正向代理与反向代理

- 正向代理
  - 既是客户端代理，代理客户端，服务端不知道实际发送请求的客户端
  - 作用：
    - vpn访问谷歌
    - 可以做缓存，加速访问资源
    - 对客户端访问授权，上网进行认证
    - 代理可以记录用户访问记录（上网行为管理），对外隐藏用户信息
- 反向代理
  - 即是服务端代理, 代理服务端, 客户端不知道实际提供服务的服务端
  - 作用：
    - 保证内网的安全，阻止web攻击。大型网站通常将反向代理作为公网访问地址，Web服务器是内网
    - 负载均衡，通过反向代理服务器来优化网站的负载

Nginx显示默认首页过程解析

server.listen监听80端口，如果有访问80端口的请求进来后，通过server.server_name，server.location.root映射到相应目录。

Nginx进程模型

- 1. 同步阻塞（Blocking I/O）： 客户端发送请求给服务端，此时服务端处理任务时间很久，则客户端则被服务端堵塞了，所以客户端会一直等待服务端的响应，此时客户端不能做 事，服务端也不会接受其他客户端的请求。这种通信机制比较简单粗暴，但是效率不高。
- 2. 同步非阻塞(New I/O)： 客户端发送请求给服务端，此时服务端处理任务时间很久，这个时候虽然客户端会一直等待响应，但是服务端可以处理其他的请求，过一会回来 的。这种方式很高效，一个服务端可以处理很多请求，不会在因为任务没有处理完而堵着，所以这是非阻塞的。
- 3. 异步阻塞： 客户端发送请求给服务端，此时服务端处理任务时间很久，但是客户端不会等待服务器响应，它可以做其他的任务，等服务器处理完毕后再把结果 端，客户端得到回调后再处理服务端的响应。这种方式可以避免客户端一直处于等待的状态，优化了用户体验，其实就是类似于网页里发起的ajax异步请求。
- 4. 异步非阻塞（ Asynchronous I/O）(NIO2)： 客户端发送请求给服务端，此时服务端处理任务时间很久，这个时候的任务虽然处理时间会很久，但是客户端可以做其他的任务，因为他是异步 回调函数里处理响应；同时服务端是非阻塞的，所以服务端可以去处理其他的任务，如此，这个模式就显得非常的高效了。
- 请简述一下BIO/NIO/AIO之间的概念与区别：
  - NIO如果还拿烧开水来说，NIO的做法是叫一个线程不断的轮询每个水壶的状态，看看是否有水壶的状态发生了改变，从而进行下一步的操作
  - 有一排水壶在烧开水，BIO的工作模式就是， 叫一个线程停留在一个水壶那，直到这个水壶烧开，才去处理下一个水壶。
  - AIO对应到烧开水中就是，为每个水壶上面装了一个开关，水烧开之后，水壶会自动通知我水烧开了

Nginx配置详情

- nginx文件结构

```
########### 每个指令必须有分号结束。#################
#user administrator administrators; #配置用户或者组，默认为nobody nobody。
#worker_processes 2; #允许生成的进程数，默认为1
#pid /nginx/pid/nginx.pid; #指定nginx进程运行文件存放地址
error_log log/error.log debug; #制定日志路径，级别。这个设置可以放入全局块，http块，
server块，级别以此为：debug|info|notice|warn|error|crit|alert|emerg
events {
 accept_mutex on; #设置网路连接序列化，防止惊群现象发生，默认为on
 multi_accept on; #设置一个进程是否同时接受多个网络连接，默认为off
 #use epoll;  #事件驱动模型，select|poll|kqueue|epoll|resig|/dev/poll|eventport
 worker_connections 1024; #最大连接数，默认为512
}
http {
 include  mime.types; #文件扩展名与文件类型映射表
 default_type application/octet-stream; #默认文件类型，默认为text/plain
 #access_log off; #取消服务日志
 log_format myFormat '$remote_addr-$remote_user [$time_local] $request $status
$body_bytes_sent $http_referer $http_user_agent $http_x_forwarded_for'; # 自定义格
式
 # 日志格式
 # 参数名 参数意义
 # $remote_addr 客户端ip
 # $remote_user 远程客户端用户名，一般为：'-'
 # $time_local 时间和时区
 # $request 请求的url以及method
 # $status 响应状态码
 # $body_bytes_send 响应客户端内容字节数
 # $http_referer 记录用户从哪个链接跳转过来的
 # $http_user_agent 用户所使用的代理，一般来时都是浏览器
 # $http_x_forwarded_for 通过代理服务器来记录客户端的ip

 access_log log/access.log myFormat; #combined为日志格式的默认值
 sendfile on; #允许sendfile方式传输文件，默认为off，可以在http块，server块，location块。
 sendfile_max_chunk 100k; #每个进程每次调用传输数量不能大于设定的值，默认为0，即不设上限。
 keepalive_timeout 65; #连接超时时间，默认为75s，可以在http，server，location块。

 upstream mysvr {
  server 127.0.0.1:7878;
  server 192.168.10.121:3333 backup; #热备
 }
 error_page 404 https://www.baidu.com; #错误页
 server {
  keepalive_requests 120; #单连接请求上限次数。
  listen  4545; #监听端口
  server_name 127.0.0.1; #监听地址
  location ~*^.+$ {  #请求的url过滤，正则匹配，~为区分大小写，~*为不区分大小写。
   #root path; #根目录
   #index vv.txt; #设置默认页
   proxy_pass http://mysvr; #请求转向mysvr 定义的服务器列表
   deny 127.0.0.1; #拒绝的ip
   allow 172.18.5.54; #允许的ip
  }
 }
```

```
}
```

- 2.配置文件详情

```
#运行用户
user www-data;
#启动进程,通常设置成和cpu的数量相等
worker_processes 1;

#全局错误日志及PID文件
error_log /var/log/nginx/error.log;
pid    /var/run/nginx.pid;

#工作模式及连接数上限
events {
  use  epoll;        #epoll是多路复用IO(I/O Multiplexing)中的一种方式,但是仅用于linux2.6以上内核,可以大大提高nginx的性能
  worker_connections 1024;#单个后台worker process进程的最大并发链接数
  # multi_accept on;
}

#设定http服务器，利用它的反向代理功能提供负载均衡支持
http {
   #设定mime类型,类型由mime.type文件定义
  include    /etc/nginx/mime.types;
  default_type application/octet-stream;
  #设定日志格式
  access_log  /var/log/nginx/access.log;

  #sendfile 指令指定 nginx 是否调用 sendfile 函数（zero copy 方式）来输出文件，对于普通应用，
  #必须设为 on,如果用来进行下载等应用磁盘IO重负载应用，可设置为 off，以平衡磁盘与网络I/O处理速度，降低系统的uptime.
  sendfile    on;
  #tcp_nopush   on;

  #连接超时时间
  #keepalive_timeout 0;
  keepalive_timeout 65;
  tcp_nodelay    on;

  #开启gzip压缩
  gzip on;
  gzip_disable "MSIE [1-6]\.(?!.*SV1)";

  #设定请求缓冲
  client_header_buffer_size  1k;
  large_client_header_buffers 4 4k;

  include /etc/nginx/conf.d/*.conf;
  include /etc/nginx/sites-enabled/*;

  #设定负载均衡的服务器列表
```

```
 upstream mysvr {
#weigth参数表示权值，权值越高被分配到的几率越大
#本机上的Squid开启3128端口
server 192.168.8.1:3128 weight=5;
server 192.168.8.2:80 weight=1;
server 192.168.8.3:80 weight=6;
}


server {
#侦听80端口
  listen    80;
  #定义使用www.xx.com访问
  server_name www.xx.com;

  #设定本虚拟主机的访问日志
  access_log logs/www.xx.com.access.log main;

#默认请求
location / {
   root  /root;   #定义服务器的默认网站根目录位置
   index index.php index.html index.htm;  #定义首页索引文件的名称

   fastcgi_pass www.xx.com;
   fastcgi_param SCRIPT_FILENAME $document_root/$fastcgi_script_name;
   include /etc/nginx/fastcgi_params;
  }

# 定义错误提示页面
error_page  500 502 503 504 /50x.html;
  location = /50x.html {
  root  /root;
}

#静态文件，nginx自己处理
location ~ ^/(images|javascript|js|css|flash|media|static)/ {
  root /var/www/virtual/htdocs;
  #过期30天，静态文件不怎么更新，过期可以设大一点，如果频繁更新，则可以设置得小一点。
  expires 30d;
}
#PHP 脚本请求全部转发到 FastCGI处理. 使用FastCGI默认配置.
location ~ \.php$ {
  root /root;
  fastcgi_pass 127.0.0.1:9000;
  fastcgi_index index.php;
  fastcgi_param SCRIPT_FILENAME /home/www/www$fastcgi_script_name;
  include fastcgi_params;
}
#设定查看Nginx状态的地址
location /NginxStatus {
  stub_status      on;
  access_log       on;
  auth_basic       "NginxStatus";
  auth_basic_user_file conf/htpasswd;
}
#禁止访问 .htxxx 文件
location ~ /\.ht {
  deny all;
```

```
  }

   }
}
```

- 3.负载均衡

```
#设定http服务器，利用它的反向代理功能提供负载均衡支持
http {
   #设定mime类型,类型由mime.type文件定义
  include    /etc/nginx/mime.types;
  default_type application/octet-stream;
  #设定日志格式
  access_log  /var/log/nginx/access.log;

  #省略上文有的一些配置节点

  #。。。。。。。。。。

  #设定负载均衡的服务器列表
   upstream mysvr {
  #weigth参数表示权值，权值越高被分配到的几率越大
  server 192.168.8.1x:3128 weight=5;#本机上的Squid开启3128端口
  server 192.168.8.2x:80 weight=1;
  server 192.168.8.3x:80 weight=6;
  }

  upstream mysvr2 {
  #weigth参数表示权值，权值越高被分配到的几率越大

  server 192.168.8.x:80 weight=1;
  server 192.168.8.x:80 weight=6;
  }

  #第一个虚拟服务器
  server {
  #侦听192.168.8.x的80端口
    listen    80;
    server_name 192.168.8.x;

   #对aspx后缀的进行负载均衡请求
  location ~ .*\.aspx$ {

      root  /root;   #定义服务器的默认网站根目录位置
      index index.php index.html index.htm;  #定义首页索引文件的名称

      proxy_pass http://mysvr ;#请求转向mysvr 定义的服务器列表

      #以下是一些反向代理的配置可删除.

      proxy_redirect off;

      #后端的Web服务器可以通过X-Forwarded-For获取用户真实IP
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      client_max_body_size 10m;  #允许客户端请求的最大单文件字节数
```

```
      client_body_buffer_size 128k; #缓冲区代理缓冲用户端请求的最大字节数，
      proxy_connect_timeout 90; #nginx跟后端服务器连接超时时间(代理连接超时)
      proxy_send_timeout 90;     #后端服务器数据回传时间(代理发送超时)
      proxy_read_timeout 90;      #连接成功后，后端服务器响应时间(代理接收超时)
      proxy_buffer_size 4k;        #设置代理服务器（nginx）保存用户头信息的缓冲区大小
      proxy_buffers 4 32k;         #proxy_buffers缓冲区，网页平均在32k以下的话，这样设置
      proxy_busy_buffers_size 64k;  #高负荷下缓冲大小（proxy_buffers*2）
      proxy_temp_file_write_size 64k; #设定缓存文件夹大小，大于这个值，将从upstream服务器
传

    }

   }
}
```

惊群现象

- 一个网路连接到来，多个睡眠的进程被同事叫醒，但只有一个进程能获得链接，这样会影响系统性能。
- 解决：Nginx中规定同一时刻只能有唯一一个的worker进程监听Web端口，这样就不会发生惊群了，此时新连接事件只能唤醒唯一正在监听端口的worker进程。具体做法：：利用一把进程间锁

Nginx常见命令

- 验证配置是否正确: nginx -t
- 查看Nginx的详细的版本号：nginx -V
- 查看Nginx的简洁版本号：nginx -v
- 启动Nginx：start nginx
- 快速停止或关闭Nginx：nginx -s stop
- 正常停止或关闭Nginx：nginx -s quit
- 配置文件修改重装载命令：nginx -s reload

location 的匹配规则

- 空格： 默认匹配，普通匹配
  location / {
  root /home;
  }
- =： 精确匹配
  location = /imooc/img/face1.png {
  root /home;
  }
- ~*： 匹配正则表达式，不区分大小写
  #符合图片的显示
  location ~ .(GIF|jpg|png|jpeg) {
  root /home;
  }
- ~： 匹配正则表达式，区分大小写
  #GIF必须大写才能匹配到
  location ~ .(GIF|jpg|png|jpeg) {

root /home;
}

- ^~ ： 以某个字符路径开头
location ^~ /imooc/img {
root /home;
}

Gzip压缩作用

- 将响应报文发送至客户端之前可以启用压缩功能，这能够有效地节约带宽，并提高响应至客户端的速度。Gzip压缩可以配置http,server和location模块下。
- Nginx开启Gzip压缩参数说明：推荐压缩（gzip_types text/css;）

```
gzip on;                    #决定是否开启gzip模块，on表示开启，off表示关闭；
gzip_min_length 1k;         #设置允许压缩的页面最小字节(从header头的Content-Length中获取)，当返回内容大于此值时才会使用gzip进行压缩,以K为单位,当值为0时，所有页面都进行压缩。建议大于1k
gzip_buffers 4 16k;         #设置gzip申请内存的大小,其作用是按块大小的倍数申请内存空间,param2:int(k) 后面单位是k。这里设置以16k为单位,按照原始数据大小以16k为单位的4倍申请内存
gzip_http_version 1.1;      #识别http协议的版本,早起浏览器可能不支持gzip自解压,用户会看到乱码
gzip_comp_level 2;          #设置gzip压缩等级，等级越底压缩速度越快文件压缩比越小，反之速度越慢文件压缩比越大；等级1-9，最小的压缩最快 但是消耗cpu
gzip_types text/plain application/x-javascript text/css application/xml;     #设置需要压缩的MIME类型,非设置值不进行压缩，即匹配压缩类型
gzip_vary on;               #启用应答头"Vary: Accept-Encoding"

gzip_proxied off;
nginx做为反向代理时启用,off(关闭所有代理结果的数据的压缩),expired(启用压缩,如果header头中包括"Expires"头信息),no-cache(启用压缩,header头中包含"Cache-Control:no-cache"),no-store(启用压缩,header头中包含"Cache-Control:no-store"),private(启用压缩,header头中包含"Cache-Control:private"),no_last_modefied(启用压缩,header头中不包含
  "Last-Modified"),no_etag(启用压缩,如果header头中不包含"Etag"头信息),auth(启用压缩,如果header头中包含"Authorization"头信息)

gzip_disable msie6;
(IE5.5和IE6 SP1使用msie6参数来禁止gzip压缩 )指定哪些不需要gzip压缩的浏览器(将和User-Agents进行匹配),依赖于PCRE库

##############################################################################
#######################
#如下：修改nginx配置文件 /usr/local/nginx/conf/nginx.conf
[root@localhost ~]# vim /usr/local/nginx/conf/nginx.conf          #将以下配置放到nginx.conf的http{ ... }区域中

#修改配置为
gzip on;                        #开启gzip压缩功能
gzip_min_length 10k;            #设置允许压缩的页面最小字节数; 这里表示如果文件小于10个字节，就不用压缩，因为没有意义，本来就很小.
gzip_buffers 4 16k;             #设置压缩缓冲区大小，此处设置为4个16K内存作为压缩结果流缓存
gzip_http_version 1.1;          #压缩版本
gzip_comp_level 2;              #设置压缩比率，最小为1，处理速度快，传输速度慢；9为最大压缩比，处理速度慢，传输速度快; 这里表示压缩级别，可以是0到9中的任一个，级别越高，压缩就越小，节省了带宽资源，但同时也消耗CPU资源，所以一般折中为6
gzip types text/css text/xml application/javascript;       #制定压缩的类型,线上配置时尽可能配置多的压缩类型!
```

```
gzip_disable "MSIE [1-6]\.";        #配置禁用gzip条件，支持正则。此处表示ie6及以下不启用gzip（因为ie低版本不支持）
gzip vary on;    #选择支持vary header；改选项可以让前端的缓存服务器缓存经过gzip压缩的页面；这个可以不写，表示在传送数据时，给客户端说明我使用了gzip压缩
```

- 如下是线上常使用的Gzip压缩配置

```
 gzip  on;
 gzip_min_length  1k;
 gzip_buffers     4 16k;
 gzip_http_version 1.1;
 gzip_comp_level 9;
 gzip_types       text/plain application/x-javascript text/css application/xml
text/javascript application/x-httpd-php application/javascript application/json;
 gzip_disable "MSIE [1-6]\.";
 gzip_vary on;
```

- Nginx的Gzip压缩功能虽然好用，但是下面两类文件资源不太建议启用此压缩功能。
  - 图片类型资源 (还有视频文件)
    图片如jpg、png文件本身就会有压缩，所以就算开启gzip后，压缩前和压缩后大小没有多大区别，所以开启了反而会白白的浪费资源。（可以试试将一张jpg图片压缩为zip，观察大小并没有多大的变化。虽然zip和gzip算法不一样，但是可以看出压缩图片的价值并不大）
  - 大文件资源
    原因：会消耗大量的cpu资源，且不一定有明显的效果。

Nginx 跨域配置支持

```
#允许跨域请求的域，*代表所有
add_header 'Access-Control-Allow-Origin' *;
#允许带上cookie请求
add_header 'Access-Control-Allow-Credentials' 'true';
#允许请求的方法，比如 GET/POST/PUT/DELETE
add_header 'Access-Control-Allow-Methods' *;
#允许请求的header
add_header 'Access-Control-Allow-Headers' *;
```

Nginx 防盗链配置支持

```
#对源站点验证
valid_referers *.imooc.com;
#非法引入会进入下方判断
if ($invalid_referer) {
return 404;
}
```

Nginx 模块化体系

- nginx core
- http
- mail
- event module
- phase handler
- output filter

- upstream
- load balancer
- extend module

负载均衡算法

- 轮询
  轮询方式，依次将请求分配到各个后台服务器中，默认的负载均衡方式。
  适用于后台机器性能一致的情况。
  挂掉的机器可以自动从服务列表中剔除。
- 权重
  根据权重来分发请求到不同的机器中，指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况

```
upstream bakend {
server 192.168.0.14 weight=10;
server 192.168.0.15 weight=10;
}
```

- IP_hash
  根据请求者ip的hash值将请求发送到后台服务器中，可以保证来自同一ip的请求被打到固定的机器上，可以解决session问题。
  使用ip_hash的注意点：不能把后台服务器直接移除，只能标记 down

```
upstream bakend {
ip_hash;
server 192.168.0.14:88;
server 192.168.0.15:80;
}
```

- 一致性hash算法
  - 因为对于hash(k)的范围在int范围，所以我们将0~2^32作为一个环。其步骤为：
  - 1.求出每个服务器的hash（服务器ip）值，将其配置到一个 0~2^n 的圆环上（n通常取32）。
  - 2.用同样的方法求出待存储对象的主键 hash值，也将其配置到这个圆环上，然后从数据映射到的位置开始顺时针查找，将数据分布到找到的第一个服务器节点上。
- url_hash
  根据每次请求的url地址，hash后访问到固定的服务器节点。
- least_conn 最少连接数

```
upstream tomcats {
# url hash
hash $request_uri;
# 最少连接数
# least_conn
    server 192.168.1.173:8080;
    server 192.168.1.174:8080;
    server 192.168.1.175:8080;
}
server {
    listen 80;
    server_name www.tomcats.com;
    location / {
```

```
        proxy_pass http://tomcats;
    }
}
```

upstream的指令参数

- max_conns 节点的最大连接数
- slow_start 缓慢启动时间 商业版本使用
- down 节点下线（不可用状态）
- backup 备用节点（只有当前使用的全部节点挂掉后才会启用）
- max_fails 允许的最大失败数（达到最大失败数后，被认为宕机）
- fail_timeout 超过最大失败数后的等待时间（失败后，被认为宕机，等待时间后重新挂起）

Keepalived 提高吞吐量（慎用）

- keepalived： 设置长连接处理的数量
- proxy_http_version： 设置长连接http版本为1.1
- proxy_set_header： 清除connection header 信息

```
upstream tomcats {
    # server 192.168.1.173:8080 max_fails=2 fail_timeout=1s;
    server 192.168.1.190:8080;
    # server 192.168.1.174:8080 weight=1;
    # server 192.168.1.175:8080 weight=1;
    keepalive 32;
}
server {
    listen 80;
    server_name www.tomcats.com;
    location / {
        proxy_pass http://tomcats;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
```

Nginx控制浏览器缓存

```
# proxy_cache_path 设置缓存目录
# keys_zone 设置共享内存以及占用空间大小
# max_size 设置缓存大小
# inactive 超过此时间则被清理
# use_temp_path 临时目录，使用后会影响nginx性能
proxy_cache_path /usr/local/nginx/upstream_cache keys_zone=mycache:5m
max_size=1g inactive=1m use_temp_path=
location / {
proxy_pass http://tomcats;
# 启用缓存，和keys_zone一致
proxy_cache mycache;
# 针对200和304状态码缓存时间为8小时
proxy_cache_valid 200 304 8h;
}
```