（总结）Nginx配置文件nginx.conf中文详解

发表于: Nginx, Web Server, 个人日记,运维经验 | 作者: 谋万世全局者

PS：Nginx使用有两三年了，现在经常碰到有新用户问一些很基本的问题，我也没时间一一回答，今天下午花了点时间，结合自己的使用经验，把Nginx的主要配置参数说明分享一下，也参考了一些网络的内容，这篇是目前最完整的Nginx配置参数中文说明了。更详细的模块参数请参考：<http://wiki.nginx.org/Main>

#定义Nginx运行的用户和用户组

user www www;

#nginx进程数，建议设置为等于CPU总核心数。

worker\_processes 8;

#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]

error\_log /var/log/nginx/error.log info;

#进程文件

pid /var/run/nginx.pid;

#一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（系统的值ulimit -n）与nginx进程数相除，但是nginx分配请求并不均匀，所以建议与ulimit -n的值保持一致。

worker\_rlimit\_nofile 65535;

#工作模式与连接数上限

events

{

#参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型是Linux 2.6以上版本内核中的高性能网络I/O模型，如果跑在FreeBSD上面，就用kqueue模型。

use epoll;

#单个进程最大连接数（最大连接数=连接数\*进程数）

worker\_connections 65535;

}

#设定http服务器

http

{

include mime.types; #文件扩展名与文件类型映射表

default\_type application/octet-stream; #默认文件类型

#charset utf-8; #默认编码

server\_names\_hash\_bucket\_size 128; #服务器名字的hash表大小

client\_header\_buffer\_size 32k; #上传文件大小限制

large\_client\_header\_buffers 4 64k; #设定请求缓存

client\_max\_body\_size 8m; #设定请求缓存

sendfile on; #开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。

autoindex on; #开启目录列表访问，合适下载服务器，默认关闭。

tcp\_nopush on; #防止网络阻塞

tcp\_nodelay on; #防止网络阻塞

keepalive\_timeout 120; #长连接超时时间，单位是秒

#FastCGI相关参数是为了改善网站的性能：减少资源占用，提高访问速度。下面参数看字面意思都能理解。

fastcgi\_connect\_timeout 300;

fastcgi\_send\_timeout 300;

fastcgi\_read\_timeout 300;

fastcgi\_buffer\_size 64k;

fastcgi\_buffers 4 64k;

fastcgi\_busy\_buffers\_size 128k;

fastcgi\_temp\_file\_write\_size 128k;

#gzip模块设置

gzip on; #开启gzip压缩输出

gzip\_min\_length 1k; #最小压缩文件大小

gzip\_buffers 4 16k; #压缩缓冲区

gzip\_http\_version 1.0; #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）

gzip\_comp\_level 2; #压缩等级

gzip\_types text/plain application/x-javascript text/css application/xml;

#压缩类型，默认就已经包含text/html，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。

gzip\_vary on;

#limit\_zone crawler $binary\_remote\_addr 10m; #开启限制IP连接数的时候需要使用

#**upstream**目前支持5种方式的分配:

1.轮询（默认）

2.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

3.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。

4.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。与weight分配策略类似

5.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。注意：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。

#负载均衡模块：使nginx跨越单机的限制，完成网络数据的接收、处理和转发

upstream blog.baidu.com {

ip\_hash;

server 192.168.80.121:80 weight=3;

server 192.168.80.122:80 weight=2;

server 192.168.80.123:80 weight=3;

server 10.0.0.11:9090 down;

server 10.0.0.11:6060 max\_fails=3 fail\_timeout=30s;

server 10.0.0.11:7070 backup;

此外upstream每个后端的可设置参数为：

1.down: 表示此台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。

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，应急措施。所以这台机器压力会最轻。

}

}

Server示例：



#虚拟主机的配置

server

{

#监听的端口

listen 80;

#监听的域名，可以有多个，用空格隔开，也可以用通配符

#nginx将按照准确的server\_name、通配符开始、以通配符结束的、匹配正则表达式的顺序对server name进行匹配，只有有一项匹配以后就会停止搜索

server\_name www.baidu.com baidu.com;

index index.html index.htm index.php;

root /data/www/baidu;

location ~ .\*.(php|php5)?$ {

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

include fastcgi.conf;

}

#图片缓存时间设置

location ~ .\*.(gif|jpg|jpeg|png|bmp|swf)$ {

expires 10d;

}

#JS和CSS缓存时间设置

location ~ .\*.(js|css)?${

expires 1h;

}

#匹配路径

location ^~ /static/ {

#规则C

}

#防盗链

location ~\* \.(gif|jpg|swf)$ {  
 valid\_referers none blocked mp.qq.com px.qq.com;  
 if ($invalid\_referer) {  
 rewrite ^/ http://$host/tips.png;  
 }  
}

#日志格式设定

log\_format access '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" $http\_x\_forwarded\_for';

#定义本虚拟主机的访问日志

access\_log /var/log/nginx/baiduaccess.log access;

#对 "/" 启用反向代理

location / {

proxy\_pass http://127.0.0.1:88;

proxy\_redirect off;

proxy\_set\_header X-Real-IP $remote\_addr;

#后端的Web服务器可以通过X-Forwarded-For获取用户真实IP

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

#以下是一些反向代理的配置，可选。

proxy\_set\_header Host $host;

#设置cookie存储的代理路径

proxy\_cookie\_path /mobilepc114 /;

proxy\_set\_header Cookie $http\_cookie

client\_max\_body\_size 10m; #允许客户端请求的最大单文件字节数（上传文件用到）

client\_body\_buffer\_size 128k; #缓冲区代理缓冲用户端请求的最大字节数

proxy\_connect\_timeout 90; #nginx跟后端服务器连接超时时间(代理连接超时)

proxy\_send\_timeout 90; #后端服务器数据回传时间(代理发送超时)

proxy\_read\_timeout 90; #连接成功后，后端服务器响应时间(代理接收超时)

proxy\_buffer\_size 4k; #设置代理服务器（nginx）保存用户头信息的缓冲区大小

proxy\_buffers 4 32k; #proxy\_buffers缓冲区，网页平均在32k以下的设置

proxy\_busy\_buffers\_size 64k; #高负荷下缓冲大小（proxy\_buffers\*2）

proxy\_temp\_file\_write\_size 64k;

#设定缓存文件夹大小，大于这个值，将从upstream服务器传

}

#设定查看Nginx状态的地址

location /NginxStatus {

stub\_status on;

access\_log on;

auth\_basic "NginxStatus";

auth\_basic\_user\_file conf/htpasswd;

#htpasswd文件的内容可以用apache提供的htpasswd工具来产生。

}

#本地动静分离反向代理配置

#所有jsp的页面均交由tomcat或resin处理

location ~ .(jsp|jspx|do)?$ {

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_pass http://127.0.0.1:8080;

}

}

## [关于一些对location认识的误区](http://www.zrqblog.org/?p=699)

**1、 location 的匹配顺序是“先匹配正则，再匹配普通”。**

矫正： location 的匹配顺序其实是“先匹配普通，再匹配正则”。我这么说，大家一定会反驳我，因为按“先匹配普通，再匹配正则”解释不了大家平时习惯的按“先匹配正则，再匹配普通”的实践经验。这里我只能暂时解释下，造成这种误解的原因是：正则匹配会覆盖普通匹配（实际的规则，比这复杂，后面会详细解释）。

**2、 location 的执行逻辑跟 location 的编辑顺序无关。**

矫正：这句话不全对，“普通 location ”的匹配规则是“最大前缀”，因此“普通 location ”的确与 location 编辑顺序无关；但是“正则 location ”的匹配规则是“顺序匹配，且只要匹配到第一个就停止后面的匹配”；“普通location ”与“正则 location ”之间的匹配顺序是？先匹配普通 location ，再“考虑”匹配正则 location 。注意这里的“考虑”是“可能”的意思，也就是说匹配完“普通 location ”后，有的时候需要继续匹配“正则 location ”，有的时候则不需要继续匹配“正则 location ”。两种情况下，不需要继续匹配正则 location ：（ 1 ）当普通 location 前面指定了“ ^~ ”，特别告诉 Nginx 本条普通 location 一旦匹配上，则不需要继续正则匹配；（ 2 ）当普通location 恰好严格匹配上，不是最大前缀匹配，则不再继续匹配正则。

**总结一句话：  “正则 location 匹配让步普通 location 的严格精确匹配结果；但覆盖普通 location 的最大前缀匹配结果”**

1. #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。
2. #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
3. # 第一个必选规则
4. location = / {
5. proxy\_pass http://tomcat:8080/index
6. }
8. # 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
9. # 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
10. location ^~ /static/ {
11. root /webroot/static/;
12. }
13. location ~\* \.(gif|jpg|jpeg|png|css|js|ico)$ {
14. root /webroot/res/;
15. }
17. #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
18. #非静态文件请求就默认是动态请求，自己根据实际把握
19. #毕竟目前的一些框架的流行，带.php,.jsp后缀的情况很少了
20. location / {
21. proxy\_pass http://tomcat:8080/
22. }

[配置expires](http://blog.csdn.net/gameboyx/article/details/10894057)

语法： expires [time|epoch|max|off]

默认值： expires off

作用域： http, server, location

使用本指令可以控制HTTP应答中的“Expires”和“Cache-Control”的头标，（起到控制页面缓存的作用）。

可以在time值中使用正数或负数。“Expires”头标的值将通过当前系统时间加上您设定的 time 值来获得。如：30d、1h、-1

epoch 指定“Expires”的值为 1 January, 1970, 00:00:01 GMT。

max 指定“Expires”的值为 31 December 2037 23:59:59 GMT，“Cache-Control”的值为10年。

-1 指定“Expires”的值为 服务器当前时间 -1s,即永远过期

“Cache-Control”头标的值由您指定的时间来决定：

* 负数：Cache-Control: no-cache
* 正数或零：Cache-Control: max-age = #, # 为您指定时间的秒数。

"off" 表示不修改“Expires”和“Cache-Control”的值