1. Nginx虚拟主机:

1.1 虚拟主机作用:

虚拟主机提供了在同一台服务器上运行多个网站的功能。

1.2 虚拟主机的三种模式:

- 基于多域名方式配置虚拟主机
- 基于多端口配置虚拟主机
- 基于多ip配置虚拟主机

基于多域名的虚拟主机是最常见的一种虚拟主机。

只需配置你的DNS服务器，将每个主机名映射到正确的IP地址，然后配置Nginx服务器，令其识别不同的主机名就可以了。

```
网域名称系统（DNS，Domain Name System，将域名和IP地址相互映射的一个分布式数据库）是因特网的一项核
心服务，它作为可以将域名和IP地址相互映射的一个分布式数据库，能够使人更方便地访问互联网，而不用去记
住能够被机器直接读取的IP地址数串。
参考：
https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D/86062?fr=aladdin
```

基于多端口的虚拟主机，可以使用同一个ip，通过访问不同的端口来访问。

基于多IP的虚拟主机可以通过添加多个网卡或者在一块物理网卡上绑定多个IP地址来实现，

ps:如果没有特殊要求，最好还是使用基于多域名的虚拟主机。

1.2.1 多域名配置虚拟主机:

1.2.1.1 配置hosts文件:

hosts文件：在本机计算机上面，配置本地的ip地址和域名映射关系，通常用于测试。

- Windows 本地hosts文件

```
C:\Windows\System32\drivers\etc\hosts

192.168.75.132   www.jfedu.com
```

- Linux 本地hosts 文件

```
vim /etc/hosts

192.168.75.132   www.jfedu.com
```

1.2.1.2 创建虚拟主机配置文件:

```
#在http指定块中添加:
include  /usr/local/nginx/conf/vhost/*.conf;
```

```
#创建配置主机配置文件

mkdir -p /usr/local/nginx/conf/vhost
vim /usr/local/nginx/conf/vhost/www.jfedu.com.conf

server {
    listen 80;
    server_name www.jfedu.com;
    location / {
        root /usr/local/nginx/html/jfedu;
        index index.html index.htm;
    }
}

vim  /usr/local/nginx/conf/vhost/www.jf.com.conf

server {
    listen 80;
    server_name www.jf.com;
    location / {
        root /usr/local/nginx/html/jf;
        index index.html index.htm;
    }
}
```

1.2.2 多端口配置虚拟主机：

```
server {
        listen 8080;
        server_name www.jfedu.com;
        location /  {
                root /usr/share/nginx/html/jfedu;
                index index.html;
                }
        }
```

1.2.3 多IP配置虚拟主机：

```
cp /etc/sysconfig/network-scripts/ifcfg-ens32{,:1}
vim /etc/sysconfig/network-scripts/ifcfg-ens32:1

修改以下信息:
NAME="ens32:1"
DEVICE="ens32:1"
IPADDR=192.168.75.188

# 重启服务:
systemctl restart network

server {
        listen 192.168.75.188:80;
        server_name www.jfedu.com;
        location /  {
                root /usr/share/nginx/html/jfedu;
                index index.html;
                }
        }

重启nginx服务:

systemctl  restart nginx
```

2. Nginx反向代理：

2.1 反向代理概念：

反向代理是nginx的一个重要功能，在编译安装时会默认编译该模块。在配置文件中主要配置proxy_pass指令。

代理服务器接受客户端的请求，然后把请求代理给后端真实服务器进行处理，然后再将服务器的响应结果返给客户端。

2.2 反向代理作用：

与正向代理（**正向代理主要是代理客户端的请求**）相反，**反向代理主要是代理服务器返回的数据**，所以它的作用主要有以下两点：

1. 可以防止内部服务器被恶意攻击（内部服务器对客户端不可见）。
2. 为负载均衡和动静分离提供技术支持。

2.3 反向代理语法：

```
Syntax:  proxy_pass URL;
Default:    —
Context: location, if in location, limit_except
```

代理服务器的**协议**，可支持http与https。

地址可以指定为域名或IP地址，以及可选端口。

例如：

```
proxy_pass http://192.168.0.188;
proxy_pass http://192.168.0.188:8080;
proxy_pass http://localhost:9000/uri/;
```

2.4 实例一：

location和proxy_pass都不带uri路径。

代理服务器：192.168.0.109

后端服务器：192.168.0.114

代理服务器的简单配置：

```
location / {
            proxy_pass http://192.168.0.114;
        }
# proxy_pass  转发请求给后端服务器
```

后端服务器的配置：

```
 location / {
            echo $host;
            root   html;
            index  index.html index.htm;
        }

# echo $host  这个主要是来看下后端接收到的Host是什么。
```

2.4.1 验证：

```
[root@localhost ~]# curl 192.168.0.109
192.168.0.114

# 获取的请求Host是后端服务器ip，去掉该指令，验证请求结果。

[root@localhost ~]# curl 192.168.0.109
                        this is 114 page

# 可以看到我们访问的是109，但是得到的结果是114的发布目录文件。
```

2.5 实例二：

proxy_pass没有设置uri路径，但是**代理服务器的location 有uri**，那么代理服务器将把客户端请求的地址传递给后端服务器。

```
# 代理服务器的配置：

location /document/data/ {
            proxy_pass http://192.168.0.114;
}

# 后端服务器的配置：

 location / {
           # echo $host;
            root   html/uri;
            index  index.html index.htm;
        }
```

2.5.1 验证：

```
[root@localhost ~]# mkdir -p
/usr/local/nginx/html/uri/document/data/
[root@localhost ~]# echo "this is
/usr/local/nginx/html/uri/document/data/ test" >
/usr/local/nginx/html/uri/document/data/index.html
[root@localhost ~]# curl 192.168.0.109/document/data/
this is /usr/local/nginx/html/uri/document/data/ test

# 完整请求路径 是在后端服务器的/usr/local/nginx/html/uri 后追加
客户端请求的路径 /document/data/
```

2.6 实例三：

如果**proxy_pass设置了uri路径**，则需要注意，此时，proxy_pass指令所指定的uri会覆盖location的uri。

```
# 代理服务器的配置：

location / {
            proxy_pass http://192.168.0.114/data/;
        }

# 后端服务器的配置：

location / {
            root   html;
            index  index.html index.htm;
        }
```

2.6.1 验证：

```
[root@localhost ~]# mkdir -p /usr/local/nginx/html/data/
[root@localhost ~]# echo "this is
/usr/local/nginx/html/data test。" >
/usr/local/nginx/html/data/index.html
[root@localhost ~]# curl 192.168.0.109
this is /usr/local/nginx/html/data test。

这样看好像很正常，但是稍作修改。
```

这次加上location的uri，后端服务器加个子目录：

```
代理服务器的配置：


location /document/ {
            proxy_pass http://192.168.0.114/data/;
}

后端服务器的配置：

 location / {
            #echo $host;
            root   html;
            index  index.html index.htm;
        }
```

2.6.2 再次验证：

```
[root@localhost ~]# curl 192.168.0.109/document/
this is /usr/local/nginx/html/data test。

# 该路径还是 proxy_pass 指定的uri路径，与location 没有关系了！
```

2.7 获取远程客户端真实ip地址：

```
# 代理服务器配置：
location / {
            proxy_pass http://192.168.0.114;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
}
# 后端服务器配置：
log_format  main  '$remote_addr - $remote_user
[$time_local] "$request" '
                      '$status $body_bytes_sent
"$http_referer" '
                      '"$http_user_agent"
"$http_x_real_ip" "$http_x_forwarded_for"';
access_log  logs/access.log  main;
```

3. 缓存代理服务器实战：

在代理服务器的磁盘中保存请求目标的内容，加快响应速度，减少应用服务器（后端服务器）上的资源开销，比如**多客户端**请求相同的资源，代理缓存命中后，对于应用服务器来说，只发生了一次资源调度。

而浏览器上的缓存配置，一般来说是用来减少本地IO的，请求目标的内容会存放在浏览器本地。

```
# 代理服务器配置：
proxy_cache_path /data/nginx/cache  max_size=10g
 levels=1:2 keys_zone=nginx_cache:10m inactive=10m
use_temp_path=off;

upstream nginx {
```

```
        server 192.168.0.114;
}

server {
    listen       80;
    server_name  localhost;
location / {
    root   html;
    index  index.html index.htm;
    proxy_pass http://nginx;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_cache nginx_cache;
    proxy_cache_key $host$uri$is_args$args;
    proxy_cache_valid 200 304 302 1d;
    }
}
# /data/nginx/cache  #缓存资源存放路径
# levels             #设置缓存资源的递归级别，默认为
levels=1:2，表示Nginx为将要缓存的资源生成的key从后依次设置两级保
存。
key_zone           #在共享内存中设置一块存储区域来存放缓存的key和
metadata，这样nginx可以快速判断一个request是否命中或者未命中缓存，
1m可以存储8000个key，10m可以存储80000个key
max_size           #最大cache空间，如果不指定，会使用掉所有disk
space，当达到配额后，会删除不活跃的cache文件
inactive           #未被访问文件在缓存中保留时间，本配置中如果60分钟
未被访问则不论状态是否为expired，缓存控制程序会删掉文件。inactive默
认是10分钟。需要注意的是，inactive和expired配置项的含义是不同的，
expired只是缓存过期，但不会被删除，inactive是删除指定时间内未被访问
的缓存文件
use_temp_path    #如果为off，则nginx会将缓存文件直接写入指定的
cache文件中，而不是使用temp_path存储，official建议为off，避免文件
在不同文件系统中不必要的拷贝
proxy_cache      #启用proxy cache，并指定key_zone。如果
proxy_cache off表示关闭掉缓存。
```