nginx配置

upstream配置

```
upstream
ktvsky_api {

server 10.10.14.241:
9100 max_fails=
3 fail_timeout=
30s weight=4;

# Nginx是基于连接探测的，如果发现后端异常，在单位周期为fail_timeout设置的时间中失败次数达到max_fails次，这个周期次数内，如果后端同一个节点不可用，那么就将把节点标记为不可用，并等待下一个周期（同样时长为fail_timeout）再一次去请求，判断是否连接是否成功这样就能说明我们发现的现象了。即在30s以内后端失败了3次【即一次请求超时】，那么这个后端就被标识为不可用了，所以在接下来的30s期间，nginx都会把请求分配给正常的后端【即多次的请求正常】。

keepalive 16;

keepalive_timeout 75;

# 设置keep-alive客户端连接在服务器端保持开启的超时值（默认75s）；值为0会禁用keep-alive客户端连接；

keepalive_requests 100;
```

keepalive_requests指令用于设置一个keep-alive连接上可以服务的请求的最大数量，当最大请求数量达到时，连接被关闭，值为0会也禁用keep-alive客户端连接；。默认是100。

}

keepalive:

- keepalive的值表示最大的空闲链接数，并不是到上游服务的最大连接数
- keepalive的值不需要配置的特别大，占用过多的连接
- keepalive的值如果配置的太小的话，会导致空闲链接超过最大值，不停的被回收，也是会有端口号的浪费的
- keepalive指定的数值是针对每个worker的，并且是针对所有后端的

nginx通过 setsockopt(ls[i].fd, SOL_SOCKET, SO_KEEPALIVE,(const void *) &value, sizeof(int))开启keepalive后，会始终和客户端保持长连接如此会出现一个很严峻的问题，每个woker的能保持的连接数是有限的

（ep = epoll_create(cycle->connection_n / 2); cycle->connection_n / 2 为epoll能管理的fd上限），连接数很快就被耗尽,这时可通过 keepalive_timeout, keepalive_requests 来管理长连接

server配置

=

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。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

。

```
server {

listen 80;

listen

443 ssl;

server_name api.ktvsky.com;

ssl_certificate  /home/work/nginx/conf/ktvsky.com.chained.2020.crt;
```

```
ssl_certificate_key /home/work/nginx/conf/ktvsky.com.key;

add_header
'Access-Control-Allow-Origin'
'*';

location /ad/pos/
ads {

proxy_pass
http://ktvsky_api;
  }

# =开头表示精确匹配

location = /ad/
state {

proxy_pass
http://ktvsky_api;
  }

location
~
^/ad/
stat {

default_type application/json;
```

```
return
200
'{"errcode": "200", "errmsg": "请求成功"}';
  }

location
~
^/ad/
(caption
|region
|pos
) {

proxy_set_header
Host $host:$server_port;

proxy_set_header
X-Real-IP   $remote_addr;

proxy_set_header
X-Forwarded-For $proxy_add_x_forwarded_for;

# 当后端服务器返回502、...错误等状态码，自动跳转到upstream负载均衡池中的另一台服务器，实现故障转义

proxy_next_upstream
error timeout
invalid_header http_500
```

```
http_502 http_503 http_504;

# 缓存位置

proxy_cache vadd_old;

# 针对200、304状态码缓存12小时

proxy_cache_valid
200 304 1h;

proxy_cache_valid
any 2m;

# 定义缓存Key值的格式，Nginx将Key值HASH后再存储到指定的二级缓存目录,Key值存储格式为（请求主机、请求的URL、如果$args设置为？，否则为空、GET请求中的参数）

proxy_cache_key $host$uri$scheme$is_args$args;

add_header
Nginx-Cache
"$upstream_cache_status";

expires 1d;

proxy_pass
http://ktvsky_api;

proxy_redirect default;
```

```
  }

location /ad/
policy {

default_type application/json;

return
200
'{"errcode": "200", "errmsg": "请求成功"}';
  }

location /
erp {

proxy_pass
http://ktvsky_api;
  }
```

不同location匹配

```
server {

listen     80;

server_name www.test.com;

# 情形A
```

```
# 访问 http://www.test.com/testa/aaaa

# 后端的request_uri为: /testa/aaaa

location
^
~ /testa/ {

proxy_pass
http://127.0.0.1:8801;
  }


# 情形B

# 访问 http://www.test.com/testb/bbbb

# 后端的request_uri为: /bbbb

location
^
~ /testb/ {

proxy_pass
http://127.0.0.1:8801/;
  }

# 情形C
```

```
# 下面这段location是正确的

location
~ /
testc {

proxy_pass
http://127.0.0.1:8801;
  }

# 情形D

# 下面这段location是错误的

#

# nginx -t 时，会报如下错误:

#

# nginx: [emerg] "proxy_pass" cannot have URI part in location given by regular

# expression, or inside named location, or inside "if" statement, or inside

# "limit_except" block in /opt/app/nginx/conf/vhost/test.conf:17

#
```

```
# 当location为正则表达式时，proxy_pass 不能包含URI部分。本例中包含了"/"

location
~ /
testd {

proxy_pass
http://127.0.0.1:8801/;
# location为正则表达式时，不能这样写
  }

# 情形E

# 访问 http://www.test.com/ccc/bbbb

# 后端的request_uri为: /aaa/ccc/bbbb

location /ccc/ {

proxy_pass
http://127.0.0.1:8801/aaa$request_uri;
  }

# 情形F

# 访问 http://www.test.com/namea/ddd

# 后端的request_uri为: /yongfu?namea=ddd
```

```
location /namea/ {

rewrite    /namea/
([
^/]+
) /
yongfu?namea=
$1 break;

proxy_pass
http://127.0.0.1:8801;
  }

# 情形G

# 访问 http://www.test.com/nameb/eee

# 后端的request_uri为: /yongfu?nameb=eee

location /nameb/ {

rewrite    /nameb/
([
^/]+
) /
yongfu?nameb=
$1 break;

proxy_pass
```

```
http://127.0.0.1:8801/;
  }


location = /favicon.
ico {

empty_gif;

access_log off;
  }


location /fb/ {

root /home/work/online/src/
wow-web;

try_files
$uri $uri/ /fb/
index.html;
  }


#如果一个请求的URI是/t/a.html时，web服务器将会返回服务器上的/www/root/html/t/a.html的文件。

location
^
```

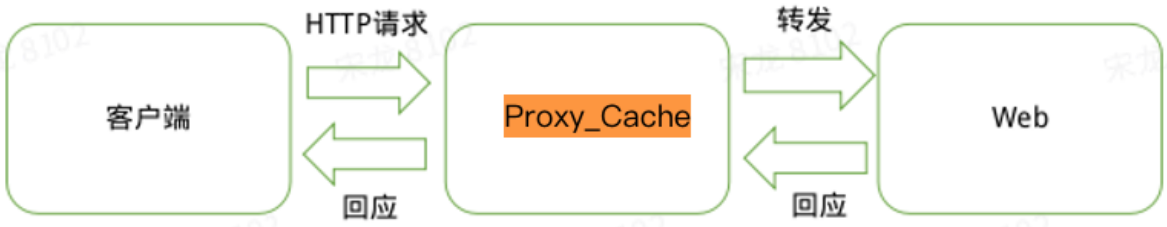
```
~ /t/ {

root /www/
root/html/;
    }

#如果一个请求的URI是/t/a.html时，web服务器将会返回服务器上
的/www/root/html/new_t/a.html的文件。因为alias会把location后面配置的路径丢弃掉，把当前匹
配到的目录指向到指定的目录。

location
^
~ /t/ {

alias /www/
root/html/new_t/;
  }

access_log /data/logs/www/www.test.com.log;
}
server {

listen     8801;

server_name www.test.com;


root      /data/www/test;
```

```
index
index.
php index.html;

rewrite
^
(.*
)
$ /test.
php?u=
$1 last;

location
~
\.
php$ {

try_files
$uri =404;

fastcgi_pass unix:/tmp/
php-cgi.sock;

fastcgi_index
index.php;

include fastcgi.conf;
  }
```

```
access_log /data/logs/www/www.test.com.8801.log;
}
```

proxy_cache

nginx提供的缓存，把URL以及相关信息当成key，用MD5编码哈希后，把数据文件保存在硬盘上，并且只能为指定的URL或者状态码设置过期时间



1.Nginx Proxy_Cache收到请求后，会首先捕获，对请求的URL进行HASH，查看自己的缓存文件是否命中
2.Nginx如果命中（HIT），直接从缓存中回应客户端请求
3.如未命中，代理将请求转发给后端Web，Web回应内容，Nginx获取数据HASH后缓存并回应给客户端

rewite

在server块下，会优先执行rewrite部分，然后才会去匹配location块

server中的rewrite break和last没什么区别，都会去匹配location，所以没必要用last再发起新的请求，可以留空

location中的rewirte：

不写last和break - 那么流程就是依次执行这些rewrite

1. rewrite break

    url重写后，直接使用当前资源，不再执行location里余下的语句，完成本次请求，地址栏url不变

2. rewrite last

    url重写后，马上发起一个新的请求，再次进入server块，重试location匹配，超过10次匹配不到报500错误，地址栏url不变

3. rewrite redirect

    返回302临时重定向，地址栏显示重定向后的url，爬虫不会更新url（因为是临时）

4. rewrite permanent

返回301永久重定向, 地址栏显示重定向后的url，爬虫更新url

使用last会对server标签重新发起请求

如果location中rewrite后是对静态资源的请求，不需要再进行其他匹配，一般要使用break或不写，直接使用当前location中的数据源，完成本次请求

如果location中rewrite后，还需要进行其他处理，如动态fastcgi请求(.PHP,.jsp)等，要用last继续发起新的请求

(根的location使用last比较好, 因为如果有.php等fastcgi请求还要继续处理)