1. 说说Nginx和apache有什么区别

```
# 架构设计不同
Nginx
事件驱动架构：
使用异步非阻塞 I/O 和事件循环，单线程可以处理成千上万的连接。
每个工作进程使用事件模型处理多个请求，资源占用低。

Apache
进程/线程驱动架构：
默认使用多进程模型（prefork 模式），每个请求由一个独立的进程处理。
也支持多线程模式（worker 模式）或事件模式（event 模式），但线程和进程间的切换开销较高。

# 内存与资源占用
Nginx
内存占用低：
单线程模型更节省内存，适合大规模并发场景。
适合高负载环境。

Apache
内存占用高：
每个连接需要一个独立的线程/进程，资源开销大。
适合小流量环境。

# 负载均衡和反向代理
Nginx
内置强大的负载均衡和反向代理功能，支持多种均衡算法（如轮询、最小连接）。
直接支持健康检查、请求缓存等高级功能。

Apache
虽然可以通过模块（如 mod_proxy、mod_lb）实现负载均衡和反向代理，但配置复杂，性能不如 Nginx。

# HTTPS 和 HTTP/2
Nginx
原生支持 HTTP/2 和 SSL/TLS，性能优化较好。
配置简单，广泛用于 HTTPS 网站。
Apache
支持 HTTP/2 和 SSL/TLS，但需要加载额外模块（如 mod_ssl）。
性能不如 Nginx。
```

2. 当有攻击者使用未注册的域名访问服务器时，默认的 Nginx 配置可能会处理这些请求。如何解决?

```
server {
    listen 80 default_server;
    server_name _;
    location / {
        root /var/www/error_pages;
        index 404.html;
    }
}
```

3. 请简述Nginx的进程结构

```
Master 进程 --> Worker进程 --> Cache manager --> Cache loader
```

4. Nginx的优点是什么

```
# 高并发，高性能
大部分的程序或者web服务器，随着并发连接数的上升，它的RPS会下降

nginx同时具备高并发，和并性能，往往高并发只要每个连接所使用的内存尽可能少就能达到，而具有高并发的同时具有高性能，需要一个非常好的设计

nginx可以达到的标准：比如一些主流服务器，32核64G内存，可以达到千万级别的并发连接，如果处理一些简单的静态资源请求，可以达到一百万级别的RPS

# 可扩展性好
模块化设计，生态圈丰富

# 高可靠性
nginx可以在服务器上不间断运行数年

# 热部署（在不停止服务的情况下，升级Nginx）

# BSD许可证（开源免费，并且在有特定需求的场景下，可以修改Nginx的源代码，然后运行在商业场景下）
```

5. 什么是正向代理和反向代理?

```
# 正向代理
正向代理是客户端的代理，位于客户端和服务器之间，帮助客户端向服务器发出请求。服务器并不知道客户端的真实身份，只知道请求是来自代理服务器。

# 反向代理
反向代理是服务器的代理，位于客户端和服务器之间，帮助服务器接收客户端的请求并转发给内部的真实服务器。客户端并不知道真实服务器的存在，只与反向代理服务器通信。
```

6. Nginx负载均衡的算法怎么实现的?策略有哪些

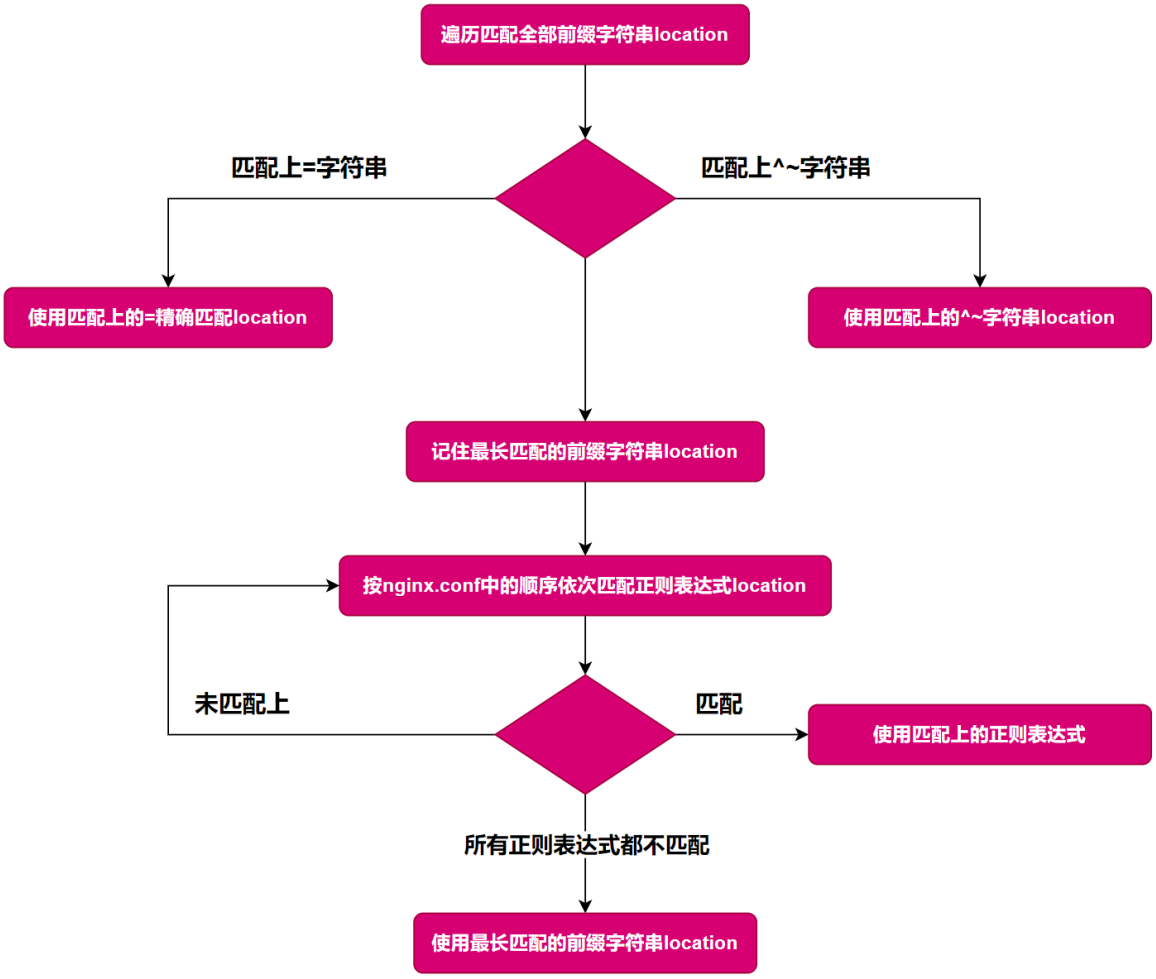
```
#round-robin
#加权round-robin
upstream rrups {
    server 127.0.0.1:8011 weight=2 max_conns=2 max_fails=2 fail_timeout=5;
    server 127.0.0.1:8012;
    keepalive 32;
}

#ip_hash
Syntax: ip_hash;
Default: --
Context: upstream

# hash模块
Syntax: hash key [consistent];
Default: --
Context: upstream

upstream iphashups {
    ip_hash;
    # hash user_$arg_usename;
    # 由于我们使用了ip_hash因此此处的weight权重不会生效
    server 127.0.0.1:8011 weight=2 max_conns=2 max_fails=2 fail_timeout=5;
    server 127.0.0.1:8012 weight=1;
}

# 一致性hash算法
Syntax: hash key [consistent];
Default: --
Context: upstream

# 最少连接算法
Syntax: least_conn;
Default: --
Context: upstream

# upstream_zone
Syntax: zone name [size];
Default: --
Context: upstream
```

7. 你对nginx做过哪些优化

```
CPU结亲缘
指定worker进程数，使其一个CPU核心上执行一个进程
优先级（worker_priority）
   优先级越高，worker进程分的时间片时间越长，进程间切换频率越低
增加work_connection的数值，默认512，这个是nginx单个work进程能够处理的连接数
开启防惊群设置`accept_mutex on`默认是off,会产生惊群问题
适当设置反向代理的keepalived连接池数量

极端情况下，可以基于url的大小调整连接内存池大小
```

8. 说一下nginx的location中匹配优先级

```
# 语法规则
location [ = | ~ | ~* | ^~ ] uri {...}

=  # 用于标准uri前，表示请求字符串和uri精确匹配，大小敏感
^~ # ^~ 是用于 匹配 URL 路径的前缀匹配 的关键字。它表示在所有其他正则表达式匹配之前，优先匹配指定的前缀路径。如果一个请求的 URL 匹配到带有 ^~ 的路径规则，Nginx 会立即使用该规则并停止进一步匹配，即使后续正则表达式可能更精确。
~  # 用于标准uri前，表示包含正则，区分大小写
~* # 用于标准uri前，表示包含正则，不区分大小写
不带符号 # 匹配起始于此uri的所有uri
\  # 用于标准uri前，表示包含正则表达式并且转义字符，可以将.*?等转译为普通符号

# 匹配优先级:
=,  ^~, ~/~*, 不带符号
```



9. 你用过的nginx的常用模块有哪些

```
Postread阶段：realip模块
Server rewrite阶段：rewrite模块
preaccess阶段：limit_req模块，limit_conn模块
access阶段：access模块，auth_basic模块，auth_request模块
precontect阶段：try_files模块
content阶段：index模块，autoindex模块，反向代理模块（upsteam）


过滤模块：在log阶段之前，content阶段之后进行处理的，比如：gzip模块，image_filter模块（缩略图）等
变量相关模块：referer模块

log阶段：log模块
```

10. 实验题：

公司开发了一个网站：

- 客户端访问proxyServer10.0.0.150, www.magedu.com
- 动态内容由后端服务10.0.0.151处理（比如：www.magedu.com/api/下的文件是动态资源）

- 静态资源有后端服务10.0.0.152处理（比如：www.magedu.com/static/下的文件是静态资源）

为了优化性能，要求：

1. 对静态资源启用缓存，缓存到 `/var/cache/nginx`，缓存有效期为 5分钟，并且每1分钟访问一次后端静态服务看是否有更新。

验证：客户端访问ProxyServer，如果是动态资源，转发到10.0.0.151,如果是静态资源，转发到10.0.0.152

提示：IP地址任选，上述ip仅供参考

```
# proxyServer
proxy_cache_path /apps/nginx/proxycache levels=1:2 keys_zone=proxycache:20m
inactive=5m max_size=1g;


server {
    server_name feng.magedu.org;
    root /apps/nginx/html;

    location /api/ {
            proxy_pass http://10.0.0.151;
            proxy_set_header Host "api.feng.org";
        }

    location /static/ {
            proxy_pass http://10.0.0.152;
            proxy_set_header Host "static.feng.org";
            proxy_cache proxycache;
            proxy_cache_key $request_uri;
            proxy_cache_valid 200 302 301 60s;
            proxy_cache_valid any 2m;
            add_header X-Cache $upstream_cache_status;
    }
}
```

```
# api
vim /etc/nginx/sites-enable/api.conf

server {
    server_name api.feng.org;
    root /var/www/html;
}

# static
vim /etc/nginx/sites-enable/static.conf

server {
    server_name static.feng.org;
    root /var/www/html;
}
```