# 第10单元 Nginx 转发规则

# 授课重点

1. 掌握Nginx与Rewrite规则
2. 掌握Nginx连接数优化参数

# 考核要求

1. 掌握Nginx与Rewrite规则
2. 掌握Nginx连接数优化参数

# 教学内容

## 10.1 Nginx与Rewrite规则

### 10.1.1 顺序 no优先级

Nginx配置请求转发location及rewrite规则,我们先来一个示例:

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

location ~ ^/(images|img|javascript|js|css|flash|media|static)/ {

root /home/resources/; #####静态资源的路

access\_log off; expires 30d; #####设置缓存时间 }

location ~\*\.(do)$ {

proxy\_pass http://192.168.126.168:8080;

}

示例配置解释:

* 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/  
}

### 10.1.2 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。

## 10.2 掌握Nginx连接数优化参数

nginx作为高性能web服务器，即使不特意调整配置参数也可以处理大量的并发请求。 以下的配置参数是借鉴网上的一些调优参数，仅作为参考，不见得适于你的线上业务。

**worker进程**

* **worker\_processes**

该参数表示启动几个工作进程，建议和本机CPU核数保持一致，每一核CPU处理一个进程。

* **worker*rlimit*nofile**

它表示Nginx最大可用的文件描述符个数，需要配合系统的最大描述符，建议设置为102400。 还需要在系统里执行ulimit -n 102400才可以。 也可以直接修改配置文件/etc/security/limits.conf修改 增加： \* soft nofile 655350 \* hard nofile 655350

* **worker\_connections**

该参数用来配置每个Nginx worker进程最大处理的连接数，这个参数也决定了该Nginx服务器最多能处理多少客户端请求 （worker*processes \* worker*connections)，建议把该参数设置为10240，不建议太大。

**http和tcp连接**

* use epoll

使用epoll模式的事件驱动模型，该模型为Linux系统下最优方式。

* multi\_accept on

使每个worker进程可以同时处理多个客户端请求。

* sendfile on

使用内核的FD文件传输功能，可以减少user mode和kernel mode的切换，从而提升服务器性能。

* tcp\_nopush on

当tcp*nopush设置为on时，会调用tcp*cork方法进行数据传输。 使用该方法会产生这样的效果：当应用程序产生数据时，内核不会立马封装包，而是当数据量积累到一定量时才会封装，然后传输。

* tcp\_nodelay on

不缓存data-sends（关闭 Nagle 算法），这个能够提高高频发送小数据报文的实时性。 (关于Nagle算法) 【假如需要频繁的发送一些小包数据，比如说1个字节，以IPv4为例的话，则每个包都要附带40字节的头， 也就是说，总计41个字节的数据里，其中只有1个字节是我们需要的数据。 为了解决这个问题，出现了Nagle算法。 它规定：如果包的大小满足MSS，那么可以立即发送，否则数据会被放到缓冲区，等到已经发送的包被确认了之后才能继续发送。 通过这样的规定，可以降低网络里小包的数量，从而提升网络性能。】

* keepalive\_timeout

定义长连接的超时时间，建议30s，太短或者太长都不一定合适，当然，最好是根据业务自身的情况来动态地调整该参数。

* keepalive\_requests

定义当客户端和服务端处于长连接的情况下，每个客户端最多可以请求多少次，可以设置很大，比如50000.

* reset*timeout*connection on

设置为on的话，当客户端不再向服务端发送请求时，允许服务端关闭该连接。

* client*body*timeout

客户端如果在该指定时间内没有加载完body数据，则断开连接，单位是秒，默认60，可以设置为10。

* send\_timeout

这个超时时间是发送响应的超时时间，即Nginx服务器向客户端发送了数据包，但客户端一直没有去接收这个数据包。 如果某个连接超过send\_timeout定义的超时时间，那么Nginx将会关闭这个连接。单位是秒，可以设置为3。

**buffer和cache(以下配置都是针对单个请求)**

* client*body*buffer\_size

当客户端以POST方法提交一些数据到服务端时，会先写入到client*body*buffer中，如果buffer写满会写到临时文件里，建议调整为128k。

* client*max*body\_size

浏览器在发送含有较大HTTP body的请求时，其头部会有一个Content-Length字段，client*max*body*size是用来限制Content-Length所示值的大小的。 这个限制body的配置不用等Nginx接收完所有的HTTP包体，就可以告诉用户请求过大不被接受。会返回413状态码。 例如，用户试图上传一个1GB的文件，Nginx在收完包头后，发现Content-Length超过client*max*body*size定义的值， 就直接发送413(Request Entity Too Large)响应给客户端。 将该数值设置为0，则禁用限制，建议设置为10m。

* client*header*buffer\_size

设置客户端header的buffer大小，建议4k。

* large*client*header\_buffers

对于比较大的header（超过client*header*buffer\_size）将会使用该部分buffer，两个数值，第一个是个数，第二个是每个buffer的大小。 建议设置为4 8k

* open*file*cache

该参数会对以下信息进行缓存： 打开文件描述符的文件大小和修改时间信息; 存在的目录信息; 搜索文件的错误信息（文件不存在无权限读取等信息）。 格式：open*file*cache max=size inactive=time; max设定缓存文件的数量，inactive设定经过多长时间文件没被请求后删除缓存。 建议设置 open*file*cache max=102400 inactive=20s;

* open*file*cache\_valid

指多长时间检查一次缓存的有效信息。建议设置为30s。

* open*file*cache*min*uses

open*file*cache指令中的inactive参数时间内文件的最少使用次数， 如,将该参数设置为1，则表示，如果文件在inactive时间内一次都没被使用，它将被移除。 建议设置为2。

**压缩**

对于纯文本的内容，Nginx是可以使用gzip压缩的。使用压缩技术可以减少对带宽的消耗。 由ngx*http*gzip\_module模块支持

**配置如下：**

gzip on; //开启gzip功能  
　　gzip\_min\_length 1024; //设置请求资源超过该数值才进行压缩，单位字节  
　　gzip\_buffers 16 8k; //设置压缩使用的buffer大小，第一个数字为数量，第二个为每个buffer的大小  
　　gzip\_comp\_level 6; //设置压缩级别，范围1-9,9压缩级别最高，也最耗费CPU资源  
　　gzip\_types text/plain application/x-javascript text/css application/xml image/jpeg image/gif image/png; //指定哪些类型的文件需要压缩  
　　gzip\_disable "MSIE 6\."; //IE6浏览器不启用压缩

测试：  
curl -I -H "Accept-Encoding: gzip, deflate" http://www.aminglinux.com/1.css

**日志**

* 错误日志级别调高，比如crit级别，尽量少记录无关紧要的日志。
* 对于访问日志，如果不要求记录日志，可以关闭，
* 静态资源的访问日志关闭

**静态文件过期**

对于静态文件，需要设置一个过期时间，这样可以让这些资源缓存到客户端浏览器， 在缓存未失效前，客户端不再向服务期请求相同的资源，从而节省带宽和资源消耗。

**配置示例如下：**

　location ~\* ^.+\.(gif|jpg|png|css|js)$   
　　{  
　　expires 1d; //1d表示1天，也可以用24h表示一天。  
　　}

**作为代理服务器**

Nginx绝大多数情况下都是作为代理或者负载均衡的角色。  
　　因为其他文章已经介绍过以下参数的含义，在这里只提供对应的配置参数：  
　　http  
　　{  
 　　 proxy\_cache\_path /data/nginx\_cache/ levels=1:2 keys\_zone=my\_zone:10m inactive=300s max\_size=5g;  
 　　 ...;  
 　　 server  
 　　{  
 　　 proxy\_buffering on;  
 　　 proxy\_buffer\_size 4k;  
 　　 proxy\_buffers 2 4k;  
 　　 proxy\_busy\_buffers\_size 4k;  
 　　 proxy\_temp\_path /tmp/nginx\_proxy\_tmp 1 2;  
 　　 proxy\_max\_temp\_file\_size 20M;  
 　　 proxy\_temp\_file\_write\_size 8k;  
   
 　　location /  
 　　{  
 　　 proxy\_cache my\_zone;  
 　　 ...;  
 　　 }  
 　　 }  
　　}

**SSL优化**

* 适当减少worker\_processes数量，因为ssl功能需要使用CPU的计算。
* 使用长连接，因为每次建立ssl会话，都会耗费一定的资源（加密、解密）
* 开启ssl缓存，简化服务端和客户端的“握手”过程。

ssl\_session\_cache shared:SSL:10m; //缓存为10M  
　　ssl\_session\_timeout 10m; //会话超时时间为10分钟