跟冰河学习Nginx技术

写在前面

> 在 **冰河技术** 微信公众号中的【Nginx】专题，更新了不少文章，有些读者反馈说，在公众号中刷历史文章不太方便，有时会忘记自己看到哪一篇了，当打开一篇文章时，似乎之前已经看过了，但就是不知道具体该看哪一篇了。相信很多小伙伴都会有这样的问题。那怎么办呢？最好的解决方案就是我把这些文章整理成PDF电子书，免费分享给大家，这样，小伙伴们看起来就方便多了。希望这本电子书能够给大家带来实质性的帮助。后续，我也会持续在 **冰河技术** 公众号中更新【Nginx】专题，如果这本电子书能够给你带来帮助，请关注 **冰河技术** 微信公众号，我们一起进阶，一起牛逼。

关于作者

大数据架构师，Mykit系列开源框架作者，多年来致力于分布式系统架构、微服务、分布式数据库、分布式事务与大数据技术的研究，曾主导过众多分布式系统、微服务及大数据项目的架构设计、研发和实施落地。在高并发、高可用、高可扩展性、高可维护性和大数据等领域拥有丰富的架构经验。对Hadoop，Storm，Spark，Flink等大数据框架源码进行过深度分析，并具有丰富的实战经验。目前在研究云原生。公众号【冰河技术】作者，《海量数据处理与大数据技术实战》、《MySQL技术大全：开发、优化与运维实战》作者，基于最终消息可靠性的开源分布式事务框架mykit-transaction-message作者。

一、安装Nginx

注意：这里以CentOS 6.8服务器为例，以root用户身份来安装Nginx。

1.安装依赖环境

```
yum -y install wget gcc-c++ ncurses ncurses-devel cmake make perl bison openssl
openssl-devel gcc* libxml2 libxml2-devel curl-devel libjpeg* libpng* freetype*
autoconf automake zlib* fiex* libxml* libmcrypt* libtool-ltdl-devel* libaio
libaio-devel  bzr libtool
```

2.安装openssl

```
wget https://www.openssl.org/source/openssl-1.0.2s.tar.gz
tar -zxvf openssl-1.0.2s.tar.gz
cd /usr/local/src/openssl-1.0.2s
./config --prefix=/usr/local/openssl-1.0.2s
make
make install
```

3.安装pcre

```
wget https://ftp.pcre.org/pub/pcre/pcre-8.43.tar.gz
tar -zxvf pcre-8.43.tar.gz
cd /usr/local/src/pcre-8.43
./configure --prefix=/usr/local/pcre-8.43
make
make install
```

4.安装zlib

```
wget https://sourceforge.net/projects/libpng/files/zlib/1.2.11/zlib-
1.2.11.tar.gz
tar -zxvf zlib-1.2.11.tar.gz
cd /usr/local/src/zlib-1.2.11
./configure --prefix=/usr/local/zlib-1.2.11
make
make
```

5.安装Nginx

```
wget http://nginx.org/download/nginx-1.17.2.tar.gz
tar -zxvf nginx-1.17.2.tar.gz
cd /usr/local/src/nginx-1.17.2
./configure --prefix=/usr/local/nginx-1.17.2 --with-
openssl=/usr/local/src/openssl-1.0.2s --with-pcre=/usr/local/src/pcre-8.43 --
with-zlib=/usr/local/src/zlib-1.2.11 --with-http_ssl_module
make
make install
```

这里需要注意的是：安装Nginx时，指定的是openssl、pcre和zlib的源码解压目录，安装完成后Nginx配置文件的完整路径为：/usr/local/nginx-1.17.2/conf/nginx.conf。

二、如何获取客户端真实IP、域名、协议、端口?

Nginx最为最受欢迎的反向代理和负载均衡服务器，被广泛的应用于互联网项目中。这不仅仅是因为Nginx本身比较轻量，更多的是得益于Nginx的高性能特性，以及支持插件化开发，为此，很多开发者或者公司基于Nginx开发出了众多的高性能插件。使用者可以根据自身的需求来为Nginx指定某款插件以增强Nginx在某种特定场景下的功能或者提升Nginx在某种特定场景下的性能。

Nginx获取客户端信息

注意：本文中的客户端信息指的是：客户端真实IP、域名、协议、端口。

Nginx反向代理后，Servlet应用通过`request.getRemoteAddr()`取到的IP是Nginx的IP地址，并非客户端真实IP，通过`request.getRequestURL()`获取的域名、协议、端口都是Nginx访问Web应用时的域名、协议、端口，而非客户端浏览器地址栏上的真实域名、协议、端口。

直接获取信息存在哪些问题?

例如在某一台IP为192.168.1.100的服务器上，Jetty或者Tomcat端口号为8080，Nginx端口号80，Nginx反向代理8080端口：

```
server {
    listen 80;
    location / {
        proxy_pass http://127.0.0.1:8080; # 反向代理应用服务器HTTP地址
    }
}
```

在另一台机器上用浏览器打开http://192.168.1.100/test访问某个Servlet应用，获取客户端IP和URL：

```
System.out.println("RemoteAddr: " + request.getRemoteAddr());
System.out.println("URL: " + request.getRequestURL().toString());
```

打印的结果信息如下：

```
RemoteAddr: 127.0.0.1
URL: http://127.0.0.1:8080/test
```

可以发现，Servlet程序获取到的客户端IP是Nginx的IP而非浏览器所在机器的IP，获取到的URL是Nginx proxy_pass配置的URL组成的地址，而非浏览器地址栏上的真实地址。如果将Nginx用作https服务器反向代理后端的http服务，那么`request.getRequestURL()`获取的URL是http前缀的而非https前缀，无法获取到浏览器地址栏的真实协议。如果此时将`request.getRequestURL()`获取得到的URL用作拼接Redirect地址，就会出现跳转到错误的地址，这也是Nginx反向代理时经常出现的一个问题。

如何解决这些问题?

既然直接使用Nginx获取客户端信息存在问题，那我们该如何解决这个问题呢?

我们整体上需要从两个方面来解决这些问题：

（1）由于Nginx是代理服务器，所有客户端请求都从Nginx转发到Jetty/Tomcat，如果Nginx不把客户端真实IP、域名、协议、端口告诉Jetty/Tomcat，那么Jetty/Tomcat应用永远不会知道这些信息，所以需要Nginx配置一些HTTP Header来将这些信息告诉被代理的Jetty/Tomcat；

（2）Jetty/Tomcat这一端，不能再获取直接和它连接的客户端（也就是Nginx）的信息，而是要从Nginx传递过来的HTTP Header中获取客户端信息。

具体实践

配置nginx

首先，我们需要在Nginx的配置文件nginx.conf中添加如下配置。

```
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

各参数的含义如下所示。

- `Host`包含客户端真实的域名和端口号；
- `X-Forwarded-Proto`表示客户端真实的协议（http还是https）；
- `X-Real-IP`表示客户端真实的IP；
- `X-Forwarded-For`这个Header和`X-Real-IP`类似，但它在多层代理时会包含真实客户端及中间每个代理服务器的IP。

此时，再试一下`request.getRemoteAddr()`和`request.getRequestURL()`的输出结果：

```
RemoteAddr: 127.0.0.1
URL: http://192.168.1.100/test
```

可以发现URL好像已经没问题了，但是IP还是本地的IP而非真实客户端IP。但是如果是用Nginx作为https服务器反向代理到http服务器，会发现浏览器地址栏是https前缀但是`request.getRequestURL()`获取到的URL还是http前缀，也就是仅仅配置Nginx还不能彻底解决问题。

通过Java方法获取客户端信息

仅仅配置Nginx不能彻底解决问题，那如何才能解决这个问题呢？一种解决方式就是通过Java方法获取客户端信息，例如下面的Java方法。

```
/***
 * 获取客户端IP地址;这里通过了Nginx获取;X-Real-IP
 */
public static String getClientIP(HttpServletRequest request) {
    String fromSource = "X-Real-IP";
    String ip = request.getHeader("X-Real-IP");
    if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
        ip = request.getHeader("X-Forwarded-For");
        fromSource = "X-Forwarded-For";
    }
    if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
        ip = request.getHeader("Proxy-Client-IP");
        fromSource = "Proxy-Client-IP";
    }
    if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
        ip = request.getHeader("WL-Proxy-Client-IP");
        fromSource = "WL-Proxy-Client-IP";
    }
    if (ip == null || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {
        ip = request.getRemoteAddr();
        fromSource = "request.getRemoteAddr";
    }
    return ip;
}
```

这种方式虽然能够获取客户端的IP地址，但是我总感觉这种方式不太友好，因为既然Servlet API提供了`request.getRemoteAddr()`方法获取客户端IP，那么无论有没有用反向代理对于代码编写者来说应该是透明的。

接下来，我就分别针对Jetty服务器和Tomcat服务器为大家介绍下如何进行配置才能更加友好的获取客户端信息。

Jetty服务器

在Jetty服务器的jetty.xml文件中，找到`httpConfig`，加入配置：

```
<New id="httpConfig" class="org.eclipse.jetty.server.HttpConfiguration">
    ...
  <Call name="addCustomizer">
    <Arg><New class="org.eclipse.jetty.server.ForwardedRequestCustomizer"/>
</Arg>
  </Call>
</New>
```

重新启动Jetty，再用浏览器打开http://192.168.1.100/test测试，结果：

```
RemoteAddr: 192.168.1.100
URL: http://192.168.1.100/test
```

此时可发现通过`request.getRemoteAddr()`获取到的IP不再是`127.0.0.1`而是客户端真实IP，`request.getRequestURL()`获取的URL也是浏览器上的真实URL，如果Nginx作为https代理，`request.getRequestURL()`的前缀也会是https。

另外，Jetty将这个功能封装成一个模块：http-forwarded。如果不想改jetty.xml配置文件的话，也可以启用http-forwarded模块来实现。

例如可以通过命令行启动Jetty：

```
java -jar start.jar --module=http-forwarded
```

更多Jetty如何启用模块的相关资料可以参考：
http://www.eclipse.org/jetty/documentation/current/startup.html

Tomcat服务器

和Jetty类似，如果使用Tomcat作为应用服务器，可以通过配置Tomcat的server.xml文件，在Host元素内最后加入：

```
<Valve className="org.apache.catalina.valves.RemoteIpValve" />
```

三、实现负载均衡、限流、缓存、黑白名单和灰度发布

在《【高并发】面试官问我如何使用Nginx实现限流，我如此回答轻松拿到了Offer！》一文中，我们主要介绍了如何使用Nginx进行限流，以避免系统被大流量压垮。除此之外，Nginx还有很多强大的功能，例如：负载均衡、缓存、黑白名单、灰度发布等。今天，我们就来一起探讨Nginx支持的这些强大的功能！

Nginx负载均衡配置

1.负载均衡配置

```
http {
        ……
    upstream real_server {
       server 192.168.103.100:2001 weight=1;  #轮询服务器和访问权重
       server 192.168.103.100:2002 weight=2;
    }

    server {
        listen  80;

        location / {
            proxy_pass http://real_server;
        }
    }
}
```

2.失败重试配置

```
upstream real_server {
   server 192.168.103.100:2001 weight=1 max_fails=2 fail_timeout=60s;
   server 192.168.103.100:2002 weight=2 max_fails=2 fail_timeout=60s;
}
```

意思是在fail_timeout时间内失败了max_fails次请求后，则认为该上游服务器不可用，然后将该服务地址踢除掉。fail_timeout时间后会再次将该服务器加入存活列表，进行重试。

Nginx限流配置

1.配置参数

limit_req_zone指令设置参数

```
limit_req_zone $binary_remote_addr zone=mylimit:10m rate=10r/s;
```

- limit_req_zone定义在http块中，$binary_remote_addr表示保存客户端IP地址的二进制形式。
- Zone定义IP状态及URL访问频率的共享内存区域。zone=keyword标识区域的名字，以及冒号后面跟区域大小。16000个IP地址的状态信息约1MB，所以示例中区域可以存储160000个IP地址。
- Rate定义最大请求速率。示例中速率不能超过每秒10个请求。

2.设置限流

```
location / {
        limit_req zone=mylimit burst=20 nodelay;
        proxy_pass http://real_server;
}
```

burst排队大小，nodelay不限制单个请求间的时间。

3.不限流白名单

```
geo $limit {
default              1;
```

```
192.168.2.0/24  0;
}

map $limit $limit_key {
1 $binary_remote_addr;
0 "";
}

limit_req_zone $limit_key zone=mylimit:10m rate=1r/s;

location / {
        limit_req zone=mylimit burst=1 nodelay;
        proxy_pass http://real_server;
}
```

上述配置中，192.168.2.0/24网段的IP访问是不限流的，其他限流。

IP后面的数字含义：

- 24表示子网掩码:255.255.255.0
- 16表示子网掩码:255.255.0.0
- 8表示子网掩码:255.0.0.0

Nginx缓存配置

1.浏览器缓存

静态资源缓存用expire

```
location ~*  .(jpg|jpeg|png|gif|ico|css|js)$ {
   expires 2d;
}
```

Response Header中添加了Expires和Cache-Control,

静态资源包括（一般缓存）

- 普通不变的图像，如logo，图标等
- js、css静态文件
- 可下载的内容，媒体文件

协商缓存（add_header ETag/Last-Modified value）

- HTML文件
- 经常替换的图片
- 经常修改的js、css文件
- 基本不变的API接口

不需要缓存

- 用户隐私等敏感数据
- 经常改变的api数据接口

2.代理层缓存

```
//缓存路径，inactive表示缓存的时间，到期之后将会把缓存清理
proxy_cache_path /data/cache/nginx/ levels=1:2 keys_zone=cache:512m inactive =
1d max_size=8g;
```

```
location / {
    location ~ \.(htm|html)?$ {
        proxy_cache cache;
        proxy_cache_key    $uri$is_args$args;      //以此变量值做HASH，作为KEY
        //HTTP响应首部可以看到X-Cache字段，内容可以有HIT,MISS,EXPIRES等等
        add_header X-Cache $upstream_cache_status;
        proxy_cache_valid 200 10m;
        proxy_cache_valid any 1m;
        proxy_pass  http://real_server;
        proxy_redirect     off;
    }
    location ~ .*\.(gif|jpg|jpeg|bmp|png|ico|txt|js|css)$ {
        root /data/webapps/edc;
        expires      3d;
        add_header Static Nginx-Proxy;
    }
}
```

在本地磁盘创建一个文件目录，根据设置，将请求的资源以K-V形式缓存在此目录当中，KEY需要自己定义（这里用的是url的hash值），同时可以根据需要指定某内容的缓存时长，比如状态码为200缓存10分钟，状态码为301，302的缓存5分钟，其他所有内容缓存1分钟等等。
可以通过purger的功能清理缓存。

AB测试/个性化需求时应禁用掉浏览器缓存。

Nginx黑名单

1.一般配置

```
location / {
    deny  192.168.1.1;
    deny 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny  all;
}
```

2. Lua+Redis动态黑名单(OpenResty)

安装运行

```
yum install yum-utils
yum-config-manager --add-repo
https://openresty.org/package/centos/openresty.repo
yum install openresty
yum install openresty-resty
查看
yum --disablerepo="*" --enablerepo="openresty" list available
运行
service openresty start
```

配置(/usr/local/openresty/nginx/conf/nginx.conf)

```
lua_shared_dict ip_blacklist 1m;

server {
    listen  80;

    location / {
        access_by_lua_file lua/ip_blacklist.lua;
        proxy_pass http://real_server;
    }
}
```

lua脚本（ip_blacklist.lua）

```
local redis_host    = "192.168.1.132"
local redis_port    = 6379
local redis_pwd     = 123456
local redis_db = 2

-- connection timeout for redis in ms.
local redis_connection_timeout = 100

-- a set key for blacklist entries
local redis_key     = "ip_blacklist"

-- cache lookups for this many seconds
local cache_ttl     = 60

-- end configuration

local ip                = ngx.var.remote_addr
local ip_blacklist      = ngx.shared.ip_blacklist
local last_update_time  = ip_blacklist:get("last_update_time");

-- update ip_blacklist from Redis every cache_ttl seconds:
if last_update_time == nil or last_update_time < ( ngx.now() - cache_ttl ) then

  local redis = require "resty.redis";
  local red = redis:new();

  red:set_timeout(redis_connect_timeout);

  local ok, err = red:connect(redis_host, redis_port);
  if not ok then
    ngx.log(ngx.ERR, "Redis connection error while connect: " .. err);
  else
    local ok, err = red:auth(redis_pwd)
    if not ok then
      ngx.log(ngx.ERR, "Redis password error while auth: " .. err);
    else
        local new_ip_blacklist, err = red:smembers(redis_key);
        if err then
            ngx.log(ngx.ERR, "Redis read error while retrieving ip_blacklist: "
.. err);
        else
        ngx.log(ngx.ERR, "Get data success:" .. new_ip_blacklist)
          -- replace the locally stored ip_blacklist with the updated values:
```

```
            ip_blacklist:flush_all();
          for index, banned_ip in ipairs(new_ip_blacklist) do
            ip_blacklist:set(banned_ip, true);
          end
          -- update time
          ip_blacklist:set("last_update_time", ngx.now());
      end
    end
  end
end

if ip_blacklist:get(ip) then
  ngx.log(ngx.ERR, "Banned IP detected and refused access: " .. ip);
  return ngx.exit(ngx.HTTP_FORBIDDEN);
end
```

Nginx灰度发布

1.根据Cookie实现灰度发布

根据Cookie查询version值，如果该version值为v1转发到host1，为v2转发到host2，都不匹配的情况下转发到默认配置。

```
upstream host1 {
   server 192.168.2.46:2001 weight=1;  #轮询服务器和访问权重
   server 192.168.2.46:2002 weight=2;
}

upstream host2 {
   server 192.168.1.155:1111  max_fails=1 fail_timeout=60;
}

upstream default {
   server 192.168.1.153:1111  max_fails=1 fail_timeout=60;
}

map $COOKIE_version $group {
   ~*v1$ host1;
   ~*v2$ host2;
   default default;
}

lua_shared_dict ip_blacklist 1m;

server {
    listen  80;

    #set $group "default";
    #if ($http_cookie ~* "version=v1"){
    #    set $group host1;
    #}
    #if ($http_cookie ~* "version=v2"){
    #    set $group host2;
    #}

    location / {
        access_by_lua_file lua/ip_blacklist.lua;
```

```
        proxy_pass http://$group;
    }
}
```

2.根据来路IP实现灰度发布

```
server {
  ..............
  set $group default;
  if ($remote_addr ~ "192.168.119.1") {
      set $group host1;
  }
  if ($remote_addr ~ "192.168.119.2") {
      set $group host2;
  }
```

3.更细粒度灰度发布

参考：https://github.com/sunshinelyz/ABTestingGateway

四、面试官竟然问我Nginx如何生成缩略图

今天想写一篇使用Nginx如何生成缩略图的文章，想了半天题目也没想好，这个题目还是一名读者帮我起的。起因就是这位读者最近出去面试，面试官正好问了一个Nginx如何生成缩略图的问题。还别说，就是这么巧呀！！就冲这标题，也要写一篇干货满满的技术好文！！

关于Nginx的安装，小伙伴们可以参考《【Nginx】实现负载均衡、限流、缓存、黑白名单和灰度发布，这是最全的一篇了！》

生成缩略图方案

为了手机端浏览到与手机分辨率相匹配的图片，提高 APP 访问速度以及减少用户的手机流量，需要将图片生成缩略图，这边共有以下解决方案。

- A.发布新闻生成多重缩略图 – 无法匹配到各种尺寸图片
- B.当相应缩略图不存在，则使用 PHP 或者 Java 等程序生成相应缩略图 – 需要程序员协助
- C.使用 Nginx 自带模块生成缩略图 – 运维即可完成
- D.使用 Nginx + Lua 生成缩略图

经过多方的考虑，决定使用方案 C，使用 Nginx 自带模块生成缩略图。

Nginx生成缩略图

配置Nginx

使用 Nginx 自带模块生成缩略图，模块： --with-http_image_filter_module，例如，我们可以使用如下参数安装Nginx：

```
./configure --prefix=/usr/local/nginx-1.19.1 --with-http_stub_status_module --with-http_realip_module --with-http_image_filter_module --with-debug
```

接下来，修改 nginx.conf 配置文件，或者将下面的配置放到nginx.conf文件相应的 server 块中。

```
location ~* /(\d+)\.(jpg)$ {
    set $h $arg_h; # 获取参数 h 的值
    set $w $arg_w; # 获取参数 w 的值
    #image_filter crop $h $w;
    image_filter resize $h $w;# 根据给定的长宽生成缩略图
}
location ~* /(\d+)_(\d+)x(\d+)\.(jpg)$ {
    if ( -e $document_root/$1.$4 ) { # 判断原图是否存在
        rewrite /(\d+)_(\d+)x(\d+)\.(jpg)$ /$1.$4?h=$2&w=$3 last;
    }
    return 404;
}
```

访问图片

配置完成后，我们就可以使用类似如下的方式来访问图片。

http://www.binghe.com/123_100x10.jpg

当我们在浏览器地址栏中输入上面的链接时，Nginx会作出如下的逻辑处理。

- 首先判断是否存在原图 123.jpg,不存在直接返回 404（如果原图都不存在，那就没必要生成缩略图了）
- 跳转到 http://www.binghe.com/123.jpg?h=100&w=10，将参数高 h=100 和宽 w=10 带到 url 中。
- Image_filter resize 指令根据 h 和 w 参数生成相应缩略图。

注意：使用Nginx生成等比例缩略图时有一个长宽取小的原则， `例如原图是 100*10,你传入的是 10*2，那么Nginx会给你生成 10*1 的图片` **。生成缩略图只是 image_filter 功能中的一个，它一共支持 4 种参数：**

- test：返回是否真的是图片
- size：返回图片长短尺寸，返回 json 格式数据
- corp：截取图片的一部分，从左上角开始截取，尺寸写小了，图片会被剪切
- resize：缩放图片，等比例缩放

Nginx 生成缩略图优缺点

优点：

- 根据传入参数即可生成各种比例图片
- 不占用任何硬盘空间

缺点：

- 消耗 CPU
- 访问量大将会给服务器带来比较大的负担

建议：

生成缩略是个消耗 CPU 的操作，如果访问量比较大的站点，最好考虑使用程序生成缩略图到硬盘上，或者在前端加上 Cache缓存或者使用 CDN。

五、如何封禁IP和IP段?

Nginx不仅仅只是一款反向代理和负载均衡服务器，它还能提供很多强大的功能，例如：限流、缓存、黑白名单和灰度发布等等。在之前的文章中，我们已经介绍了Nginx提供的这些功能。小伙伴们可以到【Nginx专题】进行查阅。今天，我们来介绍Nginx另一个强大的功能：禁用IP和IP段。

禁用IP和IP段

Nginx的ngx_http_access_module 模块可以封配置内的ip或者ip段，语法如下：

```
deny IP;
deny subnet;
allow IP;
allow subnet;
# block all ips
deny    all;
# allow all ips
allow    all;
```

如果规则之间有冲突，会以最前面匹配的规则为准。

配置禁用ip和ip段

下面说明假定nginx的目录在/usr/local/nginx/。

首先要建一个封ip的配置文件blockips.conf，然后vi blockips.conf编辑此文件，在文件中输入要封的ip。

```
deny 1.2.3.4;
deny 91.212.45.0/24;
deny 91.212.65.0/24;
```

然后保存此文件，并且打开nginx.conf文件，在http配置节内添加下面一行配置：

```
include blockips.conf;
```

保存nginx.conf文件，然后测试现在的nginx配置文件是否是合法的：

```
/usr/local/nginx/sbin/nginx -t
```

如果配置没有问题，就会输出：

```
the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
configuration file /usr/local/nginx/conf/nginx.conf test is successful
```

如果配置有问题就需要检查下哪儿有语法问题，如果没有问题，需要执行下面命令，让nginx重新载入配置文件。

```
/usr/local/nginx/sbin/nginx -s reload
```

仅允许内网ip

如何禁止所有外网ip，仅允许内网ip呢?

如下配置文件

```
location / {
  # block one workstation
  deny    192.168.1.1;
  # allow anyone in 192.168.1.0/24
  allow   192.168.1.0/24;
  # drop rest of the world
  deny    all;
}
```

上面配置中禁止了192.168.1.1，允许其他内网网段，然后deny all禁止其他所有ip。

格式化nginx的403页面

如何格式化nginx的403页面呢?

首先执行下面的命令：

```
cd /usr/local/nginx/html
vi error403.html
```

然后输入403的文件内容，例如：

```
<html>
<head><title>Error 403 - IP Address Blocked</title></head>
<body>
Your IP Address is blocked. If you this an error, please contact binghe with
your IP at test@binghe.com
</body>
</html>
```

如果启用了SSI，可以在403中显示被封的客户端ip，如下：

```
Your IP Address is <!--#echo var="REMOTE_ADDR" --> blocked.
```

保存error403文件，然后打开nginx的配置文件vi nginx.conf,在server配置节内添加下面内容。

```
# redirect server error pages to the static page
 error_page   403  /error403.html;
 location = /error403.html {
         root   html;
 }
```

然后保存配置文件，通过nginx -t命令测试配置文件是否正确，若正确通过nginx -s reload载入配置。

六、如何按日期分割Nginx日志?

Nginx是没有以日期格式作为文件名来存储的，也就是说，Nginx不像Tomcat，每天自动生成一个日志文件，所有的日志都是以一个名字来存储，时间久了日志文件会变得很大。这样非常不利于分析。虽然nginx没有这个功能但我们可以写一个小脚本配合计划任务来达到这样的效果。即让Nginx每天产生一个日志文件，方便我们进行后续的数据分析。

分割Nginx日志

首先，我们要创建一个脚本文件，用来分割Nginx日志，具体脚本如下：

```
vim /usr/local/nginx-1.19.1/cutnginxlog.sh
```

脚本内容如下：

```
#!/bin/sh
# Program:
#     Auto cut nginx log script.

# nginx日志路径
LOGS_PATH=/usr/local/nginx-1.19.1/logs
TODAY=$(date -d 'today' +%Y-%m-%d)

# 移动日志并改名
mv ${LOGS_PATH}/error.log ${LOGS_PATH}/error_${TODAY}.log
mv ${LOGS_PATH}/access.log ${LOGS_PATH}/access_${TODAY}.log

# 向nginx主进程发送重新打开日志文件的信号
kill -USR1 $(cat /usr/local/nginx-1.19.1/logs/nginx.pid)
```

接下来就是给cutnginxlog.sh文件授权。

```
chmod a+x cutnginxlog.sh
```

接下来添加计划任务，定时执行cutnginxlog.sh脚本，以root用户执行如下命令：

```
echo '59 23 * * * root /usr/local/nginx-1.19.1/cutnginxlog.sh >>
/usr/local/nginx-1.19.1/cutnginxlog.log 2>&1' >> /etc/crontab
```

意思就是在每天的23点59分执行脚本。将自动任务的执行日志（错误和正确的日志）自动写入cutnginxlog.log，“命令 >> 2>&1” 表示以追加方式将正确输出和错误输出都保存到同一个文件中。

七、如何配置Nginx日志?

日志对于统计排错来说非常有利的。本文总结了 Nginx 日志相关的配置如 access_log、 log_format、open_log_file_cache、 log_not_found、 log_subrequest、 rewrite_log、 error_log。

配置Nginx日志

Nginx 有一个非常灵活的日志记录模式。每个级别的配置可以有各自独立的访问日志。日志格式通过log_format命令来定义。 ngx_http_log_module 是用来定义请求日志格式的。

access_log 指令

语法:

```
access_log path [format [buffer=size [flush=time]]];
access_log path format gzip[=level] [buffer=size] [flush=time];
access_log syslog:server=address[,parameter=value] [format];
access_log off;
```

默认值:

```
access_log logs/access.log combined;
```

配置段:

```
http, server, location, if in location, limit_except
```

- gzip 压缩等级。
- buffer 设置内存缓存区大小。
- flush 保存在缓存区中的最长时间

不记录日志:

```
access_log off;
```

使用默认 combined 格式记录日志:

```
access_log logs/access.log
```

或者

```
access_log logs/access.log combined;
```

log_format 指令

语法:

```
log_format name string …;
```

默认值:

```
log_format combined “…” ;
```

配置段:

```
http
```

- name 表示格式名称
- string 表示等义的格式。

log_format 有一个默认的无需设置的 combined 日志格式，相当于apache 的 combined 日志格式，如下所示:

```
log_format combined '$remote_addr - $remote_user [$time_local] '
' "$request" $status $body_bytes_sent '
' "$http_referer" "$http_user_agent" ';
```

如果 nginx 位于负载均衡器， squid， nginx 反向代理之后， web 服务器无法直接获取到客户端真实的 IP 地址了。
$remote_addr 获取反向代理的 IP 地址。反向代理服务器在转发请求的 http 头信息中，可以增加 X-ForwardedFor 信息，用来记录 客户端 IP 地址和客户端请求的服务器地址。 如下所示:

```
log_format porxy '$http_x_forwarded_for - $remote_user [$time_local] '
' "$request" $status $body_bytes_sent '
' "$http_referer" "$http_user_agent" ';
```

日志格式允许包含的变量注释如下：

- $remote_addr, $http_x_forwarded_for 记录客户端 IP 地址
- $remote_user 记录客户端用户名称
- $request 记录请求的 URL 和 HTTP 协议
- $status 记录请求状态
- $body_bytes_sent 发送给客户端的字节数，不包括响应头的大小； 该变量与 Apache 模块 mod_log_config 里的“%B”参数兼容。
- $bytes_sent 发送给客户端的总字节数。
- $connection 连接的序列号。
- $connection_requests 当前通过一个连接获得的请求数量。
- $msec 日志写入时间。单位为秒，精度是毫秒。
- $pipe 如果请求是通过 HTTP 流水线(pipelined)发送， pipe 值为“p”，否则为“.”。
- $http_referer 记录从哪个页面链接访问过来的
- $http_user_agent 记录客户端浏览器相关信息
- $request_length 请求的长度（包括请求行，请求头和请求正文）。
- $request_time 请求处理时间，单位为秒，精度毫秒； 从读入客户端的第一个字节开始，直到把最后一个字符发送给客户端后进行日志写入为止。
- $time_iso8601 ISO8601 标准格式下的本地时间。
- $time_local 通用日志格式下的本地时间。

注意：发送给客户端的响应头拥有“sent_http_”前缀。 比如$sent_http_content_range。

实例如下：

```
http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
    '"$status" $body_bytes_sent "$http_referer" '
    '"$http_user_agent" "$http_x_forwarded_for" '
    '"$gzip_ratio" $request_time $bytes_sent $request_length';
    log_format srcache_log '$remote_addr - $remote_user [$time_local] "$request"
'
    '"$status" $body_bytes_sent $request_time $bytes_sent
    $request_length '
    '[$upstream_response_time] [$srcache_fetch_status]
    [$srcache_store_status] [$srcache_expire]';
    open_log_file_cache max=1000 inactive=60s;
    server {
        server_name ~^(www\.)?(.+)$;
        access_log logs/$2-access.log main;
        error_log logs/$2-error.log;
        location /srcache {
            access_log logs/access-srcache.log srcache_log;
        }
    }
}
```

open_log_file_cache 指令

语法:

```
open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time];
open_log_file_cache off;
```

默认值:

```
open_log_file_cache off;
```

配置段:

```
http, server, location
```

对于每一条日志记录，都将是先打开文件，再写入日志，然后关闭。可以使用 open_log_file_cache 来设置日志文件缓存(默认是 off)。

参数注释如下:

- max:设置缓存中的最大文件描述符数量，如果缓存被占满，采用 LRU 算法将描述符关闭。
- inactive:设置存活时间，默认是 10s
- min_uses:设置在 inactive 时间段内，日志文件最少使用多少次后，该日志文件描述符记入缓存中，默认是 1 次
- valid:设置检查频率，默认 60s
- off：禁用缓存

实例如下:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```

log_not_found 指令

语法:

```
log_not_found on | off;
```

默认值:

```
log_not_found on;
```

配置段:

```
http, server, location
```

是否在 error_log 中记录不存在的错误。默认是。

log_subrequest 指令

语法:

```
log_subrequest on | off;
```

默认值:

```
log_subrequest off;
```

配置段:

```
http, server, location
```

是否在 access_log 中记录子请求的访问日志。默认不记录。

rewrite_log 指令

由 ngx_http_rewrite_module 模块提供的。用来记录重写日志的。对于调试重写规则建议开启。 Nginx 重写规则指南

语法:

```
rewrite_log on | off;
```

默认值:

```
rewrite_log off;
```

配置段:

```
http, server, location, if
```

启用时将在 error log 中记录 notice 级别的重写日志。

error_log 指令

语法:

```
error_log file | stderr | syslog:server=address[,parameter=value] [debug | info
| notice |
warn | error | crit | alert | emerg];
```

默认值:

```
error_log logs/error.log error;
```

配置段:

```
main, http, server, location
```

配置错误日志。

八、如何为已安装的Nginx动态添加模块?

很多时候，我们根据当时的项目情况和业务需求安装完Nginx后，后续随着业务的发展，往往会给安装好的Nginx添加其他的功能模块。在为Nginx添加功能模块时，要求Nginx不停机。这就涉及到如何为已安装的Nginx动态添加模块的问题。本文，就和小伙伴们一起探讨如何为已安装的Nginx动态添加模块的问题。

为Nginx动态添加模块

这里以安装第三方ngx_http_google_filter_module模块为例。

Nginx的模块是需要重新编译Nginx，而不是像Apache一样配置文件引用.so

下载第三方扩展模块ngx_http_google_filter_module

```
# cd /data/software/
# git clone https://github.com/cuber/ngx_http_google_filter_module
```

查看nginx编译安装时安装了哪些模块

将命令行切换到Nginx执行程序所在的目录并输入./nginx -V，具体如下：

```
[root@binghe sbin]# ./nginx -V
nginx version: nginx/1.19.1
built by gcc 4.4.7 20120313 (Red Hat 4.4.7-17) (GCC)
built with OpenSSL 1.0.2 22 Jan 2015
TLS SNI support enabled
configure arguments: --prefix=/usr/local/nginx-1.19.1 --with-
openssl=/usr/local/src/openssl-1.0.2 --with-pcre=/usr/local/src/pcre-8.37 --
with-zlib=/usr/local/src/zlib-1.2.8 --with-http_ssl_module
[root@binghe sbin]#
```

可以看出编译安装Nginx使用的参数如下：

```
--prefix=/usr/local/nginx-1.19.1 --with-openssl=/usr/local/src/openssl-1.0.2 --
with-pcre=/usr/local/src/pcre-8.37 --with-zlib=/usr/local/src/zlib-1.2.8 --with-
http_ssl_module
```

加入需要安装的模块，重新编译

这里添加 --add-module=/data/software/ngx_http_google_filter_module

具体如下：

```
./configure  --prefix=/usr/local/nginx-1.19.1 --with-
openssl=/usr/local/src/openssl-1.0.2 --with-pcre=/usr/local/src/pcre-8.37 --
with-zlib=/usr/local/src/zlib-1.2.8 --with-http_ssl_module --add-
module=/data/software/ngx_http_google_filter_module
```

如上，将之前安装Nginx的参数全部加上，最后添加 --add-
module=/data/software/ngx_http_google_filter_module

之后，我们要进行编译操作，如下：

```
# make    //千万不要make install，不然就真的覆盖
```

这里，需要注意的是：不要执行make install命令。

替换nginx二进制文件

```
# 备份原来的nginx执行程序
# mv /usr/local/nginx-1.19.1/sbin/nginx /usr/local/nginx-1.19.1/sbin/nginx.bak
# 将新编译的nginx执行程序复制到/usr/local/nginx-1.19.1/sbin/目录下
# cp /opt/nginx/sbin/nginx /usr/local/nginx-1.19.1/sbin/
```

九、如何格式化日志并推送到远程服务器?

Nginx作为最常用的反向代理和负载均衡服务器，被广泛的应用在众多互联网项目的前置服务中，很多互联网项目直接将Nginx服务器作为整个项目的流量入口。这就使得我们可以通过对Nginx服务器日志的分析，就可以分析出整个网站的访问总量、PV、UV、VV等信息。实际上，企业的业务线众多，很难使用一台Nginx服务器来代理所有的线上服务，这就导致企业会在线上部署多台Nginx服务器。而我们如果想分析所有Nginx服务器的总流量信息时，如果分别对每个Nginx服务器进行分析，再汇总所有的信息，一方面增加了分析的复杂度，另一方面也不好维护这些日志信息。所以，大部分企业会将这些日志信息统一汇总到某个数据存储集群中，以方便的进行数据存储、维护与分析统计。那么如何对Nginx的日志进行格式化并推送到远程的服务器呢？今天，我们就一起来探讨下这个问题。

配置Nginx

格式化Nginx日志并推送到远程服务器，其实很简单，我们只需要在Nginx服务器的配置文件nginx.conf中进行简单的配置即可。例如，我们可以在nginx.conf文件中添加如下配置。

```
log_format common
"$remote_addr,$http_ip,$http_mac,$time_local,$status,$request_length,$bytes_sent
,$body_bytes_sent,$http_user_agent,$http_referer,$request_method,$request_time,$
request_uri,$server_protocol,$request_body,$http_token";

log_format main
"$remote_addr,$http_ip,$http_mac,$time_local,$status,$request_length,$bytes_sent
,$body_bytes_sent,$http_user_agent,$http_referer,$request_method,$request_time,$
request_uri,$server_protocol,$request_body,$http_token";

access_log  logs/access.log  common;

access_log
syslog:server=192.168.1.100:9999,facility=local7,tag=nginx,severity=info main;
  map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
 }
```

上述配置是将Nginx的日志各项参数以逗号分隔的形式进行输出，同时将Nginx日志实时推送到192.168.1.100:9999上。

此时，我们只需要在192.168.1.100服务器上部署一个TCP或UDP服务，监听端口为9999，并在192.168.1.100服务器的防火墙开放9999端口。我们写的TCP或UDP服务就会实时接收到Nginx服务器发送过来的日志。

通过这种方式，我们就可以将Nginx日志实时收集到某个存储集群中，对Nginx日志进行统一存储、维护和分析。

十、面试官问我Nginx能不能配置WebSocket?

当今互联网领域，不管是APP还是H5，不管是微信端还是小程序，只要是一款像样点的产品，为了增加用户的交互感和用户粘度，多多少少都会涉及到聊天功能。而对于Web端与H5来说，实现聊天最简单的就是使用WebSocket了。而在实现WebSocket聊天的过程中，后台也往往会部署多个WebSocket服务，多个WebSocket服务之间，可以通过Nginx进行负载均衡。今天，我们就来一起说说Nginx是如何配置WebSocket的。

Nginx配置WebSocket

Nginx配置WebSocket也比较简单，只需要在nginx.conf文件中进行相应的配置。这种方式很简单，但是很有效，能够横向扩展WebSocket服务端的服务能力。

先直接展示配置文件，如下所示(使用的话直接复制，然后改改ip和port即可)

```
map $http_upgrade $connection_upgrade {
    default upgrade;
    '' close;
}
upstream wsbackend{
    server ip1:port1;
    server ip2:port2;
    keepalive 1000;
}

server {
    listen 20038;
    location /{
        proxy_http_version 1.1;
        proxy_pass http://wsbackend;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_read_timeout 3600s;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }
}
```

接下来，我们就分别分析上述配置的具体含义。

首先:

```
map $http_upgrade $connection_upgrade {
    default upgrade;
    '' close;
}
```

表示的是:

- 如果 $http_upgrade 不为 '' (空)，则 $connection_upgrade 为 upgrade 。
- 如果 $http_upgrade 为 '' (空)，则 $connection_upgrade 为 close。

其次:

```
upstream wsbackend{
    server ip1:port1;
    server ip2:port2;
    keepalive 1000;
}
```

表示的是 nginx负载均衡：

- 两台服务器 (ip1:port1)和(ip2:port2) 。
- keepalive 1000 表示的是每个nginx进程中上游服务器保持的空闲连接，当空闲连接过多时，会关闭最少使用的空闲连接.当然，这不是限制连接总数的，可以想象成空闲连接池的大小，设置的值应该是上游服务器能够承受的。

最后：

```
server {
    listen 20038;
    location /{
        proxy_http_version 1.1;
        proxy_pass http://wsbackend;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_read_timeout 3600s;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }
}
```

表示的是监听的服务器的配置

- listen 20038 表示 nginx 监听的端口
- locations / 表示监听的路径(/表示所有路径，通用匹配，相当于default)
- proxt_http_version 1.1 表示反向代理发送的HTTP协议的版本是1.1，HTTP1.1支持长连接
- proxy_pass http://wsbackend; 表示反向代理的uri，这里可以使用负载均衡变量
- proxy_redirect off; 表示不要替换路径，其实这里如果是/则有没有都没关系，因为default也是将路径替换到proxy_pass的后边
- proxy_set_header Host $host; 表示传递时请求头不变， $host是nginx内置变量，表示的是当前的请求头，proxy_set_header表示设置请求头
- proxy_set_header X-Real-IP $remote_addr; 表示传递时来源的ip还是现在的客户端的ip
- proxy_read_timeout 3600s； 表的两次请求之间的间隔超过 3600s 后才关闭这个连接，默认的60s，自动关闭的元凶
- proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; 表示X-Forwarded-For头不发生改变
- proxy_set_header Upgrade $http_upgrade; 表示设置Upgrade不变
- proxy_set_header Connection $connection_upgrade; 表示如果 $http_upgrade为upgrade，则请求为upgrade(websocket)，如果不是，就关闭连接

十一、如何使用Nginx实现MySQL数据库的负载均衡?

Nginx能够实现HTTP、HTTPS协议的负载均衡，也能够实现TCP协议的负载均衡。那么，问题来了，可不可以通过Nginx实现MySQL数据库的负载均衡呢？答案是：可以。接下来，就让我们一起探讨下如何使用Nginx实现MySQL的负载均衡。

前提条件

注意：使用Nginx实现MySQL数据库的负载均衡，前提是要搭建MySQL的主主复制环境，关于MySQL主主复制环境的搭建，后续会在MySQL专题为大家详细阐述。这里，我们假设已经搭建好MySQL的主主复制环境，MySQL服务器的IP和端口分别如下所示。

- 192.168.1.101　3306
- 192.168.1.102　3306

通过Nginx访问MySQL的IP和端口如下所示。

- 192.168.1.100　3306

Nginx实现MySQL负载均衡

nginx在版本1.9.0以后支持tcp的负载均衡，具体可以参照官网关于模块ngx_stream_core_module的叙述，链接地址为：http://nginx.org/en/docs/stream/ngx_stream_core_module.html#tcp_nodelay。

nginx从1.9.0后引入模块ngx_stream_core_module，模块是没有编译的，需要用到编译，编译时需添加--with-stream配置参数，stream负载均衡官方配置样例如下所示。

```
worker_processes auto;
error_log /var/log/nginx/error.log info;

events {
    worker_connections  1024;
}

stream {
    upstream backend {
        hash $remote_addr consistent;

        server backend1.example.com:12345 weight=5;
        server 127.0.0.1:12345            max_fails=3 fail_timeout=30s;
        server unix:/tmp/backend3;
    }

    upstream dns {
       server 192.168.0.1:53535;
       server dns.example.com:53;
    }

    server {
        listen 12345;
        proxy_connect_timeout 1s;
        proxy_timeout 3s;
        proxy_pass backend;
    }

    server {
        listen 127.0.0.1:53 udp;
        proxy_responses 1;
        proxy_timeout 20s;
        proxy_pass dns;
    }

    server {
        listen [::1]:12345;
```

```
        proxy_pass unix:/tmp/stream.socket;
    }
}
```

说到这里，使用Nginx实现MySQL的负载均衡就比较简单了。我们可以参照上面官方的配置示例来配置MySQL的负载均衡。这里，我们可以将Nginx配置成如下所示。

```
user  nginx;
#user root;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
events {
    worker_connections  1024;
}
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  /var/log/nginx/access.log  main;
    sendfile        on;
    #tcp_nopush     on;
    keepalive_timeout  65;
    #gzip  on;
    include /etc/nginx/conf.d/*.conf;
}

stream{
    upstream mysql{
        server 192.168.1.101:3306 weight=1;
        server 192.168.1.102:3306 weight=1;
    }

    server{
        listen 3306;
        server_name 192.168.1.100;
        proxy_pass mysql;
    }
}
```

配置完成后，我们就可以通过如下方式来访问MySQL数据库。

```
jdbc:mysql://192.168.1.100:3306/数据库名称
```

此时，Nginx会将访问MySQL的请求路由到IP地址为192.168.1.101和192.168.1.102的MySQL上。

十二、使用Nginx如何解决跨域问题?

当今互联网行业，大部分Web项目基本都是采用的前后端分离模式。前端为H5项目，后端为Java、PHP、Python等项目。而且大部分后端服务并不会只部署一套服务，而是会采用Nginx对后端服务进行负载均衡。那么，此时就会出现一个问题了：如果一个请求url的 **协议、域名、端口** 三者之间任意一个与当前页面url不同就会产生跨域的现象。那么如何使用Nginx解决跨域问题呢？接下来，我们就一起探讨下这个问题。

为何会跨域?

出于浏览器的同源策略限制。同源策略（Sameoriginpolicy）是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。同源策略会阻止一个域的javascript脚本和另外一个域的内容进行交互。所谓同源（即指在同一个域）就是两个页面具有相同的协议（protocol），主机（host）和端口号（port）。

Nginx如何解决跨域?

这里，我们利用Nginx的反向代理功能解决跨域问题，至于，什么是Nginx的反向代理，大家就请自行百度或者谷歌吧。

Nginx作为反向代理服务器，就是把http请求转发到另一个或者一些服务器上。通过把本地一个url前缀映射到要跨域访问的web服务器上，就可以实现跨域访问。对于浏览器来说，访问的就是同源服务器上的一个url。而Nginx通过检测url前缀，把http请求转发到后面真实的物理服务器。并通过rewrite命令把前缀再去掉。这样真实的服务器就可以正确处理请求，并且并不知道这个请求是来自代理服务器的。

Nginx解决跨域案例

使用Nginx解决跨域问题时，我们可以编译Nginx的nginx.conf配置文件，例如，将nginx.conf文件的server节点的内容编辑成如下所示。

```
server {
        location / {
            root   html;
            index  index.html index.htm;
            //允许cros跨域访问
            add_header 'Access-Control-Allow-Origin' '*';

        }
        //自定义本地路径
        location /apis {
            rewrite  ^.+apis/?(.*)$ /$1 break;
            include  uwsgi_params;
            proxy_pass   http://www.binghe.com;
       }
}
```

然后我把项目部署在nginx的html根目录下，在ajax调用时设置url从http://www.binghe.com/apistest/test 变为 http://www.binghe.com/apis/apistest/test然后成功解决。

假设，之前我在页面上发起的Ajax请求如下所示。

```
$.ajax({
        type:"post",
        dataType: "json",
        data:{'parameter':JSON.stringify(data)},
        url:"http://www.binghe.com/apistest/test",
        async: flag,
        beforeSend: function (xhr) {

            xhr.setRequestHeader("Content-Type", submitType.Content_Type);
            xhr.setRequestHeader("user-id", submitType.user_id);
            xhr.setRequestHeader("role-type", submitType.role_type);
            xhr.setRequestHeader("access-token", getAccessToken().token);
```

```
        },
        success:function(result, status, xhr){

        }
        ,error:function (e) {
            layerMsg('请求失败，请稍后再试')
        }
    });
```

修改成如下的请求即可解决跨域问题。

```
$.ajax({
        type:"post",
        dataType: "json",
        data:{'parameter':JSON.stringify(data)},
        url:"http:www.binghe.com/apis/apistest/test",
        async: flag,
        beforeSend: function (xhr) {

            xhr.setRequestHeader("Content-Type", submitType.Content_Type);
            xhr.setRequestHeader("user-id", submitType.user_id);
            xhr.setRequestHeader("role-type", submitType.role_type);
            xhr.setRequestHeader("access-token", getAccessToken().token);
        },
        success:function(result, status, xhr){

        }
        ,error:function (e) {
            layerMsg('请求失败，请稍后再试')
        }
    });
```

十三、图片显示过慢，文件下载不完全，竟然是Nginx的锅！！

最近，一名读者跟我说他通过浏览器访问自己的服务器时，图片显示的非常慢，以至于在浏览器中都无法完全加载出来，下载文件时，更是恼火，文件根本就无法完全下载下来。而且奇怪的是这位读者所在的网络是没啥问题的。于是，我便开始帮他排查各种问题。。。

问题定位

经过一系列的排查（中间过程我就省略了，直接写重点了！），最终定位到是Nginx的问题。当我打开这位读者的网站后台管理系统，发现图片显示非常慢，在Nginx前端代理上查出如下错误信息。

```
[error] 28423#0: *5 connect() failed (111: Connection refused) while connecting
to upstream
```

直接在后台服务器上用后台服务器的IP地址去访问，发现速度相当快，于是怀疑是Nginx的配置问题。

注意：当下载大的附件，或是页面中有大图片时，就会下载中断或是图片无法显示，也许你会说我用的Nginx缺省的配置也从来没有碰到过这种问题呀！我想说的是：那是因为你的网站没有大文件，至少没有大到使用Nginx的默认配置加载不出来。

这里，我给出一段Nginx的配置，如下所示。

```
location /file {
     root /home/file;
     index  index.html index.htm;
     proxy_set_header X-Real-IP $remote_addr;
     proxy_set_header   Host $host;
     proxy_pass http://127.0.0.1:8080 ;
     client_max_body_size     100m;
     client_body_buffer_size  128k;
     proxy_connect_timeout    600;
     proxy_read_timeout       600;
     proxy_send_timeout       600;
     proxy_buffer_size        32k;
     proxy_buffers           4 64k;
     proxy_busy_buffers_size 64k;
     proxy_temp_file_write_size 64k;
}
```

其中几个重要的参数如下所示。

- proxy_connect_timeout 600; #nginx跟后端服务器连接超时时间(代理连接超时)
- proxy_read_timeout 600; #连接成功后，后端服务器响应时间(代理接收超时)
- proxy_send_timeout 600; #后端服务器数据回传时间(代理发送超时)
- proxy_buffer_size 32k; #设置代理服务器（nginx）保存用户头信息的缓冲区大小
- proxy_buffers 4 32k; #proxy_buffers缓冲区，网页平均在32k以下的话，这样设置
- proxy_busy_buffers_size 64k; #高负荷下缓冲大小（proxy_buffers*2）
- proxy_temp_file_write_size 16k; #设定缓存文件夹大小，大于这个值，将从upstream服务器传

看到这里，发现问题了，这位读者的Nginx有下面一行配置。

```
proxy_temp_file_write_size 16k;
```

而他服务器上的图片基本都在100K~5M之间。

问题就出在proxy_temp_file_write_size上，当服务器上的文件超过该参数设置的大小时，Nginx会先将文件写入临时目录(缺省为Nginx安装目下/proxy_temp目录)，缺省Nginx是以nobody身份启动的，用ls -al 命令查看proxy_temp目录 nobody是proxy_temp目录的所有者，怪了那为什么没权限呢？接下来查看proxy_temp的父目录既Nginx安装目录。发现nobody竟然没权限，怪不得会出现上面的问题。

解决问题

定位到问题，接下来解决问题就比较简单了。可以使用两种方式解决这个问题，如下所示。

- 设置任何人都可以写 proxy_temp目录，重启 Nginx 即可解决。
- 直接更改proxy_temp_file_write_size的值，将其修改为大于图片和文件的大小，重启Nginx。

如果是以第一种方式解决问题的话，比如我的proxy_temp目录是/usr/local/nginx/proxy_temp，用如下命令将/usr/local/nginx/proxy_temp目录设置为任何人都可以写，问题解决。

```
chmod -R 777 /usr/local/nginx/proxy_temp/
```

如果是使用第二种方式解决问题的话，就可以直接修改nginx.conf文件，如下所示。

```
location /file {
     root /home/file;
     index  index.html index.htm;
     proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_set_header   Host $host;
        proxy_pass http://127.0.0.1:8080 ;
        client_max_body_size     100m;
        client_body_buffer_size  256k;
        proxy_connect_timeout    1200;
        proxy_read_timeout       1200;
        proxy_send_timeout       6000;
        proxy_buffer_size        32k;
        proxy_buffers            4 64k;
        proxy_busy_buffers_size  128k;
        proxy_temp_file_write_size 10m;
    }
```

当然，我也帮这位读者优化了一些其他的配置项。

十四、如何使用Nginx搭建流媒体服务器实现直播?

最近几年，直播行业比较火，无论是传统行业的直播，还是购物、游戏、教育，都在涉及直播。作为在互联网行业奋斗了多年的小伙伴，你有没有想过如果使用Nginx搭建一套直播环境，那我们该如何搭建呢？别急，接下来，我们就一起使用Nginx来搭建一套直播环境。

安装Nginx

注意：这里以CentOS 6.8服务器为例，以root用户身份来安装Nginx。

1.安装依赖环境

```
yum -y install wget gcc-c++ ncurses ncurses-devel cmake make perl bison openssl
openssl-devel gcc* libxml2 libxml2-devel curl-devel libjpeg* libpng* freetype*
autoconf automake zlib* fiex* libxml* libmcrypt* libtool-ltdl-devel* libaio
libaio-devel  bzr libtool
```

2.安装openssl

```
wget https://www.openssl.org/source/openssl-1.0.2s.tar.gz
tar -zxvf openssl-1.0.2s.tar.gz
cd /usr/local/src/openssl-1.0.2s
./config --prefix=/usr/local/openssl-1.0.2s
make
make install
```

3.安装pcre

```
wget https://ftp.pcre.org/pub/pcre/pcre-8.43.tar.gz
tar -zxvf pcre-8.43.tar.gz
cd /usr/local/src/pcre-8.43
./configure --prefix=/usr/local/pcre-8.43
make
make install
```

4.安装zlib

```
wget https://sourceforge.net/projects/libpng/files/zlib/1.2.11/zlib-
1.2.11.tar.gz
tar -zxvf zlib-1.2.11.tar.gz
cd /usr/local/src/zlib-1.2.11
./configure --prefix=/usr/local/zlib-1.2.11
make
make
```

5.下载nginx-rtmp-module

nginx-rtmp-module的官方github地址：https://github.com/arut/nginx-rtmp-module

使用命令：

```
git clone https://github.com/arut/nginx-rtmp-module.git
```

6.安装Nginx

```
wget http://nginx.org/download/nginx-1.19.1.tar.gz
tar -zxvf nginx-1.19.1.tar.gz
cd /usr/local/src/nginx-1.19.1
./configure --prefix=/usr/local/nginx-1.19.1 --with-
openssl=/usr/local/src/openssl-1.0.2s --with-pcre=/usr/local/src/pcre-8.43 --
with-zlib=/usr/local/src/zlib-1.2.11 --add-module=/usr/local/src/nginx-rtmp-
module --with-http_ssl_module
make
make install
```

这里需要注意的是：安装Nginx时，指定的是openssl、pcre和zlib的源码解压目录，安装完成后Nginx配置文件的完整路径为：/usr/local/nginx-1.19.1/conf/nginx.conf。

配置Nginx

配置Nginx主要是对Nginx的nginx.conf文件进行配置，我们可以在命令行输入如下命令编辑nginx.conf文件。

```
vim /usr/local/nginx-1.19.1/conf/nginx.conf
```

在文件中添加如下内容。

```
rtmp {
    server {
        listen 1935;  #监听的端口
        chunk_size 4096;
        application hls {  #rtmp推流请求路径
            live on;
            hls on;
            hls_path /usr/share/nginx/html/hls;
            hls_fragment 5s;
        }
    }
}
```

其中，hls_path需要可读可写的权限。接下来，我们创建/usr/share/nginx/html/hls 目录。

```
mkdir -p /usr/share/nginx/html/hls
chmod -R 777 /usr/share/nginx/html/hls
```

接下来，修改http中的server模块：

```
server {
    listen       81;
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location / {
        root   /usr/share/nginx/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;
    }
}
```

然后启动Nginx：

```
/usr/local/nginx-1.19.1/sbin/nginx -c /usr/local/nginx-1.19.1/conf/nginx.conf
```

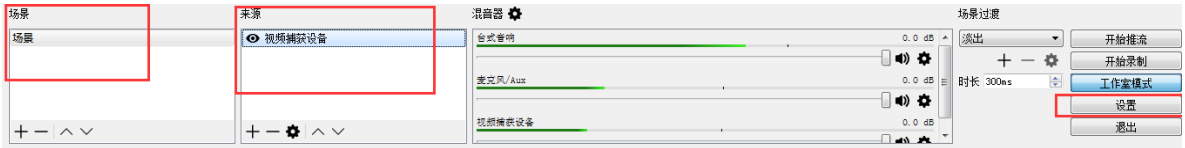
使OBS推流

OBS（Open Broadcaster Software） 是以互联网流媒体直播内容为目的免费和开放源码软件。需要下载这个软件，借助这个软件进行推流（电脑没有摄像头的貌似安装不了。。。）

OBS的下载链接为：https://obsproject.com/zh-cn/download。

安装后，桌面上会有一个如下所示的图表。

打开后我们需要有一个场景，并且在这个场景下有一个流的来源(可以是窗口，如果选的是视频则会自动识别摄像头)，接下来就是设置了。



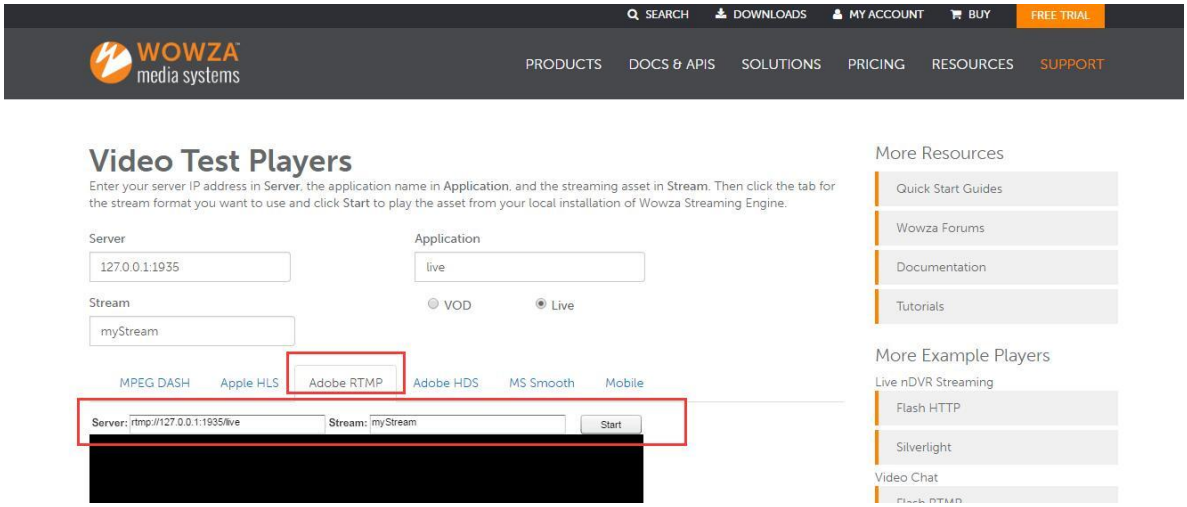
在配置中最需要关注的就是流的配置，由于是自建的流媒体服务器所以我们按照如下所示的方式进行配置。

```
rtmp://你的服务器ip:端口(1935)/live #URL填写流的地址
```

设置完成我们就可以 开始推流了。

拉流测试地址

推荐一个拉流的测试地址，里面针对各种协议都能测试拉流测试，需要注意图中几个地方，由于我们使用的rtmp协议，我们选择这一栏，底下填写我们推流的地址和我们在上面obs的设置里面配置的流的名称，start， ok搞定！！！



十五、并发量太高，Nginx扛不住？这次我错怪Nginx了！！

最近，在服务器上搭建了一套压测环境，不为别的，就为压测下Nginx的性能，到底有没有传说中的那么牛逼！具体环境为：11台虚拟机，全部安装CentOS 6.8 64位操作系统，1台安装部署Nginx，其他10台作为客户端同时以压满CPU的线程向Nginx发送请求，对Nginx进行压测。没想到，出现问题了！！

Nginx报错

Nginx服务器访问量非常高，在Nginx的错误日志中不停的输出如下错误信息。

```
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
2020-07-23 02:53:49 [alert] 13576#0: accept() failed (24: Too many open files)
```

根据错误日志的输出信息，我们可以看出：是打开的文件句柄数太多了，导致Nginx报错了！那我们该如何解决这个问题呢？

问题分析

既然我们能够从Nginx的错误日志中基本能够确定导致问题的原因，那这到底是不是Nginx本身的问题呢？答案为：是，也不全是！

为啥呢？原因很简单：Nginx无法打开那么多的文件句柄，一方面是因为我没有配置Nginx能够打开的最大文件数；另一方面是因为CentOS 6.8操作系统本身对打开的最大文件句柄数有限制，我同样没有配置操作系统的最大文件句柄数。所以说，不全是Nginx的锅！在某种意义上说，我错怪Nginx了！

在CentOS 6.8服务器中，我们可以在命令行输入如下命令来查看服务器默认配置的最大文件句柄数。

```
[root@binghe150 ~]# ulimit -n
1024
```

可以看到，在CentOS 6.8服务器中，默认的最大文件句柄数为1024。

此时，当Nginx的连接数超过1024时，Nginx的错误日志中就会输出如下错误信息。

```
[alert] 13576#0: accept() failed (24: Too many open files)
```

解决问题

那我们该如何解决这个问题呢？其实，也很简单，继续往下看！

使用如下命令可以把打开文件句柄数设置的足够大。

```
ulimit -n 655350
```

同时修改nginx.conf ， 添加如下配置项。

```
worker_rlimit_nofile 655350;
```

注意：上述配置需要与error_log同级别。

这样就可以解决Nginx连接过多的问题，Nginx就可以支持高并发（这里需要配置Nginx）。

另外， `ulimit -n`还会影响到MySQL的并发连接数。把它提高，也可以提高MySQL的并发。

注意： 用 `ulimit -n 655350` 修改只对当前的shell有效，退出后失效。

永久解决问题

若要令修改ulimits的数值永久生效，则必须修改配置文件，可以给ulimit修改命令放入/etc/profile里面，这个方法实在是不方便。

还有一个方法是修改/etc/security/limits.conf配置文件，如下所示。

```
vim /etc/security/limits.conf
```

在文件最后添加如下配置项。

```
* soft nofile 655360
* hard nofile 655360
```

保存并退出vim编辑器。

其中：星号代表全局， soft为软件，hard为硬件，nofile为这里指可打开的文件句柄数。

最后，需要注意的是：要使 limits.conf 文件配置生效，必须要确保 pam_limits.so 文件被加入到启动文件中。查看 /etc/pam.d/login 文件中是否存在如下配置。

```
session required /lib64/security/pam_limits.so
```

不存在，则需要添加上述配置项。

十六、如何实现Nginx的高可用负载均衡?

不得不说，最近小伙伴们的学习热情是越来越高，不断向冰河提出新的想学习的技术。这不，又有小伙伴问我：冰河，你在【Nginx专题】写的文章基本上都是Nginx单机版的，能不能写一篇关于Nginx的高可用的文章呢？我：没问题，安排上！这不，就有了这篇文章！！

Keepalived 简要介绍

Keepalived 是一种高性能的服务器高可用或热备解决方案， Keepalived 可以用来防止服务器单点故障的发生，通过配合 Nginx 可以实现 web 前端服务的高可用。

Keepalived 以 VRRP 协议为实现基础，用 VRRP 协议来实现高可用性(HA)。 VRRP(Virtual RouterRedundancy Protocol)协议是用于实现路由器冗余的协议， VRRP 协议将两台或多台路由器设备虚拟成一个设备，对外提供虚拟路由器 IP(一个或多个)，而在路由器组内部，如果实际拥有这个对外 IP 的路由器如果工作正常的话就是 MASTER，或者是通过算法选举产生， MASTER 实现针对虚拟路由器 IP 的各种网络功能，如 ARP 请求， ICMP，以及数据的转发等；其他设备不拥有该虚拟 IP，状态是 BACKUP，除了接收 MASTER 的VRRP 状态通告信息外，不执行对外的网络功能。

当主机失效时， BACKUP 将接管原先 MASTER 的网络功能。VRRP 协议使用多播数据来传输 VRRP 数据， VRRP 数据使用特殊的虚拟源 MAC 地址发送数据而不是自身网卡的 MAC 地址， VRRP 运行时只有 MASTER 路由器定时发送 VRRP 通告信息，表示 MASTER 工作正常以及虚拟路由器 IP(组)， BACKUP 只接收 VRRP 数据，不发送数据，如果一定时间内没有接收到 MASTER 的通告信息，各 BACKUP 将宣告自己成为 MASTER，发送通告信息，重新进行 MASTER 选举状态。

方案规划

VIP	IP	主机名	Nginx端口	默认主从
192.168.50.130	192.168.50.133	binghe133	88	MASTER
192.168.50.130	192.168.50.134	binghe134	88	BACKUP

操作系统与安装软件如下：

- CentOS 6.8 x64
- keepalived-1.2.18.tar.gz
- nginx-1.19.1.tar.gz

安装Nginx

1.安装依赖环境

```
yum -y install wget gcc-c++ ncurses ncurses-devel cmake make perl bison openssl
openssl-devel gcc* libxml2 libxml2-devel curl-devel libjpeg* libpng* freetype*
autoconf automake zlib* fiex* libxml* libmcrypt* libtool-ltdl-devel* libaio
libaio-devel  bzr libtool
```

2.安装openssl

```
wget https://www.openssl.org/source/openssl-1.0.2s.tar.gz
tar -zxvf openssl-1.0.2s.tar.gz
cd /usr/local/src/openssl-1.0.2s
./config --prefix=/usr/local/openssl-1.0.2s
make
make install
```

3.安装pcre

```
wget https://ftp.pcre.org/pub/pcre/pcre-8.43.tar.gz
tar -zxvf pcre-8.43.tar.gz
cd /usr/local/src/pcre-8.43
./configure --prefix=/usr/local/pcre-8.43
make
make install
```

4.安装zlib

```
wget https://sourceforge.net/projects/libpng/files/zlib/1.2.11/zlib-
1.2.11.tar.gz
tar -zxvf zlib-1.2.11.tar.gz
cd /usr/local/src/zlib-1.2.11
./configure --prefix=/usr/local/zlib-1.2.11
make
make
```

5.下载nginx-rtmp-module

nginx-rtmp-module的官方github地址：https://github.com/arut/nginx-rtmp-module

使用命令：

```
git clone https://github.com/arut/nginx-rtmp-module.git
```

6.安装Nginx

```
wget http://nginx.org/download/nginx-1.19.1.tar.gz
tar -zxvf nginx-1.19.1.tar.gz
cd /usr/local/src/nginx-1.19.1
./configure --prefix=/usr/local/nginx-1.19.1 --with-
openssl=/usr/local/src/openssl-1.0.2s --with-pcre=/usr/local/src/pcre-8.43 --
with-zlib=/usr/local/src/zlib-1.2.11 --add-module=/usr/local/src/nginx-rtmp-
module --with-http_ssl_module
make
make install
```

这里需要注意的是：安装Nginx时，指定的是openssl、pcre和zlib的源码解压目录，安装完成后Nginx配置文件的完整路径为：/usr/local/nginx-1.19.1/conf/nginx.conf。

配置Nginx

在命令行输入如下命令编辑Nginx的nginx.conf文件，如下所示。

```
# vim /usr/local/nginx-1.19.1/conf/nginx.conf
```

编辑后的文件内容如下所示。

```
user root;
worker_processes 1;
#error_log logs/error.log;
#error_log logs/error.log notice;
#error_log logs/error.log info;
#pid logs/nginx.pid;
events {
    worker_connections 1024;
}
http {
    include mime.types;
    default_type application/octet-stream;
    #log_format main '$remote_addr - $remote_user [$time_local] "$request" '
    # '$status $body_bytes_sent "$http_referer" '
    # '"$http_user_agent" "$http_x_forwarded_for"';
    #access_log logs/access.log main;
    sendfile on;
    #tcp_nopush on;
    #keepalive_timeout 0;
    keepalive_timeout 65;
    #gzip on;
    server {
        listen 88;
        server_name localhost;
        #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;
        }
    }
}
```

修改 Nginx 欢迎首页内容（用于后面测试， 用于区分两个节点的 Nginx）：

在binghe133服务器上执行如下操作。

```
# vim /usr/local/nginx-1.19.1/html/index.html
```

在文件title节点下添加如下代码。

```
<h1>Welcome to nginx! 1</h1>
```

在binghe134服务器上执行如下操作。

```
# vim /usr/local/nginx-1.19.1/html/index.html
```

在文件title节点下添加如下代码。

```
<h1>Welcome to nginx! 2</h1>
```

开放端口

在服务器的防火墙中开放88端口，如下所示。

```
vim /etc/sysconfig/iptables
```

添加如下配置。

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 88 -j ACCEPT
```

接下来，输入如下命令重启防火墙。

```
service iptables restart
```

测试Nginx

测试Nginx是否安装成功

```
# /usr/local/nginx-1.19.1/sbin/nginx -t
nginx: the configuration file /usr/local/nginx-1.19.1/conf/nginx.conf syntax is
ok
nginx: configuration file /usr/local/nginx-1.19.1/conf/nginx.conf test is
successful
```

启动Nginx

```
# /usr/local/nginx-1.19.1/sbin/nginx
```

重启 Nginx

```
# /usr/local/nginx-1.19.1/sbin/nginx -s reload
```

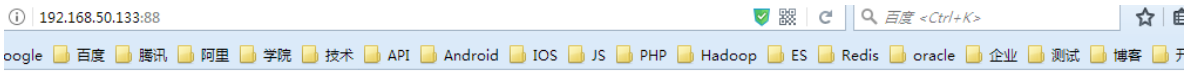
设置Nginx开机自启动

```
# vim /etc/rc.local
```

加入如下一行配置。

```
/usr/local/nginx-1.19.1/sbin/nginx
```

接下来，分别访问两台服务器上Nginx，如下所示。



安装 Keepalived

官方下载链接为：http://www.keepalived.org/download.html 。

上传或下载 keepalived

上传或下载 keepalived（keepalived-1.2.18.tar.gz） 到 /usr/local/src 目录

解压安装

```
# cd /usr/local/src
# tar -zxvf keepalived-1.2.18.tar.gz
# cd keepalived-1.2.18
# ./configure --prefix=/usr/local/keepalived
# make && make install
```

将 keepalived 安装成 Linux 系统服务

因为没有使用 keepalived 的默认路径安装（默认是/usr/local），安装完成之后，需要做一些工作复制默认配置文件到默认路径

```
# mkdir /etc/keepalived
# cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/
```

复制 keepalived 服务脚本到默认的地址

```
# cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/
# cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
# ln -s /usr/local/sbin/keepalived /usr/sbin/
# ln -s /usr/local/keepalived/sbin/keepalived /sbin/
```

设置 keepalived 服务开机启动。

```
# chkconfig keepalived on
```

修改 Keepalived 配置文件

MASTER 节点配置文件（192.168.50.133）

```
# vim /etc/keepalived/keepalived.conf

! Configuration File for keepalived
global_defs {
    ## keepalived 自带的邮件提醒需要开启 sendmail 服务。 建议用独立的监控或第三方 SMTP
    router_id binghe133 ## 标识本节点的字条串，通常为 hostname
}
## keepalived 会定时执行脚本并对脚本执行的结果进行分析，动态调整 vrrp_instance 的优先级。如果脚本执行结果为 0，并且 weight 配置的值大于 0，则优先级相应的增加。如果脚本执行结果非 0，并且 weight配置的值小于 0，则优先级相应的减少。其他情况，维持原本配置的优先级，即配置文件中 priority 对应的值。
vrrp_script chk_nginx {
    script "/etc/keepalived/nginx_check.sh" ## 检测 nginx 状态的脚本路径
    interval 2 ## 检测时间间隔
    weight -20 ## 如果条件成立，权重-20
}
## 定义虚拟路由， VI_1 为虚拟路由的标示符，自己定义名称
vrrp_instance VI_1 {
    state MASTER ## 主节点为 MASTER， 对应的备份节点为 BACKUP
    interface eth0 ## 绑定虚拟 IP 的网络接口，与本机 IP 地址所在的网络接口相同， 我的是 eth0
    virtual_router_id 33 ## 虚拟路由的 ID 号， 两个节点设置必须一样， 可选 IP 最后一段使用，相同的 VRID 为一个组，他将决定多播的 MAC 地址
    mcast_src_ip 192.168.50.133 ## 本机 IP 地址
    priority 100 ## 节点优先级， 值范围 0-254， MASTER 要比 BACKUP 高
    nopreempt ## 优先级高的设置 nopreempt 解决异常恢复后再次抢占的问题
    advert_int 1 ## 组播信息发送间隔，两个节点设置必须一样， 默认 1s
    ## 设置验证信息，两个节点必须一致
    authentication {
        auth_type PASS
        auth_pass 1111 ## 真实生产，按需求对应该过来
    }
```

```
    ## 将 track_script 块加入 instance 配置块
    track_script {
        chk_nginx ## 执行 Nginx 监控的服务
    } #
    # 虚拟 IP 池, 两个节点设置必须一样
    virtual_ipaddress {
        192.168.50.130 ## 虚拟 ip，可以定义多个
    }
}
```

BACKUP 节点配置文件（192.168.50.134）

```
# vim /etc/keepalived/keepalived.conf

! Configuration File for keepalived
global_defs {
    router_id binghe134
}
vrrp_script chk_nginx {
    script "/etc/keepalived/nginx_check.sh"
    interval 2
    weight -20
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth1
    virtual_router_id 33
    mcast_src_ip 192.168.50.134
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        chk_nginx
    }
    virtual_ipaddress {
        192.168.50.130
    }
}
```

编写 Nginx 状态检测脚本

编写 Nginx 状态检测脚本 /etc/keepalived/nginx_check.sh (已在 keepalived.conf 中配置)脚本要求：如果 nginx 停止运行，尝试启动，如果无法启动则杀死本机的 keepalived 进程， keepalied将虚拟 ip 绑定到 BACKUP 机器上。 内容如下。

```
# vim /etc/keepalived/nginx_check.sh

#!/bin/bash
A=`ps -C nginx -no-header |wc -l`
if [ $A -eq 0 ];then
/usr/local/nginx/sbin/nginx
sleep 2
if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then
    killall keepalived
fi
fi
```

保存后，给脚本赋执行权限：

```
# chmod +x /etc/keepalived/nginx_check.sh
```

启动 Keepalived

```
# service keepalived start
Starting keepalived: [ OK ]
```

Keepalived+Nginx 的高可用测试

同时启动192.168.50.133和192.168.50.134上的Nginx和Keepalived，我们通过VIP(192.168.50.130)来访问Nginx，如下所示。

我们关闭192.168.50.133上的Keepalived和Nginx，在192.168.50.133执行如下命令。

```
service keepalived stop
/usr/local/nginx-1.19.1/sbin/nginx -s stop
```

此时，再通过VIP(192.168.50.130)来访问Nginx，如下所示。

我们再开启192.168.50.133上的Keepalived和Nginx，在192.168.50.133执行如下命令：

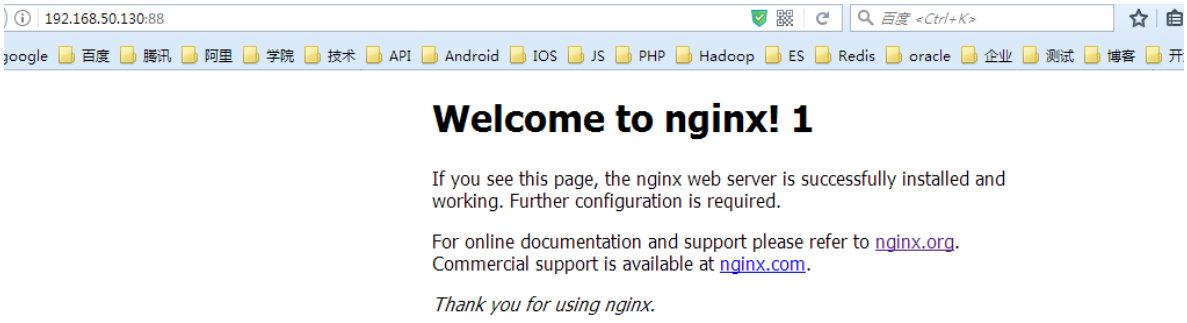
```
/usr/local/nginx-1.19.1/sbin/nginx
service keepalived start
```

或者只执行

```
service keepalived start
```

因为我们写了脚本nginx_check.sh，这个脚本会为我们自动自动Nginx。

此时，我们再通过VIP(192.168.50.130)来访问Nginx，如下所示。



至此，Keepalived + Nginx 实现高可用 Web 负载均衡搭建完毕。

温馨提示

小伙伴们可以到下面的链接下载Keepalived + Nginx 实现高可用 Web 负载均衡的配置文件。

http://download.csdn.net/detail/l1028386804/9855362

十七、如何使用自签CA配置HTTPS加密反向代理访问?

随着互联网的发展，很多公司和个人越来越重视网络的安全性，越来越多的公司采用HTTPS协议来代替了HTTP协议。为何说HTTPS协议比HTTP协议安全呢？小伙伴们自行百度吧！我就不说了。今天，我们就一起来聊聊如何使用自签CA配置Nginx的HTTPS加密反向代理。咳咳，小伙伴们快上车。

如果这篇文章对你有所帮助，请文末留言，点个赞，给个在看和转发，大家的支持是我持续创作的最大动力！

Nginx实现HTTPS

出于安全访问考虑，采用的CA是本机Openssl自签名生成的，因此无法通过互联网工信Root CA验证，所以会出现该网站不受信任或安全证书无效的提示，直接跳过，直接访问即可！

HTTPS的原理和访问过程

服务器必要条件

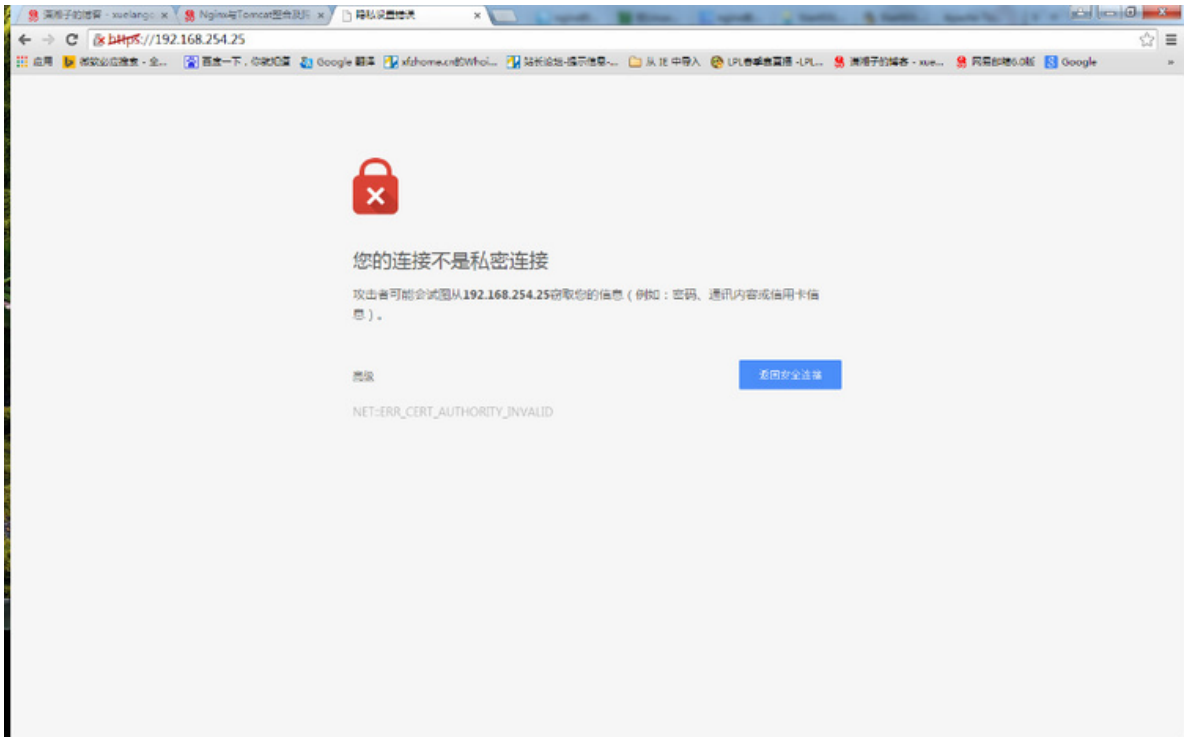
- 一个服务器私钥 KEY文件
- 一张与服务器域名匹配的CA证书（公钥，根据私钥key生成）

访问过程

(1)客户端浏览器通过https协议访问服务器的443端口，并获得服务器的证书（公钥）；客户端浏览器这时候会去找一些互联网可信的RootCA（权威证书颁发机构）验证当前获取到的证书是否合法有效，
PS：这些RootCA是随操作系统一起预设安装在了系统里面的；

(2)如果RootCA验证通过，表示该证书是可信的，并且若证书中标注的服务器名称与当前访问的服务器URL地址一致，就会直接使用该证书中包含的公钥解密服务器通过自己的KEY（私钥）加密后传输过来的网页内容，从而正常显示页面内容；

(3)如果RootCA验证不通过，说明该证书是未获得合法的RootCA签名和授权，因此也就无法证明当前所访问的服务器的权威性，客户端浏览器这时候就会显示一个警告，提示用户当前访问的服务器身份无法得到验证，询问用户是否继续浏览！（通常自签名的CA证书就是这种情况）



这里需要注意，验证CA的有效性，只是证明当前服务器的身份是否合法有效，是否具有公信力以及身份唯一性，防止其他人仿冒该网站；但并不会影响到网页的加密功能，尽管CA证书无法得到权威证明，但是它所包含的公钥和服务器上用于加密页面的私钥依然是匹配的一对，所以服务器用自己的私钥加密的网页内容，客户端浏览器依然是可以用这张证书来解密，正常显示网页内容，所以当用户点击“继续浏览此网站（不推荐）”时，网页就可以打开了；

自签名CA证书生成

1.用Openssl随机生成服务器密钥，和证书申请文件CSR

```
#mkdir /etc/cert       //建立证书和key的保存目录，路径自己决定；
#cd /etc/cert
#openssl genrsa -out moonfly.net.key 1024      //生成1024位加密的服务器私钥moonfly.net.key
#openssl req -new -key moonfly.net.key -out moonfly.net.csr   //制作CSR证书申请文件，需要填写很多信息！

------------制作CSR问题回答过程，红色为输入内容-------------

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:CN                //填写国码CN
State or Province Name (full name) [Berkshire]:ShangHai      //省份
Locality Name (eg, city) [Newbury]:ShangHai              //城市
Organization Name (eg, company) [My Company Ltd]:Moonfly.net  //公司组织名称，自己随便填；

Organizational Unit Name (eg, section) []:IT-Department      //部门名称，可以随便填；
Common Name (eg, your name or your server's hostname) []:www.moonfly.net      //服务器名称，这里注意了，一定要填写你网站的完整域名，因为浏览器会验证证书中的这个服务器名称和访问的服务器URL是否匹配！

Email Address []:moon@moonfly.net       //管理员，负责人邮件地址，可以填写你自己的邮箱；

Please enter the following 'extra' attributes
to be sent with your certificate request      //这里的提示是说如果你想通过第三方权威证书签名机构给你签名这张证书，则需要认真回答下面的问题，我们做自签名，也就是自己给自己压个钢印，所以不需要这么严格了，后面的2个问题就直接回车跳过即可！
A challenge password []:
An optional company name []:
```

2.自己给自己签发证书

在服务器命令行输入如下命令办法证书。

```
#opensslx509 -req -days 3650 -in moonfly.net.csr -signkeymoonfly.net.key -outmoonfly.net.crt
```

- -days 3650 证书的有效期，自己给自己颁发证书，想有多久有效期，就弄多久，我一下弄了10年的有效期；
- -inmoonfly.net.csr指定CSR文件
- -signkeymoonfly.net.key指定服务器的私钥key文件
- -outmoonfly.net.crt 设置生成好的证书文件名

一条命令，自己给自己压钢印的身份证 moonfly.net.crt 就诞生了！

注：其实严格来讲，这里生成的只是一张RootCA，并不是严格意义上的服务器证书ServerCA，真正的ServerCA是需要利用这张RootCA再给服务器签署办法出来的证书才算；不过我们这里只讲如何实现网页的SSL加密，所以就直接使用RootCA了，也是能正常实现加密功能的！

NGINX配置启用HTTPS并配置加密反向代理

```
# HTTPS server
#
server {
    listen       443 ssl;       //监听443端口
    server_name  localhost;
    ssl          on;            //启用ssl加密

    ssl_certificate      /etc/cert/moonfly.net.crt;  //服务器证书crt文件
    ssl_certificate_key  /etc/cert/moonfly.net.key;  //服务器私钥key文件

    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;

    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;

    location / {
#        root   html;
#        index  index.html index.htm;
     proxy_pass http://192.168.254.25:8080/;    //此处注意，反向代理路径
    }
}
```

配置文件修改完毕后，用nginx -t 测试下配置无误，就reload一下nginx服务，检查443端口是否在监听：

```
[root@xuelang opt]# /usr/local/nginx/sbin/nginx -t
nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful
[root@xuelang opt]# /usr/local/nginx/sbin/nginx -s reload
[root@xuelang opt]# netstat -lan | grep 443
tcp        0      0 0.0.0.0:443                 0.0.0.0:*                   LISTEN
[root@xuelang opt]#
```

配置完毕，https已经在工作了，现在可以通过https访问网站了

十八、如何基于主从模式搭建Nginx+Keepalived双机热备环境?

最近出版了《海量数据处理与大数据技术实战》，详情可以关注 **冰河技术** 微信公众号，查看《我的《海量数据处理与大数据技术实战》出版啦！》一文。

也有不少小伙伴让我更新一篇基于主从模式搭建Nginx+Keepalived双机热备的环境，怎么办呢？那必须安排上啊！不多说了，我们直接进入正文。

负载均衡技术

负载均衡技术对于一个网站尤其是大型网站的web服务器集群来说是至关重要的！做好负载均衡架构，可以实现故障转移和高可用环境，避免单点故障，保证网站健康持续运行。

由于业务扩展，网站的访问量不断加大，负载越来越高。现需要在web前端放置nginx负载均衡,同时结合keepalived对前端nginx实现HA高可用。

1）nginx进程基于Master+Slave(worker)多进程模型，自身具有非常稳定的子进程管理功能。在Master进程分配模式下，Master进程永远不进行业务处理，只是进行任务分发，从而达到Master进程的存活高可靠性，Slave(worker)进程所有的业务信号都 由主进程发出，Slave(worker)进程所有的超时任务都会被Master中止，属于非阻塞式任务模型。

2）Keepalived是Linux下面实现VRRP备份路由的高可靠性运行件。基于Keepalived设计的服务模式能够真正做到主服务器和备份服务器故障时IP瞬间无缝交接。二者结合，可以构架出比较稳定的软件LB方案。

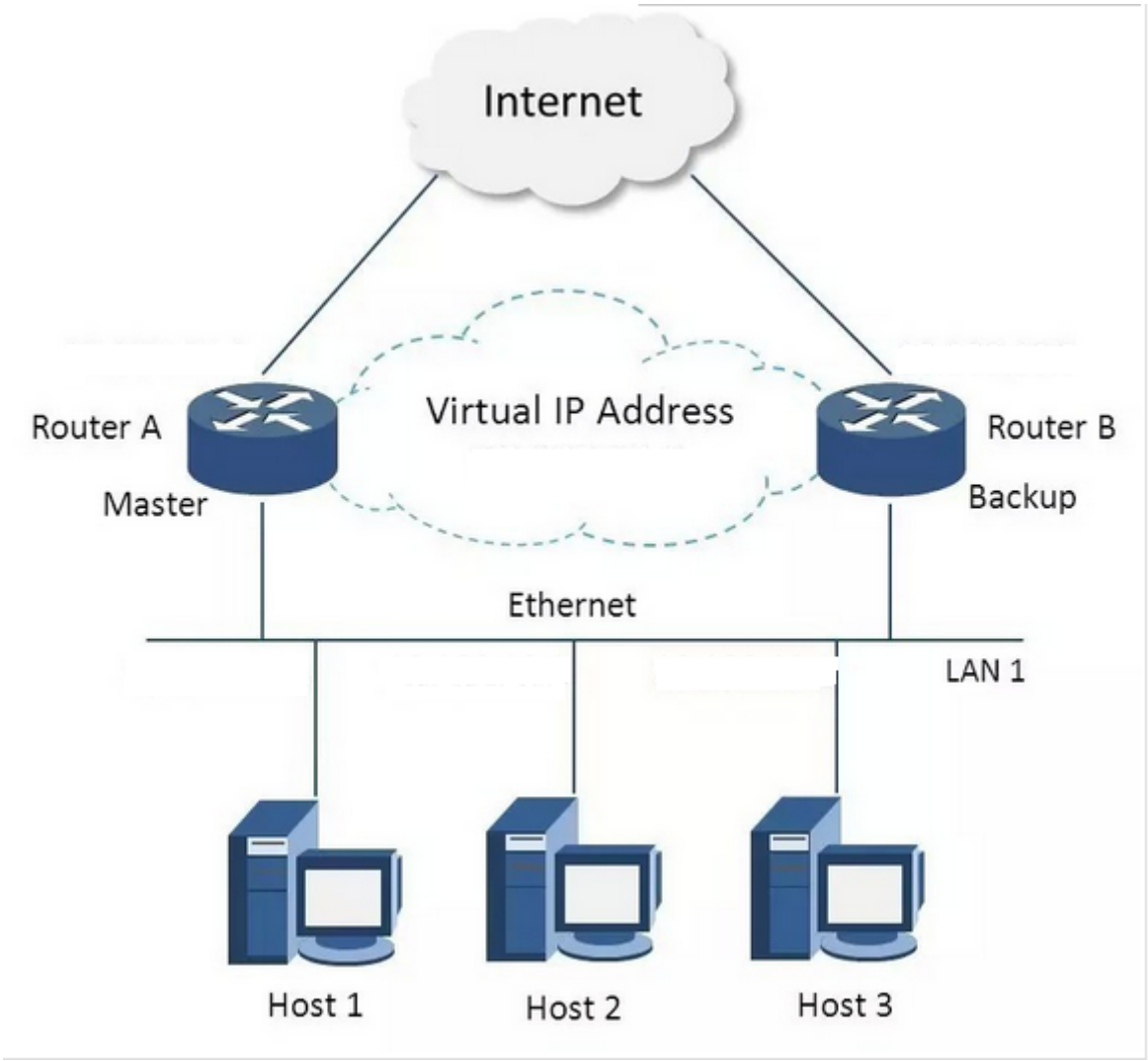
Keepalived介绍

Keepalived是一个基于VRRP协议来实现的服务高可用方案，可以利用其来避免IP单点故障，类似的工具还有heartbeat、corosync、pacemaker。但是它一般不会单独出现，而是与其它负载均衡技术（如lvs、haproxy、nginx）一起工作来达到集群的高可用。

VRRP协议

VRRP全称 Virtual Router Redundancy Protocol，即 虚拟路由冗余协议。可以认为它是实现路由器高可用的容错协议，即将N台提供相同功能的路由器组成一个路由器组(Router Group)，这个组里面有一个master和多个backup，但在外界看来就像一台一样，构成虚拟路由器，拥有一个虚拟IP（vip，也就是路由器所在局域网内其他机器的默认路由），占有这个IP的master实际负责ARP相应和转发IP数据包，组中的其它路由器作为备份的角色处于待命状态。master会发组播消息，当backup在超时时间内收不到vrrp包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master，保证路由器的高可用。

在VRRP协议实现里，虚拟路由器使用 00-00-5E-00-01-XX 作为虚拟MAC地址，XX就是唯一的 VRID（Virtual Router IDentifier），这个地址同一时间只有一个物理路由器占用。在虚拟路由器里面的物理路由器组里面通过多播IP地址 224.0.0.18 来定时发送通告消息。每个Router都有一个 1-255 之间的优先级别，级别最高的（highest priority）将成为主控（master）路由器。通过降低master的优先权可以让处于backup状态的路由器抢占（pro-empt）主路由器的状态，两个backup优先级相同的IP地址较大者为master，接管虚拟IP。



keepalived与heartbeat/corosync等比较

Heartbeat、Corosync、Keepalived这三个集群组件我们到底选哪个好呢?

首先要说明的是，Heartbeat、Corosync是属于同一类型，Keepalived与Heartbeat、Corosync，根本不是同一类型的。
Keepalived使用的vrrp协议方式，虚拟路由冗余协议 (Virtual Router Redundancy Protocol，简称VRRP)；
Heartbeat或Corosync是基于主机或网络服务的高可用方式；

简单的说就是，Keepalived的目的是模拟路由器的高可用，Heartbeat或Corosync的目的是实现Service的高可用。
所以一般Keepalived是实现前端高可用，常用的前端高可用的组合有，就是我们常见的LVS+Keepalived、Nginx+Keepalived、HAproxy+Keepalived。而Heartbeat或Corosync是实现服务的高可用，常见的组合有Heartbeat v3(Corosync)+Pacemaker+NFS+Httpd 实现Web服务器的高可用、Heartbeat v3(Corosync)+Pacemaker+NFS+MySQL 实现MySQL服务器的高可用。

总结一下，Keepalived中实现轻量级的高可用，一般用于前端高可用，且不需要共享存储，一般常用于两个节点的高可用。而Heartbeat(或Corosync)一般用于服务的高可用，且需要共享存储，一般用于多节点的高可用。这个问题我们说明白了。

那heartbaet与corosync又应该选择哪个好?

一般用corosync，因为corosync的运行机制更优于heartbeat，就连从heartbeat分离出来的pacemaker都说在以后的开发当中更倾向于corosync，所以现在corosync+pacemaker是最佳组合。

双机高可用一般是通过虚拟IP（飘移IP）方法来实现的，基于Linux/Unix的IP别名技术。

双机高可用方法目前分为两种：

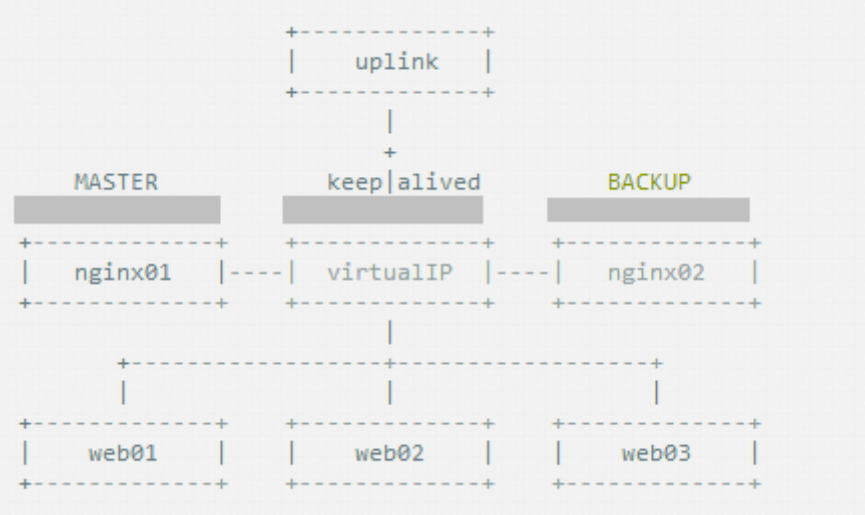
1）双机主从模式：即前端使用两台服务器，一台主服务器和一台热备服务器，正常情况下，主服务器绑定一个公网虚拟IP，提供负载均衡服务，热备服务器处于空闲状态；当主服务器发生故障时，热备服务器接管主服务器的公网虚拟IP，提供负载均衡服务；但是热备服务器在主机器不出现故障的时候，永远处于浪费状态，对于服务器不多的网站，该方案不经济实惠。

2）双机主主模式：即前端使用两台负载均衡服务器，互为主备，且都处于活动状态，同时各自绑定一个公网虚拟IP，提供负载均衡服务；当其中一台发生故障时，另一台接管发生故障服务器的公网虚拟IP（这时由非故障机器一台负担所有的请求）。这种方案，经济实惠，非常适合于当前架构环境。

今天在此分享下Nginx+keepalived实现高可用负载均衡的主从模式的操作记录：

keepalived可以认为是VRRP协议在Linux上的实现，主要有三个模块，分别是core、check和vrrp。

- core模块为keepalived的核心，负责主进程的启动、维护以及全局配置文件的加载和解析。
- check负责健康检查，包括常见的各种检查方式。
- vrrp模块是来实现VRRP协议的。



环境说明

操作系统：centos6.8，64位
master机器（master-node）：103.110.98.14/192.168.1.14
slave机器（slave-node）：103.110.98.24/192.168.1.24
公用的虚拟IP（VIP）：103.110.98.20　//负载均衡器上配置的域名都解析到这个VIP上

应用环境如下

应用系统	域名/虚拟目录	对应的后端应用服务器及URL
svn	http://dev.wangshibo.com/svn	http://192.168.1.108/svn
svn web管理	http://dev.wangshibo.com/submin	http://192.168.1.108/submin
网站	http://www.wangshibo.com	http://192.168.1.101 http://192.168.1.102 http://192.168.1.118
OA	http://oa.wangshibo.com	http://192.168.1.101:8080 http://192.168.1.102:8080

环境安装

安装nginx和keepalive服务（master-node和slave-node两台服务器上的安装操作完全一样）。

安装依赖

```
[root@master-node ~]# yum -y install gcc pcre-devel zlib-devel openssl-devel
```

大家可以到链接：https://download.csdn.net/download/l1028386804/10376846下载nginx1.9.7+keepalive1.3.2，也可以将nginx.1.9.7更新为nginx1.19.2。nginx1.19.2与nginx1.9.7的安装方式相同，这里我以nginx1.9.7为例进行安装。

```
[root@master-node ~]# cd /usr/local/src/
[root@master-node src]# wget http://nginx.org/download/nginx-1.9.7.tar.gz
[root@master-node src]# wget http://www.keepalived.org/software/keepalived-
1.3.2.tar.gz
```

安装nginx

```
[root@master-node src]# tar -zvxf nginx-1.9.7.tar.gz
[root@master-node src]# cd nginx-1.9.7
```

添加www用户，其中-M参数表示不添加用户家目录，-s参数表示指定shell类型

```
[root@master-node nginx-1.9.7]# useradd www -M -s /sbin/nologin
[root@master-node nginx-1.9.7]# vim auto/cc/gcc
#将这句注释掉 取消Debug编译模式 大概在179行
#CFLAGS="$CFLAGS -g"
[root@master-node nginx-1.9.7]# ./configure --prefix=/usr/local/nginx --user=www
--group=www --with-http_ssl_module --with-http_flv_module --with-
http_stub_status_module --with-http_gzip_static_module --with-pcre
[root@master-node nginx-1.9.7]# make && make install
```

安装keepalived

```
[root@master-node src]# tar -zvxf keepalived-1.3.2.tar.gz
[root@master-node src]# cd keepalived-1.3.2
[root@master-node keepalived-1.3.2]# ./configure
[root@master-node keepalived-1.3.2]# make && make install
[root@master-node keepalived-1.3.2]# cp /usr/local/src/keepalived-
1.3.2/keepalived/etc/init.d/keepalived /etc/rc.d/init.d/
[root@master-node keepalived-1.3.2]# cp /usr/local/etc/sysconfig/keepalived
/etc/sysconfig/
[root@master-node keepalived-1.3.2]# mkdir /etc/keepalived
[root@master-node keepalived-1.3.2]# cp
/usr/local/etc/keepalived/keepalived.conf /etc/keepalived/
[root@master-node keepalived-1.3.2]# cp /usr/local/sbin/keepalived /usr/sbin/
```

将nginx和keepalive服务加入开机启动服务

```
[root@master-node keepalived-1.3.2]# echo "/usr/local/nginx/sbin/nginx" >>
/etc/rc.local
[root@master-node keepalived-1.3.2]# echo "/etc/init.d/keepalived start" >>
/etc/rc.local
```

配置服务

先关闭SElinux、配置防火墙 （master和slave两台负载均衡机都要做）

```
[root@master-node ~]# vim /etc/sysconfig/selinux
#SELINUX=enforcing                      #注释掉
#SELINUXTYPE=targeted                  #注释掉
SELINUX=disabled                        #增加
[root@master-node ~]# setenforce 0                                  #使配置立即生效

[root@master-node ~]# vim /etc/sysconfig/iptables
.......
-A INPUT -s 103.110.98.0/24 -d 224.0.0.18 -j ACCEPT                      #允许
组播地址通信
-A INPUT -s 192.168.1.0/24 -d 224.0.0.18 -j ACCEPT
-A INPUT -s 103.110.98.0/24 -p vrrp -j ACCEPT                             #
允许 VRRP（虚拟路由器冗余协）通信
-A INPUT -s 192.168.1.0/24 -p vrrp -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT      #开通80端口
访问

[root@master-node ~]# /etc/init.d/iptables restart
#重启防火墙使配置生效
```

配置nginx

master-node和slave-node两台服务器的nginx的配置完全一样,主要是配置/usr/local/nginx/conf/nginx.conf的http，当然也可以配置vhost虚拟主机目录，然后配置vhost下的比如LB.conf文件。

其中:
多域名指向是通过虚拟主机（配置http下面的server）实现;
同一域名的不同虚拟目录通过每个server下面的不同location实现;
到后端的服务器在vhost/LB.conf下面配置upstream,然后在server或location中通过proxy_pass引用。

要实现前面规划的接入方式，LB.conf的配置如下（添加proxy_cache_path和proxy_temp_path这两行，表示打开nginx的缓存功能）

```
[root@master-node ~]# vim /usr/local/nginx/conf/nginx.conf
user  www;
worker_processes  8;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    worker_connections  65535;
}


http {
    include       mime.types;
    default_type  application/octet-stream;
    charset utf-8;

    ######
    ## set access log format
    ######
    log_format  main  '$http_x_forwarded_for $remote_addr $remote_user
[$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_cookie" $host $request_time';

    #######
    ## http setting
    #######
    sendfile       on;
    tcp_nopush     on;
    tcp_nodelay    on;
    keepalive_timeout  65;
    proxy_cache_path /var/www/cache levels=1:2 keys_zone=mycache:20m
max_size=2048m inactive=60m;
    proxy_temp_path /var/www/cache/tmp;

    fastcgi_connect_timeout 3000;
    fastcgi_send_timeout 3000;
    fastcgi_read_timeout 3000;
    fastcgi_buffer_size 256k;
    fastcgi_buffers 8 256k;
    fastcgi_busy_buffers_size 256k;
    fastcgi_temp_file_write_size 256k;
    fastcgi_intercept_errors on;

    #
    client_header_timeout 600s;
    client_body_timeout 600s;
   # client_max_body_size 50m;
    client_max_body_size 100m;                #允许客户端请求的最大单个文件字节数
```

```
    client_body_buffer_size 256k;              #缓冲区代理缓冲请求的最大字节数，可以理解为先保存到本地再传给用户

    gzip  on;
    gzip_min_length  1k;
    gzip_buffers     4 16k;
    gzip_http_version 1.1;
    gzip_comp_level 9;
    gzip_types       text/plain application/x-javascript text/css application/xml text/javascript application/x-httpd-php;
    gzip_vary on;

    ## includes vhosts
    include vhosts/*.conf;
}
```

```
[root@master-node ~]# mkdir /usr/local/nginx/conf/vhosts
[root@master-node ~]# mkdir /var/www/cache
[root@master-node ~]# ulimit 65535
```

```
[root@master-node ~]# vim /usr/local/nginx/conf/vhosts/LB.conf
upstream LB-WWW {
      ip_hash;
      server 192.168.1.101:80 max_fails=3 fail_timeout=30s;     #max_fails = 3为允许失败的次数，默认值为1
      server 192.168.1.102:80 max_fails=3 fail_timeout=30s;     #fail_timeout = 30s 当max_fails次失败后，暂停将请求分发到该后端服务器的时间
      server 192.168.1.118:80 max_fails=3 fail_timeout=30s;
    }

upstream LB-OA {
      ip_hash;
      server 192.168.1.101:8080 max_fails=3 fail_timeout=30s;
      server 192.168.1.102:8080 max_fails=3 fail_timeout=30s;
}

  server {
      listen      80;
      server_name dev.wangshibo.com;

      access_log  /usr/local/nginx/logs/dev-access.log main;
      error_log  /usr/local/nginx/logs/dev-error.log;

      location /svn {
         proxy_pass http://192.168.1.108/svn/;
         proxy_redirect off ;
         proxy_set_header Host $host;
         proxy_set_header X-Real-IP $remote_addr;
         proxy_set_header REMOTE-HOST $remote_addr;
         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
         proxy_connect_timeout 300;              #跟后端服务器连接超时时间，发起握手等候响应时间
         proxy_send_timeout 300;                 #后端服务器回传时间，就是在规定时间内后端服务器必须传完所有数据
         proxy_read_timeout 600;                 #连接成功后等待后端服务器的响应时间，已经进入后端的排队之中等候处理
```

```
        proxy_buffer_size 256k;                 #代理请求缓冲区,会保存用户的头信息以供
nginx进行处理
        proxy_buffers 4 256k;                   #同上，告诉nginx保存单个用几个buffer
最大用多少空间
        proxy_busy_buffers_size 256k;           #如果系统很忙时候可以申请最大的
proxy_buffers
        proxy_temp_file_write_size 256k;        #proxy缓存临时文件的大小
        proxy_next_upstream error timeout invalid_header http_500 http_503
http_404;
        proxy_max_temp_file_size 128m;
        proxy_cache mycache;
        proxy_cache_valid 200 302 60m;
        proxy_cache_valid 404 1m;
       }

      location /submin {
        proxy_pass http://192.168.1.108/submin/;
        proxy_redirect off ;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_connect_timeout 300;
        proxy_send_timeout 300;
        proxy_read_timeout 600;
        proxy_buffer_size 256k;
        proxy_buffers 4 256k;
        proxy_busy_buffers_size 256k;
        proxy_temp_file_write_size 256k;
        proxy_next_upstream error timeout invalid_header http_500 http_503
http_404;
        proxy_max_temp_file_size 128m;
        proxy_cache mycache;
        proxy_cache_valid 200 302 60m;
        proxy_cache_valid 404 1m;
       }
    }

server {
     listen       80;
     server_name  www.wangshibo.com;

      access_log  /usr/local/nginx/logs/www-access.log main;
      error_log  /usr/local/nginx/logs/www-error.log;

     location / {
        proxy_pass http://LB-WWW;
        proxy_redirect off ;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_connect_timeout 300;
        proxy_send_timeout 300;
        proxy_read_timeout 600;
        proxy_buffer_size 256k;
        proxy_buffers 4 256k;
        proxy_busy_buffers_size 256k;
```

```
          proxy_temp_file_write_size 256k;
          proxy_next_upstream error timeout invalid_header http_500 http_503
http_404;
          proxy_max_temp_file_size 128m;
          proxy_cache mycache;
          proxy_cache_valid 200 302 60m;
          proxy_cache_valid 404 1m;
         }
}

 server {
       listen       80;
       server_name  oa.wangshibo.com;

      access_log  /usr/local/nginx/logs/oa-access.log main;
      error_log  /usr/local/nginx/logs/oa-error.log;

       location / {
         proxy_pass http://LB-OA;
         proxy_redirect off ;
         proxy_set_header Host $host;
         proxy_set_header X-Real-IP $remote_addr;
         proxy_set_header REMOTE-HOST $remote_addr;
         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
         proxy_connect_timeout 300;
         proxy_send_timeout 300;
         proxy_read_timeout 600;
         proxy_buffer_size 256k;
         proxy_buffers 4 256k;
         proxy_busy_buffers_size 256k;
         proxy_temp_file_write_size 256k;
         proxy_next_upstream error timeout invalid_header http_500 http_503
http_404;
         proxy_max_temp_file_size 128m;
         proxy_cache mycache;
         proxy_cache_valid 200 302 60m;
         proxy_cache_valid 404 1m;
        }
}
```

验证Nginx配置

验证方法（保证从负载均衡器本机到后端真实服务器之间能正常通信）：
1）首先在本机用IP访问上面LB.cong中配置的各个后端真实服务器的url
2）然后在本机用域名和路径访问上面LB.cong中配置的各个后端真实服务器的域名/虚拟路径

后端应用服务器的nginx配置，这里选择192.168.1.108作为例子进行说明。

由于这里的192.168.1.108机器是openstack的虚拟机，没有外网ip，不能解析域名。所以在server_name处也将ip加上，使得用ip也可以访问。

```
[root@108-server ~]# cat /usr/local/nginx/conf/vhosts/svn.conf
server {
listen 80;
#server_name dev.wangshibo.com;
server_name dev.wangshibo.com 192.168.1.108;
```

```
access_log /usr/local/nginx/logs/dev.wangshibo-access.log main;
error_log /usr/local/nginx/logs/dev.wangshibo-error.log;

location / {
root /var/www/html;
index index.html index.php index.htm;
}
}

[root@108-server ~]# ll /var/www/html/
drwxr-xr-x. 2 www www 4096 Dec 7 01:46 submin
drwxr-xr-x. 2 www www 4096 Dec 7 01:45 svn
[root@108-server ~]# cat /var/www/html/svn/index.html
this is the page of svn/192.168.1.108
[root@108-server ~]# cat /var/www/html/submin/index.html
this is the page of submin/192.168.1.108

[root@108-server ~]# cat /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.1.108 dev.wangshibo.com

[root@108-server ~]# curl http://dev.wangshibo.com       //由于是内网机器不能联网，亦不能解析域名。所以用域名访问没有反应。只能用ip访问
[root@ops-server4 vhosts]# curl http://192.168.1.108
this is 192.168.1.108 page!!!
[root@ops-server4 vhosts]# curl http://192.168.1.108/svn/            //最后一个/符号要加上，否则访问不了。
this is the page of svn/192.168.1.108
[root@ops-server4 vhosts]# curl http://192.168.1.108/submin/
this is the page of submin/192.168.1.108
```

然后在master-node和slave-node两台负载机器上进行测试（iptables防火墙要开通80端口）：

```
[root@master-node ~]# curl http://192.168.1.108/svn/
this is the page of svn/192.168.1.108
[root@master-node ~]# curl http://192.168.1.108/submin/
this is the page of submin/192.168.1.108
```

浏览器访问：
在本机host绑定dev.wangshibo.com，如下，即绑定到master和slave机器的公网ip上测试是否能正常访问（nginx+keepalive环境正式完成后，域名解析到的真正地址是VIP地址）
103.110.98.14 dev.wangshibo.com
103.110.98.24 dev.wangshibo.com

keepalived配置

1）master-node负载机上的keepalived配置

```
[root@master-node ~]# cp /etc/keepalived/keepalived.conf
/etc/keepalived/keepalived.conf.bak
[root@master-node ~]# vim /etc/keepalived/keepalived.conf
```

```
! Configuration File for keepalived     #全局定义

global_defs {
notification_email {      #指定keepalived在发生事件时(比如切换)发送通知邮件的邮箱
ops@wangshibo.cn    #设置报警邮件地址，可以设置多个，每行一个。 需开启本机的sendmail服务
tech@wangshibo.cn
}

notification_email_from ops@wangshibo.cn    #keepalived在发生诸如切换操作时需要发送
email通知地址
smtp_server 127.0.0.1       #指定发送email的smtp服务器
smtp_connect_timeout 30     #设置连接smtp server的超时时间
router_id master-node      #运行keepalived的机器的一个标识，通常可设为hostname。故障发生
时，发邮件时显示在邮件主题中的信息。
}

vrrp_script chk_http_port {       #检测nginx服务是否在运行。有很多方式，比如进程，用脚本
检测等等
    script "/opt/chk_nginx.sh"    #这里通过脚本监测
    interval 2                    #脚本执行间隔，每2s检测一次
    weight -5                     #脚本结果导致的优先级变更，检测失败（脚本返回非0）则优先
级 -5
    fall 2                     #检测连续2次失败才算确定是真失败。会用weight减少优先级（1-
255之间）
    rise 1                     #检测1次成功就算成功。但不修改优先级
}

vrrp_instance VI_1 {     #keepalived在同一virtual_router_id中priority（0-255）最大的
会成为master，也就是接管VIP，当priority最大的主机发生故障后次priority将会接管
    state MASTER     #指定keepalived的角色，MASTER表示此主机是主服务器，BACKUP表示此主机
是备用服务器。注意这里的state指定instance(Initial)的初始状态，就是说在配置好后，这台服务器的
初始状态就是这里指定的，但这里指定的不算，还是得要通过竞选通过优先级来确定。如果这里设置为
MASTER，但如若他的优先级不及另外一台，那么这台在发送通告时，会发送自己的优先级，另外一台发现优
先级不如自己的高，那么他会就回抢占为MASTER
    interface em1           #指定HA监测网络的接口。实例绑定的网卡，因为在配置虚拟IP的时候必
须是在已有的网卡上添加的
    mcast_src_ip 103.110.98.14  # 发送多播数据包时的源IP地址，这里注意了，这里实际上就是
在哪个地址上发送VRRP通告，这个非常重要，一定要选择稳定的网卡端口来发送，这里相当于heartbeat的
心跳端口，如果没有设置那么就用默认的绑定的网卡的IP，也就是interface指定的IP地址
    virtual_router_id 51          #虚拟路由标识，这个标识是一个数字，同一个vrrp实例使用唯
一的标识。即同一vrrp_instance下，MASTER和BACKUP必须是一致的
    priority 101                  #定义优先级，数字越大，优先级越高，在同一个
vrrp_instance下，MASTER的优先级必须大于BACKUP的优先级
    advert_int 1                  #设定MASTER与BACKUP负载均衡器之间同步检查的时间间隔，
单位是秒
    authentication {              #设置验证类型和密码。主从必须一样
        auth_type PASS            #设置vrrp验证类型，主要有PASS和AH两种
        auth_pass 1111            #设置vrrp验证密码，在同一个vrrp_instance下，MASTER与
BACKUP必须使用相同的密码才能正常通信
    }
```

```
    virtual_ipaddress {           #VRRP HA 虚拟地址 如果有多个VIP，继续换行填写
        103.110.98.20
    }

    track_script {                          #执行监控的服务。注意这个设置不能紧挨着写在vrrp_script配置块的后面（实验中碰过的坑），否则nginx监控失效！！
       chk_http_port                        #引用VRRP脚本，即在 vrrp_script 部分指定的名字。定期运行它们来改变优先级，并最终引发主备切换。
    }
}
```

slave-node负载机上的keepalived配置

```
[root@slave-node ~]# cp /etc/keepalived/keepalived.conf /etc/keepalived/keepalived.conf.bak
[root@slave-node ~]# vim /etc/keepalived/keepalived.conf
```

```
! Configuration File for keepalived

global_defs {
notification_email {
ops@wangshibo.cn
tech@wangshibo.cn
}

notification_email_from ops@wangshibo.cn
smtp_server 127.0.0.1
smtp_connect_timeout 30
router_id slave-node
}

vrrp_script chk_http_port {
    script "/opt/chk_nginx.sh"
    interval 2
    weight -5
    fall 2
    rise 1
}

vrrp_instance VI_1 {
    state BACKUP
    interface em1
    mcast_src_ip 103.110.98.24
    virtual_router_id 51
    priority 99
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        103.110.98.20
    }

    track_script {
       chk_http_port
```

```
    }

}
```

让keepalived监控NginX的状态：

1）经过前面的配置，如果master主服务器的keepalived停止服务，slave从服务器会自动接管VIP对外服务；
一旦主服务器的keepalived恢复，会重新接管VIP。 但这并不是我们需要的，我们需要的是当NginX停止服务的时候能够自动切换。
2）keepalived支持配置监控脚本，我们可以通过脚本监控NginX的状态，如果状态不正常则进行一系列的操作，最终仍不能恢复NginX则杀掉keepalived，使得从服务器能够接管服务。

如何监控NginX的状态
最简单的做法是监控NginX进程，更靠谱的做法是检查NginX端口，最靠谱的做法是检查多个url能否获取到页面。

注意：这里要提示一下keepalived.conf中vrrp_script配置区的script一般有2种写法：

1）通过脚本执行的返回结果，改变优先级，keepalived继续发送通告消息，backup比较优先级再决定。这是直接监控Nginx进程的方式。
2）脚本里面检测到异常，直接关闭keepalived进程，backup机器接收不到advertisement会抢占IP。这是检查NginX端口的方式。

上文script配置部分，"killall -0 nginx"属于第1种情况，"/opt/chk_nginx.sh" 属于第2种情况。个人更倾向于通过shell脚本判断，但有异常时exit 1，正常退出exit 0，然后keepalived根据动态调整的vrrp_instance 优先级选举决定是否抢占VIP：

- 如果脚本执行结果为0，并且weight配置的值大于0，则优先级相应的增加
- 如果脚本执行结果非0，并且weight配置的值小于0，则优先级相应的减少
- 其他情况，原本配置的优先级不变，即配置文件中priority对应的值。

提示：
优先级不会不断的提高或者降低

可以编写多个检测脚本并为每个检测脚本设置不同的weight（在配置中列出就行）

不管提高优先级还是降低优先级，最终优先级的范围是在[1,254]，不会出现优先级小于等于0或者优先级大于等于255的情况
在MASTER节点的 vrrp_instance 中 配置 nopreempt ，当它异常恢复后，即使它 prio 更高也不会抢占，这样可以避免正常情况下做无谓的切换

以上可以做到利用脚本检测业务进程的状态，并动态调整优先级从而实现主备切换。

另外：在默认的keepalive.conf里面还有 virtual_server,real_server 这样的配置，我们这用不到，它是为lvs准备的。

如何尝试恢复服务
由于keepalived只检测本机和他机keepalived是否正常并实现VIP的漂移，而如果本机nginx出现故障不会则不会漂移VIP。
所以编写脚本来判断本机nginx是否正常，如果发现NginX不正常，重启之。等待3秒再次校验，仍然失败则不再尝试，关闭keepalived，其他主机此时会接管VIP；

根据上述策略很容易写出监控脚本。此脚本必须在keepalived服务运行的前提下才有效！如果在keepalived服务先关闭的情况下，那么nginx服务关闭后就不能实现自启动了。

该脚本检测ngnix的运行状态，并在nginx进程不存在时尝试重新启动ngnix，如果启动失败则停止keepalived，准备让其它机器接管。
监控脚本如下（master和slave都要有这个监控脚本）：

```
[root@master-node ~]# vim /opt/chk_nginx.sh

#!/bin/bash
counter=$(ps -C nginx --no-heading|wc -l)
if [ "${counter}" = "0" ]; then
    /usr/local/nginx/sbin/nginx
    sleep 2
    counter=$(ps -C nginx --no-heading|wc -l)
    if [ "${counter}" = "0" ]; then
        /etc/init.d/keepalived stop
    fi
fi
```

```
[root@master-node ~]# chmod 755 /opt/chk_nginx.sh
[root@master-node ~]# sh /opt/chk_nginx.sh
80/tcp open http
```

此架构需考虑的问题
1）master没挂，则master占有vip且nginx运行在master上
2）master挂了，则slave抢占vip且在slave上运行nginx服务
3）如果master上的nginx服务挂了，则nginx会自动重启，重启失败后会自动关闭keepalived，这样vip资源也会转移到slave上。
4）检测后端服务器的健康状态
5）master和slave两边都开启nginx服务，无论master还是slave，当其中的一个keepalived服务停止后，vip都会漂移到keepalived服务还在的节点上；
如果要想使nginx服务挂了，vip也漂移到另一个节点，则必须用脚本或者在配置文件里面用shell命令来控制。（nginx服务宕停后会自动启动，启动失败后会强制关闭keepalived，从而致使vip资源漂移到另一台机器上）

最后验证（将配置的后端应用域名都解析到VIP地址上）：关闭主服务器上的keepalived或nginx，vip都会自动飘到从服务器上。

验证keepalived服务故障情况：
1）先后在master、slave服务器上启动nginx和keepalived，保证这两个服务都正常开启:

```
[root@master-node ~]# /usr/local/nginx/sbin/nginx
[root@master-node ~]# /etc/init.d/keepalived start
[root@slave-node ~]# /usr/local/nginx/sbin/nginx
[root@slave-node ~]# /etc/init.d/keepalived start
```

2）在主服务器上查看是否已经绑定了虚拟IP

```
[root@master-node ~]# ip addr
.......
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
link/ether 44:a8:42:17:3d:dd brd ff:ff:ff:ff:ff:ff
inet 103.110.98.14/26 brd 103.10.86.63 scope global em1
valid_lft forever preferred_lft forever
inet 103.110.98.20/32 scope global em1
valid_lft forever preferred_lft forever
inet 103.110.98.20/26 brd 103.10.86.63 scope global secondary em1:0
valid_lft forever preferred_lft forever
inet6 fe80::46a8:42ff:fe17:3ddd/64 scope link
valid_lft forever preferred_lft forever
```

3）停止主服务器上的keepalived:

```
[root@master-node ~]# /etc/init.d/keepalived stop
Stopping keepalived (via systemctl): [ OK ]
[root@master-node ~]# /etc/init.d/keepalived status
[root@master-node ~]# ps -ef|grep keepalived
root 26952 24348 0 17:49 pts/0 00:00:00 grep --color=auto keepalived
[root@master-node ~]#
```

4）然后在从服务器上查看，发现已经接管了VIP：

```
[root@slave-node ~]# ip addr
.......
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
link/ether 44:a8:42:17:3c:a5 brd ff:ff:ff:ff:ff:ff
inet 103.110.98.24/26 brd 103.10.86.63 scope global em1
inet 103.110.98.20/32 scope global em1
inet6 fe80::46a8:42ff:fe17:3ca5/64 scope link
valid_lft forever preferred_lft forever
.......
```

发现master的keepalived服务挂了后，vip资源自动漂移到slave上，并且网站正常访问，丝毫没有受到影响！

5）重新启动主服务器上的keepalived，发现主服务器又重新接管了VIP，此时slave机器上的VIP已经不在了

```
[root@master-node ~]# /etc/init.d/keepalived start
Starting keepalived (via systemctl): [ OK ]
[root@master-node ~]# ip addr
.......
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
link/ether 44:a8:42:17:3d:dd brd ff:ff:ff:ff:ff:ff
inet 103.110.98.14/26 brd 103.10.86.63 scope global em1
valid_lft forever preferred_lft forever
inet 103.110.98.20/32 scope global em1
valid_lft forever preferred_lft forever
inet 103.110.98.20/26 brd 103.10.86.63 scope global secondary em1:0
valid_lft forever preferred_lft forever
inet6 fe80::46a8:42ff:fe17:3ddd/64 scope link
valid_lft forever preferred_lft forever
......

[root@slave-node ~]# ip addr
.......
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
link/ether 44:a8:42:17:3c:a5 brd ff:ff:ff:ff:ff:ff
inet 103.110.98.24/26 brd 103.10.86.63 scope global em1
inet6 fe80::46a8:42ff:fe17:3ca5/64 scope link
valid_lft forever preferred_lft forever
```

接着验证下nginx服务故障，看看keepalived监控nginx状态的脚本是否正常?
如下：手动关闭master机器上的nginx服务，最多2秒钟后就会自动起来（因为keepalive监控nginx状态的脚本执行间隔时间为2秒）。域名访问几乎不受影响！

```
[root@master-node ~]# /usr/local/nginx/sbin/nginx -s stop
[root@master-node ~]# ps -ef|grep nginx
root 28401 24826 0 19:43 pts/1 00:00:00 grep --color=auto nginx
[root@master-node ~]# ps -ef|grep nginx
root 28871 28870 0 19:47 ? 00:00:00 /bin/sh /opt/chk_nginx.sh
root 28875 24826 0 19:47 pts/1 00:00:00 grep --color=auto nginx
[root@master-node ~]# ps -ef|grep nginx
root 28408 1 0 19:43 ? 00:00:00 nginx: master process
/usr/local/nginx/sbin/nginx
www 28410 28408 0 19:43 ? 00:00:00 nginx: worker process
www 28411 28408 0 19:43 ? 00:00:00 nginx: worker process
www 28412 28408 0 19:43 ? 00:00:00 nginx: worker process
www 28413 28408 0 19:43 ? 00:00:00 nginx: worker process
```

最后可以查看两台服务器上的/var/log/messages，观察VRRP日志信息的vip漂移情况~~~~

可能出现的问题

1）VIP绑定失败
原因可能有：
-> iptables开启后，没有开放允许VRRP协议通信的策略（也有可能导致脑裂）；可以选择关闭iptables
-> keepalived.conf文件配置有误导致，比如interface绑定的设备错误

2）VIP绑定后，外部ping不通
可能的原因是：
-> 网络故障，可以检查下网关是否正常；
-> 网关的arp缓存导致，可以进行arp更新，命令是"arping -I 网卡名 -c 5 -s VIP 网关"

十九、面试官：给我讲讲Nginx如何实现四层负载均衡?

这次又被问到Nginx四层负载均衡的问题了，别慌，我们一起来细细分析这个看似简单的问题。

负载均衡可以分为静态负载均衡和动态负载均衡，接下来，我们就一起来分析下Nginx如何实现四层静态负载均衡和四层动态负载均衡。

静态负载均衡

Nginx的四层静态负载均衡需要启用ngx_stream_core_module模块，默认情况下，ngx_stream_core_module是没有启用的，需要在安装Nginx时，添加--with-stream配置参数启用，如下所示。

```
./configure --prefix=/usr/local/nginx-1.17.2 --with-
openssl=/usr/local/src/openssl-1.0.2s --with-pcre=/usr/local/src/pcre-8.43 --
with-zlib=/usr/local/src/zlib-1.2.11 --with-http_realip_module --with-
http_stub_status_module --with-http_ssl_module --with-http_flv_module --with-
http_gzip_static_module --with-cc-opt=-O3 --with-stream  --with-http_ssl_module
```

配置四层负载均衡

配置HTTP负载均衡时，都是配置在http指令下，配置四层负载均衡，则是在stream指令下，结构如下所示.

```
stream {
    upstream mysql_backend {
        ......
    }
    server {
        ......
    }
}
```

配置upstream

```
upstream mysql_backend {
    server 192.168.175.201:3306 max_fails=2 fail_timeout=10s weight=1;
    server 192.168.175.202:3306 max_fails=2 fail_timeout=10s weight=1;
    least_conn;
}
```

配置server

```
server {
    #监听端口，默认使用的是tcp协议，如果需要UDP协议，则配置成listen 3307 udp;
    listen 3307;
    #失败重试
    proxy_next_upstream on;
    proxy_next_upstream_timeout 0;
    proxy_next_upstream_tries 0;
    #超时配置
    #配置与上游服务器连接超时时间，默认60s
    proxy_connect_timeout 1s;
    #配置与客户端上游服务器连接的两次成功读/写操作的超时时间，如果超时，将自动断开连接
    #即连接存活时间，通过它可以释放不活跃的连接，默认10分钟
    proxy_timeout 1m;
    #限速配置
    #从客户端读数据的速率，单位为每秒字节数，默认为0，不限速
    proxy_upload_rate 0;
    #从上游服务器读数据的速率，单位为每秒字节数，默认为0，不限速
    proxy_download_rate 0;
    #上游服务器
    proxy_pass mysql_backend;
}
```

配置完之后，就可以连接Nginx的3307端口，访问数据库了。

Nginx完整配置

完整的Nginx配置如下：

```
user  hadoop hadoop;
worker_processes  auto;

error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;
```

```
events {
    use epoll;
    worker_connections  1024;
}

stream {
    upstream mysql_backend {
        server 192.168.175.100:3306 max_fails=2 fail_timeout=10s weight=1;
        least_conn;
    }
    server {
        #监听端口，默认使用的是tcp协议，如果需要UDP协议，则配置成listen 3307 udp;
        listen 3307;
        #失败重试
        proxy_next_upstream on;
        proxy_next_upstream_timeout 0;
        proxy_next_upstream_tries 0;
        #超时配置
        #配置与上游服务器连接超时时间，默认60s
        proxy_connect_timeout 1s;
        #配置与客户端上游服务器连接的两次成功读/写操作的超时时间，如果超时，将自动断开连接
        #即连接存活时间，通过它可以释放不活跃的连接，默认10分钟
        proxy_timeout 1m;
        #限速配置
        #从客户端读数据的速率，单位为每秒字节数，默认为0，不限速
        proxy_upload_rate 0;
        #从上游服务器读数据的速率，单位为每秒字节数，默认为0，不限速
        proxy_download_rate 0;
        #上游服务器
        proxy_pass mysql_backend;
    }
}
```

动态负载均衡

配置Nginx四层静态负载均衡后，重启Nginx时，Worker进程一直不退出，会报错，如下所示。

```
nginx: worker process is shutting down;
```

这是因为Worker进程维持的长连接一直在使用，所以无法退出，只能杀掉进程。可以使用Nginx的四层动态负载均衡解决这个问题。

使用Nginx的四层动态负载均衡有两种方案：使用商业版的Nginx和使用开源的nginx-stream-upsync-module模块。
注意：四层动态负载均衡可以使用nginx-stream-upsync-module模块，七层动态负载均衡可以使用nginx-upsync-module模块。

使用如下命令为Nginx添加nginx-stream-upsync-module模块和nginx-upsync-module模块，此时，Nginx会同时支持四层动态负载均衡和HTTP七层动态负载均衡。

```
git clone https://github.com/xiaokai-wang/nginx-stream-upsync-module.git
git clone https://github.com/weibocom/nginx-upsync-module.git
git clone https://github.com/CallMeFoxie/nginx-upsync.git
cp -r nginx-stream-upsync-module/* nginx-upsync/nginx-stream-upsync-module/
cp -r nginx-upsync-module/* nginx-upsync/nginx-upsync-module/

./configure --prefix=/usr/local/nginx-1.17.2 --with-
openssl=/usr/local/src/openssl-1.0.2s --with-pcre=/usr/local/src/pcre-8.43 --
with-zlib=/usr/local/src/zlib-1.2.11 --with-http_realip_module --with-
http_stub_status_module --with-http_ssl_module --with-http_flv_module --with-
http_gzip_static_module --with-cc-opt=-O3 --with-stream --add-
module=/usr/local/src/nginx-upsync --with-http_ssl_module
```

配置四层负载均衡

配置HTTP负载均衡时，都是配置在http指令下，配置四层负载均衡，则是在stream指令下，结构如下所示，

```
stream {
    upstream mysql_backend {
        ......
    }
    server {
        ......
    }
}
```

配置upstream

```
upstream mysql_backend {
    server 127.0.0.1:1111;  #占位server
    upsync 192.168.175.100:8500/v1/kv/upstreams/mysql_backend upsync_timeout=6m
upsync_interval=500ms upsync_type=consul strong_dependency=off;
    upsync_dump_path /usr/local/nginx-1.17.2/conf/mysql_backend.conf;
}
```

- upsync指令指定从consul哪个路径拉取上游服务器配置；
- upsync_timeout配置从consul拉取上游服务器配置的超时时间；
- upsync_interval配置从consul拉取上游服务器配置的间隔时间；
- upsync_type指定使用consul配置服务器；
- strong_dependency配置nginx在启动时是否强制依赖配置服务器，如果配置为on，则拉取配置失败时Nginx启动同样失败。
- upsync_dump_path指定从consul拉取的上游服务器后持久化到的位置，这样即使consul服务器出现问题，本地还有一个备份。

配置server

```
server {
    #监听端口，默认使用的是tcp协议，如果需要UDP协议，则配置成listen 3307 udp;
    listen 3307;
    #失败重试
    proxy_next_upstream on;
    proxy_next_upstream_timeout 0;
    proxy_next_upstream_tries 0;
```

```
    #超时配置
    #配置与上游服务器连接超时时间，默认60s
    proxy_connect_timeout 1s;
    #配置与客户端上游服务器连接的两次成功读/写操作的超时时间，如果超时，将自动断开连接
    #即连接存活时间，通过它可以释放不活跃的连接，默认10分钟
    proxy_timeout 1m;
    #限速配置
    #从客户端读数据的速率，单位为每秒字节数，默认为0，不限速
    proxy_upload_rate 0;
    #从上游服务器读数据的速率，单位为每秒字节数，默认为0，不限速
    proxy_download_rate 0;
    #上游服务器
    proxy_pass mysql_backend;
}
```

从Consul添加上游服务器

```
curl -X PUT -d "{\"weight\":1, \"max_fails\":2, \"fail_timeout\":10}"
http://192.168.175.100:8500/v1/kv/upstreams/mysql_backend/192.168.175.201:3306
curl -X PUT -d "{\"weight\":1, \"max_fails\":2, \"fail_timeout\":10}"
http://192.168.175.100:8500/v1/kv/upstreams/mysql_backend/192.168.175.202:3306
```

从Consul删除上游服务器

```
curl -X DELETE
http://192.168.175.100:8500/v1/kv/upstreams/mysql_backend/192.168.175.202:3306
```

配置upstream_show

```
server {
    listen 13307;
    upstream_show;
}
```

配置upstream_show指令后，可以通过curl http://192.168.175.100:13307/upstream_show查看当前动态负载均衡上游服务器列表。

Nginx完整配置

Nginx的完整配置如下：

```
user  hadoop hadoop;
worker_processes  auto;

error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    use epoll;
    worker_connections  1024;
}
```

```
stream {
    upstream mysql_backend {
        server 127.0.0.1:1111;  #占位server
        upsync 192.168.175.100:8500/v1/kv/upstreams/mysql_backend
upsync_timeout=6m upsync_interval=500ms upsync_type=consul
strong_dependency=off;
        upsync_dump_path /usr/local/nginx-1.17.2/conf/mysql_backend.conf;
    }
    server {
        #监听端口，默认使用的是tcp协议，如果需要UDP协议，则配置成listen 3307 udp;
        listen 3307;
        #失败重试
        proxy_next_upstream on;
        proxy_next_upstream_timeout 0;
        proxy_next_upstream_tries 0;
        #超时配置
        #配置与上游服务器连接超时时间，默认60s
        proxy_connect_timeout 1s;
        #配置与客户端上游服务器连接的两次成功读/写操作的超时时间，如果超时，将自动断开连接
        #即连接存活时间，通过它可以释放不活跃的连接，默认10分钟
        proxy_timeout 1m;
        #限速配置
        #从客户端读数据的速率，单位为每秒字节数，默认为0，不限速
        proxy_upload_rate 0;
        #从上游服务器读数据的速率，单位为每秒字节数，默认为0，不限速
        proxy_download_rate 0;
        #上游服务器
        proxy_pass mysql_backend;
    }
    server {
        listen 13307;
        upstream_show;
    }
}
```

我的《海量数据处理与大数据技术实战》出版啦！

这本书是一本真正以实战案例为主线的大数据实战案例型书籍。这里，我给大家推荐几个购买链接。

京东购买链接：https://item.jd.com/10020420941243.html

当当购买链接：http://product.dangdang.com/29115124.html

重磅福利

微信搜一搜【冰河技术】微信公众号，关注这个有深度的程序员，每天阅读超硬核技术干货，公众号内回复【PDF】有我准备的一线大厂面试资料和我原创的超硬核PDF技术文档，以及我为大家精心准备的多套简历模板（不断更新中），希望大家都能找到心仪的工作，学习是一条时而郁郁寡欢，时而开怀大笑的路，加油。如果你通过努力成功进入到了心仪的公司，一定不要懈怠放松，职场成长和新技术学习一样，不进则退。如果有幸我们江湖再见！

另外，我开源的各个PDF，后续我都会持续更新和维护，感谢大家长期以来对冰河的支持！！

写在最后

> 如果你觉得冰河写的还不错，请微信搜索并关注「 **冰河技术** 」微信公众号，跟冰河学习高并发、分布式、微服务、大数据、互联网和云原生技术，「 **冰河技术** 」微信公众号更新了大量技术专题，每一篇技术文章干货满满！不少读者已经通过阅读「 **冰河技术** 」微信公众号文章，吊打面试官，成功跳槽到大厂；也有不少读者实现了技术上的飞跃，成为公司的技术骨干！如果你也想像他们一样提升自己的能力，实现技术能力的飞跃，进大厂，升职加薪，那就关注「 **冰河技术** 」微信公众号吧，每天更新超硬核技术干货，让你对如何提升技术能力不再迷茫！

打开“微信 / 发现 / 搜一搜”搜索