第1章 Haproxy介绍

1.官方地址

```
https://www.haproxy.com/
```

2.Haproxy和Nginx区别

```
haproxy只做反向代理负载均衡功能
Nginx除了能做代理转发，也可以做web服务器
```

3.Haproxy应用场景

```
4层负载
7层负载
```

第2章 Haproxy实战

1.Haproxy安装部署

注意：yum安装的haproxy有两个版本，名字不一样，默认haproxy为1.5版本，而最新的1.8版本名称为haproxy18

```
#查找yum仓库里的haproxy版本
[root@lb-5 ~]# yum search haproxy|grep ^haproxy
haproxy18.x86_64 : HAProxy reverse proxy for high availability environments
haproxy.x86_64 : TCP/HTTP proxy and load balancer for high availability

#安装haproxy
[root@lb-5 ~]# yum install haproxy -y

#查看haproxy版本
[root@lb-5 ~]# haproxy -v
HA-Proxy version 1.5.18 2016/05/10
Copyright 2000-2016 Willy Tarreau <willy@haproxy.org>

#查看haproxy配置文件有哪些
[root@lb-5 ~]# rpm -qc haproxy
/etc/haproxy/haproxy.cfg
/etc/logrotate.d/haproxy
/etc/sysconfig/haproxy
```

安装1.8版本命令

```
rpm -ivh http://www.nosuchhost.net/~cheese/fedora/packages/epel-7/x86_64/cheese-release-7-1.noarch.rpm
yum install haproxy
haproxy -v
```

2.Haproxy配置文件解释

官方文档：需翻墙

```
http://www.haproxy.org/download/1.4/doc/configuration.txt
```

配置说明:

```
HAProxy配置中分成五部分内容，当然这些组件不是必选的，可以根据需要选择部分作为配置。
global：参数是进程级的，通常和操作系统相关。这些参数一般只设置一次，如果配置无误，就不需要再次配置进行修改
defaults：配置默认参数的，这些参数可以被利用配置到frontend，backend，listen组件
frontend：接收请求的前端虚拟节点，Frontend可以根据规则直接指定具体使用后端的backend(可动态选择)。
backend：后端服务集群的配置，是真实的服务器，一个Backend对应一个或者多个实体服务器。
listen：Frontend和Backend的组合体。
```

全剧配置说明:

```
global
    log         127.0.0.1 local2           #log
    chroot      /var/lib/haproxy           #锁定运行目录
    pidfile     /var/run/haproxy.pid     #pid文件路径
    maxconn     4000                                #每个进程的最大并发数
    user        haproxy                          #haproxy运行用户
    group       haproxy                          #haproxy运行用户组
    daemon                                                  #以后台进程模式运行
    stats socket /var/lib/haproxy/stats     #socket文件路径
```

代理配置-defaults说明:

```
defaults
    mode                    http                                #默认运行的模式，有http和tcp两种
    log                     global                  #日志路径，这里表示以global定义的为准
    option                  httplog                 #丰富日志格式，包含更多信息
    option                  dontlognull         #日志不记录空连接
    option http-server-close                                    #在服务端启用HTTP连接关闭
    option forwardfor       except 127.0.0.0/8  #启用X-Forwarded-For标头，记录用户真实IP
    option                  redispatch          #在连接失败的情况下启用或禁用会话重新分发
    retries                 3                                   #设置连接失败后在服务器上执行的重试次数
    timeout http-request    10s                             #设置等待完整的HTTP请求所允许的最长时间
    timeout queue           1m                              #设置在队列中等待空闲连接插槽的最长时间
    timeout connect         10s                             #设置等待连接服务器成功的最长时间
    timeout client          1m                              #在客户端上设置最长不活动时间
```

```
    timeout server          1m                      #在服务器端设置最大不活动时间
    timeout http-keep-alive 10s                     #设置等待新HTTP请求出现的最长时间
    timeout check           10s                     #设置其他检查超时，但仅在已经建立连接之后
    maxconn                 3000                    #将每个进程的最大并发连接数
```

代理配置-frontend说明:

```
frontend http_80_in
    bind 0.0.0.0:80   #监听端口
    mode http                 #http的7层模式
    log global              #应用全局的日志配置
    option httplog    #启用http的log
    option httpclose  #每次请求完毕后主动关闭http通道，HA-Proxy不支持keep-alive模式
    option forwardfor #获得客户端IP记录到forwardfor字段
```

代理配置-backend说明:

```
backend mms_server
    mode http                           #工作在http的7层模式
    balance roundrobin      #负载均衡的方式，roundrobin为默认平均轮询
    option  #配置选项可以配置以下的参数用于后端健康检查
        httpchk
        mysql-check
        pgsql-check
        ssl-hello-chk
    server  #后端服务器的配置可以配置以下参数
        check           #健康检查
        inter N     #健康检查间隔时间
        fall N    #后端服务器失败检查次数
        rise N      #后端服务器恢复检查次数
        weight      #权重，0表示不参负载均衡
```

代理配置-listens说明:

```
listen web
    bind 10.0.0.5:80
    mode http
    balance roundrobin
    server web-7 10.0.0.7:80 check inter 3000 fall 3 rise 5
    server web-8 10.0.0.8:80 check inter 3000 fall 3 rise 5
```

3.Haproxy代理实验

3.1 实验环境说明

```
lb-5    haproxy代理服务器
web-7   后端web服务器
web-8   后端web服务器
```

3.2 准备后端nginx测试环境

```
echo $(hostname) > /usr/share/nginx/html/index.html
systemctl restart nginx
curl 127.0.0.1
```

3.3 创建haproxy配置文件

```
cat > /etc/haproxy/haproxy.cfg << 'EOF'
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 web_in_80
    bind 10.0.0.5:80
    mode http
    use_backend web_server_80

backend web_server_80
    mode  http
    option forwardfor
    server web7 10.0.0.7:80 check inter 3000 fall 3 rise 5
    server web8 10.0.0.8:80 check inter 3000 fall 3 rise 5

#listen可以把frontend和backend整合在一起，写起来更简洁
listen web_server_8080
    bind 10.0.0.5:8080
    mode http
    balance roundrobin
    server web7 10.0.0.7:8080 check inter 3000 fall 3 rise 5
    server web8 10.0.0.8:8080 check inter 3000 fall 3 rise 5
EOF
```

3.4 启动Haproxy

```
systemctl start haproxy
```

3.5 访问并测试

```
curl 10.0.0.5
```

第3章 Haproxy调度算法

1.调度算法介绍

```
Haproxy拥有很多中调度算法
```

2.基于权重的负载

```

```

3.基于轮询的负载

```

```

4.使用socat工具动态下线/上线后端节点

4.1 socat 工具介绍

```
socat是一个功能强大的网络调试工具，支持众多协议，比如socket,IP,TCP等。
我们可以使用socat对haproxy的sokcat文件进行操作，使用haproxy支持的命令动态的上/下线节点，优势是不需要修改配置文件。
```

4.2 socat命令使用

```
#!.安装工具
yum -y install socat

#2.查看命令帮助
socat -h

#3.查看haproxy的socket命令帮助
echo "help" | socat stdio /var/lib/haproxy/haproxy.sock

#4.下线节点
echo disable server web/web-7| socat stdio /var/lib/haproxy/haproxy.sock

#5.上线节点
echo enable server web/web-8| socat stdio /var/lib/haproxy/haproxy.sock
```

4.3 jenkins代码上线滚动更新脚本

```
#!/bin/bash

PATH_CODE=/var/lib/jenkins/workspace/my-freestyle-job
PATH_WEB=/usr/share/nginx
if [ "$NODE" == "ALL" ];then
   IP="10.0.0.7 10.0.0.8"
elif [ "$NODE" == "10.0.0.7" ];then
   IP="10.0.0.7"
else
   IP="10.0.0.8"
fi

#打包代码
code_tar(){
        cd ${PATH_CODE}
        tar zcf /opt/web-${git_version}.tar.gz ./*
}

#负载均衡器下线节点
node_down(){
        ssh root@10.0.0.5 "echo disable server web/${NODE}| socat stdio /var/lib/haproxy/haproxy.sock"
        echo "${NODE} 从 10.0.0.5 移除成功"
}

#拷贝打包好的代码发送到web服务器代码目录
code_scp(){
        ssh ${NODE} "mkdir ${PATH_WEB}/web-${git_version} -p"
        scp /opt/web-${git_version}.tar.gz ${NODE}:${PATH_WEB}/web-${git_version}
}

#web服务器解压代码
code_xf(){
        ssh ${NODE} "cd ${PATH_WEB}/web-${git_version} && tar xf web-${git_version}.tar.gz && rm -rf web-${git_version}.tar.gz"
}

#创建代码软链接
code_ln(){
        ssh ${NODE} "cd ${PATH_WEB} && rm -rf html && ln -s web-${git_version} html"
}

#测试访问是否正常
web_test(){
        curl -s -I -m 10 -o /dev/null -w %{http_code} http://${NODE}/index.html
}

#负载均衡上线节点
node_up(){
        ssh root@10.0.0.5 "echo enable server web/${NODE}| socat stdio /var/lib/haproxy/haproxy.sock"
        echo "${NODE} 从 10.0.0.5 添加成功"
```

```
}


#选择发布还是回滚
if [ "${deploy_env}" == "deploy" ];then
    code_tar
    for NODE in ${IP};do
        node_down
        code_scp
        code_xf
        code_ln
        web_test
        node_up
    done
elif [ "${deploy_env}" == "rollback" ];then
    for NODE in ${IP};do
        node_down
        code_ln
        web_test
        node_up
    done
fi
```

第4章 ACL匹配

1.ACL配置说明

文字说明:

```
ACL(Access Control List)访问控制列表
是一种可以根据匹配的条件进行不同的跳转实现方法，比如可以根据浏览器类型，IP地址，URL，文件后缀名等条件进行匹配和转发。

在haproxy中，需要先定义ACL，然后再引用对应的ACL
```

配置语法:

```
acl  规则名称  匹配条件  匹配模式  执行操作  操作的对象
```

匹配模式说明:

```
-i 不区分大小写
-m 使用指定的pattern
-n 不做DNS解析
```

2.ACL匹配语法

2.1 hdr 请求报文首部匹配条件

匹配说明

```
hdr_dom  匹配请求host名称 如 www.oldboyedu.com
hdr_beg  匹配请求host开头 如 www. bbs. blog
hdr_end  匹配请求host开头结尾 如 .com .org
hdr_sub
```

举例:

```

```

2.2 path 请求URL路径匹配条件

匹配说明:

```
path_beg  匹配URL开头
path_end  匹配URL结尾
path_reg  匹配URL里的字符串
path_dir    匹配路径
```

举例:

```

```

2.3 基于IP和端口匹配

匹配说明:

```
dst             目标IP
dst_port  目标PORT
src       源IP
src_port  源PORT
```

举例:

```

```

3.匹配规则实战

3.1 基于浏览器类型匹配

```
#前端配置
frontend web_in_80
    bind 10.0.0.5:80
    mode http
    use_backend web_server_80

    #acl设置
    acl acl_user_agent_mobile hdr_sub(User-Agent) -i iphone Android
    acl acl_user_agent_pc hdr_sub(User-Agent) -i chrome
    acl acl_user_agent_linux hdr_sub(User-Agent) -i curl wget

    #acl调用
    use_backend mobile_hosts if acl_user_agent_mobile
```

```
    use_backend linux_hosts if acl_user_agent_linux
    default_backend pc_hosts if acl_user_agent_pc

#后端配置
backend mobile_hosts
    mode http
    server web1 10.0.0.7:80 check inter 3000 fall 3 rise 5

backend linux_hosts
    mode http
    server web1 10.0.0.8:80 check inter 3000 fall 3 rise 5

backend pc_hosts
    mode http
    server web1 10.0.0.7:80 check inter 3000 fall 3 rise 5
    server web1 10.0.0.8:80 check inter 3000 fall 3 rise 5
```

3.2 基于文件后缀名匹配

```
#前端配置
frontend web_in_80
    bind 10.0.0.5:80
    mode http
    use_backend web_server_80

    #acl设置
    acl acl_img path_end -i .jpg .png

    #acl调用
    use_backend static_hosts if acl_img
    default_backend web_hosts

#后端配置
backend static_hosts
    mode http
    server web1 10.0.0.7:80 check inter 3000 fall 3 rise 5

backend web_hosts
    mode http
    server web1 10.0.0.7:80 check inter 3000 fall 3 rise 5
    server web1 10.0.0.8:80 check inter 3000 fall 3 rise 5
```

3.3 基于域名匹配

```
#前端配置
frontend web_in_80
    bind 10.0.0.5:80
    mode http
    use_backend web_server_80

    #acl设置
    acl acl_img path_end -i .jpg .png

    #acl调用
    use_backend static_hosts if acl_img
    default_backend web_hosts
```

```
13
14 #后端配置
15 backend static_hosts
16     mode http
17     server web1 10.0.0.7:80 check inter 3000 fall 3 rise 5
18
19 backend web_hosts
20     mode http
21     server web1 10.0.0.7:80 check inter 3000 fall 3 rise 5
22     server web1 10.0.0.8:80 check inter 3000 fall 3 rise 5
```

3.4 基于IP匹配

```
1  #前端配置
2  frontend web_in_80
3      bind 10.0.0.5:80
4      mode http
5      use_backend web_server_80
6
7      #acl设置
8      acl acl_ip_172 src 172.16.1.0/24
9
10     #acl调用
11     use_backend 172_hosts if acl_ip_172
12
13 #后端配置
14 backend 172_hosts
15     mode http
16     server web1 172.16.1.7:80 check inter 3000 fall 3 rise 5
```

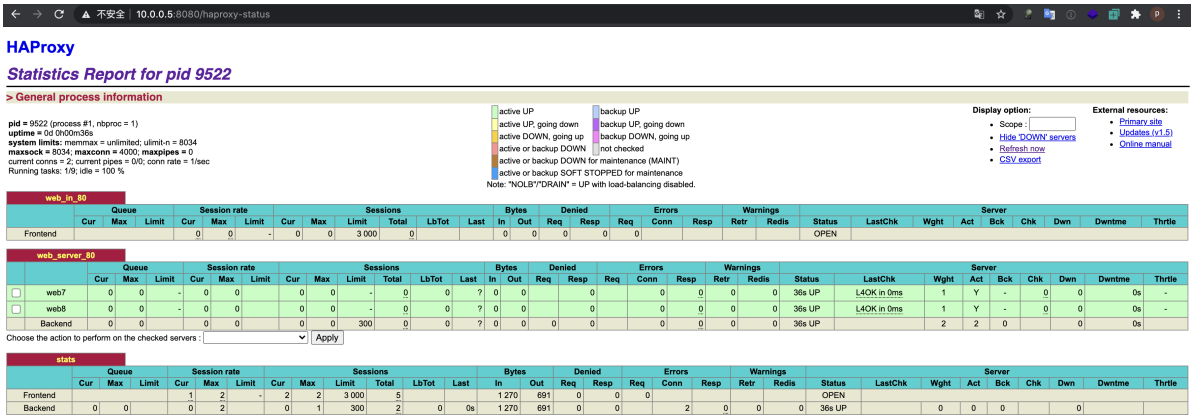
第4章 Haproxy高级功能

1.web服务状态监控

配置启动监控状态页面：

```
1 listen stats
2     bind :8080                    #监听端口
3     stats enable                  #启用状态页
4     stats hide-version  #隐藏版本号
5     stats uri /haproxy-status #自定义状态页路径
6     stats realm HAProxy\ Statistics\ Page #自定义提示信息
7     stats auth oldboy:123456         #定义普通用户密码
8     stats auth admin:123456          #定义admin用户密码
9     stats admin if TRUE                  #启用管理功能
```

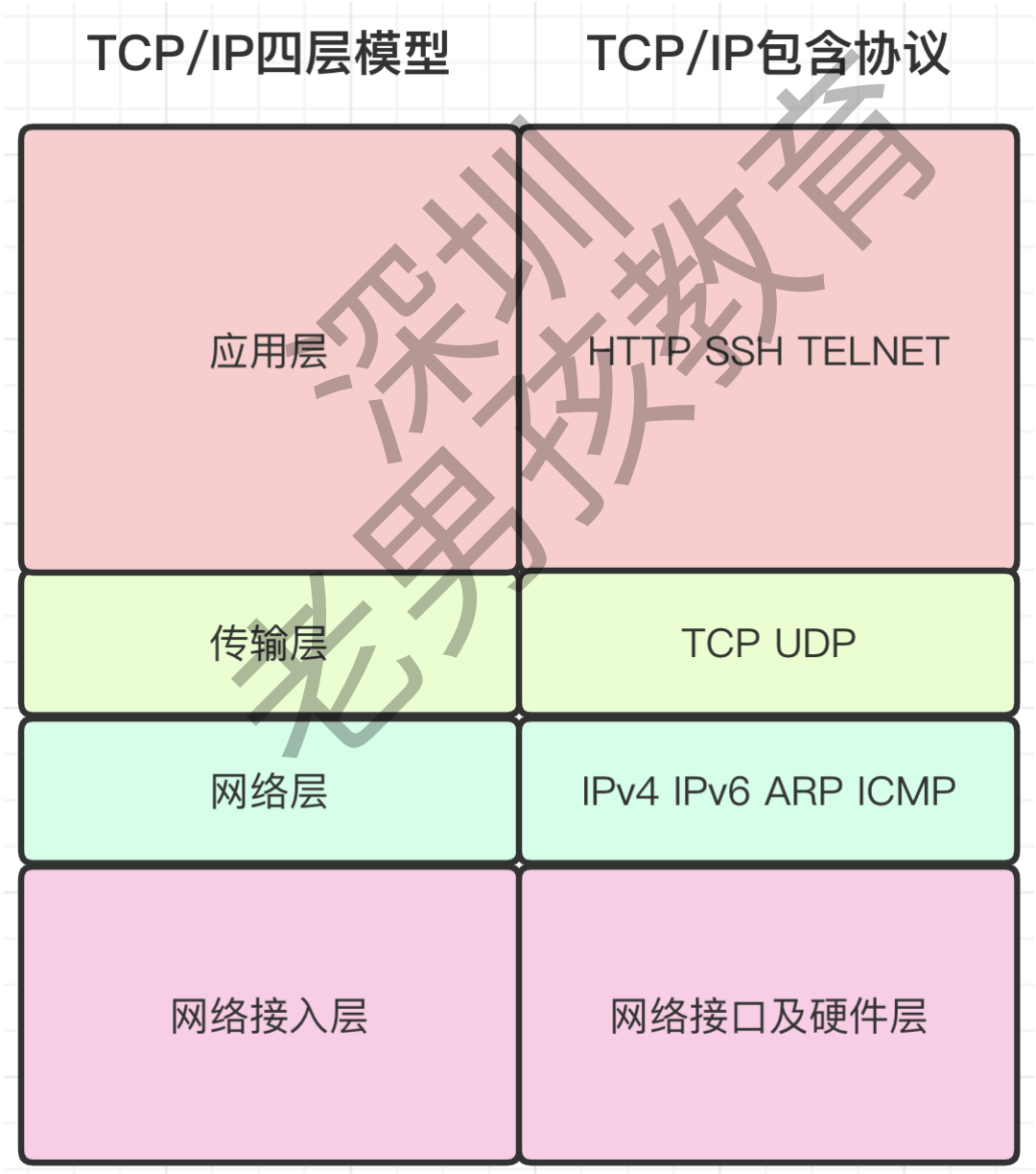
访问并查看状态页面：



状态字段解释：

```
1
```

2.基于四层的负载



四层负载与七层负载区别:

```
四层负载：
所谓四层负载均衡指的是OSI七层模型中的传输层
Nginx已经能支持TCP/IP的控制，所以只需要对客户端的请求进行TCP/IP协议的包转发就可以实现负载均衡
它的好处是性能非常快、只需要底层进行应用处理，而不需要进行一些复杂的逻辑

七层负载均衡：
七层负载是指应用层，它可以完成很多应用方面的协议请求
比如我们说的http应用的负载均衡，它可以实现http信息的改写、头信息的改写、安全应用规则控制、URL匹配规则控制、以及转发,rewrite等等的规则
所以在应用层的服务里面，我们可以做的内容就更多，那么Nginx则是一个典型的七层负载均衡
```

haproxy四层负载配置:

```
listen redis-server
    bind 10.0.0.5:6379
    mode tcp
    balance leastconn
    server redis1 10.0.0.51:6379 check
    server redis2 10.0.0.52:6379 check backup
```

haproxy四层负载访问测试: 前提是后端服务器安装好了redis并配置允许远程访问

```
redis-cli 10.0.0.5
>keys *
```