一、知识点

1. Nginx 基本概念

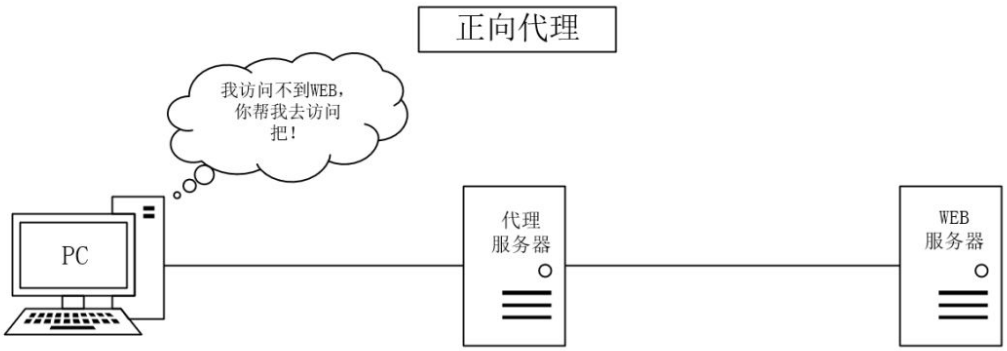
1.1 Nginx 是什么，做什么事情

- **Nginx 是一个高性能的HTTP和反向代理服务器，特点是占用内存少，并发能力强，在同类型的网页服务器中表现较好**
- **Nginx 专为性能优化而开发，性能是其最重要的考量，实现上非常注重效率，能经受高负载的考验，有报告表明能支持高达50000个并发连接数**

1.2 反向代理

- **正向代理**

  - 是一个位于客户端和目标服务器之间的服务器(代理服务器)，为了从目标服务器取得内容，客户端向代理服务器发送一个请求并指定目标，然后代理服务器向目标服务器转交请求并将获得的内容返回给客户端。



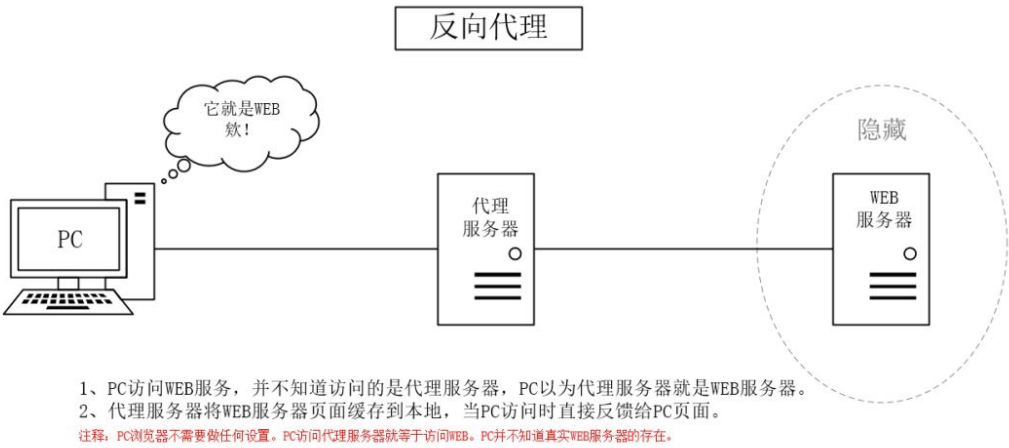
1、PC无法直接访问WEB服务器，但代理服务器可以访问。
2、代理服务器帮助PC请求页面并缓存到本地，并将页面返回给PC。
注释，PC只需要浏览器设置代理服务器IP和端口即可。PC知道代理服务器和WEB服务器的存在。

正向代理，其实是"代理服务器"代理了"客户端"，去和"目标服务器"进行交互

  - 用途
    - **突破访问限制**
    - **提高访问速度**
    - **隐藏客户端真实IP**

- **反向代理**

  - 是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

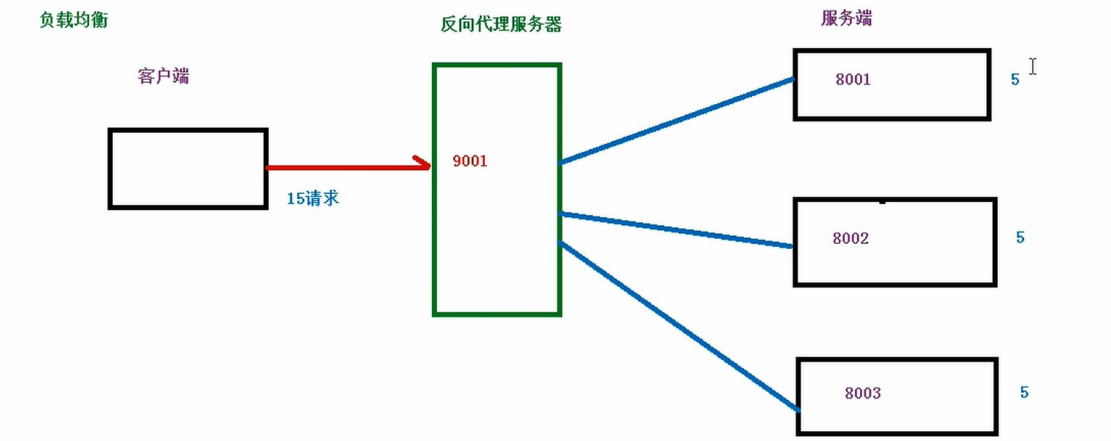


反向代理，其实是"代理服务器"代理了"目标服务器"，去和"客户端"进行交互。

- 用途
  - **隐藏服务器真实IP**
  - **负载均衡**
  - **提高访问速度**
  - **提供安全保障**

1.3 负载均衡

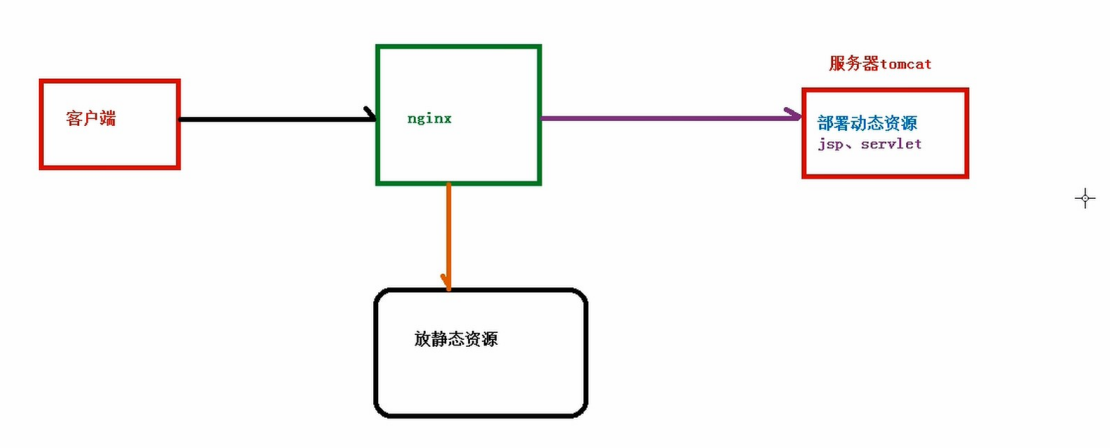
指将负载（工作任务）进行平衡、分摊到多个操作单元上进行运行，例如FTP服务器、Web服务器、企业核心应用服务器和其它主要任务服务器等，从而协同完成工作任务。



单个服务器解决不了，增加服务器数量，将请求分发到各个服务器上，将原先集中请求单个服务器的情况改为将请求分发到多个服务器

1.4 动静分离

为了加快网站的解析速度，可以**把动态页面和静态页面由不同的服务器来解析**，加快解析速度，降低原来单个服务器的压力。



2. Nginx 安装、常用命令和配置文件

2.1 在Linux 系统中安装 Nginx

- 使用远程连接工具连接到远程Linux服务器
- nginx相关依赖安装
  - 安装pcre依赖[普通文件安装]

    1. 把安装压缩文件放到linux系统中
    2. 解压缩文件 [tar -xvf xxx.tar]
    3. 进入解压后文件目录，执行 ./configure
    4. 使用make && make install
    5. 安装之后，使用命令，查看版本号 pcre-config --version
  - 安装其他依赖

    ```
    yum -y install make zlib zilb-devel gcc-c++ libtool openssl openssl-devel
    ```
  - 安装 nginx

    1. 解压缩nginx-xx.tar.gz 包 [tar -xvf nginx-xx.tar]
    2. 进入解压后目录，执行

       ```
       ./configure
       ```
    3. 执行目录

       ```
       make && make install
       ```

    安装完成之后，在usr 多出来一个文件夹 local/nginx，在nginx有sbin有启动脚本
  - 测试

    1. 进入 nginx 的sbin目录，执行 ./nginx
    2. 页面访问 localhost:80
    3. linux防火墙问题解决方案
       - 查看开放的端口号\

         ```
         firewall-cmd --list-all
         ```

- 设置开发的端口号

```
firewall-cmd --add-service=http -permanent

sudo firewall-cmd --add-port=80/tcp --permanent
```

- 重启防火墙

```
firewall-cmd -reload
```

2.2 Nginx 常用命令

1. 使用 nginx 操作命令前提条件是，必须进入 nginx 的目录

```
cd /usr/local/nginx/sbin
```

2. 查看 nginx 版本号

```
./nginx -v
```

3. 启动 nginx

```
./nginx
```

4. 关闭 nginx

```
./nginx -s stop
```

5. 重新加载 nginx

```
./nginx -s reload
```

2.3 Nginx 配置文件

1. **nginx 配置文件位置**

```
cd /usr/local/nginx/conf/nginx.conf
```

2. **nginx 配置文件组成**
   - **全局块**

     从配置文件开始到 events 块之间的内容，主要会设置一些影响服务器整体运行的配置指令，主要包括配置运行 **nginx 服务器的用户（组）、允许生成的 worker process数、进程 PID 存放路径，日志存放路径和类型以及配置文件的引入**等。

```
#user  nobody;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;
```

worker_processes：这是nginx 服务器并发处理服务的关键配置，worker_processes值越大，可以支持的并发处理量也越多，但是会受到硬件、软件等设备的制约

- **events 块**

  events 块涉及的指令主要影响 nginx 服务器与用户的网络连接，常用的设置包括**是否开启对多 work process 下的网络连接进行序列化、是否允许同时接收多个网络连接、选取哪种事件驱动模型来处理连接请求、每个word process 可以同时支持的最大连接数**等。

  ```
  events {
      worker_connections  1024;
  }
  ```

  worker_connections：表示每个work process 支持的最大连接数；这部分的配置对 nginx 的性能影响较大，在实际中应该灵活配置

- **http 块**

  这算是 nginx 服务器配置中最频繁的部分，代理、缓存和日志定义等绝大多数功能和第三方模块的配置都在这里

  ```
  http {
      include       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  logs/access.log  main;

      sendfile        on;
      #tcp_nopush     on;

      #keepalive_timeout  0;
      keepalive_timeout  65;

      #gzip  on;

      server {
          listen       80;
          server_name  localhost;
  ```

```
        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        location / {
            root   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   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;
        #}
    }


    # another virtual host using mix of IP-, name-, and port-
based configuration
    #
    #server {
    #    listen       8000;
    #    listen       somename:8080;
    #    server_name  somename  alias  another.alias;

    #    location / {
```

```
    #        root   html;
    #        index  index.html index.htm;
    #    }
    #}


    # HTTPS server
    #
    #server {
    #    listen       443 ssl;
    #    server_name  localhost;

    #    ssl_certificate      cert.pem;
    #    ssl_certificate_key  cert.key;

    #    ssl_session_cache    shared:SSL:1m;
    #    ssl_session_timeout  5m;

    #    ssl_ciphers  HIGH:!aNULL:!MD5;
    #    ssl_prefer_server_ciphers  on;

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

}
```

需要注意的是：http 块也可以包括 http 全局块，server 块

- **http 全局块**

  > http 全局块配置的指令包括**文件引入、MIME-TYPE定义、日志自定义、连接超时时间、单连接请求数上限**等

- **server 块**

  > 这块和虚拟主机有密切关系，虚拟主机从用户角度看，和一台独立的硬件主机是完全一样的，该技术的产生是为了节省互联网服务硬件成本。
  >
  > 每个 http 块可以包括多个 server 块，而每个 server 块就相当于一个虚拟主机
  >
  > 而每个 server 块也分为全局 server 块，以及可以同时包含多个 location 块
  >
  > - **全局 server 块**
  >
  >   > 最常见的配置是本虚拟机主机的监听配置和本虚拟机主机的名称或 IP 配置
  >
  > - **location**
  >
  >   > 一个 server 块可以配置多个 location 块
  >   >
  >   > 这块的主要作用是基于 nginx 服务器收到的请求字符串（例如 server_name/uri-string），**对虚拟主机名称（也可以是 IP 别名）之外的字符串（例如：前面的 /uri-string）进行匹配、对特定的请求进行处理，地址定向、数据缓存和应答控制等功能，还有许多**

第三方模块的配置也在这里进行。

3. Nginx 配置实例 - 反向代理

1. 实现效果

    1. 打开浏览器，在浏览器地址栏输入地址www.123.com ，跳转到 linux系统 tomcat 主页面中

2. 准备工作

    1. 在linux 系统中安装 tomcat，使用默认端口 8080

       解压tomcat文件，启动tomcat ./startup.sh

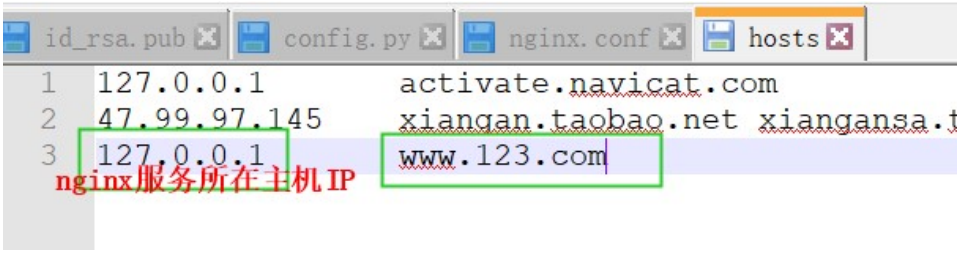
       防火墙开启8080端口

       ```
       firewall-cmd --add-port=8080/tcp --permanent
       firewall-cmd -reload
       # 查看已经开放的端口
       firewall-cmd --list-all
       ```

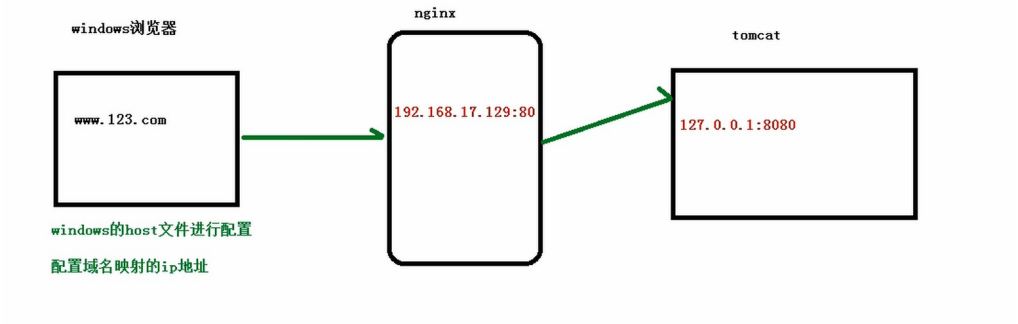
       浏览器访问tomcat地址 localhost:8080

    2. 配置域名

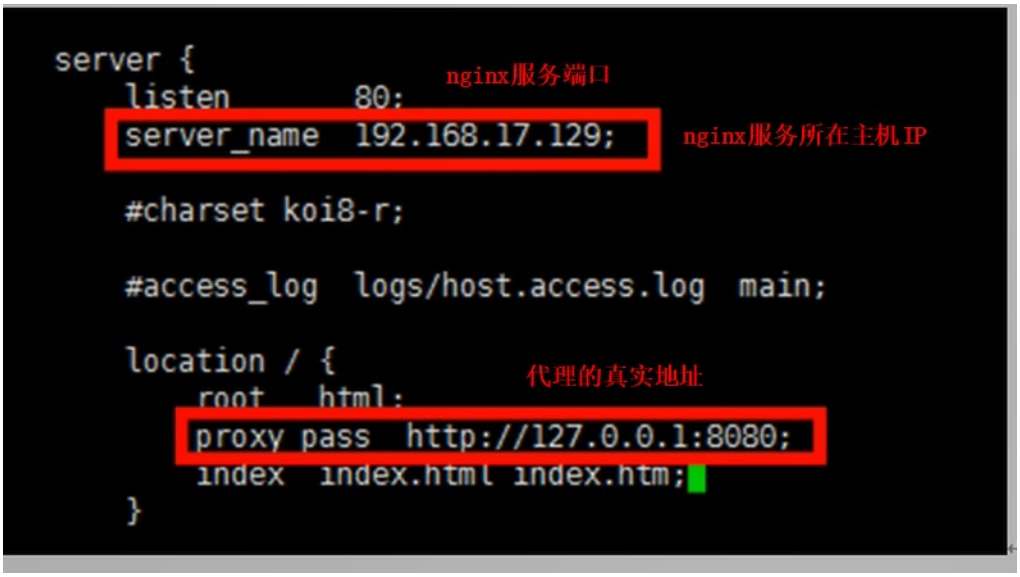
       本地电脑 hosts文件添加 映射

3. 访问过程分析

4. 反向代理配置



重启服务，或者 ./nginx -s reload

5. 测试

浏览器输入 访问 www.123.com 将跳转到tomcat主页

- 实现效果

1. 浏览器输入不同的url，跳转到不同的地址

   输入 http://ip:port/edu:9001 跳转到 127.0.0.1:8080

   输入 http://ip:port/vod:9001 跳转到 127.0.0.1:8081

2. 在2个tomcat下分别创建edu和vod文件夹，里面存放a.html文件

- 准备工作

1. linux服务器准备2个tomcat服务器，启动默认8080，

   修改另外一个tomcat的server.xml文件，修改端口为8081并启动服务

- 访问过程分析

输入 http://ip:port/edu:9001 跳转到 127.0.0.1:8080

输入 http://ip:port/vod:9001 跳转到 127.0.0.1:8081

- 反向代理配置

```
    server {
        listen       9001;
        server_name  127.0.0.1;

        location ~ /edu/ {
            proxy_pass http://127.0.0.1:8080;
        }

        location ~ /vod/ {
            proxy_pass http://127.0.0.1:8081;
        }
    }
```

- 测试

> 浏览器输入地址可以正常访问：
>
> http://www.123.com:9001/edu/a.html
>
> http://www.123.com:9001/vod/a.html

location 配置说明：

```
location [ = | ~ | ~* | ^~ ] uri {

}
```

- = 表示uri严格匹配
- ~ 表示正则匹配，区分大小写
- ~* 表示正则匹配，不区分大小写
- ^~ 表示用于不含正则表达式的 uri前，要求nginx服务器找到识别 uri 和请求字符串匹配度最高的 location后，立即使用此 location处理请求，而不在使用location块中的正则uri和请求字符串做匹配

4. Nginx 配置实例 - 负载均衡

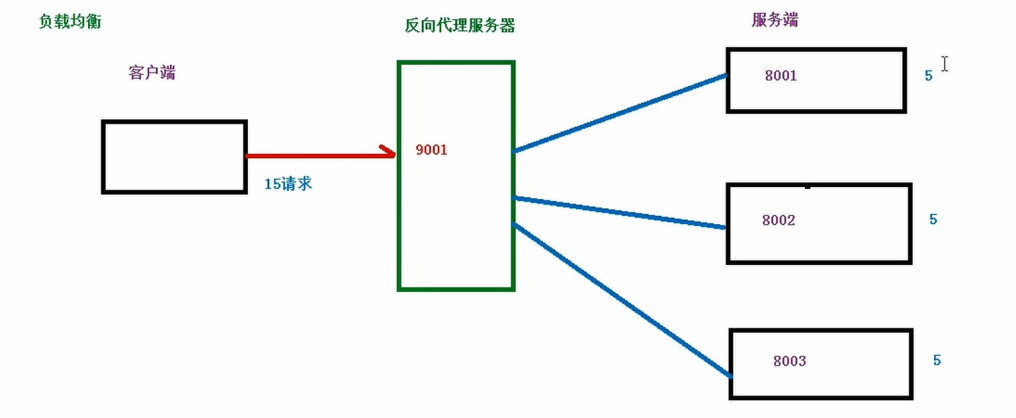
1. 实现效果

   > 浏览器输入 http://ip:port/edu/a.html，负载均衡效果，平均8080和8081端口中

2. 准备工作

   > 准备2台tomcat服务器，一台8080，一台8081
   >
   > 两台服务器webapps下创建doc文件夹，页面a.html，用于测试

3. 访问过程分析

4. 负载均衡配置

```
# 负载均衡
upstream myserver {
    server 127.0.0.1:8080;
    server 127.0.0.1:8081;
}
server {
    listen       9001;
    server_name  127.0.0.1;

    location / {
        proxy_pass http://myserver;
    }
}
```

5. 测试

> 浏览器输入 http://www.123.com:9001/demo/a.html ，查看网页内容

nginx 分配服务器的策略：

- 轮询（默认）

  每个请求按照时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉了，能自动剔除

- 权重（weight）

  weight表示权重，默认为1；值越高被分配的客户端越多

  ```
  # 负载均衡
  upstream myserver {
      server 127.0.0.1:8080 weight=10;
      server 127.0.0.1:8081 weight=20;
  }
  ```

- ip_hash

  每个请求按照访问ip的hash结果分配，这样每个客户端访问固定的一个后端服务器，可以解决session的问题

  ```
  # 负载均衡
  upstream myserver {
      ip_hash
      server 127.0.0.1:8080;
      server 127.0.0.1:8081;
  }
  ```

- fair

  按后端服务器的响应时间来分配请求，响应时间快的优先分配

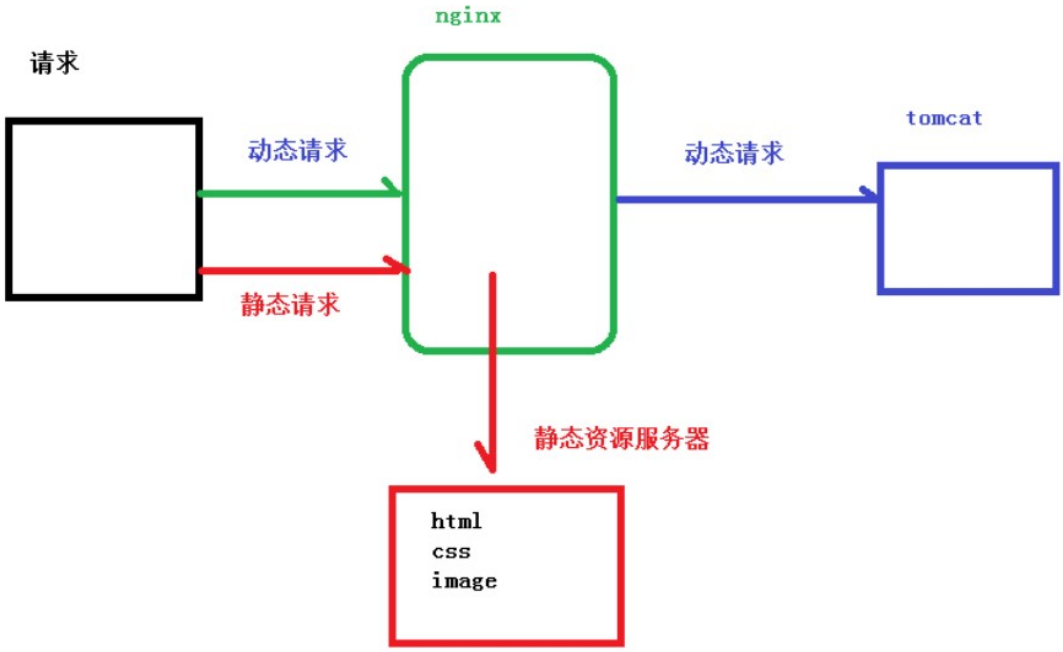
  ```
  # 负载均衡
  upstream myserver {
      server 127.0.0.1:8080;
      server 127.0.0.1:8081;
      fair;
  }
  ```

5. Nginx 配置实例 - 动静分离

nginx 动静分离简单来说就是把动态跟静态请求分开，不能理解成只是单纯的把动态页面和静态页面物理分离，严格意义上说应该是动态请求与静态请求分开，可以理解成使用 nginx 处理静态页面，tomcat 处理动态页面，动静分离从目前实现角度来讲大致分为两种：

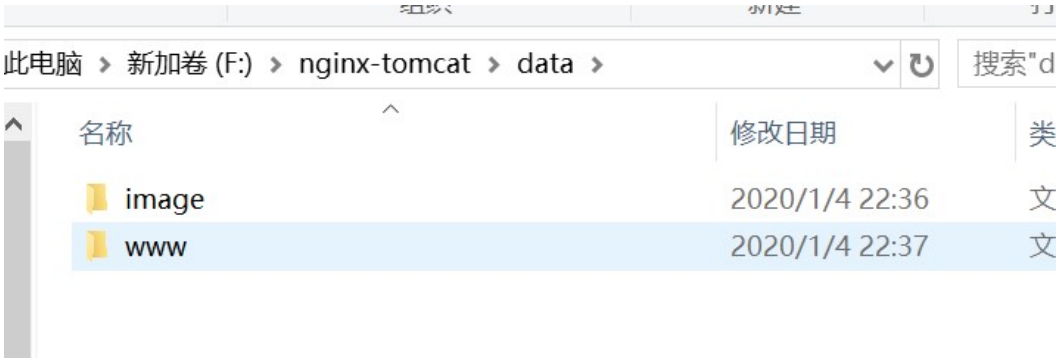
1. 纯粹把静态文件独立成单独的域名，放在独立的服务器上，也是目前主流推崇的方案
2. 将动态跟静态文件混合在一起发布，通过 nginx 来分开

通过 location 指定不同的后缀名实现不同的请求转发。通过 expires 参数设置，可以使浏览器缓存过期时间，减少与服务器之前的请求和流量。具体 Expires 定义：是给一个资源设定一个过期时间，也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可，所以不会产生额外的流量。此种方法非常适合不经常变动的资源。（如果经常更新的文件，不建议使用 Expires 来缓存），我这里设置 3d，表示在这 3 天之内访问这个 URL，发送一个请求，比对服务器该文件最后更新时间没有变化，则不会从服务器抓取，返回状态码 304，如果有修改，则直接从服务器重新下载，返回状态码 200。



1. 准备工作

   在linux系统中准备静态资源，用于进行访问



2. 静态资源配置

```
# 动静分离
server {
    listen       9003;
    server_name  127.0.0.1;
```

```
        location /www/ {
            root F:/nginx-tomcat/data;
        }

        location /image/ {
            root F:/nginx-tomcat/data;
            autoindex on;
        }
    }
```

3. 测试

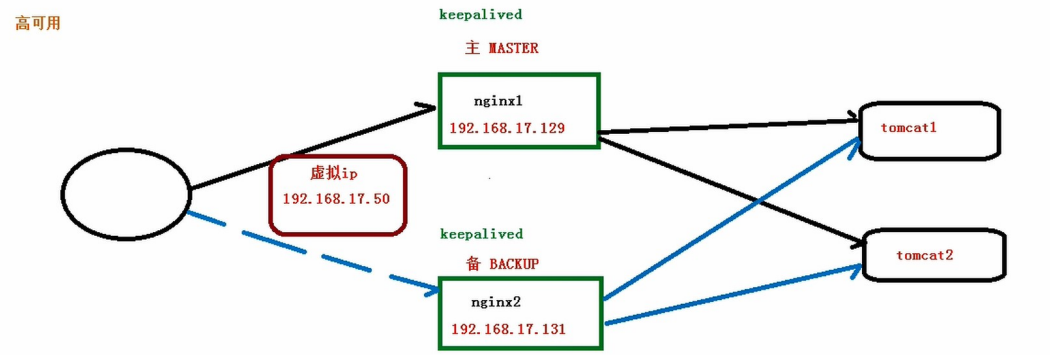
   浏览器输入：

   http://ip:port/image

   http://ip:port/www/a.html

6. Nginx 配置高可用集群

1. 什么是 nginx 高可用



   - 需要两台 nginx 服务器
   - 需要 keepalived
   - 需要虚拟 IP

2. 配置高可用的准备工作
   - 需要两台服务器 192.168.17.129 和 192.168.17.131
   - 在两台服务器安装 nginx [见上面步骤]
   - 在两台服务器安装 keepalived

```
yum install keepalived -y
rpm -qa keepalived
```

     安装之后，在 etc 里面生成目录 keepalived，有文件 keepalived.conf

3. 完成高可用配置（主从配置）
   - 修改 /etc/keepalived/keepalived.conf配置文件

```
global_defs {
   notification_email {
     acassen@firewall.loc
     failover@firewall.loc
     sysadmin@firewall.loc
   }
   notification_email_from      Alexandre.Cassen@firewall.loc
   smtp_server 192.168.17.129
```

```
   smtp_connect_timeout 30
   # 访问到主机，在 /etc/hosts 文件中进行配置 127.0.0.1
LVS_DEVELMASTER
   router_id LVS_DEVELMASTER
}

vrrp_script_chk_http_port{
    # 检测脚本位置
    script "/usr/local/src/nginx_check.sh"
    # 检测脚本执行的间隔，每2s检测一次
    interval 2
    # 设置当前服务器的权重
    weight -20
}

vrrp_instance VI_1 {
    # 主机使用MASTER，备份服务器使用BACKUP
    state MASTER
    # 网卡，使用ifconfig查看
    interface eth0
    # 主、备机的virtual_router_id 必须相同
    virtual_router_id 51
    # 主、备机取不同的优先级，主机值较大，备份机值较小
    priority 100
    # 心跳检测 ，每隔1s检测一次
    advert_int 1
    # 权限校验
    authentication {
        # 权限方式-密码
        auth_type PASS
        # 密码值
        auth_pass 1111
    }
    # 虚拟IP
    virtual_ipaddress {
        # VRRP H 虚拟地址，最终使用的IP
        192.168.17.50
    }
}
```

- 在/usr/local/src 添加检测脚本

```
#!/bin/bash
A=`ps -C nginx -no-header |wc -l`
if [ $A -eq 0 ];then
    # nginx 启动脚本的位置
    /usr/local/nginx/sbin/nginx
    sleep 2
    if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then
        killall keepalived
    fi
fi
```

- 把两台服务器上的 nginx 和 keepalived 启动

  先启动 nginx：./nginx

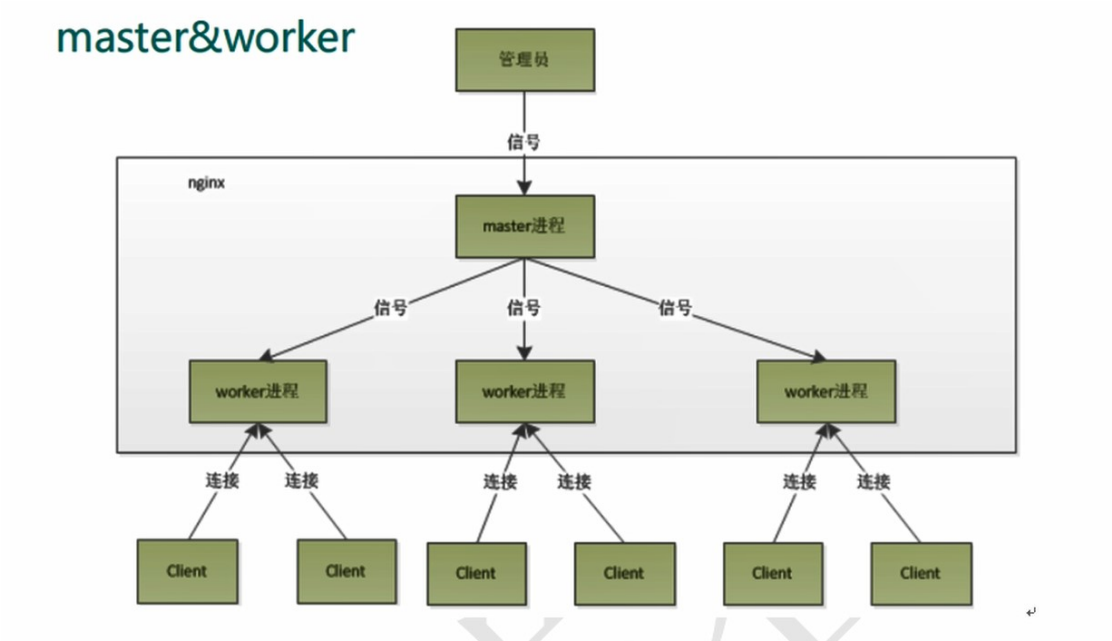
  后启动keepalived：systemctl start keepalived.service

- 测试

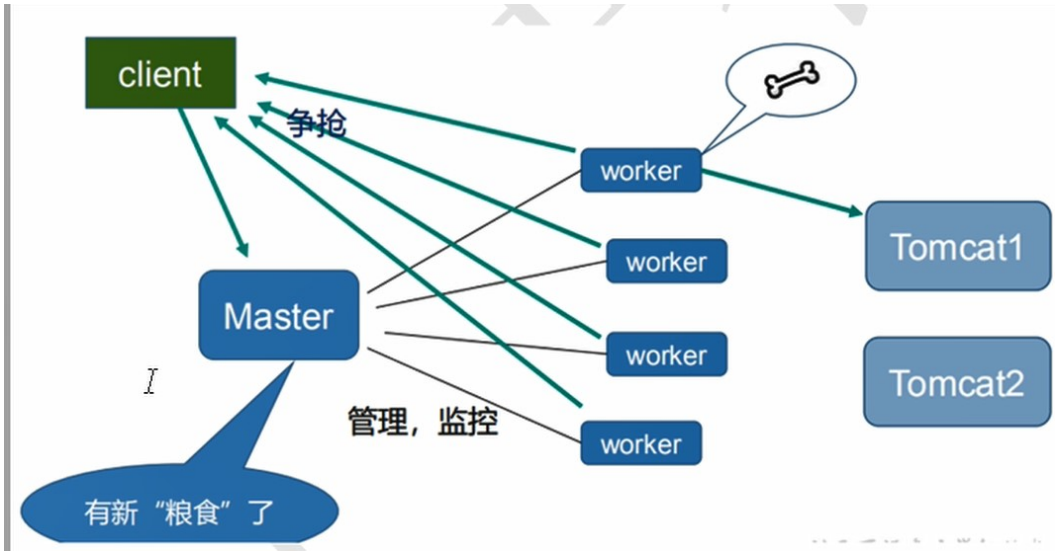
  浏览器输入 192.168.17.50

  停止其中一台 nginx服务器，再此输入 192.168.17.50 查看是否正常访问

7. Nginx 原理



- **master 和 worker**
- **worker 如何进行工作的**



- 一个master 和 多个 worker 的好处
  - 可以使用 nginx -s reload 热部署，利于 nginx进行热部署
  - 每个 worker 是一个独立的进程，如果其中一个 worker 出现问题，其他的 worker 是独立的，继续进行争抢，实现请求过程，不会造成服务中断
- 设置多少个worker

  worker 数和服务器 CPU 合数相等是最为适宜的

- 连接数 worker_connection

  问题：发送请求，占用了worker的几个连接数?

  > 2 或者 4个

> 问题：nginx 有一个master，有四个worker，每个worker支持最大的连接数1024，支持的最大并发数是多少？
>
> > 普通的静态访问最大并发数是：worker_connections * worker_processes/2
> >
> > 如果是HTTP作为反向代理来说，最大并发数应该是worker_connections * worker_processes/4

2020-01-王钦笔记