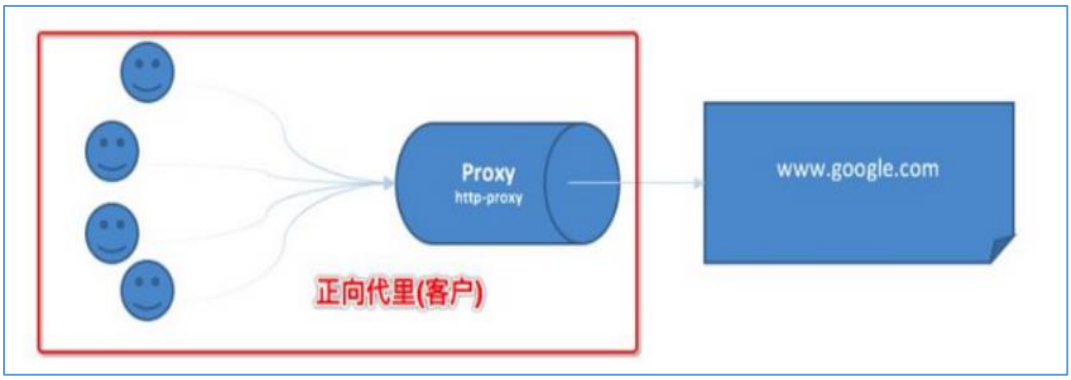
1 Nginx 入门

1.1背景分析

1.1.1 正向代理

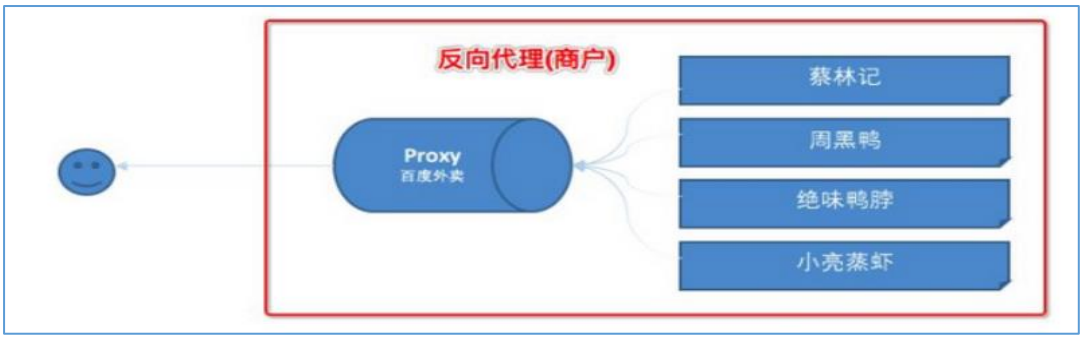
正向代理类似一个跳板机，代理访问外部资源，比如我们国内访问谷歌，直接访问访问不到，我们可以通过一个正向代理服务器，请求发到代理服，代理服务器能够访问谷歌，这样由代理去谷歌取到返回数据，再返回给我们，这样我们就能访问谷歌了，如图所示：



特点说明:正向代理服务器一般都是**客户端代理，代理客户端执行业务**

1.1.2 反向代理

反向代理服务器位于**用户**与目标**服务器**之间，对于用户而言，反向代理服务器就相当于目标服务器，即用户直接访问反向代理服务器就可以获得目标服务器的资源。同时，用户不需要知道目标服务器的地址，也无须在用户端作任何设定。反向代理服务器通常可用来作为 Web 加速，即使用反向代理作为 Web 服务器的前置机来降低网络和服务器的负载，提高访问效率。如图所示：



特点说明:反向代理服务器一般都是**服务器端代理**.用户无需关心真实的服务器是谁.

1.2 Nginx 简介

1.2.1 概述

Nginx (engine x) 是一个高性能的 HTTP 和反向代理 web 服务器，同时也提供了 IMAP/POP3/SMTP 服务。Nginx 是由伊戈尔·赛索耶夫为俄罗斯访问量第二的 Rambler.ru 站点（俄文：Рамблер）开发的，第一个公开版本 0.1.0 发布于 2004 年 10 月 4 日。

其将源代码以类 BSD 许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011 年 6 月 1 日，nginx 1.0.4 发布。

Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在 BSD-like 协议下发行。其特点是**占有内存少，并发能力强**，事实上 nginx 的并发能力在同类型的网页服务器中表现较好，中国大陆使用 nginx 网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

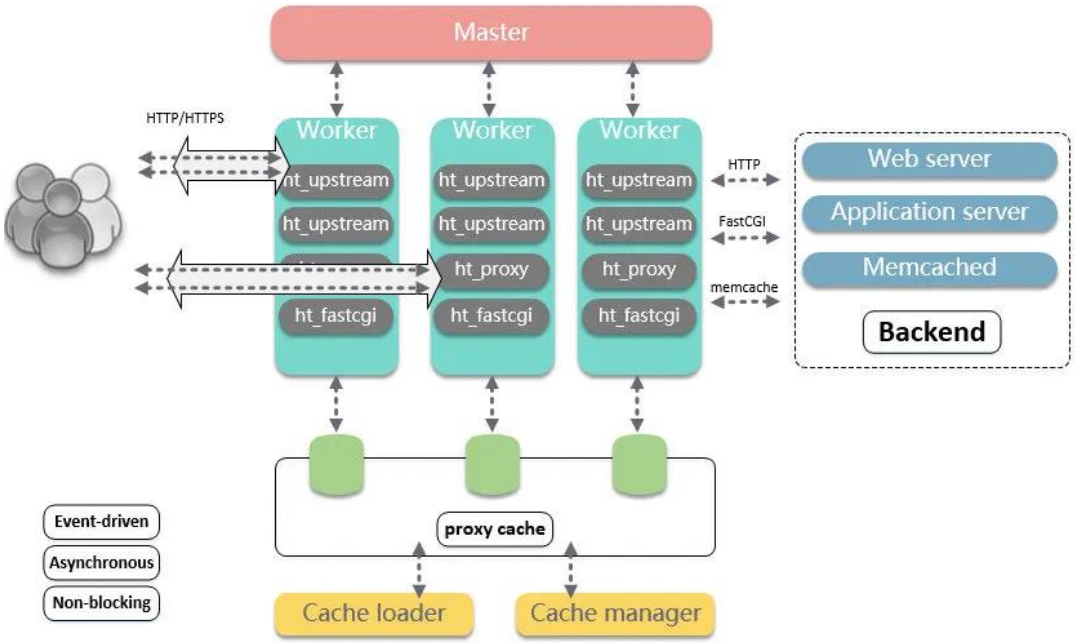
特点总结:占用内存小(不到 3M C 语言开发),并发能力强（5 万/秒）

1.2.2 体系结构分析

nginx 启动后，以 daemon 形式在后台运行，后台进程包含一个 master 进程和多个 worker 进程。可通过如下指令进行查看。

```
ps -ef | grep nginx
```

nginx 是由一个 master 管理进程，多个 worker 进程（处理工作）。基础架构设计如图所示：



其中。master 负责管理 worker 进程，worker 进程负责处理网络事件。整个框架被设计为一种依赖事件驱动、异步、非阻塞的模式。

如此设计的优点：

1.可以充分利用多核机器，增强并发处理能力。
2.多 worker 间可以实现负载均衡。
3.Master 监控并统一管理 worker 行为。在 worker 异常后，可以主动拉起 worker 进程，从而提升了系统的可靠性。并且由 Master 进程控制服务运行中的程序升级、配置项修改等操作，从而增强了整体的动态可扩展与热更的能力。

1.3 Nginx 入门

1.3.1 常用命令

Docker 下 nginx 容器的创建和启动(假如已有则无需再创建启动)

```
sudo docker run -p 80:80 --restart always --name nginx \
-v /usr/local/docker/nginx/:/etc/nginx/ \
-v /usr/local/docker/nginx/conf.d:/etc/nginx/conf.d \
-d nginx
```

说明:nginx 的启动必须在根目录中执行.

```
docker start nginx
docker restart nginx
docker stop nginx
docker exec -it nginx bash
nginx -v # 查看 nginx 版本 （docker 中需要在容器内部执行）
service nginx reload 重新加载配置文件(docker 中需要在容器内部执行)
```

1.3.2 核心配置文件

Nginx 的核心配置为 conf 目录下的 nginx.conf，其初始默认配置如下：

```
user  nginx;

worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
```

```
    }


    http {
        include       /etc/nginx/mime.types;
        default_type  application/octet-stream;

        log_format  main  '$remote_addr - $remote_user [$time_local]
"$request" '
                          '$status $body_bytes_sent "$http_referer" '
                          '"$http_user_agent" "$http_x_forwarded_for"';

        access_log  /var/log/nginx/access.log  main;

        sendfile        on;
        #tcp_nopush     on;

        keepalive_timeout  65;

        #gzip  on;

        include /etc/nginx/conf.d/*.conf; #嵌套
    }
```

其中，nginx 的配置有三部分构成，在 docker 环境中 nginx 采用了嵌套加载方式，即主配置在 /etc/nginx/nginx.conf 中，然而平时用到的 server 配置在 /etc/nginx/conf.d 中，在主配置中见 include 指令部分，在 conf.d 目录下默认会有一个 default.conf 文件，这部分配置文件就是基本的 server 配置。无论采用怎样的配置方式，nginx.conf 都只有这三部分构成，例如：

- 全局块：配置文件开始到 events 中间的部分内容，主要是结合硬件资源进行配置
- events 块：这块主要是网络配置相关内容，硬件性能好，连接数可以配置更多
- http 块:nginx 配置中最核心部分，可以配置请求转发，负载均衡等。

1.3.3 入门配置案例

第一步：准备 web 服务，并打成 jar 包(例如 tomcat8901.jar)

```
@RestController
public class HelloController {

    @Value("${server.port}")
    private String port;

    //要求动态获取真实服务器端口号
    @RequestMapping("/hello")
    public String doSayHello() {

        return "server:"+port+" say hello ";
```

```
    }
}
```

第二步:将这个 jar 包扔到 linux 宿主机上一份,假设地址为/home/servers

第三步:基于 jdk:8 镜像文件,运行 jar 文件

```
docker run -d -p 8901:8901 --name tomcat8901 -v /home/servers:/usr/sca
jdk:8 java –jar /usr/sca/tomcat8901.jar
```

服务启动以后,可通过 docker container logs tomcat8901 检查日志,也可以直接在宿主机上通过 http://localhost:8901/hello 进行访问测试

第四步：配置请求转发。

配置 nginx 实现请求的转换，在 docker 环境下可编辑 /usr/local/default/nginx/conf.d/目录中的 default.conf 文件，详细配置请求转发见红色代码如下：

```
server {
    listen       80;
    listen  [::]:80;
    server_name  localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location / {
       proxy_pass http://192.168.174.130:8901;
       #root   /usr/share/nginx/html;
       #index  index.html index.htm;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}
# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #fastcgi_param  SCRIPT_FILENAME /scripts$fastcgi_script_name;
```

```
        #    include       fastcgi_params;
        #}

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #    deny  all;
        #}
    }
```

第三步：配置完以后重新启动 nginx。

```
docker restart nginx
```

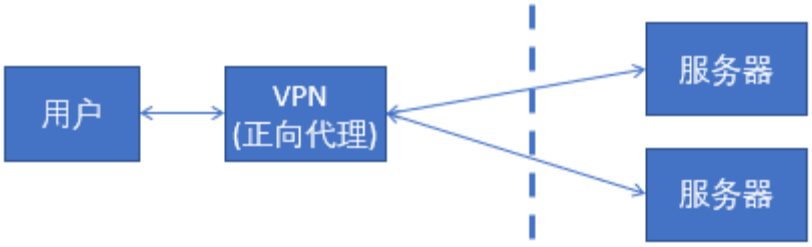
第四步：对资源进行访问检测请求转发实现。

http://192.168.174.130:80/hello

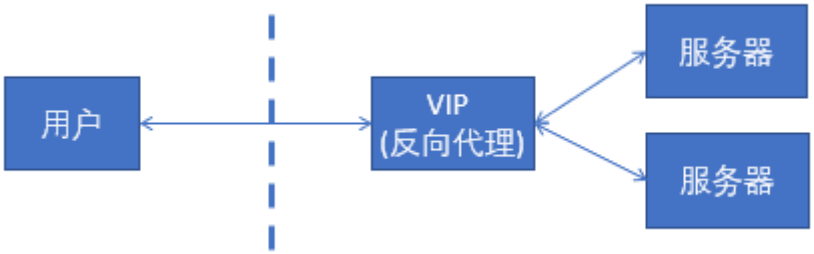
1.4 小节面试分析

- 如何理解正向和反向代理？

  正向代理即是客户端代理,服务端不知道实际发起请求的客户端.如图所示：



  反向代理即是服务端代理,客户端不知道实际提供服务的服务端 如图所示：



这两者的区别,总结起来一句话:正向代理隐藏真实客户端,反向代理隐藏真实服务端。正向代理代理客户端， 反向代理代理服务器。

2 Nginx 最佳实践分析

2.1负载均衡实现

2.1.1 业务说明

需要搭建 tomcat 服务器集群,共同抗击高并发.这时需要使用反向代理服务器.同时配置负载均衡.

2.1.2 集群搭建

说明:准备 3 台 tomcat 服务器.端口号分别为 8901/8902/8903.

2.1.3 启动集群

```
docker run -d -p 8901:8901 --name tomcat8901 -v /root/servers:/usr/sca
jdk:8 java –jar /usr/sca/tomcat8901.jar
```

```
docker run -d -p 8902:8902 --name tomcat8902 -v /root/servers:/usr/sca
jdk:8 java –jar /usr/sca/tomcat8902.jar
```

```
docker run -d -p 8903:8903 --name tomcat8903 -v /root/servers:/usr/sca
jdk:8 java –jar /usr/sca/tomcat8902.jar
```

2.2负载均衡策略

2.2.1 轮询策略

说明:根据配置文件的顺序,依次访问服务器.

```
#配置 web 服务集群 默认是轮询策略,打开 conf.d 目录下 default.conf 文件,进行
```

负载均衡配置,代码参考如下:

```
upstream gateways{
    server 192.168.174.130:8901;
    server 192.168.174.130:8902;
    server 192.168.174.130:8903;
}

#配置后台管理服务器
server {
    listen 80;
    server_name localhost;
    location  / {
        #实现 http 请求的转发
        proxy_pass http://gateways;

    }
}
```

2.2.2 权重策略

说明:可以为某些服务器添加权重,让该服务器更多的为用户提供服务

```
#配置 windows 集群 默认是轮循策略 权重

upstream gateways{
    server 192.168.174.130:8901 weight=1;
    server 192.168.174.130:8902 weight=2;
    server 192.168.174.130:8903 weight=6;
}
```

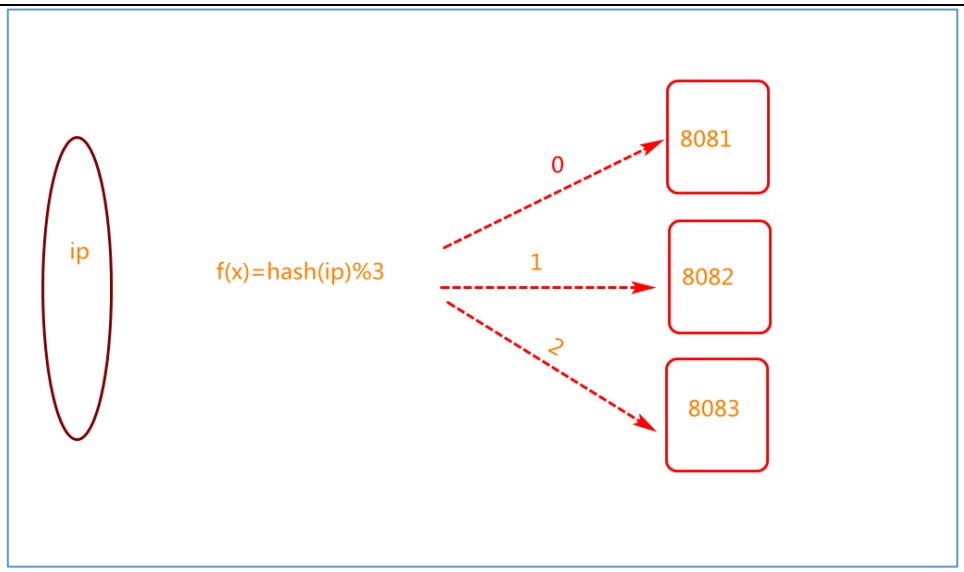
2.2.3 IPHASH 策略(了解)

问题说明:如果采用集群的部署,如果做敏感操作时,要求用户必须登录.但是由于 nginx 实现了负载均衡的操作,导致用户的 Session 数据不同共享.从而导致用户频繁登录.用户体验较差.

问题: nginx 实现了 tomcat 负载均衡. 导致用户每次访问都是不同的服务器.

解决方案: 能否让用户每次访问同一台服务器 IPHASH 策略

IPHASH 调用原理 如图所示:



配置如下:

```
#配置 windows 集群 默认是轮询策略 权重
    upstream geteways {
        ip_hash;
        server 192.168.227.131:10001 weight=6;
        server 192.168.227.131:10002 weight=3;
        server 192.168.227.131:10003 weight=1;
    }
```

- **IPHASH 存在的问题**

1. IPHASH 如果一旦服务器出现异常,导致业务失效.
2. 可能会出现负载不均的现象.负载有高有低(可在测试中试用).

一般不会使用 IPHASH,一般在测试中使用.

2.3NGINX 常用属性

2.3.1 Down 属性

说明:如果服务器宕机,可以在配置文件中标识为 down.这样以后不会再访问故障机.

```
upstream geteways {
        #ip_hash;
        server 192.168.227.131:10001 down;
        server 192.168.227.131:10002;
        server 192.168.227.131:10003;
}
```

2.3.2 BACKUP 设计

说明:备用机设置,正常情况下该服务器不会被访问.当主机全部宕机或者主机遇忙时,该服务器才会访问.

```
upstream geteways {
        #ip_hash;
        server 192.168.227.131:10001 down;
        server 192.168.227.131:10002:8082;
        server 192.168.227.131:10003 backup;
}
```

2.3.3 宕机服务器高可用实现

说明:当服务器宕机时,如果访问时的失败次数达到最大失败次数,则标识为 down.自动完成.在一定的**周期**之内,如果服务器恢复正常,则还会尝试访问故障机.

max_fails=1 最大的失败次数

fail_timeout=60s; 设定周期为 60 秒

```
upstream geteways {
        #ip_hash;
        server 192.168.227.131:10001 max_fails=1 fail_timeout=60s;
        server 192.168.227.131:10002 max_fails=1 fail_timeout=60s;
        server 192.168.227.131:10003 max_fails=1 fail_timeout=60s;
}
```

2.4 Nginx 面试问题分析

- **为什么不采用多线程模型管理连接?**

1) 采用独立的进程，可以让互相之间不会影响。一个进程异常崩溃，其他进程的服务不会中断，提升了架构的可靠性。
2) 进程之间不共享资源，不需要加锁，所以省掉了锁带来的开销。

- **为什么不采用多线程处理逻辑业务?**

1) 进程数已经等于核心数,再新建线程处理任务,只会抢占现有进程,增加切换代价。
2) 作为接入层，基本上都是数据转发业务，网络 IO 任务的等待耗时部分，已经被处理为非阻塞/全异步/事件驱动模式，在没有更多 CPU 的情况下，再利用多线程处理，意义不大。并且如果进程中有阻塞的处理逻辑，应该由各个业务进行解决，比如 openResty 中利用了 Lua 协程，对阻塞业务进行了优化。

3 总结（Summary）