### 启动

brew services start nginx 加入开机启动

nginx 简单启动

安装目录：/usr/local/etc/nginx/

配置文件路径：/usr/local/etc/nginx/nginx.conf

nginx -t ：测试你的配置文件得语法，并告诉你配置文件是否写得正确，同时也告诉了你配置文件得路径

### 反向代理

反向代理服务器作用在服务器端，它在服务器端接收客户端的请求，然后将请求分发给具体的服务器进行处理，然后再将服务器的相应结果反馈给客户端。

### 基本命令

|  |
| --- |
| nginx -s <signal> |

-s意思是向主进程发送信号，signal可以为以下四个中的一个:

* 1. stop — 快速关闭
  2. quit — 优雅关闭
  3. reload — 重新加载配置文件
  4. reopen — 重新打开日志文件

### 设置虚拟服务器

Nginx 配置文件至少包含一个 server 命令 ，用来定义虚拟服务器。当请求到来时， Nginx 会首先选择一个虚拟服务器来处理该请求。

#### listen

监听本机 IP 和端口号

|  |
| --- |
| server {  listen 127.0.0.1:8080; # IPv4地址，8080端口  } |

如果不写端口号，默认使用80端口，如果不写 IP ，则监听本机所有 IP

#### server\_name

对应请求头中的Host;

其值可以为：

1. 完整的主机名，如：api.lufficc.com 。
2. 含有通配符（含有 \*），如：\*.lufficc.com 或 api.\* 。
3. 正则表达式，以 ~ 开头。

注意：通配符只能在开头或结尾，而且只能与一个 . 相邻。

正则可以任意匹配：~^www\..+\.example\.org$ 和 ~^w.\*\.example\.org$

若正则里有使用(), 后面可以用$1,$2来取值

|  |
| --- |
| server {  server\_name ~^(www\.)?(.+)$;  location / {  root /sites/$2;  }  } |

若server\_name有多个匹配规则，顺序入下：

1. 完整的主机名，如 api.lufficc.com。
2. 最长的，且以 \* 开头的通配名，如：\*.lufficc.com。
3. 最长的，且以 \* 结尾的通配名，如：api.\* 。
4. 第一个匹配的正则表达式。（按照配置文件中的顺序）

如果 Host 头部不匹配任何一个 server\_name ,Nginx 将请求路由到默认虚拟服务器。默认虚拟服务器是指：nginx.conf 文件中第一个 server 或者 显式用 default\_server 声明：

|  |
| --- |
| server {  listen 80 **default\_server**;  ...  } |

#### location

当选择好 server 之后，Nginx 会根据 URIs 选择合适的 location 来决定代理请求或者返回文件。

location 指令接受两种类型的参数：

1. 前缀字符串（路径名称）,必须以该字符串开始
2. 正则表达式

|  |
| --- |
| location /images/ {  root /data;  } |

http://localhost/images/example.png 会被映射到/data/images/example.png上

对于正则表达式，以 ~ 开头表示大小写敏感，以 ~\* 开头表示大小写不敏感。

**正则表达式**的优先级大于前缀字符串。如果找到匹配的前缀字符串，仍继续搜索正则表达式，但如果前缀字符串以 ^~ 开头，则不再检查正则表达式。

具体的搜索匹配流程如下：

1. 将 URI 与所有的前缀字符串进行比较。
2. = 修饰符表明 URI 必须与前缀字符串相等（不是开始，而是相等），如果找到，则搜索停止。（**拥有最高的优先级**）
3. 如果找到的最长前缀匹配字符串以 ^~ 开头，则不再搜索正则表达式是否匹配。
4. 存储匹配的最长前缀字符串。
5. 测试对比 URI 与正则表达式。
6. 找到第一个匹配的正则表达式后停止。
7. 如果没有正则表达式匹配，使用 4 存储的前缀字符串对应的 location。

代理服务器

接受客户端的请求并将其转发给代理服务器，再接受代理服务器发来的响应，将它们发送到客户端

|  |
| --- |
| server {  location / {  # proxy\_pass指令的参数为：协议+主机名+端口号  **proxy\_pass** http://localhost:8080;  }  location ~ \.(gif|jpg|png)$ {  root /data/images;  }  } |

第二个location 为正则匹配；以~开头，匹配以.gif，.jpg或.png结尾的所有URI

#### 使用变量

|  |  |
| --- | --- |
| 变量名称 | 作用 |
| $uri | 请求中的URI(不带请求参数)，如：/foo/bar.html |
| $arg\_name | 请求中的的**参数名**，“?”后面的arg\_name=arg\_value形式的arg\_name |
| $hostname | 主机名 |
| $args | 请求中的参数值 |
| $request | 代表客户端的请求地址 |
| $request\_uri | URI+args，如/cnphp/test.php?arg=freemouse |

#### 返回特定状态码

如果你的网站上的一些资源永久移除了，最快最简洁的方法就是使用 return 指令直接返回：

|  |
| --- |
| location /wrong/url {  return 404;  } |

return CODE URL；

第一个参数是响应代码；

可选的第二个参数可以是重定向（对应于代码301，302，303和307）的 URL 或在响应正文中返回的文本

|  |
| --- |
| location /permanently/moved/url {  return 301 http://www.example.com/moved/here;  } |

#### 错误处理

当404 错误发生时返回 /404.html 页面

|  |
| --- |
| error\_page 404 /404.html; |

当 Nginx 找不到页面时，它将使用代码301替换代码404，并将客户端重定向到 http://example.com/new/path.html

|  |
| --- |
| location /old/path.html {  error\_page 404 =301 http:/example.com/new/path.html;  } |

#### 重写URIs

|  |
| --- |
| location /users/ {  rewrite ^/users/(.\*)$ /show?user=$1 break;  } |

rewrite第一个参数：URI需要匹配的正则表达式；

第二个参数：将要替换的 URI

第三个参数：是否继续可以重写或者返回重定向代码（last/break）,可选；

可以在server和location上下文包含多个rewrite指令，Nginx按照他们发生的顺序一个一个执行；

nginx执行一组 rewrite 指令之后，它根据新的 URI 选择 location；如果所选 location 仍旧包含 rewrite 指令，它们将依次执行。

last 与 break的区别是：

**last**：跳出当前 server 或 location 上下文中， Nginx 继续搜索与重写的URI匹配的 location，并应用新 location 中的任何 rewrite 指令（这意味着 URI 可能再次改变）。

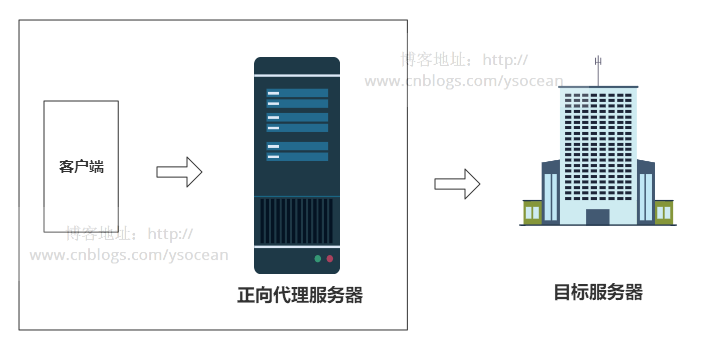
**break**：跳出当前上下文，并停止搜索与新URI匹配的location，就此结束；

### 正向代理和反向代理

反向代理和正向代理的区别就是：**正向代理代理客户端，反向代理代理服务器**。

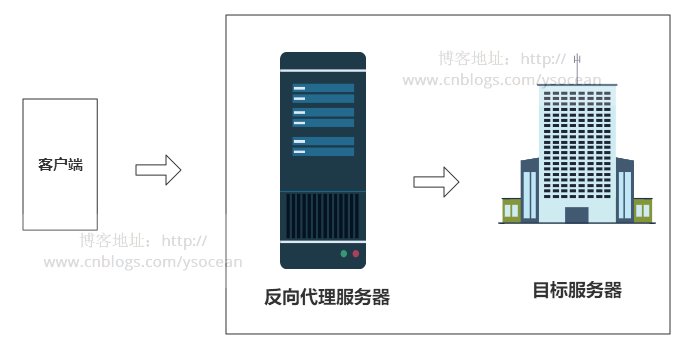
理解这两种代理的关键在于代理服务器所代理的对象是什么：正向代理代理的是客户端，我们需要在客户端进行一些代理的设置。而反向代理代理的是服务器，作为客户端的我们是无法感知到服务器的真实存在的。

#### 正向代理



举一个例子：大家都知道，现在国内是访问不了 Google的，那么怎么才能访问 Google呢？我们又想，美国人不是能访问 Google吗（这不废话，Google就是美国的），如果我们电脑的对外公网 IP 地址能变成美国的 IP 地址，那不就可以访问 Google了。你很聪明，VPN 就是这样产生的。我们在访问 Google 时，先连上 VPN 服务器将我们的 IP 地址变成美国的 IP 地址，然后就可以顺利的访问了。

#### 反向代理



反向代理，其实客户端对代理是无感知的，因为客户端不需要任何配置就可以访问，我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器IP地址。

### nginx容器

from: http://www.ruanyifeng.com/blog/2018/02/nginx-docker.html

#### http服务

|  |
| --- |
| docker container run \  -d \  -p 8080:80 \  --rm \  --name mynginx \  nginx |

启动nginx,此时可以通过localhost:8080访问首页

建目录：mkdir nginx-docker-demo && cd nginx-docker-demo

将nginx的配置文件（etc/nginx）和根目录（/usr/share/nginx/html）拷贝到本地，方便修改

|  |
| --- |
| docker container cp mynginx:/etc/nginx .  docker container cp mynginx:/usr/share/nginx/html . |

此时nginx-docker-demo下会新增html文件和nginx文件

简单修改下html/index.php

#### 自签名证书

现在要为容器加入 HTTPS 支持，第一件事就是生成私钥和证书。正式的证书需要证书当局（CA）的签名，这里是为了测试，搞一张自签名（self-signed）证书就可以了。

在nginx下新建certs文件并进入：

cd nginx && mkdir certs && cd certs

|  |
| --- |
| $ sudo openssl req \  -x509 \  -nodes \  -days 365 \  -newkey rsa:2048 \  -keyout example.key \  -out example.crt |

上面参数含义：

|  |
| --- |
| req：处理证书签署请求。  -x509：生成自签名证书。  -nodes：跳过为证书设置密码的阶段，这样 Nginx 才可以直接打开证书。  -days 365：证书有效期为一年。  -newkey rsa:2048：生成一个新的私钥，采用的算法是2048位的 RSA。  -keyout：新生成的私钥文件为当前目录下的example.key。  -out：新生成的证书文件为当前目录下的example.crt。 |

其中最重要的一个问题是 Common Name，正常情况下应该填入一个域名，这里可以填 127.0.0.1。

此时certs新增两个文件example.key和example.crt。

#### https配置

在nginx/conf.d/default.conf后面追加：

这样及支持http又支持https访问

|  |
| --- |
| server {  **listen 443 ssl** http2;  server\_name localhost;  ssl\_certificate /etc/nginx/certs/example.crt;  ssl\_certificate\_key /etc/nginx/certs/example.key;  ssl\_session\_timeout 5m;  ssl\_ciphers HIGH:!aNULL:!MD5;  ssl\_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;  ssl\_prefer\_server\_ciphers on;  location / {  root /usr/share/nginx/html;  index index.html index.htm;  }  } |

在nginx-docker-demo目录下

启动一个新容器：

|  |
| --- |
| docker container run \  --rm \  --name mynginx \  -p 8080:80 \  -p 8081:443 \  --volume "$PWD/html":/usr/share/nginx/html \  --volume "$PWD/nginx":/etc/nginx \  nginx |

改容器不仅映射了80端口，也映射了443(https)端口；

**[https:](https://127.0.0.1/8081)**[//127.0.0.1/8081](https://127.0.0.1/8081)去访问，浏览器会提示不安全，继续访问即可。

Tips:

nginx的配置文件在nginx/nginx.conf，里包含了nginx/conf.d/default.conf

|  |
| --- |
| include /etc/nginx/conf.d/\*.conf; |

#### http自动跳转https

|  |
| --- |
| # 将Http请求转化成Https请求  server {  listen 80;  server\_name xxx;  rewrite ^/(.\*) https://$server\_name$request\_uri? permanent;  } |