那些实用的Nginx规则

2018-01-15 09:37 | 阅读 3.4k | 评论 0

1. 概述

大家都知道Nginx有很多功能模块，比如反向代理、缓存等，这篇文章总结下我们这些年实际环境中那些有用的Nginx规则和模块，大部分是用法的概括及介绍，具体细节在实际配置时再自行google。

2. 内置语法

先介绍Nginx默认已支持的内置功能，靠这些基本就满足大部分的web服务需求。

2.1 proxy代理

proxy常用于两类应用场景，一类是中转，如异地科学的上网方式，另外一类是到后端服务的负载均衡方案。

用反向代理时候，需要特别注意里面的域名默认是在nginx启动时候就解析了，除非reload否则一直用的是当初解析的域名，也就是说不能动态解析。

但这个问题是可以通过别的模块或者用内置字典变量方式来解决。

```
resolver 114.114.114.114;
server {
    location / {
        set $servers github.com;
        proxy_pass http://$servers;
    }
}
```

2.1.1 中转

针对某个域名进行中转：

```
server {
listen 172.16.10.1:80;
    server_name pypi.python.org;
    location ~ /simple {
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_pass http://pypi.python.org;
    }
}
```

注意如果是前后端域名不一样的话需要处理proxy_redirect的301跳转之类的显示，否则在跳转时候会跳转到proxy_pass的域名。

另外可以直接代理所有80端口的http流量：

```
server {
    listen 80;
    server_name _;
    resolver 114.114.114.114;
    set $URL $host;
    location / {
        proxy_pass http://$URL;
    }
}
```

如果是想代理https的站点也不是不可能，只是需要自行处理CA证书导入即可，而且经过https中转的流量对nginx是透明的，也就是有证书的时候做窃听和劫持的情况。

2.1.2 负载均衡

这是代理的另外一个常见用法，通过upstream到多个后端，可以通过weight来调节权重或者backup关键词来指定备份用的后端，通常默认就可以 了，或者可以指定类似ip_hash这样的方式来均衡，配置很简单，先在http区域添加upstream定义：

```
upstream backend {
    ip_hash;
    server backend1.example.com weight=5;
    server backend2.example.com weight=5;;
}
```

然后在server里面添加proxy_pass：

```
location / {
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
}
```

做负载均衡的时候可以智能识别后端服务器状态，虽然可以智能地proxy_next_upstream到另外的后端，但还是会定期损失一些正常的“尝试性”的连接，比如过了max_fails 次尝试之后，休息fail_timeout时间，过了这个时间之后又会去尝试，这个时候可以使用第三方的upstream_check模块来在后台定期地自动探索，类似这样：

```
check interval=3000 rise=2 fall=5 timeout=2000 type=http;
```

这样替代用户正常的连接来进行尝试的方式进一步保障了高可用的特性。

还有就是在做前端代理的时候也是这样的方式，直接proxy_pass到后端即可，比如CDN的场景。

2.2 防盗链

普通的防盗链是通过referer来做，比如：

```
location ~* \.(gif|jpg|png|bmp)$ {
    valid_referers none blocked *.example.com server_names ~\.google\. ~\.baidu\.;
    if ($invalid_referer) {
        return 403;
    }
}
```

再精细一点的就是URL加密，针对一些用户IP之类的变量生成一个加密URL通常是针对文件下载时候用到，可以通过openresty来写lua脚本或者是accesskey之类的模块来实现。

2.3 变量

nginx里面支持正则匹配和变量配置，默认的变量比如remote_addr、request_filename、query_string、server_name之类的，这些组合在一起可以做很多规则，或者还有日志里面status、http_cookie等。

还有在进行多域名配置时候可以用通配符，比如：

```
server_name ~^(www\.)?(.+)$;
root /data/web/$2;
```

这样就实现了自动进行域名的目录指派。

变量方面，比如配置变量a=1：

```
set $a 1;
```

下面这个案例配合if判断来做有更大的用处。

2.4 if判断

nginx里面支持一些简单的if判断，但是没有多重逻辑的语法，多个判断条件用起来需要结合变量的方式来实现，比如允许ip地址为10.10.61段和和192.168.100段的用户访问，其余的拒绝，返回405状态码：

```
set $err 0;
    if ( $remote_addr ~ 10.10.61.){
        set $err 0;
    }
    if ( $remote_addr ~ 192.168.100.){
        set $err 0;
    }
    if ( $err = 1){
```

```
        return 405;
    }
```

这样通过一个err变量比较巧妙实现了需求。

2.5 error_page

有用到后端proxy的地方需要加上这句话才可以传到状态码到nginx:

```
fastcgi_intercept_errors on;
```

具体配置一般是配置到具体的错误URL页面，比如:

```
#返回具体状态码
error_page 404 403 /4xx.html
#返回200状态码
error_page 404 403 =200  /error.html
```

或者采用callback的方式统一做处理:

```
error_page 404 403 = @fallback;
location @fallback {
    proxy_pass http://backend;
    access_log /data/logs/404_error.log access;
}
```

这样在重定向时不会改变URL，然后把404页面直接返回。

2.6 rewrite

rewrite做一些301、302之类的跳转，同时也可以在CDN前端做“去问号”缓存的效果。

```
location /db.txt {
    rewrite (.*) $1? break;
    include proxy.conf;
}
```

另外最常见的跳转写法:

```
rewrite ^/game/(.*) /$1;
```

把/game/test跳转为/test的效果，注意这样是没有状态码的，如果访问正常就直接返回200状态码。

可以在后面加个permanent参数，就变为了301 Moved Permanently，或者添加redirect改为302跳转。

同理，还可以进行多个正则匹配进行URL重组，比如:

```
rewrite ^/download/(.*)/lastest/(.*)$ /file/$1?ver=$2 break;
```

2.7 日志字段

想针对每个连接进行日志留档，可以在nginx日志那里配置好字段，比如记录cookie之类的数据。

在log_format字段里面加入$http_cookie变量即可。

另外post的数据可以永久保留在文件里面，比如用来做http的日志备份，包括get和post的原始数据，把这个值开启即可：

```
client_body_in_file_only  on;
```

然后post的数据就会保存在nginx/client_body_temp文件夹里面。

2.8 internal关键词

这个关键词很少见，但有时候是很有用的，比如在有很多规则时候，突然需要针对某个目录转为nginx内部处理。

```
location ^~ /upload/down/ {
alias /data/web/dts/dtsfile/down/;
internal;
}
```

2.9 try_files

字面意思是尝试，后面可以接多个目录或者文件，比如kohana框架：

```
try_files $uri /index.php?$query_string;
```

先看是否有URL这个文件，没有的话再调用index.php来处理，或者支持状态码处理：

```
try_files /foo /bar/ =404;
```

没有这两个文件的话返回404状态。

2.10 auth认证

可以做简单的用户登录认证方式，其中的passwd_file得通过apache的htpasswd命令来生成。

```
auth_basic "Restricted";
auth_basic_user_file passwd_file;
```

认证通过之后每次访问会在头部添加Authorization字段包含用户名密码的base64加密密文给服务端。

2.11 gzip

普通的线上web站点gzip压缩是必须要开的，压缩一些文本类型的文件再返回给用户。

注意必须手动指定全需要压缩的类型，比如css、js之类的，线上配置如下：

```
gzip on;
gzip_min_length  2048;
gzip_buffers     4 16k;
gzip_vary   on;
gzip_http_version 1.1;
gzip_types  text/plain  text/css text/xml application/xml application/javascript application/x-javascript ;
```

2.12 mime配置

很久以前基本是忽略这个配置，但手游流行之后就发现异常了，需要让手机浏览器知道返回的apk后缀是什么类型，否则类似IE浏览器会以zip后缀返回，需要加上：

```
application/vnd.android.package-archive apk;
application/iphone pxl ipa;
```

2.13 限速

限速包括限制请求的并发数和请求的下载速度。

简单的限制某个线程的下载速度就直接加上一句话就可以了：

```
limit_rate 1024k;
```

要限制某个IP的并发数之类的就需要用ngx_http_limit_req_module和ngx_http_limit_conn_module模块了，不过是默认就编译好的。

比如使用一个 10M 大小的状态缓存区，针对每个IP每秒只接受20次的请求：

```
limit_req_zone $binary_remote_addr zone=NAME:10m rate=20r/s;
```

2.14 location匹配

location匹配有多种方式，常见的比如

```
location  = /
location  /
location ^~ /test{
```

是有优先级的，直接 ”=” 的优先级是最高的，一般就用”~”这个符号来匹配php就好了，不过是区分了大小写的：

```
location ~ .*\.php$
```

2.15 文件缓存

返回给用户的文件一般都配置了过期时间，让浏览器缓存起来。

比如缓存14天：

```
expires 14d;
```

针对某些特殊的文件就需要location匹配之后进行禁止缓存配置：

```
add_header Cache-Control no-cache;
add_header Cache-Control no-store;
expires off;
```

2.16 缓存文件

nginx可以作为ATS这样的缓存服务器来缓存文件，配置也比较简单，不过我们很少用，除非一些特殊的场合，参考配置：

```
#先在全局下面定义好缓存存放的目录
proxy_cache_path  /data/cache/ levels=1:2 keys_zone=cache_one:10m inactive=7d max_size=10g;
proxy_temp_path   /data/cache/proxy_temp_path;
proxy_cache_key   $host$uri$is_args$args;
#然后在server里面的location匹配好目的文件，加入下一段即可
proxy_cache cache_one;
proxy_cache_valid 200 304 24h;
proxy_cache_valid any 10m;
proxy_pass https://$host;
proxy_cache_key $host$uri$is_args$args;
add_header  Nginx-Cache "$upstream_cache_status"; 3. 内置模块
```

3. 内置模块

nginx含有大量的模块可以支持多种复杂的需求，比如源码目录src/http/modules里面就有很多c模块的代码，或者直接通过./configure –help|grep module来查看有哪些内置模块，编译时候直接加上就可以了。

除了nginx内置的模块，网络上还有很多第三方的模块，可以通过编译时候加参数–add-module=PATH指定模块源码来编译。

下面介绍一些我们线上用过而且比较赞的内置模块。

3.1 stream

端口转发的模块，从nginx1.9版本才开始支持，包含tcp和udp的支持，和IPTABLES相比这个虽然是应用层，会监听端口，但是配置起来很方便，比IPTABLES灵活，在tcp模块下面添加类似vhost的server就可以了，方便自动化管理，参考配置：

```
server {
    listen PORT;
    proxy_pass IP:PORT;
    access_log /data/logs/tcp/PORT.log;
}
```

3.2 http_realip_module

nginx反向代理之后，如何让后端web直接获取到的IP不是反向代理的iP，而是直接获取到用户的真实IP呢，就需要这个模块了，不需要代码那里再做类似X-Real-IP的变量特殊判断。

3.3 http_slice_module

在做CDN时候可以用到，让一个大文件分片，分成多个小文件通过206断点续传到后端，然后再组合起来，避免大文件直接回源导致多副本和多次回源的问题。

3.4 http_secure_link_module

前面说到的防盗链可以用这个来做，但是这个一般是针对那种文件下载时候用到的，比如从网页下载时候，服务端生成一个加密URL给用户，然后这个URL有过期时间之类的，避免此URL被多次分享出去，不过普通的素材加载还是用普通的防盗链即可。

3.5 http_sub_module

替换响应给用户的内容，相对于sed之后再返回，比如可以在需要临时全局修改网站背景或者title时候可以一次性处理好。

4. 扩展项目

简单介绍下大名鼎鼎的两个基于nginx的扩展项目，也是我们线上有很多地方用到的。

4.1 openresty

集成lua脚本，几乎可以完成任何普通web相关的需求。

比如URL加密进行防劫持和防盗链，服务端动态生成一串aes加密的URL给CDN，CDN的openresty解密之后用普通的URL转发到后端，然后再返回给用户正确的内容。

4.2 tengine

淘宝的nginx修改版，实现了很多nginx的收费功能或者是特殊功能，比如动态加载、concat合并请求，动态解析等。

我们python开发的后台基本都是用的这个版本，主要是利用了concat的合并素材的功能。

5. 结语

Nginx是个非常实用软件，部分功能已经超越了普通的web服务定位，同时它具备开源、轻量、自动化等特性，能有效解决实际工作中很多特殊场景的需求，祝Nginx在全球的份额持续攀升~

原文来自微信公众号：运维军团