

**语法规则**

location [=|~|~\*|^~|!~|!~\*] /uri/ { … }

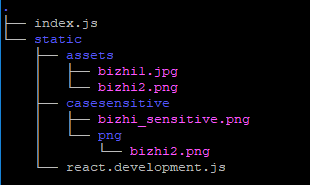
| **模式** | **含义** |
| --- | --- |
| location = /uri | = 表示精确匹配 |
| location ^~ /uri | ^ 进行前缀匹配，~ 表示区分大小写 |
| location ~ pattern | ~ 区分大小写的匹配 |
| location ~\* pattern | ~\* 不区分大小写的匹配 |
| location /uri | 不带任何修饰符，也表示前缀匹配，但是在正则匹配之后 |
| location / | 通用匹配，任何未匹配到其它 location 的请求都会匹配到，相当于 switch 中的 default |
| location !~ | 区分大小写不匹配 |
| location !~\* | 不区分大小写不匹配 |

**匹配优先级**

* 首先精确匹配 =
* 其次前缀匹配 ^~
* 其次是按文件中顺序的正则匹配
* 然后匹配不带任何修饰的前缀匹配
* 最后是交给 / 通用匹配
* 当有匹配成功时候，停止匹配，按当前匹配规则处理请求

匹配的时候依照最佳匹配规则，按照能匹配到的最多的规则进行匹配  
如 location ^~ /test/react/ 和 location ^~ /test/，请求 http://localhost/test/react/react.dev.js，会匹配 location /test/react/

**使用示例**



# t.c => /index

location = / {

proxy\_pass http://127.0.0.1:8888/index;

}

# http://t.c/static/react.development.js => /test-nginx/react.development.js

location ^~ /static/ {

root /home/uftp/test-nginx/;

}

# http://t.c/bizhi1.jpg => /test-nginx/static/assets/bizhi1.jpg

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

root /home/uftp/test-nginx/static/assets/;

}

# http://t.c/bizhi\_sensitive.png 命中 casesensitive/bizhi\_sensitive.png

location ~ \.png$ {

root /home/uftp/test-nginx/static/casesensitive/;

}

# http://t.c/api/plmnji => http://127.0.0.1:8888/apitt/plmnji

location ^~ /api {

proxy\_pass http://127.0.0.1:8888/apitt;

}

# http://t.c/test/react/react.dev.js => http://127.0.0.1:8888/testreact/react.dev.js 不会匹配下面的规则，被上面的规则优先匹配了

# http://t.c/test/react/akkk => http://127.0.0.1:8888/testreact/akkk

location ^~ /test/react {

proxy\_pass http://127.0.0.1:8888/testreact;

}

# http://t.c/test/qwerty => http://127.0.0.1:8888/test/qwerty

location ^~ /test {

proxy\_pass http://127.0.0.1:8888/test;

}

# http://t.c/vue/dasdas => http://127.0.0.1:8888/thisisvue/dasdas

location /vue {

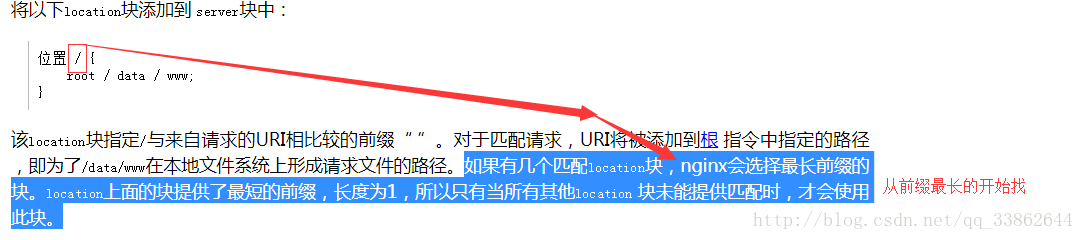
proxy\_pass http://127.0.0.1:8888/thisisvue;

}

## 语法详解

语法规则： location [=|~|~\*|^~] /uri/ { … }

* = 开头表示精确匹配
* ^~ 开头表示uri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（注意是空格）。以xx开头
* ~ 开头表示区分大小写的正则匹配                     以xx结尾
* ~\* 开头表示不区分大小写的正则匹配                以xx结尾
* !~和!~\*分别为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（参考资料而来，还未实际验证，试试就知道了，不必拘泥，仅供参考）：



* 首先精确匹配 =-》其次以xx开头匹配^~-》然后是按文件中顺序的正则匹配-》最后是交给 / 通用匹配。
* 当有匹配成功时候，停止匹配，按当前匹配规则处理请求。
* 例子，有如下匹配规则：
* location = / {
* #规则A
* }
* location = /login {
* #规则B
* }
* location ^~ /static/ {
* #规则C
* }
* location ~ \.(gif|jpg|png|js|css)$ {
* #规则D，注意：是根据括号内的大小写进行匹配。括号内全是小写，只匹配小写
* }
* 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!）。规则F，规则G属于排除法，符合匹配规则也不会匹配到，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nginx转发请求给后端应用[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)，比如FastCGI（php），tomcat（jsp），nginx作为方向代理服务器存在。
* 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/ {                     //以xx开头
* root /webroot/static/;
* }
* location ~\* \.(gif|jpg|jpeg|png|css|js|ico)$ { //以xx结尾
* 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
* location / {
* proxy\_pass http://tomcat:8080/
* }