

nginx 基础介绍

1. nginx简述与安装
2. nginx 命令
3. nginx本地文件解析
4. nginx的应用场景
5. location部分
6. rewrite部分

nginx简述与安装

nginx 是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务。

homebrew是macOS系统的软件包的管理器，可以用它来安装nginx：

附上homebrew的官网：<https://brew.sh/index>

首先安装homebrew：

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

成功后安装nginx，终端执行：

```
brew install nginx
```

Homebrew 会将软件包安装到独立目录，并将其文件软链接至 /usr/local 。

nginx安装文件目录：/usr/local/Cellar/nginx

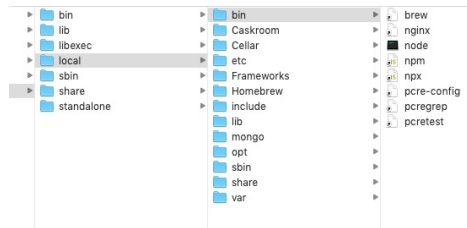
nginx配置文件目录：/usr/local/etc/nginx

服务器默认路径：/usr/local/var/www

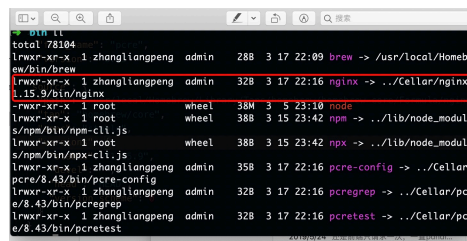
当我们敲下nginx命令时，实际上是执行了一个脚本，我们可以用which命令是查找命令是否存在，以及命令的存放位置在哪儿

```
➜ ~ which nginx
/usr/local/bin/nginx
```

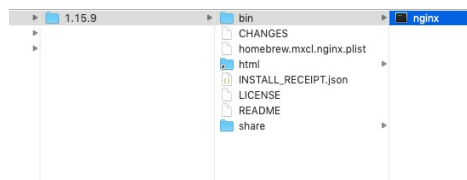
Mac os 系统（基于Unix系统）一般的应用都会放在/usr/local文件夹下面，/usr文件夹一般是对用户隐藏的，可以通过命令访问。



这里可以看到nginx是指向的一个软连接，最后执行的文件是/usr/local/Cellar/nginx/1.15.9/bin/nginx（可以通过ll命令查看文件的软连接信息）



简单说下软连接：软连接类似window的快捷方式，它是可以跨磁盘块，目的是为了复用模块，系统中有很多地方都用到软连接。我们可以看到最后执行的脚本文件位于/usr/local/Cellar/nginx/1.15.9/bin/nginx



引申：软连接跟硬连接的区别：<https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html>

nginx命令

常用nginx命令（管理员权限加sudo）：

nginx #打开 nginx

nginx -t #测试配置文件是否有语法错误

nginx -s reopen #重启Nginx

`nginx -s reload` #重新加载Nginx配置文件，然后以优雅的方式重启Nginx

`nginx -s stop` #强制停止Nginx服务

`nginx -s quit` #优雅地停止Nginx服务（即处理完所有请求后再停止服务）

`nginx -c` 配置文件地址 #设置配置文件

我们执行nginx重启命令有时候会遇到以下错误：

```
nginx: [error] open() "/usr/local/var/run/nginx.pid" failed (2: No such file or directory)
```

字面大概意思是没有nginx.pid文件，进到/usr/local/var/run/目录发现确实没有这个文件，大家都知道一般解决办法都是用 `sudo nginx -c /usr/local/etc/nginx/nginx.conf`，那为什么执行这个命令就有这个文件了呢？

可以知道 `nginx -c` 命令是设置配置文件，正常运行之后我们可以执行 `cat /usr/local/var/run/nginx.pid` 查看该文件的内容，发现内容只有一行数字。

```
Password:
➔ cat /usr/local/var/run/nginx.pid
7504
```

这个数字其实是该进程的id，这个文件的作用是为了防止启动多个进程副本

我们可以用 `ps -ef | grep nginx` 查看nginx的进程信息：

```
➔ ps -ef | grep nginx
0 7504 1 0 8:43下午 ?? 0:00.00 nginx: master process nginx -
c /usr/local/etc/nginx/nginx.conf
-2 7505 7504 0 8:43下午 ?? 0:00.00 nginx: worker process
501 7520 797 0 8:45下午 ttys000 0:00.00 grep --color=auto --exclude-d
ir=.bzr --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.sv
n nginx
```

可以看到主进程的id跟上面文件内容是一样的，这个时候可能会产生疑问，为什么会有多个id？

nginx遵循Master-Worker设计模式，是以多进程的方式来工作的，nginx在启动后，会有一个master进程和多个worker进程，master进程主要用来管理worker进程（可以用 `kill -QUIT 主进程号` 等方法杀死进程）。

可以得出结论：当主进程存在时，nginx.pid文件就会存在，内容为主进程id，当进程关掉时nginx.pid文件也就自动删除了，所以需要我们去指定配置文件。

Nginx本地文件解析

//定义nginx运行的用户（用户涉及到文件的权限）

```
#user nobody;
```

//nginx进程数，可以用ps -ef|grep nginx查看进程

```
worker_processes 1;
```

//全局错误日志定义类型，hombrew放在/usr/local/var/log/nginx/error.log

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

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

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

//进程文件

```
#pid logs/nginx.pid;
```

//events模块来用指定nginx的工作模式及连接数上限

```
events {
```

// 单个进程最大链接数（即接受前端请求的链接数）

```
worker_connections 1024;
```

```

}

//设定http服务器 ()

http { //这个是协议级别

    //文件扩展名与文件类型映射表

    include      mime.types;

    //默认文件类型

    default_type  application/octet-stream;

    //日志格式定义，变量见下面定义

    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '

        # '$status $body_bytes_sent "$http_referer" '

        # '"$http_user_agent" "$http_x_forwarded_for"';

    //定义访问日志目录，hombrew放在/usr/local/var/log/nginx/access.log

    # access_log  logs/access.log  main;

    //开启高效文件传输模式

    sendfile      on;

```

//集中发包，提高效率，sendfile on 情况下才可以打开

```
#tcp_nopush      on;
```

//长链接超时时间

```
keepalive_timeout 65;
```

//开始gzip压缩，服务器压缩，浏览器解压

```
#gzip on;
```

//单个虚拟主机的配置

```
server { //这个是服务器级别
```

//监听的端口

```
listen          8080;
```

//监听的服务域名，可以有多个，用逗号隔开

```
server_name localhost;
```

//默认编码

```
#charset koi8-r;
```

//该虚拟主机日志的存放位置

```
#access_log logs/host.access.log main;
```

```

//对应的路由展示

location / { //这个是请求级别

    //文件目录

    root    html;

    index   index.html index.htm;

}

//错误页面

error_page   500 502 503 504   /50x.html;

location = /50x.html {

    //相对的路径存放目录

    root    html;

}

}

// 增加配置可以include其他配置文件

include configurable.conf;

}

```

访问 nginx的error.log文件：

```
tail /usr/local/var/log/nginx/error.log （默认查最后十行）
```

访问nginx的access.log文件：

```
tail /usr/local/var/log/nginx/access.log （默认查最后十行）
```

nginx内置变量：

<code>\$uri</code>	请求的URI，可能会经过重定向导致跟最初的值有不同
<code>\$http_user_agent</code>	客户端信息
<code>\$args</code>	请求参数；
<code>\$body_bytes_sent</code>	已发送的消息体字节数
<code>\$content_length</code>	header头信息里的"Content-Length"
<code>\$content_type</code>	header头信息里的"Content-Type"
<code>\$document_root</code>	针对当前请求的根路径设置值
<code>\$document_uri</code>	与\$uri相同
<code>\$host</code>	请求信息中的"Host"，没有Host行，则等于设置的服务器名；
<code>\$http_cookie</code>	cookie 信息
<code>\$http_referer</code>	来源地址
<code>\$http_via</code>	最后一个访问服务器的Ip地址

<code>\$http_x_forwarded_for</code>	相当于网络访问路径。
<code>\$limit_rate</code>	对连接速率的限制
<code>\$remote_addr</code>	客户端地址
<code>\$remote_port</code>	客户端端口号
<code>\$remote_user</code>	客户端用户名，认证用
<code>\$request</code>	用户请求信息
<code>\$request_body</code>	用户请求主体
<code>\$request_body_file</code>	发往后端的本地文件名称
<code>\$request_filename</code>	当前请求的文件路径名
<code>\$request_method</code>	请求的方法，比如"GET"、"POST"等
<code>\$request_uri</code>	请求的URI，带参数
<code>\$server_addr</code>	服务器地址
<code>\$server_name</code>	请求到达的服务器名
<code>\$server_port</code>	请求到达的服务器端口号
<code>\$server_protocol</code>	请求的协议版本，"HTTP/1.0"或"HTTP/1.1"

nginx的应用场景

1. 静态资源web服务器
2. 代理服务器
3. 负载均衡

静态资源服务器

nginx采用的是异步非阻塞的通信机制(epoll模型),支持更大的并发连接.所谓的epoll模型:当事件没有准备好时,就放入epoll(队列)里面。如果有事件准备好了,那么就去处理;实现由进程循环处理多个准备好的事件,从而实现高并发和轻量级。

预先定义好本地的静态资源: (后面的例子也会用到这些)

/usr/local/test-img/follow.png,

/usr/local/test-img/403.png,

/usr/local/test-html/forward.html,

/usr/local/test-html/taobao/forward.html,

/usr/local/test-html/taobao/taobao.html,

/usr/local/test-html/taobaowang/taobao.html,

/usr/local/test-html/upstream/1.html,

/usr/local/test-html/upstream/2.html,

/usr/local/test-html/upstream/3.html,

/usr/local/test-html/upstream/4.html

Gzip压缩

静态资源就会涉及到Gzip压缩问题:

`syntax:gzip on | off`

```
default:gzip off
```

```
context:http, server, if in location
```

配置语法:

```
//打开或者关闭gzip压缩的功能
```

```
gzip on;
```

```
// 最小压缩长度, 被压缩的内容超过这个长度才会被压缩, 否则直接输出
```

```
gzip_min_length 1024;
```

```
// 压缩级别, 分为1-9
```

```
gzip_comp_level 2;
```

```
// 列出来的内容类型才会被压缩, 其他类型的内容不会被压缩, 类型指的是MIME类型
```

```
gzip_types text/plain application/x-javascript text/css application/xml text/
javascript image/jpeg image/gif image/png;
```

```
// 会在响应头增加vary:Accept-Encoding, 代表已经进行服务端压缩
```

```
gzip_vary on
```

```
//设置nginx 服务器是否对后端返回的结果进行gzip压缩, 反向代理的时候有效
```

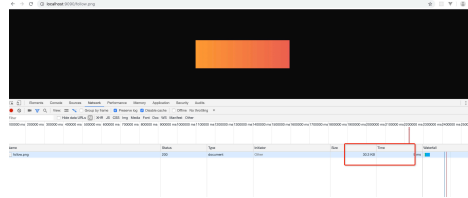
```
gzip_proxine
```

```
// 存放静态资源的文件路径
```

```
root /usr/local/test-img;
```

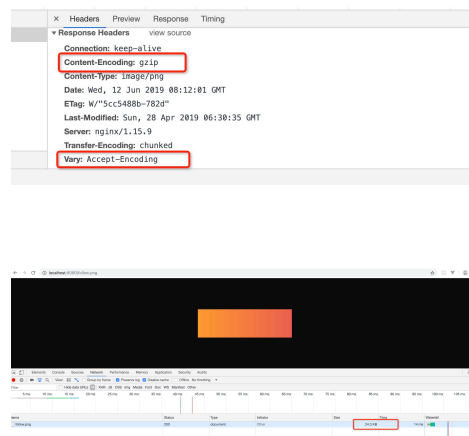
在浏览器访问：<http://localhost:9090/follow.png> 验证：

开启压缩前：



开启压缩后：

看到这个代表已经开启压缩：



可以看到文件体积已经变小了

防盗链

首先说一下盗链，举个例子：别人把你网站上的图片链接放到自己的网站上，这样在访问别人的网站时，实际上在调用你网站上的图片，还要用你服务器的流量带宽

防盗链是基于验证referer来实现的，referer表示一个网站的请求来源，伪装referer头部是非常简单的事情，所以这个模块只能用于阻止大部分非法请求.我们应该知道，有些合法的请求是不会带referer来源头部的,所以有时候不要拒绝来源头部（referer）为空的请求。比如直接在浏览器的地址栏中输入一个资源的URL地址，那么这种请求是不会包含 Referer 字段的。

一般配置：

```
valid_referers none blocked server_names \.baidu\.;

if ($invalid_referer) {

    #return 403; // 返回403

    rewrite ^/ http://127.0.0.1:7000/403.png; // 链接到403图片

}
```

nginx防盗链指令：

```
syntax: valid_referers none | blocked | server_names | string...;

default: -

context:server, location
```

参数解释：

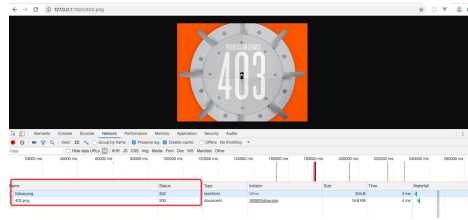
none：表示来源头部为空的情况

blocked：表示来源头部不为空，但是里面的值被代理或者防火墙删除了，这些值都不以http:// 或者https:// 开头。

server_names：表示来源头部包含当前的server_names

string：任意字符串,定义服务器名或者可选的URI前缀.主机名可以使用 * 开头或者结尾，在检测来源头部这个过程中，来源域名中的主机端口将会被忽略掉

正则表达式：~ 表示排除https://或http://开头的字符串.



代理服务器

前提：本次是在一台服务器上做验证，用不同的端口来模拟不同服务器之间的交互。

代理分为正向代理跟反向代理；正向代理是为客户端做代理，代替客户端去访问服务器，而反向代理是为服务器做代理，代替服务器接受客户端请求。

两者的区别可以自行百度一下

前端常用的代理是反向代理，下面讲解下反向代理：

反向代理是指以代理服务器来接受网络上的连接请求，然后将请求转发给内部网络上的服务器，把数据返回给客户端，此时代理服务器对外就表现为一个源服务器。

Nginx 反向代理的指令不需要新增额外的模块，默认自带 `proxy_pass` 指令，只需要修改配置文件就可以实现反向代理。

```
location / {
```

```
// 处理跨域请求
```

```
add_header Access-Control-Allow-Origin *;
```

```
// 请求头支持的传递字段
```

```
add_header Access-Control-Allow-Headers "Origin, Content-Type";
```

```
//涉及预检请求，服务器需要允许该方法
```

```

add_header Access-Control-Allow-Methods "OPTIONS";

// 代理网路请求到本地3000端口

proxy_pass http://localhost:3000;

// 重写主机名，防止后端真实的服务器设置有类似防盗链或者根据http请求头中的host字段来进行路由
或判断功能

proxy_set_header Host $host;

// 重写服务器ip ，防止后端有防攻击策略的话，机器会被封掉

proxy_set_header X-Forwarded-For $remote_addr

// 请求端真实的IP

proxy_add_x_forwarded_for: client ;

}

```

Upstream模块实现负载均衡

负载均衡的作用：实现在不同地域的服务器间的流量调配，保证使用最佳的服务器服务离自己最近的客户，从而确保访问质量

在http层面下添加upstream节点:

```

upstream clusters {

    server 127.0.0.1:9001;
}

```

```
server 127.0.0.1:9002;

server 127.0.0.1:9003;

server 127.0.0.1:9004;

}
```

本地添加静态资源服务作为被请求服务器

请求服务器配置：

```
server {

    listen      9005;

    server_name localhost;

    location / {

        proxy_pass http://clusters;

    }

}
```

用 `curl http://localhost:9005` 或者在浏览器请求去验证负载均衡是否起作用。

可以看到每次的请求都被均匀的分配到不同的服务器

Upstream可以为每个服务单独设置状态值：

`down`：表示当前server暂时不参与负载

`backup`： 预留的备份服务器,压力最小

`max_fails`： 允许请求失败的次数

fail_timeout : 经过max_fails失败后, 服务暂停的时间

max_conns: 限制最大的接收的连接数

每个服务的调度算法讲解:

轮询: 按时间顺序逐一分配到不同的后端服务器

weight: 默认为1 weight越大, 匹配的机会越多

```
upstream clusters {  
  
    server 127.0.0.1:9001; // 访问比率: 20%  
  
    server 127.0.0.1:9002; //访问比率: 20%  
  
    server 127.0.0.1:9003 weight=2; //访问比率: 40%  
  
    server 127.0.0.1:9004 weight=1; //访问比率: 20%  
  
}
```

ip_hash: 每个请求按访问ip的hash结果分配, 这样来自同一个ip的固定访问一个后端服务器, 可以解决服务端的用户session问题

```
upstream clusters {  
  
    ip_hash;  
  
    server 127.0.0.1:9001;  
  
    server 127.0.0.1:9002;  
  
    server 127.0.0.1:9003;
```

```
server 127.0.0.1:9004;

}
```

url_hash: 按照访问的url的hash结果来分配请求, 是每个url定向到同一个后端服务器, 可以解决缓存失效问题

```
upstream clusters {

    // $request_uri是nginx内部抛出的变量, 指的是除了域名的部分

    hash $request_uri;
```

```
server 127.0.0.1:9001;

server 127.0.0.1:9002;

server 127.0.0.1:9003;

server 127.0.0.1:9004;

}
```

least_conn : 最少链接数, 那个机器连接数少就分发

用 curl http://localhost:9005或者在浏览器请求去验证这些参数的作用。

location部分

1. 书写匹配location规则的时候会有一些纠结加不加/的问题, 下面讨论下匹配url加/与不加/的区别; 转发请求路径(也就是proxy_pass 后面路径)加/与不加/的区别。

匹配url加不加/的区别:

预先在127.0.0.1:9006机器上定义好了静态资源:

```
/usr/local/test-html/taobao/taobao.html
```

```
/usr/local/test-html/taobaowang/taobao.html
```

我们先定义请求路径（本地资源）：

```
http://localhost:9007/taobao/taobao.html,
```

```
http://localhost:9007/taobaowang/taobao.html
```

先看下面加/的配置:

```
location /taobao/ {  
  
    proxy_pass http://127.0.0.1:9006;  
  
}
```

分别请求上面两个路径（可在浏览器端也可以用下面的命令）：

```
curl http://localhost:9007/taobao/taobao.html
```

```
<!DOCTYPE html>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>  
<title>nginx静态资源服务器</title>  
</head>  
<body>  
<div>你看到的是taobao资源</div>  
</body>  
</html>
```

```
curl http://localhost:9007/taobaowang/taobao.html
```

```
<html>  
<head><title>404 Not Found</title></head>  
<body>  
<center><h1>404 Not Found</h1></center>  
<h2><center>nginx/1.15.9</center>  
</body>  
</html>
```

再来看一下不加/的配置:

```
location /taobao {

    proxy_pass http://127.0.0.1:9006;

}
```

分别请求上面两个路径（可在浏览器端也可以用下面的命令）：

```
curl http://localhost:9007/taobao/taobao.html
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
    <div>你看到的是taobao资源</div>
</body>
</html>
```

```
curl http://localhost:9007/taobaowang/taobao.html
```

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
    <div>你看到的是taobaowang资源</div>
</body>
</html>
```

通过比较：**加/**只能匹配到 /usr/local/test-html/taobao/taobao.html资源，而/usr/local/test-html/taobaowang/taobao.html资源匹配不到；**不加/**两个资源都能得到。

可以得出结论：由于location进行的是模糊匹配，所以对于加/的这种情况只能匹配像/**taobao/any**这种url，不加/的情况可以匹配/**taobao[any]**这种url

转发的请求路径加不加/的区别：

预先在127.0.0.1:9006机器上定义好了静态资源：

```
/usr/local/test-html/taobao/forward.html
```

```
/usr/local/test-html/forward.html
```

我们先定义请求路径为：`http://localhost:9007/taobao/forward.html`

先看下面加/的配置：

```
location /taobao/ {
```

```
proxy_pass http://127.0.0.1:9006/;
```

```
}
```

请求定义路径（可在浏览器端也可以用下面的命令）：

```
curl http://localhost:9007/taobao/forward.html
```

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
  <div>匹配路径为 /usr/local/test-html/forward.html</div>
</body>
</html>
```

再来看下不加/的配置：

```
location /taobao/ {
```

```
proxy_pass http://127.0.0.1:9006;
```

```
}
```

请求定义路径（可在浏览器端也可以用下面的命令）：

```
curl http://localhost:9007/taobao/forward.html
```

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
  <div>匹配路径为 /usr/local/test-html/taobao/forward.html</div>
</body>
</html>
```

通过比较：**加/**访问的资源是/usr/local/test-html/forward.html,

不加/访问的资源是/usr/local/test-html/taobao/forward.html

通过比较可以得出结论：**加/**的话相当于绝对路径，不会把location中匹配的url代理走，**不加/**的话会把匹配的路径部分也给代理走

2.实际项目中每个虚拟主机中会有多个location配置，那这样就会涉及到匹配location的顺序问题：

```
location [=|~|~*|^~|@ ] /url/ {config}
```

= 表示精确匹配

~ 表示正则匹配，区分大小写

~*表示正则匹配，不区分大小写

^~表示不匹配正则

@表示internally redirected（内部重定向，表示forward）

首先分类下：分为普通的location跟正则location

正则location: ~ |~*

一般location: = | ^~|@

下面验证优先级：

1.两个普通的location配置：

```
location /taobao/ {

    root /usr/local/test-html;

    allow all;

}

location /taobao/taobao.html {

    root /usr/local/test-html;

    deny all;
```

```
}
```

执行`curl http://localhost:9008/taobao/taobao.html`，得到：

```
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.15.9</center>
</body>
</html>
```

现在把下面的location注释：

```
location /taobao/ {
```

```
    root /usr/local/test-html;
```

```
    allow all;
```

```
}
```

```
#location /taobao/taobao.html {
```

```
    #root /usr/local/test-html;
```

```
    #deny all;
```

```
#}
```

执行`curl http://localhost:9008/taobao/taobao.html`，得到：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
<div>你看到的是taobao资源</div>
</body>
</html>
```

结论：普通location之间的顺序规则是有个最大匹配原则，越精确优先级越高

2.正则location配置:

```
location ~ /\.html$ {
```

```
    root /usr/local/test-html;
```

```
    allow all;
```

```
}
```

```
location ~ /taobao.html {
```

```
    root /usr/local/test-html;
```

```
    deny all;
```

```
}
```

执行curl http://localhost:9008/taobao/taobao.html, 得到:

```
<!DOCTYPE html>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
    <div>你看到的是taobao资源</div>
</body>
</html>
```

现在把两个配置换个位置:

```
location ~ /taobao.html {
```

```
    root /usr/local/test-html;
```

```
    deny all;
```

```
}
```



```
location ~ /\.html$ {

    root /usr/local/test-html;

    allow all;

}
```

执行curl http://localhost:9008/taobao/taobao.html, 得到:

```
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.15.9</center>
</body>
</html>
```

结论：正则location之间会有顺序，写在前面的会被优先匹配到

3.优先级高的普通location跟正则location配置：

```
location /taobao/taobao.html {

    root /usr/local/test-html;

    deny all;

}
```

```
location ~ /\.html$ {

    root /usr/local/test-html;

    allow all;

}
```

执行curl http://localhost:9008/taobao/taobao.html, 得到:

```

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
<div>你看到的是taobao资源</div>
</body>
</html>

```

结论：正则location优先级要高于普通的location

4.精确location跟正则location的配置

```
location = /taobao/taobao.html {
```

```
    root /usr/local/test-html;
```

```
    deny all;
```

```
}
```

```
location ~ /\.html$ {
```

```
    root /usr/local/test-html;
```

```
    allow all;
```

```
}
```

执行curl http://localhost:9008/taobao/taobao.html, 得到:

```

<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.15.9</center>
</body>
</html>

```

结论：精确location优先级要高于正则location

5.特殊情况^~ location跟正则location还有普通location的配置

```
location /taobao/taobao.html {
```

```
root /usr/local/test-html;
```

```
deny all;
```

```
}
```

```
location ^~ /taobao/ {
```

```
root /usr/local/test-html;
```

```
deny all;
```

```
}
```

```
location ~ /\.html$ {
```

```
root /usr/local/test-html;
```

```
allow all;
```

```
}
```

执行`curl http://localhost:9008/taobao/taobao.html`，得到：

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nginx静态资源服务器</title>
</head>
<body>
  <div>你看到的是taobao页面</div>
</body>
</html>
```

可以看到起作用的是最后一个配置，好像`^~` 没有起作用

我们把配置改变一下：

```
#location /taobao/taobao.html {
```

```
#root /usr/local/test-html;
```

```
#deny all;
```

```
#}
```

```
location ^~ /taobao/ {
```

```
root /usr/local/test-html;
```

```
deny all;
```

```
}
```

```
location ~ /\.html$ {
```

```
root /usr/local/test-html;
```

```
allow all;
```

```
}
```

注释第一个配置，执行`curl http://localhost:9008/taobao/taobao.html`，得到：

```
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.15.9</center>
</body>
</html>
```

可以看到第二个配置已经起作用`^~`，代表不匹配正则表达式

那为什么现在起作用呢？根据上面的配置我们把普通location中的优先级高的配置注释掉了。那是不是这个优先级高的配置把`^~`的配置覆盖掉了呢？

我们现在再把配置改变一下：

```
location ^~ /taobao/taobao.html {
```

```
    root /usr/local/test-html;
```

```
    deny all;
```

```
}
```

```
location /taobao/ {
```

```
    root /usr/local/test-html;
```

```
    deny all;
```

```
}
```

```
location ~ /\.html$ {
```

```
    root /usr/local/test-html;
```

```
    allow all;
```

```
}
```

执行`curl http://localhost:9008/taobao/taobao.html`，得到：

```
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.15.9</center>
</body>
</html>
```

现在可以看到这个普通location优先级高的配置把`^~`覆盖掉了，所以这个`^~`也属于普通location。

结论：`^~`属于普通的location，遵循普通location的规则，如果被覆盖，后面还有正则location的话，则正则location优先级更高

总体结论：精确匹配 (=) > 正则location（有顺序限制，只匹配第一个） > 普通location（最大匹配原则），这里有个特殊情况是遇到~，在不被覆盖的情况下，不匹配后面的正则location。

rewrite模块

背景：新建本地服务器，端口为9009

先说下rewrite的指令：

break

if

return

rewrite

set

break:停止执行该模块的指令集

if: 根据条件决定是否执行语句

return: 返回一个状态值给客户端

rewrite: 根据表达式来更改url

set:可以设置一个变量

if指令：

syntax: if (condition);

default:-

```
context:server, location,if
```

验证条件逻辑:

表达式只是一个变量时, 值为""或任何以0开头的字符串都会当做false

直接时使用=或!=, 跟js有区别

正则表达式匹配, ~区分大小写, ~*不区分大小写的匹配, !~, !~表示不匹配

' ' -f和!-f用来检测一个文件是否存在

-d和!-d用来检测一个目录是否存在

-e和!-e用来检测是否存在一个文件, 一个目录或者一个符号链接

-x和!-x用来检测一个文件是否可执行

举个例子: (见以下配置)

```
if ($http_user_agent ~ 10_14_3) {  
  
    return 401;  
  
}
```

在浏览器访问http://localhost:9009/, 得到:



*rewrite*指令:

syntax: rewrite regex replacement [flag];

default:-

`context:server, location,if`

regex: 正则表达式

replacement: 新的url

flag: 包含这几个值: last, break, redirect, permanent

last:停止处理rewrite模块的指令集, 并根据replacement继续匹配location

break: 停止处理rewrite模块的指令集

redirect:返回302临时重定向

permanent:返回301永久重定向

last跟break的区别:

配置:

```
location / {
```

```
    rewrite ^/code/ /test last;
```

```
    return 403;
```

```
}
```

```
location /test {
```

```
    return 500;
```

```
}
```

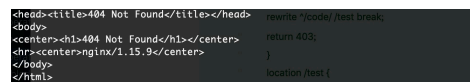
执行`curl http://localhost:9009/code/*` 得到:

```
<html>
<head><title>500 Internal Server Error</title></head>
<body>
<center><h1>500 Internal Server Error</h1></center>
<hr><center>nginx/1.15.3</center>
</body>
</html>
```


现在更改下配置：

```
location / {  
  
    rewrite ^/code/ /test break;  
  
    return 403;  
  
}  
  
location /test {  
  
    return 500;  
  
}
```

执行curl http://localhost:9009/code/* 得到：



```
<head><title>404 Not Found</title></head>    rewrite ^/code/ /test break;  
<body>    return 403;  
<center><h1>404 Not Found</h1></center>    }  
<hr><center>nginx/1.15.9</center>    location /test {  
</body>  
</html>
```

没有这两条指令时：

```
location / {  
  
    rewrite ^/code/ /test;  
  
    return 403;  
  
}  
  
location /test {
```

```
return 500;
```

```
}
```

执行 `curl http://localhost:9009/code/*` 得到:

```
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.15.9</center>
</body>
</html>
```

结论：last跟break都能停止rewrite模块的指令集，但是last会继续匹配，break就地终止。

另外说下请求参数的问题：

下面看个例子：

```
location / {
```

```
rewrite /code /testparams permanent;
```

```
}
```

浏览器请求：`http://localhost:9009/code?a=1`将会看到浏览器地址被重定向到

`http://localhost:9009/testparams?a=1`，旧参数被添加到新的url上了

我们下面来改一下配置：

```
location / {
```

```
rewrite /code /testparams? permanent;
```

```
}
```

浏览器请求：`http://localhost:9009/code?a=1`将会看到浏览器地址被重定向到

`http://localhost:9009/testparams`，旧参数被省略掉了

结论：默认情况下旧的url请求的参数会放在新替换的url 后面，如果想省略旧的请求参数在新的url后面加上?就好了。

Nginx中文网地址: <http://www.nginx.cn/doc/index.html>