6_Operation03NIGNX 调度器优化

一 案例 1：Nginx 反向代理(http 代理:7 层代理)

代理:帮你干活的人或物,拥有调度的功能

反向代理功能,实现 web 服务器的负载均衡,服务器集群高可用;对 web 服务器有健康检查功能;在案例中,Nginx 服务器不部署 web 服务,仅作为代理(调度)服务器.

1.1 问题

使用 Nginx 实现 Web 反向代理功能，实现如下功能：

后端 Web 服务器两台，可以使用 httpd 实现

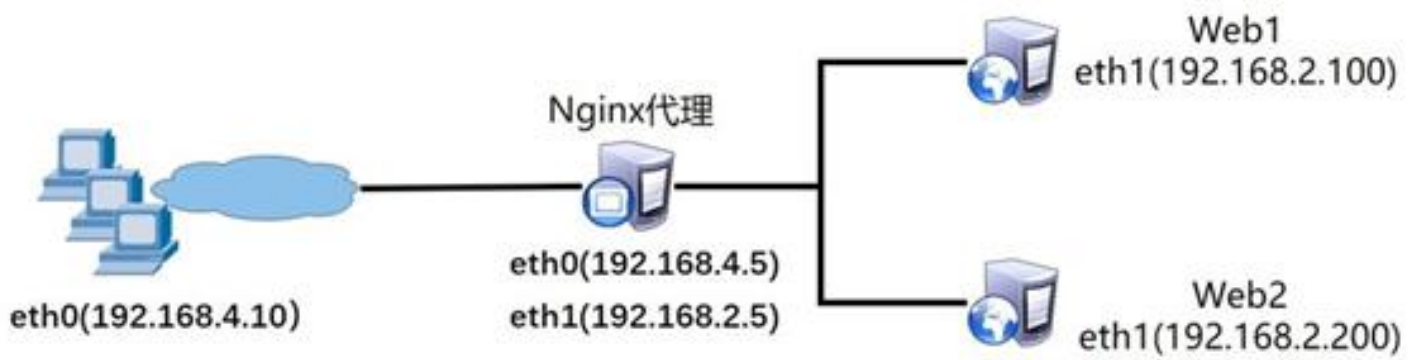
Nginx 采用轮询的方式调用后端 Web 服务器

两台 Web 服务器的权重要求设置为不同的值

最大失败次数为 1，失败超时时间为 30 秒

1.2 方案

使用 4 台 RHEL7 虚拟机，其中一台作为 Nginx 代理服务器，该服务器需要配置两块网卡，IP 地址分别为 192.168.4.5 和 192.168.2.5，两台 Web 服务器 IP 地址分别为 192.168.2.100 和 192.168.2.200。客户端测试主机 IP 地址为 192.168.4.10。如图所示。



1.3 步骤

前置条件 nginx 能 ping 通其他 3 台机器,client 不能和 web ping 通

1.3.1 部署实施后端 Web 服务器

1）部署后端 Web1 服务器

后端 Web 服务器可以简单使用 yum 方式安装 httpd 实现 Web 服务，为了可以看出后端服务器的不同，可以将两台后端服务器的首页文档内容设置为不同的内容。

```
web1 ~]# yum  -y  install  httpd
web1 ~]# echo "192.168.2.100" > /var/www/html/index.html
web1 ~]# systemctl restart httpd
web1 ~]# firewall-cmd --set-default-zone=trusted
web1 ~]# setenforce 0
```

2）部署后端 Web2 服务器

```
web2 ~]# yum  -y  install  httpd
web2 ~]# echo "192.168.2.200" > /var/www/html/index.html
web2 ~]# systemctl restart httpd
web2 ~]# firewall-cmd --set-default-zone=trusted
web2 ~]# setenforce 0
```

1.3.2 配置 Nginx 服务器，添加服务器池，实现反向代理功能

1）修改/usr/local/nginx/conf/nginx.conf 配置文件

```
proxy~]# vim /usr/local/nginx/conf/nginx.conf
```

#使用 upstream 定义后端服务器集群，集群名称任意(如 webserver)

#使用 server 定义集群中的具体服务器和端口 #在 server 上面写

```
upstream webserver {
            server 192.168.2.100:80;
            server 192.168.2.200:80;
}
#gzip on;        #定位语句,在此句上面定义集群
server {
     listen        80;
     server_name  localhost;
          location / {
          proxy_pass http://webserver;
#通过 proxy_pass 将用户的请求转发给 webserver 集群,优先级比 root 高
     }
}
```

2）重新加载配置

```
proxy~]# /usr/local/nginx/sbin/nginx -s reload
```

3）客户端使用浏览器访问代理服务器测试轮询效果

```
client~]# curl http://192.168.4.5     #使用该命令多次访问查看效果
```

1.3.3 配置 upstream 服务器集群池属性

1）设置失败次数，超时时间，权重 weight 可以设置后台服务器的权重，max_fails 可以设置后台服务器的失败次数，fail_timeout 可以设置后台服务器的失败超时时间。

```
proxy~]# vim /usr/local/nginx/conf/nginx.conf
upstream webserver {
     server 192.168.2.100 weight=1 max_fails=1 fail_timeout=30;
     server 192.168.2.200 weight=2 max_fails=2 fail_timeout=30;
     server 192.168.2.101 down;
      }
     #空格分隔
     #weight 设置服务器权重值，默认值为 1,设置该台服务器在服务器集群负载分担:本服务器负载分担比例=本服务器权重值/服务器集群重权重值
     #max_fails 设置最大失败次数,若调度该机失败 n 次,则表示该机不健康
     #fail_timeout 设置失败超时时间，单位为秒,表示多少秒内不调度该机
     #down 标记服务器已关机，不参与集群调度
```

2）重新加载配置

```
proxy~]# /usr/local/nginx/sbin/nginx -s reload
```

3）关闭一台后端服务器（如 web1）

```
web1 ~]# systemctl stop httpd
```

4）客户端使用浏览器访问代理服务器测试轮询效果

client~]# curl http://192.168.4.5 #使用该命令多次访问查看效果

5）再次启动后端服务器的 httpd（如 web1）

web1 ~]# systemctl start httpd

6）客户端再次使用浏览器访问代理服务器测试轮询效果

client~]# curl http://192.168.4.5 #使用该命令多次访问查看效果

1.3.4 配置 upstream 服务器集群的调度算法

1）设置相同客户端访问相同 Web 服务器

proxy~]# vim /usr/local/nginx/conf/nginx.conf

```
upstream webserver {
      ip_hash;  #通过 ip_hash 设置调度规则为：相同客户端访问相同服务器
      server 192.168.2.100 weight=1 max_fails=2 fail_timeout=10;
      server 192.168.2.200 weight=2 max_fails=2 fail_timeout=10;
       }
```

2）重新加载配置

proxy~]# /usr/local/nginx/sbin/nginx -s reload

3）客户端使用浏览器访问代理服务器测试轮询效果

client~]# curl http://192.168.4.5   #使用该命令多次访问查看效果

#当某客户端的相同服务器出问题时,该客户机的访问会被调度到其他服务器,当该机的相同服务器恢复后,该客户机的访问会被调度到恢复后的相同服务器.

二 案例 2：Nginx 的 TCP/UDP 调度器(4 层调度器)

2.1 问题

使用 Nginx 实现 TCP/UDP 调度器功能，实现如下功能：

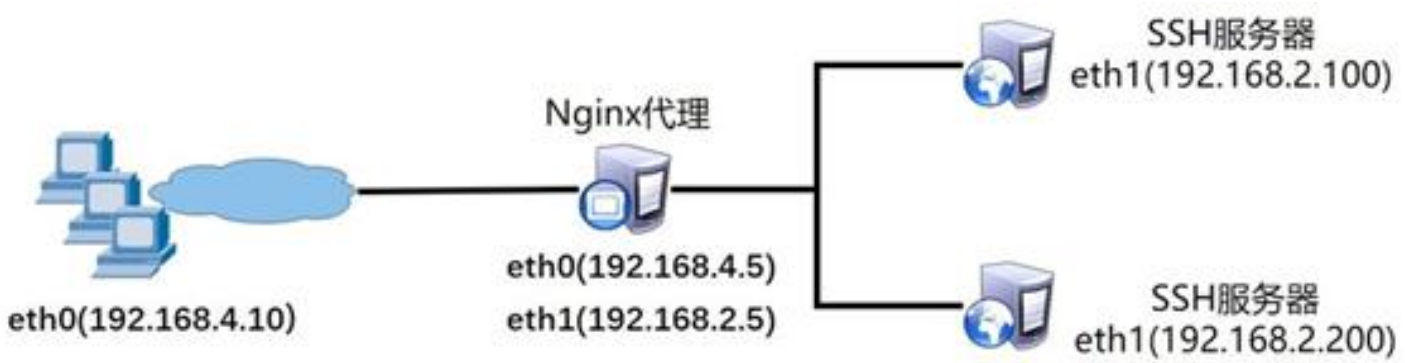
后端 SSH 服务器两台

Nginx 编译安装时需要使用--with-stream，开启 ngx_stream_core_module 模块

Nginx 采用轮询的方式调用后端 SSH 服务器

2.2 方案

使用 4 台 RHEL7 虚拟机，其中一台作为 Nginx 代理服务器，该服务器需要配置两块网卡，IP 地址分别为 192.168.4.5 和 192.168.2.5，两台 SSH 服务器 IP 地址分别为 192.168.2.100 和 192.168.2.200。客户端测试主机 IP 地址为 192.168.4.10。如图所示。



2.3 步骤

2.3.1 部署支持 4 层 TCP/UDP 代理的 Nginx 服务器(4 层代理)

1）部署 nginx 服务器

卸载 nginx:killall nginx,rm -rf /usr/local/nginx,重装

编译安装必须要使用--with-stream 参数开启 4 层代理模块。

```
proxy ~]# yum -y install gcc pcre-devel openssl-devel
proxy ~]# tar  -xf   nginx-1.12.2.tar.gz
proxy ~]# cd  nginx-1.12.2
proxynginx-1.12.2]# ./configure   \
> --with-http_ssl_module  \            #开启 SSL 加密功能
> --with-stream                        #开启 4 层反向代理功能
proxynginx-1.12.2]# make && make install    #编译并安装
```

2.3.2 配置 Nginx 服务器，添加服务器池，实现 TCP/UDP 反向代理功能

1）修改/usr/local/nginx/conf/nginx.conf 配置文件

```
proxy~]# vim /usr/local/nginx/conf/nginx.conf
```

配置文件中书写在 http{ } 之外:

```
stream {
     upstream backend {          #定义集群
     server 192.168.2.100:22;    #后端 SSH 服务器的 IP 和端口
     server 192.168.2.200:22;
     }
     server {
          listen 12345;  #Nginx 监听的端口,同 1 端口只能 1 个服务使用
          proxy_connect_timeout 1s;  #连接的超时时间，可选配置
          proxy_timeout 3s;  #连接的超时时间，可选配置
```

```
            proxy_pass backend;  #将请求转发给 backend
        }
}
```

2）重新加载配置

```
proxy~]# /usr/local/nginx/sbin/nginx -s reload
```

3）客户端使用访问代理服务器测试轮询效果

```
client~]# ssh 192.168.4.5 -p 12345  #使用该命令多次访问查看效果
```

三 案例 3：Nginx 常见问题处理

3.1 问题

本案例要求对 Nginx 服务器进行适当优化，解决如下问题，以提升服务器的处理性能：

如何自定义返回给客户端的 404 错误页面

如何查看服务器状态信息

如果客户端访问服务器提示“Too many open files”如何解决

如何解决客户端访问头部信息过长的问题

如何让客户端浏览器缓存数据

日志切割

开启 gzip 压缩功能，提高数据传输效率

开启文件缓存功能

然后客户机访问此 Web 服务器验证效果：

使用 ab 压力测试软件测试并发量

编写测试脚本生成长头部信息的访问请求

客户端访问不存在的页面，测试 404 错误页面是否重定向

3.2 步骤

实现此案例需要按照如下步骤进行。

3.2.1 自定义报错 404 页面

1）优化前，客户端使用浏览器访问不存在的页面，会提示 404 文件未找到

```
client~]# firefox http://192.168.4.5/xxxxx  #访问一个不存在的页面
```

2）修改 Nginx 配置文件，自定义报错页面

```
proxy~]# vim /usr/local/nginx/conf/nginx.conf
    charset utf-8;  #仅需要中文时需要改选项，可选项
    error_page   404  /404.html;  #自定义错误页面(有注释,解除即可)
proxy~]# vim /usr/local/nginx/html/404.html  #生成错误页面
Oops,No NO no page …
proxy~]# /usr/local/nginx/sbin/nginx -s reload
```

3）优化后，客户端使用浏览器访问不存在的页面，会提示自己定义的 40x.html 页面

```
client~]# firefox http://192.168.4.5/xxxxx   #访问一个不存在的页面
```

4）常见 http 状态码

状态码	功能描述	
200	一切正常	
301	永久重定向	#300 类代表重定向

302 临时重定向

401 用户名或密码错误 #400 类表示用户端错误

403 禁止访问(客户端 IP 地址被拒绝)

404 文件(页面)不存在

414 请求 URI 头部过长

500 服务器内部错误 #500 类表示服务器端错误

502 bad gateway

3.2.2 如何查看服务器状态信息（非常重要的功能）

1）编译安装时使用--with-http_stub_status_module 开启状态页面模块

```
proxy~]# tar  -zxvf   nginx-1.12.2.tar.gz
proxy~]# cd  nginx-1.12.2
proxynginx-1.12.2]# ./configure   \
> --with-http_ssl_module \                  #开启 SSL 加密功能
> --with-stream \                           #开启 TCP/UDP 代理模块
> --with-http_stub_status_module            #开启 status 状态页面
proxynginx-1.12.2]# make && make install  #编译并安装
```

2）启用 Nginx 服务并查看监听端口状态

ss 命令可以查看系统中启动的端口信息，该命令常用选项如下：

-a 显示所有端口的信息

-n 以数字格式显示端口号

-t 显示 TCP 连接的端口

-u 显示 UDP 连接的端口

-l 显示服务正在监听的端口信息，如 httpd 启动后，会一直监听 80 端口

-p 显示监听端口的服务名称是什么（也就是程序名称）

注意：在 RHEL7 系统中可以使用 ss 命令替代 netstat 命令，功能一样，选项一样。

```
proxy~]# /usr/local/nginx/sbin/nginx
proxy~]# netstat  -anptu  |  grep nginx
tcp   0   0 0.0.0.0:80   0.0.0.0:*   LISTEN   10441/nginx
proxy~]# ss  -anptu  |  grep nginx
```

3）修改 Nginx 配置文件，定义状态页面

```
proxy~]# cat /usr/local/nginx/conf/nginx.conf
#error_page  404 行上面
location /status {
            stub_status on;
                #allow IP 地址;    #可选项
                #deny IP 地址;     #可选项
      }
proxy~]# nginx
```

4）优化后，查看状态页面信息

```
proxy~]# curl  http://192.168.4.5/status
```

```
Active connections: 1
server accepts handled requests
 10 10 3
Reading: 0 Writing: 1 Waiting: 0
```

Active connections：当前活动的连接数量。

Accepts：已经接受客户端的连接总数量。

Handled：已经处理客户端的连接总数量。

（一般 handled 与 accepts 一致，除非服务器限制了连接数量）。

Requests：客户端发送的请求数量。

Reading：当前服务器正在读取客户端请求头的数量。

Writing：当前服务器正在写响应信息的数量。

Waiting：当前多少客户端在等待服务器的响应。

```
pv= wc -l access.log
```

uv= awk 正则统计 IP 数量

3.2.3 优化 Nginx 并发量

1）优化前使用 ab 高并发测试(没优化前的测试)

```
proxy~]# ab -c 2000 -n 2000 http://192.168.4.5/
```

-c 人数;-n number,总访问量;必须 n>=c

```
Benchmarking 192.168.4.5 (be patient)
socket: Too many open files (24)  #提示打开文件数量过多
```

2）修改 Nginx 配置文件，增加并发量

```
proxy~]# vim /usr/local/nginx/conf/nginx.conf
worker_processes  2;  #与 CPU 核心数量一致
events {
worker_connections 65535;  #每个 worker 最大并发连接数
}
proxy~]# /usr/local/nginx/sbin/nginx -s reload
```

3）优化 Linux 内核参数（最大文件数量）

```
proxy~]# ulimit -a                  #查看所有属性值
proxy~]# ulimit -Hn 100000          #设置硬限制（临时规则）20000 最大
proxy~]# ulimit -Sn 100000          #设置软限制（临时规则）20000 最大
```

-H:硬件限制;-S:软件限制;-n:numerber,数量

```
proxy~]# vim /etc/security/limits.conf
#该配置文件分 4 列，分别如下：nofile=number of file
#用户或组 硬限制或软限制 需要限制的项目    限制的值
*         soft         nofile         100000
*         hard         nofile         100000
```

4）优化后测试服务器并发量（因为客户端没调内核参数，所以在 proxy 测试）

```
proxy~]# ab -n 2000 -c 2000 http://192.168.4.5/
```

3.2.4 优化 Nginx 数据包头缓存

1）优化前，使用脚本测试长头部请求是否能获得响应

```
proxy~]# cat lnmp_soft/buffer.sh
#!/bin/bash
URL=http://192.168.4.5/index.html?
for i in {1..5000}
do
    URL=${URL}v$i=$i
done
curl $URL  #经过 5000 次循环后，生成一个长的 URL 地址栏
proxy~]# ./buffer.sh
<center><h1>414 Request-URI Too Large</h1></center>
      #提示头部信息过大
```

2）修改 Nginx 配置文件，增加数据包头部缓存大小

```
proxy~]# vim /usr/local/nginx/conf/nginx.conf
.. ..
http {
client_header_buffer_size 1k;    #默认请求包头信息的缓存
large_client_header_buffers 4 1m;#最大请求包头部信息的缓存个数与容量
.. ..                                    #4 个 4k,或 4 个 1m
}
```

```
proxy~]# /usr/local/nginx/sbin/nginx -s reload
```

3）优化后，使用脚本测试长头部请求是否能获得响应

```
proxy~]#cat cat buffer.sh
#!/bin/bash
URL=http://192.168.4.5/index.html?
for i in {1..5000}
do
    URL=${URL}v$i=$i
done
curl $URL
proxy~]# ./buffer.sh
```

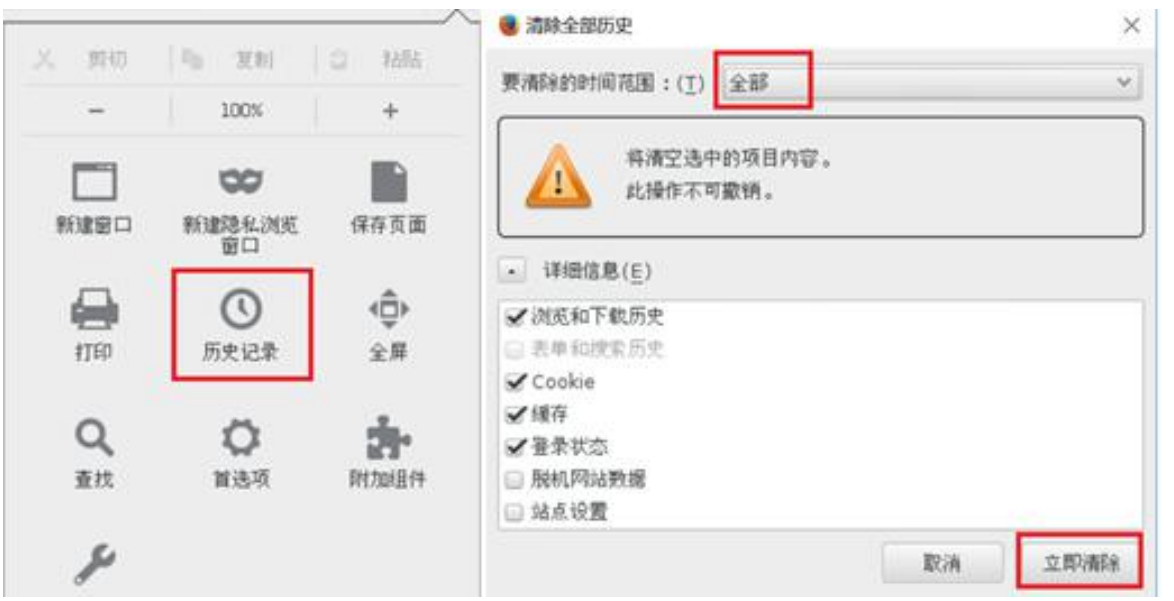
3.2.5 浏览器本地缓存静态数据

图片,音频,视频等静态资料适合缓存,缓存的时间长短右服务器决定

1）使用 Firefox 浏览器查看缓存

以 Firefox 浏览器为例,在 Firefox 地址栏内输入 **about:cache** 将显示 Firefox 浏览器的缓存信息，如图所示，点击 List Cache Entries 可以查看详细信息。

2）清空 firefox 本地缓存数据，如图所示。



3）修改 Nginx 配置文件，定义对静态页面的缓存时间

```
proxy~]# vim /usr/local/nginx/conf/nginx.conf
server {    在#charset koi8-r;语句上写
    listen       80;
```

```
    server_name  localhost;
    location / {
        root   html;
        index  index.html index.htm;
     }
location ~* \.(jpg|jpeg|gif|png|css|js|ico|xml)$ {
expires        30d;   #定义客户端缓存时间为30天
}   #正则表达式:~包含   *不区分大小写   \转义后面的.号
}
proxy~]# cp /usr/share/backgrounds/day.jpg /usr/local/nginx/html
proxy~]# /usr/local/nginx/sbin/nginx -s reload
```

4）优化后，使用 Firefox 浏览器访问图片，再次查看缓存信息

```
client~]# firefox http://192.168.4.5/day.jpg
```

在 firefox 地址栏内输入 about:cache，查看本地缓存数据，查看是否有图片以及过期时间是否正确。

3.2.6 日志切割

日志文件越来越大怎么办？单个文件 10G？如何切割？（非常常见的面试题）

步骤：

1. 把旧的日志重命名

2. kill -USR1 PID(nginx 的进程 PID 号)

kill -USR1 相当于 **kill -10, kill -l** 查看所有 **kill** 的选项

1）手动执行

备注：**/usr/local/nginx/logs/nginx.pid** 文件中存放的是 **nginx** 的进程 **PID** 号。

```
proxy~]#  mv access.log access2.log
proxy~]# kill -USR1 $(cat /usr/local/nginx/logs/nginx.pid)
```

2）自动完成

每周 5 的 03 点 03 分自动执行脚本完成日志切割工作。

```
proxy~]# vim /usr/local/nginx/logbak.sh
#!/bin/bash
date=`date +%Y%m%d`   #定义变量 date
logpath=/usr/local/nginx/logs    #定义变量 logpath
mv $logpath/access.log $logpath/access-$date.log  #备份访问日志
mv $logpath/error.log $logpath/error-$date.log    #备份错误日志
kill -USR1 $(cat $logpath/nginx.pid)              #创建新日志
proxy~]# crontab -e   #创建计划任务
03 03 * * 5  /usr/local/nginx/logbak.sh
```

3.2.7 对页面进行压缩处理

1）修改 **Nginx** 配置文件

```
proxy~]# cat /usr/local/nginx/conf/nginx.conf
```

```
http {
gzip on;  #服务器很重要的优化之一,提高网速  #开启压缩,解除此行的注释
gzip_min_length 1000;  #压缩文件最小长度(大小)1000 字节
gzip_comp_level 4;  #压缩比率,范围 1-9,数字越大压缩效果越好,速度最慢
gzip_types text/plain text/css application/json application/x
-javascript text/xml application/xml application/xml+rss text/
javascript;
#对特定文件压缩，类型参考/usr/local/nginx/conf/mime.types
}
```

3.2.8 服务器内存缓存

1）如果需要处理大量静态文件，可以将文件缓存在内存，下次访问会更快。

```
http {
     open_file_cache  max=2000  inactive=20s;
     open_file_cache_valid      60s;
     open_file_cache_min_uses   5;
     open_file_cache_errors     off;  #关闭缓存错误报告
#设置服务器最大缓存 2000 个文件句柄，关闭 20 秒内无请求的文件句柄
#文件句柄的有效时间是 60 秒，60 秒后过期  #只有访问次数超过 5 次会被缓存
}
```