Nginx反向代理服务器及负载均衡服务配置实战

前言：什么是Nginx?

Nginx (engine x) 是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务，能够支持高达 50,000 个并发连接数的响应。Nginx是由伊戈尔·赛索耶夫为俄罗斯访问量第二的Rambler.ru站点（俄文：Рамблер）开发的，第一个公开版本0.1.0发布于2004年10月4日。

其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011年6月1日，nginx 1.0.4发布。

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力在同类型的网页服务器中表现较好，中国大陆使用nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

nginx的优点：

1、轻量级，同样起web 服务，比apache 占用更少的内存及资源。

2、抗并发，nginx 处理请求是异步非阻塞的，而apache 则是阻塞型的，在高并发下nginx 能保持低资源低消耗高性能。官方号称可达5万并发，而apache tomcat默认则只有150，经过调优一般也不超过500，否则性能会降到极限

3、高度模块化的设计。

1、Centos7下Nginx的安装和配置

1.1 安装依赖环境

```
yum install gcc-c++ pcre pcre-devel zlib zlib-devel openssl openssl-devel
```

1.2 下载安装包并解压,安装

```
wget http://nginx.org/download/nginx-1.12.2.tar.gz
tar zxvf nginx-1.12.2.tar.gz
cd nginx-1.12.2
#配置编译选项
./configure --prefix=/usr/local/nginx/
#编译安装
make && make install
```

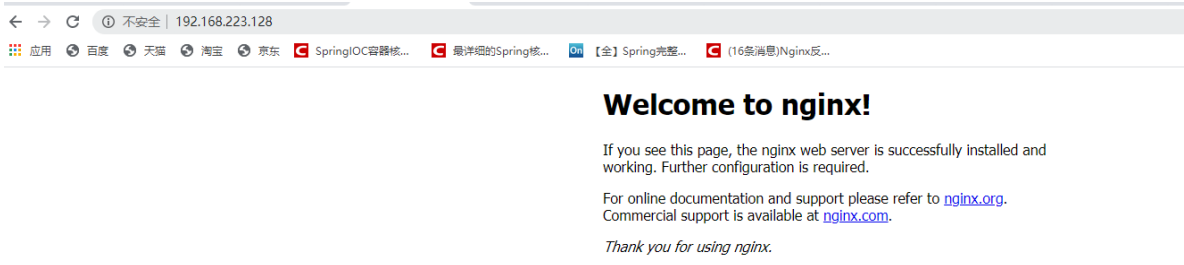
1.3 启动

我们安装后会在nginx-1.12.2外层目录下看到一个nginx的目录，进去，脚本启动

```
cd /usr/local/nginx/
./sbin/nginx #启动
./sbin/nginx -s reload #重启
```

```
./sbin/nginx -s stop #关闭
./sbin/nginx -t #测试配置文件是否正确

当然，我们也可以将Nginx配置成服务进行启动
vim /lib/systemd/system/nginx.service
#把以下内容复制进去

[Unit]
Description=nginx
After=network.target

[Service]
Type=forking
ExecStart=/usr/local/nginx/sbin/nginx #启动脚本
ExecReload=/usr/local/nginx/sbin/nginx -s reload #重启脚本
ExecStop=/usr/local/nginx/sbin/nginx -s quit #关闭脚本
PrivateTmp=true

[Install]
WantedBy=multi-user.target

#执行（如有报错按人家提示来）
systemctl daemon-reload #如果是第一次运行，需要开启保护模式加载
systemctl start nginx
```

看到这个，表示安装启动成功！



2、Nginx能做什么

- 反向代理
- 负载均衡
- HTTP服务器（包含动静分离）
- 正向代理

3、反向代理

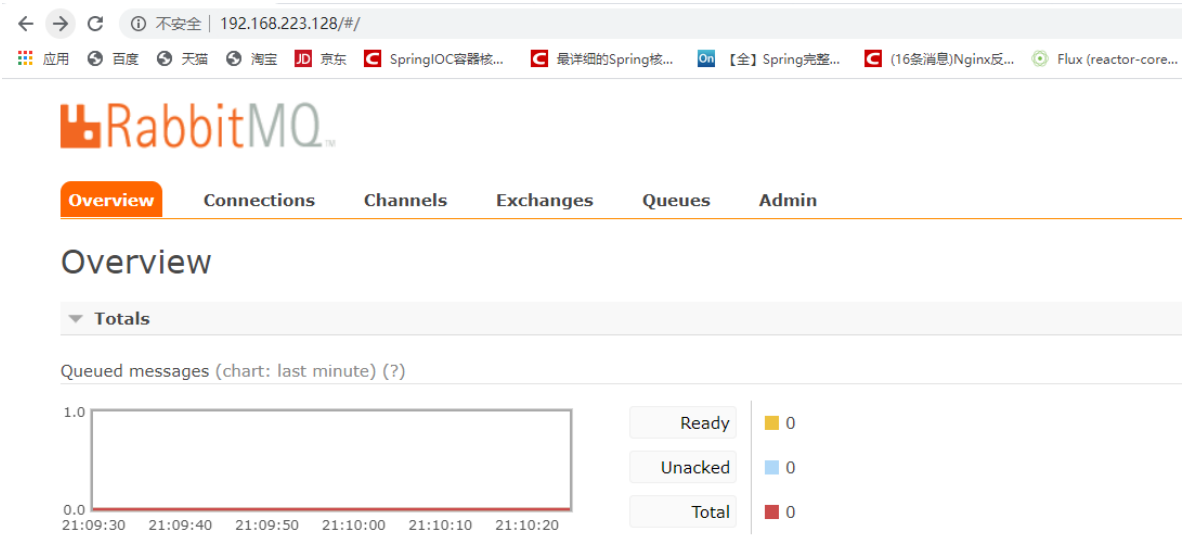
反向代理应该是Nginx做的最多的一件事了，什么是反向代理呢，以下是百度百科的说法：反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。简单来说就是真实的服务器不能直接被外部网络访问，所以需要一台代理服务器，而代理服务器能被外部网络访问的同时又跟真实服务器在同一个网络环境，当然也可能是同一台

服务器，端口不同而已。
下面贴上一段简单的实现反向代理的代码

nginx.conf配置：

```
    server {
        listen       80; //对外暴露的监听端口

        server_name  localhost; //对外暴露的监听地址IP

        client_max_body_size 1024M; //客户端最大请求体

        location / {
            proxy_pass http://192.168.223.129:15672; //代理转发的内部服务器地址
        }
    }
```

保存配置文件后启动Nginx，这样当我们访问192.168.223.128:80的时候，就相当于访问192.168.223.128:15672了



4、负载均衡

负载均衡也是Nginx常用的一个功能，负载均衡其意思就是分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。简单而言就是当有2台或以上服务器时，根据规则随机的将请求分发到指定的服务器上处理，**负载均衡配置一般都需要同时配置反向代理，通过反向代理跳转到负载均衡。**而Nginx目前支持自带3种负载均衡策略，还有2种常用的第三方策略。

4.1 轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

nginx.conf配置：

```
    upstream test {
        server 192.168.223.128:15672; #内部服务器1
        server 192.168.223.129:15672; #内部服务器2
    }
    server {
        listen       80;
        server_name  localhost;
        client_max_body_size 1024M;

        location / {
            proxy_pass http://test; #负载均衡服务集群
        }
    }
```

这里我配置了2台服务器，当然实际上是一台，只是端口不一样而已，两台服务器正常的情况下，两者会交替调用。

即使8081的服务器是不存在的,也就是说访问不到，但是我们访问http://localhost 的时候,也不会有问题，会默认跳转到http://192.168.0.122:8080 具体是因为Nginx会自动判断服务器的状态，如果服务器处于不能访问（服务器挂了），就不会跳转到这台服务器，所以也避免了一台服务器挂了影响使用的情况，由于Nginx默认是轮询策略，所以我们不需要其他更多的设置。

4.2 权重

指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

```
    upstream test {
        server 192.168.223.128:15672 weight=9; #内部服务器1
        server 192.168.223.129:15672 weight=1; #内部服务器2
        server 192.168.223.130:15672 backup; #只要128,129都挂了，它才会起作用
    }
```

那么10次一般只会有1次会访问到8081，而有9次会访问到8080,注意了，并不是一定要8080执行了九次，8081才会执行，这是一个概率事件!

4.3 ip_hash

上面的2种方式都有一个问题，那就是下一个请求来的时候请求可能分发到另外一个服务器，当我们的程序不是无状态的时候（采用了session保存数据），这时候就有一个很大的很问题了，比如把登录信息保存到了session中，那么跳转到另外一台服务器的时候就需要重新登录了，所以很多时候我们需要一个客户只访问一个服务器，那么就需要用ip_hash了，ip_hash的每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

PS：外网IP哦!

```
    upstream test {
        ip_hash;
        server 192.168.223.128:15672; #内部服务器1
        server 192.168.223.129:15672; #内部服务器2
    }
```

4.4 fair（第三方）

注意：因为插件兼容性问题，需要使用nginx-1.11.5.tar.gz版本，安装过程跟nginx-1.12.2.tar.gz一样

需要先下载fair包：nginx-upstream-fair-master.zip（已备好），放在nginx安装目录下比较好

```
unzip nginx-upstream-fair-master.zip #解压安装包

#如果已安装nginx，那么需要添加nginx-upstream-fair-master模块
cd /usr/local/nginx-1.11.5
./configure --prefix=/usr/local/nginx --add-module=/usr/local/nginx-upstream-fair-master #nginx-upstream-fair-master解压目录
#编译
make & make install
```

fair采用的不是内建负载均衡使用的轮换的均衡算法，而是按后端服务器的响应时间来分配请求，响应时间短的优先，智能的进行负载均衡。也可以称为最优服务器策略

```
    upstream test {
        fair;
        server 192.168.223.128:15672; #内部服务器1
        server 192.168.223.129:15672; //内部服务器2
    }
```

4.5 url_hash（第三方）

下载安装第三方模块：

```
wget https://github.com/replay/ngx_http_consistent_hash/archive/master.zip
unzip master.zip

cd /usr/local/nginx-1.12.2
./configure --prefix=/usr/local/nginx --add-module=/usr/local/ngx_http_consistent_hash-master
#编译
make & make install
```

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。 在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method是使用的hash算法（可选，默认使用）

```
    upstream test {
        hash $request_uri;
         server 192.168.223.128:15672; #内部服务器1
         server 192.168.223.129:15672; #内部服务器2
    }
```

测试结果：同一个url访问会进入到同一个后端服务器

总结：以上5种负载均衡各自适用不同情况下使用，所以可以根据实际情况选择使用哪种策略模式,不过fair和url_hash需要安装第三方模块才能使用

5、HTTP服务器

Nginx本身也是一个静态资源的服务器，当只有静态资源的时候，就可以使用Nginx来做服务器，同时现在也很流行动静分离，就可以通过Nginx来实现，首先看看Nginx做静态资源服务器

```
server {
        listen       80;
        server_name  localhost;
        client_max_body_size 1024M;

        location / { #当然，该处你也可以设置访问的静态资源类型
               root   html; #表示/usr/local/html目录下所有文件按都可以作为静态资源访问
               index  index.html;
        }
}
```

测试结果：

6、正向代理

正向代理：局域网中的客户端不能直接访问Internet，则需要通过代理服务器来访问，这种代理服务就称为正向代理。Nginx本身只支持http的正向代理，并通过ngx_http_proxy_connect_module模块支持http、https的正向代理；

假设现在的环境是，局域网中只有一台机器：192.168.223.128配有某个公网IP，如此可以直接访问公网，而其他局域网服务器需要通过128上的代理来访问公网：

6.1 安装第三方https代理模块

Nginx本身是不支持https协议请求转发，为了让nginx能达到这一效果需要借助第三方模块ngx_http_proxy_connect_module。首先下载这一模块：https://github.com/chobits/ngx_http_proxy_connect_module到服务器：

```
git clone https://github.com/chobits/ngx_http_proxy_connect_module.git
cd /usr/local/nginx-1.12.2
patch -p1 < /usr/local/ngx_http_proxy_connect_module/patch/proxy_connect.patch
./configure --prefix=/usr/local/nginx --add-
module=/usr/local/ngx_http_proxy_connect_module
make &&  make install
```

6.2 配置正向代理

编辑正向代理配置，支持两种HTTP,HTTPS

```
 server {
      listen 80;
      resolver 8.8.8.8; #DNS服务器ip
      resolver_timeout 5s; #DNS服务器超时时间
      proxy_connect; #开启代理链接
      proxy_connect_allow 443 563; #代理连接允许端口，80已默认
      proxy_connect_connect_timeout 10s; #代理连接超时
      proxy_connect_read_timeout 10s; #代理连接读取超时
      proxy_connect_send_timeout 10s; #代理连接发送超时
      location / {
          proxy_pass $scheme://$host$request_uri; #代理链接$scheme：目标协议类型
$host：目标IP $request_uri：目标接口
          proxy_set_header Host $http_host; #解决如果 URL 中带 "."（点）后 Nginx
503 错误
          proxy_buffers 256 4k; #配置缓存大小
          proxy_max_temp_file_size 0; #关闭磁盘缓存读写，减少I/O
          proxy_connect_timeout 3000; #访问时代理连接超时时间
      }
    }
```

6.3 测试

```
----------------------------HTTP-----------------------------------------
[root@ydt1 nginx]# curl  -I --proxy 192.168.223.128:80 http://www.baidu.com
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Sat, 08 Aug 2020 08:07:08 GMT
Content-Type: text/html
Content-Length: 277
Connection: keep-alive
Accept-Ranges: bytes
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Etag: "575e1f6f-115"
Last-Modified: Mon, 13 Jun 2016 02:50:23 GMT
Pragma: no-cache



-------------------------------------------------------------------------
[root@ydt1 nginx]# curl  -I --proxy 192.168.223.128:80 https://www.baidu.com
HTTP/1.1 200 Connection Established
Proxy-agent: nginx

HTTP/1.1 200 OK
```

```
Accept-Ranges: bytes
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Connection: keep-alive
Content-Length: 277
Content-Type: text/html
Date: Sat, 08 Aug 2020 08:06:53 GMT
Etag: "575e1f6f-115"
Last-Modified: Mon, 13 Jun 2016 02:50:23 GMT
Pragma: no-cache
Server: bfe/1.0.8.18
```

6.4 Java API调用

```
import java.io.IOException;
import java.net.*;

public class Test {

    public static void main(String[] args) throws IOException {
        //目标地址
        URL url  = new URL("https://www.baidu.com");
        //代理服务器对象
        Proxy proxy
                = new Proxy(Proxy.Type.HTTP,new
InetSocketAddress("192.168.223.128",80));
        //通过代理服务器生成连接对象
        HttpURLConnection conn = (HttpURLConnection) url.openConnection(proxy);
        conn.connect();
        System.out.println(conn.getResponseCode()+":"
+conn.getResponseMessage());
    }
}
```