[nginx配置location总结及rewrite规则写法](https://segmentfault.com/a/1190000002797606)

* [nginx](https://segmentfault.com/t/nginx/blogs)

[**seanlook**](https://segmentfault.com/u/seanlook) 2015年05月26日发布

* 推荐 **24** 推荐
* 收藏 **258** 收藏，71.1k 浏览

location正则写法

一个示例：

location = / {

# 精确匹配 / ，主机名后面不能带任何字符串

[ configuration A ]

}

location / {

# 因为所有的地址都以 / 开头，所以这条规则将匹配到所有请求

# 但是正则和最长字符串会优先匹配

[ configuration B ]

}

location /documents/ {

# 匹配任何以 /documents/ 开头的地址，匹配符合以后，还要继续往下搜索

# 只有后面的正则表达式没有匹配到时，这一条才会采用这一条

[ configuration C ]

}

location ~ /documents/Abc {

# 匹配任何以 /documents/ 开头的地址，匹配符合以后，还要继续往下搜索

# 只有后面的正则表达式没有匹配到时，这一条才会采用这一条

[ configuration CC ]

}

location ^~ /images/ {

# 匹配任何以 /images/ 开头的地址，匹配符合以后，停止往下搜索正则，采用这一条。

[ configuration D ]

}

location ~\* \.(gif|jpg|jpeg)$ {

# 匹配所有以 gif,jpg或jpeg 结尾的请求

# 然而，所有请求 /images/ 下的图片会被 config D 处理，因为 ^~ 到达不了这一条正则

[ configuration E ]

}

location /images/ {

# 字符匹配到 /images/，继续往下，会发现 ^~ 存在

[ configuration F ]

}

location /images/abc {

# 最长字符匹配到 /images/abc，继续往下，会发现 ^~ 存在

# F与G的放置顺序是没有关系的

[ configuration G ]

}

location ~ /images/abc/ {

# 只有去掉 config D 才有效：先最长匹配 config G 开头的地址，继续往下搜索，匹配到这一条正则，采用

[ configuration H ]

}

location ~\* /js/.\*/\.js

* 已=开头表示精确匹配  
  如 A 中只匹配根目录结尾的请求，后面不能带任何字符串。
* ^~ 开头表示uri以某个常规字符串开头，不是正则匹配
* ~ 开头表示区分大小写的正则匹配;
* ~\* 开头表示不区分大小写的正则匹配
* / 通用匹配, 如果没有其它匹配,任何请求都会匹配到

顺序 no优先级： (location =) > (location 完整路径) > (location ^~ 路径) > (location ~,~\* 正则顺序) > (location 部分起始路径) > (/)

上面的匹配结果 按照上面的location写法，以下的匹配示例成立：

* / -> config A  
  精确完全匹配，即使/index.html也匹配不了
* /downloads/download.html -> config B  
  匹配B以后，往下没有任何匹配，采用B
* /images/1.gif -> configuration D  
  匹配到F，往下匹配到D，停止往下
* /images/abc/def -> config D  
  最长匹配到G，往下匹配D，停止往下  
  你可以看到 任何以/images/开头的都会匹配到D并停止，FG写在这里是没有任何意义的，H是永远轮不到的，这里只是为了说明匹配顺序
* /documents/document.html -> config C  
  匹配到C，往下没有任何匹配，采用C
* /documents/1.jpg -> configuration E  
  匹配到C，往下正则匹配到E
* /documents/Abc.jpg -> config CC  
  最长匹配到C，往下正则顺序匹配到CC，不会往下到E

实际使用建议

所以实际使用中，个人觉得至少有三个匹配规则定义，如下：

**#直接匹配网站根，通过域名访问网站首页比较频繁，使用这个会加速处理，官网如是说。**

**#这里是直接转发给后端应用服务器了，也可以是一个静态首页**

**# 第一个必选规则**

location = / {

proxy\_pass http://tomcat:8080/index

}

**# 第二个必选规则是处理静态文件请求，这是nginx作为http服务器的强项**

**# 有两种配置模式，目录匹配或后缀匹配,任选其一或搭配使用**

location ^~ /**static**/ {

root /webroot/**static**/;

}

location ~\* \.(gif|jpg|jpeg|png|css|js|ico)$ {

root /webroot/res/;

}

**#第三个规则就是通用规则，用来转发动态请求到后端应用服务器**

**#非静态文件请求就默认是动态请求，自己根据实际把握**

**#毕竟目前的一些框架的流行，带.php,.jsp后缀的情况很少了**

location / {

proxy\_pass http://tomcat:8080/

}

<http://tengine.taobao.org/book/chapter_02.html>  
<http://nginx.org/en/docs/http/ngx_http_rewrite_module.html>

Rewrite规则

rewrite功能就是，使用nginx提供的全局变量或自己设置的变量，结合正则表达式和标志位实现url重写以及重定向。rewrite只能放在server{},location{},if{}中，并且只能对域名后边的除去传递的参数外的字符串起作用，例如 http://seanlook.com/a/we/index.php?id=1&u=str 只对/a/we/index.php重写。语法rewrite regex replacement [flag];

如果相对域名或参数字符串起作用，可以使用全局变量匹配，也可以使用proxy\_pass反向代理。

表明看rewrite和location功能有点像，都能实现跳转，主要区别在于rewrite是在同一域名内更改获取资源的路径，而location是对一类路径做控制访问或反向代理，可以proxy\_pass到其他机器。很多情况下rewrite也会写在location里，它们的执行顺序是：

1. 执行server块的rewrite指令
2. 执行location匹配
3. 执行选定的location中的rewrite指令

如果其中某步URI被重写，则重新循环执行1-3，直到找到真实存在的文件；循环超过10次，则返回500 Internal Server Error错误。

flag标志位

* last : 相当于Apache的[L]标记，表示完成rewrite
* break : 停止执行当前虚拟主机的后续rewrite指令集
* redirect : 返回302临时重定向，地址栏会显示跳转后的地址
* permanent : 返回301永久重定向，地址栏会显示跳转后的地址

因为301和302不能简单的只返回状态码，还必须有重定向的URL，这就是return指令无法返回301,302的原因了。这里 last 和 break 区别有点难以理解：

1. last一般写在server和if中，而break一般使用在location中
2. last不终止*重写后*的url匹配，即新的url会再从server走一遍匹配流程，而break终止重写后的匹配
3. break和last都能组织继续执行后面的rewrite指令

if指令与全局变量

**if判断指令**  
语法为if(condition){...}，对给定的条件condition进行判断。如果为真，大括号内的rewrite指令将被执行，if条件(conditon)可以是如下任何内容：

* 当表达式只是一个变量时，如果值为空或任何以0开头的字符串都会当做false
* 直接比较变量和内容时，使用=或!=
* ~正则表达式匹配，~\*不区分大小写的匹配，!~区分大小写的不匹配

-f和!-f用来判断是否存在文件  
-d和!-d用来判断是否存在目录  
-e和!-e用来判断是否存在文件或目录  
-x和!-x用来判断文件是否可执行

例如：

**if** ($http\_user\_agent ~ MSIE) {

rewrite ^(.\*)$ /msie/$1 break;

} //如果UA包含"MSIE"，rewrite请求到/msid/目录下

**if** ($http\_cookie ~\* "id=([^;]+)(?:;|$)") {

set $id $1;

} //如果cookie匹配正则，设置变量$id等于正则引用部分

**if** ($request\_method = POST) {

return 405;

} //如果提交方法为POST，则返回状态405（Method not allowed）。return不能返回301,302

**if** ($slow) {

limit\_rate 10k;

} //限速，$slow可以通过 set 指令设置

**if** (!-f $request\_filename){

break;

proxy\_pass http://127.0.0.1;

} //如果请求的文件名不存在，则反向代理到localhost 。这里的break也是停止rewrite检查

**if** ($args ~ post=140){

rewrite ^ http://example.com/ permanent;

} //如果query string中包含"post=140"，永久重定向到example.com

location ~\* \.(gif|jpg|png|swf|flv)$ {

valid\_referers none blocked www.jefflei.com www.leizhenfang.com;

**if** ($invalid\_referer) {

return 404;

} //防盗链

}

**全局变量**  
下面是可以用作if判断的全局变量

* $args ： #这个变量等于请求行中的参数，同$query\_string
* $content\_length ： 请求头中的Content-length字段。
* $content\_type ： 请求头中的Content-Type字段。
* $document\_root ： 当前请求在root指令中指定的值。
* $host ： 请求主机头字段，否则为服务器名称。
* $http\_user\_agent ： 客户端agent信息
* $http\_cookie ： 客户端cookie信息
* $limit\_rate ： 这个变量可以限制连接速率。
* $request\_method ： 客户端请求的动作，通常为GET或POST。
* $remote\_addr ： 客户端的IP地址。
* $remote\_port ： 客户端的端口。
* $remote\_user ： 已经经过Auth Basic Module验证的用户名。
* $request\_filename ： 当前请求的文件路径，由root或alias指令与URI请求生成。
* $scheme ： HTTP方法（如http，https）。
* $server\_protocol ： 请求使用的协议，通常是HTTP/1.0或HTTP/1.1。
* $server\_addr ： 服务器地址，在完成一次系统调用后可以确定这个值。
* $server\_name ： 服务器名称。
* $server\_port ： 请求到达服务器的端口号。
* $request\_uri ： 包含请求参数的原始URI，不包含主机名，如：”/foo/bar.php?arg=baz”。
* $uri ： 不带请求参数的当前URI，$uri不包含主机名，如”/foo/bar.html”。
* $document\_uri ： 与$uri相同。

例：http://localhost:88/test1/test2/test.php  
$host：localhost  
$server\_port：88  
$request\_uri：<http://localhost:88/test1/test2/test.php>  
$document\_uri：/test1/test2/test.php  
$document\_root：/var/www/html  
$request\_filename：/var/www/html/test1/test2/test.php

常用正则

* . ： 匹配除换行符以外的任意字符
* ? ： 重复0次或1次
* + ： 重复1次或更多次
* \* ： 重复0次或更多次
* \d ：匹配数字
* ^ ： 匹配字符串的开始
* $ ： 匹配字符串的介绍
* {n} ： 重复n次
* {n,} ： 重复n次或更多次
* [c] ： 匹配单个字符c
* [a-z] ： 匹配a-z小写字母的任意一个

小括号()之间匹配的内容，可以在后面通过$1来引用，$2表示的是前面第二个()里的内容。正则里面容易让人困惑的是\转义特殊字符。

rewrite实例

*例1*：

**http** {

# 定义image日志格式

**log\_format** imagelog '[$time\_local] ' $image\_file ' ' $image\_type ' ' $body\_bytes\_sent ' ' $status;

# 开启重写日志

**rewrite\_log** on;

**server** {

**root** /home/www;

**location** / {

# 重写规则信息

**error\_log** logs/rewrite.log notice;

# 注意这里要用‘’单引号引起来，避免{}

**rewrite** '^/images/([a-z]{2})/([a-z0-9]{5})/(.\*)\.(png|jpg|gif)$' /data?file=$3.$4;

# 注意不能在上面这条规则后面加上“last”参数，否则下面的set指令不会执行

**set** $image\_file $3;

**set** $image\_type $4;

}

**location** /data {

# 指定针对图片的日志格式，来分析图片类型和大小

**access\_log** logs/images.log mian;

**root** /data/images;

# 应用前面定义的变量。判断首先文件在不在，不在再判断目录在不在，如果还不在就跳转到最后一个url里

**try\_files** /$arg\_file /image404.html;

}

**location** = /image404.html {

# 图片不存在返回特定的信息

**return** 404 "image not found\n";

}

}

对形如/images/ef/uh7b3/test.png的请求，重写到/data?file=test.png，于是匹配到location /data，先看/data/images/test.png文件存不存在，如果存在则正常响应，如果不存在则重写tryfiles到新的image404 location，直接返回404状态码。

*例2*：

**rewrite** ^/images/(.\*)\_(\d+)x(\d+)\.(png|jpg|gif)$ /resizer/$1.$4?width=$2&height=$3? last;

对形如/images/bla\_500x400.jpg的文件请求，重写到/resizer/bla.jpg?width=500&height=400地址，并会继续尝试匹配location。

*例3*：  
见 [ssl部分页面加密](http://seanlook.com/2015/05/28/nginx-ssl) 。