1、location [=|~|~*|^~] /uri/ { … }

```
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
```

2、符号解释

=　　开头表示精确匹配

^~　　开头表示url以某个常规字符串开头，理解为匹配url路径即可，nginx不对url做编码，因此请求为/static/20%/aa,可以被规则 ^$ /static/ /aa 匹配到

~　　区分大小写的正则匹配

~*　　不区分大小写的正则匹配

!~ !~*　　区分大小写不匹配及不区分大小写不匹配的正则

/　　通用匹配，任何请求都会匹配到

多个 location 配置的情况下匹配顺序为首先匹配 = 其次匹配 ^~ 其次是按文件中的顺序的正则匹配，最后是交给 / 通用匹配。 当匹配成功的时候，停止匹配，按当前匹配规则处理请求。

```
location = /login {
  #规则A
}
location = /login {
  #规则B
}
location ^~ /static/ {
  #规则C
}
location ~ \.(gif|jpg|png|js|css)$ {
  #规则D
}
location ~* \.png$ {
  #规则E
}
location !~ \.xhtml$ {
  #规则F
}
location !~* \.xhtml$ {
  #规则G
}
location / {
```

```
  #规则H
}
```

那么产生的效果如下：
访问根目录 /， 比如 http://localhost/ 将匹配规则 A
访问 http://localhost/login 将匹配规则 B，http://localhost/register 则匹配规则 H
访问 http://localhost/static/a.html 将匹配规则 C
访问 http://localhost/a.gif, http://localhost/b.jpg 将匹配规则D和规则E，但是规则 D 顺序优先，规则 E 不起作用，而 http://localhost/static/c.png 则优先匹配到规则 C
访问 http://localhost/a.PNG 则匹配规则 E，而不会匹配规则 D，因为规则 E 不区分大小写。
访问 http://localhost/a.xhtml 不会匹配规则 F 和规则 G ，http://localhost/a.XHTML 不会匹配规则 G，因为不区分大小写。规则 F ，规则 G 属于排除法，符合匹配规则但是不会匹配到，所以想想看实际应用中哪里会用到。
访问 http://localhost/category/id/1111 则最终匹配到规则 H ，因为以上规则都不匹配，这个时候应该是 nginx 转发请求给后端应用服务器，比如 FastCGI（php），tomcat（jsp），nginx 作为反向代理服务器存在。

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/
}
```

3、rewrite语法

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：

执行 server 块的 rewrite 指令

执行 location 匹配

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。

```
server {
  listen 80;
  server_name start.igrow.cn;
  index index.html index.php;
  root html;
  if ($http_host !~ “^star\.igrow\.cn$" {
    rewrite ^(.*) http://star.igrow.cn$1 redirect;
  }
}
```

4、flag标志位

last – 基本上都用这个 Flag。

break – 中止 Rewirte，不在继续匹配

redirect - 返回临时重定向的 HTTP 状态302
permanent - 返回永久重定向的 HTTP 状态301

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：
last 一般写在 server 和 if 中，而 break 一般使用在 location 中
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
break 和 last 都能组织继续执行后面的 rewrite 指令

5、if指令与全局变量
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

1、下面是可以用来判断的表达式：
-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

2、下面是可以用作判断的全局变量

```
# 如果UA包含"MSIE"，rewrite请求到/msid/目录下
if ($http_user_agent ~ MSIE) {
    rewrite ^(.*)$ /msie/$1 break;
}

# 如果cookie匹配正则，设置变量$id等于正则引用部分
if ($http_cookie ~* "id=([^;]+)(?:;|$)") {
    set $id $1;
}

# 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 ($request_method = POST) {
```

```
    return 405;
}

# 限速，$slow可以通过 set 指令设置
if ($slow) {
    limit_rate 10k;
}

# 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 (!-f $request_filename){
    break;
    proxy_pass  http://127.0.0.1;
}

# 如果query string中包含"post=140"，永久重定向到example.com
if ($args ~ post=140){
    rewrite ^ http://example.com/ permanent;
}
location ~* \.(gif|jpg|png|swf|flv)$ {
    valid_referers none blocked www.jefflei.com www.leizhenfang.com;
    # 防盗链
    if ($invalid_referer) {
        return 404;
    }
}

# 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：D:\nginx/html
$request_filename：D:\nginx/html/test1/test2/test.php
```

6、防盗链

```
location ~* \.(gif|jpg|swf)$ {
    valid_referers none blocked start.igrow.cn sta.igrow.cn;
    if ($invalid_referer) {
        rewrite ^/ http://$host/logo.png;
    }
}
```

7、根据文件类型设置过期时间

```
location ~* \.(js|css|jpg|jpeg|gif|png|swf)$ {
    if (-f $request_filename) {
```

```
        expires 1h;
        break;
    }
}
```

8、常用变量

$args： #这个变量等于请求行中的参数，同$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。
$status # 请求状态
$body_bytes_sent # 发送字节
$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。