 89

<virtualhost \*:89>

ServerName xxx.com

documentroot "/data/web89/"

<Proxy balancer://backxx>

BalancerMember http://10.99.1.61:8080 timeout=2

BalancerMember http://10.99.1.248:8822 timeout=2 loadfactor=3

ProxySet lbmethod=bytraffic

</Proxy>

<Location /xx > #也可在Location标签里指定反向代理路径

ProxyPass balancer://backxx

ProxyPassReverse balancer://backxx

</Location>

CustomLog logs/web89\_access.log combined

<directory "/data/web89">

require all granted

</directory>

</virtualhost>

###############################

# httpd有3种负载均衡算法：

byrequests ：默认，基于请求数量计算权重

bytraffic ：基于I/O流量大小计算权重

bybusyness ：基于挂起的请求(排队暂未处理)数量计算权重

#

#如果想让某个子目录不进行反向代理，而是在本地处理。可以设置第二个参数为"!"，如：

ProxyPass "/mirror/foo/i" "!"

ProxyPass "/mirror/foo" "http://back-server.com"

**章六、frp内网穿透反向代理（https,tcp,udp）**

在前面几章的常规反向代理服务器，它们要求本服务能直接访问到后端服务器的ip及端口号，但有时，我们想让它反向代理到我们局域网里的服务器，这些局域网里的服务器可能共用一个不固定的公网ip上网，导致代理服务器无法直接访问到这些后端服务器。这时，可以换种思路，让局域网里的后端服务器主动向 代理服务器发起连接，然后建立一条连接隧道，这样代理服务器可以走隧道访问到后端服务器了。

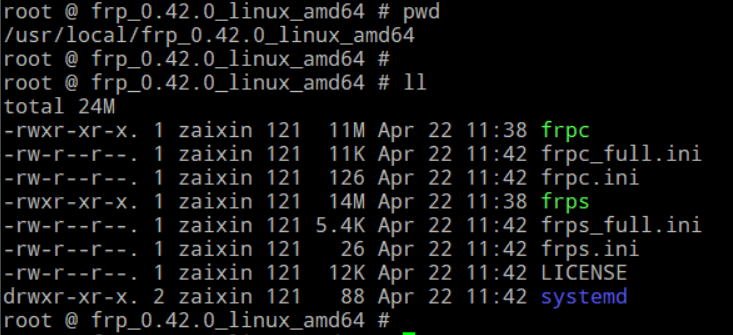
frp（fast reverse proxy）这个开源的工具就可以实现这样的功能，我们一般把这种功能称为 内网穿透反向代理。开源地址： https://github.com/fatedier/frp 始于2016-02

下载linux x86\_64的工具包并传到云端有公网ip的服务器上，局域网内的后端服务器也要上传这个工具包

#假如上传到/usr/local路径下，frp软件包名frp\_0.42.0\_linux\_amd64.tar.gz

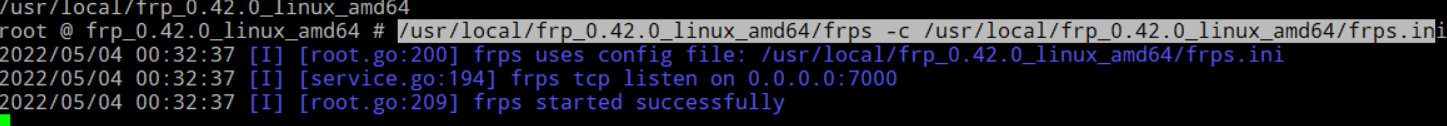
# tar -xvf frp\_0.42.0\_linux\_amd64.tar.gz

# cd frp\_0.42.0\_linux\_amd64 #进入解压目录



解压目录下的frpc和frps为2个可执行文件，即frp的客户端（后端服务器）及frp服务端（有固定公网ip的代理服务器）程序文件，\*.ini文件为配置文件。编辑配置文件，然后执行程序文件时带 -c 参数，后面指定配置文件名即可

# /usr/local/frp\_0.42.0\_linux\_amd64/frps -c /usr/local/frp\_0.42.0\_linux\_amd64/frps.ini &



#命令最后的 & 表示后台运行此程序

**★把frps/frpc做成系统服务，受systemd管理**

解压目录下有个名为systemd的子目录，里面有frps.service和frpc.service文件，修改文件里的ExecStart=配置项以及ExecReload=配置项为实际的文件绝对路径，再复制文件到/usr/lib/systemd/system/路径下或者/etc/systemd/system/路径下都可以，再 systemctl enable frps/frpc就可以使用systemctl命令去管理frps/frpc服务了

**# frps.ini服务端common配置**

[common] #正式的配置文件 不可以在配置行后面写#注释，得换行注释

bind\_addr = 0.0.0.0 #本代理服务器监听的ip

bind\_port = 7501 #监听的端口号

dashboard\_addr = 0.0.0.0 #可选，监控页面监听的地址

dashboard\_port = 7502 #可选，监控页面监听的端口号，http访问

dashboard\_user = adminx #可选，监控页面登录用户名

dashboard\_pwd = passwdxx #可选，监控页面登录密码

log\_file = /usr/local/frp\_0.42.0\_linux\_amd64/frp\_0.42.0.log

log\_level = info

log\_max\_days = 15 #一天一份日志文件，共15天的

authentication\_method = token

token = sx123456xx #局域网里的后端服务器与本服务器建立连接时的验证密码

tls\_only = true #从0.34.0版本开始，启用tls\_only时需要指定ssl证书

tls\_cert\_file = /usr/local/frp\_0.42.0\_linux\_amd64/frps.xxxx.com.crt

tls\_key\_file = /usr/local/frp\_0.42.0\_linux\_amd64/frps.xxxx.com.key

#ssl证书需要有SAN扩展可选名

tls\_trusted\_ca\_file = /usr/local/frp\_0.42.0\_linux\_amd64/frpca.crt

#

**#frpc.ini后端服务器common配置**

[common]

server\_addr = frps.xxxx.com #需要用ssl证书里SAN字段里有的ip/域名

#如果server\_addr为域名，确保客户端本地hosts文件里有解析到服务端公网ip

server\_port = 7501 #服务端common里监听的端口号，后端服务器主动去连接服务端的此端口，这个端口并不是提供给外部客户访问的反向代理端口

token = sx123456xx

tls\_enable = true

tls\_cert\_file = frpc-1.xxxx.com.crt

tls\_key\_file = frpc-1.xxxx.com.key

tls\_trusted\_ca\_file = frpca.crt #CA文件与服务端是同一个，ssl文件是后端服务器自己的，后端服务器与代理服务器的ssl文件都是由这个CA签发的

http\_proxy = http://10.1.1.222:8080 #后端可通过http代理去与服务端建立连接（可选）

★frps的代理服务端配置是固定的，其他具体需要代理哪些端口，反向转发到哪里去，是由frpc后端服务器去写的

**TCP代理示例：**

[sshxx] #代理名称标识，随便取，和[common]同级，直接在frpc.ini末尾添加

type = tcp #如果需要监听udp，把tcp改为udp即可

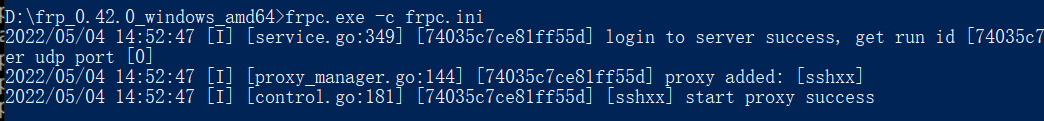
local\_ip = 10.99.1.61 #本局域网里的内网ip，指向frpc自己时可写127.0.0.1

local\_port = 22

remote\_port = 7503 #表示在frps代理服务器上监听此端口，然后外部客户访问frps服务器加7503端口时，流量反向转到本frpc客户端，frpc客户端再反向代理去访问局域网里的local\_ip:local\_port



cmd> frpc.exe -f frpc.ini #运行frpc程序，结果如下图



然后外部客户ssh访问 frps.xxxx.com:7503 时，ssh连接到了10.99.1.61:22这里

**TCP负载均衡示例：**

# frpc-client1.ini

[tcp\_ser1]

type = tcp

local\_ip = 10.99.1.61 #本局域网里的内网ip，指向frpc自己时可写127.0.0.1

local\_port = 22

remote\_port = 7503 #要求frps服务端监听端口相同

group\_key = kxxxxxxy #同一个负载组的key要相同

health\_check\_type = tcp #后端健康检查

health\_check\_timeout\_s = 3

health\_check\_max\_failed = 2

health\_check\_interval\_s = 6

# frpc-client2.ini

[tcp\_ser2]

type = tcp

local\_ip = 10.6.1.62 #本局域网里的内网ip，指向frpc自己时可写127.0.0.1

local\_port = 22

remote\_port = 7503 #要求frps服务端监听端口相同

group\_key = kxxxxxxy #同一个负载组的key要相同

health\_check\_type = tcp

health\_check\_timeout\_s = 3

health\_check\_max\_failed = 2

health\_check\_interval\_s = 6

然后外部客户访问 frps.xxxx.com:7503 时，流量反向转发到局域网里的serv1和serv2

**frpc其他配置：**

端口组映射：

local\_port = 22,600-700

remote\_port = 7503,7600-7700

带宽限制：

bandwidth\_limit = 10MB #MB, KB

Tcp多路复用：

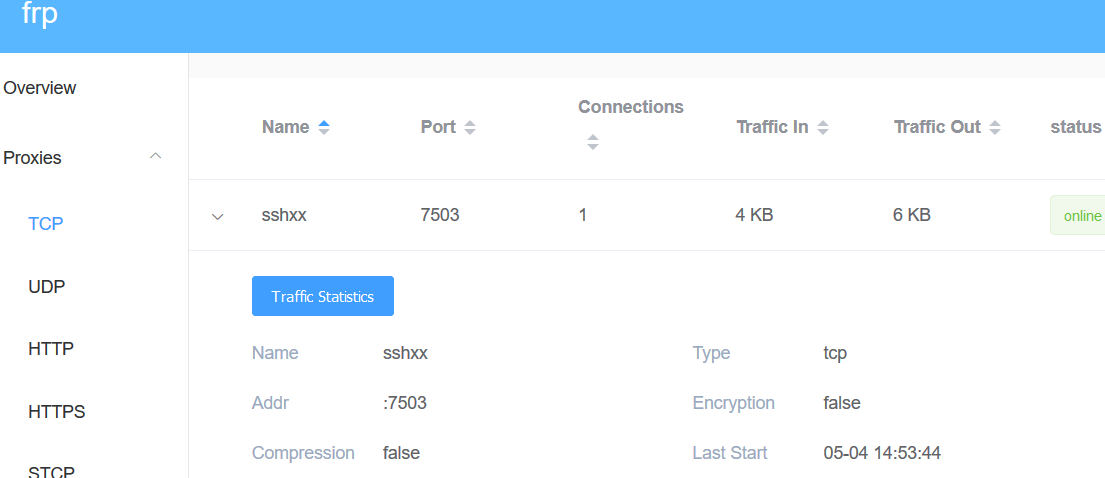
tcp\_mux = true #默认开启，如果不想开，则frps和frpc都得关闭

★frp还支持http(s)的代理，以及p2p,stcp等其他配置，本文档不列举了，可参考官方说明：

https://github.com/fatedier/frp

**frps监控页面：**





**章七、正向代理原理及使用**

**什么是网络代理**？

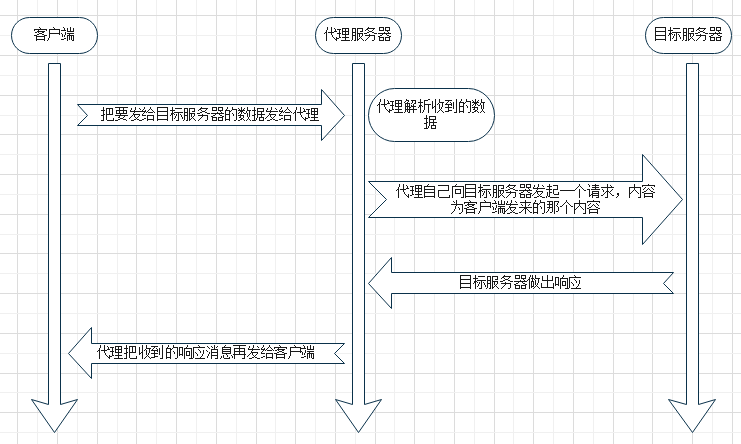
上网这种事原来就只是客户端和服务器之间的事，后来，客户端这边的上网情况变了，可能由于网管做了上网行为的限制，又或是接入运营商做了一些访问限制，导致客户端这边无法访问到某些目标服务器了。不过客户端还能访问其他未被限制的服务器，于是一个不算太难想到的想法出现了：让客户端能正常访问的那个服务器帮我们上网，我（指客户端）把要访问的数据发给它（能访问到的服务器），它再帮我去访问目标服务器，得到结果后再发回给我，我便突破了本地网管/运营商的限制，这种上网方式便是**正向代理上网**。正向代理技术不一定就是这么诞生的，但一定是为了突破上网限制而兴起的。可能是网管的限制，可能是运营商的限制，也可能是服务端对客户端源IP的限制。

最早的上网主要是指访问网站，使用http的协议，于是要代理上网的话也是基于http的代理，后来服务数据可能不走http了，用了ssl或仅基于tcp，于是基于tcp层的socket代理出现，当然也有基于udp层的代理。代理上网技术主要是在传输层及以上的层次，有没有网络层及以下的代理呢，那个可能就不叫代理技术了，而叫VPN技术，本文不做这方面的讲解。正向代理技术主要有三种：HTTP代理、HTTPS代理、socks代理

**HTTP正向代理**：

http协议使用的是明文，早期的网络协议都是用的明文，所以很容易遭到中间人攻击，http的正向代理服务器就充当了这个中间人，正规的http服务器当然不会去随意窃取和篡改我们的信息，所以我们要使用开源的http正向代理服务器。http正向代理的原理是：

在配置了http代理的客户端，它的浏览器在上网时，不会直接把请求的数据包发给目标服务器，而是发给了代理服务器，代理服务器收到数据包后，解析http头部的字段，获取客户端要访问的目标，然后代理服务器自己作为一个客户端向目标服务器发起相同的请求，获得目标服务器返回的结果后，再把数据发给最终的客户端



因为http协议是明文的，所以代理服务器很容易就看到了客户端发来的报文里的内容，客户端发来的Http报文如下（示例）

GET http://www.test.com/ HTTP/1.1

User-Agent: xxxx

Proxy-Connection: Keep-Alive

Connection: Keep-Alive

Host: www.test.com

Cookie: fsfsdfsdfsfsfds

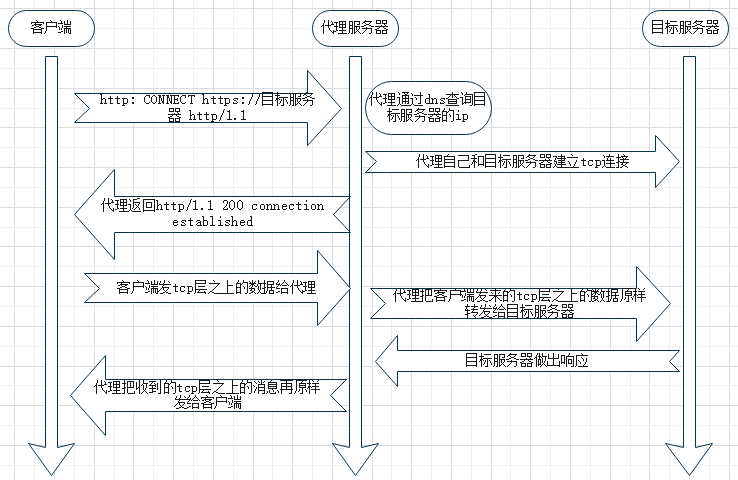
Proxy-Authorization: Basic Y28mOkNlbnRvczxy==\r\n

... ... ...

然后代理服务器知道了客户想向www.test.com（目标服务器）发起get请求，于是代理先查www.test.com的Ip，再向www.test.com发起tcp连接，再发起get请求，目标服务器有回复后，代理再把回复的内容发给客户端。代理服务器做了一个中间人的角色。原理和实现的方法都是比较简单的，http正向代理服务器的功能不止如此，还可以做一些过滤，如过滤目标服务器发来的广告信息和不良信息，以及阻止客户端访问某些内容。http正向代理服务器在帮客户端上网之前可以向客户端索要验证信息，比如用户名和密码，这些信息也是放是http报文的头部某字段里，是明文的，客户端验证通过后才能正常通过代理上网。

**HTTPS正向代理**：

https用到了ssl协议，在ssl层建立了安全连接后，才发起http请求，如果https代理服务器也充当中间人角色的话，它不可能有其他目标服务器的ssl证书及私钥，它若用自己的证书的话，也和其他目标网站不匹配，客户端浏览器就会提示连接不安全。所以一般https代理服务器不会做中间人角色，它只做了tcp层之上的数据转发，和端口转发差不多，只不过多了前面的一些事宜。



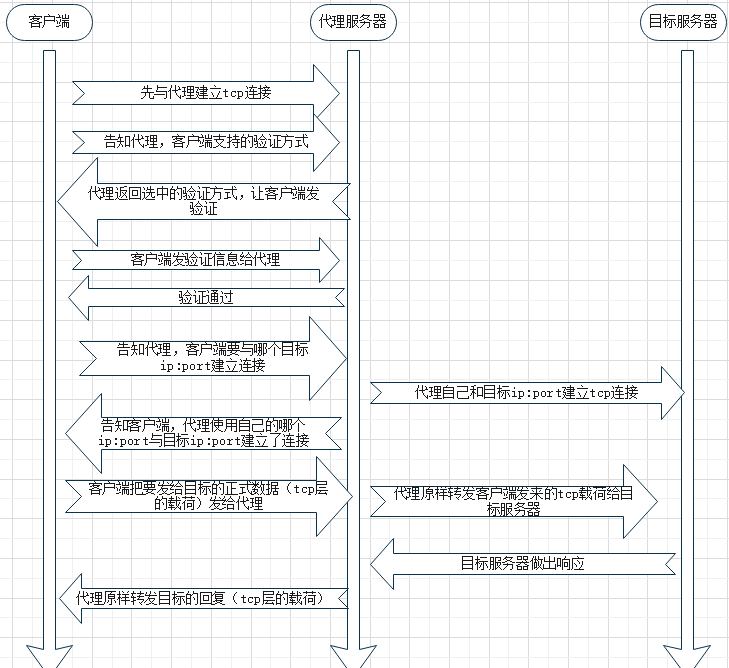
客户端首先也是发起http请求，**请求的方法**不再是GET/PUT/POST之类的，而是CONNECT，表示希望与目标服务器建立tcp层的隧道连接，让代理服务器原样转发tcp层的载荷数据，代理收到此http报文后，通过dns查询目标服务器的ip，再和此目标建立tcp连接，连接建立后，再向客户端返回http报文，内容为http/1.1 200 表示连接建立了。然后客户端把正式的要向目标服务器发送的数据（如ssl层的交互消息等）原样发给代理，代理原样发给目标，目标返回后，代理再原样发给客户端。因为客户端和目标是通过ssl层来传输的http，即使用https协议，所以代理这个中转站只是转发了tcp层的数据，无法查看它们具体发了什么。

如果代理服务器需要客户端的身份验证的话，也是在前面的Http报文中说明，把用户名及密码放在http报文的头部中，验证通过再进行下面的tcp报文转发。

**Socks5正向代理**：

socks代理协议有多个版，常用的有socks4、socks4a、socks5

socks4仅支持tcp转发，socks4a多了可以在代理那边做dns解析，socks5多了udp的转发支持。socks5支持tcp的转发及udp的转发，可以在代理那边做dns解析。这个tcp转发和Https代理里的那个转发一样，只是前面的协商协议不是http的，是socks5特有的，也是明文传输。



具体的交互报文就不写了，了解原理就行。

**常用的Linux下的正向代理服务器**：

|  |  |  |  |
| --- | --- | --- | --- |
| 代理服务器软件 | 代理类型 | 默认port | 特点 |
| Squid | http/https | 3128 | 带缓存的 |
| Privoxy | http/https | 8118 | 带过滤，可二次代理，不支持客户端身份验证 |
| Varnish | http/https | 6081 | 带缓存，性能比squid强 |
| Polipo | http/https | 8123 | 不过滤，可二次代理 |
| TinyProxy | http/https | 8888 | 可嵌入式，体积小 |
| SS5 | socks5 | 1080 | 版本旧，适用于Centos6 |
| Dante | socks5 | 1080 | 版本较新，可用于Centos7 |

有的客户端软件本身不支持走代理通道上网，可以借助其他的代理接管器去上网，windows下的代理接管器软件有SocksCap或Netch，具体的用法这里不讲解。

Netch开源地址： https://github.com/netchx/netch

前面讲的http/https/socks5正向代理，在与代理交互时使用的协议都是明文的，不太安全，而且由于是明文的，运营商的上网行为管理系统很容易就识别了，秒秒钟就可以把这个流量给阻断了。所以前面的那些代理服务器常用于公司内网。要想通过公网去使用以上的正向代理服务，得看代理服务器放在哪里，如果是在国内，一般可以正常访问，如果是在境外，很快就被过滤了。

所以有人开发了可以跨越万里长城的正向代理服务器软件及客户端软件。原理也很简单，就是在客户端和代理服务器之间的流量使用了加密的算法进行加密，让其他人（运营商）无法识别，当然用的人多了，某些特征就比较明显了，也容易被墙。于是有了升级的版本，把代理通道的流量和正常的https流量混杂在一起，让上网行为管理系统也无法识别。因为外表上它就是正常的Https流量。这类正向代理服务有 ShadowSocks、ShadowSocksR、v2ray等。

还有其他的类似软件，是由其他商家开发的，用的协议也是上面的几个，只是服务器和客户端软件是他们自己写的，要收费的（即我们通常认为的vpn翻墙软件）

**这里说明一下**：国内的用户只能通过正常合法的通道上网，比如在电信运营商那里开通宽带/光纤上网，或使用有上网许可的手机进行2/3/4/5G数据接入上网，若要使用vpn或代理技术上网的话，得使用 有备案及相应运营许可的服务商提供的VPN、代理服务器。个人未备案就搭建vpn/代理服务器是非法的。

**使用正向代理上网**

进入windows设置总界面，再点击“网络和Internet”，点击“代理”



或者按下win键+R键，在运行栏里输入inetcpl.cpl打开Internet属性对话框，点击“连接”选项卡，再点击下面的“局域网设置”



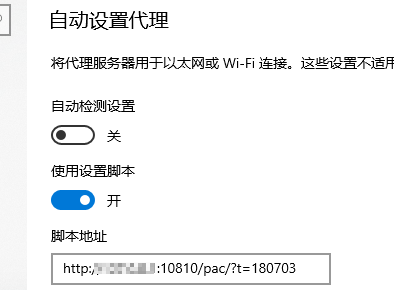
在“**自动设置代理**”这里，一般是选择“使用设置脚本”，此设置脚本是一个配置文件，用来告诉系统何时该使用代理上网，用哪个代理服务器，以及什么时候不用代理等信息。脚本文件可以是存放在系统本地的，如C盘或D盘里，也可以存放在云端的网站上。使用“自动设置代理”的好处就是我们可以在访问不同的资源时使用不同的代理服务器，以及使用的代理类型，或者在访问某些资源时不使用代理。

而“**手动设置代理**”那里，我们必须指明要使用的代理地址及端口号，且默认是使用http/https的代理类型，不能用socks代理。可以使用简单的ip过滤，比如访问某些网段时不使用此代理。网段写法如下：

localhost;127.\*;10.\*;172.16.\*;172.17.\*;172.18.\*;172.19.\*;172.20.\*;172.21.\*;172.22.\*;172.23.\*;172.24.\*;172.25.\*;172.26.\*;172.27.\*;172.28.\*;172.29.\*;172.30.\*;172.31.\*;192.168.\*

（上面其实是一行，不断开的），网段或ip之间用分号; 隔开，不留空格。上面表示访问本机ip及私有网段地址时，不使用代理上网。

**自动配置脚本**



这个脚本文件就是代理自动配置文件，为文本文件，后缀一般为.pac，内容就是js脚本代码，主要实现一个函数：function FindProxyForURL(url, host)

就是根据url或host找代理，返回的值有三种：DIRECT, PROXY, SOCKS

DIRECT 表示不使用代理，直接访问

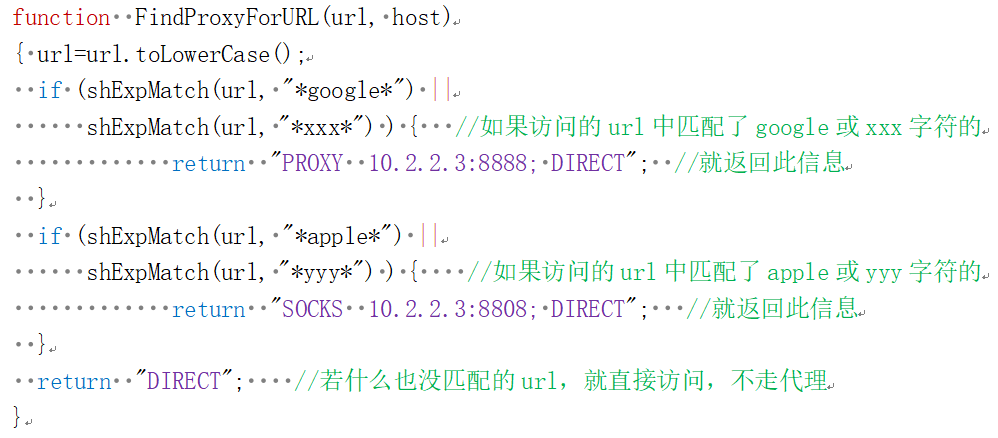
PROXY 表示使用http/https代理，后边紧跟代理的ip及端口号

SOCKS 表示使用socks代理，后边紧跟代理的ip及端口号

代理配置脚本文件可以在本地，填写时就写： file://D:\xxx\xx.pac

也可以位于某个网站上，如： http://xx.com:8888/xxx.pac 一般还带个参数，用于验证用户身份 http://xx.com:8888/xx.pac/?t=9734924279

**简单的pac代理自动配置文件内容示例:**



返回结果解析：

PROXY 10.2.2.3:8888; DIRECT表示走http/https代理，代理ip为10.2.2.3端口号8888，如果代理无响应或不通则走direct直连，就是本机直接访问，不走代理了。

SOCKS 10.2.2.3:8808; DIRECT表示走socks代理，代理ip为10.2.2.3端口号8808，如果代理无响应或不通则走direct直连，就是本机直接访问，不走代理了。

PAC文件示例：

function FindProxyForURL(url,host) {

url=url.toLowerCase();

host=host.toLowerCase();

if ( shExpMatch(host, "10.1.1.1") && isInNet(host,"10.99.0.0","255.255.0.0")

) {

return "DIRECT";

}

if ( shExpMatch(url, "\*.gg.\*") ||

shExpMatch(url, "\*.baidu.com")

) {

return "PROXY 10.99.1.1:10809";

}

if ( shExpMatch(url, "\*.ip0.\*") ||

shExpMatch(url, "\*.ip3.cn")

) {

return "PROXY 10.0.0.1:10806; SOCKS 192.168.0.1:8080";

}

return "DIRECT";

}

**章八、Squid（正向代理http）**

# yum install squid #安装squid服务，带缓存的

# systemctl enable squid

# cat /etc/squid/squid.conf | grep -v '^#' #查看默认配置

acl localnet src 10.0.0.0/8

acl localnet src 172.16.0.0/12

acl localnet src 192.168.0.0/16

acl localnet src fc00::/7 #这里定义了一个名为localnet的acl

acl localnet src fe80::/10 #5条rule匹配内网ip

acl SSL\_ports port 443 #定义一个名为SSL\_ports的acl匹配443端口

#以下定义一个名为Safe\_ports的acl，10条rule匹配常用端口号

acl Safe\_ports port 80 # http

acl Safe\_ports port 21 # ftp

acl Safe\_ports port 443 # https

acl Safe\_ports port 70 # gopher

acl Safe\_ports port 210 # wais

acl Safe\_ports port 1025-65535 # unregistered ports

acl Safe\_ports port 280 # http-mgmt

acl Safe\_ports port 488 # gss-http

acl Safe\_ports port 591 # filemaker

acl Safe\_ports port 777 # multiling http

acl CONNECT method CONNECT

http\_access deny !Safe\_ports #禁止客户端访问 非Safe\_ports的端口

http\_access deny CONNECT !SSL\_ports #禁止请求方法为CONNECT的流量访问非安全端口

http\_access allow localhost manager

http\_access deny manager

http\_access allow localnet

http\_access allow localhost

http\_access deny all

http\_port 3128 #代理服务监听的端口，默认3128

coredump\_dir /var/spool/squid

refresh\_pattern ^ftp: 1440 20% 10080

refresh\_pattern ^gopher: 1440 0% 1440

refresh\_pattern -i (/cgi-bin/|\?) 0 0% 0

refresh\_pattern . 0 20% 4320

**其他配置：**

access\_log /var/log/squid/access.log #访问日志文件

cache\_log /var/log/squid/cache.log

acl OverConnLimit maxconn 128 #限制同一IP客户端的最大连接数

http\_access deny OverConnLimit

acl allow\_dst\_ip dst 10.99.1.0/24 #设置允许客户端访问的IP地址

http\_access deny !allow\_dst\_ip

#允许本地管理

acl Manager proto cache\_object

acl Localhost src 127.0.0.1 10.99.1.248

http\_access allow Manager Localhost

maximum\_object\_size 4 MB #磁盘缓存最大文件，超过4MB的文件不保存到磁盘

minimum\_object\_size 0 KB #磁盘缓存最小文件

maximum\_object\_size\_in\_memory 4 MB #内存缓存最大文件

cache\_dir ufs /var/spool/squid 512 16 256 #设置缓存存放路径，缓存目录容量512 MB，一级缓存目录数量16，二级缓存目录数量256

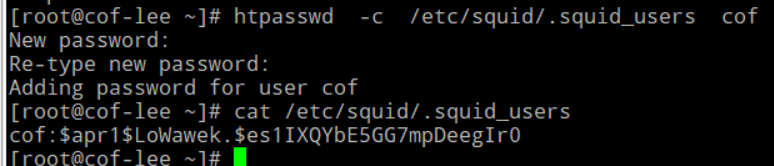
cache\_swap\_high 95 #缓存目录使用达95%时，开始清理旧的缓存

cache\_swap\_low 90 #缓存目录清理到90%时停止

**使用用户名密码验证**

# yum install httpd-tools

# htpasswd -c /etc/squid/.squid\_users cof #新增一个用户（名为cof）信息保存在/etc/squid/.squid\_users文件里



# htpasswd /etc/squid/.squid\_users userNamexx #新增其他用户

**#在/etc/squid/squid.conf里添加以下内容**

auth\_param basic program /usr/lib64/squid/basic\_ncsa\_auth /etc/squid/.squid\_users

#让squid服务知道在哪个文件里找用户名和密码

auth\_param basic chidren 5 #指定生成squid身份验证器过程的最大数量

auth\_param basic realm Proxy Authentication Required

auth\_param basic credentialsttl 2 hours

auth\_param basic casesensitive off

acl auth\_users proxy\_auth cof

http\_access allow auth\_users

**★通用代理配置示例（无需验证）**

http\_port 3128 #代理服务监听的端口，默认3128

maximum\_object\_size 4 MB #磁盘缓存最大文件

cache\_dir ufs /var/spool/squid 512 16 256 #设置缓存存放路径，缓存目录容量512 MB，一级缓存目录数量16，二级缓存目录数量256

access\_log /var/log/squid/access.log #访问日志文件

acl allow\_clients src 10.99.1.0/24 #定义允许访问本代理服务的客户端网段

acl deny\_dst\_ip dst 10.0.0.0/8 #不允许客户通过代理去访问的目的网段

acl deny\_dst\_ip dst 172.16.0.0/12 #不允许客户通过代理去访问的目的网段

acl deny\_dst\_ip dst 192.168.0.0/16 #不允许客户通过代理去访问的目的网段

http\_access deny deny\_dst\_ip

http\_access allow allow\_clients

http\_access deny all

**★需用户验证代理配置示例**

http\_port 3128 #代理服务监听的端口，默认3128

maximum\_object\_size 4 MB #磁盘缓存最大文件

cache\_dir ufs /var/spool/squid 512 16 256 #设置缓存存放路径，缓存目录容量512 MB，一级缓存目录数量16，二级缓存目录数量256

access\_log /var/log/squid/access.log #访问日志文件

acl allow\_clients src 10.99.1.0/24 #定义允许访问本代理服务的客户端网段

acl deny\_dst\_ip dst 10.0.0.0/8 #不允许客户通过代理去访问的目的网段

acl deny\_dst\_ip dst 172.16.0.0/12 #不允许客户通过代理去访问的目的网段

acl deny\_dst\_ip dst 192.168.0.0/16 #不允许客户通过代理去访问的目的网段

auth\_param basic program /usr/lib64/squid/basic\_ncsa\_auth /etc/squid/.squid\_users

#让squid服务知道在哪个文件里找用户名和密码

auth\_param basic chidren 5 #指定生成squid身份验证器过程的最大数量

auth\_param basic realm Proxy Authentication Required

auth\_param basic credentialsttl 2 hours

auth\_param basic casesensitive off

acl auth\_users proxy\_auth cof

http\_access deny deny\_dst\_ip #要确保这条在http\_access的最前面

#http\_access deny !allow\_clients

http\_access allow auth\_users #一旦客户认证通过了，就不再检查下面这条了

http\_access allow allow\_clients #要让这条生效，得放前面，用deny !

http\_access deny all

**章九、PriVoxy（正向代理http）**

privoxy不支持用户密码验证，可二次反向代理，无缓存

**#下载源码包**

下载地址：https://sourceforge.net/projects/ijbswa/files/Sources/

下载stable的源码包并上传到服务器上

**#安装依赖项**

# yum install autoconf automake libtool -y

# tar -xvf privoxy-3.0.33-stable-src.tar.gz  **#解压源码包**

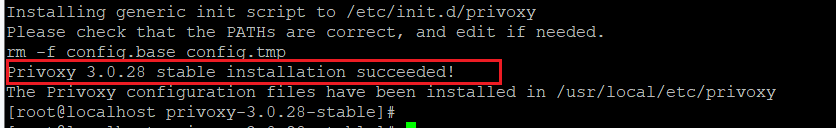
# cd privoxy-3.0.33-stable #进入解压目录，编译安装

# autoheader && autoconf

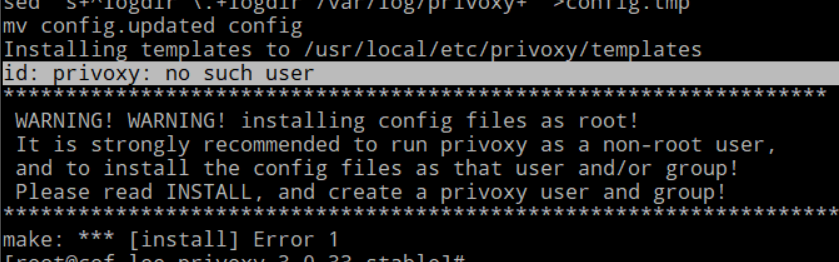
# ./configure

# make && make install

出现Privoxy x.x.x stable installation succeeded! 时表示安装成功



如果失败了，报错如下：说明需要创建privoxy用户



# useradd privoxy #添加privoxy用户

# make && make install #重试，成功

源码编译安装的 可执行文件为 /usr/local/sbin/privoxy ，默认配置路径为

/usr/local/etc/privoxy，配置文件为/usr/local/etc/privoxy/config

# /sbin/chkconfig privoxy on #设置为开机自启，源码安装时默认不受systemd管理

# /etc/init.d/privoxy {start|stop|restart} #启动，停止，重启服务

执行以下命令手动启动prixovy服务，建议只用上面的方法

# /usr/local/sbin/privoxy /usr/local/etc/privoxy/config

★**epel-release的yum仓库有privoxy软件包**

# yum install epel-release #或者直接创建epel.repo文件：

[epel]

name=Extra Packages for Enterprise Linux 7 - $basearch

#baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch

metalink=https://mirrors.fedoraproject.org/metalink?repo=epel-7&arch=$basearch

failovermethod=priority

enabled=1

gpgcheck=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7

[epel-debuginfo]

name=Extra Packages for Enterprise Linux 7 - $basearch - Debug

#baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch/debug

metalink=https://mirrors.fedoraproject.org/metalink?repo=epel-debug-7&arch=$basearch

failovermethod=priority

enabled=0

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7

gpgcheck=1

[epel-source]

name=Extra Packages for Enterprise Linux 7 - $basearch - Source

#baseurl=http://download.fedoraproject.org/pub/epel/7/SRPMS

metalink=https://mirrors.fedoraproject.org/metalink?repo=epel-source-7&arch=$basearch

failovermethod=priority

enabled=0

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7

gpgcheck=1

# yum install privoxy

# systemctl enable privoxy

yum安装的privoxy可执行文件为 /usr/sbin/privoxy ，默认配置路径/etc/privoxy

**查看默认配置文件内容**

# cat /usr/local/etc/privoxy/config | grep -v '^#' #编译安装的配置文件

# cat /etc/privoxy/config | grep -v '^#' #yum安装的配置文件

confdir /usr/local/etc/privoxy

logdir /var/log/privoxy

actionsfile match-all.action # Actions that are applied to all sites and maybe overruled later on.

actionsfile default.action # Main actions file

actionsfile user.action # User customizations

filterfile default.filter

filterfile user.filter # User customizations

logfile logfile

listen-address 127.0.0.1:8118

toggle 1 #广告过滤，0为关闭，1为开启

enable-remote-toggle 0

enable-remote-http-toggle 0

enable-edit-actions 0 #是否允许在web管理页面上编辑配置

enforce-blocks 0

buffer-limit 4096

enable-proxy-authentication-forwarding 0 #是否转发客户的代理用户验证信息给上级代理

forwarded-connect-retries 0

accept-intercepted-requests 0

allow-cgi-request-crunching 0

split-large-forms 0

keep-alive-timeout 5

tolerate-pipelining 1

socket-timeout 300

#

★当客户端配置了代理后，浏览器访问 http://p.p/ 或http://config.privoxy.org/show-status可进入privoxy的管理页面（前提是配置了confdir /etc/privoxy）

**最简单通用代理配置示例：**

listen-address 0.0.0.0:8081 #监听本地任意ip的8081端口

forward / . # /斜杠表示客户要访问的目标为任意地址，后面的点表示本代理服务器自己去请求。privoxy可以把客户端流量转给另一个代理服务器（上级代理）

permit-access 10.99.1.0/24 #允许访问本代理服务的客户端ip

forwarded-connect-retries 1 #转发重试次数

max-client-connections 256 #最大客户端连接数

**二级代理配置示例：**

listen-address 0.0.0.0:8081 #监听本地任意ip的8081端口

forward 10.128.\*.\*/ 10.1.1.1:333 #访问10.128.\*.\*的流量转给10.1.1.1:333去处理

forward .baidu.com/ 10.99.1.248:3128 #访问\*.baidu.com的流量转给10.99.1.248:3128

forward-socks5 .google.com/ 10.99.1.1:10808 . #访问\*.google.com的流量转给10.99.1.1:10808（socks5代理服务器） forward表示后面的代理服务类型为http(s)

#forward-socks5表示后面的代理服务类型为socks5

forward / . # /斜杠表示客户要访问的目标为任意地址，后面的点表示本代理服务器自己去请求。privoxy可以把客户端流量转给另一个代理服务器（上级代理）

permit-access 10.99.1.0/24 #允许访问本代理服务的客户端ip

forwarded-connect-retries 1 #转发重试次数

max-client-connections 256 #最大客户端连接数

enable-proxy-authentication-forwarding 1 #privoxy本身不需要用户密码验证，但有的上级代理需要，可以配置允许privoxy转发客户的用户密码验证信息给上级代理

**章十、Varnish（正向代理http）**

在

**章十一、Dante（正向代理socks5）**

在 https://www.inet.no/dante/download.html 官网下载较新版本源码包并上传到目标服务器上，比如 dante-1.4.3.tar.gz

# tar -xvf dante-1.4.3.tar.gz #解压源码包

# cd dante-1.4.3 #进入解压目录

# ./configure

# make & make install #编译安装

程序文件：/usr/local/sbin/sockd 默认配置文件：无，可以参考解压目录下的example/子目录下的socks.conf文件

启动程序：

# /usr/local/sbin/sockd -f */配置文件* -D #以指定配置启动，后台运行

#不指定配置文件时 默认去加载 /etc/sockd.conf

**★无需用户密码验证的配置示例**

internal: ens33 port=1088 #对内提供代理服务时监听的网卡及端口

external: ens34 #代理服务自己去访问目标服务时使用的网卡，可和internal相同

socksmethod: none #无用户密码验证

user.notprivileged: nobody

errorlog: /var/log/sockd.errlog

logoutput: /var/log/sockd.log

client pass {

from: 10.0.0.0/8 to: 0.0.0.0/0 #允许from后的网段客户端连接本代理服务器

log: connect disconnect error

}

socks pass {

from: 10.0.0.0/8 to: 0.0.0.0/0

command: bind connect udpassociate #允许客户端访问dns,tcp,udp

log: error connect disconnect iooperation

}

socks pass {

from: 0.0.0.0/0 to: 10.0.0.0/8

command: bindreply udpreply #允许真实服务端回复的dns,udp

log: error connect disconnect iooperation

}

**★用户密码验证的配置示例**

internal: ens33 port=1088 #对内提供代理服务时监听的网卡及端口

external: ens34 #代理服务自己去访问目标服务时使用的网卡，可和internal相同

socksmethod: username #使用系统用户

user.privileged: root

user.unprivileged: nobody

errorlog: /var/log/sockd.errlog

logoutput: /var/log/sockd.log

client pass {

from: 10.0.0.0/8 to: 0.0.0.0/0 #允许from后的网段客户端连接本代理服务器

log: connect disconnect error

}

socks pass {

from: 10.0.0.0/8 to: 0.0.0.0/0

command: bind connect udpassociate #允许客户端访问dns,tcp,udp

log: error connect disconnect iooperation

}

socks pass {

from: 0.0.0.0/0 to: 10.0.0.0/8

command: bindreply udpreply #允许真实服务端回复的dns,udp

log: error connect disconnect iooperation

}

**章十二、v2ray**

v2ray支持http(s), socks5正向代理，tcp/udp反向代理，还支持多级代理，将代理的流量转到其他上级代理服务去处理，2个v2ray服务器之间可以使用ssl加密流量转发用户流量

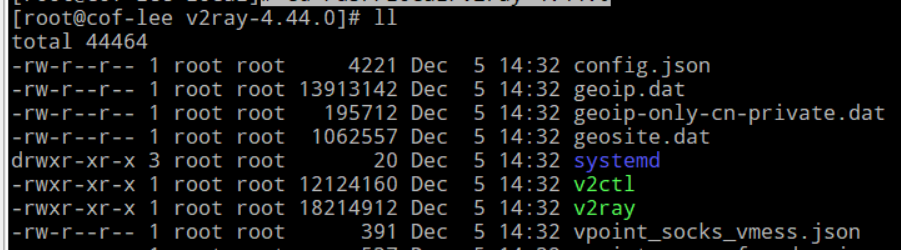
v2ray开源地址： https://github.com/v2fly/v2ray-core 始于2015-09

下载v2ray-linux-64-4.44.0.zip压缩包到代理服务器上，

# mkdir /usr/local/v2ray-4.44.0 #创建v2ray工作目录

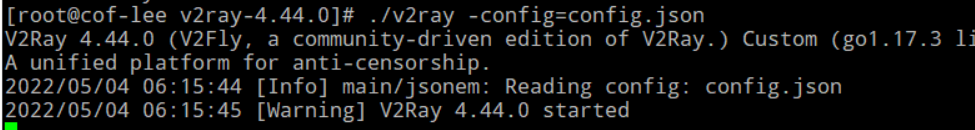
# unzip -d /usr/local/v2ray-4.44.0 v2ray-linux-64-4.44.0.zip #解压到工作目录

# cd /usr/local/v2ray-4.44.0 #进入解压目录



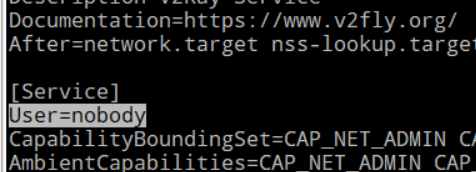
解压目录下的v2ray文件是主程序可执行文件，config.json为配置文件，启动v2ray服务方法为执行文件 加参数 -config=配置文件

# ./v2ray -config=config.json



或者在解压目录下的systemd/system/v2ray.service为服务文件，可作成systemd服务

**注意事项：**默认的服务使用nobody用户，如果配置了日志，要访问某日志文件的话，得提前创建日志文件并授予nobody读写权限



官方文档： https://www.v2fly.org/config/overview.html

**★v2ray配置讲解，是一个json配置**

{

"log": {}, //日志配置

"inbounds": [], //流量入口，前端监听的代理配置，可为http(s), tcp, udp, socks

"outbounds": [], //流量出站，可以是本服务器自己去访问用户的目标，也可转给上级代理，或者block

"routing": {}, //流量代理/转发规则

"dns": {} //DNS配置，v2ray内置的dns服务仅支持A,AAAA记录

}

**★日志配置**

"log": {

"loglevel": "info", //debug,info,warning,error, none不记录

"access": "/var/log/v2ray-access.log",

"error": "/var/log/v2ray-error.log" //或 D:\\xxx\\xx.log

},

**★inbounds入站配置（正向代理）**

"inbounds": [

**{** //这一段为**http正向代理**配置

"tag": "http-proxy", //入口流量打上tag，方便后面做转发策略，名称随便写

"port": 8080,

"protocol": "http",

"sniffing": {

"enabled": true, //开启sniffing识别流量类型，用于转发策略匹配

"destOverride": [

"http",

"tls"

]

},

"settings": {

"timeout": 0,

"auth": "password", //需要用户密码验证，noauth则不需要验证

"accounts": [ //验证为password时需要配置，可有多个账号

{

"user": "user1",

"pass": "passwdxx"

},

{

"user": "user2",

"pass": "passwdxxx"

}

],

"allowTransparent": true, //false只转发代理请求的http流量，true则转发所有http请求

"userLevel": 0

}

**}**, //end of http正向代理配置

**{** //这一段为tcp/udp反向代理

"tag": "tcp99-proxy",

"port": 99,

"protocol": "dokodemo-door",

"settings": {

"address": "10.99.1.61",

"port": 22,

"network": "tcp" //可为"tcp","udp","tcp,udp"

}

**}**, //end of tcp/udp反向代理

**{** //这一段为**socks正向代理**

"tag": "socks-proxy",

"port": 1088,

"listen": "0.0.0.0",

"protocol": "socks", //支持socks4, socks4a, socks5

"settings": {

"auth": "noauth",

"udp": true,

}

**}** //end of socks代理

],

#

**★outbounds出站配置（自己处理或转给上级代理）**

"outbounds": [

{

"tag": "direct",

"protocol": "freedom" //本服务器**自己去处理**用户请求的流量

},

{

"tag": "block",

"protocol": "blackhole", //将用户流量丢弃

"settings": {

"response": {

"type": "http" //block时会发回一个http 403的包，"none"则不回复

}

}

},

{ //转给上级http正向代理服务器

"tag": "up-to-http-proxy",

"protocol": "http",

"settings": {

"servers": [

{

"address": "10.99.1.22",

"port": 8080,

"users": [ //如果上级代理不需用户验证则写 "auth": "noauth"

{

"username": "usernamex",

"password": "passwdxx"

}

]

}

]

},

"streamSettings": {

"security": "none", //若转给上级时使用https代理，则写"tls"

"tlsSettings": {

"allowInsecure": true

}

}

}, //end of 转给上级http代理服务

{ //转给上级socks正向代理服务器

"tag": "up-to-socks-proxy",

"protocol": "socks",

"settings": {

"servers": [

{

"address": "10.99.1.212",

"port": 8090,

"users": [ //如果上级代理不需用户验证则写 "auth": "noauth"

{

"username": "usernamex",

"password": "passwdxx",

"level": 0

}

]

}

]

}

}

], //流量出站

#

**★routing转发策略**

"routing": {

"domainStrategy": "IPIfNonMatch", //先域名匹配，未匹配时再解析为ip进行匹配

"rules": [

{ // rule-1 以下各规则关系为 **AND**，都匹配上了才打outboundTag

"type": "field", //固定值

"inboundTag": [], //匹配入口的流量tag

"source": [], //匹配源ip

"network": "tcp", //匹配tcp流量

"port": null, // "53,54,100-200",匹配目的端口

"ip": [], //匹配流量的目的ip

"domain": [], //匹配流量的目的域名

"protocol": [], //匹配流量类型，"http","tls","bittorrent"，需要在入口开启sniffing

"outboundTag": "up-to-http-proxy" //上面都匹配上就走这个出口

},

{ // rule-2 第一条rule-1没匹配上的，接着匹配rule-2

"type": "field",

"ip": [ //同一规则下的每行关系为 **OR**

"10.99.0.0/16", //匹配目的网段

"geoip:cn", //匹配geoip.dat文件里面的标记为cn的ip

"192.168.1.1" //匹配单个目的ip

],

"outboundTag": "up-to-socks-proxy" //上面都匹配上就走这个出口

},

{ // rule-3

"type": "field",

"domain": [ //以下每行关系为 **OR**

"xxxx.com", //匹配单个域名

"geosite:cn", //匹配geosite.dat里面的标记为cn的域名

"domain:goo.com" //匹配 \*.goo.com

],

"outboundTag": "direct" //上面都匹配上就走这个出口

} //如果rule-3也没有匹配上，则用户流量默认走第一个outbound出口

] //end of rules

}, //流量转发策略

#

**★dns解析策略**

"dns": {

"hosts": {

"xxxx.com": "10.99.1.24",

"domain:msn.com": "10.2.3.4", // \*.msn.com

"wwwy.com": "2.2.2.3"

},

"servers": [

{

"address": "1.1.1.1",

"port": 53,

"domains": [ //当需要解析以下域名时，使用上面的dns

"yyyy.com",

"domain:xyxyx.com" //\*.xyxyx.com

]

},

"8.8.8.8", //从上到下按顺序来，此dns解析不了时，向下走

"localhost"

]

} //DNS配置，v2ray内置的dns服务仅支持A, AAAA记录

# v2ray内置的dns去向dns servers查询时的流量，也是要走routing策略匹配的，未匹配到时，走第一个出口

**★v2ray特有转发协议**

在服务器向上级转发用户流量时，常规的协议有http,socks反向代理，这些是明文的，不太安全，v2ray目前开发有vmess, torjan协议，它们可以封装在tls流量下

**①下游vmess出站**

VMess 依赖于系统时间，请确保系统 UTC 时间误差在 90 秒之内，时区无关

"outbounds": [

{

"tag": "up-to-vmess",

"protocol": "vmess",

"settings": {

"vnext": [

{

"address": "10.1.1.2",

"port": 3333, //上游的https端口

"users": [

{

"id": "uuid",

"alterId": 62,

"email": "xx@xx.com",

"security": "auto"

}

]

}

]

},

"streamSettings": {

"network": "ws",

"security": "tls",

"tlsSettings": {

"allowInsecure": true,

"serverName": "xxxx.com"

},

"wsSettings": {

"connectionReuse": true,

"path": "/vtestx",

"headers": {

"Host": "xxxx.com"

}

}

},

"mux": {

"enabled": true,

"concurrency": 8

}

}

]

**②上游vmess入站**

在

"inbounds": [

{

"tag": "in-vmess",

"port": 3859,

"protocol": "vmess",

"settings": {

"clients": [

{ //其中一个用户验证信息

"id": "uuid",

"alterId": 62,

"email": "xx@xx.com",

"security": "auto"

} //可有多个client

]

},

"streamSettings": {

"network": "ws",

"wsSettings": {

"path": "/vtestx" //http流量，url路径/vtestx，前面需要用一个https:3333反向转到此http:3859来

}

}

}

]

**③下游trojan出站**

在

"outbounds": [

{

"tag": "up-to-trojan-proxy",

"protocol": "trojan",

"settings": {

"servers": [

{

"address": "10.1.1.2",

"password": "b4c0b6f8-f545-4f04-8482-712c406ad71b",

"port": 10028,

"level": 1,

"flow": ""

}

]

},

"streamSettings": {

"network": "tcp",

"security": "tls",

"tlsSettings": {

"allowInsecure": true,

"serverName": "xxxx.com"

}

},

"mux": {

"enabled": false, // true

"concurrency": -1 // 8

}

}

]

**④上游trojan入站**

在

"inbounds": [

{

"tag": "in-trojan",

"protocol": "trojan",

"port": 3859,

"settings": {

"clients":[

{

"password": "password", //密码为任意字符串，常用uuid

"email": "love@v2fly.org", //可选，用于标识用户

"level": 0 //用于本地策略的，如连接超时设置

} //可有多个用户账号

]

},

"streamSettings": {

"network": "tcp",

"security": "tls", // 启用 TLS

"tlsSettings": {

"certificates": [

{

"certificateFile": "/etc/v2ray/v2ray.crt", // 证书文件

"keyFile": "/etc/v2ray/v2ray.key" // 密钥文件

}

]

}

}

}

]

**★v2ray上游代理配置示例**

{

"log": {

"loglevel": "error",

"access": "/usr/local/v2ray-4.44.0/v2ray-access.log",

"error": "/usr/local/v2ray-4.44.0/v2ray-error.log"

},

"inbounds": [

{

"tag": "in-trojan",

"port": 7505,

"protocol": "trojan",

"settings": {

"clients":[

{

"password": "b4c0b6f8-f545-4f04-8482-712c406ad71b",

"email": "userxx@v2fly.org",

"level": 0

}

]

},

"streamSettings": {

"network": "tcp",

"security": "tls",

"tlsSettings": {

"certificates": [{

"certificateFile": "/usr/local/v2ray-4.44.0/xxxx.com.crt",

"keyFile": "/usr/local/v2ray-4.44.0/xxxx.com.key"

}]

}

}

}

],

"outbounds": [

{

"tag": "direct",

"protocol": "freedom"

}

],

"dns": {

"servers": [

"8.8.8.8",

"localhost"

]

},

"routing": {

"domainStrategy": "IPIfNonMatch",

"rules": [

{

"type": "field",

"inboundTag": "in-trojan",

"outboundTag": "direct"

}

]

}

}

#

**★v2ray下游代理配置示例**

{

"log": {

"loglevel": "info",

"access": "D:\\v2ray\_4.44\_windows\_64\\v2ray-access.log",

"error": "D:\\v2ray\_4.44\_windows\_64\\v2ray-error.log"

},

"inbounds": [

{

"tag": "tcp-922-proxy",

"port": 922,

"protocol": "dokodemo-door",

"settings": {

"address": "127.0.0.1",

"port": 22,

"network": "tcp"

}

}

],

"outbounds": [

{

"tag": "up-to-trojan-proxy",

"protocol": "trojan",

"settings": {

"servers": [

{

"address": "10.12.16.168",

"password": "b4c0b6f8-f545-4f04-8482-712c406ad71b",

"port": 7505,

"level": 1,

"flow": ""

}

]

},

"streamSettings": {

"network": "tcp",

"security": "tls",

"tlsSettings": {

"allowInsecure": true,

"serverName": "xxxx.com"

}

},

"mux": {

"enabled": true,

"concurrency": 8

}

}

],

"dns": {

"servers": [

"8.8.8.8",

"localhost"

]

},

"routing": {

"domainStrategy": "IPIfNonMatch",

"rules": [

{

"type": "field",

"inboundTag": "tcp-922-proxy",

"outboundTag": "up-to-trojan-proxy"

}

]

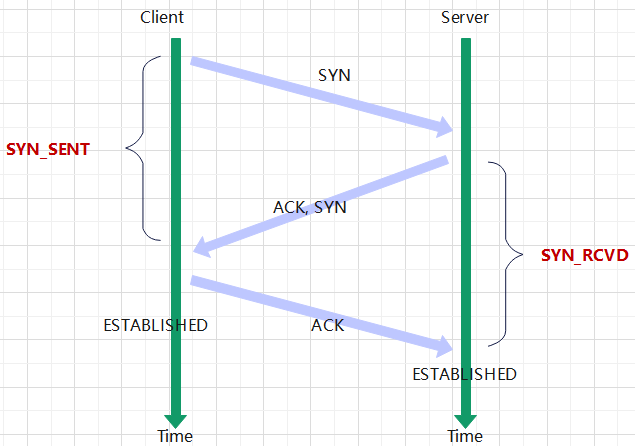
}

}

#

**附录：TCP连接超时问题**

**①TCP建立连接（三次握手）**

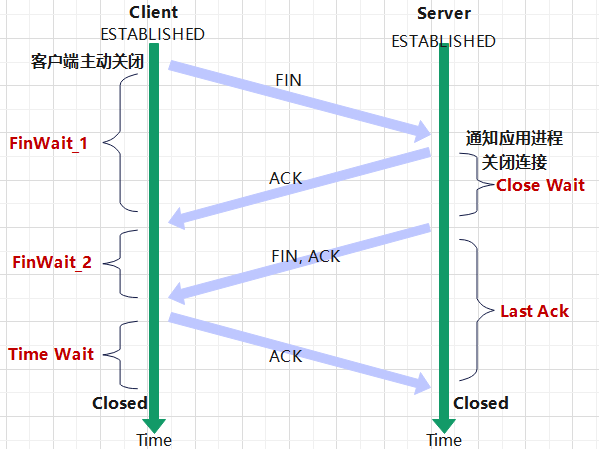


**仅2个超时：**

SYN\_SENT（TCPinitialRTT）等对方回ACK, SYN

SYN\_RCVD 等发起方回复最后一个ACK

**②TCP双向关闭（四次挥手）**



**5个超时：**

|  |  |
| --- | --- |
| FinWait\_1 | （KeepAliveInterval）windows默认1秒；linux开始1秒，重试依次翻倍 |
| CloseWait | （KeepAliveTime）windows及linux默认2小时 |
| FinWait\_2 | （TCPFinWait2Delay）windows默认240秒， |
| LastAck | （KeepAliveInterval）同FinWait\_1 |
| TimeWait | （默认2倍MSL时间）**MSL时间**（Maximum Segment Lifetime），报文最大生存时间，根据操作系统而异：windows 120秒，Linux 60秒，Unix 30秒 |

Windows系统 TCP超时

|  |  |
| --- | --- |
| KeepAliveInterval  （Finwait\_1, LastAck） | HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters  缺省1000毫秒 |
| KeepAliveTime  （CloseWait） | HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters  缺省7200 000毫秒，2小时（建议设置300000五分钟） |
| TCPFinWait2Delay  （FinWait\_2） | HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters  缺省240秒（可设置范围30~300） |
| TcpTimedWaitDelay  （TimeWait） | HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters  缺省240秒（可设置范围30~300） |
| TcpMaxConnectRetries | 建议3次 |
| TcpMaxDataRetransmissions  TcpMaxDataRetries | 建议3次 |
| TCPInitialRTT | SYN连接超时，默认3秒 |

Linux系统TCP超时（/etc/sysctl.conf）

|  |  |  |
| --- | --- | --- |
| 配置项=默认值 | 建议 | 说明 |
| net.ipv4.tcp\_syn\_retries = 6 | 2 | 建立tcp连接失败后重试连接的次数 |
| net.ipv4.tcp\_keepalive\_time = 7200 | 60 | CloseWait，默认2小时（每2小时触发探测 |
| net.ipv4.tcp\_keepalive\_probes = 9 | 3 | 探测次数 |
| net.ipv4.tcp\_keepalive\_intvl = 75 | 3 | 探测超时 |
| net.ipv4.tcp\_fin\_timeout = 120 | 30 | TimeWait |
| net.ipv4.tcp\_fin\_wait\_2\_timeout = 30 | 30 | FinWait\_2，TCP标准未定义，默认？ |
| net.ipv4.tcp\_orphan\_retries = 7 | 3 | 试图关闭tcp重试的次数 |
| net.ipv4.ip\_local\_port\_range = |  | 默认 10240 65000 两个数字，空格隔开 |

Linux系统Syn建立连接时，重发间隔 依次翻倍递增

|  |  |  |
| --- | --- | --- |
| 第一次发SYN报文 | 超时达1秒则判定失败 | 累计1秒 |
| 重发第1次 | 超时达2秒则判定失败 | 累计3秒 |
| 重发第2次 | 超时达4秒则判定失败 | 累计7秒 |
| 重发第3次 | 超时达8秒则判定失败 | 累计15秒 |
| 重发第4次 | 超时达16秒则判定失败 | 累计31秒 |
| 重发第5次 | 超时达32秒则判定失败 | 累计63秒 |
| 重发第6次 | 超时达64秒则判定失败 | 累计127秒 |