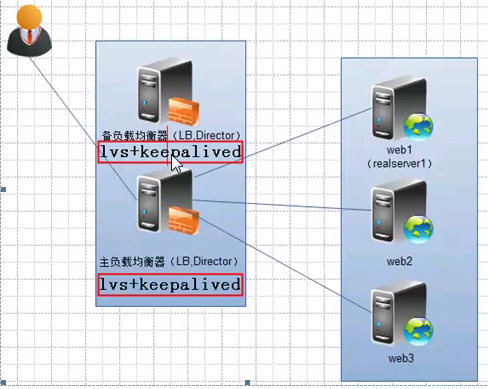
Nginx 深入学习



# 机器规划

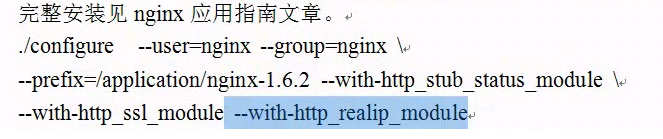
## 机器列表：

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 主机名 | 作用 | 备注 |
| 192.168.137.70 | centos6-lb-server | 负载均衡 |  |
| 192.168.137.81 | centos6-server1 | nginx,httpd |  |
| 192.168.137.82 | centos6-server2 | nginx,httpd |  |
|  |  |  |  |

## 软件准备



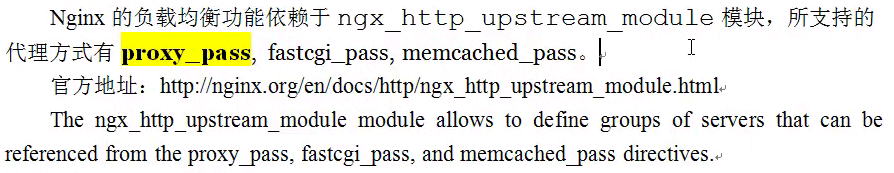
## nginx 安装



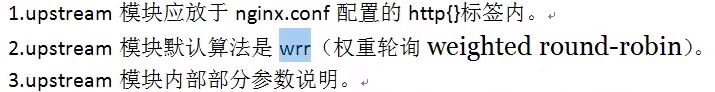
# Nginx相关知识

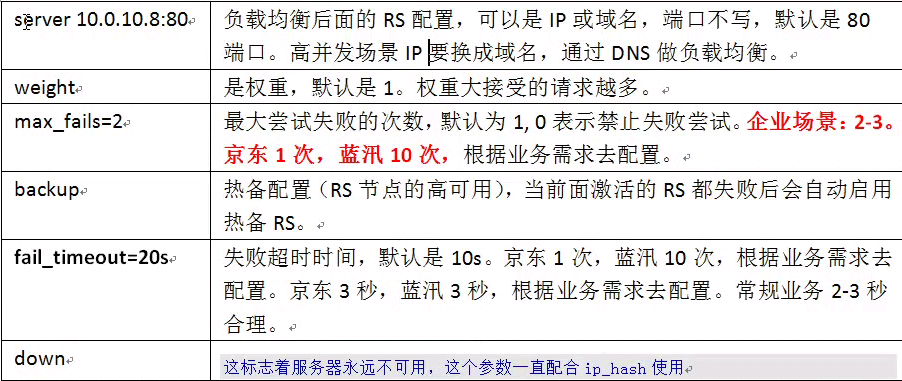
## UPSTREAM

<http://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream>



### 模块相关说明

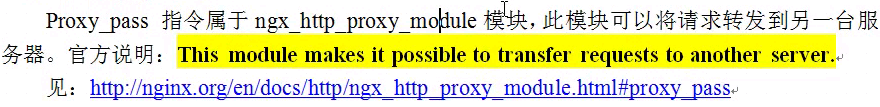




以上参数和专业的haproxy 参数几乎一样。

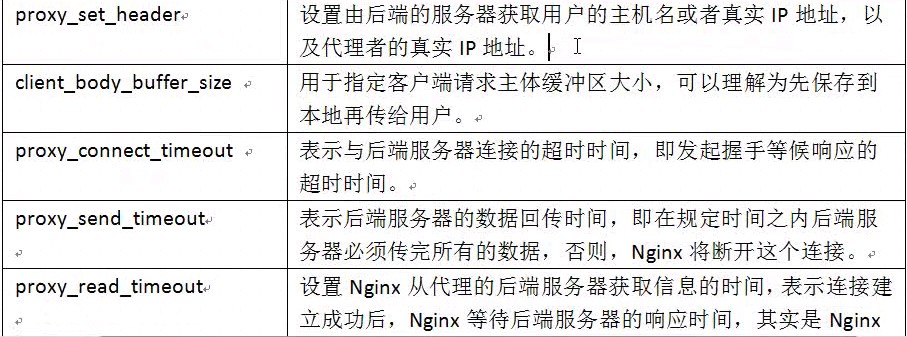
http://nginx.org/en/docs/http/ngx\_http\_proxy\_module.html

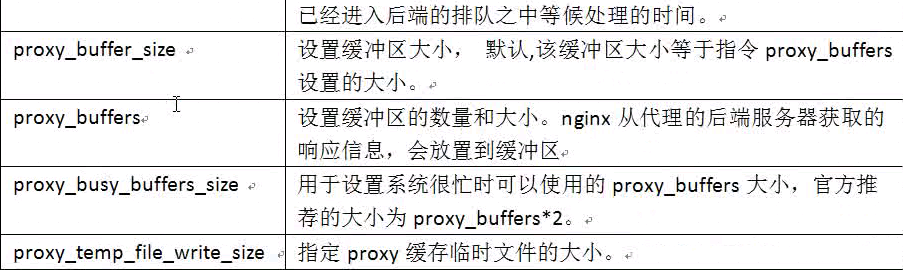
## Proxy\_pass

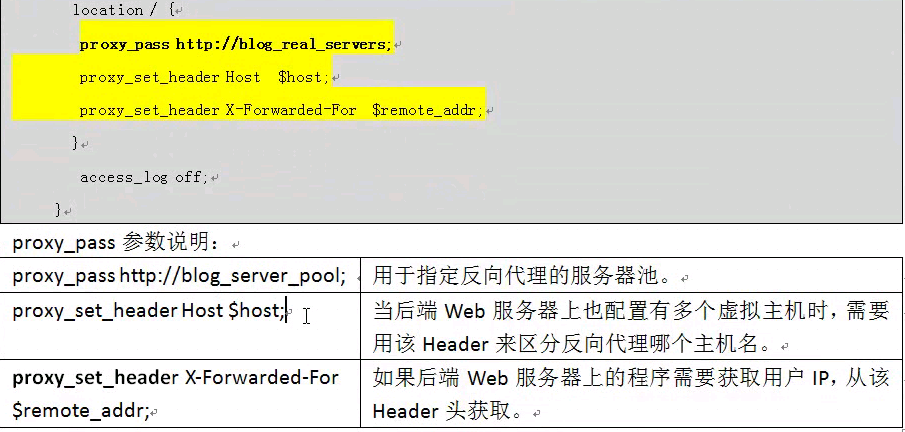


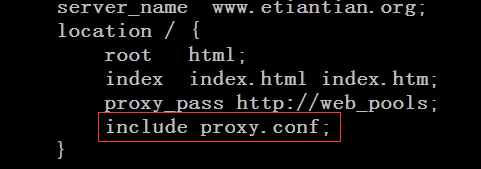
<http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass>

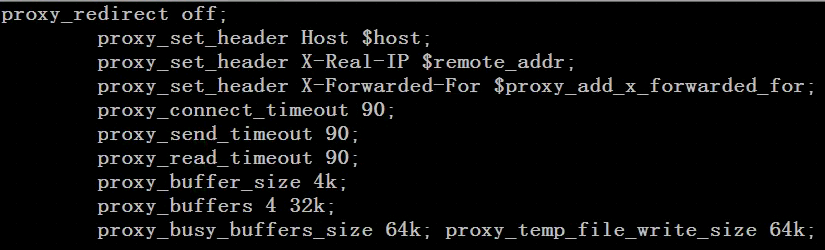
|  |  |
| --- | --- |
| Syntax: | **proxy\_pass** *URL*; |
| Default: | — |
| Context: | location, if in location, limit\_except |
| 相关参数 |  |











## 调度策略

nginx 的upstream目前支持4种方式的分配

### 轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器 ，如果后端服务器down掉，能自动剔除。

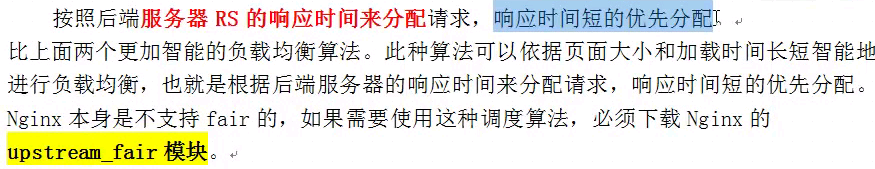
### weight

指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。  例如：  
    upstream bakend {  
         server 192.168.0.14 weight=10;  
         server 192.168.0.15 weight=10;  
    }

### ip\_hash

每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session 的问题。例如：  
    upstream bakend {  
         ip\_hash;  
         server 192.168.0.14:88;  
         server 192.168.0.15:80;  
    }

### fair（第三方）

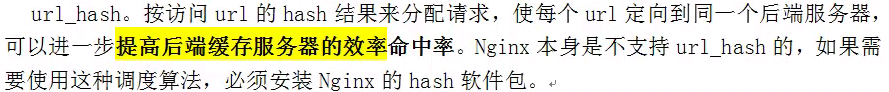


按后端服务器的响应时间来分配请求，响应时间短的优先分配。  
 upstream backend {  
     server server1;  
     server server2;  
     fair;  
 }

### url\_hash（第三方）

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash\_method是使用的hash算法



upstream backend {  
    server squid1:3128;  
    server squid2:3128;  
    hash   $request\_uri;  
    hash\_method crc32;  
}

upstream bakend{#定义负载均衡 设备的Ip及设备状态  
ip\_hash;  
    server 127.0.0.1:9090 down;  
    server 127.0.0.1:8080 weight=2;  
    server 127.0.0.1:6060;  
    server 127.0.0.1:7070 backup;  
}  
在需要使用负载均衡的server中增加  
proxy\_pass http://bakend/ ;

### Least\_connction



## **小技巧参数说明为:**

1.down 表示当前的server暂时不参与负载  
2.weight 默认为1.weight越大，负载的权重就越大2。

3.max\_fails ：允许请求失败的次数默认为1.当超过最大次数时，返回proxy\_next\_upstream 模块定义的错误  
4.fail\_timeout:max\_fails次失败后，暂停的时间。  
5.backup： 其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

nginx支持同时设置多组的负载均衡，用来给不用的server来使用。

client\_body\_in\_file\_only 设置为On 可以讲client post过来的数据记录到文件中用来做debug  
client\_body\_temp\_path 设置记录文件的目录 可以设置最多3层目录

location 对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡

## proxy\_pass后的url加不加/的区别

下面四种情况分别用http://hello.com/css/test.html 进行访问。

第一种：

location  /css/ {

          proxy\_pass http://tomcat/;

}

会被代理到http://192.168.134.142:8080/test.html 这个url

第二种(相对于第一种，最后少一个 /)

location  /css/ {

          proxy\_pass http://tomcat;

}

会被代理到http://192.168.134.142:8080/css/test.html 这个url

第三种：

location  /css/ {

          proxy\_pass http://tomcat/js/;

}

会被代理到http://192.168.134.142:8080/js/test.html 这个url。

第四种情况(相对于第三种，最后少一个 / )：

location  /css/ {

          proxy\_pass http://tomcat/js;

}

会被代理到http://192.168.134.142:8080/js/test.html 这个url

带“/”就会把 <http://servername:port/path/>替换掉

## Nginx对URL进行匹配

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）。

~ 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

!~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：

首先匹配 =，其次匹配^~, 其次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。

“$”符号表示URL结尾

1. 有如下匹配规则例子：

location = / {

#规则A

proxy\_pass http://tomcat/index;

}

location = /login {

#规则B

proxy\_pass http://tomcat/login.jsp;

}

location ^~ /static/ {

#规则C

root /usr/share/nginx/html;

index index.html index.htm;

}

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

#规则D

root /usr/share/nginx/html;

}

location ~\* \.png$ {

#规则E

proxy\_pass http://tomcat;

}

location ~\.shtml$ {

#规则F

root /usr/share/nginx/html;

if (!-e $request\_filename) {

proxy\_pass http://tomcat;

}

}

location ~\*\.shtml$ {

#规则G

root /usr/share/nginx/html;

if (!-e $request\_filename) {

proxy\_pass http://tomcat;

}

}

location / {

#规则H

proxy\_pass http://tomcat;

}

访问根目录/， 比如http://hello.com/ 将匹配规则A

访问http://hello.com/login将匹配规则B，http://hello.com/register则匹配规则H

访问http://hello.com/static/a.html将匹配规则C

访问http://hello.com/a.gif,http://hello.com/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://hello.com/static/c.png则优先匹配到 规则C

访问http://hello.com/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://hello.com/a.shtml不会匹配规则F和规则G，http://hello.com/a.SHTML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://hello.com/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作为方向代理服务器存在。

### Nginx对URL进行匹配-常用

所以实际使用中，至少有三个匹配规则定义常用，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/index](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/](http://tomcat:8080/)

}

## Nginx一些可用的全局变量

$arg\_PARAMETER #这个变量包含GET请求中，如果有变量PARAMETER时的值。

$args     #这个变量等于请求行中(GET请求)的参数，例如foo=123&bar=blahblah;

$binary\_remote\_addr #二进制的客户地址。

$body\_bytes\_sent #响应时送出的body字节数数量。即使连接中断，这个数据也是精确的。

$content\_length #请求头中的Content-length字段。

$content\_type #请求头中的Content-Type字段。

$cookie\_COOKIE #cookie COOKIE变量的值

$document\_root #当前请求在root指令中指定的值。

$document\_uri #与$uri相同。

$host #请求主机头字段，否则为服务器名称。

$is\_args #如果有$args参数，这个变量等于”?”，否则等于”"，空值。

$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。

$query\_string #与$args相同。

$request\_body\_file #客户端请求主体信息的临时文件名。

$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\_completion #如果请求结束，设置为OK. 当请求未结束或如果该请求不是请求链串的最后一个时，为空(Empty)。

$request\_method #GET或POST

$request\_filename #当前请求的文件路径，由root或alias指令与URI请求生成。

$request\_uri #包含请求参数的原始URI，不包含主机名 如：”/foo/bar.php?arg=baz”。不能修改。

$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。

$uri #不带请求参数的当前URI，$uri不包含主机名，如”/foo/bar.html”。该值有可能和$request\_uri 不一致。$request\_uri是浏览器发过来的值。该值是rewrite后的值。例如做了internal redirects后。

## Nginx对URL重写

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

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

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

### Redirect语法

server {

listen 80;

server\_name start.igrow.cn;

index index.html index.php;

root html;

if ($http\_host !~ “^star\.igrow\.cn$&quot {

rewrite ^(.\*) http://star.igrow.cn$1 redirect;

#### 禁止访问某个目录

location ~\* \.(txt|doc)${

root /data/www/wwwroot/linuxtone/test;

deny all;

}

点开头的文件禁止访问，避免以“点”开头的文件（例如.htaccess）被其他用户访问到

location ~ /. {

deny  all;

}

#### 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;

}

}

#### 1.简单例子： rewrite "/zixun/([0-9]+)(/\*).html$" /zixun/$1/ last;

“/zixun/([0-9]+)(/\*).html$”为正则表达式匹配你输入的url地址，表示/zixun/任意数字，至少出现一次，/出现0次或者多次，以.html结尾  
/zixun/$1/ last; 符合以上规则的url 转发到 /zixun/$1/到这个链接上，这个就是你实现要获得数据的链接了 ，last为后面的不进行匹配了

如：http://www.xx.con/zixun/56.html 会把这个请求转发到 www.xx.con/zixun/56/的servlet上获得数据

#### 2.多目录转成参数 abc.hello.com/sort/2 => abc.hello.com/info.jsp?act=sort&name=abc&id=2

if ($host ~\* (.\*)\.hello\.com) {

set $sub\_name $1;

rewrite ^/sort\/(\d+)\/?$ /info.jsp?act=sort&cid=$sub\_name&id=$1 last;

}

3.目录对换  
/123456/xxxx/ -> /xxxx.jsp?id=123456

rewrite ^/(\d+)/(.+)/ /$2.jsp?id=$1 last;

4.例如下面设定nginx在用户使用ie的使用重定向到/nginx-ie目录下：

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

rewrite ^(.\*)$ /nginx-ie/$1 break;

}

5.目录自动加“/”

if (-d $request\_filename){

rewrite ^/(.\*)([^/])$ http://$host/$1$2/ permanent;

}

6.禁止htaccess

location ~/\.ht {

         deny all;

     }

7.禁止多个目录

location ~ ^/(cron|templates)/ {

         deny all;

break;

     }

8.**文件反盗链并设置过期时间**  
这里的return 412 为自定义的http状态码，默认为403，方便找出正确的盗链的请求  
“rewrite ^/ http://hello.com/leech.gif;”显示一张防盗链图片  
“access\_log off;”不记录访问日志，减轻压力  
“expires 3d”所有文件3天的浏览器缓存

location ~\* ^.+\.(jpg|jpeg|gif|png|swf|rar|zip|css|js)$ {

valid\_referers none blocked \*.hello.com localhost;

if ($invalid\_referer) {

    rewrite ^/ http://hello.com/leech.gif;

    return 412;

    break;

}

                 access\_log   off;

                 root /opt/lampp/htdocs/web;

expires 3d;

break;

     }

9.域名跳转

server     {

             listen       80;

             server\_name  h.com;

             index index.html index.htm;

             root  /usr/share/nginx/html;

             rewrite ^/ http://www.baidu.com/;

             access\_log  off;

     }

10.多域名转向

server\_name  hello.com hello.net ~(.\*).hello.com;

             index index.html index.htm index.php;

             root  /usr/share/nginx/html;

if ($host ~ “hello\.net") {

rewrite ^(.\*) http://hello.com/net$1 redirect;

}

11.三级域名跳转

if ($http\_host ~\* "^(.\*)\.i\.hello\.com$") {

rewrite ^(.\*) http://top.hello.com$1;

break;

}

12.域名镜向

server     {

             listen       80;

             server\_name  mirror.hello.com;

             index index.html index.htm;

             root  /usr/share/nginx/html;

             rewrite ^/(.\*) http://www.hello.com/$1 last;

             access\_log  off;

     }

13.某个子目录作镜向

location ^~ /test{

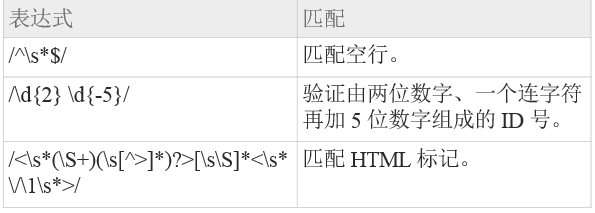
  rewrite ^.+ http://tst.hello.com/ last;

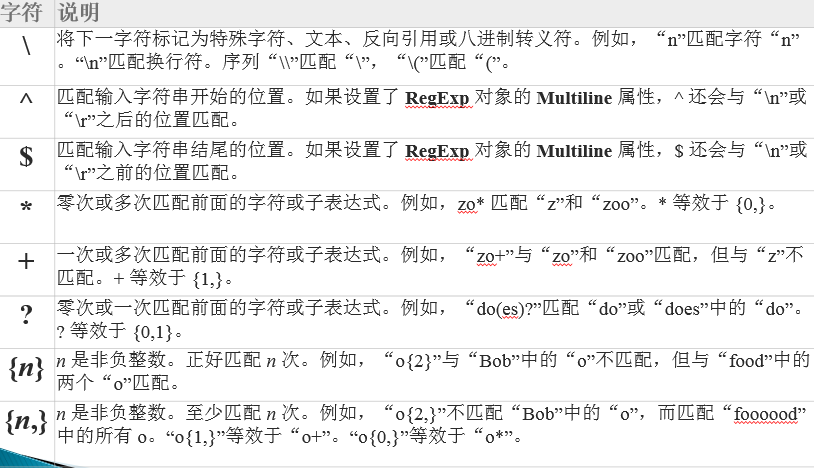
  break;

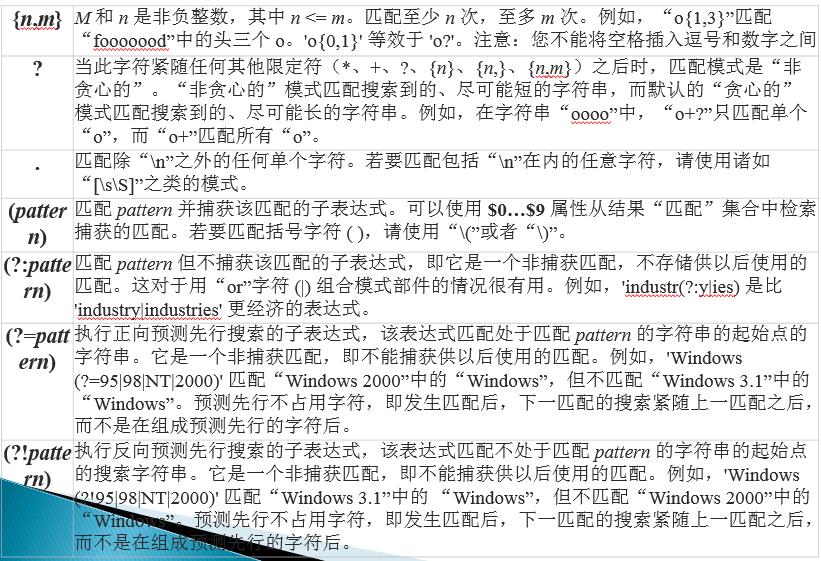
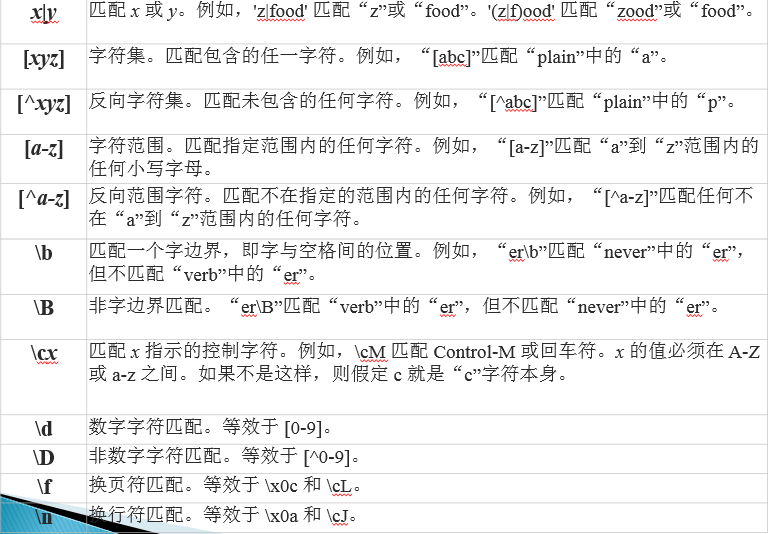
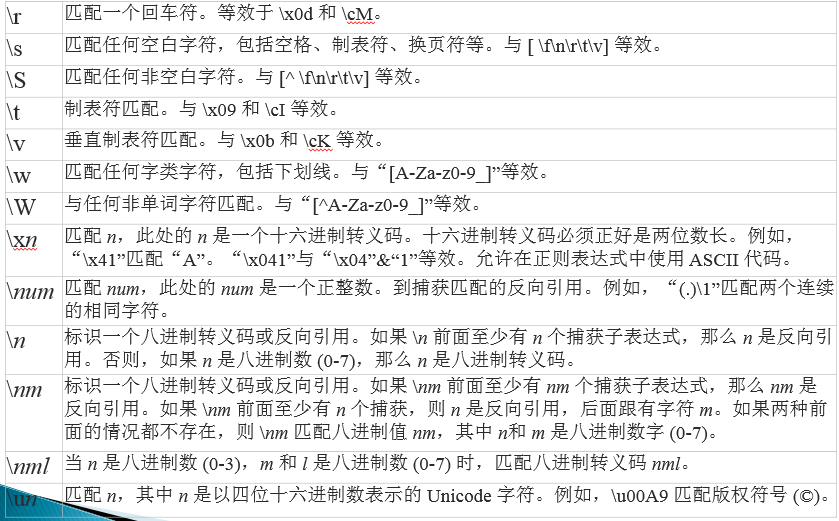
     }

## 正则表达式语法

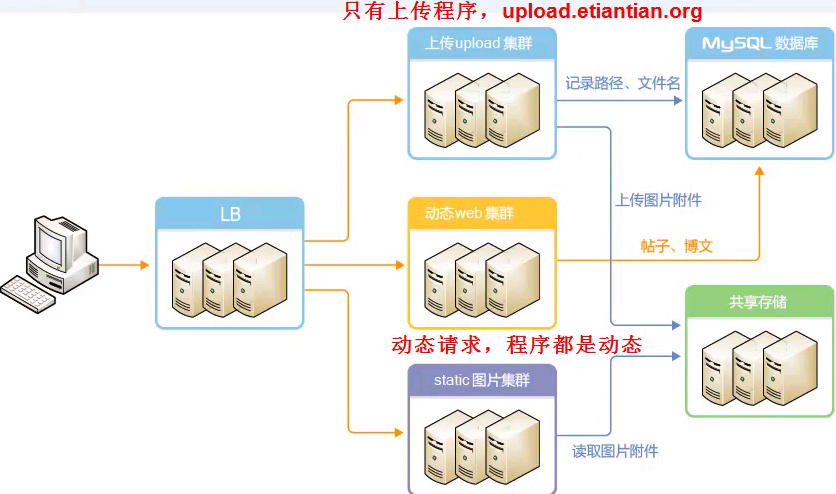
正则表达式是一种文本模式，包括普通字符（例如，a 到 z 之间的字母）和特殊字符（称为“*元字符*”）。模式描述在搜索文本时要匹配的一个或多个字符串。正则表达式示例:





## 动静分离



### 方案1：

location /static/ {

proxy\_pass <http://static_pool>

include proxy.conf

}

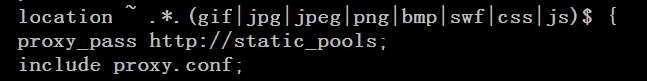
location /dynamic/ { #default

proxy\_pass <http://dynamic_pool>

include proxy.conf

}

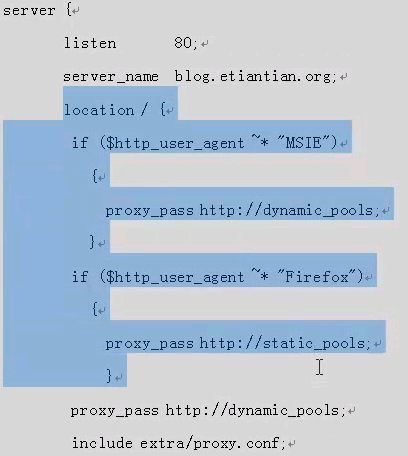
### 方案2 利用扩展名



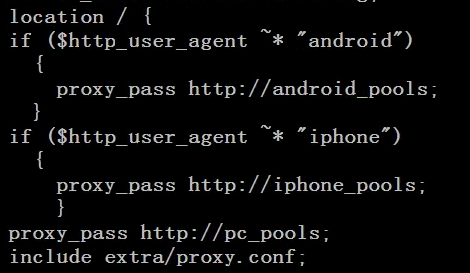
or



## 根据user-agent 转发



### 区分安卓，苹果：



## proxy\_next\_upstream

