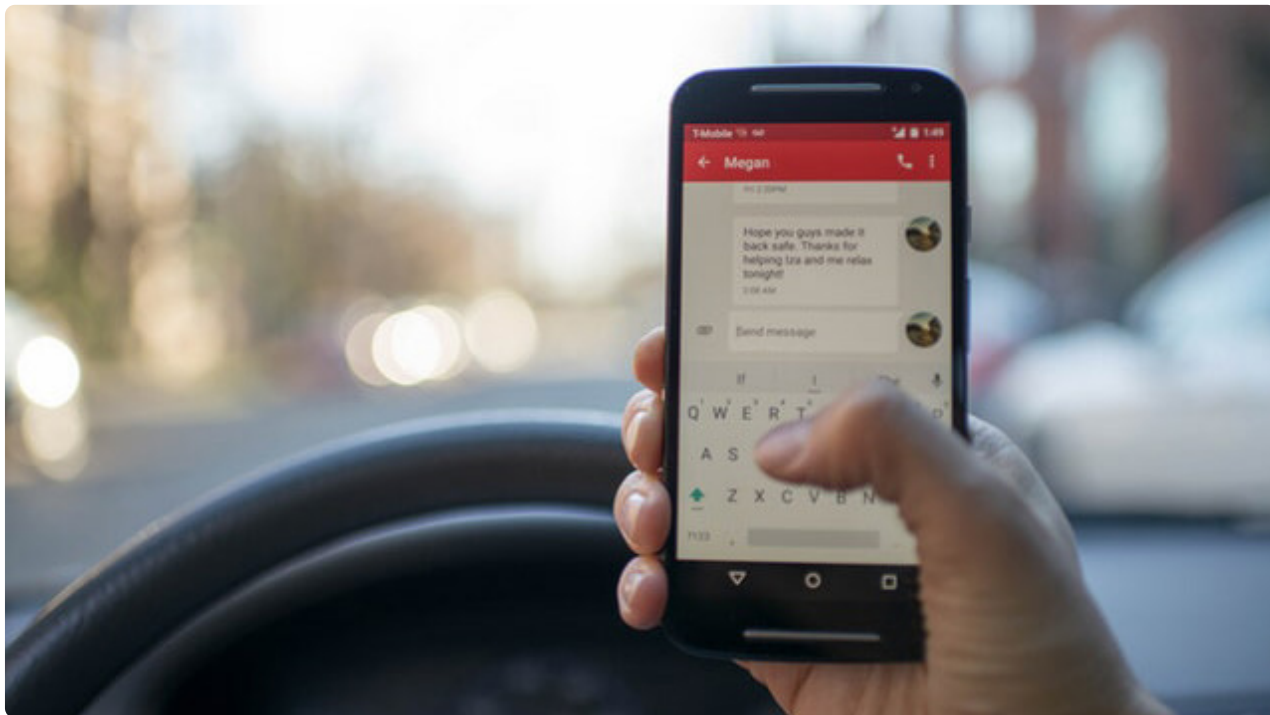


12 深入浅出 Rewrite 语法

更新时间：2020-01-02 16:56:46



“

人生的旅途，前途很远，也很暗。然而不要怕，不怕的人的面前才有路。—— 鲁 迅

”

前言

几乎所有的网站都会进行个性化配置满足自己的业务需要，而 **URL** 重写几乎是每个网站都必做的事情，**Nginx** 提供了一把瑞士军刀—**Rewrite** 模块来实现这个功能。

Rewrite 是由 **Nginx** 的 **ngx_http_rewrite_module** 模块实现的功能。这个模块实现了一个脚本引擎，提供了一个脚本编程的功能。我们可以通过一些简单的 **编程** 来实现特殊化需求。

Rewrite 模块指令

我们先看一下 **Rewrite** 模块的概览，它包含了下图中的几个指令：



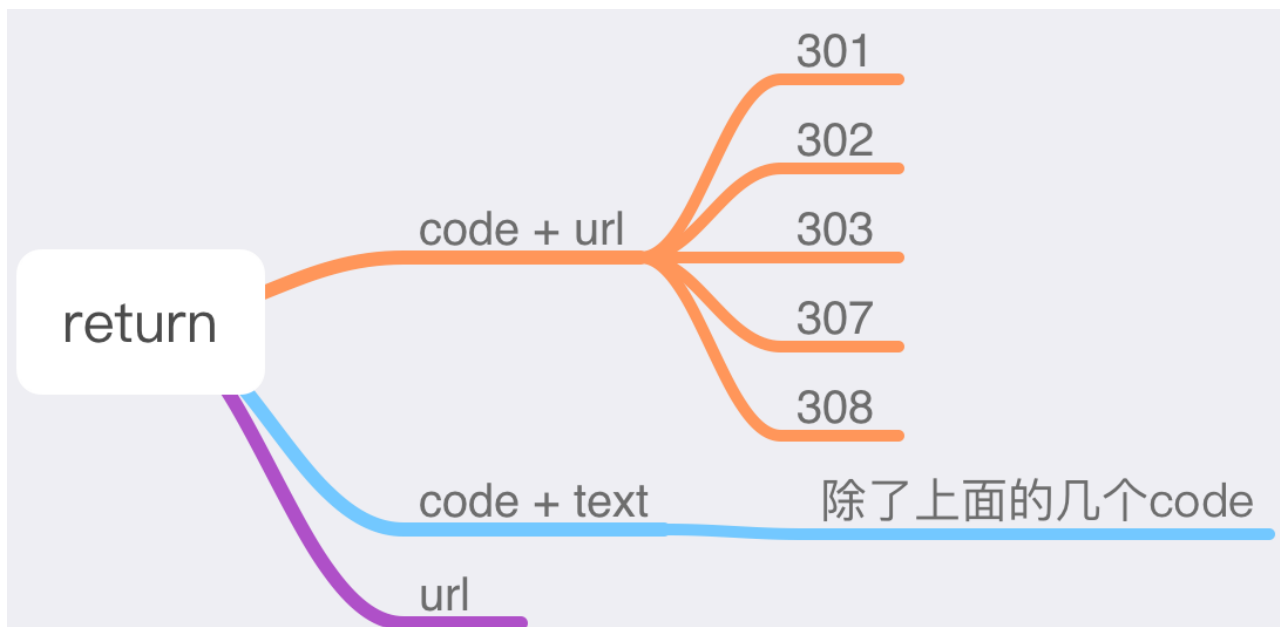
Rewrite 日志

在学习 **Rewrite** 模块的时候，最好的方法是打开 **Rewrite** 日志开关，从日志中看整个流程。

rewrite_log 指令就是控制是否打开日志开关的，打开之后就会把每次 **rewrite** 的步骤记录到 **error_log** 日志中。

return 命令

和其他的编程语言类似，**return** 就是直接返回，停止后续的处理直接返回。



我们看一个例子：

```
http {
    include      mime.types;
    default_type application/octet-stream;

    server {
        listen      80;
        server_name localhost;

        location /book/ {
            return 200 "hello world";
        }
    }
}
```

重启 `Nginx` 之后，访问 `/book/` 接口，如下：

```
[root@7135c374c767 conf]# curl -i http://localhost/book/
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Thu, 02 Jan 2020 06:25:57 GMT
Content-Type: application/octet-stream
Content-Length: 11
Connection: keep-alive

hello world[root@7135c374c767 conf]#
```

rewrite 指令

其实本节中我重点想要讲解的是 `rewrite` 指令，这个指令在平时的工作中经常的用到，但是很多人对该指令的理解都是片面的。

先看一下 `rewrite` 的语法格式：

```
rewrite regex replacement [flag];
```



`rewrite` 指令是使用指定的正则表达式（上一节我们介绍过的哦~）`regex` 来匹配当前请求的 `URI`，如果匹配成功，则使用 `replacement` 更改 `URI`。

`rewrite` 指令按照它们在配置文件中出现的顺序执行(下面我们总结了 `rewrite` 的执行顺序)，可以使用 `flag` 参数来控制指令的进一步处理。如果替换字符串 `replacement` 以 `http://`，`https://` 或 `$scheme` 开头，则停止处理后续内容，并直接重定向返回给客户端。

我们先从整体上看一下 `rewrite` 的执行顺序，过程如下：

1. 按照 `rewrite` 指令在 `server` 中出现的顺序逐个执行，确定 **最终** 的 `URI`。
2. 循环执行下面的步骤：（上一步确定了 `URI`）
 - 根据 `URI` 找到匹配的 `location`
 - 按 `rewrite` 指令在 `location` 中出现的顺序逐个执行
 - 如果上一步中对 `URI` 进行了重写，那么重复执行第 2 步骤，最多重复执行 10 次。

上面的 **最终** 指的是在 `server` 级别的 `rewrite` 全部按照要求执行之后得到的 `URI`，`Nginx` 会使用这个 `URI` 查找 `Location`。

可能大家会觉得上面的步骤很生硬，请不要着急，我会在下面一一展开讲解。

`rewrite` 指令最让人难理解原因是 `flag` 参数，特别是 `break` 和 `last` 这两个参数。

- **last**: 停止处理当前的 `ngx_http_rewrite_module` 的指令集，并开始搜索与更改后的 `URI` 相匹配的 `location`。（这里的 **last** 是 **持续，继续** 的意思，而不是 **最后**）
- **break**: 停止处理当前的 `ngx_http_rewrite_module` 指令集

上面是 `Nginx` 官网上的解释，其实这里面有一个比较难以理解的地方，什么是 **当前** 的 `ngx_http_rewrite_module` 的指令集？大家如果知道了这个概念，其实这两个参数的区别就挺好理解的了。

在文章的开始我们说过，`ngx_http_rewrite_module` 模块实现了一个脚本引擎，脚本引擎执行的过程需要上下文，所有在同一个 `server` 级别的 `ngx_http_rewrite_module` 模块的指令共享同一个上下文。所有在同一个 `location` 级别的 `ngx_http_rewrite_module` 模块的指令共享同一个上下文。所以上面的 **当前** 指的就是同一个上下文。`ngx_http_rewrite_module` 模块的指令集就是本文开头说的那几个。

来几个例子看一下，配置文件如下：

```
server {
    rewrite_log on;
    listen      80;
    server_name localhost;

    location /first {
        rewrite /first/(.*) /second/$1 last;
        return 200 "hello first\n";
    }

    location /second {
        rewrite /second/(.*) /third/$1 ;
        return 200 "hello second\n";
    }

    location /third {
        return 200 "hello third\n";
    }
}
```

目录结构如下：

```
[root@7135c374c767 conf]# mkdir -p /usr/local/nginx/html/first /usr/local/nginx/html/second /usr/local/nginx/html/third
[root@7135c374c767 conf]# cd /usr/local/nginx/html/
[root@7135c374c767 html]# echo "1.txt" > first/1.txt
[root@7135c374c767 html]# echo "2.txt" > second/2.txt
[root@7135c374c767 html]# echo "3.txt" > third/3.txt
[root@7135c374c767 html]# tree
.
|-- first
|   '-- 1.txt
|-- second
|   '-- 2.txt
|-- third
|   '-- 3.txt
3 directories, 3 files
```

重启nginx之后，我们请求如下接口：

```
[root@7135c374c767 conf]#
[root@7135c374c767 conf]# curl http://localhost/first/2.txt -i
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Thu, 02 Jan 2020 08:21:04 GMT
Content-Type: text/plain
Content-Length: 13
Connection: keep-alive

hello second
[root@7135c374c767 conf]#
```

可以看到，返回的值是 **hello second**，为什么会这样的？

```
2020/01/02 08:21:04 [notice] 131#0: *10 "/first/(.*)" matches "/first/2.txt", client: 127.0.0.1, server: localhost, request: "GET /first/2.txt HTTP/1.1", host: "localhost"
2020/01/02 08:21:04 [notice] 131#0: *10 rewritten data: "/second/2.txt", args: "", client: 127.0.0.1, server: localhost, request: "GET /first/2.txt HTTP/1.1", host: "localhost"
2020/01/02 08:21:04 [notice] 131#0: *10 "/second/(.*)" matches "/second/2.txt", client: 127.0.0.1, server: localhost, request: "GET /first/2.txt HTTP/1.1", host: "localhost"
2020/01/02 08:21:04 [notice] 131#0: *10 rewritten data: "/third/2.txt", args: "", client: 127.0.0.1, server: localhost, request: "GET /first/2.txt HTTP/1.1", host: "localhost"
```

我们看上面的 **rewrite_log** 日志，发生了两次 **rewrite**，最终匹配到了 **location /second**，虽然在 **location /second** 中被重写为了 **/third/2.txt**，但是由于 **rewrite** 后面没有 **break** 指令，所以会继续执行下面的 **return** 指令，最终返回 **hello second**。

现在我们给 `location /second` 加上 `break`，如下：

```
location /first {
    rewrite /first/(.*) /second/$1 last;
    return 200 "hello first\n";
}

location /second {
    rewrite /second/(.*) /third/$1 break;
    return 200 "hello second\n";
}

location /third {
    return 200 "hello third\n";
}
```

请求结果：

```
[root@7135c374c767 conf]# curl http://localhost/first/3.txt -i
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Thu, 02 Jan 2020 08:28:00 GMT
Content-Type: text/plain
Content-Length: 6
Last-Modified: Thu, 02 Jan 2020 08:10:55 GMT
Connection: keep-alive
ETag: "5e0da58f-6"
Accept-Ranges: bytes

3.txt
```

`rewrite log` 日志如下：

```
2020/01/02 08:28:00 [notice] 143#0: *12 "/first/(.*)" matches "/first/3.txt", client: 127.0.0.1, server: localhost, request: "GET /first/3.txt HTTP/1.1", host: "localhost"
2020/01/02 08:28:00 [notice] 143#0: *12 rewritten data: "/second/3.txt", args: "", client: 127.0.0.1, server: localhost, request: "GET /first/3.txt HTTP/1.1", host: "localhost"
2020/01/02 08:28:00 [notice] 143#0: *12 "/second/(.*)" matches "/second/3.txt", client: 127.0.0.1, server: localhost, request: "GET /first/3.txt HTTP/1.1", host: "localhost"
2020/01/02 08:28:00 [notice] 143#0: *12 rewritten data: "/third/3.txt", args: "", client: 127.0.0.1, server: localhost, request: "GET /first/3.txt HTTP/1.1", host: "localhost"
```

和第一次请求相比，我们这里多了一个 `break`，所以没有执行后面的 `return` 指令，而是请求了 `/third/3.txt`。

上面是对 `rewrite` 的一些讲解，大家可以多动手实践一下，通过日志可以清除的了解每一步的执行顺序。

if 指令

`if` 指令其实很简单，和各种编程语言中的 `if` 作用相同，都是进行一些条件的判断，如果条件为真则执行 `if` 块的内容，如果为假则不执行。

总结

这边文件是根据本人在学习 `Nginx` 过程中所遇到的问题总结而来，我觉得只要大家能够清晰的理解 `rewrite` 指令的使用，那么足以搞定绝大多数的情形。

}

