反向代理

> 反向代理是代理服务器的一种。它根据客户端的请求，从后端的服务器（如Web服务器）上获取资源，然后再将这些资源返回给客户端。与反向代理不同，正向代理作为一个媒介将互联网上获取的资源返回给相关联的客户端，而反向代理是在服务器端（如Web服务器）作为代理使用，而不是客户端。客户端通过正向代理可以访问很多不同的资源，而反向代理是很多客户端都通过它访问不同后端服务器上的资源，而不需要知道这些后端服务器的存在，而以为所有资源都来自于这个反向代理服务器。

nginx

> nginx采用master主进程负责管理worker进程的工作模式。master进程接收外界信号给worker进程发送信号，及监控管理worker进程。nginx采用异步非阻塞的方式处理请求，子进程竞争接受新连接的方式，每个worker进程在某个瞬间只能处理一个连接，每个worker进程有一个独立的连接池，连接池大小是worker_connections，即每个进程支持的最大连接数。

nginx反向代理

> 在nginx中，实现反向代理功能的模块是`ngx_http_proxy_module`模块。

示例配置

> nginx反向代理模块的配置指令除少数几个之外，都可以配置在http, server, location的配置段。

```
location / {
    proxy_pass        http://localhost:8000;
    proxy_set_header Host      $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

配置指令

> **proxy_pass**
>
> *proxy_pass配置指令配置代理服务器余后端服务器通信的的协议及所在服务映射位置,通常配置在location段。客户请求location段指定的URL时，映射到后端服务器所对应的路径。*

```
# 1.用户向代理服务器发出的请求URL是http://192.168.1.150/xuekaixin/，则会映射到http://192.168.1.151/app/
location /xuekaixin/ {
    proxy_pass http://192.168.1.151/app/;
}
# 2.如果location中使用了rewrite，则映射的是重定向后的URL，即此时proxy_pass定义的URL失效，客户请求的URL是http://192.168.1.150/web1/则会跳转至http://192.168.1.150/web2/
location / {
    rewrite /web1/ /web2/$1 break;
    proxy_pass http://192.168.1.151/web1/;
    index  index.html index.htm;
}
# 3.location中使用正则表达式匹配URL，则不进行映射，只是将location中的路径补到proxy_pass的URL后，
# 且parocy_pass后头不能再写路径，否则被认为是语法错误。客户请求的URL是http://192.168.1.150/web1/，
# 则会查找后端服务器web1目录下的资源
location ~* /web1/ {
    proxy_pass http://192.168.1.151;
    index  index.html index.htm;
}
# 错误示例：
location ~* /web1/ {
    proxy_pass http://192.168.1.151/web2/;
    index  index.html index.htm;
}
```

proxy_set_header

后端服务器接收到代理服务器的请求时，会记录日志，但日志中的源IP是代理服务器的IP，不能记录真正的客户端IP地址。可以通过proxy_set_header指令利用nginx内置变量将客户端IP传递给后端服务器，然后由后端服务器读取变量并记录进日志。将报文头部重新定义或附加到传递给代理服务器的请求报文中。(自定义头部信息传递给后端服务器)。

```
location ~* /web1/ {
    proxy_pass http://192.168.1.151/web2/;
    index  index.html index.htm;
    proxy_set_header X-Real-IP $remote_addr;
}
# 后端服务器日志配置
LogFormat "%{X-Real-IP}i %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

proxy_ignore_client_abort

当客户端断开与服务器的连接且不等待响应时，是否关闭代理服务器与后端服务器的连接。

```
proxy_ignore_client_abort on|off;
```

proxy_intercept_errors

当客户端请求状态码大于或等于300时，是否拦截并重定向到nginx的error_page。

```
proxy_intercept_errors on|off;
```

proxy_method

指定代理服务器向后端服务器请求方法。

```
proxy_method method;
```

nginx缓存功能

> nginx反向代理服务器将客户端请求的内容缓存到本地，如果有客户端下次请求的内容在缓存中，则代理服务器不用向上游服务器取资源，而是快速响应，直接将内容返回给客户端。

示例配置

```
http {
    proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=cache-name:10m;
}
location / {
    proxy_cache cache-name;
}
```

配置指令(location配置段)

> **proxy_cache_path**
>
> *定义缓存存放路径及其属性。`proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=one:10m;`*

```
# #缓存存放目录
/data/nginx/cache
# 冒号隔开的几段，表示可以使用几级缓存子目录，每一段的数字，表示哪一级的缓存目录名称字符数,如果levels参数没有配置，则NGINX会将所有的文件放到同一个目录中
levels=1:2
# 内存中保存缓存的空间名称及大小，用来保存键
keys_zone=cache-name:10m
# 指定了缓存在不被访问的情况下能够在内存中保持的时间。默认是10分钟，如果在指定时间内资源被访问了，则刷新删除时间
inactive=10m
# 临时文件存放设定。如果为on，则为proxy_temp_path指定的路径存放，如果为off，则和缓存存放 为一个路径，建议为off，避免过多的拷贝数据
use_temp_path=off
# 设定最大缓存上限
max_size=10g
```

> **proxy_cache**
>
> *引用缓存定义*

```
proxy_cache cache-name;
```

> **proxy_temp_path**
>
> *定义用于存储具有从代理服务器接收的数据的临时文件的目录。最多可以在指定目录下使用三级子目录层次结构。*

```
proxy_temp_path /spool/nginx/proxy_temp 1 2;
```

> **proxy_cache_use_stale**
>
> *定义当代理服务器与后端服务器通信发生那种错误时，代理服务器可以用旧的缓存响应客户端。updating选项，可以最大限度地减少更新缓存数据时对代理服务器的访问次数。意思当客户端访问的资源正在从后端服务器更新至缓存时，将旧的内容发给客户端，直到更新完毕后，客户端再次请求时，将新的资源发给客户端。*

```
proxy_cache_use_stale error timeout http_500 http_502
http_503 http_504;
```

> **proxy_cache_methods**
>
> *指定对客户端的某些请求方法才缓存，默认是GET、HEAD*

```
proxy_cache_methods GET|HEAD|POST ...;
```

> **proxy_connect_timeout**
>
> *定义与代理服务器建立连接的超时时间。应该注意的是，这个超时通常不能超过75秒,可配置在http，server，location段*

```
proxy_connect_timeout 60s;
```

> **proxy_cache_min_uses**
>
> *定义客户端请求多少次之后将被缓存下来*

```
proxy_cache_min_uses 1;
```

> **proxy_cache_lock**
>
> *当多个客户同时请求多个缓存中不存在资源时，代理服务器只向后端请求一次，之后的将缓存返回给客户，避免代理服务器过多的与后端服务器的通信。*

```
proxy_cache_lock on;
```

> **proxy_cache_valid**
>
> *为不同的响应代码设置缓存时间*

```
proxy_cache_valid 200 302 10m;
proxy_cache_valid 404 1m;
proxy_cahce_valid any 1m;
```

> **proxy_cache_revalidate**

> *启用使用带有"If-Modified-Since"和"If-None-Match"头字段的条件请求的过期缓存项目的重新验证。*

```
proxy_cache_revalidate on|off;
```

> **proxy_read_timeout**
>
> *表示从后端服务器读数据的超时时间，如果两次连续的读操作时间间隔大于设定值，则断开连接*

```
proxy_read_timeout 180s;
```

> **proxy_send_timeout**
>
> *向后端写数据的超时时间，如果两次连续的写操作时间间隔大于设定值，则断开连接*

```
proxy_send_timeout 60s;
```

> **proxy_buffering**
>
> *启用或关闭来自后端服务器的响应缓冲，当启用缓冲后，代理服务器收到后端服务器的响应后，将其保存至proxy_buffer_size和proxy_buffers*指令设置的缓冲区中，当缓冲关闭时，代理服务器收到响应后立马返回给请求方。

```
proxy_buffering on|off;
```

检测缓存状态

```
add_header X-Cache-Status $upstream_cache_status;
```

> 在对客户端的响应中添加了一个`X-Cache-Status`HTTP响应头，从而可以知道缓存的状态，$upstream_cache_status的可能值。

1. `MISS`：响应在缓存中没找到，从原始服务器取得，此次请求之后可能被缓存下来
2. `HIT`：响应包含来自缓存的最新有效的内容

3. `BYPASS`：响应来自原始服务器而不是缓存，这个响应之后可能会被缓存起来
4. `EXPIRED`：缓存过期，请求从后端服务器来
5. `STALE`：内容陈旧是因为原始服务器不能正确响应。需要配置 proxy_cache_use_stale
6. `UPDATING`：缓存过期，正在更新返回旧的内容