NGINX配置文件

一、nginx配置文件配置：

1.各配置信息详解

```
user  nobody nobody;                # Nginx的worker进程运行用户以及用户组
worker_processes  1;                # Nginx开启的进程数，建议设置为等于CPU总核心数。
worker_processes auto;              # 多核心CPU设置。
cat  /proc/cpuinfo                 # 查看当前CPU 的信息。

worker_processes 4                  # 4核CPU

worker_cpu_affinity 0001 0010 0100 1000;
                    #亲核
                    #CPU亲核，设置工作进程与 CPU 绑定。
                    #指定了哪个cpu分配给哪个进程，一般来说不用特殊指定。如果一定要设的话，用0和1指定分配方式.
                    #这样设就是给1-4个进程分配单独的核来运行，出现第5个进程是就是随机分配了。
error_log    logs/error.log  info;        #定义全局错误日志定义类型，[debug|info|notice|warn|crit]
pid         logs/nginx.pid;                #指定进程ID存储文件位置
worker_rlimit_nofile    65535;            #一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n的值保持一致。
ulimit -a   查看所有    ulimit -n 查看 最大文件数量    ulimit -u   查看最大进程数量
每个用户打开的 文件数量    nofile      打开的进程数量   nprco
# 打开worker_rlimit_nofile同时也要将系统的limit打开
vim /etc/security/limits.conf
*                soft    nproc          65535
```

```
*                hard    nproc          65535
*                soft    nofile          65535
*                hard    nofile         65535
# 注意：设置了这个后，修改worker_connections值时，是不能超过worker_rlimit_nofile的这个值。
# 设定 http 的部分
http {
    include /etc/nginx/mime.types;       #文件扩展名与文件类型映射表,设定mime类型,类型由mime.type文件定义
    default_type  application/octet-stream;          #默认文件类型
    charset utf-8;               #服务器 默认编码

    include /etc/nginx/proxy.conf;               #反向代理配置，可以打开proxy.conf看看
    include /etc/nginx/fastcgi.conf;
#fastcgi配置，可以打开fastcgi.conf看看
# 生成日志的格式定义
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
                    #定义日志的格式。后面定义要输出的内容。
                                #1.$remote_addr 与$http_x_forwarded_for 用以记录客户端的ip地址；
                                #2.$remote_user 用来记录客户端用户名称；
                                #3.$time_local 用来记录访问时间与时区；
                                #4.$request          用来记录请求的url与http协议；
                                #5.$status          用来记录请求状态；
                                #6.$body_bytes_sent     记录发送给客户端文件主体内容大小；
                                #7.$http_referer 用来记录从那个页面链接访问过来的；
```

```
                                      #8.$http_user_agent
    记录客户端浏览器的相关信息

# 访问日志
    access_log  /var/log/nginx/access.log  main;
    sendfile    on;
# 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: 设置为on表示启动高效传输文件的模式。
# sendfile可以让Nginx在传输文件时直接在磁盘和tcp socket之间传输数据。
# 如果这个参数不开启，会先在用户空间（Nginx进程空间）申请一个buffer，用read函数把数据从磁盘读到内核cache，再从内核 cache读取到用户空间nginx 的buffer，再用write函数把数据从用户空间的buffer写入到内核的buffer，最后到tcp socket。
# 启这个参数后可以让数据不用经过用户buffer。
# 速web服务器在传输文件方面的效率。
    autoindex on;
# 开启目录列表访问（网站以树目录展示），合适下载服务器，默认关闭。
    tcp_nopush     on;
# tcp_nopush：在linux的实现里，其实就是tcp_cork 。
# tcp_nodelay 最多不过是在等等，看能不能再搞点数据，如果实在是没数据了，小包它也发（200ms）。
# 而这个tcp_cork直接就禁止了小包的发送。
# 也就是说，如果你开着这个选项，你的nginx发送的包都是满的。
# 包都是满的，那ACK就少，网络利用率就起来了
# 默认 ：off
    keepalive_timeout  65;
# 连接超时时间，单位是秒
    tcp_nodelay  on;
#启动TCP_NODELAY，就意味着禁用了 Nagle 算法，允许小包的发送。可降低延迟，但是会增加网络的负担
# 关闭TCP_NODELAY，则是应用了 Nagle 算法。 数据只有在写缓存中累积到一定量之后，才会被发送出去，这样明显提高了 网络利用率（实际传输数据payload与协议头的比例大大提高）。但是这由不可避免地增加了延时。
# 默认：  on
```

```
    server_names_hash_bucket_size 128;
# 服务器名字的hash表大小
    client_header_buffer_size 32k;
# 客户端请求头部的缓冲区大小，一般一个请求的头部大小不会超过1k
    large_client_header_buffers 4 64k;
# header过大，它会使用large_client_header_buffers来读取
    client_max_body_size 8m;
# 接收 客户端 主体 的最大体积，可以限制用户上传单个文件的大小。
# FastCGI相关参数是为了改善网站的性能：减少资源占用，提高访问速度。
                fastcgi_connect_timeout 300;
                fastcgi_send_timeout 300;
                fastcgi_read_timeout 300;
                fastcgi_buffer_size 64k;
                fastcgi_buffers 4 64k;
                fastcgi_busy_buffers_size 128k;
                fastcgi_temp_file_write_size 128k;
# gzip模块设置
gzip on;                    #开启gzip压缩输出
gzip_min_length 1k;          #最小压缩文件大小
gzip_buffers 4 16k;          #压缩缓冲区
gzip_http_version 1.0;       #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）
gzip_comp_level 2;           #压缩等级
gzip_types text/plain application/x-javascript text/css application/xml;
                             #压缩类型，默认就已经包含text/html，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。
gzip_vary on;
limit_zone crawler $binary_remote_addr 10m;     #开启限制IP连接数的时候需要使用

include /etc/nginx/conf.d/*.conf;              #附加配置文件
include /etc/nginx/sites-enabled/*;

# upstream的负载均衡（nginx 作为web 代理角色 ）。
# weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。
```

```
upstream tomcat1 {
    server 127.0.0.1:8080 weight=1;
    server 192.168.1.116:8081 weight=1;
}

# 设定负载均衡的服务器列表,可以配置多个负载均衡的服务器列表
# upstream的负载均衡，weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。
upstream  tomcat2 {
    server 127.0.0.1:8080 weight=1;
    server 192.168.1.11:8081 weight=1;
}

# 配置虚拟主机
server {

    listen  80;       #监听端口
    server_name 192.168.1.11;       #域名可以有多个，用空格隔开， 多个虚拟主机主要依靠这一选项来区分，重要!
    index index.html index.htm index.jsp;      #定义索引首页
    root /home/public;           #定义站点根目录
    error_page       400 402 403 404 405 406 410 411 413 416 500 501 502 503 504  /error.html;
    error_page       505  /error.html;                #定义错误页面
    log_format access '$remote_addr - $remote_user [$time_local] "$request" '    #日志格式设定
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" $http_x_forwarded_for';
    access_log /var/log/nginx/access.log access;
 #定义本虚拟主机的访问日志
############################## 使用 location 标签对  URI 进行控制 ###################################
location / {
    expires 2m;
    add_header Cache-Control “public, must-revalidate, proxy-revalidate”;    #添加自定义首部
```

```
    add_header Cache-Control “public, must-revalidate, proxy-revalidate”;
        }
location ~ /rest/(api|images|files)/ {            #所有的rest请求都是以api开头的
     proxy_pass http://localhost:8080;
     proxy_next_upstream error timeout;
     proxy_connect_timeout 8s;
     proxy_intercept_errors on;
     proxy_set_header X-Forwarded-Host httphost;
     proxysetheaderX-Forwarded-Serverhost;
     proxy_set_header X-Forwarded-For proxy_add_x_forwarded_for;
     proxy_set_header  X-Real-IPremote_addr;
     proxy_set_header Host $http_host;
 }
location / {                                  #默认请求    #对“/”启用反向代理
     proxy_pass localhost;
     proxy_redirect off;
     proxy_set_header        X-Real-IP $remote_addr;

     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
# 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;                 #以下是一些反向代理的配置，可选。
     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 128;             #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
     }
location ~ ^/(images|javascript|js|css|flash|media|static)/ {

#以下为设置静态资源，nginx自己处理
        root /home;
        expires 30d;  #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。
     }
location ~ .*.(gif|jpg|jpeg|png|bmp|swf)$ {
     #图片缓存时间设置
        expires 30d;
     }
location ~ .*.(js|css)?$ {              #JS和CSS缓存时间设置
        expires 30h;
     }
location ~ .(jsp|jspx|do)?$ {          #所有jsp的页面均交由tomcat或resin处理
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:8080;
     }
location ~ .*.(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|pdf|xls|mp3|wma)$ {          #所有静态文件由nginx直接读取不经过tomcat或rest
        expires 15d;
     }
location ~ .*.(js|css)?$ {
        expires 1h;
      }
```

```
location /NginxStatus {                    #设定查看Nginx状态的地址
    stub_status on;
    access_log on;
    auth_basic "NginxStatus";
    auth_basic_user_file conf/htpasswd;
    htpasswd文件的内容可以用apache提供的htpasswd工具来产生。
 }
    }
}
################################### 其他知识
#####################################################
# 如果进入一个页面就下载东西，那是nginx无法解析目标内容。
```

二、关于虚拟主机

CGI = Common Gateway Interface

顾名思义，它是一种接口规范。该规范详细定义了Web服务器中运行的服务器代理程序， 怎样获取及返回网页生成过程中，服务器环境上下文和HTTP协议中的参数名称，如大家所熟知的：REQUEST_METHOD，QUERY_STRING，CONTENT_TYPE等等。

绝大部分的Web服务器程序，是以脚本的形式代理接受并处理HTTP请求，返回HTTP页面或响应。这些脚本程序，就是大家所熟知的PHP、ASP、JSP等等。

FCGI = Fast

CGI它其实是CGI在具体实现中的的一个变种。其设计思路是，通过减少CGI代理程序和Web宿主服务程序的通信开销， 从而达到提高Web服务性能的最终目的。由此可见，FCGI在规范上跟CGI并没有不同，只是具体实现方式上有所改进：

CGI的做法是，对于每个HTTP请求，Web宿主服务程序都建立新的进程以调用服务器脚本，响应该请求；

FCGI的做法是，建立一个独立的FCGI服务程序进程，和 Web宿主服务程序进程通信，FCGI服务进程被一旦启动后，自己分配资源、创建线程响应 HTTP请求、并决定自身生命周期，从而大大降低了系统为了创建进程而做出的资源开销。

现代流行的Web服务器程序，如PHP、ASP.Net，基本都是FCGI的实现。

SCGI = Simple

CGI它是FCGI在精简数据协议和响应过程后的产物。其设计目的是为了适应越来越多基于AJAX或REST的HTTP请求，

而做出更快更简洁的应答。并且SCGI约定，当服务器返回对一个HTTP协议请求响应后，立刻关闭该HTTP连接。

所以不难看出，SCGI更加适合于普遍意义上SOA所提倡的“请求-忘记”这种通信模式。

WSGI = Web Server Gateway Interface

此协议是Python语言的专利，它定义了一组在Web服务宿主程序和HTTP响应代理程序之间通信的普遍适用的接口。

它的产生是因为Python程序员注意到，对于Web框架和Web宿主服务器程序间，有严重的耦合性。

NGINX创建虚拟主机

> 虚拟主机技术是互联网服务器采用的节省服务器硬件成本的技术，虚拟主机技术主要应用于HTTP（Hypertext Transfer Protocol，超文本传输协议）服务，将一台服务器的某项或者全部服务内容逻辑划分为多个服务单位，对外表现为多个服务器，从而充分利用服务器硬件资源。

> 虚拟主机是使用特殊的软硬件技术，把一台真实的物理服务器主机分割成多个逻辑存储单元。每个逻辑单元都没有物理实体，但是每一个逻辑单元都能像真实的物理主机一样在网络上工作，具有单独的IP地址（或共享的IP地址）、独立的域名以及完整的Internet服务器（支持WWW、FTP、E-mail等）功能。

> 虚拟主机的关键技术在于，即使在同一台硬件、同一个操作系统上，运行着为多个用户打开的不同的服务器程式，也互不干扰。而各个用户拥有自己的一部分系统资源（IP地址、文档存储空间、内存、CPU等）。各个虚拟主机之间完全独立，在外界看来，每一台虚拟主机和一台单独的主机的表现完全相同。所以这种被虚拟化的逻辑主机被形象地称为“虚拟主机”。

一.基于域名的方式

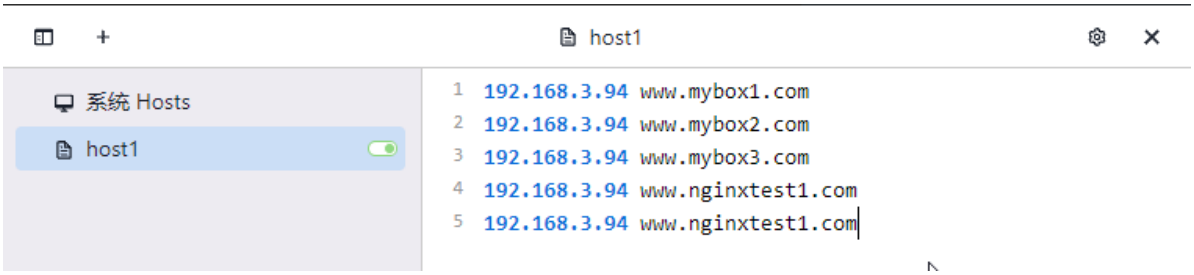
1.创建基于域名的配置文件

```
mkdir -v /etc/nginx/test
mkdir -v /etc/nginx/myhost
mkdir -v /etc/nginx/logs
vim /etc/nginx/myhost/test_ym.conf
server {
    listen 80;
    server_name www.nginxtest1.com;
    access_log  /etc/nginx/logs/nginxtest1.log
    location / {
        root test;
        index index.html nginxtest1.html;
    }
}
server {
    listen 80;
    server_name www.nginxtest2.com;
    access_log  /etc/nginx/logs/nginxtest2.log
    location / {
        root test;
        index index.html nginxtest2.html;
    }
}
# 在主配置文件的http模块加入include
    include myhost/test_ym.conf;

# 创建测试文件
echo "Welcom to nginxtest1\!" >
/etc/nginx/test/nginxtest1.html
echo "Welcom to nginxtest2\!" >
/etc/nginx/test/nginxtest2.html
```

2.本地windows添加域名解析

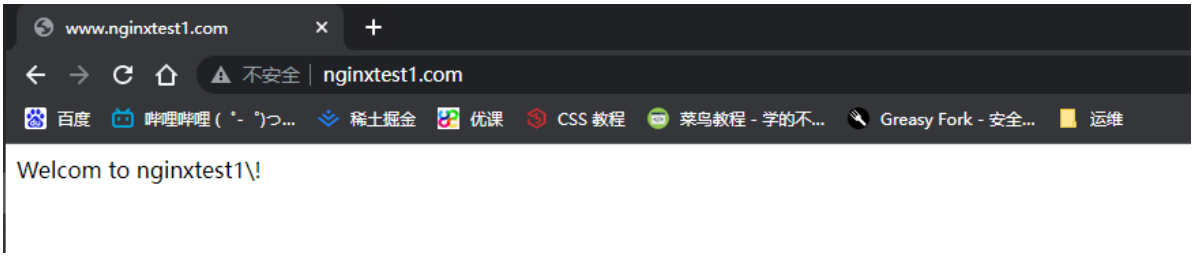
```
C:\WINDOWS\system32\drivers\etc\hosts
192.168.3.94 www.nginxtest1.com
192.168.3.94 www.nginxtest2.com
```

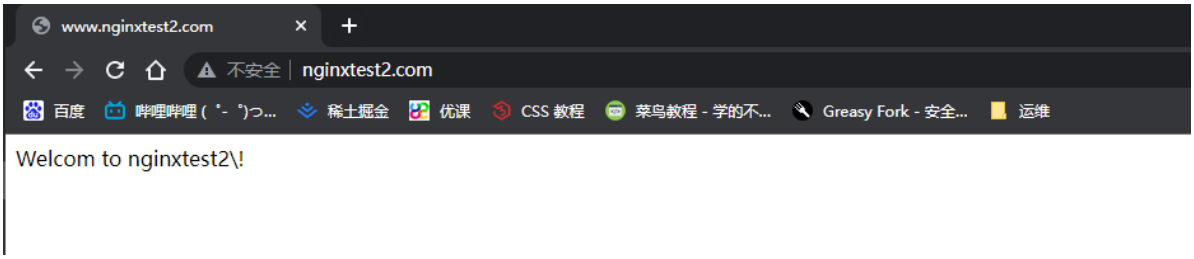


3.重启nginx

```
nginx -t
nginx -c /etc/nginx/nginx.conf
nginx -s reload
```

4.网页属于域名测试





二、基于端口的方式

1.创建配置文件

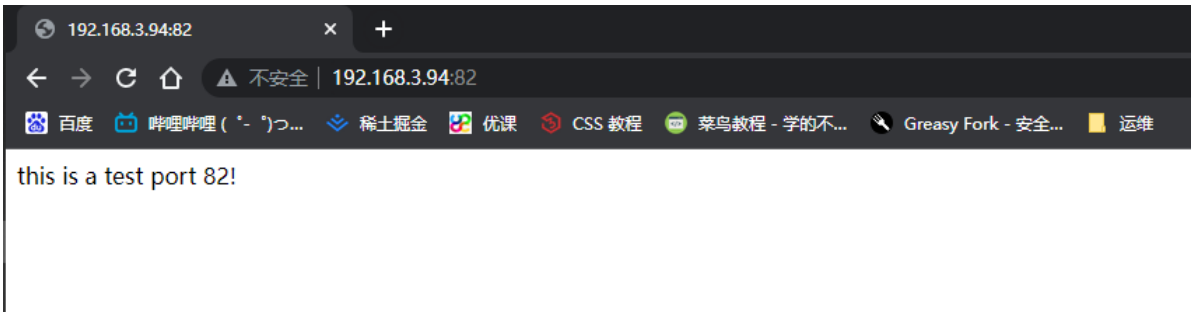
```
vim test_port.conf
server {
    listen 81;
    server_name _;
    location / {
        root test;
        index index.html port81.html;
    }
}
server {
```

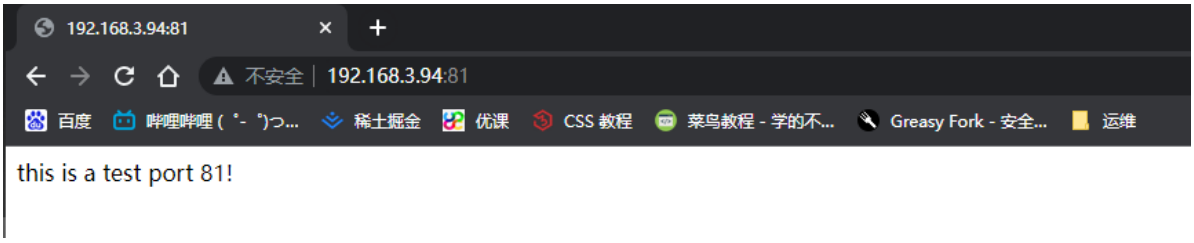
```
    listen 82;
    server_name _;
    location / {
        root test;
        index index.html port82.html;
    }
}
# 在住配置文件创建包含文件
    include myhost/test_port.conf
# 创建测试文件
echo 'this is a test port 81!' >
/etc/nginx/test/port81.html
echo 'this is a test port 82!' >
/etc/nginx/test/port82.html
# 重启nginx
nginx -t
nginx -c /etc/nginx/nginx.conf
nginx -s reload
```

2.本地windows添加域名解析

```
192.168.3.94 www.nginxtest1.com
192.168.3.94 www.nginxtest2.com
```

在浏览器中测试：





三、基于IP的方式

1.创建配置文件

```
vim /etc/nginx/myhost/test_IP.conf
server {
    listen 80;
    server_name 192.168.3.94;
    location / {
        root test;
        index index.html IP94.html;
    }
}
server {
    listen 80;
    server_name 192.168.3.95;
    location / {
        root test;
        index index.html IP94.html;
    }
}
# 在主配置文件http模块中包含test_IP.conf
    include myhost/test_IP.conf
# 创建测试文件
echo 'this is IP94!' > /etc/nginx/test/IP94.html
echo 'this is IP95!' > /etc/nginx/test/IP95.html
# 重启nginx
nginx -t
nginx -c /etc/nginx/nginx.conf
nginx -s reload
```

2.申请临时IP

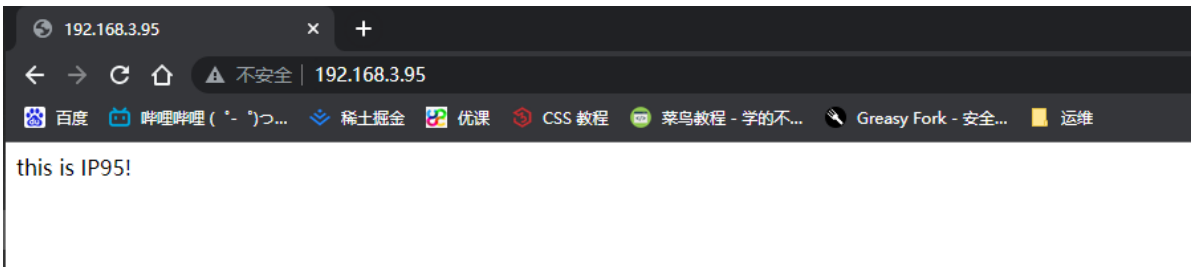
```
ifconfig ens33:1 192.168.3.95
```

```
[root@superbox nginx]# ifconfig ens33:1 192.168.3.95
[root@superbox nginx]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.94  netmask 255.255.255.0  broadcast 192.168.3.255
        inet6 fe80::20c:29ff:fe6d:681a  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:6d:68:1a  txqueuelen 1000  (Ethernet)
        RX packets 107482  bytes 137348457 (130.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13839  bytes 8855538 (8.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens33:1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.95  netmask 255.255.255.0  broadcast 192.168.3.255
        ether 00:0c:29:6d:68:1a  txqueuelen 1000  (Ethernet)

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1151  bytes 321951 (314.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1151  bytes 321951 (314.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3.网页测试



NGINX location匹配

一、location语法

1.语法规则

```
语法规则：  location [=|~|~*|^~] /uri/ { … }
    =     # 用于标准 URI前，要求请求字符串与其精准匹配，成功则立即处理，nginx停止搜索其他匹配。
```

```
    ^~    # 用于标准 URI前，并要求一旦匹配到就会立即处理，不再去匹配其他的那些个正则 URI，一般用来匹配目录。nginx不对url做编码，因此请求为/static/20%/aa，可以被规则^~ /static/ /aa匹配到（注意是空格）。
    ~     # 用于正则 URI前，表示区分大小写的正则匹配
    ~*    # 用于**正则 URI** 前， 表示 URI 包含正则表达式，**不区分**大小写
    !~    # 和!~*分别为区分大小写 不匹配 及不区分大小写 不匹配的正则
    /     # 通用匹配，任何请求都会匹配到。
    @     # @ 定义一个命名的 location，@ 定义的locaiton名字一般用在内部定向，例如error_page, try_files命令中。它的功能类似于编程中的goto。
# location匹配顺序
location 的匹配并不完全按照其在配置文件中出现的顺序来匹配，请求 URI 会按如下规则进行匹配：
1.先精准匹配 = ，精准匹配成功则会立即停止其他类型匹配；
2.没有精准匹配成功时，进行前缀匹配。先查找带有 ^~ 的前缀匹配，带有 ^~ 的前缀匹配成功则立即停止其他类型匹配，普通前缀匹配（不带参数 ^~ ）成功则会暂存，继续查找正则匹配；
3.= 和 ^~ 均未匹配成功前提下，查找正则匹配 ~ 和 ~* 。当同时有多个正则匹配时，按其在配置文件中出现的先后顺序优先匹配，命中则立即停止其他类型匹配；
4.所有正则匹配均未成功时，返回步骤 2 中暂存的普通前缀匹配（不带参数 ^~ ）结果
5.当有匹配成功时候，停止匹配，按当前匹配规则处理请求
# 以上规则简单总结就是优先级从高到低依次为（序号越小优先级越高）：
1. location =     # 精准匹配
2. location ^~    # 带参前缀匹配
3. location ~     # 正则匹配（区分大小写）
4. location ~*    # 正则匹配（不区分大小写）
5. location /a    # 普通前缀匹配，优先级低于带参数前缀匹配。
6. location /     # 任何没有匹配成功的，都会匹配这里处理

注意：前缀匹配，如果有包含关系时，按最大匹配原则进行匹配。比如在前缀匹配：location /dir01 与 location /dir01/dir02，如有请求 http://localhost/dir01/dir02/file 将最终匹配到 location /dir01/dir02
```

2.ReWrite语法

1.基本语法

> last – 基本上都用这个Flag。
> break – 中止Rewirte，不在继续匹配
> redirect – 返回临时重定向的HTTP状态302
> permanent – 返回永久重定向的HTTP状态301
> 注：last和break最大的不同在于break 是终止当前location的rewrite检测,而且不再进行location匹配 - last是终止当前location的rewrite检测,但会继续重试location匹配并处理区块中的rewrite规则

2.下面是可以用来判断的表达式：

> -f和!-f用来判断是否存在文件
> -d和!-d用来判断是否存在目录
> -e和!-e用来判断是否存在文件或目录
> -x和!-x用来判断文件是否可执行

3.下面是可以用作判断的全局变量

```
$args #这个变量等于请求行中的参数。
$content_length # 请求头中的Content-length字段。
$content_type # 请求头中的Content-Type字段。
$document_root # 当前请求在root指令中指定的值。
$host # 请求主机头字段，否则为服务器名称。
$http_user_agent # 客户端agent信息
$http_cookie # 客户端cookie信息
$limit_rate # 这个变量可以限制连接速率。
$request_body_file # 客户端请求主体信息的临时文件名。
$request_method # 客户端请求的动作，通常为GET或POST。
$remote_addr # 客户端的IP地址。
$remote_port # 客户端的端口。
$remote_user # 已经经过Auth Basic Module验证的用户名。
$request_filename # 当前请求的文件路径，由root或alias指令与URI请求生成。
$query_string # 与$args相同。
$scheme #HTTP 方法（如http，https）。
$server_protocol # 请求使用的协议，通常是HTTP/1.0或HTTP/1.1。
$server_addr # 服务器地址，在完成一次系统调用后可以确定这个值。
$server_name # 服务器名称。
```

```
$server_port # 请求到达服务器的端口号。
$request_uri # 包含请求参数的原始URI，不包含主机名，如：”/foo/bar.php?arg=baz”。
$uri # 不带请求参数的当前URI，$uri不包含主机名，如”/foo/bar.html”。
$document_uri # 与$uri相同。

例：http://localhost:88/test1/test2/test.php
$host：localhost
$server_port：88
$request_uri：http://localhost:88/test1/test2/test.php
$document_uri：/test1/test2/test.php
$document_root：D:\nginx/html
$request_filename：D:\nginx/html/test1/test2/test.php
```

4.Redirect语法

1).多目录转成参数

```
abc.domian.com/sort/2 => abc.domian.com/index.php?act=sort&name=abc&id=2
      if ($host ~* (.*)\.domain\.com) {
      set $sub_name $1;
      rewrite ^/sort\/(\d+)\/?$ /index.php?act=sort&cid=$sub_name&id=$1 last;
      }
```

2).目录对换

```
/123456/xxxx -> /xxxx?id=123456
      rewrite ^/(\d+)/(.+)/ /$2?id=$1 last;
# 例如下面设定nginx在用户使用ie的使用重定向到/nginx-ie目录下：
  if ($http_user_agent ~ MSIE) {
      rewrite ^(.*)$ /nginx-ie/$1 break;
      }
# 目录自动加“/”
  if (-d $request_filename){
      rewrite ^/(.*)([^/])$ http://$host/$1$2/ permanent;
      }
```

```
# 禁止htaccess
  location ~/\.ht {
      deny all;
      }
```

3).禁止多个目录

```
 location ~ ^/(cron|templates)/ {
     deny all;
     break;
     }
       # 禁止以/data开头的文件
       # 可以禁止/data/下多级目录下.log.txt等请求;
 location ~ ^/data {
     deny all;
     }
```

4).禁止单个目录

```
# 不能禁止.log.txt能请求
  location /searchword/cron/ {
      deny all;
      }
        # 禁止单个文件
  location ~ /data/sql/data.sql {
      deny all;
      }
        # 给favicon.ico和robots.txt设置过期时间;
        # 这里为favicon.ico为99 天,robots.txt为7天并不记录404错误日志
  location ~(favicon.ico) {
      log_not_found off;
      expires 99d;
      break;
      }
  location ~(robots.txt) {
      log_not_found off;
      expires 7d;
      break;
      }
```

5).设定某个文件的过期时间;这里为600秒，并不记录访问日志

```
 location ^~ /html/scripts/loadhead_1.js {
     access_log   off;
     root /opt/lampp/htdocs/web;
     expires 600;
     break;
     }
```

6).文件反盗链并设置过期时间

```
# 这里的 return 412 为自定义的http状态码，默认为403，方便找出正确的盗链的请求
“rewrite ^/ http://leech.c1gstudio.com/leech.gif;”  # 显示一张防盗链图片
“access_log off;”          # 不记录访问日志，减轻压力
“expires 3d”               # 所有文件3天的浏览器缓存
  location ~* ^.+\.(jpg|jpeg|gif|png|swf|rar|zip|css|js)$ {
      valid_referers none blocked *.c1gstudio.com *.c1gstudio.net localhost 208.97.167.194;
      if ($invalid_referer) {
      rewrite ^/ http://leech.c1gstudio.com/leech.gif;
      return 412;
      break;
   }
      access_log   off;
      root /opt/lampp/htdocs/web;
      expires 3d;
      break;
}
         # 只充许固定ip访问网站，并加上密码
      root  /opt/htdocs/www;
      allow   208.97.167.194;
      allow   222.33.1.2;
      allow   231.152.49.4;
      deny    all;
      auth_basic "C1G_ADMIN";
      auth_basic_user_file htpasswd;
         # 将多级目录下的文件转成一个文件，增强seo效果
```

```
        /job-123-456-789.html 指向/job/123/456/789.html
      rewrite ^/job-([0-9]+)-([0-9]+)-([0-9]+)\.html$ /job/$1/$2/jobshow_$3.html last;
        # 将根目录下某个文件夹指向2级目录
        # 如/shanghaijob/ 指向 /area/shanghai/
        # 如果你将last改成permanent，那么浏览器地址栏显是/location/shanghai/
      rewrite ^/([0-9a-z]+)job/(.*)$ /area/$1/$2 last;
        # 上面例子有个问题是访问/shanghai 时将不会匹配
      rewrite ^/([0-9a-z]+)job$ /area/$1/ last;
      rewrite ^/([0-9a-z]+)job/(.*)$ /area/$1/$2 last;
        # 这样/shanghai 也可以访问了，但页面中的相对链接无法使用，
        # 如./list_1.html真实地址是/area/shanghia/list_1.html会变成/list_1.html,导至无法访问。
        # 那我加上自动跳转也是不行咯
        # (-d $request_filename)它有个条件是必需为真实目录，而我的rewrite不是的，所以没有效果
      if (-d $request_filename){
      rewrite ^/(.*)([^/])$ http://$host/$1$2/ permanent;
      }
        # 知道原因后就好办了，让我手动跳转吧
      rewrite ^/([0-9a-z]+)job$ /$1job/ permanent;
      rewrite ^/([0-9a-z]+)job/(.*)$ /area/$1/$2 last;
        # 文件和目录不存在的时候重定向：
      if (!-e $request_filename) {
      proxy_pass http://127.0.0.1;
      }
        # 域名跳转
      server
      {
      listen       80;
      server_name  jump.c1gstudio.com;
      index index.html index.htm index.php;
      root  /opt/lampp/htdocs/www;
      rewrite ^/ http://www.c1gstudio.com/;
      access_log  off;
      }
```

```
        # 多域名转向
      server_name  www.c1gstudio.com www.c1gstudio.net;
      index index.html index.htm index.php;
      root  /opt/lampp/htdocs;
      if ($host ~ "c1gstudio\.net") {
      rewrite ^(.*) http://www.c1gstudio.com$1
permanent;
      }
        # 三级域名跳转
      if ($http_host ~* "^(.*)\.i\.c1gstudio\.com$") {
      rewrite ^(.*) http://top.yingjiesheng.com$1;
      break;
      }
        # 域名镜向
      server
      {
      listen       80;
      server_name  mirror.c1gstudio.com;
      index index.html index.htm index.php;
      root  /opt/lampp/htdocs/www;
      rewrite ^/(.*) http://www.c1gstudio.com/$1 last;
      access_log  off;
      }
```

NGINX访问控制与状态模块

一、NGINX访问控制

基于用户： **账号密码访问**

基于主机：某些IP允许，某些不允许。

```
vim /etc/nginx/myhost/need_password.conf
server {
        listen   80;
        server_name  [www.news.com;]
(http://www.news.com;)
        access_log  /data/web-
data/logs/new.com.access.log  main;
```

```
        error_page  404 403           /error/404.html;
        error_page  500 502 503 504  /error/50x.html;
    location  / {
           root   /data/web-data/news.com;
           index  index.html;
        }

    location  ~ \.txt$ {             # 用location 表示文件级别的访问控制;
                deny all;
        }
        **location = /nginx_status {**
             auth_basic  "nginx access test!";
      ##基于账号密码访问控制
             auth_basic_user_file /usr/local/nginx/conf/htpassword;

# 自行在 /usr/local/nginx/conf/htpassword 下用 htpasswd -mc  admin  命令创建
# 或者直接使用
# admin:$apr1$mptsVmQk$1dk3mviIDse1OT9N4Uhjy0
# 密码:123123
            stub_status on;           ##启用状态查询模块。
            allow 192.168.10.1;      ## 主机访问控制
            deny all;
        }
    }

################################### 示范操作 ##################################################
cat > /etc/nginx/myhost/need_password.conf << eof
server {
        listen   80;
        server_name  www.mybox3.com;
        access_log /etc/nginx/logs/passwd.com.access.log;
        error_page  404 403           /error/404.html;
        error_page  500 502 503 504  /error/50x.html;
    location  / {
```

```
        root   test;
        index  index.html;
 }

    location  ~ \.txt$ {
    deny all;
 }
    location = /nginx_status {
        auth_basic  "nginx access test!";
        auth_basic_user_file
/etc/nginx/password/mybox3.psd;
        stub_status on;
        allow 192.168.3.1;
        deny all;
 }
}
eof
# 下载httpd-tools
yum -y install httpd-tools
# 添加密码
htpasswd -cm /etc/nginx/myhost/need_password.conf zy
#
```

```
[root@superbox nginx]# #mybox3.psd
[root@superbox nginx]# htpasswd -cm /etc/nginx/password/mybox3.psd zy
New password:
Re-type new password:
Adding password for user zy
```

二、NGINX状态监控模块

基本活跃指标

名称	描述	指标类型
Accepts （接受）	NGINX 所接受的客户端连接数	资源: 功能
Handled （已处理）	成功的客户端连接数	资源: 功能
Dropped （已丢弃，计算得出）	丢弃的连接数（接受 - 已处理）	工作：错误*
Requests （请求数）	客户端请求数	工作：吞吐量

```
# 编译安装Nginx时需要加上参数。
--with-http_stub_status_module
 # 配置nginx配置文件的server模块，打开status模块。
server {
        listen 80;
        server_name localhost;
        location /nginx-status {
                stub_status     on;                  # 打开监控模块
                access_log      on;
                }
}
# 用 curl localhost/nginx-status 或者 在浏览器中输入 http://IP/nginx-status 查看
[root@superbox myhost]# curl 192.168.3.124/nginx-status
Active connections: 3
server accepts handled requests
 13836 13836 14962
Reading: 0 Writing: 1 Waiting: 2
```


```
Active connections: 3    当前活动的连接数
server accepts        handled        requests
13860                  13826          14962
server accepts    # 从启动到现在接受的 请求数量
```

```
handled           # 从启动到现在成功处理的 请求数
requests          # 总共 收到的请求数(requests)

# 失败连接 =（总连接数－成功连接数）
connection    # 连接数，tcp连接
request         # http请求，GET/POST/DELETE/UPLOAD
Reading: 0 Writing: 1 Waiting: 2
Reading: 0        # 正在读取客户端 Header的信息数       请求头
Writing: 1        # 返回给客户端的 Header的信息数       响应头
Waiting: 2        # 长连接模式下，保持连接的 等待的请求数
### 重启服务器 ，  计数器会清理。。。
```

NGINX做代理服务器

1.简单的代理

此例子需要先安装tomcat

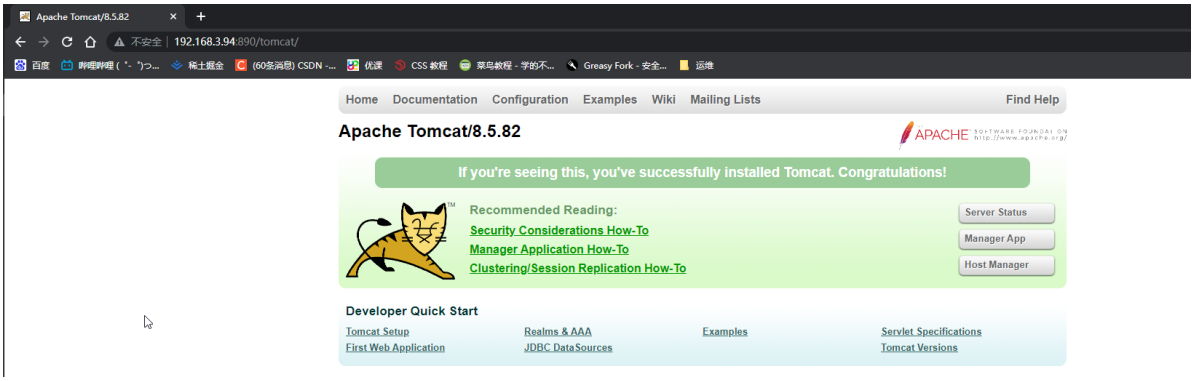
负载服务器：192.168.3.120

次元服务器： 192.168.3.94

```
vim /etc/nginx/nginx.conf
server {
     listen              890;
     server_name         localhost;
     default_type        text/html;

     location /tomcat {
         proxy_pass http://192.168.3.120:8080/;
 }
}

# 浏览器中输入本地IP/tomcat实现跳转
```



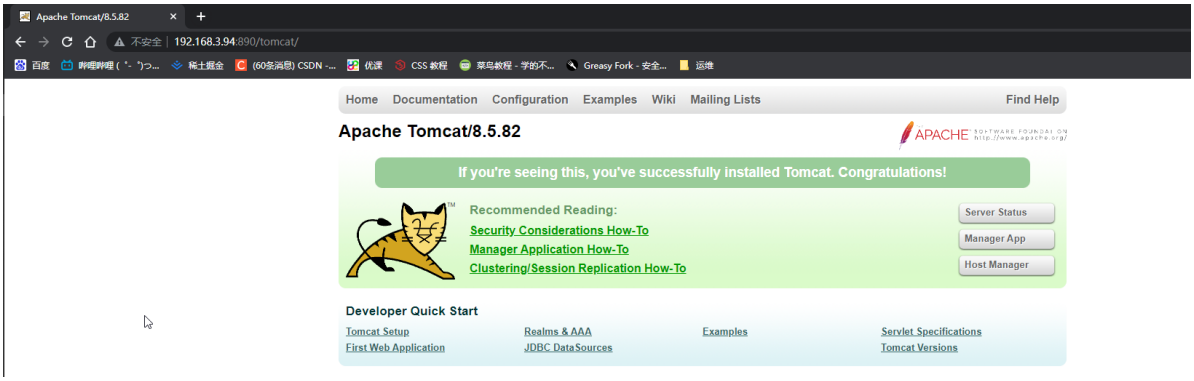

```
192.168.3.247 - - [16/Sep/2022:23:02:06 +0800] "GET /tomcat/ HTTP/1.1" 200 11230 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/105.0.0.0 Safari/537.36"
192.168.3.247 - - [16/Sep/2022:23:03:18 +0800] "GET /tomcat/ HTTP/1.1" 200 11230 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/105.0.0.0 Safari/537.36"
192.168.3.247 - - [16/Sep/2022:23:03:20 +0800] "GET /tomcat/ HTTP/1.1" 200 11230 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/105.0.0.0 Safari/537.36"
192.168.3.247 - - [16/Sep/2022:23:03:22 +0800] "GET /tomcat/ HTTP/1.1" 200 11230 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/105.0.0.0 Safari/537.36"
192.168.3.247 - - [16/Sep/2022:23:05:26 +0800] "GET /tomcat/ HTTP/1.1" 200 11230 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/105.0.0.0 Safari/537.36"
192.168.3.247 - - [16/Sep/2022:23:05:28 +0800] "GET /tomcat/ HTTP/1.1" 200 11230 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/105.0.0.0 Safari/537.36"
```

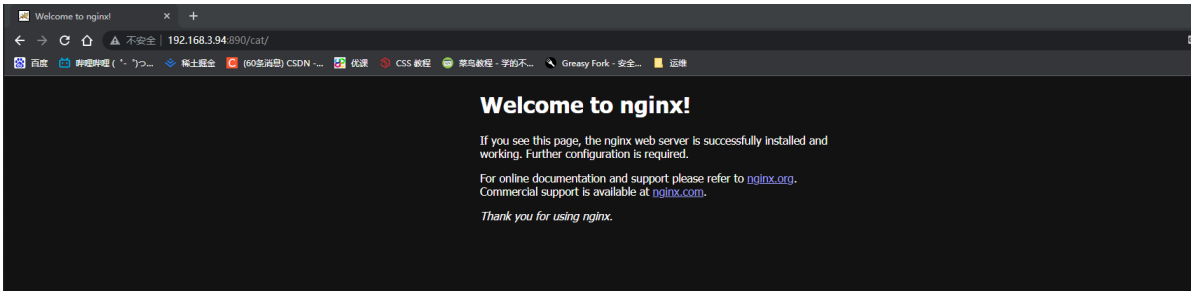
2.NGINX实现反向代理

```
# 在服务器192.168.3.94上操作：
vim /etc/nginx/nginx.conf
server {
     listen              890;
     server_name         localhost;
     default_type        text/html;

     location /tomcat {
         proxy_pass http://192.168.3.120:8080/;
 }
     location /cat/ {
         proxy_pass http://192.168.3.120:80/;
 }
}

# 在网页中输入 192.168.3.94:890/tomcat/ 跳转至 192.168.3.120:8080
# 在网页中输入 192.168.3.94:890/cat/ 跳转至 192.168.3.120:80
```

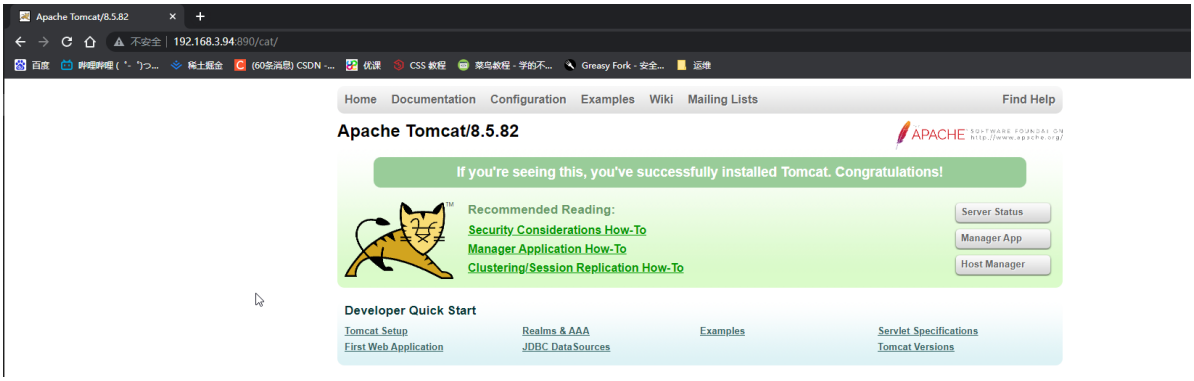


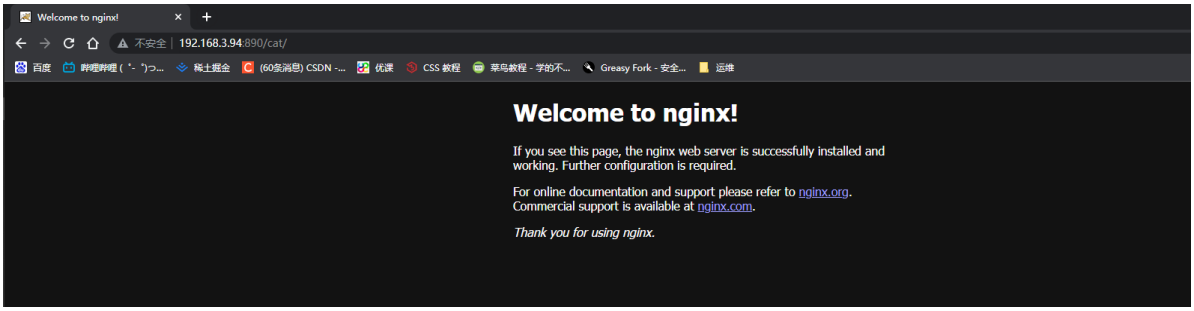


3.nginx实现均衡负载

```
# 在服务器192.168.3.94进行以下配置实现轮询的负载操作：
vim /etc/nginx/nginx.conf
upstream tomcat {
    server 192.168.3.120:8080;
    server 192.168.3.120:80;
}
server {
     listen              890;
     server_name         localhost;
     default_type        text/html;

     location /cat {
         proxy_pass http://tomcat/;
 }
}
# 以上配置实现的为服务器的轮询操作。
# 在浏览器输入192.168.3.94:890/cat/进行验证
```





4.通过设置权重实现均衡负载

```
# 在服务器192.168.3.94进行以下配置实现赋权的负载操作：
vim /etc/nginx/nginx.conf
upstream tomcat {
    server 192.168.3.120:8080 weight=10;
    server 192.168.3.120:80 weight=1;
}
server {
     listen              890;
     server_name         localhost;
     default_type        text/html;

     location /cat {
         proxy_pass http://tomcat/;
 }
}
# 以上配置实现的为服务器的轮询操作。
# 在浏览器输入192.168.3.94:890/cat/进行验证
# 会发现tomcat页面出现的频率大于nginx的页面
```

5.

NGINX防盗链

一、

1.添加hosts文件

资源主机：192.168.3.94
防盗主机：192.168.3.120

```
cat >> /etc/hosts <<eof
192.168.3.94  www.nginx1.com
192.168.3.120 www.nginx2.com
eof
```

2.编辑测试文件

```
vim /etc/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to word!</h1>
<p>If you see this page, the nginx web server is
successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
```

```
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

NGINX四层代理

1.编译添加--whith-stream模块

```
# 进入预编译目录根文件夹下
./configure --prefix=/etc/nginx --sbin-path=/usr/sbin/nginx --modules-path=/usr/lib64/nginx/modules --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --lock-path=/var/run/nginx.lock --http-client-body-temp-path=/var/cache/nginx/client_temp --http-proxy-temp-path=/var/cache/nginx/proxy_temp --http-fastcgi-temp-path=/var/cache/nginx/fastcgi_temp --http-uwsgi-temp-path=/var/cache/nginx/uwsgi_temp --http-scgi-temp-path=/var/cache/nginx/scgi_temp --user=nginx --group=nginx --with-compat --with-file-aio --with-threads --with-http_addition_module --with-http_auth_request_module --with-http_dav_module --with-http_flv_module --with-http_gunzip_module --with-http_gzip_static_module --with-http_mp4_module --with-http_random_index_module --with-http_realip_module --with-http_secure_link_module --with-http_slice_module --with-http_ssl_module --with-http_stub_status_module --with-http_sub_module --with-http_v2_module --with-mail --with-mail_ssl_module --with-stream --with-stream_realip_module --with-stream_ssl_module --with-stream --with-stream_ssl_preread_module --with-cc-opt='-O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4 -grecord-gcc-switches -m64 -mtune=generic -fPIC' --with-ld-opt='-Wl,-z,relro -Wl,-z,now -pie'
make -j
# 注意不要make install，会将原来的文件夹覆盖!
# make结束，在当前文件夹下会出现一个新的nginx可执行文件
# 找到当前系统下的nginx可执行文件
find / -name nginx
# 将旧文件备份，将新文件放到原目录下
mv /usr/sbin/nginx /usr/sbin/nginx.bak
mv nginx /usr/sbin/
# 使用nginx -V 即可查看新nginx的信息
```

2.配置NGINX4层代理

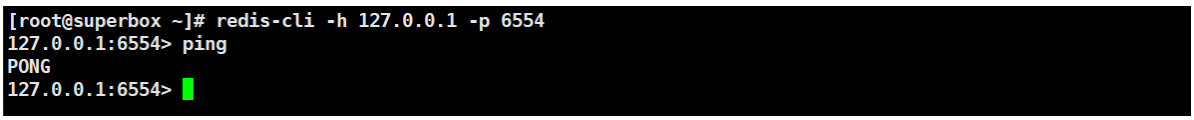
Nginx是基于Proxy Store来实现 的，其原理是把URL及相关组合当做Key,在使用MD5算法对Key进行哈 希，得到硬盘上对应的哈希目录路径，从而将缓存内容保存在该目录 中。它可以支持任意URL连接，同时也支持404/301/302这样的非200状 态码。Nginx即可以支持对指定URL或者状态码设置过期时间，也可以使 用purge命令来手动清除指定URL的缓存。

```
# 添加映射
192.168.3.124 www.nginx1.com
192.168.3.120 www.nginx2.com
# 配置nginx.conf
stream {
log_format proxy '$remote_addr $remote_port - $msec - [$time_local] $status $protocol'
                                                # 远程客户端地址 远程客户端随机端口 毫秒    时间    状态    协议
                  '"$upstream_addr" "$upstream_bytes_sent" "$upstream_connect_time"';
                                                # 虚拟链接池的主机地址    大小     连接的时间
   server {
     listen 6553;
     proxy_connect_timeout 1s;
     proxy_pass tom;
  }
   server {
     listen 6554;
         proxy_connect_timeout 1s;
         proxy_pass redis;
  }
    upstream tom {
        server www.nginx1.com:8080 weight=5 max_fails=3 fail_timeout=30s;
```

```
        server www.nginx2.com:8080 weight=3 max_fails=3
fail_timeout=30s;
          }
    upstream redis {
        server www.nginx2.com:6379;
          }
}
```

本地（124）测试redis能否连接上（120）的redis:

```
# 注意，该连接方法需要将120机上面redis的配置文件bind 127.0.0.1
改为 bind 0.0.0.0 或者 bind 192.168.3.124
redis-cli -h 127.0.0.1 -p 6554
```



本地（124）测试redis能否连接上（120）的mysql:

```
# 120主机上面mysql创建测试数据库
create database nginxsql;
# 120创建远程访问用户
grant all on *.* to admin@'%' identified by '123456'
with grant option;
flush privileges;
# 124主机通过nginx 4层代理连接120主机的数据库
# 配置文件如下
stream {
log_format proxy '$remote_addr $remote_port - $msec -
[$time_local] $status $protocol'
          # 远程客户端地址 远程客户端随机端口 毫秒          时间
      状态      协议
                  '"$upstream_addr"
"$upstream_bytes_sent" "$upstream_connect_time"';
            # 虚拟链接池的主机地址          大小
     连接的时间
   server {
     listen 6553;
        proxy_connect_timeout 1s;
     proxy_pass tom;
```

```
  }
   server {
     listen 6554;
         proxy_connect_timeout 1s;
         proxy_pass redis;
  }
   server {
     listen 6555;
         proxy_connect_timeout 1s;
         proxy_pass mysql;
  }
    upstream tom {
        server www.nginx1.com:8080 weight=5 max_fails=3 fail_timeout=30s;
        server www.nginx2.com:8080 weight=3 max_fails=3 fail_timeout=30s;
          }
    upstream redis {
        server www.nginx2.com:6379;
          }
    upstream mysql {
        server 192.168.3.120:3307;
          }
}
# 测试
mysql -uadmin -p123456 -h192.168.3.125 -P6555
```

```
[root@superbox ~]# mysql -uadmin -p123456 -h192.168.3.125 -P6555
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 5.7.38-log Source distribution

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| nginxsql           |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

NGINX代理缓存和清除

概述：

代理缓存适用7层，不适合4层。

内容缓存位于客户端和源服务器 (upstream) 之间，并保存它看到的所有内容的副本。如果客户端请求缓存已存储的内容，则它会直接返回内容而不连接源服务器。这提高了性能，因为内容缓存更靠近客户端，并且更有效地使用应用程序服务器，因为它们不必每次都从头开始生成页面。

Web 浏览器和应用程序服务器之间可能存在多个缓存：客户端的浏览器缓存，中间缓存，内容交付网络（CDN）以及位于应用程序服务器前面的负载平衡器或反向代理。即使在反向代理/负载均衡器级别，缓存也可以极大地提高性能。

Nginx 通常作为应用程序堆栈中的反向代理或负载平衡器部署，并具有一整套缓存功能。下面我们将讨论如何使用 Nginx 配置基本缓存。

1.设置和配置基本缓存

只需要两个指令即可启用基本缓存：**proxy_cache_path** 和 **proxy_cache**。

proxy_cache_path 指令设置缓存的路径和配置，**proxy_cache** 用来指令激活它。

```
# 创建缓存目录
mkdir /etc/nginx/data/cache
# 添加配置文件
vim /etc/nginx/myhost/cache.conf
proxy_cache_path /etc/nginx/data/cache levels=1:2
keys_zone=my_cache:10m max_size=10g   inactive=60m
 use_temp_path=off;
server {
    listen 7000;
    server_name _;
    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf|flv|ico)$
{
    #proxy_pass http://docker;(可有可无  让代理缓存)
    expires 30d;         #缓存30天
    access_log off; #关闭访问日志
    }
    location / {
```

```
        proxy_cache my_cache;
        expires 3d
        proxy_pass http://my_upstream;
    }
}
# proxy_cache_path 指令的参数定义了以下设置：
缓存的本地磁盘目录称为 /etc/nginx/data/cache
levels # 在/path/to/cache/ 下设置一个两级目录层次结构。在单个目录中包含大量文件会降低文件访问速度，因此我们建议对大多数部署使用两级目录层次结构。如果 levels 未包含该参数，Nginx 会将所有文件放在同一目录中。
keys_zone # 设置共享内存区域，用于存储缓存键和元数据，例如使用计时器。拥有内存中的密钥副本，Nginx 可以快速确定请求是否是一个 HIT 或 MISS 不必转到磁盘，从而大大加快了检查速度。1 MB 区域可以存储大约 8,000 个密钥的数据，因此示例中配置的 10 MB 区域可以存储大约 80,000 个密钥的数据。
max_size # 设置缓存大小的上限（在本例中为 10 千兆字节）。它是可选的；不指定值允许缓存增长以使用所有可用磁盘空间。当缓存大小达到限制时，一个称为缓存管理器的进程将删除最近最少使用的缓存，将大小恢复到限制之下的文件。
inactive # 指定项目在未被访问的情况下可以保留在缓存中的时间长度。在此示例中，缓存管理器进程会自动从缓存中删除 60 分钟未请求的文件，无论其是否已过期。默认值为 10 分钟（10m）。非活动内容与过期内容不同。Nginx 不会自动删除缓存 header 定义为已过期内容（例如 Cache-Control:max-age=120）。过期（陈旧）内容仅在指定时间内未被访问时被删除。访问过期内容时，Nginx 会从原始服务器刷新它并重置 inactive 计时器。
Nginx # 首先将发往高速缓存的文件写入临时存储区域，use_temp_path=off 指令指示 NGINX 将它们写入将被高速缓存的相同目录。我们建议您将此参数设置 off 为避免在文件系统之间进行不必要的数据复制。use_temp_path 在 Nginx 1.7.10 中引入。
# 最后，该 proxy_cache 指令激活与父 location 块的 URL 匹配的所有内容的缓存（在示例中为/）。您还可以在 server 块中包含 proxy_cache 指令；它适用于没有自己的 location 指令的服务器的所有块。
```

2.如何不使用 Nginx 缓存

使用proxy_cache_bypass 指令。

```
location / {
    proxy_cache_bypass $cookie_nocache $arg_nocache;
    # ...
}
```

3.当上游服务器关闭时提供缓存内容

Nginx 内容缓存的一个强大功能是，Nginx 可以配置为在无法从原始服务器获取新内容时从缓存中提供已缓存的内容。如果缓存资源的所有源服务器都已关闭或暂时占用，则会发生这种情况。

Nginx 不是将错误传递给客户端，而是从缓存中提供文件的陈旧版本。这为 Nginx 代理的服务器提供了额外的容错能力，并确保在服务器故障或流量高峰时的正常运行时间。

要启用此功能，请包含 proxy_cache_use_stale 指令：

```
location / {
    listen 80;
    server_name  _;
    proxy_cache_use_stale error timeout http_500
http_502 http_503 http_504;
}
# 使用此示例配置，如果 Nginx 从原始服务器收到一个 error，
timeout 或任何指定的 5xx 错误，并且在其缓存中具有所请求文件的过时
版本，则它会传递过时文件，而不是将错误转发到客户端。
```

4.如何提高缓存性能

Nginx 具有丰富的可选设置，可用于微调缓存的性能。

```
proxy_cache_path /etc/nginx/data/cache levels=1:2
keys_zone=my_cache:10m max_size=10g
                                        inactive=60m
use_temp_path=off;
```

```
server {
    listen 80;
    server_name  _;
    location / {
        proxy_cache my_cache;
        proxy_cache_revalidate on;
        proxy_cache_min_uses 3;
        proxy_cache_use_stale error timeout updating
http_500 http_502
                              http_503 http_504;
        proxy_cache_background_update on;
        proxy_cache_lock on;
        proxy_pass http://my_upstream;
    }
}
# 配置说明
proxy_cache_revalidate # 指示 Nginx 在使用 GET 条件请求时，从源服务器刷新内容。如果客户端请求缓存但是由缓存控制头定义的过期的内容，则 Nginx将 If-Modified-Since 字段包含在 GET 请求的标头中将它发送到源服务器。因为服务器只有在 Nginx 最初缓存它时自附加到文件的标题 Last-Modified 中记录的时间以来修改了整个项目。
proxy_cache_min_uses # 设置客户端在 Nginx 缓存之前必须请求多少次才被缓存。如果缓存不断填满，这将非常有用，因为它可确保只将最常访问的项添加到缓存中。默认 proxy_cache_min_uses 设置为 1。
指令 updating 参数 proxy_cache_use_stale 与启用 proxy_cache_background_update 指令相结合，# 指示当客户端请求已过期或正在从原始服务器更新的项目时，Nginx 会传递过时的内容。所有更新都将在后台完成。在完全下载更新的文件之前，将为所有请求返回陈旧文件。
与 proxy_cache_lock 启用，# 如果多个客户端请求的文件不在缓存（MISS），只有第一个这些请求是通过原始服务器的。其余请求等待满足该请求，然后从缓存中提取文件。如果 proxy_cache_lock 未启用，会导致缓存未命中的所有请求都将直接发送到源服务器。
```

5.跨多个硬盘拆分缓存

如果您有多个硬盘驱动器，可以使用 Nginx 在它们之间拆分缓存。以下示例根据请求 URI 将客户端均匀分布在两个硬盘驱动器上：

```
proxy_cache_path /etc/nginx/data/cache levels=1:2 keys_zone=my_cache_hdd1:10m
                                                  max_size=10g inactive=60m use_temp_path=off;
proxy_cache_path /etc/nginx/data/cache levels=1:2 keys_zone=my_cache_hdd2:10m
                                                  max_size=10g inactive=60m use_temp_path=off;
split_clients $request_uri $my_cache {
               50%          “my_cache_hdd1”;
               50%          “my_cache_hdd2”;
}
server {
    # ...
    location / {
        proxy_cache $my_cache;
        proxy_pass http://my_upstream;
    }
}
# 这两个 proxy_cache_path 指令在两个不同的硬盘驱动器上定义了两个缓存（my_cache_hdd1 和 my_cache_hdd2）。
# split_clients 配置块指定从一半的请求（结果50%）被缓存在 my_cache_hdd1 与另一半中 my_cache_hdd2。基于 $request_uri 变量的哈希（请求URI）确定每个请求使用哪个缓存，结果是对给定URI的请求总是缓存在同一缓存中。
#
```

6.对 Nginx Cache 进行检测

可以在响应头中加入 $upstream_cache_status 变量以进行检测。

```
add_header X-Cache-Status $upstream_cache_status;
```

此示例 X-Cache-Status 在响应客户端时添加 HTTP 标头。以下是可能的值 $upstream_cache_status：
MISS - 在缓存中找不到响应，因此从原始服务器获取响应。然后缓存响应。

> 1.BYPASS - 响应是从原始服务器获取的，而不是从缓存中提供的，因为请求与 proxy_cache_bypass 指令匹配
> 2.EXPIRED - 缓存中的条目已过期。响应包含来自原始服务器的新内容。
> 3.STALE- 内容过时，因为源服务器未正确响应但 proxy_cache_use_stale 已配置。
> 4.UPDATING- 内容过时，因为条目当前正在更新以响应先前的请求，并且 proxy_cache_use_stale updating 已配置。
> 5.REVALIDATED- proxy_cache_revalidate 指令已启用，Nginx 验证当前缓存的内容是否仍然有效通过（If-6.Modified-Since或If-None-Match）。
> 7.HIT - 响应直接来自有效的缓存

7.Nginx 如何确定是否要缓存响应

默认情况下，Nginx 尊重 Cache-Control 源服务器的标头。它不缓存响应 Cache-Control 设置为 Private，No-Cache 或 No-Store 或 Set-Cookie 在响应头。Nginx 只缓存 GET 和 HEAD 客户端请求。您可以按照以下答案中的说明覆盖这些默认值。

如果 proxy_buffering 设置为 off，Nginx 不会缓存响应。on 默认的。

8.Nginx 是否可以忽略 Cache-Control

使用 proxy_ignore_headers 指令可以忽略 Cache-Control

```
location /images/ {
    proxy_cache my_cache;
    proxy_ignore_headers Cache-Control;
    proxy_cache_valid any 30m;
    # ...
}
# Nginx 忽略 /images/ Cache-Control 下所有内容的标题。该指令强制缓存数据到期，如果忽略标头则需要。Nginx 不会缓存没有过期的文件。
```

9.Nginx 如何缓存 POST 请求

使用 proxy_cache_methods 指令。

```
proxy_cache_methods GET HEAD POST;
```

10.Nginx 如何缓存动态内容

只要 Cache-Control 标头允许。即使在很短的时间内缓存动态内容也可以减少原始服务器和数据库的负载，从而缩短第一个字节的时间，因为不必为每个请求重新生成页面。

11.如何不使用 Nginx 缓存

proxy_cache_bypass 指令。

```
location / {
    proxy_cache_bypass $cookie_nocache $arg_nocache;
    # ...
}
# 该指令定义了 Nginx 立即从源服务器请求内容的请求类型，而不是首先尝试在缓存中找到它。这有时被称为通过缓存 “打孔”。
```

12.Nginx 使用什么缓存密钥

Nginx 生成的密钥的默认形式类似于以下 Nginx 变量的 MD5 哈希：`$scheme$proxy_host$request_uri`；使用的实际算法稍微复杂一些。

```
proxy_cache_path /path/to/cache levels=1:2
keys_zone=my_cache:10m max_size=10g
                 inactive=60m use_temp_path=off;
server {
    # ...
    location / {
        proxy_cache my_cache;
        proxy_pass http://my_upstream;
    }
}
# 对于此示例配置，缓存密钥
http://www.example.org/my_image.jpg 计算为
md5("http://my_upstream:80/my_image.jpg")。
# 请注意，proxy_host 变量用于散列值而不是实际主机名
（www.example.com）。proxy_host 定义为 proxy_pass 指令中指
定的代理服务器的名称和端口。
# 要更改用作密钥基础的变量，请使用该 proxy_cache_key 指令。
```

13.使用 Cookie 作为我的缓存密钥的一部分

缓存键可以配置为任意值，例如：

```
proxy_cache_key
$proxy_host$request_uri$cookie_jessionid;
```

14.Nginx 使用 ETag 标头

在 Nginx 1.7.3 及更高版本中，ETag 标头完全支持 If-None-Match。

15.Nginx 如何处理字节范围请求

如果文件在高速缓存中是最新的，则 Nginx 遵循字节范围请求并仅向项目客户端提供项目的指定字节。如果文件未缓存，或者文件过时，Nginx 会从原始服务器下载整个文件。

如果请求是针对单个字节范围的，则 Nginx 会在下载流中遇到该范围后立即将该范围发送到客户端。如果请求在同一文件中指定了多个字节范围，则 Nginx 会在下载完成时将整个文件传送到客户端。

下载完成后，Nginx 会将整个资源移动到缓存中，以便从缓存中立即满足所有未来的字节范围请求，无论是单个范围还是多个范围。

请注意，upstream 服务器必须支持 Nginx 的字节范围请求，以支持对该 upstream 服务器的字节范围请求。

16.Nginx 如何处理 Pragma 标头

在 Pragma:no-cache 报头由客户加入到绕过所有中间缓存，直接进入到源服务器的请求的内容。Pragma 默认情况下，Nginx 不支持标头，但您可以使用以下 proxy_cache_bypass 指令配置该功能：

```
location /images/ {
    proxy_cache my_cache;
    proxy_cache_bypass $http_pragma;
    # ...
}
```

17.Nginx 是否支持标头 stale-while-revalidate 和 stale-if-error 以及扩展的 Cache-Control

Nginx 1.11.10 及更高版本中支持。这些扩展做了什么：

如果当前正在更新 stale-while-revalidate，Cache-Control HTTP 标头的扩展允许使用陈旧的缓存响应。HTTP 标头的 stale-if-error 扩展 Cache-Control 允许在发生错误时使用陈旧的缓存响应。这些头具有比较低优先级， proxy_cache_use_stale 指令如上所述。

18.Nginx 是否支持 Vary 标头

Nginx 1.7.7 以及更高版本中是支持 Vary 标头的 。

NGINX的跨域问题

概述：

由于浏览器的同源策略，即属于不同域的页面之间不能相互访问各自的页面内容 (注：同源策略，单说来就是同协议，同域名，同端口)，另外开发时使用较多的方式为前后端分离，即前端代码和后端代码分开编写。采用前后端分离的方式编写代码后，在部署时难免会遇到跨域问题。

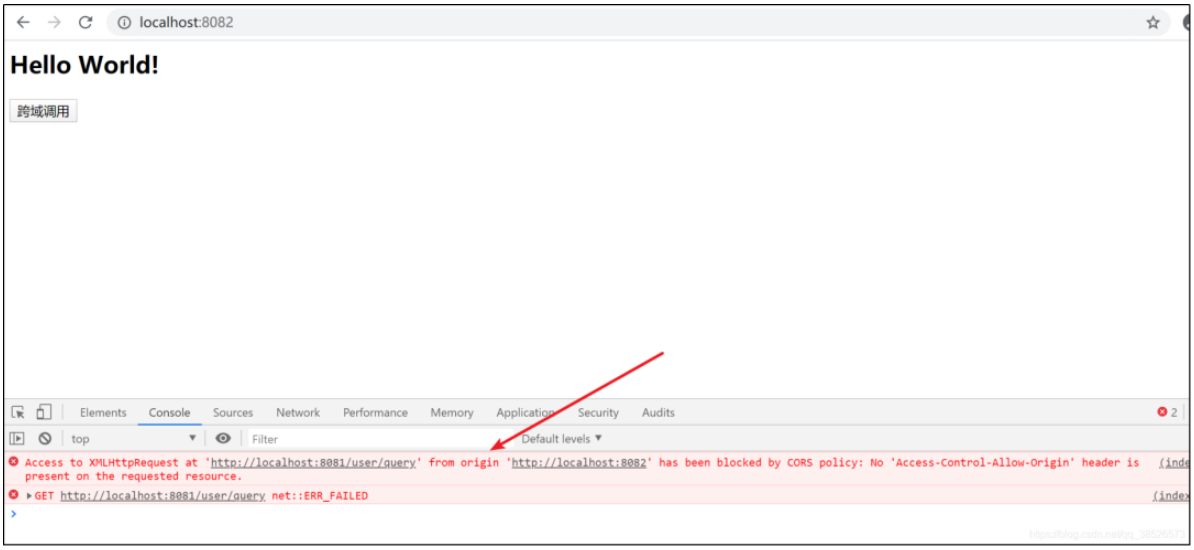
场景模拟：

```
################################## 添加测试文件
#########################################################
vim /etc/nginx/web/test/kuayu.html
<%@ page language="java" contentType="text/html;
charset=utf-8" pageEncoding="utf-8"%>
<html>
<head>
    <meta http-equiv="content-type" content="text/html;
charset=utf-8">
</head>
<body>
<h2>Hello World!</h2>
<script type="text/javascript">
    function fun1(){
        var request = new XMLHttpRequest();

request.open("GET","http://localhost:8081/user/query")
        request.send();
        request.onreadystatechange = function(){
            if(request.status==200 &&
request.readyState == 4){
                console.log("响应的结果" +
request.responseText)
            }
        }
    }
</script>
</body>
    <input type="button" value="跨域调用"
onclick="fun1()">
</html>
```

```
25 ################################### 添加配置文件
   ##################################################
26 # 配置web服务
27 vim /etc/nginx/myhost/kuayuweb.conf
28 server {
29    listen 8081;
30    server_name _;
31    root web/test;
32    index index.html index.htm kuayu.html;
33 }
34
35 # 配置代理服务
36 vim /etc/nginx/myhost/kuayudiaoyong.con
37 server {
38     listen 8082;
39     server_name _;
40     location / {
41       proxy_pass http://192.168.3.128:8081;
42     }
43 }
44 # 在主配置文件的http模块中包含上述服务
45 include myhost/kuayuweb.conf
46 include myhost/kuayudiaoyong.con
```

然后在7002的服务中通过ajax来访问7001的服务，这就不满足同源策略，就会出现跨域问题：



跨域配置：

```
##################################### 跨域配置开始 #####################################
    #nginx 配置文件 server，解决跨域问题
    set $cors_origin "";
        if ($http_origin ~* "^http://www.kuayu.com$") {
                set $cors_origin $http_origin;
        }
        if ($http_origin ~* "^https://www.kuayu.com$") {
                set $cors_origin $http_origin;
        }
    add_header 'Access-Control-Allow-Origin' $cors_origin; #动态设置允许跨域的域名
    add_header 'Access-Control-Allow-Credentials' "true"; #是否携带cookie
    add_header 'Access-Control-Allow-Headers' 'X-Requested-With,token,Content-Type,Access-Token,x-token';#允许的header名称
    add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';#允许的请求方法
    add_header 'Access-Control-Max-Age' 86400;#预检测请求的缓存时间另外浏览器控制面板的Disable cache不勾选才可生效
#nginx 伪静态 location
    if ($request_method = 'OPTIONS') {#拦截预检测请求
      return 200;
    }
 ################################### 跨域配置结束 ###################################

    proxy_pass http://127.0.0.1:7574;//springboot的项目端口为7574
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
```

```
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_connect_timeout 5;
}

################################## 跨域参数说明 ##########################################
```

Access-Control-Allow-Origin：服务器默认是不被允许跨域的。给Nginx服务器配置Access-Control-Allow-Origin *后，表示服务器可以接受所有的请求源（Origin）,即接受所有跨域的请求。

Access-Control-Allow-Headers：预防报错（Request header field Content-Type is not allowed by Access-Control-Allow-Headers in preflight response.），这个错误表示当前请求Content-Type的值不被支持。其实是我们发起了"application/json"的类型请求导致的。这里涉及到一个概念：预检请求（preflight request）,服务器回应时，返回的头部信息如果不包含Access-Control-Allow-Headers: Content-Type则表示不接受非默认的的Content-Type。即出现以下错误：

Access-Control-Allow-Methods：是为了防止出现错误（Content-Type is not allowed by Access-Control-Allow-Headers in preflight response.）

```
################################## 预检请求概念 ##################################
```

预检请求（preflight request）

41 跨域资源共享(CORS)标准新增了一组 HTTP 首部字段，允许服务器声明哪些源站有权限访问哪些资源。另外，规范要求，对那些可能对服务器数据产生副作用的HTTP 请求方法（特别是 GET 以外的 HTTP 请求，或者搭配某些 MIME 类型的 POST 请求），浏览器必须首先使用 OPTIONS 方法发起一个预检请求（preflight request），从而获知服务端是否允许该跨域请求。服务器确认允许之后，才发起实际的 HTTP 请求。在预检请求的返回中，服务器端也可以通知客户端，是否需要携带身份凭证（包括 Cookies 和 HTTP 认证相关数据）。 其实Content-Type字段的类型为application/json的请求就是上面所说的搭配某些 MIME 类型的 POST 请求,CORS规定，Content-Type不属于以下MIME类型的，都属于预检请求 所以 application/json的请求 会在正式通信之前，增加一次"预检"请求，这次"预检"请求会带上头部信息 Access-Control-Request-Headers: Content-Type

NGINX的Https证书申请

1.生成秘钥

```
cd /etc/pki/CA/private

openssl genrsa -out cakey.pem  2048

chmod  400 cakey.pem

#基于pem根文件生成个公钥 生成10年
openssl  req  -new  -x509  -key  cakey.pem  -out  /etc/pki/CA/cacert.pem  -days  3650

#配置信息
#Country Name (2 letter code) [XX]:CN
#State or Province Name (full name) []:GD
#Locality Name (eg, city) [Default City]:GZ
#Organization Name (eg, company) [Default Company Ltd]:wan
#Organizational Unit Name (eg, section) []:ONE
```

```
#Common Name (eg, your name or your server's hostname) []:hg.com
#Email Address []:9999@qq.com

cd /etc/pki/CA
touch  index.txt
echo 01> serial

mkdir /usr/local/nginx/conf/key

cd /usr/local/nginx/conf/key

#生成一个私钥
openssl genrsa -out www.gz.com.key 1024

#生成一个信任证书
openssl req -new -key www.gz.com.key  -out www.gz.com.csr

#Country Name (2 letter code) [XX]:CN
#State or Province Name (full name) []:GD
#Locality Name (eg, city) [Default City]:GZ
#Organization Name (eg, company) [Default Company Ltd]:wan
#Organizational Unit Name (eg, section) []:ONE
#Common Name (eg, your name or your server's hostname) []:hg.com
#Email Address []:9999@qq.com
#Please enter the following 'extra' attributes
#to be sent with your certificate request
#A challenge password []:       (回车)
#An optional company name []:   (回车)

cd /usr/local/nginx/conf/key
cp www.gz.com.csr /etc/pki/CA/

#注意签发必须在这目录下
cd /etc/pki/CA/

echo 01 > serial
```

```
52
53 #签发该请求 有效期 1年
54 #目的:生成www.gz.com.crt
55 openssl  ca -in www.gz.com.csr -out www.gz.com.crt  -
   days 365
56
57 #cat一下若有数字即为成功
58 cat /etc/pki/CA/serial
```

2.拷贝密钥

```
1  cd /etc/pki/CA
2  cp www.gz.com.crt /usr/local/nginx/conf/key/
3
4  #ls /usr/local/nginx/conf/key/
5  #www.gz.com.crt  www.gz.com.csr  www.gz.com.key
6
7  vim /usr/local/nginx/conf/vhost/hg.conf
8
9  server {
10         listen 80;
11         listen 443 ssl;          #必加
12         server_name www.hg.com hg.com;
13         charset utf-8;
14         root /data/hg;
15         index index.html index.html;
16         #必加这两行
17         ssl_certificate
   /usr/local/nginx/conf/key/www.gz.com.crt;
18         ssl_certificate_key
   /usr/local/nginx/conf/key/www.gz.com.key;
19 }
```

3.浏览器测试

```
浏览器打开
https://www.hg.com/
# 将CA 的  证书，拷贝到  桌面。    /etc/pki/CA/cacert.pem
# 修改 后缀，为   cacert.crt
cp /etc/pki/CA/cacert.pem /tmp/cacert.crt
sz
#双击---安装----证书安装向导
```

NGINX平滑升级

```
cd /data/

wget http://nginx.org/download/nginx-1.23.1.tar.gz
tar -xf nginx-1.23.1.tar.gz

#备份防止文件损坏
cp /usr/sbin/nginx /usr/sbin/nginx2.bak

cd /data/nginx-1.23.1

#查看原来编译(复制)
nginx -V

#重新编译到make阶段
```

```
./configure --prefix=/etc/nginx --sbin-path=/usr/sbin/nginx --modules-path=/usr/lib64/nginx/modules --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --lock-path=/var/run/nginx.lock --http-client-body-temp-path=/var/cache/nginx/client_temp --http-proxy-temp-path=/var/cache/nginx/proxy_temp --http-fastcgi-temp-path=/var/cache/nginx/fastcgi_temp --http-uwsgi-temp-path=/var/cache/nginx/uwsgi_temp --http-scgi-temp-path=/var/cache/nginx/scgi_temp --user=nginx --group=nginx --with-compat --with-file-aio --with-threads --with-http_addition_module --with-http_auth_request_module --with-http_dav_module --with-http_flv_module --with-http_gunzip_module --with-http_gzip_static_module --with-http_mp4_module --with-http_random_index_module --with-http_realip_module --with-http_secure_link_module --with-http_slice_module --with-stream --with-http_ssl_module --with-http_stub_status_module --with-http_sub_module --with-http_v2_module --with-mail --with-mail_ssl_module --with-stream --with-stream_realip_module --with-stream_ssl_module --with-stream_ssl_preread_module --with-cc-opt='-O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4 -grecord-gcc-switches -m64 -mtune=generic -fPIC' --with-ld-opt='-Wl,-z,relro -Wl,-z,now -pie'
make -j

#编译完成会产生objs文件
cd /data/nginx-1.23.1/objs

#需要加 -f 参数，否则报文件正忙
\cp -f nginx /usr/sbin/nginx

查看是否成功
nginx -V
#nginx version: nginx/1.23.1
```

```
27
28 #寻找旧版本master
29 ps -ef | grep nginx
30 #root 919 1 0 18:42 ? 00:00:00 nginx: master process
   /usr/sbin/nginx
31
32 #find / -name nginx.pid
33
34 #kill -USR2  旧版本的主进程号 （让旧版本的worker进程不再接受请
   求）
35 kill -USR2 919
36 kill -USR2 `cat /usr/local/nginx/logs/nginx.pid`
37
38 #查看进程的时候有两个master
39 #ps -ef | grep nginx
40 #root 919 1 0 ? 00:00:00 nginx: master process
   /usr/sbin/nginx
41 #root 4761 919 0 ? 00:00:00 nginx: master process
   /usr/sbin/nginx
42
43 #先关闭再退出
44 #敲入该指令919的worker退出了
45 kill -WINCH 919
46 kill -WINCH `cat
   /usr/local/nginx/logs/nginx.pid.oldbin`
47
48 #QUIT
49 kill -QUIT 919
50 kill -QUIT `cat /usr/local/nginx/logs/nginx.pid.oldbin`
```