

Nginx

Nginx

1. 安装与配置文件的简单解读
 - 相关教学视频
 - 1.1 安装与环境变量
 - 安装方式1
 - 安装方式2
 - Nginx配置成系统服务
 - Nginx命令配置到系统环境
 - 安装tree
 - 查看进程
 - 1.2 配置文件简介
2. 命令
 - 2.1 信号量
 - 2.2 命令行控制
3. 版本升级和模块安装
 - 环境准备
 - 方案一:使用Nginx服务信号进行升级
 - 方案二:使用Nginx安装目录的make命令完成升级

配置文件

4. 配置文件一 全局块
 - 4.1 user
 - 4.2 master_process
 - 4.3 worker_processes
 - 4.4 daemon
 - 4.5 pid file
 - 4.6 error_log file
 - 4.7 include
 - 4.10
- 5 配置文件二 -- events
 - 5.1 accept_mutex
 - 5.2 multi_accept
 - 5.3 worker_connections
 - 5.4 use
 - 5.10
6. 配置文件3 -- http
 - 6.1 mime.type
 - 6.2 default_type
 - 6.3 自定义服务日志
 - 6.3.1 access_log
 - 6.3.2 log_format
 - 6.4 send file
 - 6.5 keepalive_timeout
 - 6.6 keepalive_requests
 - 6.10
7. 配置文件4 -- server 和 location
 - 7.1 相应的小练习

静态资源部署

8 配置文件

- 8.1 listen指令
- 8.2 server_name
- 8.3 location
- 8.4 请求资源目录 root/alias
- 8.5 index
- 8.6 error_page

9 静态资源优化

- 9.1 sendfile
- 9.2 tcp_nopush
- 9.3 tcp_nodelay

10 静态资源压缩

- 10.1 ngx_http_gzip_module
 - 10.1.1
 - 10.1.2 gzip_types
 - 10.1.3 gzip_comp_level
 - 10.1.4 gzip_vary
 - 10.1.5 gzip_buffers
 - 10.1.6 gzip_disable
 - 10.1.7 gzip_http_version
 - 10.1.8 gzip_min_length
 - 10.1.9 gzip_proxied
- 10.1 Gzip压缩功能的实例配置
- 10.1 sendfile和gz共存问题[引出]
- 10.2 ngx_http_gzip_static_module
 - 10.2.1 模块安装
 - 10.2.2 使用 gzip_static

11 静态资源的缓存处理

- 11.1 expires
- 11.2 add_header

12 nginx的跨域问题处理

- 12.1 同源策略
- 12.2 解决方案

13 nginx防盗链

- 13.1 valid_referers
 - 针对目录进行防盗链

转发

14 Rewrite指令使用

- 14.1 set指令
- 14.2 nginx内置全局变量
- 14.3 if
- 14.4 break
- 14.5 return
- 14.6 rewrite
- 14.7 rewrite_log指令
- 14.10 案例
 - 域名跳转
 - 域名镜像
 - 独立域名
 - 目录自动添加"/"
 - 合并目录

防盗链

反向代理

15 反向代理

15.1.1 proxy_pass

15.1.2 proxy_set_header

15.1.3 proxy_redirect

安全控制

16 SSL

16.1 nginx模块anzhuang

16.2 ssl开启

16.3 生成证书

开启SSL实例

17 反向代理系统调优

Nginx 负载均衡

17 负载均衡

17.1 nginx 负载均衡

upstream 指令

server指令

17.2 负载均衡状态

down

backup

max_conns

max_fails和fail_timeout

17.3 负载均衡策略

轮询

weight加权[加权轮询]

ip_hash

url_hash

fair

18 nginx四层负载均衡

添加stream模块的支持

Nginx四层负载均衡的指令

stream指令

upstream指令

Nginx缓存集成

19 Nginx的web缓存服务

19.1 proxy_cache_path

19.2 proxy_cache

19.3 proxy_cache_key

19.4 proxy_cache_valid

19.5 proxy_cache_min_uses

19.6 proxy_cache_methods

19.7 配置实例

20 Nginx缓存的清除

方式一:删除对应的缓存目录

方式二:使用第三方扩展模块

ngx_cache_purge

21 设置资源不缓存

\$cookie_nocache、\$arg_nocache、\$arg_comment

动静分离

Nginx高可用环境

Keepalived

- VRRP介绍
- 环境准备
- Keepalived配置文件介绍
- 访问测试
- keepalived之vrrp_script

nginx 制作下载站点

Nginx的用户认证模块

Lua

- 21 lua的基本语法
 - 概念
 - 特性
 - 应用场景
 - Lua的安装
 - 标识符
 - 关键字
 - 运算符
 - 全局变量&局部变量
 - Lua数据类型
 - nil
 - boolean
 - number
 - string
 - table
 - function
 - thread
 - userdata
 - Lua控制结构
 - if then elseif else
 - while循环
 - repeat循环
 - for循环

ngx_lua模块概念

- 安装
 - 方式一:lua-nginx-module
 - 方式二:OpenResty
- 概述
- 安装

- Lua的使用
 - init_by_lua*
 - init_worker_by_lua*
 - set_by_lua*
 - rewrite_by_lua*
 - access_by_lua*
 - content_by_lua*
 - header_filter_by_lua*
 - body_filter_by_lua*
 - log_by_lua*
 - balancer_by_lua*
 - ssl_certificate_by_*

ngx_lua操作Redis

ngx_lua操作Mysql

- lua-resty-mysql

[使用lua-resty-mysql实现数据库的查询](#)
[使用lua-cjson处理查询结果](#)
[lua-resty-mysql实现数据库的增删改](#)
[综合小案例](#)

1. 安装与配置文件的简单解读

[相关文档](#)

相关教学视频

[黑马程序员nginx](#) [时间 2020.05]

[尚硅谷nginx](#) [时间]

```
# user wjl;
worker_processes 2;
error_log logs/error.log;
pid logs/nginx.pid;
daemon on;

events {
    accept_mutex on;
    multi_accept on;
    worker_connections 1024;
    use epoll;
}

#
#stream {
#    upstream redisbackend {
#        server 192.168.10.102:6379;
#        server 192.168.10.102:6370;
#    }
#
#    upstream tomcatbackend {
#        server 192.168.10.102:8080;
#    }
#
#    server {
#        listen 81;
#        proxy_pass redisbackend;
```

```

#   }
#
#   server {
#       listen 82;
#       proxy_pass tomcatbackend;
#   }
#}
#

http {
    # 设置缓存路径 指定参数没有固定顺序 levels=2:1:2 表示三层目录
    proxy_cache_path /opt/nginx/proxy_cache levels=2:1:2
    keys_zone=wjl_key:200m inactive=1d max_size=1g;

    include mime.types;
    default_type application/octet-stream;

    # 下面四个一起开启
    sendfile on;
    keepalive_timeout 65;

    tcp_nopush on;
    tcp_nodelay on;

    # 开启压缩
    gzip off;
    gzip_types application/javascript image/webp video/mp4;
    gzip_comp_level 6;
    gzip_vary on;
    # gzip_min_length 1M;

    gzip_static on;

    add_header wjl niubi;

    # 引入这个目录下所有以 '.conf' 结尾的文件 '*' 通配符
    # include /home/wjl/conf.d/*.conf;

    upstream backend {
        server 192.168.10.102:8080;
    }
    server {
        listen 81;
        server_name location;
        charset utf-8;
    }
}

```

```

# 开启缓存 指定缓存名称
proxy_cache wjl_key;
# 指定缓存key # 默认也是这个
# proxy_cache_key $scheme$proxy_host$request_uri;
proxy_cache_key wjl08;
#
proxy_cache_valid 200 1h;
proxy_cache_min_uses 1;
proxy_cache_methods GET;

location / {
    add_header nginx-cache "$upstream_cache_status";
    proxy_pass http://backend/js/;
}

location ~ /purge(/.*) {
    proxy_cache_purge wjl_key wjl08;
}
}

server {
    listen 82;
    server_name location;

    location / {
        default_type text/plain;
        add_header lo 123;
        return 200
cookie_nocache==>$cookie_nocache==arg_nocache==>$arg_nocache==
arg_comment==>$arg_comment==http_lo==>$arg_wjl;
    }

    location /addcookie {
        add_header Set-Cookie 'nocache=888';
        return 200 $cookie_nocache;
    }
}

upstream tomcat{
    server 192.168.56.2:8080;
}

upstream toms {
    server 192.168.10.102:8080;
    server 192.168.10.102:8082;
    # server 192.168.10.102:8083;
}

server {
    listen 80;
    # server_name www.wjl1128.top;
    server_name ~^www\.(\w+)\.com$;

```

```

charset utf-8;

location /tom {
    proxy_pass http://tomcat/;
}

# 静态下载站点
location /opt {
    root /;
    auth_basic '请输入密码';
    auth_basic_user_file htpasswd;

    autoindex on;
    autoindex_exact_size on;
    autoindex_format html;
    autoindex_localtime off;
}

# 动态资源访问
location /demo {
    proxy_no_cache 1;
    # root html;
    # index index.html index.htm;
    proxy_pass http://toms;
}

# 静态资源
location ~ /\.*\.(js|png|jpg|html)$ {
    root html/web;
    index index.html;
}

location =/getjson {
    default_type application/json;
    add_header localtion_get OK;
    return 200 '{"tom":"cat"}';
}

error_page 500 502 503 504 /50x.html;
location = /50x.html {
    root html;
}

# 发生404[找不到] 重定向到百度
# error_page 404 http://www.baidu.com;

error_page 404 =200 @wjl_error;
location @wjl_error {
    default_type text/plain;

```



```

        return 200 '发生错误!!!';
    }

    # location /wjl/ {
    #     alias /usr/local/nginx/html/;
    #     index 123.webp video/1.mp4;
    # }

    #location ~ .*\. (jpg|webp|png|mp4|html|js|pdf)$ {
    #     # 设置缓存 10d 代表10天 默认以秒为单位 [max]代表10年
    #     add_header file jpg|webp|png|mp4|html|js|pdf;
    #     expires max;
    #     # none 为空 string正则表达式 没有匹配到 $invalid_referer
    #     # 为1 否则0
    #     valid_referers none string ~localhost:52330;
    #     if ($invalid_referer){
    #         return 403;
    #     }
    #     root    html;
    #     index  index.html;
    # }

}

server {
    listen 8000;
    server_name www.wjl1128.top;
    server_name_in_redirect on;
    charset utf-8;

    #location /{
    #     root html;
    #     default_type text/plain;
    #     if (!-f $request_filename){
    #         return 200 '您访问的文件不存在';
    #     }
    # }

    location =/if {
        default_type text/plain;
        if ($request_method = GET){
            return 200 success;
        }
        if ($request_method = PUT){
            return 405;
        }
        return 404 error;
    }
}

```

```

}

location =/getUrl {
    # 302重定向状态码
    return 302 http://www.baidu.com;
}

location /rewrite {
    rewrite_log on;
    error_log logs/rewrite.log notice;
    # rewrite 正则表达式规则    重写的路径    flag[可不写]
    rewrite ^/rewrite/url\w*$ http://www.baidu.com;
    # 括号表达式    $1 == test
    rewrite ^/rewrite/(test)\w*$ /$1 last;
    rewrite ^/rewrite/(demo)\w*$ /$1 redirect;
}

location =/test {
    default_type text/plain;
    return 200 'test重写成功';
}
location =/demo {
    default_type text/plain;
    return 200 'demo重写成功';
}

location /80 {
    rewrite ^/80/80$ http://www.wjl1128.top last;
}
}

server {
    listen 8001;
    server_name www.wjl1128.top;
    # 访问此端口时 可以直接定向 8000
    rewrite ^(.*) http://www.wjl1128.top:8000/$1 last;
}

server {
    listen 8002;
    server_name localhost www.wjl1128.top;
    charset utf-8;

    location /tom {
        proxy_pass http://tomcat/;
    }
}

```

```

    location /102 {
        proxy_set_header username wjlzszs;
        proxy_pass http://192.168.10.102/;
        # 注意 相应的端口号与路径 也要复写 无论是代理端的还是被代理的
:8002/102
        proxy_redirect http://192.168.10.102/
http://www.wjl1128.top:8002/102;
        #proxy_redirect default;
    }
}

upstream zook {
    # server 192.168.10.102:8080 down;
    # server 192.168.10.102:8080 weight=3;
    ip_hash;
    server 192.168.10.102:8080;
    server 192.168.10.102:8000;
    server 192.168.10.102:8001;
}

server {

    listen 8003 ssl;
    server_name localhost www.wjl1128.top;
    charset utf-8;
    # 指定pem证书
    ssl_certificate /opt/nginx/7547995_www.wjl1128.top.pem;
    # 指定 key
    ssl_certificate_key
/opt/nginx/7547995_www.wjl1128.top.key;

    error_log /opt/nginx/error.log notice;

    location / {
        proxy_pass http://zook;
    }
}
}

```

1.1 安装与环境变量

```
# 环境
# Linux系统安装Nginx所需依赖 # 确保有安装GCC
yum install -y pcre pcre-devel zlib zlib-devel openssl openssl-
devel
yum install -y gcc

## 相应的端口开放
firewall-cmd --zone=public --add-port=80/tcp --permanent
firewall-cmd --reload
```

安装方式1

```
# 官网下载
wget http://nginx.org/download/nginx-1.20.2.tar.gz

# 解压缩
tar -zxvf nginx-1.20.2.tar.gz

# 进入目录 执行安装
./configure --prefix=/usr/local/nginx \
--sbin-path=/usr/local/nginx/sbin/nginx \
--modules-path=/usr/local/nginx/modules \
--conf-path=/usr/local/nginx/conf/nginx.conf \
--error-log-path=/usr/local/nginx/logs/error.log \
--http-log-path=/usr/local/nginx/logs/access.log \
--pid-path=/usr/local/nginx/logs/nginx.pid \
--lock-path=/usr/local/nginx/logs/nginx.lock
# 或者./configure

make
make install
# 查看安装是否成功
cd /usr/local/nginx
# 出现以下目录
[root@wjl08 nginx]# ls
conf html logs sbin

./sbin/nginx # 执行

./nginx -v # 查看版本
./nginx -V # 查看版本与详细信息

yum install -y tree
```

```
# 安装树型结构工具`  
# 执行语法 `tree` 目录  
tree /usr/local/nginx/
```

通过 `./configure` 来对编译参数进行设置，需要我们手动来指定。那么都有哪些参数可以进行设置，接下来我们进行一个详细的说明。

`PATH`:是和路径相关的配置信息

`with`:是启动模块，默认是关闭的

`without`:是关闭模块，默认是开启的

我们先来认识一些简单的路径配置已经通过这些配置来完成一个简单的编译：

`--prefix=PATH`

指向Nginx的安装目录，默认值为`/usr/local/nginx`

`--sbin-path=PATH`

指向(执行)程序文件(nginx)的路径,默认值为`<prefix>/sbin/nginx`

`--modules-path=PATH`

指向Nginx动态模块安装目录，默认值为`<prefix>/modules`

`--conf-path=PATH`

指向配置文件(nginx.conf)的路径,默认值为`<prefix>/conf/nginx.conf`

`--error-log-path=PATH`

指向错误日志文件的路径,默认值为`<prefix>/logs/error.log`

`--http-log-path=PATH`

指向访问日志文件的路径,默认值为`<prefix>/logs/access.log`

`--pid-path=PATH`

指向Nginx启动后进行ID的文件路径，默认值为`<prefix>/logs/nginx.pid`

`--lock-path=PATH`

指向Nginx锁文件的存放路径,默认值为<prefix>/logs/nginx.lock

安装方式2

[官网安装文档](#)

```
# 安装yum工具包
sudo yum install yum-utils

vim /etc/yum.repos.d/nginx.repo
# 在以上文件编辑
[nginx-stable]
name=nginx stable repo
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true

[nginx-mainline]
name=nginx mainline repo
baseurl=http://nginx.org/packages/mainline/centos/$releasever/$basearch/
gpgcheck=1
enabled=0
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true

# 安装
sudo yum install nginx
```

```
# 执行
./usr/sbin/nginx
```

Nginx配置成系统服务

把Nginx应用服务设置成为系统服务,方便对Nginx服务的启动和停止等相关操作,具体实现步骤:

(1) 在 `/usr/lib/systemd/system` 目录下添加nginx.service,内容如下:

```
vim /usr/lib/systemd/system/nginx.service
```

```
[Unit]
Description=nginx web service
Documentation=http://nginx.org/en/docs/
After=network.target

[Service]
Type=forking
PIDFile=/usr/local/nginx/logs/nginx.pid
ExecStartPre=/usr/local/nginx/sbin/nginx -t -c
/usr/local/nginx/conf/nginx.conf
ExecStart=/usr/local/nginx/sbin/nginx
ExecReload=/usr/local/nginx/sbin/nginx -s reload
ExecStop=/usr/local/nginx/sbin/nginx -s stop
PrivateTmp=true

[Install]
WantedBy=default.target
```

(2)添加完成后如果权限有问题需要进行权限设置

```
chmod 755 /usr/lib/systemd/system/nginx.service
```

(3)使用系统命令来操作Nginx服务

```
启动: systemctl start nginx
停止: systemctl stop nginx
重启: systemctl restart nginx
重新加载配置文件: systemctl reload nginx
查看nginx状态: systemctl status nginx
开机启动: systemctl enable nginx
```

Nginx命令配置到系统环境

前面我们介绍过Nginx安装目录下的二级制可执行文件 `nginx` 的很多命令，要想使用这些命令前提是需要进入sbin目录下才能使用，很不方便，如何去优化，我们可以将该二进制可执行文件加入到系统的环境变量，这样的话在任何目录都可以使用nginx对应的相关命令。具体实现步骤如下：

演示可删除

```
/usr/local/nginx/sbin/nginx -V
cd /usr/local/nginx/sbin nginx -V
如何优化? ? ?
```

(1)修改 `/etc/profile` 文件

```
vim /etc/profile  
在最后一行添加  
export PATH=$PATH:/usr/local/nginx/sbin
```

(2)使之立即生效

```
source /etc/profile
```

(3)执行nginx命令

```
nginx -V
```

安装tree

```
yum install -y tree  
# 安装树型结构工具'  
# 执行语法 `tree` 目录  
tree /usr/local/nginx/  
[root@wj108 nginx]# tree /usr/local/nginx/  
/usr/local/nginx/  
├── client_body_temp  
├── conf  
│   ├── fastcgi.conf  
│   ├── fastcgi.conf.default  
│   ├── fastcgi_params  
│   ├── fastcgi_params.default  
│   ├── koi-utf  
│   ├── koi-win  
│   ├── mime.types  
│   ├── mime.types.default  
│   ├── nginx.conf  
│   ├── nginx.conf.default  
│   ├── scgi_params  
│   ├── scgi_params.default  
│   ├── uwsgi_params  
│   ├── uwsgi_params.default  
│   └── win-utf  
├── fastcgi_temp  
├── html  
│   ├── 50x.html  
│   └── index.html  
└── logs  
    ├── access.log  
    └── error.log
```



```
|   └─ nginx.pid
|   └─ proxy_temp
|   └─ sbin
|       └─ nginx # 二进制可执行文件
|   └─ scgi_temp
|   └─ uwsgi_temp
```

```
# mime.types 默认对数据格式的后缀名
# nginx.conf 核心配置文件
# access.log 访问日志
# error.log 错误日志
[root@wjl08 nginx]# tail -f logs/nginx.pid
6412
```

查看进程

```
# 此时只启动了一个服务
[root@wjl08 nginx]# ps -aux|grep nginx
root          6412  0.0  0.0  34356   380 ?        Ss   14:56   0:00
nginx: master process ./sbin/nginx
nobody        6413  0.0  0.1  69112  4248 ?        S    14:56   0:00
nginx: worker process
```

1.2 配置文件简介

```
# 工作进程数
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    # 引入配置文件 服务器给浏览器的请求头 对应的文件后缀名
    include mime.types;

    # 默认的类型 如果 [mime.types] 不包含 会在这里面选
    default_type application/octet-stream;

    # 零拷贝 [减少一次调度的过程]
    sendfile on;
```

```

# 保持连接超时
keepalive_timeout 65;

# 代表的“主机” [虚拟主机{vhost}]
server {
    # 监听的端口号
    listen 80;
    # 域名/主机名
    server_name localhost;
    # 文字编码
    charset utf-8;

    # server 子目录/路径 localhost:80/
    location / {
        # root 文件匹配的 location 的主目录 html 相对于nginx的目
        root html;
        # 默认页面
        index index.html index.htm;
    }

    # 服务器错误code的对应的页面 对应的地址
    error_page 500 502 503 504 /50x.html;
    # 以上错误对应到这里 用户访问 localhost(或者 当前IP):80/50x.html
    # 会对应到 'html' 这个目录里面找 也就是 'html/50x.html'
    location = /50x.html {
        root html;
    }
}

```

2. 命令

```

# 查看进程 [pid]
[root@localhost ~]# cat /usr/local/nginx/logs/nginx.pid
8858

```

2.1 信号量

信号	作用
TERM/INT	立即关闭整个服务
QUIT	"优雅"地关闭整个服务
HUP	重读配置文件并使用服务对新配置项生效
USR1	重新打开日志文件，可以用来进行日志切割
USR2	平滑升级到最新版的nginx
WINCH	所有子进程不在接收处理新连接，相当于给work进程发送QUIT指令

注意大写

```
# 命令 kill -信号 pid/进程号
kill -QUIT 8858
# kill 信号 `more /usr/local/nginx/logs/nginx.pid`
```

USR2

```
kill -USR2 9104
```

执行这条命令 时 会生成一个新的nginx的master，然后生成新的pid 旧进程的pid文件变成 `pid.oldbin` 当新的进程生成pid成功并且执行成功之后，需要发送 `QUIT` 命令给就旧的进程

发送USR2信号给master进程，告诉master进程要平滑升级，这个时候，会重新开启对应的master进程和work进程，整个系统中将会有两个master进程，并且新的master进程的PID会被记录在 `/usr/local/nginx/logs/nginx.pid` 而之前的旧的master进程PID会被记录

在 `/usr/local/nginx/logs/nginx.pid.oldbin` 文件中，接着再次发送QUIT信号给旧的master进程，让其处理完请求后再进行关闭

```

[root@localhost ~]# kill -USR2 9104
[root@localhost ~]# ls /usr/local/nginx/logs/
access.log  error.log  nginx.pid  nginx.pid.oldbin
[root@localhost ~]# ps -ef | grep nginx
root      9104      1  0 18:26 ?          00:00:00 nginx: master
process /usr/local/nginx/sbin/nginx
nobody    9105    9104  0 18:26 ?          00:00:00 nginx: worker
process
root      9159    9104  0 18:30 ?          00:00:00 nginx: master
process /usr/local/nginx/sbin/nginx
nobody    9160    9159  0 18:30 ?          00:00:00 nginx: worker
process
root      9214    8632  0 18:35 pts/0      00:00:00 grep --color=auto
nginx
[root@localhost ~]# cat /usr/local/nginx/logs/nginx.pid.oldbin
9104
[root@localhost ~]# kill -QUIT 9104
[root@localhost ~]#

```

2.2 命令行控制

- ?和-h:显示帮助信息
- v:打印版本号信息并退出
- V:打印版本号信息和配置信息并退出
- t:测试nginx的配置文件语法是否正确并退出
- T:测试nginx的配置文件语法是否正确并列出用到的配置文件信息然后退出
- q:在配置测试期间禁止显示非错误消息
- tq 只输出错误信息
- s:signal信号, 后面可以跟 :
 - stop[快速关闭, 类似于TERM/INT信号的作用]
 - quit[优雅的关闭, 类似于QUIT信号的作用]
 - reopen[重新打开日志文件类似于USR1信号的作用]
 - reload[类似于HUP信号的作用]
- p:prefix, 指定Nginx的prefix路径, (默认为: /usr/local/nginx/)

-c:filename,指定Nginx的配置文件路径,(默认为: conf/nginx.conf)

-g:用来补充Nginx配置文件, 向Nginx服务指定启动时应用全局的配置

3. 版本升级和模块安装

环境准备

- (1) 先准备两个版本的Nginx分别是 1.14.2和1.16.1
- (2) 使用Nginx源码安装的方式将1.14.2版本安装成功并正确访问

```
进入安装目录
./configure
make && make install
```

- (3) 将Nginx1.16.1进行参数配置和编译, 不需要进行安装。

```
进入安装目录
./configure
make
```

方案一:使用Nginx服务信号进行升级

第一步:将1.14.2版本的sbin目录下的nginx进行备份

```
cd /usr/local/nginx/sbin
mv nginx nginxold
```

第二步:将Nginx1.16.1安装目录编译后的objs目录下的nginx文件, 拷贝到原来 `/usr/local/nginx/sbin` 目录下

```
cd ~/nginx/core/nginx-1.16.1/objs
cp nginx /usr/local/nginx/sbin
```

第三步:发送信号USR2给Nginx的1.14.2版本对应的master进程

第四步:发送信号QUIT给Nginx的1.14.2版本对应的master进程

```
kill -QUIT `more /usr/local/logs/nginx.pid.oldbin`
```

方案二:使用Nginx安装目录的make命令完成升级

第一步:将1.14.2版本的sbin目录下的nginx进行备份

```
cd /usr/local/nginx/sbin  
mv nginx nginxold
```

第二步:将Nginx1.16.1安装目录编译后的objs目录下的nginx文件, 拷贝到原来 `/usr/local/nginx/sbin` 目录下

```
cd ~/nginx/core/nginx-1.16.1/objs  
cp nginx /usr/local/nginx/sbin
```

第三步:进入到**安装目录**, 执行 `make upgrade`

```
[root@localhost objs]# cp nginx /usr/local/nginx/sbin/nginx  
[root@localhost objs]# cd /usr/local/nginx/sbin  
[root@localhost sbin]# ll  
total 7396  
-rwxr-xr-x 1 root root 3825088 Feb 12 03:00 nginx  
-rwxr-xr-x 1 root root 3746336 Feb 12 02:52 nginxold
```

第四步:查看是否更新成功

```
./nginx -v
```

在整个过程中, 其实Nginx是一直对外提供服务的。

配置文件

```
## {  
#   配置文件表达  
#   []: 代表可忽略的  
#   - : 代表默认值  
#   | : 或者  
#   注意分号 [;]  
## }
```

4. 配置文件一 全局块

4.1 user

```
# nginx.conf
# user 用户名 [分组]
# - user nobody ---- {nobody: 没有人的意思}
user wjl;

# 此时再次加载 之后查看进程
[root@localhost nginx]# ps -ef | grep nginx
root      2858      1  0 02:21 ?          00:00:00 nginx: master
process /usr/local/nginx/sbin/nginx
wjl       3601    2858  0 03:28 ?          00:00:00 nginx: worker
process
root      3633    2760  0 03:31 pts/0      00:00:00 grep --color=auto
nginx
```

作用：使用user指令可以指定启动运行工作进程的用户及用户组，这样对于系统的权限访问控制的更加精细，也更加安全。

4.2 master_process

```
# 是否开启工作进程 on|off # 配置玩需要重启服务器 nginx -s stop
systemctl start nginx
# master_process - on
master_process on;

# 关闭
[root@localhost nginx]# ps -ef | grep nginx
root      3886      1  0 03:55 ?          00:00:00
/usr/local/nginx/sbin/nginx
root      3890    2760  0 03:55 pts/0      00:00:00 grep --color=auto
nginx

# 开启
[root@localhost nginx]# ps -ef | grep nginx
```

```

root      3928      1  0 03:56 ?          00:00:00 nginx: master
process /usr/local/nginx/sbin/nginx
wjl       3929    3928  0 03:56 ?          00:00:00 nginx: worker
process
root      3938    2760  0 03:56 pts/0      00:00:00 grep --color=auto
nginx

```

4.3 worker_processes

```

# worker_processes 数字/auto    配置玩需要重启服务器 nginx -s stop
systemctl start nginx
# worker_processes - 1
worker_processes 1;

[root@localhost nginx]# ps -ef | grep nginx
root      4044      1  0 04:02 ?          00:00:00 nginx: master
process /usr/local/nginx/sbin/nginx
wjl       4045    4044  0 04:02 ?          00:00:00 nginx: worker
process
wjl       4046    4044  0 04:02 ?          00:00:00 nginx: worker
process
wjl       4047    4044  0 04:02 ?          00:00:00 nginx: worker
process
wjl       4048    4044  0 04:02 ?          00:00:00 nginx: worker
process
root      4052    2760  0 04:02 pts/0      00:00:00 grep --color=auto
nginx

```

`worker_processes`: 用于配置Nginx生成工作进程的数量，这个是Nginx服务器实现并发处理服务的关键所在。理论上来说`worker_processes`的值越大，可以支持的并发处理量也越多，但事实上这个值的设定是需要受到来自服务器自身的限制，建议将该值和服务器CPU的内核数保存一致。

4.4 daemon


```
# 是否以守护进程的方式运行
# daemon - on/off;
daemon off;

[root@localhost nginx]# systemctl start nginx

# CTRL+C 直接结束进程
```

4.5 pid file

pid:用来配置Nginx当前master进程的进程号ID存储的文件路径。

语法	pid file;
默认值	默认为:/usr/local/nginx/logs/nginx.pid
位置	全局块

该属性可以通过 `./configure --pid-path=PATH` 来指定

4.6 error_log file

error_log:用来配置Nginx的错误日志存放路径

语法	error_log file [日志级别];
默认值	error_log logs/error.log error;
位置	全局块、http、server、location

该属性可以通过 `./configure --error-log-path=PATH` 来指定

其中日志级别的值有：`debug|info|notice|warn|error|crit|alert|emerg`，翻译过来为**试|信息|通知|警告|错误|临界|警报|紧急**，这块建议大家设置的时候不要设置成info以下的等级，因为会带来大量的磁盘I/O消耗，影响Nginx的性能。

```
error_log logs/wjl_error.log debug;

# error_log  路径          等级
error_log    logs/wjl_error.log debug;
```

4.7 include

include:用来引入其他配置文件,使Nginx的配置更加灵活

语法	include file;
默认值	无
位置	任何位置

```
include mime.types;
# 也可以将在别处定义的配置引入到 nginx启动依赖的配置文件[nginx.conf]
# 在里面定义一些通用的 繁琐的 不冲突的
include wjl.conf
include /opt/nginx/wjl.conf;
```

4.10

```
# user 用户名 [分组]
# - user nobody ---- {nobody: 没有人的意思}

user wjl;

# 是否开启工作进程 on|off # 配置玩需要重启服务器 nginx -s stop
systemctl start nginx
# master_process - on
master_process on;

# worker_processes 数字/auto 配置玩需要重启服务器 nginx -s stop
systemctl start nginx
# worker_processes - 1
worker_processes auto;

# 是否以守护进程的方式运行
# daemon - on/off;
daemon on;

# error_log 路径 等级
error_log logs/wjl_error.log debug;
```

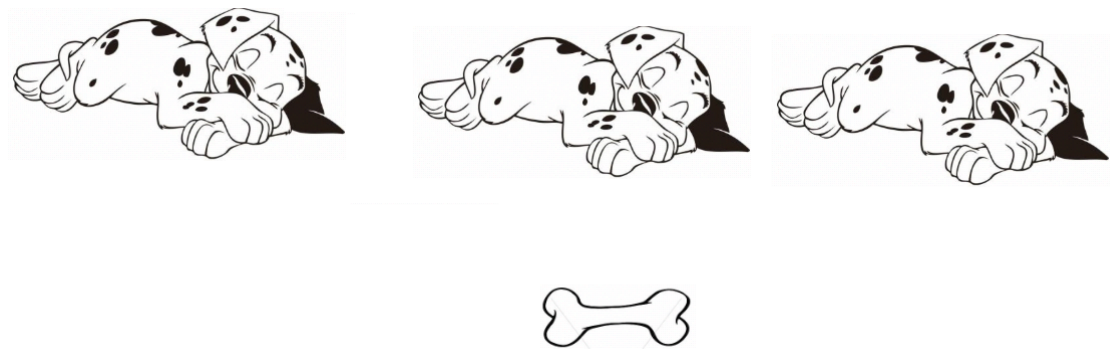
5 配置文件二 -- events

5.1 accept_mutex

accept_mutex:用来设置Nginx网络连接序列化

语法	accept_mutex on off;
默认值	accept_mutex on;
位置	events

这个配置主要可以用来解决常说的"惊群"问题。大致意思是在某一个时刻，客户端发来一个请求连接，Nginx后台是以多进程的工作模式，也就是说有多个worker进程会被同时唤醒，但是最终只会有一个进程可以获取到连接，如果每次唤醒的进程数目太多，就会影响Nginx的整体性能。如果将上述值设置为on(开启状态)，将会对多个Nginx进程接收连接进行序列号，一个个来唤醒接收，就防止了多个进程对连接的争抢。



5.2 multi_accept

multi_accept : 配置一个进程是否可以同时接受多个连接

语法	multi_accept on off;
默认值	multi_accept off;
位置	events

如果multi_accept被禁止了，nginx一个工作进程只能同时接受一个新的连接。否则，一个工作进程可以同时接受所有的新连接

5.3 worker_connections

worker_connections: 用来配置单个worker进程最大的连接数

语法	worker_connections number;
默认值	worker_connections 512;
位置	events

这里的连接数不仅仅包括和前端用户建立的连接数，而是包括所有可能的连接数。另外，number值不能大于操作系统支持打开的最大文件句柄数量。[新版本1024]

5.4 use

use:用来设置Nginx服务器选择哪种事件驱动来处理网络消息。

语法	use method;
默认值	根据操作系统定
位置	events

注意：此处所选择事件处理模型是Nginx优化部分的一个重要内容，method的可选值有select/poll/epoll/kqueue等，之前在准备centos环境的时候，我们强调过要使用linux内核在2.6以上，就是为了能使用epoll函数来优化Nginx。

另外这些值的选择，我们也可以在编译的时候使用

`--with-select_module`、`--without-select_module`、
`--with-poll_module`、`--without-poll_module` 来设置是否需要将对应的事件驱动模块编译到Nginx的内核。

5.10

```
events {  
    # 防止多个worker进程对请求的争抢  
    # accept_mutex - on | off  
    accept_mutex          on;  
  
    # 设置一个进程可以接受多个连接  
    multi_accept          on;  
  
    # 这里的连接数不仅仅包括和前端用户建立的连接数，而是包括所有可能的连接数  
    worker_connections    1024;
```

```
# 用来设置Nginx服务器选择哪种事件驱动来处理网络消息
use                epoll;
}
```

6. 配置文件3 -- http

6.1 mime.type

浏览器中可以显示的内容有HTML、XML、GIF等种类繁多的文件、媒体等资源，浏览器为了区分这些资源，就需要使用MIME Type。所以说MIME Type是网络资源的媒体类型。Nginx作为web服务器，也需要能够识别前端请求的资源类型。

在Nginx的配置文件中，默认有两行配置

```
include mime.types;
# 以上对浏览器的响应无法处理时会触发下面哪一行
default_type application/octet-stream;
# application/octet-stream; 会以下载二进制附件的方式进行操作
```

6.2 default_type

default_type:用来配置Nginx响应前端请求默认的MIME类型。

语法	default_type mime-type;
默认值	default_type text/plain;
位置	http、server、location

```
location /get_text {
    default_type text/plain;
    return 200 "<h1>this is Nginx's Text </h1>";
}
```

当访问地址时

<h1>this is Nginx's Text </h1>

在default_type之前还有一句 `include mime.types` ,include, 相当于把mime.types文件中MIME类型与相关类型文件的文件后缀名的对应关系加入到当前的配置文件中。

举例来说明:

有些时候请求某些接口的时候需要返回指定的文本字符串或者json字符串, 如果逻辑非常简单或者干脆是固定的字符串, 那么可以使用nginx快速实现, 这样就不用编写程序响应请求了, 可以减少服务器资源占用并且响应性能非常快。

如何实现:

```
# 200相当于状态码
location /get_text {
    #这里也可以设置成text/plain
    default_type text/html;
    return 200 "This is nginx's text";
}
location /get_json{
    default_type application/json;
    return 200 '{"name":"TOM","age":18}';
}
```

6.3 自定义服务日志

Nginx中日志的类型分access.log、error.log。

access.log:用来记录用户所有的访问请求。

error.log:记录nginx本身运行时的错误信息, 不会记录用户的访问请求。

Nginx服务器支持对服务日志的格式、大小、输出等进行设置, 需要使用到两个指令, 分别是access_log和log_format指令。

6.3.1 access_log

access_log:用来设置用户访问日志的相关属性。

语法	access_log path [format[buffer=size]]
默认值	access_log logs/access.log combined;
位置	http , server , location

6.3.2 log_format

语法	log_format name [escape=default json none] string....;
默认值	log_format combined "...";
位置	http

log_format 只能定义于http块

access_log 几个区域中就近原则

```
# 定义一个log_format    wjl_logft
log_format    wjl_logft    '"{$remote_addr}"... "{$remote_port}";
# access_log 文件路径 定义的log_format对应格式
access_log    /opt/logs/wjl.log    wjl_logft;
```

6.4 send file

sendfile:用来设置Nginx服务器是否使用sendfile()传输文件，该属性可以大大提高Nginx处理静态资源的性能

语法	sendfile on off;
默认值	sendfile off;
位置	http、server、location

```
# 零拷贝 [减少一次调度的过程]
sendfile    on;
```

6.5 keepalive_timeout

为什么要使用keepalive?

我们都知道HTTP是一种无状态协议，客户端向服务端发送一个TCP请求，服务端响应完毕后断开连接。

如何客户端向服务端发送多个请求，每个请求都需要重新创建一次连接，效率相对来说比较多，使用keepalive模式，可以告诉服务器端在处理完一个请求后保持这个TCP连接的打开状态，若接收到来自这个客户端的其他请求，服务端就会利用这个未被关闭的连接，而不需要重新创建一个新连接，提升效率，但是这个连接也不能一直保持，这样的话，连接如果过多，也会是服务端的性能下降，这个时候就需要我们进行设置其的超时时间。

语法	<code>keepalive_timeout time;</code>
默认值	<code>keepalive_timeout 75s;</code>
位置	<code>http、server、location</code>

```
# 保持连接超时
keepalive_timeout 65;
```

6.6 keepalive_requests

语法	<code>keepalive_requests number;</code>
默认值	<code>keepalive_requests 100;</code>
位置	<code>http、server、location</code>

`keepalive_requests`指令用于设置一个keep-alive连接上可以服务的请求的最大数量，当最大请求数量达到时，连接被关闭。默认是100。

这个参数的真实含义，是指一个keep alive建立之后，nginx就会为这个连接设置一个计数器，记录这个keep alive的长连接上已经接收并处理的客户端请求的数量。如果达到这个参数设置的最大值时，则nginx会强行关闭这个长连接，逼迫客户端不得不重新建立新的长连接。

[相关链接](#)

6.10

```
## {
#   配置文件表达
#       []: 代表可忽略的
#       - : 代表默认值
#       | : 或者
#   注意分号 [;]
## }

#   user 用户名 [分组]
# - user nobody ---- {nobody: 没有人的意思}

user wjl;

# 是否开启工作进程 on|off # 配置玩需要重启服务器 nginx -s stop
systemctl start nginx
# master_process - on
master_process on;

# worker_processes 数字/auto 配置玩需要重启服务器 nginx -s stop
systemctl start nginx
# worker_processes - 1
worker_processes auto;

# 是否以守护进程的方式运行
# daemon - on/off;
daemon on;

# error_log 路径 等级
error_log logs/wjl_error.log debug;

events {
    # 防止多个worker进程对请求的争抢
    # accept_mutex - on | off
    accept_mutex on;

    # 设置一个进程可以接受多个连接
    multi_accept on;

    # 这里的连接数不仅仅包括和前端用户建立的连接数，而是包括所有可能的连接数
    worker_connections 1024;

    # 用来设置Nginx服务器选择哪种事件驱动来处理网络消息
    use epoll;
```

```
}
```

```
http {
```

```
# 引入配置文件 服务器给浏览器的请求头 对应的文件后缀名
```

```
include mime.types;
```

```
# 默认的类型 如果 [mime.types] 不包含 会在这里面选
```

```
default_type application/octet-stream;
```

```
# 零拷贝 [减少一次调度的过程]
```

```
sendfile on;
```

```
# 保持连接超时
```

```
keepalive_timeout 65;
```

```
include /opt/nginx/wjl.conf;
```

```
# 定义一个log_format wjl_logft
```

```
log_format wjl_logft '{"$remote_addr}" ... "{$remote_port}";
```

```
# access_log 文件路径 定义的log_format对应格式
```

```
access_log /opt/logs/wjl.log wjl_logft;
```

```
# 代表的“主机” [虚拟主机{vhost}]
```

```
server {
```

```
# 监听的端口号
```

```
listen 80;
```

```
# 域名/主机名
```

```
server_name localhost;
```

```
# 文字编码
```

```
charset utf-8;
```

```
# server 子目录/路径 localhost:80/
```

```
location / {
```

```
# root 文件匹配的 location 的主目录 html 相对于nginx的目
```

```
root html;
```

```
# 默认页面
```

```
index index.html index.htm;
```

```
}
```

```
# 服务器错误code的对应的页面 对应的地址
```

```
# 404 是自己自己额外添加的 当找不到页面时同样会跳转到这里
```

```
error_page 500 502 503 504 404 /50x.html;
```

```
# 以上错误对应到这里 用户访问 localhost(或者 当前IP):80/50x.html
```

录

```

# 会对应到 'html' 这个目录里面找 也就是 'html/50x.html'
location = /50x.html {
    root    html;
}

# 200相当于状态码
location /get_text {
    default_type    text/html;
    return 200 "<h1>this is Nginx's Text </h1>";
}

location /get_json {
    default_type    application/json;
    return 404 '{"name":"TOM","age":18}';
}
}
}

```

7. 配置文件4 -- server 和 location

7.1 相应的小练习

```

# nginx.conf
user    wjl;
worker_processes  2;
error_log  logs/error.log;
pid  logs/nginx.pid;
daemon  on;

events {
    accept_mutex  on;
    multi_accept  on;
    worker_connections  1024;
    use  epoll;
}

http {
    include        mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;
}

```

```

log_format server1 '=====> Server1 Log';
log_format server2 '=====> Server2 Log';

# 引入这个目录下所有以 '.conf' 结尾的文件 '*' 通配符
include /home/wjl/conf.d/*.conf;
# 由于笔记直接写 * 会导致笔记代码无色 所以额外加 []
server {
    listen      80;
    server_name localhost;
    charset utf-8;

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

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root    html;
    }

}

}

# 配置文件1
server {
    listen      8080;
    server_name localhost;
    charset utf-8;

    access_log /home/wjl/myweb/server1/logs/access.log
server1;

    # 访问 location:8080/server1/location1
    location /server1/location1 {
        root /home/wjl/myweb;
        index index_str_location1.html;
    }
    location /server1/location2 {
        root /home/wjl/myweb;
        index index_str_location2.html;
    }
}

```

```

        error_page 404 /404.html;
        location =/404.html {
            root /home/wjl/myweb;
            index 404.html;
        }
    }

# 配置文件2
server {
    listen      8000;
    server_name localhost;
    charset     utf-8;

    access_log /home/wjl/myweb/server2/logs/access.log

server2;

    # 访问 location:8000/server2/location1
    location /server2/location1 {
        root /home/wjl/myweb;
        index index_str_location1.html;
    }
    location /server2/location2 {
        root /home/wjl/myweb;
        index index_str_location2.html;
    }

    error_page 404 /404.html;
    location =/404.html {
        root /home/wjl/myweb;
        index 404.html;
    }
}

```

静态资源部署

8 配置文件

通过浏览器发送一个HTTP请求实现从客户端发送请求到服务器端获取所需要内容后并把内容回显展示在页面的一个过程。这个时候，我们所请求的内容就分为两种类型，一类是静态资源、一类是动态资源。静态资源即指在服务器端真实存在并且能直接拿来展示的一些文件，比如常见的html页面、css文件、js文件、图片、视频等资源；动态资源即指在服务器端真实存在但是要想获取需要经过一定的业务逻辑处理，根据不同的条件展示在页面不同这一部分内容，比如说报表数据展示、根据当前登录用户展示相关具体数据等资源；

Nginx处理静态资源的内容，我们需要考虑下面这几个问题：

- (1) 静态资源的配置指令
- (2) 静态资源的配置优化
- (3) 静态资源的压缩配置指令
- (4) 静态资源的缓存处理
- (5) 静态资源的访问控制，包括跨域问题和防盗链问题

[相关文档](#)

8.1 listen指令

listen:用来配置监听端口。

语法	<code>listen address[:port] [default_server]...;</code> <code>listen port [default_server]...;</code>
默认值	<code>listen *:80 *:8000</code>
位置	server

listen的设置比较灵活

```
listen 127.0.0.1:8000; // listen localhost:8000 监听指定的IP和端口
listen 127.0.0.1;    监听指定IP的所有端口
listen 8000;         监听指定端口上的连接
listen *:8000;       监听指定端口上的连接
```

同一端口访问时 默认会使用第一个 也就是 `default_server` ,也可以通过listen设置

```
listen      8000  default_server;
server_name localhost;
```

default_server属性是标识符，用来将此虚拟主机设置成默认主机。所谓的默认主机指的是如果没有匹配到对应的address:port，则会默认执行的。如果不指定默认使用的是第一个server

```
server{
    listen 8080;
    server_name 127.0.0.1;
```

```

        location /{
            root html;
            index index.html;
        }
    }
    server{
        listen 8080 default_server;
        server_name localhost;
        default_type text/plain;
        return 444 'This is a error request';
    }
}

```

8.2 server_name

server_name：用来设置虚拟主机服务名称。

127.0.0.1 、 localhost 、 域名[www.baidu.com | www.jd.com]

语法	server_name name ...; name可以提供多个中间用空格分隔
默认值	server_name "";
位置	server

关于server_name的配置方式有三种，分别是：

精确匹配
通配符匹配
正则表达式匹配

配置方式一：精确匹配

如：

```

server {
    listen 80;
    server_name www.itcast.cn www.itheima.cn;
    ...
}

```

补充小知识：

hosts是一个没有扩展名的系统文件，可以用记事本等工具打开，其作用就是将一些常用的网址域名与其对应的IP地址建立一个关联“数据库”，当用户在浏览器中输入一个需要登录的网址时，系统会首先自动从hosts文件中寻找对应的IP地址，一旦找到，系统会立即打开对应网页，如果没有找到，则系统会再将网址提交DNS域名解析服务器进行IP地址的解析。

windows:C:\Windows\System32\drivers\etc

centos: /etc/hosts

因为域名是要收取一定的费用，所以我们可以使用修改hosts文件来制作一些虚拟域名来使用。
需要修改 `/etc/hosts` 文件来添加

```
vim /etc/hosts
127.0.0.1 www.itcast.cn
127.0.0.1 www.itheima.cn
```

配置方式二:使用通配符配置

server_name中支持通配符"*",但需要注意的是通配符不能出现在域名的中间，只能出现在首段或尾段，如：

```
server {
    listen 80;
    server_name *.itcast.cn www.itheima.*;
    # www.itcast.cn abc.itcast.cn www.itheima.cn www.itheima.com
    ...
}
```

下面的配置就会报错

```
server {
    listen 80;
    server_name www.*.cn www.itheima.c*
    ...
}
```

配置三:使用正则表达式配置

server_name中可以使用正则表达式，并且使用 `~` 作为正则表达式字符串的开始标记。

常见的正则表达式

代码	说明
^	匹配搜索字符串开始位置
\$	匹配搜索字符串结束位置
.	匹配除换行符\n之外的任何单个字符
\	转义字符，将下一个字符标记为特殊字符
[xyz]	字符集，与任意一个指定字符匹配
[a-z]	字符范围，匹配指定范围内的任何字符
\w	与以下任意字符匹配 A-Z a-z 0-9 和下划线,等效于[A-Za-z0-9_]
\d	数字字符匹配，等效于[0-9]
{n}	正好匹配n次
{n,}	至少匹配n次
{n,m}	匹配至少n次至多m次
*	零次或多次，等效于{0,}
+	一次或多次，等效于{1,}
?	零次或一次，等效于{0,1}

配置如下：

```
server{
    listen 80;
    server_name ~^www\.(\w+)\.com$;
    default_type text/plain;
    return 200 $1 $2 ..;
}
```

注意 ~后面不能加空格，括号可以取值

注意 ~后面不能加空格，括号可以取值

```
listen      8080;
# server_name www.wjl1128.top;
# 正则表达式以 '~'开头 中间不能有空格 (\w+) '()' 代表括号表达式 可以用
'$[括号序号]'
server_name ~^www\.\(\w+\)\.com$;
charset utf-8;

location /wjl {
    default_type text/plain;
    return 200 "=====>$1";
}
```

由于server_name指令支持通配符和正则表达式，因此在包含多个虚拟主机的配置文件中，可能会出现一个名称被多个虚拟主机的server_name匹配成功，当遇到这种情况，当前的请求交给谁来处理呢？

No1:准确匹配server_name

No2:通配符在开始时匹配server_name成功

No3:通配符在结束时匹配server_name成功

No4:正则表达式匹配server_name成功

No5:被默认的default_server处理，如果没有指定默认找第一个server

8.3 location

location:用来设置请求的URI

语法	location [= ~ ~* ^~ @] uri{...}
默认值	-
位置	server,location

uri变量是待匹配的请求字符串，可以不包含正则表达式，也可以包含正则表达式，那么nginx服务器在搜索匹配location的时候，**是先使用不包含正则表达式进行匹配，找到一个匹配度最高的一个，然后在通过包含正则表达式的进行匹配，如果能匹配到直接访问，匹配不到，就使用刚才匹配度最高的那个location来处理请求。**

属性介绍：

不带符号，要求必须以指定模式开始

```
server {
    listen 80;
    server_name 127.0.0.1;
    location /abc{
        default_type text/plain;
        return 200 "access success";
    }
}
```

以下访问都是正确的

```
http://192.168.200.133/abc
http://192.168.200.133/abc?p1=TOM
http://192.168.200.133/abc/
http://192.168.200.133/abcdef
```

= : 用于不包含正则表达式的uri前，必须与指定的模式精确匹配

```
server {
    listen 80;
    server_name 127.0.0.1;
    location =/abc{
        default_type text/plain;
        return 200 "access success";
    }
}
```

可以匹配到

```
http://192.168.200.133/abc
http://192.168.200.133/abc?p1=TOM
```

匹配不到

```
http://192.168.200.133/abc/
http://192.168.200.133/abcdef
```

~ : 用于表示当前uri中包含了正则表达式，并且区分大小写

~*: 用于表示当前uri中包含了正则表达式，并且不区分大小写

换句话说，如果uri包含了正则表达式，需要用上述两个符合来标识

```
server {
    listen 80;
    server_name 127.0.0.1;
    location ~^/abc\w${
        default_type text/plain;
        return 200 "access success";
    }
}

server {
    listen 80;
```

```

server_name 127.0.0.1;
location ~*^/abc/w${
    default_type text/plain;
    return 200 "access success";
}
}

```

^~： 用于不包含正则表达式的uri前，功能和不加符号的一致，唯一不同的是，如果模式匹配，那么就停止搜索其他模式了。

```

server {
    listen 80;
    server_name 127.0.0.1;
    location ^~/abc{
        default_type text/plain;
        return 200 "access success";
    }
}

```

8.4 请求资源目录 root/alias

root：设置请求的根目录

语法	root path;
默认值	root html;
位置	http、server、location

path为Nginx服务器接收到请求以后查找资源的根目录路径。

alias：用来更改location的URI

语法	alias path;
默认值	—
位置	

path为修改后的根路径。

以上两个指令都可以来指定访问资源的路径，那么这两者之间的区别是什么？

举例说明：

(1) 在 `/usr/local/nginx/html` 目录下创建一个 `images`目录,并在目录下放入一张图片 `mv.png` 图片

```
location /images {  
    root /usr/local/nginx/html;  
}
```

访问图片的路径为:

```
http://192.168.200.133/images/mv.png
```

(2) 如果把`root`改为`alias`

```
location /images {  
    alias /usr/local/nginx/html;  
}
```

再次访问上述地址,页面会出现404的错误,查看错误日志会发现是因为地址不对,所以验证了:

root的处理结果是: root路径+location路径
`/usr/local/nginx/html/images/mv.png`

alias的处理结果是:使用alias路径替换location路径
`/usr/local/nginx/html/images`

需要在`alias`后面路径改为

```
location /images {  
    alias /usr/local/nginx/html/images;  
}  
  
# 访问http://www.wjl1128.top/wjl/123.webp 出现相应的图片  
location /wjl/ {  
    alias /usr/local/nginx/html/;  
}
```

(3) 如果`location`路径是以`/`结尾,则`alias`也必须是以`/`结尾, `root`没有要求

将上述配置修改为

```
location /images/ {  
    alias /usr/local/nginx/html/images;  
}
```

访问就会出问题,查看错误日志还是路径不对,所以需要把`alias`后面加上 `/`

小结:

root的处理结果是: root路径+location路径
alias的处理结果是:使用alias路径替换location路径
alias是一个目录别名的定义, root则是最上层目录的含义。
如果location路径是以/结尾,则alias也必须是以/结尾, root没有要求

8.5 index

index:设置网站的默认首页

语法	<code>index file ...;</code>
默认值	<code>index index.html;</code>
位置	<code>http、server、location</code>

index后面可以跟多个设置, 如果访问的时候没有指定具体访问的资源, 则会依次进行查找, 找到第一个为止。

举例说明:

```
location / {  
    root /usr/local/nginx/html;  
    index index.html index.htm;  
}
```

访问该location的时候, 可以通过 `http://ip:port/`, 地址后面如果不添加任何内容, 则默认依次访问index.html和index.htm, 找到第一个来进行返回

8.6 error_page

error_page:设置网站的错误页面

语法	<code>error_page code ... [=[response]] uri;</code>
默认值	<code>-</code>
位置	<code>http、server、location.....</code>

当出现对应的响应code后, 如何来处理。

举例说明:

(1) 可以指定具体跳转的地址

```
server {  
    error_page 404 http://www.baidu.com;  
}
```

(2) 可以指定重定向地址

```
server{
    error_page 404 /50x.html;
    error_page 500 502 503 504 /50x.html;
    location =/50x.html{
        root html;
    }
}
```

(3) 使用location的@符合完成错误信息展示

```
server{
    error_page 404 @jump_to_error;
    location @jump_to_error {
        default_type text/plain;
        return 404 'Not Found Page...';
    }
}

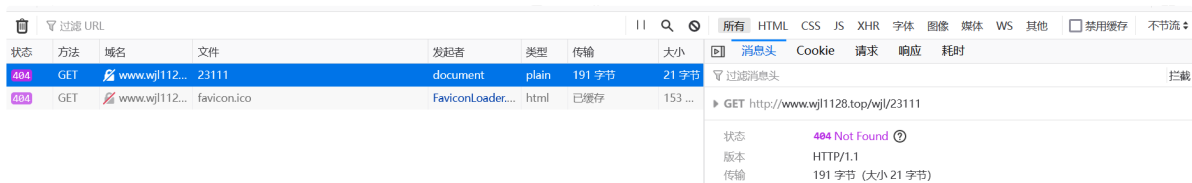
error_page 404 @wjl_error;
location @wjl_error {
    default_type text/plain;
    return 200 '发生错误!!!';
}
```

可选项 `=[response]` 的作用是用来将相应代码更改为另外一个

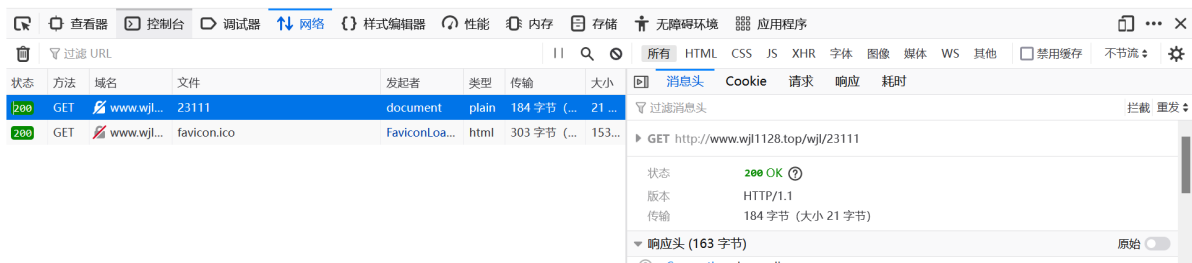
```
server{
    error_page 404 =200 /50x.html;
    location =/50x.html{
        root html;
    }
}
```

这样的话，当返回404找不到对应的资源的时候，在浏览器上可以看到，最终返回的状态码是200，这块需要注意下，编写error_page后面的内容，404后面需要加空格，200前面不能加空格

发生错误界面指定页面之后跳转的状态码任然是 404



设置修改状态码之后



这样的话，当返回404找不到对应的资源的时候，在浏览器上可以看到，最终返回的状态码是200，这块需要注意下，编写error_page后面的内容，404后面需要加空格，200前面不能加空格

9 静态资源优化

9.1 sendfile

(1) sendfile，用来开启高效的文件传输模式。

语法	<code>sendfile on off;</code>
默认值	<code>sendfile off;</code>
位置	<code>http、server、location...</code>

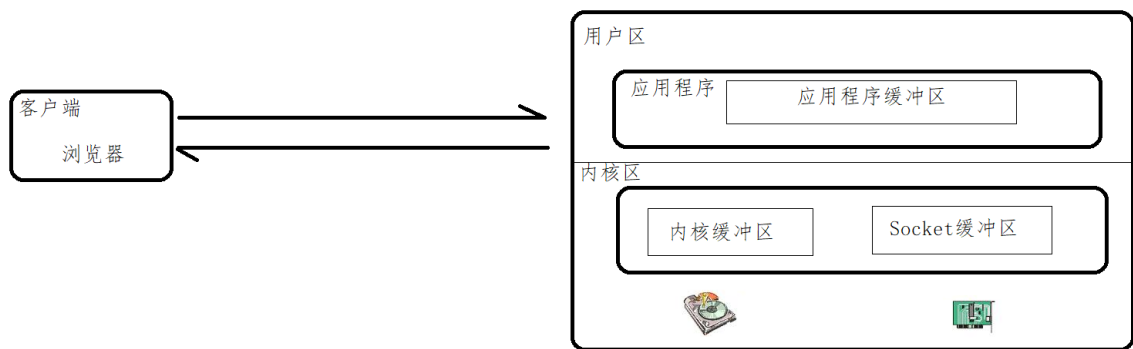
请求静态资源的过程：客户端通过网络接口向服务端发送请求，操作系统将这些客户端的请求传递给服务器端应用程序，服务器端应用程序会处理这些请求，请求处理完成以后，操作系统还需要将处理得到的结果通过网络适配器传递回去。

如：

```
server {
    listen 80;
    server_name localhost;
    location / {
        root html;
        index index.html;
    }
}
```

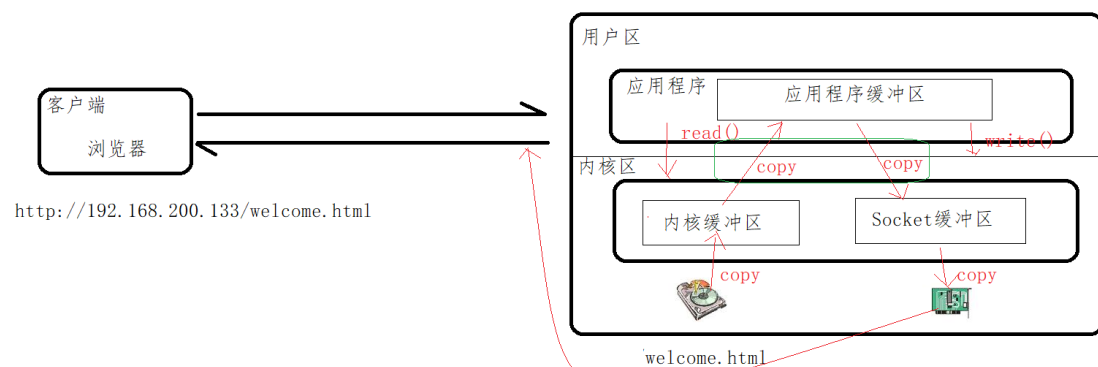
在html目录下有一个welcome.html页面，访问地址
`http://192.168.200.133/welcome.html`

未配置之前

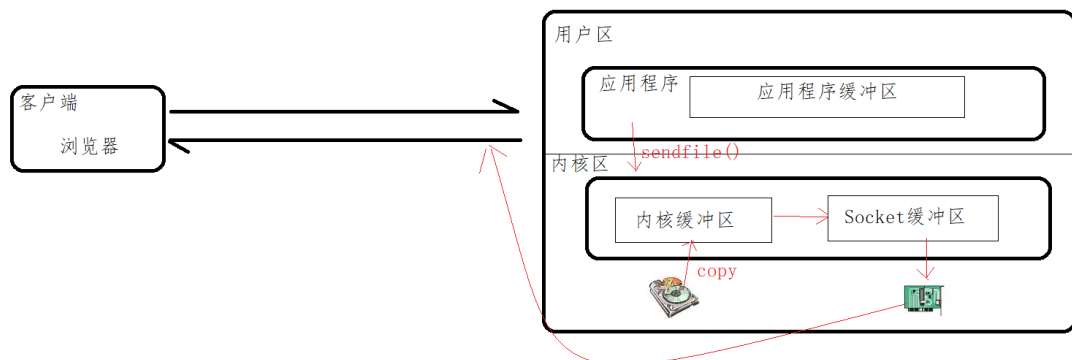


配置之后

未使用sendfile的处理流程



使用sendfile的处理流程



9.2 tcp_nopush

tcp_nopush: 该指令必须在sendfile打开的状态下才会生效，主要是用来提升网络包的传输'效率'

语法	<code>tcp_nopush on off;</code>
默认值	<code>tcp_nopush off;</code>
位置	<code>http、server、location</code>

9.3 tcp_nodelay

tcp_nodelay: 该指令必须在keep-alive连接开启的情况下才生效，来提高网络包传输的'实时性'

语法	<code>tcp_nodelay on off;</code>
默认值	<code>tcp_nodelay on;</code>
位置	<code>http、server、location</code>

`tcp_nopush`和`tcp_nodelay`“看起来是”互斥的”，那么为什么要将这两个值都打开，这个大家需要知道的是在linux2.5.9以后的版本中两者是可以兼容的，三个指令都开启的好处是，`sendfile`可以开启高效的文件传输模式，`tcp_nopush`开启可以确保在发送到客户端之前数据包已经充分“填满”，这大大减少了网络开销，并加快了文件发送的速度。然后，当它到达最后一个可能因为没有“填满”而暂停的数据包时，Nginx会忽略`tcp_nopush`参数，然后，`tcp_nodelay`强制套接字发送数据。由此可知，TCP_NOPUSH可以与TCP_NODELAY一起设置，它比单独配置TCP_NODELAY具有更强的性能。所以我们可以使用如下配置来优化Nginx静态资源的处理

```
sendfile on;
tcp_nopush on;
tcp_nodelay on;
```

10 静态资源压缩

假如在满足上述优化的前提下，我们传送一个1M的数据和一个10M的数据那个效率高？，答案显而易见，传输内容小，速度就会快。那么问题又来了，同样的内容，如果把大小降下来，我们脑袋里面要蹦出一个词就是“压缩”，接下来，我们来学习Nginx的静态资源压缩模块。

在Nginx的配置文件中可以通过配置gzip来对静态资源进行压缩，相关的指令可以配置在http块、server块和location块中，Nginx可以通过

```
ngx_http_gzip_module 模块 # nginx 安装内置
ngx_http_gzip_static_module 模块
ngx_http_gunzip_module 模块
```

对这些指令进行解析和处理。

10.1 ngx_http_gzip_module

10.1.1

1. gzip指令：该指令用于开启或者关闭gzip功能

语法	<code>gzip on off;</code>
默认值	<code>gzip off;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code> ...

注意只有该指令为打开状态，下面的指令才有效果

```
http{
    gzip on;
}
```

注意 开启此选项不代表会压缩 只是代表允许压缩操作

10.1.2 gzip_types

2. gzip_types指令：该指令可以根据响应页的MIME类型选择性地开启Gzip压缩功能

语法	<code>gzip_types mime-type ...;</code>
默认值	<code>gzip_types text/html;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code>

所选择的值可以从mime.types文件中进行查找，也可以使用"*"代表所有。

```
http{
    # 根据 js开启压缩
    gzip_types application/javascript;
}
```

开启前

状态	方法	域名	文件	发起者	类型	传输	大小	耗时
200	GET	www.wj1128.top	jquery.js	document	js	282.08 KB	281.82 KB	10 毫秒
200	GET	www.wj1128.top	favicon.ico	FaviconLoaderjsm:191 (img)	x-icon	171 字节	21 字节	1 毫秒

开启后

`gzip_types application/javascript;`

状态	方法	域名	文件	发起者	类型	传输	大小	耗时
200	GET	www.wj1128.top	jquery.js	document	js	101.47 KB	281.82 KB	18 毫秒
200	GET	www.wj1128.top	favicon.ico	FaviconLoaderjsm:191 (img)	x-icon	171 字节	21 字节	0 毫秒

多个之间用空格隔开

```
# 开启压缩 新版默认 压缩 `text/html`? 视频不建议
gzip on;
gzip_types application/javascript image/webp video/mp4;
```

10.1.3 gzip_comp_level

3. gzip_comp_level指令：该指令用于设置Gzip压缩程度，级别从1-9,1表示要是程度最低，要是效率最高，9刚好相反，压缩程度最高，但是效率最低最费时间。

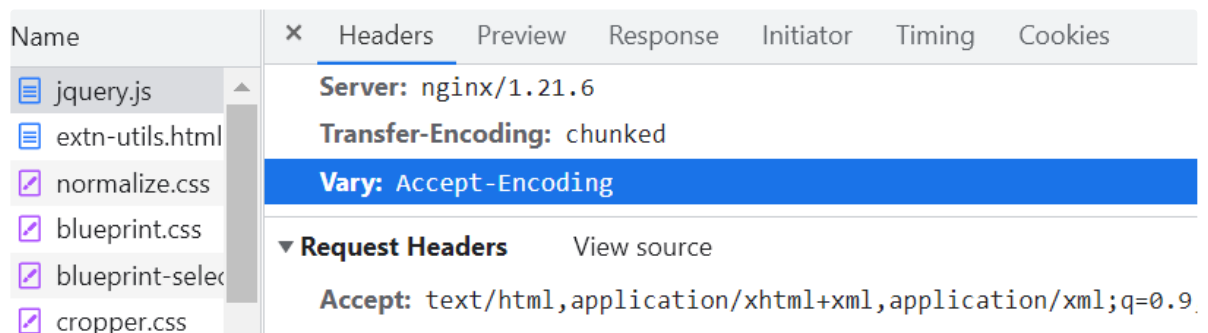
语法	gzip_comp_level level;
默认值	gzip_comp_level 1;
位置	http、server、location

```
http{
    gzip_comp_level 6;
}
```

10.1.4 gzip_vary

4. gzip_vary指令：该指令用于设置使用Gzip进行压缩发送是否携带“Vary: Accept-Encoding”头域的响应头部。主要是告诉接收方，所发送的数据经过了Gzip压缩处理

语法	gzip_vary on off;
默认值	gzip_vary off;
位置	http、server、location



10.1.5 gzip_buffers

gzip_buffers指令：该指令用于处理请求压缩的缓冲区数量和大小。

语法	gzip_buffers number size;
默认值	gzip_buffers 32 4k 16 8k;
位置	http、server、location

其中number:指定Nginx服务器向系统申请缓存空间个数，size指的是每个缓存空间的大小。主要实现的是申请number个每个大小为size的内存空间。**这个值的设定一般会和服务器的操作系统有关，所以建议此项不设置，使用默认值即可。**

```
gzip_buffers 4 16K;    #缓存空间大小
```

10.1.6 gzip_disable

6. gzip_disable指令：针对不同种类客户端发起的请求，可以选择性地开启和关闭Gzip功能。

语法	gzip_disable regex ...;
默认值	—
位置	http、server、location

regex:根据**客户端的浏览器标志(user-agent)**来设置，支持使用正则表达式。指定的浏览器标志不使用Gzip.该指令一般是用来排除一些明显不支持Gzip的浏览器。

```
gzip_disable "MSIE [1-6]\.";
```

10.1.7 gzip_http_version

7. gzip_http_version指令：针对不同的HTTP协议版本，可以选择性地开启和关闭Gzip功能。

语法	gzip_http_version 1.0 1.1;
默认值	gzip_http_version 1.1;
位置	http、server、location

该指令是指定使用Gzip的HTTP最低版本，该指令一般采用默认值即可。

10.1.8 gzip_min_length

8. `gzip_min_length`指令：该指令针对传输数据的大小，可以选择性地开启和关闭Gzip功能

语法	<code>gzip_min_length length;</code>
默认值	<code>gzip_min_length 20;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code>

nignx计量大小的单位：`bytes`[字节] / `kb`[千字节] / `M`[兆]
例如：`1024` / `10k|K` / `10m|M`
`gzip_min_length 1M; # 设置为1m 浏览器请求的资源小于这个值就不会进行压缩`

Gzip压缩功能对大数据的压缩效果明显，但是**如果要压缩的数据比较小的化，可能出现越压缩数据量越大的情况**，因此我们需要根据响应内容的大小来决定是否使用Gzip功能，响应页面的大小可以通过头信息中的 `Content-Length` 来获取。但是如何使用了Chunk编码动态压缩，该指令将被忽略。建议设置为1K或以上。

10.1.9 gzip_proxied

9. `gzip_proxied`指令：该指令设置是否对**服务端[反向代理的服务器 比如tomcat]**返回的结果进行Gzip压缩。

语法	<code>gzip_proxied off expired no-cache no-store private no_last_modified no_etag auth any;</code>
默认值	<code>gzip_proxied off;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code>

`off` - 关闭Nginx服务器对后台服务器返回结果的Gzip压缩
`expired` - 启用压缩，如果header头中包含 "Expires" 头信息
`no-cache` - 启用压缩，如果header头中包含 "Cache-Control:no-cache" 头信息
`no-store` - 启用压缩，如果header头中包含 "Cache-Control:no-store" 头信息
`private` - 启用压缩，如果header头中包含 "Cache-Control:private" 头信息
`no_last_modified` - 启用压缩,如果header头中不包含 "Last-Modified" 头信息
`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
`any` - 无条件启用压缩

10.1 Gzip压缩功能的实例配置

```

gzip on;                #开启gzip功能
gzip_types *;           #压缩源文件类型,根据具体的访问资源类型设定 生产环境
                        不要 [*]
gzip_comp_level 6;      #gzip压缩级别
gzip_min_length 1024;   #进行压缩响应页面的最小长度,content-length
gzip_buffers 4 16K;     #缓存空间大小 [可省略]
gzip_http_version 1.1;  #指定压缩响应所需要的最低HTTP请求版本
gzip_vary on;           #往头信息中添加压缩标识
gzip_disable "MSIE [1-6]\."; #对IE6以下的版本都不进行压缩
gzip_proxied off;       #nginx作为反向代理压缩服务端返回数据的条件

```

这些配置在很多地方可能都会用到,所以我们可以将这些内容抽取到一个配置文件中,然后通过include指令把配置文件再次加载到nginx.conf配置文件中,方法使用。

nginx_gzip.conf

```

gzip on;
gzip_types *;
gzip_comp_level 6;
gzip_min_length 1024;
gzip_buffers 4 16K;
gzip_http_version 1.1;
gzip_vary on;
gzip_disable "MSIE [1-6]\.";
gzip_proxied off;

```

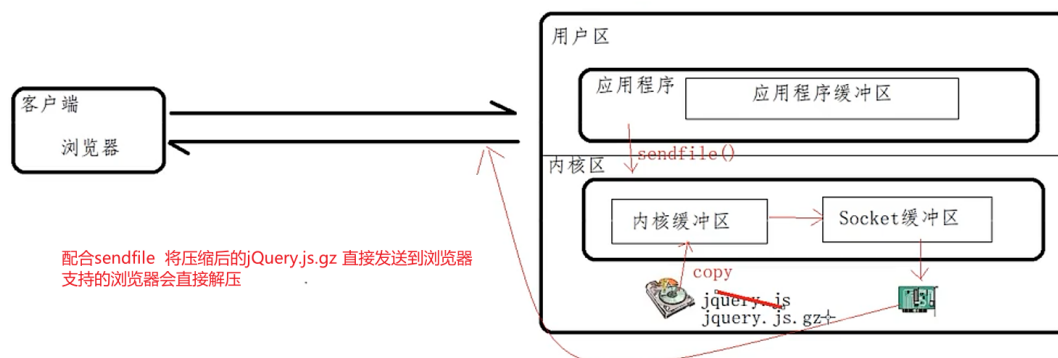
nginx.conf

```
include nginx_gzip.conf
```

10.1 sendfile和gz共存问题[引出]

开启sendfile以后,在读取磁盘上的静态资源文件的时候,可以减少拷贝的次数,可以**不经过用户进程将静态文件通过网络设备发送出去**,但是Gzip要想对资源压缩,是需要经过用户进程进行操作的。所以如何解决两个设置的共存问题。

可以使用ngx_http_gzip_static_module模块的gzip_static指令来解决。



10.2 ngx_http_gzip_static_module

[相关文档](#)

10.2.1 模块安装

1. nginx -V 查询到相关配置

```
[root@localhost nginx]# nginx -V
nginx version: nginx/1.21.6
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-44) (GCC)
configure arguments: `--prefix=/usr/local/nginx --sbin-
path=/usr/local/nginx/sbin/nginx --modules-
path=/usr/local/nginx/modules --conf-
path=/usr/local/nginx/conf/nginx.conf --error-log-
path=/usr/local/nginx/logs/error.log --http-log-
path=/usr/local/nginx/logs/access.log --pid-
path=/usr/local/nginx/logs/nginx.pid --lock-
path=/usr/local/nginx/logs/nginx.lock`
```

2. 将nginx安装目录下的sbin下面的nginx可执行文件 备份更名

```
mv /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginxold
```

3. 进入安装目录[不是指定的安装 是之前执行 `make && make install` 的目录] 清理一下

```
cd /opt/app/nginx-1.21.6/
make clean
```

4. 将之前查询的附加上

```
./configure --prefix=/usr/local/nginx --sbin-
path=/usr/local/nginx/sbin/nginx --modules-
path=/usr/local/nginx/modules --conf-
path=/usr/local/nginx/conf/nginx.conf --error-log-
path=/usr/local/nginx/logs/error.log --http-log-
path=/usr/local/nginx/logs/access.log --pid-
path=/usr/local/nginx/logs/nginx.pid --lock-
path=/usr/local/nginx/logs/nginx.lock
```



```
./configure --prefix=/usr/local/nginx --sbin-
path=/usr/local/nginx/sbin/nginx --modules-
path=/usr/local/nginx/modules --conf-
path=/usr/local/nginx/conf/nginx.conf --error-log-
path=/usr/local/nginx/logs/error.log --http-log-
path=/usr/local/nginx/logs/access.log --pid-
path=/usr/local/nginx/logs/nginx.pid --lock-
path=/usr/local/nginx/logs/nginx.lock
# 添加上面这一段
--with-http_gzip_static_module

# 出现以下就算成功
Configuration summary
+ using system PCRE library
+ OpenSSL library is not used
+ using system zlib library

nginx path prefix: "/usr/local/nginx"
nginx binary file: "/usr/local/nginx/sbin/nginx"
nginx modules path: "/usr/local/nginx/modules"
nginx configuration prefix: "/usr/local/nginx/conf"
nginx configuration file: "/usr/local/nginx/conf/nginx.conf"
nginx pid file: "/usr/local/nginx/logs/nginx.pid"
nginx error log file: "/usr/local/nginx/logs/error.log"
nginx http access log file: "/usr/local/nginx/logs/access.log"
nginx http client request body temporary files:
"client_body_temp"
nginx http proxy temporary files: "proxy_temp"
nginx http fastcgi temporary files: "fastcgi_temp"
nginx http uwsgi temporary files: "uwsgi_temp"
nginx http scgi temporary files: "scgi_temp"
```

5. 执行 `make`

6. 进入 `objs` 将nginx二进制文件拷贝到 `/usr/local/nginx/sbin` 下 接着在
nginx下载解压[也就是执行make]目录下执行 `make upgrade`

```
cd objs/
cp nginx /usr/local/nginx/sbin/
```

```
cd ../
make upgrade
```

以下安装成功的示例

```
[root@localhost nginx-1.21.6]# make upgrade
/usr/local/nginx/sbin/nginx -t
nginx: the configuration file /usr/local/nginx/conf/nginx.conf
syntax is ok
```

```
nginx: configuration file /usr/local/nginx/conf/nginx.conf test is
successful
kill -USR2 `cat /usr/local/nginx/logs/nginx.pid`
sleep 1
test -f /usr/local/nginx/logs/nginx.pid.oldbin
kill -QUIT `cat /usr/local/nginx/logs/nginx.pid.oldbin`
```

10.2.2 使用 gzip_static

gzip_static: 检查与访问资源同名的.gz文件时，response中以gzip相关的header返回.gz文件的内容。

注意 【先关闭 `gzip off`】

语法	gzip_static on off always;
默认值	gzip_static off;
位置	http、server、location

`always` 表示无论浏览器支不支持 直接发送 .gz文件

开启后 会在客户端所访问的静态资源的目录下寻找 同名的 .gz

执行 `[gzip jquery.js]` 命令 可以压缩出

###

11 静态资源的缓存处理

1. 什么是缓存?

缓存 (cache)，原始意义是指访问速度比一般随机存取存储器 (RAM) 快的一种高速存储器，通常它不像系统主存那样使用DRAM技术，而使用昂贵但较快速的SRAM技术。缓存的设置是所有现代计算机系统发挥高性能的重要因素之一。

2. 什么是web缓存

Web缓存是指一个Web资源 (如html页面，图片，js，数据等) 存在于Web服务器和客户端 (浏览器) 之间的副本。缓存会根据进来的请求保存输出内容的副本；当下一个请求来到的时候，如果是相同的URL，缓存会根据缓存机制决定是直接使用副本响应访问请求，还是向源服务器再次发送请求。比较常见的就是浏览器会缓存访问过网站的网页，当再次访问这个URL地址的时候，如果网页没有更新，就不会再次下载网页，而是直接使用本地缓存的网页。只有当网站明确标识资源已经更新，浏览器才会再次下载网页

3. web缓存的种类？

客户端缓存

浏览器缓存

服务端缓存

Nginx / Redis / Memcached等

4. 为什么使用？

是为了节约网络的资源加速浏览，浏览器在用户磁盘上对最近请求过的文档进行存储，当访问者再次请求这个页面时，浏览器就可以从本地磁盘显示文档，这样就可以加速页面的浏览。

成本最低的一种缓存实现

减少网络带宽消耗

降低服务器压力

减少网络延迟，加快页面打开速度

5. 浏览器缓存的执行流程

HTTP协议中和页面缓存相关的字段，我们先来认识下：

header	说明
Expires	缓存过期的日期和时间
Cache-Control	设置和缓存相关的配置信息
Last-Modified	请求资源最后修改时间
ETag	请求变量的实体标签的当前值，比如文件的MD5值

ETag: "62475a0a-14c0d"

Last-Modified: Fri, 01 Apr 2022 20:01:14 GMT

(1) 用户首次通过浏览器发送请求到服务端获取数据，客户端是没有对应的缓存，所以需要发送request请求来获取数据；

(2) 服务端接收到请求后，获取服务端的数据及服务端缓存的允许后，返回200的成功状态码并且在响应头上附上对应资源以及缓存信息；

(3) 当用户再次访问相同资源的时候，客户端会在浏览器的缓存目录中查找是否存在响应的缓存文件

(4) 如果没有找到对应的缓存文件，则走(2)步

(5) 如果有缓存文件，接下来对缓存文件是否过期进行判断，过期的判断标准是(Expires)，

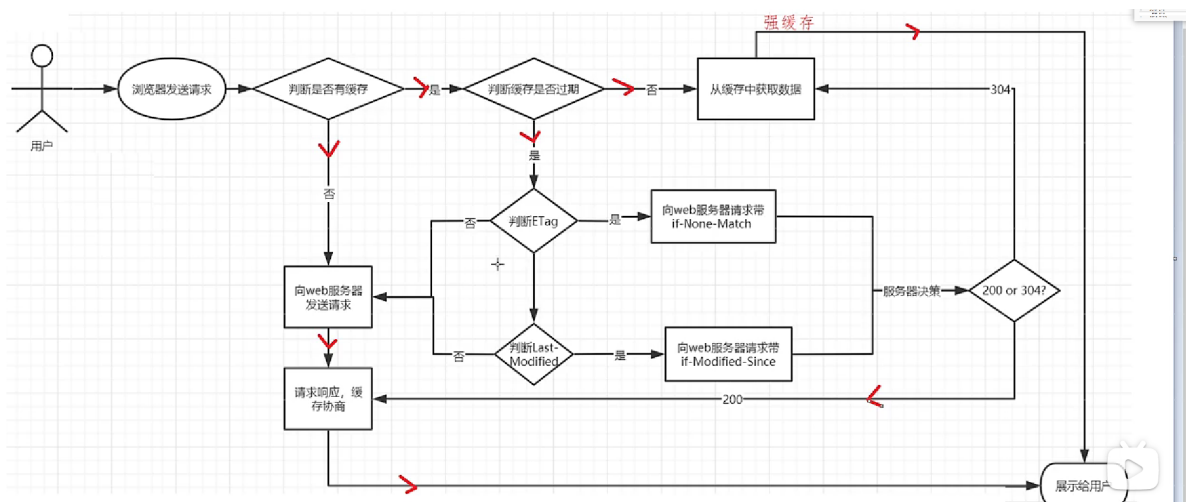
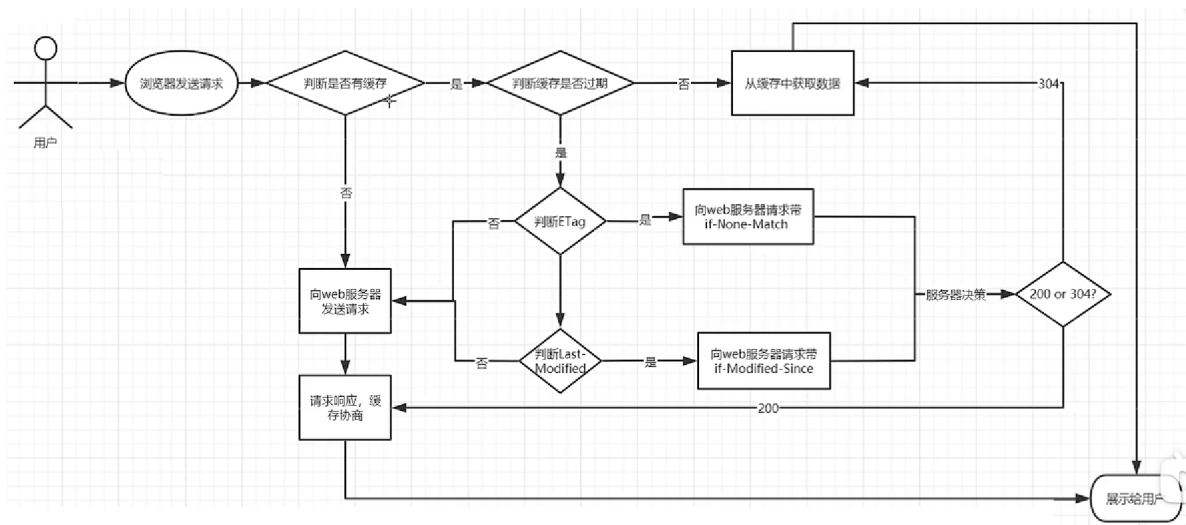
(6) 如果没有过期，则直接从本地缓存中返回数据进行展示

(7) 如果Expires过期，接下来需要判断缓存文件是否发生过变化

(8) 判断的标准有两个，一个是ETag(Entity Tag),一个是Last-Modified

(9) 判断结果是未发生变化，则服务端返回304，直接从缓存文件中获取数据

(10) 如果判断是发生了变化，重新从服务端获取数据，并根据缓存协商(服务端所设置的是否需要缓存数据的设置)来进行数据缓存。



11.1 expires

expires:该指令用来控制页面缓存的作用。可以通过该指令控制HTTP应答中的“Expires”和“Cache-Control”[这两个是响应头信息]

语法	<code>expires [modified] time</code> <code>expires epoch max off;</code>
默认值	<code>expires off;</code>
位置	http、server、location

time:可以整数也可以是负数，指定过期时间，如果是负数，Cache-Control则为no-cache，如果为整数或0，则Cache-Control的值为max-age=time;

epoch: 指定Expires的值为'1 January,1970,00:00:01 GMT'(1970-01-01 00:00:00), Cache-Control的值no-cache

max:指定Expires的值为'31 December2037 23:59:59GMT' (2037-12-31 23:59:59), Cache-Control的值为10年

off:默认不缓存。

```
location ~ .*\. (jpg|webp|png|mp4|html|js)$ {  
    # 设置缓存 10d 代表10天 默认以秒为单位  
    expires 10d;  
    root    html;  
    index  index.html;  
}
```

设置前

▼ **Response Headers** [View source](#)

Connection: keep-alive
Date: Sat, 02 Apr 2022 13:12:09 GMT
ETag: "62475a0a-14c0d"
Last-Modified: Fri, 01 Apr 2022 20:01:14 GMT
Server: nginx/1.21.6
Vary: Accept-Encoding

设置后

▼ **响应标头** [查看源代码](#)

Cache-Control: max-age=864000
Connection: keep-alive
Date: Sat, 02 Apr 2022 13:26:38 GMT
ETag: "62475a0a-14c0d"
Expires: Tue, 12 Apr 2022 13:26:38 GMT
Last-Modified: Fri, 01 Apr 2022 20:01:14 GMT
Server: nginx/1.21.6
Vary: Accept-Encoding

max-age 默认单位为秒 缓存时间

11.2 add_header

add_header指令是用来添加指定的响应头和响应值。

语法	add_header name value [always];
默认值	—
位置	http、server、location...

Cache-Control作为响应头信息，可以设置如下值：

缓存响应指令：

```
Cache-control: must-revalidate
Cache-control: no-cache
Cache-control: no-store
Cache-control: no-transform
Cache-control: public
Cache-control: private
Cache-control: proxy-revalidate
Cache-Control: max-age=<seconds>
Cache-control: s-maxage=<seconds>
```

指令	说明
must-revalidate	可缓存但必须再向源服务器进行确认
no-cache	缓存前必须确认其有效性
no-store	不缓存请求或响应的任何内容
no-transform	代理不可更改媒体类型
public	可向任意方提供响应的缓存
private	仅向特定用户返回响应
proxy-revalidate	要求中间缓存服务器对缓存的响应有效性再进行确认
max-age=<秒>	响应最大Age值
s-maxage=<秒>	公共缓存服务器响应的最大Age值

max-age=[秒]:

```
location ~.*\.(html|js|css|png)$ {
    expires max;
    add_header Cache-Control no_store;
}
```

12 nginx的跨域问题处理

12.1 同源策略

浏览器的同源策略：是一种约定，是浏览器最核心也是最基本的安全功能，如果浏览器少了同源策略，则浏览器的正常功能可能都会受到影响。

同源：**协议、域名(IP)、端口相同即为同源**

```
http://192.168.200.131/user/1  
https://192.168.200.131/user/1  
不
```

```
http://192.168.200.131/user/1  
http://192.168.200.132/user/1  
不
```

```
http://192.168.200.131/user/1  
http://192.168.200.131:8080/user/1  
不
```

```
http://www.nginx.com/user/1  
http://www.nginx.org/user/1  
不
```

```
http://192.168.200.131/user/1  
http://192.168.200.131:8080/user/1  
不
```

```
http://www.nginx.org:80/user/1  
http://www.nginx.org/user/1  
满足
```

跨域问题

简单描述下：

有两台服务器分别为A, B, 如果从服务器A的页面发送异步请求到服务器B获取数据, 如果服务器A和服务器B不满足同源策略, 则就会出现跨域问题。

12.2 解决方案

使用add_header指令，该指令可以用来添加一些头信息

语法	add_header name value ...
默认值	—
位置	http、server、location

此处用来解决跨域问题，需要添加两个头信息，一个是 Access-Control-Allow-Origin，Access-Control-Allow-Methods

Access-Control-Allow-Origin: 直译过来是允许跨域访问的源地址信息，可以配置多个（多个用逗号分隔），也可以使用 * 代表所有源

Access-Control-Allow-Methods: 直译过来是允许跨域访问的请求方式，值可以为 GET POST PUT DELETE...，可以全部设置，也可以根据需要设置，多个用逗号分隔

具体配置方式

```
location /getUser{
    add_header Access-Control-Allow-Origin *; # 允许所有的服务器跨域
    add_header Access-Control-Allow-Methods GET,POST,PUT,DELETE;
    default_type application/json;
    return 200 '{"id":1,"name":"TOM","age":18}';
}
```

13 nginx防盗链

资源盗链指的是此内容不在自己服务器上，而是通过技术手段，绕过别人的限制将别人的内容放到自己页面上最终展示给用户。以此来盗取大网站的空间和流量。简而言之就是用别人的东西成就自己的网站。

了解防盗链的原理之前，我们得先学习一个HTTP的头信息Referer，当浏览器向web服务器发送请求的时候，一般都会带上Referer，来告诉浏览器该网页是从哪个页面链接过来的。

```
Cache-Control: no-cache
Connection: keep-alive
Host: www.wjl1128.top
Pragma: no-cache
Referer: http://localhost:52330/
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.3
```

后台服务器可以根据获取到的这个Referer信息来判断是否为自己信任的网站地址，如果是则放行继续访问，如果不是则可以返回403（服务端拒绝访问）的状态信息。

13.1 valid_referers

`valid_referers`:nginx会通就过查看referer自动和`valid_referers`后面的内容进行匹配, 如果匹配到了就将`$invalid_referer`变量置0, 如果没有匹配到, 则将`$invalid_referer`变量置为1, 匹配的过程中不区分大小写。

语法	<code>valid_referers none blocked server_names string...</code>
默认值	-
位置	server、location

none: 如果Header中的Referer为空, 允许访问 【也就是直接地址栏访问】

blocked:在Header中的Referer不为空, 但是该值被防火墙或代理进行伪装过, 如不带"http://" 、 "https://"等协议头的资源允许访问。

server_names:指定具体的域名或者IP

string: 可以支持正则表达式和*的字符串。如果是正则表达式, 需要以 `~` 开头表示, 例如

注意: 在 `nginx` 中, `if` 后面加个空格 再写小括号判断条件

```
location ~*\. (png|jpg|gif){
    valid_referers none blocked www.baidu.com
    192.168.200.222 *.example.com example.* www.example.org
    ~\.google\.;
    if ($invalid_referer){
        return 403;
    }
    root /usr/local/nginx/html;
}

location ~ .*\. (jpg|webp|png|mp4|html|js|pdf)$ {
    # 设置缓存 10d 代表10天 默认以秒为单位 [max]代表10年
    add_header file jpg|webp|png|mp4|html|js|pdf;
    expires max;
    # none 为空 string正则表达式 没有匹配到 $invalid_referer 为1 否则0
    valid_referers none string ~localhost:52330;
    if ($invalid_referer){
        return 403;
    }
    root html;
    index index.html;
}
```

遇到的问题: 图片有很多, 该如何批量进行防盗链?

针对目录进行防盗链

配置如下:

```
location /images {
    valid_referers none blocked www.baidu.com
192.168.200.222 *.example.com example.* www.example.org
    ~\.google\.;
    if ($invalid_referer){
        return 403;
    }
    root /usr/local/nginx/html;
}
```

这样我们可以对一个目录下的所有资源进行翻到了操作。

遇到的问题: Referer的限制比较粗, 比如随意加一个Referer, 上面的方式是无法进行限制的。那么这个问题改如何解决?

此处我们需要用到Nginx的第三方模块 `ngx_http_accesskey_module`, 第三方模块如何实现盗链, 如果在Nginx中使用第三方模块的功能, 这些我们在后面的Nginx的模块篇再进行详细的讲解。

转发

14 Rewrite指令使用

Rewrite是Nginx服务器提供的一个重要基本功能, 是Web服务器产品中几乎必备的功能。主要的作用是用来实现URL的重写。

注意: Nginx服务器的Rewrite功能的实现依赖于PCRE的支持, 因此在编译安装Nginx服务器之前, 需要安装PCRE库。Nginx使用的是`ngx_http_rewrite_module`模块来解析和处理Rewrite功能的相关配置。

"地址重写"与"地址转发"

重写和转发的区别：

地址重写浏览器地址会发生变化而地址转发则不变
一次地址重写会产生两次请求而一次地址转发只会产生一次请求
地址重写到的页面必须是一个完整的路径而地址转发则不需要
地址重写因为是两次请求所以request范围内属性不能传递给新页面而地址转发因为是一次请求所以可以传递值
地址转发速度快于地址重写

14.1 set指令


该指令用来设置一个新的变量。

语法	<code>set \$variable value;</code>
默认值	—
位置	server、location、if

variable:变量的名称, 该变量名称要用"\$"作为变量的第一个字符, 且不能与Nginx服务器预设的全局变量同名。

value:变量的值, 可以是字符串、其他变量或者变量的组合等。

```
location =/set {  
    set $name wjl;  
    set $age 18;  
    default_type text/plain;  
    return 200 $name===age;  
}
```



← → ↻ 🔒 🔓 www.wjl1128.top:8000/set

wjl===18

14.2 nginx内置全局变量

可以用于日志等

变量	说明
\$args	变量中存放了请求URL中的请求指令。比如 http://192.168.200.133:8080?arg1=value1&args2=value2 中的"arg1=value1&arg2=value2", 功能和\$query_string一样
\$http_user_agent	变量存储的是用户访问服务的代理信息(如果通过浏览器访问, 记录的是浏览器的相关版本信息)
\$host	变量存储的是访问服务器的server_name值
\$document_uri	变量存储的是当前访问地址的URI。比如 http://192.168.200.133/server?id=10&name=zhangsan 中的"/server", 功能和\$uri一样
\$document_root	变量存储的是当前请求对应location的root值, 如果未设置, 默认指向Nginx自带html目录所在位置
\$content_length	变量存储的是请求头中的Content-Length的值
\$content_type	变量存储的是请求头中的Content-Type的值
\$http_cookie	变量存储的是客户端的cookie信息, 可以通过add_header Set-Cookie 'cookieName=cookieValue'来添加cookie数据
\$limit_rate	变量中存储的是Nginx服务器对网络连接速率的限制, 也就是Nginx配置中对limit_rate指令设置的值, 默认是0, 不限制。
\$remote_addr	变量中存储的是客户端的IP地址
\$remote_port	变量中存储了客户端与服务端建立连接的端口号
\$remote_user	变量中存储了客户端的用户名, 需要有认证模块才能获取
\$scheme	变量中存储了访问协议
\$server_addr	变量中存储了服务端的地址
\$server_name	变量中存储了客户端请求到达的服务器的名称
\$server_port	变量中存储了客户端请求到达服务器的端口号
\$server_protocol	变量中存储了客户端请求协议的版本, 比如"HTTP/1.1"
\$request_body_file	变量中存储了发给后端服务器的本地文件资源的名称
\$request_method	变量中存储了客户端的请求方式, 比如"GET", "POST"等
\$request_filename	变量中存储了当前请求的资源文件的路径名
\$request_uri	变量中存储了当前请求的URI, 并且携带请求参数, 比如 http://192.168.200.133/server?id=10&name=zhangsan 中的"/server?id=10&name=zhangsan"

14.3 if

该指令用来支持条件判断，并根据条件判断结果选择不同的Nginx配置。

语法	if (condition){...}
默认值	-
位置	server、location

condition为判定条件，可以支持以下写法：

1. 变量名。如果变量名对应的值为**空或者是0**，**if都判断为false**，其他条件为true。

```
if ($param){  
  
}
```

2. 使用"="和"!="比较变量和字符串是否相等，满足条件为true，不满足为false

```
if ($request_method = POST){  
    return 405;  
}
```

注意：此处和Java不太一样的地方是**字符串不需要添加引号**。

3. 使用正则表达式对变量进行匹配，匹配成功返回true，否则返回false。变量与正则表达式之间使用"~"，"~*"，"!~"，"!~*"来连接。

"~"代表匹配正则表达式过程中区分大小写，

"~*"代表匹配正则表达式过程中不区分大小写

"!~"和"!~*"刚好和上面取相反值，如果匹配上返回false，匹配不上返回true

```
if ($http_user_agent ~ MSIE){  
    # $http_user_agent的值中是否包含MSIE字符串，如果包含返回true  
}
```

注意：**正则表达式字符串一般不需要加引号，但是如果字符串中包含"}"或者是";"等字符时，就需要把引号加上。**

4. 判断请求的文件是否存在使用"-f"和"!-f"，

当使用"-f"时，如果请求的文件**存在返回true，不存在返回false**。

当使用"!f"时，如果请求文件不存在，但该文件所在目录存在返回true，文件和目录都不存在返回false，如果文件存在返回false

```
if (-f $request_filename){
```

```

        #判断请求的文件是否存在
    }
    if (!-f $request_filename){
        #判断请求的文件是否不存在
    }

    location /{
        root html;
        default_type text/plain;
        if (!-f $request_filename){
            return 200 '您访问的文件不存在';
        }
    }

    location =/if {
        default_type text/plain;
        if ($request_method = GET){
            return 200 success;
        }
        if ($request_method = PUT){
            return 405;
        }
        return 404 error;
    }

```

5. 判断请求的目录是否存在使用"-d"和"!-d",

当使用"-d"时, 如果请求的目录存在, if返回true, 如果目录不存在则返回false

当使用"!-d"时, 如果请求的目录不存在但该目录的上级目录存在则返回true, 该目录和它上级目录都不存在则返回false, 如果请求目录存在也返回false.

6. 判断请求的目录或者文件是否存在使用"-e"和"!-e"

当使用"-e", 如果请求的目录或者文件存在时, if返回true, 否则返回false.

当使用"!-e", 如果请求的文件和文件所在路径上的目录都不存在返回true, 否则返回false

7. 判断请求的文件是否可执行使用"-x"和"!-x"

当使用"-x", 如果请求的文件可执行, if返回true, 否则返回false

当使用"!-x", 如果请求文件不可执行, 返回true, 否则返回false

14.4 break

该指令用于中断当前相同作用域中的其他Nginx配置。与该指令处于同一作用域的Nginx配置中, 位于它前面的指令配置生效, 位于后面的指令配置无效。

语法	break;
默认值	—
位置	server、location、if

例子：

```
location /{
    if ($param){
        set $id $1;
        break;
        limit_rate 10k;
    }
}
```

也就是当前location块直接结束

14.5 return

该指令用于完成对请求的处理，直接向客户端返回响应状态代码。在return后的所有Nginx配置都是无效的。

语法	return code [text]; return code URL; return URL;
默认值	—
位置	server、location、if

code:为返回给客户端的HTTP状态代理。可以返回的状态代码为0~999的任意HTTP状态代理

text:为返回给客户端的响应体内容，支持变量的使用

URL:为返回给客户端的URL地址

```
location =/getUrl {
    # 302重定向状态码
    return 302 http://www.baidu.com;
}
```


14.6 rewrite

指令通过正则表达式的使用来改变URI。可以同时存在一个或者多个指令，按照顺序依次对URL进行匹配和处理。

URL和URI的区别：

URI:统一资源标识符

URL:统一资源定位符

语法	rewrite regex replacement [flag];
默认值	—
位置	server、location、if

regex:用来匹配URI的正则表达式

replacement:匹配成功后，用于替换URI中被截取内容的字符串。如果该字符串是以"http://"或者"https://"开头的，则不会继续向下对URI进行其他处理，而是直接返回重写后的URI给客户端。

flag:用来设置rewrite对URI的处理行为，可选值有如下：

- last: # 找location块
- break
- redirect # 重定向状态码302
- permanent # 重定向状态码301

```
location /rewrite {
    # 意思是: 如果以 rewirte 开头包含test就跳转到testif
    rewrite ^/rewrite/(test)\w*$ /testif last;
}
```

```
location /rewrite {
    # rewrite 正则表达式规则      重写的路径  flag[可不写]
    rewrite ^/rewrite/url\w*$ http://www.baidu.com;
    # 括号表达式  $1 == test
    rewrite ^/rewrite/(test)\w*$ /$1 last;
    rewrite ^/rewrite/(demo)\w*$ /$1 redirect;
}
```

```
location =/test {
    default_type text/plain;
    return 200 'test重写成功';
}
```

```
location =/demo {
    default_type text/plain;
```

```

        return 200 'demo重写成功';
    }

    server {
        listen 8001;
        server_name location www.wjl1128.top;
        # 访问此端口时 可以直接定向 8000    8001/if=8000/if
        # ^.* 表示任意多层
        rewrite ^(.*) http://www.wjl1128.top:8000/$1 redirect;
    }

```

14.7 rewrite_log指令

该指令配置是否开启URL重写日志的输出功能。

语法	rewrite_log on off;
默认值	rewrite_log off;
位置	http、server、location、if

开启后，URL重写的相关日志将以notice级别输出到error_log指令配置的日志文件汇总。

```

location /rewrite {
    # 开启
    rewrite_log on;
    error_log logs/rewrite.log notice;
    # rewrite 正则表达式规则    重写的路径    flag[可不写]
    rewrite ^/rewrite/url\w*$ http://www.baidu.com;
    # 括号表达式    $1 = test
    rewrite ^/rewrite/(test)\w*$ /$1 last;
    rewrite ^/rewrite/(demo)\w*$ /$1 redirect;
}

```

14.10 案例

域名跳转

》问题分析

先来看一个效果，如果我们想访问京东网站，大家都知道我们可以输入 `www.jd.com`，但是同样的我们也可以输入 `www.360buy.com` 同样也都能访问到京东网站。这个其实是因为京东刚开始的时候域名就是 `www.360buy.com`，后面由于各种原因把自己的域名换成了 `www.jd.com`，虽然说域名变量，但是对于以前只记住了 `www.360buy.com` 的用户来说，我们如何把这部分用

户也迁移到我们新域名的访问上来，针对于这个问题，我们就可以使用Nginx中Rewrite的域名跳转来解决。

》环境准备

- 准备两个域名 www.360buy.com | www.jd.com

```
vim /etc/hosts
```

```
192.168.200.133 www.360buy.com
192.168.200.133 www.jd.com
```

- 在/usr/local/nginx/html/hm目录下创建一个访问页面

```
<html>
  <title></title>
  <body>
    <h1>欢迎来到我们的网站</h1>
  </body>
</html>
```

- 通过Nginx实现当访问www.360buy.com访问到系统的首页

```
server {
    listen 80;
    server_name www.hm.com;
    location /{
        root /usr/local/nginx/html/hm;
        index index.html;
    }
}
```

》通过Rewrite完成将www.360buy.com的请求跳转到www.jd.com

```
server {
    listen 80;
    server_name www.360buy.com;
    rewrite ^/ http://www.jd.com permanent;
}
```

问题描述:如何在域名跳转的过程中携带请求的URI?

修改配置信息

```
server {
    listen 80;
    server_name www.itheima.com;
    rewrite ^(.*) http://www.hm.com$1 permanent;
}
```

问题描述:我们除了上述说的www.jd.com 、 www.360buy.com其实还有我们也可以通过www.jingdong.com来访问, 那么如何通过Rewrite来实现多个域名的跳转?

添加域名

```
vim /etc/hosts
192.168.200.133 www.jingdong.com
```

修改配置信息

```
server{
    listen 80;
    server_name www.360buy.com www.jingdong.com;
    rewrite ^(.*) http://www.jd.com$1 permanent;
}
```

域名镜像

上述案例中, 将www.360buy.com 和 www.jingdong.com都能跳转到www.jd.com, 那么www.jd.com我们就可以把它起名叫主域名, 其他两个就是我们所说的镜像域名, 当然如果我们不想把整个网站做镜像, 只想为其中某一个子目录下的资源做镜像, 我们可以在location块中配置rewrite功能, 比如:

```
server {
    listen 80;
    server_name rewrite.myweb.com;
    location ^~ /source1{
        rewrite ^/resource1(.*) http://rewrite.myweb.com/web$1
    last;
    }
    location ^~ /source2{
        rewrite ^/resource2(.*) http://rewrite.myweb.com/web$1
    last;
    }
}
```

独立域名

一个完整的项目包含多个模块，比如购物网站有商品商品搜索模块、商品详情模块已经购物车模块等，那么我们如何为每一个模块设置独立的域名。

需求：

```
http://search.hm.com  访问商品搜索模块
http://item.hm.com    访问商品详情模块
http://cart.hm.com    访问商品购物车模块
```

```
server{
    listen 80;
    server_name search.hm.com;
    rewrite ^(.*) http://www.hm.com/bbs$1 last;
}
server{
    listen 81;
    server_name item.hm.com;
    rewrite ^(.*) http://www.hm.com/item$1 last;
}
server{
    listen 82;
    server_name cart.hm.com;
    rewrite ^(.*) http://www.hm.com/cart$1 last;
}
```

目录自动添加"/"

问题描述

通过一个例子来演示下问题：

```
server {
    listen 80;
    server_name localhost;
    location / {
        root html;
        index index.html;
    }
}
```

要想访问上述资源，很简单，只需要通过<http://192.168.200.133>直接就能访问，地址后面不需要加/，但是如果将上述的配置修改为如下内容：

```
server {
    listen 80;
    server_name localhost;
    location /hm {
        root html;
        index index.html;
    }
}
```

这个时候,要想访问上述资源,按照上述的访问方式,我们可以通过<http://192.168.200.133/hm/>来访问,但是如果地址后面不加斜杠,页面就会出问题。如果不加斜杠, Nginx服务器内部会自动做一个301的重定向,重定向的地址会有一个指令叫server_name_in_redirect on|off;来决定重定向的地址:

如果该指令为on

重定向的地址为: http://server_name/目录名/;

如果该指令为off

重定向的地址为: <http://原URL中的域名/目录名/>;

所以就拿刚才的地址来说, <http://192.168.200.133/hm>如果不加斜杠,那么按照上述规则,如果指令server_name_in_redirect为on,则301重定向地址变为 <http://localhost/hm/>,如果为off,则301重定向地址变为<http://192.168.200.133/ht/>。后面这个是正常的,前面地址就有问题。

注意server_name_in_redirect指令在Nginx的0.8.48版本之前默认都是on,之后改成了off,所以现在这个版本不需要考虑这个问题,但是如果是0.8.48以前的版本并且server_name_in_redirect设置为on,我们如何通过rewrite来解决这个问题?

解决方案

我们可以使用rewrite功能为末尾没有斜杠的URL自动添加一个斜杠

```
server {
    listen 80;
    server_name localhost;
    server_name_in_redirect on;

    location /hm {
        if (-d $request_filename){
            rewrite ^/(.*)((^/))$ http://$host:$server_port$1$2/
            permanent;
        }
    }
}
```

```
server {
    listen      8082;
    server_name localhost;
    server_name_in_redirect on;
    location /heima {
        root html;
        index index.html;
        if (-d $request_filename){
            rewrite ^(.*)([^\/]$) http://$host:$server_port$1$2/ permanent;
        }
    }
}
```

合并目录

搜索引擎优化(SEO)是一种利用搜索引擎的搜索规则来提供目的网站的有关搜索引擎内排名的方式。我们在创建自己的站点时,可以通过很多中方式来有效的提供搜索引擎优化的程度。其中有一项就包含URL的目录层级一般不要超过三层, 否则的话不利于搜索引擎的搜索也给客户端的输入带来了负担, 但是将所有的文件放在一个目录下又会导致文件资源管理混乱并且访问文件的速度也会随着文件增多而慢下来, 这两个问题是相互矛盾的, 那么使用rewrite如何解决上述问题?

举例, 网站中有一个资源文件的访问路径时 /server/11/22/33/44/20.html, 也就是说20.html存在于第5级目录下, 如果想要访问该资源文件, 客户端的URL地址就要写成

`http://www.web.name/server/11/22/33/44/20.html` ,

```
server {
    listen 80;
    server_name www.web.name;
    location /server{
        root html;
    }
}
```

但是这个是非常不利于SEO搜索引擎优化的, 同时客户端也不好记.使用rewrite我们可以进行如下配置:

```
server {
    listen 80;
    server_name www.web.name;
    location /server{
        rewrite ^/server-([0-9]+)-([0-9]+)-([0-9]+)-([0-9]+)\.html$ /server/$1/$2/$3/$4/$5.html last;
    }
}
```

这样的花, 客户端只需要输入<http://www.web.name/server-11-22-33-44-20.html>就可以访问到20.html页面了。这里也充分利用了rewrite指令支持正则表达式的特性。

防盗链

防盗链之前我们已经介绍过了相关的知识，在rewrite中的防盗链和之前将的原理其实都是一样的，只不过通过rewrite可以将防盗链的功能进行完善下，当出现防盗链的情况，我们可以使用rewrite将请求转发到自定义的一张照片和页面，给用户比较好的提示信息。下面我们就通过根据文件类型实现防盗链的一个配置实例：

```
server{
    listen 80;
    server_name www.web.com;
    location ~* ^.*\.(gif|jpg|png|swf|flv|rar|zip)$ {
        valid_referers none blocked server_names *.web.com;
        if ($invalid_referer){
            rewrite ^/ http://www.web.com/images/forbidden.png;
        }
    }
}
```

根据目录实现防盗链配置：

```
server{
    listen 80;
    server_name www.web.com;
    location /file/{
        root /server/file/;
        valid_referers none blocked server_names *.web.com;
        if ($invalid_referer){
            rewrite ^/ http://www.web.com/images/forbidden.png;
        }
    }
}
```

反向代理

15 反向代理

正向代理代理的对象是客户端，反向代理代理的是服务端，这是两者之间最大的区别。

Nginx即可以实现正向代理，也可以实现反向代理。

Nginx反向代理模块的指令是由 `ngx_http_proxy_module` 模块进行解析，该模块在安装Nginx的时候已经自己加装到Nginx中了，接下来我们把反向代理中的常用指令

```
proxy_pass
proxy_set_header
proxy_redirect
```

15.1.1 proxy_pass

该指令用来设置被代理服务器地址，可以是主机名称、IP地址加端口号形式。

语法	<code>proxy_pass URL;</code>
默认值	—
位置	location

URL:为要设置的被代理服务器地址，包含传输协议(`http` , `https://`)、主机名称或IP地址加端口号、URI等要素。

编写`proxy_pass`的时候，后面的值要不要加`"/"`

```
server {
    listen 80;
    server_name localhost;
    location /{
        #proxy_pass http://192.168.200.146;
        proxy_pass http://192.168.200.146/;
    }
}
# 当客户端访问 http://localhost/index.html,效果是一样的
server{
    listen 80;
    server_name localhost;
    location /server{
        #proxy_pass http://192.168.200.146;
        proxy_pass http://192.168.200.146/;
    }
}
```

```
# 当客户端访问 http://localhost/server/index.html
# 这个时候, 第一个proxy_pass就变成了http://localhost/server/index.html
# 第二个proxy_pass就变成了http://localhost/index.html效果就不一样了。
```

15.1.2 proxy_set_header

该指令可以更改Nginx服务器接收到的客户端请求的请求头信息, 然后将新的请求头发送给代理的服务器

语法	proxy_set_header field value;
默认值	proxy_set_header Host \$proxy_host; proxy_set_header Connection close;
位置	http、server、location

需要注意的是, 如果想要看到结果, 必须在被代理的服务器上来获取添加的头信息。

被代理服务器: [192.168.200.146]

```
server {
    listen 8080;
    server_name localhost;
    default_type text/plain;
    # $http_username; 表示从请求头获取username
    return 200 $http_username;
}
```

代理服务器: [192.168.200.133]

```
server {
    listen 8080;
    server_name localhost;
    location /server {
        proxy_pass http://192.168.200.146:8080/;
        proxy_set_header username TOM;
    }
}
```

访问测试

15.1.3 proxy_redirect

该指令是用来重置头信息中的"Location"和"Refresh"的值。

语法	<code>proxy_redirect redirect replacement;</code> <code>proxy_redirect default;</code> <code>proxy_redirect off;</code>
默认值	<code>proxy_redirect default;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code>

》为什么要用该指令？不暴露被代理端的地址

服务端[192.168.10.102]

```
# 让代理发过来的请求找不到时被代理使用重定向
server {
    listen 8081;
    server_name localhost;
    if (!-f $request_filename){
        return 302 http://192.168.10.102;
    }
}
```

▼ 响应标头 查看源代码

```
Connection: keep-alive
Content-Length: 145
Content-Type: text/html
Date: Sun, 03 Apr 2022 15:48:29 GMT
Location: http://192.168.10.102:80/102
Server: nginx/1.21.6
```

同时重定向之后地址栏也会跟着更改

这时候在代理端加上该指令

代理服务端[192.168.56.2]

```
## 被代理端
server {
    listen 80;
    server_name localhost;
    charset utf-8;
```

```

#access_log logs/host.access.log main;

location / {
    root html;
    index index.html index.htm;
    if (!-f $request_filename){
        return 302 http://192.168.10.102/favicon.ico;
    }
}

# 代理端

server {
    listen 8081;
    server_name localhost;
    location / {
        proxy_pass http://192.168.10.102:8081/;
        proxy_redirect http://192.168.200.146 http://192.168.56.2;
    }
}

server {
    listen 8002;
    server_name localhost www.wjl1128.top;
    charset utf-8;

    location /tom {
        proxy_pass http://tomcat/;
    }

    location /102 {
        proxy_set_header username wjlzszs;
        proxy_pass http://192.168.10.102/;
        # 注意 相应的端口号与路径 也要复写 无论是代理端的还是被代理的
:8002/102
        proxy_redirect http://192.168.10.102/
http://192.168.56.2:8002/102;
        # proxy_redirect http://192.168.10.102/
http://www.wjl1128.top:8002/102;
    }
}

```

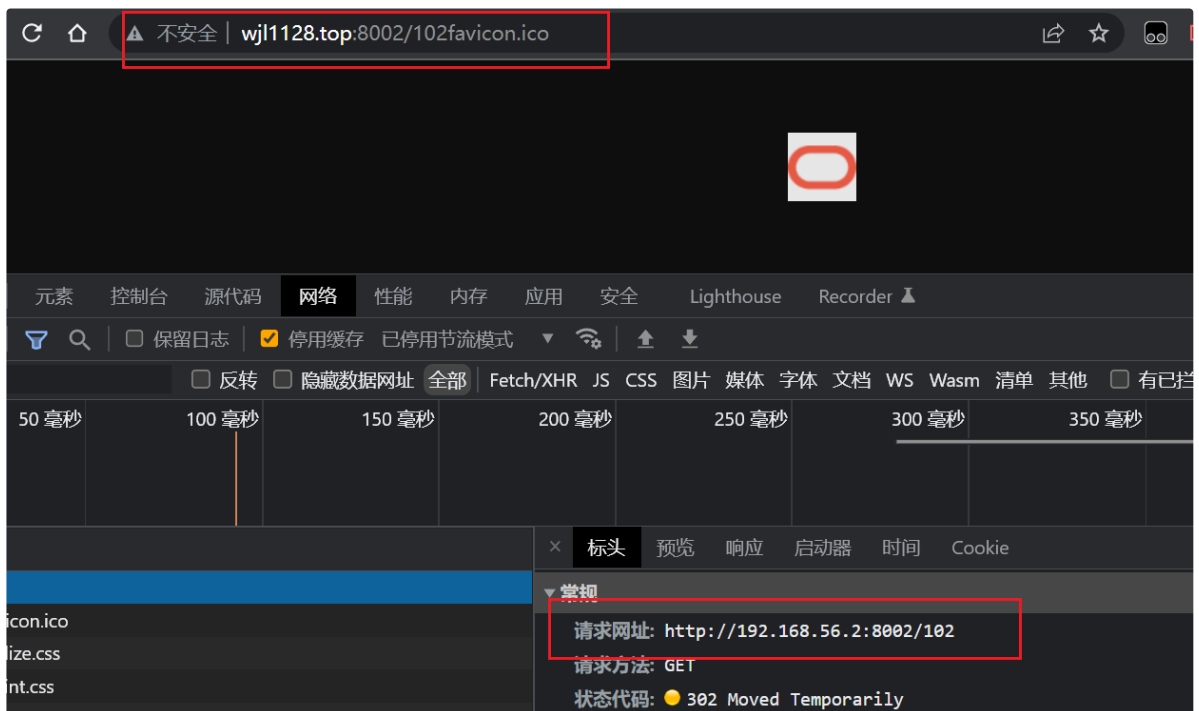
也就是 当被代理的服务器内部发生重定向时 会让地址栏也会更改 暴露相应的地址

设置 proxy_redirect 1 2 之后 如果重定向的地址为 1 时 就会更改为定义的 2

相应的 注意 1 2 的端口号 和访问路径也一并重写

注意 被代理内部重定向 [重定向到外链不会覆写localhost (端口号不同也算)]

相应的地址栏



》该指令的几组选项

```
proxy_redirect redirect replacement;
```

redirect:目标,Location的值
replacement:要替换的值

```
proxy_redirect default;
```

default;
将location块的uri变量作为replacement,
将proxy_pass变量作为redirect进行替换
#

```
proxy_redirect off;
```

关闭proxy_redirect的功能

安全控制

通过代理分开了客户端到应用程序服务器端的连接，实现了安全措施。在反向代理之前设置防火墙，仅留一个入口供代理服务器访问。

16 SSL

http请求转变成https请求，那么这两个之间的区别简单的来说两个都是HTTP协议，只不过https是身披SSL外壳的http。

HTTPS是一种通过计算机网络进行安全通信的传输协议。它经由HTTP进行通信，利用SSL/TLS建立全通信，加密数据包，确保数据的安全性。

SSL(Secure Sockets Layer)安全套接层

TLS(Transport Layer Security)传输层安全

上述这两个是为网络通信提供安全及数据完整性的一种安全协议，TLS和SSL在传输层和应用层对网络连接进行加密。

总结来说为什么要使用https：

http协议是明文传输数据，存在安全问题，而https是加密传输，相当于http+ssl，并且可以防止流量劫持。

Nginx要想使用SSL，需要满足一个条件即需要添加一个模块 `--with-http_ssl_module`，而该模块在编译的过程中又需要OpenSSL的支持

16.1 nginx模块anzhuang

- 》将原有/usr/local/nginx/sbin/nginx进行备份
- 》拷贝nginx之前的配置信息
- 》在nginx的安装源码进行配置指定对应模块 `./configure --with-http_ssl_module`
- 》通过make模板进行编译
- 》将objs下面的nginx移动到/usr/local/nginx/sbin下
- 》在源码目录下执行 `make upgrade`进行升级，这个可以实现不停机添加新模块的功能

```
--prefix=/usr/local/nginx --sbin-path=/usr/local/nginx/sbin/nginx
--modules-path=/usr/local/nginx/modules --conf-
path=/usr/local/nginx/conf/nginx.conf --error-log-
path=/usr/local/nginx/logs/error.log --http-log-
path=/usr/local/nginx/logs/access.log --pid-
path=/usr/local/nginx/logs/nginx.pid --lock-
path=/usr/local/nginx/logs/nginx.lock --with-
http_gzip_static_module --with-http_ssl_module
```

16.2 ssl开启

ssl:该指令用来在指定的服务器开启HTTPS,可以使用 `listen 443 ssl`,后面这种方式更通用些。

语法	ssl on off;
默认值	ssl off;
位置	http、server

```
server{
    # 更推荐这个
    listen 443 ssl;
}
```

ssl_certificate:为当前这个虚拟主机指定一个带有PEM格式证书的证书

语法	ssl_certificate file;
默认值	-
位置	http、server

ssl_certificate_key:该指令用来指定PEM secret key文件的路径

语法	ssl_certificate_key file;
默认值	-
位置	http、server

ssl_session_cache:该指令用来配置用于SSL会话的缓存

语法	<code>ssl_session_cache off none [builtin[:size]] [shared:name:size]</code>
默认值	<code>ssl_session_cache none;</code>
位置	http、server

off:禁用会话缓存，客户端不得重复使用会话

none:禁止使用会话缓存，客户端可以重复使用，但是并没有在缓存中存储会话参数

builtin:内置OpenSSL缓存，仅在一个工作进程中使用。

shared:所有工作进程之间共享缓存，缓存的相关信息用name和size来指定

》`ssl_session_timeout`: 开启SSL会话功能后，设置客户端能够反复使用储存在缓存中的会话参数时间。

语法	<code>ssl_session_timeout time;</code>
默认值	<code>ssl_session_timeout 5m;</code>
位置	http、server

》`ssl_ciphers`:指出允许的密码，密码指定为OpenSSL支持的格式

语法	<code>ssl_ciphers ciphers;</code>
默认值	<code>ssl_ciphers HIGH:!aNULL:!MD5;</code>
位置	http、server

可以使用 `openssl ciphers` 查看openssl支持的格式。

》`ssl_prefer_server_ciphers`: 该指令指定是否服务器密码优先客户端密码

语法	<code>ssl_prefer_server_ciphers on off;</code>
默认值	<code>ssl_prefer_server_ciphers off;</code>
位置	http、server

16.3 生成证书

方式一：使用阿里云/腾讯云等第三方服务进行购买。

方式二:使用openssl生成证书

先要确认当前系统是否有安装openssl


```
openssl version
```

安装下面的命令进行生成

```
mkdir /root/cert
cd /root/cert
openssl genrsa -des3 -out server.key 1024
openssl req -new -key server.key -out server.csr
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
openssl x509 -req -days 365 -in server.csr -signkey server.key -
out server.crt
```

开启SSL实例

```
# openssl
server {
    listen      443 ssl;
    server_name localhost;

    ssl_certificate      server.crt;
    ssl_certificate_key  server.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;
    }
}

# aliyun
server {
    listen 8003 ssl;
    server_name localhost www.wjl1128.top;
    charset utf-8;
    # 指定pem证书
    ssl_certificate      /opt/nginx/7547995_www.wjl1128.top.pem;
    # 指定 key
    ssl_certificate_key  /opt/nginx/7547995_www.wjl1128.top.key;
```

```

error_log /opt/nginx/error.log notice;

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

```



17 反向代理系统调优

反向代理值Buffer和Cache

Buffer翻译过来是"缓冲", Cache翻译过来是"缓存"。

相同点:

两种方式都是用来提供IO吞吐效率, 都是用来提升Nginx代理的性能。

不同点:

缓冲主要用来解决不同设备之间数据传递速度不一致导致的性能低的问题, 缓冲中的数据一旦此次操作完成后, 就可以删除。

缓存主要是备份, 将被代理服务器的数据缓存一份到代理服务器, 这样的话, 客户端再次获取相同数据的时候, 就只需要从代理服务器上获取, 效率较高, 缓存中的数据可以重复使用, 只有满足特定条件才会删除

(1) Proxy Buffer相关指令

》 proxy_buffering :该指令用来开启或者关闭代理服务器的缓冲区;

语法	proxy_buffering on off;
默认值	proxy_buffering on;
位置	http、server、location

》 proxy_buffers:该指令用来指定单个连接从代理服务器读取响应的缓存区的个数和大小。

语法	proxy_buffers number size;
默认值	proxy_buffers 8 4k 8K;(与系统平台有关)
位置	http、server、location

number:缓冲区的个数

size:每个缓冲区的大小, 缓冲区的总大小就是number*size

》 proxy_buffer_size:该指令用来设置从被代理服务器获取的第一部分响应数据的大小。保持与proxy_buffers中的size一致即可, 当然也可以更小。

语法	proxy_buffer_size size;
默认值	proxy_buffer_size 4k 8k;(与系统平台有关)
位置	http、server、location

》 proxy_busy_buffers_size: 该指令用来限制同时处于BUSY状态的缓冲总大小。

语法	proxy_busy_buffers_size size;
默认值	proxy_busy_buffers_size 8k 16K;
位置	http、server、location

》 proxy_temp_path:当缓冲区存满后, 仍未被Nginx服务器完全接受, 响应数据就会被临时存放在磁盘文件上, 该指令设置文件路径

语法	proxy_temp_path path;
默认值	proxy_temp_path proxy_temp;
位置	http、server、location

注意path最多设置三层。

》 proxy_temp_file_write_size: 该指令用来设置磁盘上缓冲文件的大小。

语法	proxy_temp_file_write_size size;
默认值	proxy_temp_file_write_size 8K 16K;
位置	http、server、location

通用网站的配置

```
proxy_buffering on;  
proxy_buffer_size 4 32k;  
proxy_busy_buffers_size 64k;  
proxy_temp_file_write_size 64k;
```

根据项目的具体内容进行相应的调节。

Nginx 负载均衡

17 负载均衡

系统的扩展可以分为纵向扩展和横向扩展。

纵向扩展是从单机的角度出发，通过增加系统的硬件处理能力来提升服务器的处理能力

横向扩展是通过添加机器来满足大型网站服务的处理能力。

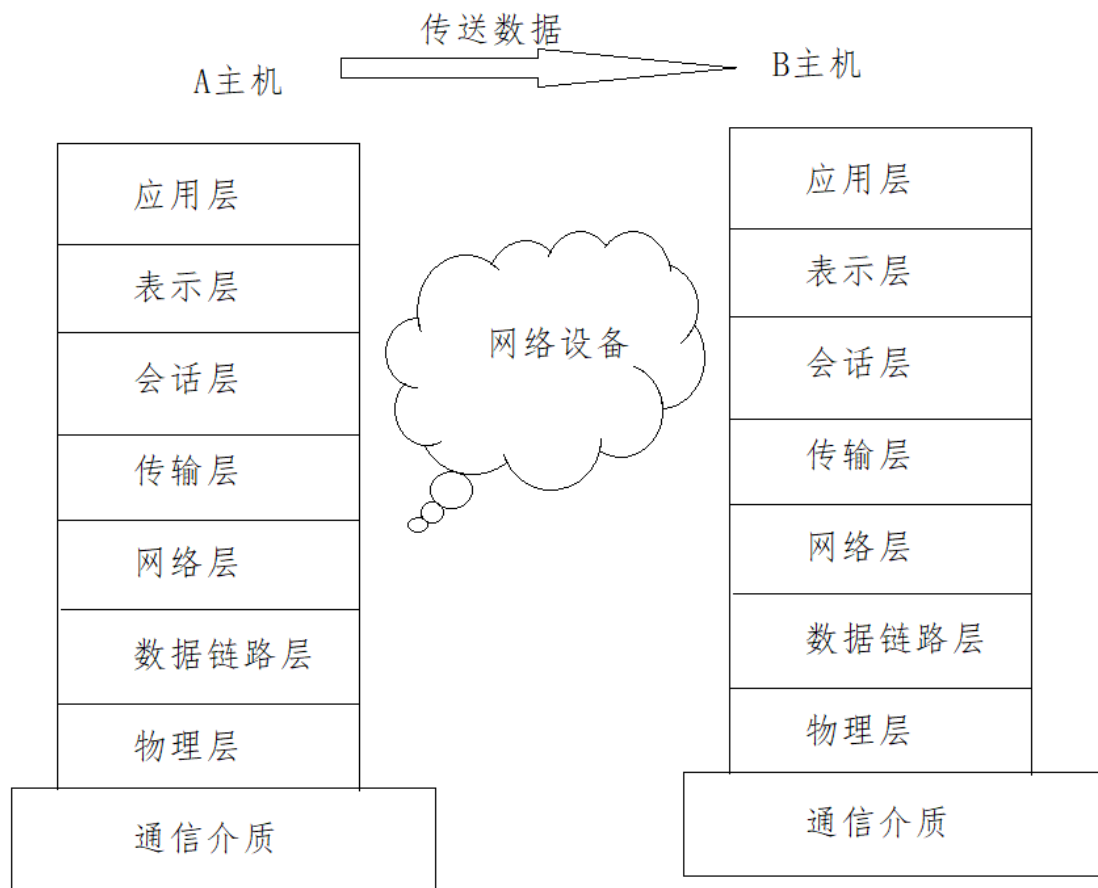
这里面涉及到两个重要的角色分别是"应用集群"和"负载均衡器"。

应用集群：将同一应用部署到多台机器上，组成处理集群，接收负载均衡设备分发的请求，进行处理并返回响应的数据。

负载均衡器：将用户访问的请求根据对应的负载均衡算法，分发到集群中的一台服务器进行处理。

- 1、解决服务器的高并发压力，提高应用程序的处理性能。
- 2、提供故障转移，实现高可用。
- 3、通过添加或减少服务器数量，增强网站的可扩展性。
- 4、在负载均衡器上进行过滤，可以提高系统的安全性。

OSI(open system interconnection),叫开放式系统互联模型，这个是由国际标准化组织ISO指定的一个不基于具体机型、操作系统或公司的网络体系结构。该模型将网络通信的工作分为七层。



应用层：为应用程序提供网络服务。

表示层：对数据进行格式化、编码、加密、压缩等操作。

会话层：建立、维护、管理会话连接。

传输层：建立、维护、管理端到端的连接，常见的有TCP/UDP。

网络层：IP寻址和路由选择

数据链路层：控制网络层与物理层之间的通信。

物理层：比特流传输。

所谓四层负载均衡指的是OSI七层模型中的传输层，主要是基于IP+PORT的负载均衡

实现四层负载均衡的方式：

硬件：F5 BIG-IP、Radware等

软件：LVS、Nginx、Hayproxy等

所谓的七层负载均衡指的是在应用层，主要是基于虚拟的URL或主机IP的负载均衡

实现七层负载均衡的方式：

软件：Nginx、Hayproxy等

四层和七层负载均衡的区别

四层负载均衡数据包是在底层就进行了分发，而七层负载均衡数据包则在最顶端进行分发，所以四层负载均衡的效率比七层负载均衡的要高。
四层负载均衡不识别域名，而七层负载均衡识别域名。

处理四层和七层负载以为其实还有二层、三层负载均衡，二层是在数据链路层基于mac地址来实现负载均衡，三层是在网络层一般采用虚拟IP地址的方式实现负载均衡。

实际环境采用的模式

四层负载(LVS)+七层负载(Nginx)

17.1 nginx 负载均衡

upstream 指令

该指令是用来定义一组服务器，它们可以是监听不同端口的服务器，并且也可以是同时监听TCP和Unix socket的服务器。服务器可以指定不同的权重，默认为1。

语法	upstream name {...}
默认值	—
位置	http

server指令

该指令用来指定后端服务器的名称和一些参数，可以使用域名、IP、端口或者unix socket

语法	server name [parameters]
默认值	—
位置	upstream

服务端设置

```
server {
    listen 9001;
    server_name localhost;
    default_type text/html;
    location /{
        return 200 '<h1>192.168.200.146:9001</h1>';
    }
}
server {
    listen 9002;
```

```

server_name localhost;
default_type text/html;
location /{
    return 200 '<h1>192.168.200.146:9002</h1>';
}
}
server {
    listen 9003;
    server_name localhost;
    default_type text/html;
    location /{
        return 200 '<h1>192.168.200.146:9003</h1>';
    }
}

```

负载均衡器设置

```

upstream backend{
    server 192.168.200.146:9091;
    server 192.168.200.146:9092;
    server 192.168.200.146:9093;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

upstream zook {
    server 192.168.10.102:8080;
    server 192.168.10.102:8000;
    server 192.168.10.102:8001;
}

server {
    listen 8003 ssl;
    server_name localhost www.wjl1128.top;
    charset utf-8;
    # 指定pem证书
    ssl_certificate /opt/nginx/7547995_www.wjl1128.top.pem;
    # 指定 key
    ssl_certificate_key /opt/nginx/7547995_www.wjl1128.top.key;
    error_log /opt/nginx/error.log notice;

    location / {

```

```
        proxy_pass http://zook;
    }
}
```

17.2 负载均衡状态

代理服务器在负责均衡调度中的状态有以下几个：

状态	概述
down	当前的server暂时不参与负载均衡
backup	预留的备份服务器
max_fails	允许请求失败的次数
fail_timeout	经过max_fails失败后，服务暂停时间
max_conns	限制最大的接收连接数

down

down:将该服务器标记为永久不可用，那么该代理服务器将不参与负载均衡。

```
upstream backend{
    server 192.168.200.146:9001 down;
    server 192.168.200.146:9002
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}
```

该状态一般会对需要停机维护的服务器进行设置。

backup

backup:将该服务器标记为备份服务器，当主服务器不可用时，将用来传递请求。


```

upstream backend{
    server 192.168.200.146:9001 down;
    server 192.168.200.146:9002 backup;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

此时需要将9094端口的访问禁止掉来模拟下唯一能对外提供访问的服务宕机以后，backup的备份服务器就要开始对外提供服务，此时为了测试验证，我们需要使用防火墙来进行拦截。

介绍一个工具 `firewall-cmd`，该工具是Linux提供的专门用来操作firewall的。

查询防火墙中指定的端口是否开放

```
firewall-cmd --query-port=9001/tcp
```

如何开放一个指定的端口

```
firewall-cmd --permanent --add-port=9002/tcp
```

批量添加开发端口

```
firewall-cmd --permanent --add-port=9001-9003/tcp
```

如何移除一个指定的端口

```
firewall-cmd --permanent --remove-port=9003/tcp
```

重新加载

```
firewall-cmd --reload
```

其中

--permanent表示设置为持久

--add-port表示添加指定端口

--remove-port表示移除指定端口

max_conns

max_conns=number:用来设置代理服务器同时活动链接的最大数量，默认为0，表示不限制，使用该配置可以根据后端服务器处理请求的并发量来进行设置，防止后端服务器被压垮。

max_fails和fail_timeout

max_fails=number:设置允许请求代理服务器失败的次数，默认为1。

fail_timeout=time:设置经过max_fails失败后，服务暂停的时间，默认是10秒。

```
upstream backend{
    server 192.168.200.133:9001 down;
    server 192.168.200.133:9002 backup;
    # 当出错三次之后 暂停15s
    server 192.168.200.133:9003 max_fails=3 fail_timeout=15;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}
```

17.3 负载均衡策略

Nginx的upstream支持如下六种方式的分配算法，分别是：

算法名称	说明
轮询	默认方式
weight	权重方式
ip_hash	依据ip分配方式
least_conn	依据最少连接方式
url_hash	依据URL分配方式
fair	依据响应时间方式

轮询

是upstream模块负载均衡默认的策略。每个请求会按时间顺序逐个分配到不同的后端服务器。轮询不需要额外的配置。

```

upstream backend{
    # 为1 j
    server 192.168.200.146:9001 weight=1;
    server 192.168.200.146:9002;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

weight加权[加权轮询]

weight=number:用来设置服务器的权重，默认为1，权重数据越大，被分配到请求的几率越大；该权重值，主要是针对实际工作环境中不同的后端服务器硬件配置进行调整的，所有此策略比较适合服务器的硬件配置差别比较大的情况。

```

upstream backend{
    server 192.168.200.146:9001 weight=10;
    server 192.168.200.146:9002 weight=5;
    server 192.168.200.146:9003 weight=3;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

ip_hash

当对后端的多台动态应用服务器做负载均衡时，ip_hash指令能够将某个客户端IP的请求通过哈希算法定位到同一台后端服务器上。这样，当来自某一个IP的用户在后端Web服务器A上登录后，在访问该站点的其他URL，能保证其访问的还是后端web服务器A。

语法	ip_hash;
默认值	—
位置	upstream

```

upstream backend{
    ip_hash;
    server 192.168.200.146:9001;
    server 192.168.200.146:9002;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

需要额外多说一点的是使用ip_hash指令无法保证后端服务器的负载均衡，可能导致有些后端服务器接收到的请求多，有些后端服务器接收的请求少，而且设置后端服务器权重等方法将不起作用。

url_hash

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，要配合缓存命中来使用。同一个资源多次请求，可能会到达不同的服务器上，导致不必要的多次下载，缓存命中率不高，以及一些资源时间的浪费。而使用url_hash，可以使得同一个url（也就是同一个资源请求）会到达同一台服务器，一旦缓存住了资源，再此收到请求，就可以从缓存中读取。

```

upstream backend{
    # &request_uri内置变量 请求地址
    hash &request_uri;
    server 192.168.200.146:9001;
    server 192.168.200.146:9002;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

fair

fair采用的不是内建负载均衡使用的轮换的均衡算法，而是可以根据页面大小、加载时间长短智能的进行负载均衡。那么如何使用第三方模块的fair负载均衡策略。

```

upstream backend{
    fair;
    server 192.168.200.146:9001;
    server 192.168.200.146:9002;
    server 192.168.200.146:9003;
}
server {
    listen 8083;
    server_name localhost;
    location /{
        proxy_pass http://backend;
    }
}

```

但是如何直接使用会报错，因为fair属于第三方模块实现的负载均衡。需要添加 `nginx-upstream-fair`，如何添加对应的模块：

1. 下载nginx-upstream-fair模块

下载地址为：

<https://github.com/gnosek/nginx-upstream-fair>

1. 将下载的文件上传到服务器并进行解压缩

```
unzip nginx-upstream-fair-master.zip
```

1. 重命名资源

```
mv nginx-upstream-fair-master fair
```

1. 使用./configure命令将资源添加到Nginx模块中

```

./configure --add-module= # 自己所解压的模块目录
# 自己的实例
--prefix=/usr/local/nginx --sbin-path=/usr/local/nginx/sbin/nginx
--modules-path=/usr/local/nginx/modules --conf-
path=/usr/local/nginx/conf/nginx.conf --error-log-
path=/usr/local/nginx/logs/error.log --http-log-
path=/usr/local/nginx/logs/access.log --pid-
path=/usr/local/nginx/logs/nginx.pid --lock-
path=/usr/local/nginx/logs/nginx.lock --with-
http_gzip_static_module --with-http_ssl_module --add-
module=/opt/fair

```

1. 编译

```
make
```

18 nginx四层负载均衡

Nginx在1.9之后，增加了一个stream模块，用来实现四层协议的转发、代理、负载均衡等。stream模块的用法跟http的用法类似，允许我们配置一组TCP或者UDP等协议的监听，然后通过proxy_pass来转发我们的请求，通过upstream添加多个后端服务，实现负载均衡。

四层协议负载均衡的实现，一般都会用到LVS、HAProxy、F5等，要么很贵要么配置很麻烦，而Nginx的配置相对来说更简单，更能快速完成工作。

添加stream模块的支持

Nginx默认是没有编译这个模块的，需要使用到stream模块，那么需要在编译的时候加上 `--with-stream` 。

完成添加 `--with-stream` 的实现步骤：

- 》将原有/usr/local/nginx/sbin/nginx进行备份
- 》拷贝nginx之前的配置信息
- 》在nginx的安装源码进行配置指定对应模块 `./configure --with-stream`
- 》通过make模板进行编译
- 》将objs下面的nginx移动到/usr/local/nginx/sbin下
- 》在源码目录下执行 `make upgrade`进行升级，这个可以实现不停机添加新模块的功能

```
--prefix=/usr/local/nginx --sbin-path=/usr/local/nginx/sbin/nginx
--modules-path=/usr/local/nginx/modules --conf-
path=/usr/local/nginx/conf/nginx.conf --error-log-
path=/usr/local/nginx/logs/error.log --http-log-
path=/usr/local/nginx/logs/access.log --pid-
path=/usr/local/nginx/logs/nginx.pid --lock-
path=/usr/local/nginx/logs/nginx.lock --with-
http_gzip_static_module --with-http_ssl_module --with-stream
```

Nginx四层负载均衡的指令

stream指令

该指令提供在其中指定流服务器指令的配置文件上下文。和http指令同级。

语法	stream { ... }
默认值	—
位置	main

upstream指令

该指令和http的upstream指令是类似的。

```
docker run --name redis6370 -p 6370:6379 -d redis:latest --requirepass "1128" --bind 0.0.0.0
```

```
stream {
    upstream redisbackend {
        server 192.168.10.102:6379;
        server 192.168.10.102:6370;
    }

    upstream tomcatbackend {
        server 192.168.10.102:8080;
    }

    server {
        listen 81;
        proxy_pass redisbackend;
    }

    server {
        listen 82;
        proxy_pass tomcatbackend;
    }
}
```

```
S C:\Users\> redis-cli -h 192.168.56.2 -p 81
92.168.56.2:81>
```

家 文档 配置 例子 维基 邮件列表

Apache Tomcat/9.0.60

如果你看到这个，你已经成功安装了 Tomcat。恭喜



推荐阅读:

[安全注意事项操作方法](#)

[管理器应用程序操作方法](#)

[集群/会话复制操作方法](#)

开发者快速入门

[Tomcat 设置](#)

[第一个网络应用程序](#)

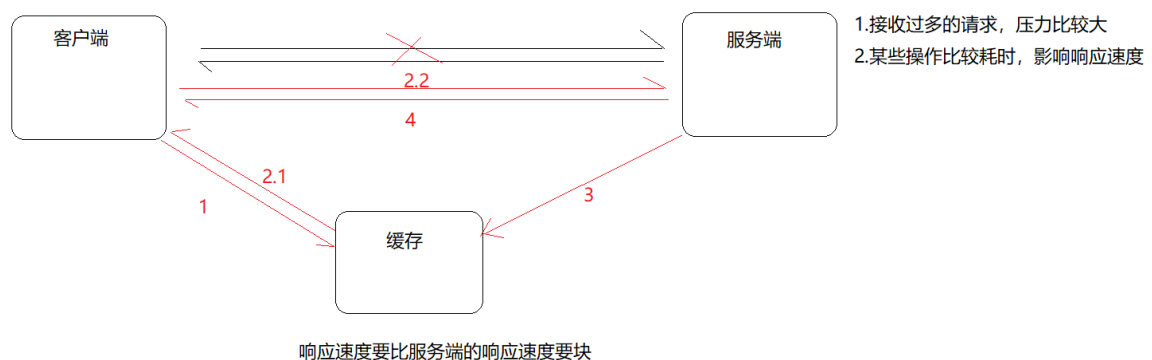
[领域和 AAA](#)

[JDBC 数据源](#)

[例子](#)

Nginx缓存集成

缓存就是数据交换的缓冲区(称作:Cache),当用户要获取数据的时候,会先从缓存中去查询获取数据,如果缓存中有就会直接返回给用户,如果缓存中没有,则会发请求从服务器重新查询数据,将数据返回给用户的同时将数据放入缓存,下次用户就会直接从缓存中获取数据。



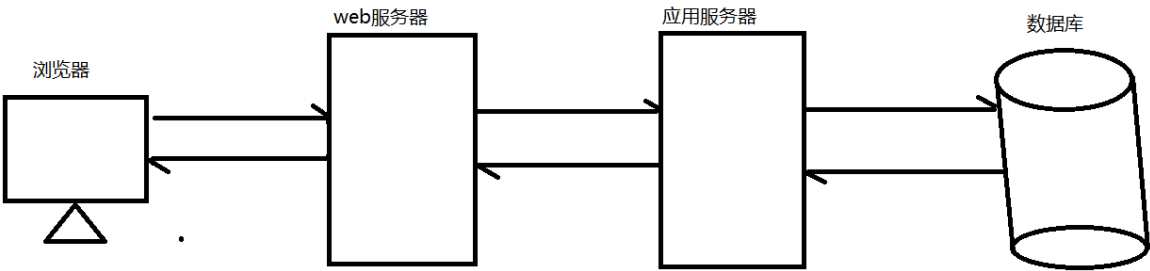
场景	作用
操作系统磁盘缓存	减少磁盘机械操作
数据库缓存	减少文件系统的IO操作
应用程序缓存	减少对数据库的查询
Web服务器缓存	减少对应用服务器请求次数
浏览器缓存	减少与后台的交互次数

缓存的优点

- 1.减少数据传输，节省网络流量，加快响应速度，提升用户体验；
- 2.减轻服务器压力；
- 3.提供服务端的高可用性；

缓存的缺点

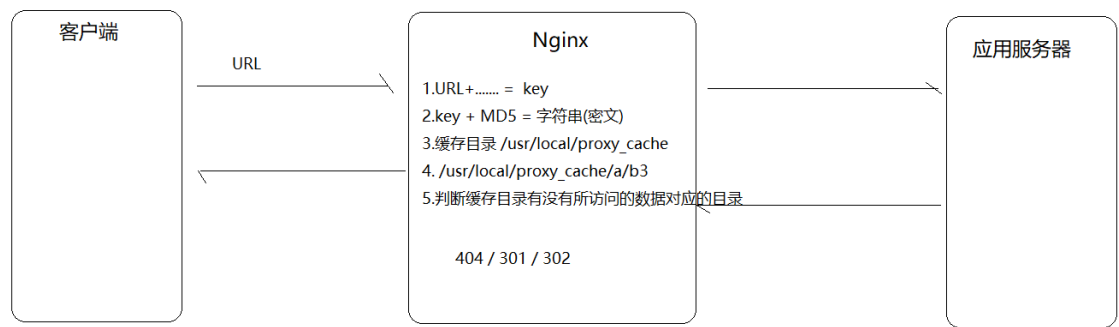
- 1.数据的不一致
- 2.增加成本



Nginx作为web服务器，Nginx作为Web缓存服务器，它介于客户端和应用服务器之间，当用户通过浏览器访问一个URL时，web缓存服务器会去应用服务器获取要展示给用户的内容，将内容缓存到自己的服务器上，当下一次请求到来时，如果访问的是同一个URL，web缓存服务器就会直接将之前缓存的内容返回给客户端，而不是向应用服务器再次发送请求。web缓存降低了应用服务器、数据库的负载，减少了网络延迟，提高了用户访问的响应速度，增强了用户的体验。

19 Nginx的web缓存服务

Nginx是从0.7.48版开始提供缓存功能。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。



Nginx的web缓存服务主要是使用 `ngx_http_proxy_module` 模块相关指令集来完成

19.1 proxy_cache_path

该指定用于设置缓存文件的存放路径

语法	<code>proxy_cache_path path [levels=number] keys_zone=zone_name:zone_size [inactive=time] [max_size=size];</code>
默认值	—
位置	http

path:缓存路径地址,如:

```
/usr/local/proxy_cache
```

levels: 指定该缓存空间对应的目录,最多可以设置3层,每层取值为1|2如 :

```
levels=1:2    缓存空间有两层目录,第一次是1个字母,第二次是2个字母
举例说明:
itheima[key]通过MD5加密以后的值为 43c8233266edce38c2c9af0694e2107d
levels=1:2    最终的存储路径为/usr/local/proxy_cache/d/07
levels=2:1:2  最终的存储路径为/usr/local/proxy_cache/7d/0/21
levels=2:2:2  最终的存储路径为??/usr/local/proxy_cache/7d/10/e2
```

keys_zone:用来为这个缓存区设置名称和指定大小,如:

```
keys_zone=itcast:200m  缓存区的名称是itcast,大小为200M,1M大概能存储8000
个keys
```

inactive:指定缓存的数据多次时间未被访问就将被删除,如:

```
inactive=1d    缓存数据在1天内没有被访问就会被删除
```

max_size:设置最大缓存空间, 如果缓存空间存满, 默认会覆盖缓存时间最长的资源, 如:

```
max_size=20g
```

配置实例:

```
http{
    proxy_cache_path /usr/local/proxy_cache keys_zone=itcast:200m
    levels=1:2:1 inactive=1d max_size=20g;
}
```

19.2 proxy_cache

该指令用来开启或关闭代理缓存, 如果是开启则自定使用哪个缓存区来进行缓存。

语法	proxy_cache zone_name off;
默认值	proxy_cache off;
位置	http、server、location

zone_name: 指定使用缓存区的名称

19.3 proxy_cache_key

该指令用来设置web缓存的key值, Nginx会根据key值MD5哈希存缓存。

语法	proxy_cache_key key;
默认值	proxy_cache_key \$scheme\$proxy_host\$request_uri;
位置	http、server、location

19.4 proxy_cache_valid

该指令用来对不同返回状态码的URL设置不同的缓存时间

语法	proxy_cache_valid [code ...] time;
默认值	—
位置	http、server、location

如:

```
proxy_cache_valid 200 302 10m;
proxy_cache_valid 404 1m;
为200和302的响应URL设置10分钟缓存，为404的响应URL设置1分钟缓存
proxy_cache_valid any 1m;
对所有响应状态码的URL都设置1分钟缓存
```

19.5 proxy_cache_min_uses

该指令用来设置资源被访问多少次后被缓存

语法	proxy_cache_min_uses number;
默认值	proxy_cache_min_uses 1;
位置	http、server、location

19.6 proxy_cache_methods

该指令用户设置缓存哪些HTTP方法

语法	proxy_cache_methods GET HEAD POST;
默认值	proxy_cache_methods GET HEAD;
位置	http、server、location

默认缓存HTTP的GET和HEAD方法，不缓存POST方法。

19.7 配置实例

```
http{
    proxy_cache_path /usr/local/proxy_cache levels=2:1
    keys_zone=wjl08:200m inactive=1d max_size=20g;
    upstream backend{
        server 192.168.200.146:8080;
    }
    server {
        listen      8080;
        server_name localhost;
        location / {
            proxy_cache wjl08;
            # proxy_cache_key wjl_key;
            # 代表访问路径
            proxy_cache_key $scheme$proxy_host$request_uri;
            proxy_cache_min_uses 5;
            proxy_cache_valid 200 5d;
        }
    }
}
```

```
# 注意 响应失败被缓存 即使访问成功 也会显示响应失败的内容 是使用
同一个cache_key的原因
proxy_cache_valid 404 30s;
proxy_cache_valid any 1m;
# upstream_cache_status 内置变量 是否命中缓存 测试使用
add_header nginx-cache "$upstream_cache_status";
proxy_pass http://backend/js/;
}
}
}
```

20 Nginx缓存的清除

方式一：删除对应的缓存目录

```
rm -rf /usr/local/proxy_cache/.....
```

方式二：使用第三方扩展模块

http://labs.frickle.com/nginx_ngx_cache_purge/

ngx_cache_purge

(1) 下载ngx_cache_purge模块对应的资源包，并上传到服务器上。

```
ngx_cache_purge-2.3.tar.gz
```

(2) 对资源文件进行解压缩

```
tar -zxf ngx_cache_purge-2.3.tar.gz
```

(3) 修改文件夹名称，方便后期配置

```
mv ngx_cache_purge-2.3 purge
```

(4) 查询Nginx的配置参数

```
nginx -V
```

(5) 进入Nginx的安装目录，使用./configure进行参数配置

```
--prefix=/usr/local/nginx --sbin-path=/usr/local/nginx/sbin/nginx
--modules-path=/usr/local/nginx/modules --conf-
path=/usr/local/nginx/conf/nginx.conf --error-log-
path=/usr/local/nginx/logs/error.log --http-log-
path=/usr/local/nginx/logs/access.log --pid-
path=/usr/local/nginx/logs/nginx.pid --lock-
path=/usr/local/nginx/logs/nginx.lock --with-
http_gzip_static_module --with-http_ssl_module --with-stream
--add-module=/opt/purge

./configure --add-module=/root/nginx/module/purge
```

(6) 使用make进行编译

```
make
```

(7) 将nginx安装目录的nginx二级制可执行文件备份

```
mv /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginxold
```

(8) 将编译后的objs中的nginx拷贝到nginx的sbin目录下

```
cp objs/nginx /usr/local/nginx/sbin
```

(9) 使用make进行升级

```
make upgrade
```

(10) 在nginx配置文件中进行如下配置

```
server{
    location ~/purge(/.*) {
        # proxy_cache_purge 缓存名称    key
        proxy_cache_purge wjl_key wjl08;
    }
}
```

访问被缓存的资源就可以直接清除

成功清除

密钥: wjl08

路径: /opt/nginx/proxy_cache/4e/6/57/72a56d00dd74c419f31e6ce43fa5764e

nginx/1.21.6

21 设置资源不缓存

对于一些经常发生变化的数据。如果进行缓存的话，就容易出现用户访问到的数据不是服务器真实的数据。所以对于这些资源我们在缓存的过程中就需要进行过滤，不进行缓存。

Nginx也提供了这块的功能设置，需要使用到如下两个指令

proxy_no_cache

该指令是用来定义**不将数据进行缓存的条件**。

语法	proxy_no_cache string ...;
默认值	—
位置	http、server、location

配置实例

```
proxy_no_cache $cookie_nocache $arg_nocache $arg_comment;  
# $cookie_nocache $arg_nocache $arg_comment; 有一个满足就不缓存
```

proxy_cache_bypass

该指令是用来设置**不从缓存中获取数据的条件**。

语法	proxy_cache_bypass string ...;
默认值	—
位置	http、server、location

配置实例

```
proxy_cache_bypass $cookie_nocache $arg_nocache $arg_comment;
```

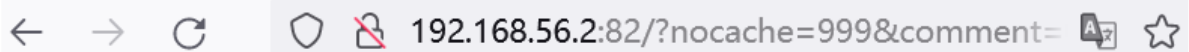
上述两个指令都有一个指定的条件，这个条件可以是多个，并且多个条件中至少有一个不为空且不等于"0"，则条件满足成立。上面给的配置实例是从官方网站获取的，里面使用到了三个变量，分别是\$cookie_nocache、\$arg_nocache、\$arg_comment

\$cookie_nocache、\$arg_nocache、\$arg_comment

这三个参数分别代表的含义是：

```
$cookie_nocache
> add_header Set-Cookie 'nocache=888';
指的是当前请求的cookie中键的名称为nocache对应的值
$arg_nocache和$arg_comment
指的是当前请求的参数中属性名为nocache和comment对应的属性值
?nocache=888

### $arg_自定义参数名
访问时 ?自定义参数名 就可以获得
```



cookie_nocache===>888===arg_nocache===>999===arg_comment===>000

```
# 不缓存js
server{
    listen 8080;
    server_name localhost;
    location / {
        if ($request_uri ~ /\.*\.$js$){
            # 自定义变量
            set $nocache/$myargs 1;
        }
        proxy_no_cache $nocache $cookie_nocache $nocache/$myargs
$arg_comment;
        proxy_cache_bypass $nocache $cookie_nocache $arg_nocache
$arg_comment;
    }
}
```


动静分离

什么是动静分离？

动：后台应用程序的业务处理

静：网站的静态资源(html, javascript, css, images等文件)

分离：将两者进行分开部署访问，提供用户进行访问。举例说明就是以后所有和静态资源相关的内容都交给Nginx来部署访问，非静态内容则交给类似于Tomcat的服务器来部署访问。

为什么要动静分离？

前面我们介绍过Nginx在处理静态资源的时候，效率是非常高的，而且Nginx的并发访问量也是名列前茅，而Tomcat则相对比较弱一些，所以把静态资源交给Nginx后，可以减轻Tomcat服务器的访问压力并提高静态资源的访问速度。

动静分离以后，降低了动态资源和静态资源的耦合度。如动态资源宕机了也不影响静态资源的展示。

如何实现动静分离？

实现动静分离的方式很多，比如静态资源可以部署到CDN、Nginx等服务器上，动态资源可以部署到Tomcat,weblogic或者websphere上。本次课程只要使用Nginx+Tomcat来实现动静分离。

```
listen      80;
# server_name www.wjl1128.top;
server_name ~^www\.(w+)\.com$;
charset utf-8;

location /tom {
    proxy_pass http://tomcat/;
}

# 动态资源访问
location /demo {
    proxy_no_cache    1;
    # root    html;
    # index  index.html index.htm;
    proxy_pass http://192.168.10.102:8080;
}

# 静态资源
location ~ /\.*\.(js|png|jpg|html)$ {
    root    html/web;
    index  index.html;
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script src="js/jquery.min.js"></script>
  <script>
    $(function(){
      $.get('/demo/getAddress',function(data){
        $("#msg").html(data);
      });
    });
  </script>
</head>
<body>

  <h3 id="msg"></h3>

  
  
</body>
</html>
```

Nginx高可用环境

Keepalived

使用Keepalived来解决, Keepalived 软件由 C 编写的, 最初是专为 LVS 负载均衡软件设计的, Keepalived 软件主要是通过 VRRP 协议实现高可用功能。

VRRP介绍



VRRP (Virtual Route Redundancy Protocol) 协议，翻译过来为虚拟路由冗余协议。VRRP协议将两台或多台路由器设备虚拟成一个设备，对外提供虚拟路由器IP,而在路由器组内部，如果实际拥有这个对外IP的路由器如果工作正常的话就是MASTER, MASTER实现针对虚拟路由器IP的各种网络功能。其他设备不拥有该虚拟IP，状态为BACKUP, 除了接收MASTER的VRRP状态通告信息以外，不执行对外的网络功能。当主机失效时，BACKUP将接管原先MASTER的网络功能。

从上面的介绍信息获取到的内容就是VRRP是一种协议，那这个协议是用来干什么的？

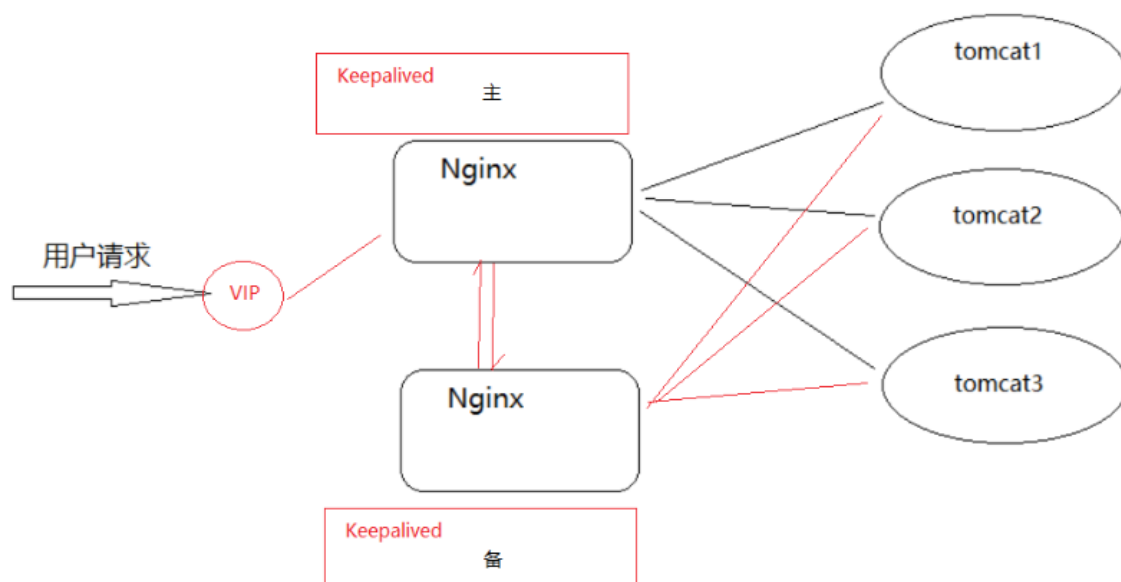
1. 选择协议

VRRP可以把一个虚拟路由器的责任动态分配到局域网上的 VRRP 路由器中的一台。其中的虚拟路由即Virtual路由是由VRRP路由群组创建的一个不真实存在的路由，这个虚拟路由也是有对应的IP地址。而且VRRP路由1和VRRP路由2之间会有竞争选择，通过选择会产生一个Master路由和一个Backup路由。

2. 路由容错协议

Master路由和Backup路由之间会有一个心跳检测，Master会定时告知Backup自己的状态，如果在指定的时间内，Backup没有接收到这个通知内容，Backup就会替代Master成为新的Master。Master路由有一个特权就是虚拟路由和后端服务器都是通过Master进行数据传递交互的，而备份节点则会直接丢弃这些请求和数据，不做处理，只是去监听Master的状态

用了Keepalived后，解决方案如下：



环境准备

VIP	IP	主机名	主/从
	192.168.200.133	keepalived1	Master
192.168.200.222			
	192.168.200.122	keepalived2	Backup

keepalived的安装

```
./configure --sysconf=/etc --prefix=/usr/local
步骤1:从官方网站下载keepalived,官网地址https://keepalived.org/
步骤2:将下载的资源上传到服务器
      keepalived-2.0.20.tar.gz
步骤3:创建keepalived目录,方便管理资源
      mkdir keepalived
步骤4:将压缩文件进行解压缩,解压缩到指定的目录
      tar -zxf keepalived-2.0.20.tar.gz -C keepalived/
步骤5:对keepalived进行配置,编译和安装
      cd keepalived/keepalived-2.0.20
      ./configure --sysconf=/etc --prefix=/usr/local
      make && make install
```

安装完成后,有两个文件需要我们认识下,一个是

`**/etc/keepalived/keepalived.conf**` (keepalived的系统配置文件,我们主要操作的就是该文件),一个是`/usr/local/sbin`目录下的 `keepalived`,是系统配置脚本,用来启动和关闭keepalived

Keepalived配置文件介绍

打开keepalived.conf配置文件

这里面会分三部,第一部分是global全局配置、第二部分是vrrp相关配置、第三部分是LVS相关配置。本次课程主要是使用keepalived实现高可用部署,没有用到LVS,所以我们重点关注的是前两部分

```
global全局部分:
global_defs {
    #通知邮件,当keepalived发送切换时需要发email给具体的邮箱地址
    notification_email {
        tom@itcast.cn
        jerry@itcast.cn
    }
    #设置发件人的邮箱信息
```

```
notification_email_from zhaomin@itcast.cn
#指定smtp服务地址
smtp_server 192.168.200.1
#指定smtp服务连接超时时间
smtp_connect_timeout 30
#运行keepalived服务器的一个标识，可以用作发送邮件的主题信息
router_id LVS_DEVEL
```

#默认是不跳过检查。检查收到的VRRP通告中的所有地址可能会比较耗时，设置此命令的意思是，如果通告与接收的上一个通告来自相同的master路由器，则不执行检查(跳过检查)

```
vrrp_skip_check_adv_addr
#严格遵守VRRP协议。
vrrp_strict
#在一个接口发送的两个免费ARP之间的延迟。可以精确到毫秒级。默认是0
vrrp_garp_interval 0
#在一个网卡上每组na消息之间的延迟时间，默认为0
vrrp_gna_interval 0
```

}
VRRP部分，该部分可以包含以下四个子模块

1. vrrp_script
2. vrrp_sync_group
3. garp_group
4. vrrp_instance

我们会用到第一个和第四个，

#设置keepalived实例的相关信息，VI_1为VRRP实例名称

```
vrrp_instance VI_1 {
    state MASTER          #有两个值可选MASTER主 BACKUP备
    interface ens33       #vrrp实例绑定的接口，用于发送VRRP包[当前服务器使用的网卡名称]
    virtual_router_id 51  #指定VRRP实例ID，范围是0-255
    priority 100          #指定优先级，优先级高的将成为MASTER
    advert_int 1          #指定发送VRRP通告的间隔，单位是秒
    authentication {      #vrrp之间通信的认证信息
        auth_type PASS    #指定认证方式。PASS简单密码认证(推荐)
        auth_pass 1111    #指定认证使用的密码，最多8位
    }
    virtual_ipaddress {   #虚拟IP地址设置虚拟IP地址，供用户访问使用，可设置多个，一行一个
        192.168.200.222
    }
}
```

配置内容如下：

服务器1

```

global_defs {
    notification_email {
        tom@itcast.cn
        jerry@itcast.cn
    }
    notification_email_from zhaomin@itcast.cn
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id keepalived1
    vrrp_skip_check_adv_addr
    vrrp_strict
    vrrp_garp_interval 0
    vrrp_gna_interval 0
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.200.222
    }
}

```

服务器2

! Configuration File for keepalived

```

global_defs {
    notification_email {
        tom@itcast.cn
        jerry@itcast.cn
    }
    notification_email_from zhaomin@itcast.cn
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id keepalived2
    vrrp_skip_check_adv_addr
    vrrp_strict
    vrrp_garp_interval 0
    vrrp_gna_interval 0
}

```

```

}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        # 可以配置多个
        192.168.200.222
    }
}

```

访问测试

1. 启动keepalived之前，咱们先使用命令 `ip a` ,查看192.168.200.133和192.168.200.122这两台服务器的IP情况。

1. 分别启动两台服务器的keepalived

```

cd /usr/local/sbin
./keepalived

```

1. 当把192.168.200.133服务器上的keepalived关闭后，再次查看ip

通过上述的测试，我们会发现，虚拟IP(VIP)会在MASTER节点上，当MASTER节点上的keepalived出问题以后，因为BACKUP无法收到MASTER发出的VRRP状态通过信息，就会直接升为MASTER。VIP也会"漂移"到新的MASTER。

上面测试和Nginx有什么关系？

我们把192.168.200.133服务器的keepalived再次启动下，由于它的优先级高于服务器192.168.200.122的，所有它会再次成为MASTER，VIP也会"漂移"过去，然后我们再次通过浏览器访问：

```
http://192.168.200.222/
```

如果把192.168.200.133服务器的keepalived关闭掉，再次访问相同的地址

效果实现了以后， 我们会发现要想让vip进行切换，就必须要把服务器上的keepalived进行关闭，而什么时候关闭keepalived呢?应该是在keepalived所在服务器的nginx出现问题后，把keepalived关闭掉，就可以让VIP执行另外一台服务器，但是在这所有的操作都是通过手动来完成的，我们如何能让系统自动判断当前服务器的nginx是否正确启动，如果没有，要能让VIP自动进行"漂移"，这个问题该如何解决？

keepalived之vrrp_script

keepalived只能做到对网络故障和keepalived本身的监控，即当出现网络故障或者keepalived本身出现问题时，进行切换。但是这些还不够，我们还需要监控keepalived所在服务器上的其他业务，比如Nginx,如果Nginx出现异常了，仅仅keepalived保持正常，是无法完成系统的正常工作的，因此需要根据业务进程的运行状态决定是否需要主备切换，这个时候，我们可以通过编写脚本对业务进程进行检测监控。

实现步骤：

1. 在keepalived配置文件中添加对应的配置像

```
vrrp_script 脚本名称
{
    script "脚本位置"
    interval 3 #执行时间间隔
    weight -20 #动态调整vrrp_instance的优先级
}
```

1. 编写脚本

ck_nginx.sh

查看nginx进程 如果没有启动 就启动 之后脚本睡两秒 然后再次检查 如果还是没有启动 就将keepalived进程杀掉

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

Linux ps命令用于显示当前进程（process）的状态。

-C(command) :指定命令的所有进程

--no-header 排除标题

1. 为脚本文件设置权限

```
chmod 755 ck_nginx.sh
```

1. 将脚本添加到

```
vrrp_script ck_nginx {  
    script "/etc/keepalived/ck_nginx.sh" #执行脚本的位置  
    interval 2          #执行脚本的周期，秒为单位  
    weight -20          #权重的计算方式  
}  
vrrp_instance VI_1 {  
    state MASTER  
    interface ens33  
    virtual_router_id 10  
    priority 100  
    advert_int 1  
    authentication {  
        auth_type PASS  
        auth_pass 1111  
    }  
    virtual_ipaddress {  
        192.168.200.111  
    }  
    track_script {  
        ck_nginx  
    }  
}
```

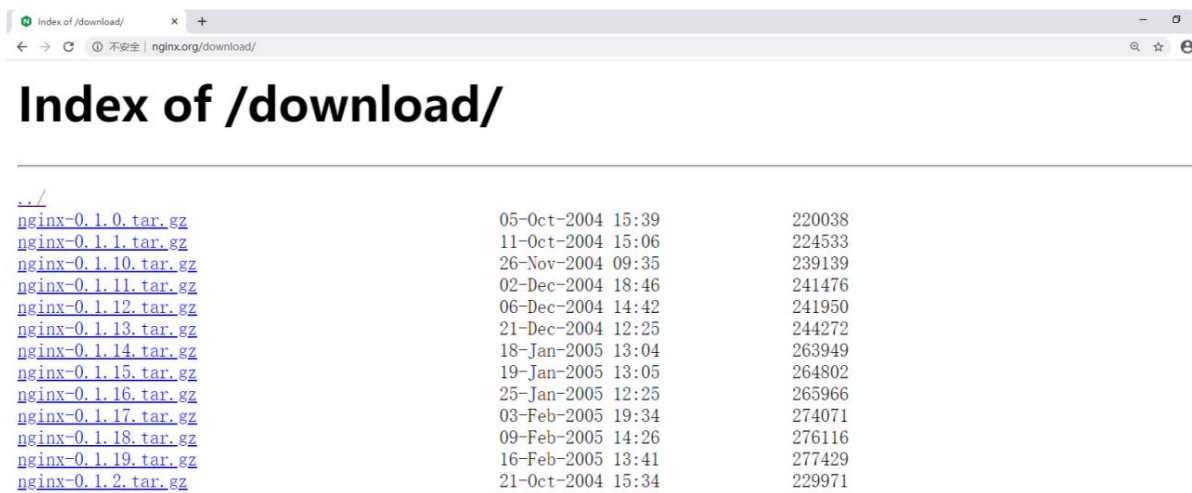
1. 如果效果没有出来，可以使用 `tail -f /var/log/messages` 查看日志信息，找对应的错误信息。

2. 测试

问题思考：

通常如果master服务死掉后backup会变成master，但是当master服务又好了的时候master此时会抢占VIP，这样就会发生两次切换对业务繁忙的网站来说是不好的。所以我们要在配置文件加入 `nopreempt` 非抢占，但是这个参数只能用于state 为backup，故我们在用HA的时候最好master 和backup的state都设置成backup 让其通过priority来竞争。

nginx 制作下载站点



如何制作一个下载站点：

nginx使用的是模块ngx_http_autoindex_module来实现的，该模块处理以斜杠("/")结尾的请求，并生成目录列表。

nginx编译的时候会自动加载该模块，但是该模块默认是关闭的，我们需要使用下列指令来完成对应的配置

(1) autoindex:启用或禁用目录列表输出

语法	<code>autoindex on off;</code>
默认值	<code>autoindex off;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code>

(2) autoindex_exact_size:对应HTML格式，指定是否在目录列表展示文件的详细大小

默认为on，显示出文件的确切大小，单位是bytes。 改为off后，显示出文件的大概大小，单位是kB或者MB或者GB

语法	<code>autoindex_exact_size on off;</code>
默认值	<code>autoindex_exact_size on;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code>

(3) autoindex_format: 设置目录列表的格式

语法	<code>autoindex_format html xml json jsonp;</code>
默认值	<code>autoindex_format html;</code>
位置	<code>http</code> 、 <code>server</code> 、 <code>location</code>

注意:该指令在1.7.9及以后版本中出现

(4) autoindex_localtime:对应HTML格式，是否在目录列表上显示时间。

默认为off，显示的文件时间为GMT时间。 改为on后，显示的文件时间为文件的服务器时间







语法	<code>autoindex_localtime on off;</code>
默认值	<code>autoindex_localtime off;</code>
位置	http、server、location

配置方式如下：

```
# 文件位于 /usr/local/download
location /download{
    root /usr/local;
    autoindex on;
    autoindex_exact_size on;
    autoindex_format html;
    autoindex_localtime on;
}

# 静态下载站点
location /opt {
    root /;
    autoindex on;
    autoindex_exact_size on;
    autoindex_format html;
    autoindex_localtime off;
}
```

XML/JSON格式[一般不用这两种方式]

   192.168.56.2/opt/   		
<h2>Index of /opt/</h2>		
<hr/>		
../	30-Mar-2022 18:18	-
app/	05-Apr-2022 12:20	-
containerd/	04-Apr-2022 11:10	-
fair/	05-Apr-2022 19:16	-
keepalived/	31-Mar-2022 21:57	-
logs/	04-Apr-2022 15:20	-
nginx/	23-Dec-2014 18:40	-
purge/	30-Oct-2018 19:17	-
rh/	30-Mar-2022 18:47	-
tomcat/	30-Mar-2022 18:34	-
txt/	04-Apr-2022 18:42	5300294
demo_simple.war	05-Apr-2022 14:59	1036063
keepalived-2.0.20.tar.gz	04-Apr-2022 16:26	12248
ngx_cache_purge-2.3.tar.gz		
<hr/>		

Nginx的用户认证模块

对应系统资源的访问，我们往往需要限制谁能访问，谁不能访问。这块就是我们通常所说的认证部分，认证需要做的就是根据用户输入的用户名和密码来判定用户是否为合法用户，如果是则放行访问，如果不是则拒绝访问。

Nginx对应用户认证这块是通过ngx_http_auth_basic_module模块来实现的，它允许通过使用"HTTP基本身份验证"协议验证用户名和密码来限制对资源的访问。默认情况下nginx是已经安装了该模块，如果不需要则使用--without-http_auth_basic_module。

该模块的指令比较简单，

(1) auth_basic:使用“ HTTP基本认证”协议启用用户名和密码的验证

语法	auth_basic string off;
默认值	auth_basic off;
位置	http,server,location,limit_except

开启后，服务端会返回401，指定的字符串会返回到客户端，给用户以提示信息，但是不同的浏览器对内容的展示不一致。

(2) auth_basic_user_file:指定用户名和密码所在文件

语法	auth_basic_user_file file;
默认值	-
位置	http,server,location,limit_except

指定文件路径，该文件中的用户名和密码的设置，密码需要进行加密。可以采用工具自动生成实现步骤：

1.nginx.conf添加如下内容

```
location /download{
    root /usr/local;
    autoindex on;
    autoindex_exact_size on;
    autoindex_format html;
    autoindex_localtime on;
    auth_basic 'please input your auth';
    auth_basic_user_file httpasswd;
}
```

2.我们需要使用 `htpasswd` 工具生成

```
yum install -y httpd-tools
htpasswd -c /usr/local/nginx/conf/htpasswd username //创建一个新文件
记录用户名和密码
htpasswd -b /usr/local/nginx/conf/htpasswd username password //在指
定文件新增一个用户名和密码
htpasswd -D /usr/local/nginx/conf/htpasswd username //从指定文件删除
一个用户信息
htpasswd -v /usr/local/nginx/conf/htpasswd username //验证用户名和密
码是否正确
```

上述方式虽然能实现用户名和密码的验证，但是大家也看到了，所有的用户名和密码信息都记录在文件里面，如果用户量过大的话，这种方式就显得有点麻烦了，这时候我们就得通过后台业务代码来进行用户权限的校验了。

Lua

21 lua的基本语法

概念

Lua是一种轻量、小巧的脚本语言，用标准C语言编写并以源代码形式开发。设计的目的是为了嵌入到其他应用程序中，从而为应用程序提供灵活的扩展和定制功能。

特性

跟其他语言进行比较，Lua有其自身的特点：

(1) 轻量级

Lua用标准C语言编写并以源代码形式开发，编译后仅仅一百余千字节，可以很方便的嵌入到其他程序中。

(2) 可扩展

Lua提供非常丰富易于使用的扩展接口和机制，由宿主语言（通常是C或C++）提供功能，Lua可以使用它们，就像内置的功能一样。

(3) 支持面向过程编程和函数式编程

应用场景

Lua在不同的系统中得到大量应用，场景的应用场景如下：

游戏开发、独立应用脚本、web应用脚本、扩展和数据库插件、系统安全上。

Lua的安装

在Linux上安装Lua非常简单，只需要下载源码包并在终端解压、编译即可使用。

Lua的官网地址为：<https://www.lua.org>

1. 点击download可以找到对应版本的下载地址，我们本次课程采用的是lua-5.3.5,其对应的资源链接地址为<https://www.lua.org/ftp/lua-5.4.1.tar.gz>,也可以使用wget命令直接下载：

```
wget https://www.lua.org/ftp/lua-5.4.1.tar.gz
```

1. 编译安装

```
cd lua-5.4.1
make linux test
make install
```

如果在执行make linux test失败，报如下错误：

```
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_2 -DLUA_USE_LINUX -c -o lua.o lua.c
lua.c:82:31: fatal error: readline/readline.h: No such file or directory
#include <readline/readline.h>
compilation terminated.
make[2]: *** [lua.o] Error 1
make[2]: Leaving directory '/root/lua-5.3.5/src'
```

说明当前系统缺少libreadline-dev依赖包，需要通过命令来进行安装

```
yum install -y readline-devel
```

验证是否安装成功

```
lua -v
```

标识符

换句话说标识符就是我们的变量名，Lua定义变量名以一个字母 A 到 Z 或 a 到 z 或下划线 _ 开头后加上0个或多个字母，下划线，数字（0到9）。这块建议大家最好不要使用下划线加大写字母的标识符，因为Lua的保留字也是这样定义的，容易发生冲突。注意Lua是区分大小写字母的。

A0

关键字

下列是Lua的关键字，大家在定义常量、变量或其他用户自定义标识符都要避免使用以下这些关键字：

and	break	do	else
elseif	end	false	for
function	if	in	local
nil	not	or	repeat
return	then	true	until
while	goto		

一般约定，以下划线开头连接一串大写字母的名字（比如 `_VERSION`）被保留用于 Lua 内部全局变量。这个也是上面我们不建议这么定义标识符的原因。

运算符

Lua中支持的运算符有算术运算符、关系运算符、逻辑运算符、其他运算符。

算术运算符：

+	加法
-	减法
*	乘法
/	除法
%	取余
^	乘幂
-	负号

例如：

10+20	→30
20-10	→10
10*20	→200
20/10	→2
3%2	→1
10^2	→100
-10	→-10

关系运算符

= 等于
~= 不等于
> 大于
< 小于
≥ 大于等于
≤ 小于等于

例如：

```
10=10      →true
10~=10     →false
20>10      →true
20<10      →false
20≥10      →true
20≤10      →false
```

逻辑运算符

and 逻辑与 A and B &&
or 逻辑或 A or B ||
not 逻辑非 取反，如果为true,则返回false ！

逻辑运算符可以作为if的判断条件，返回的结果如下：

```
A = true
B = true

A and B →true
A or B →true
not A →false

A = true
B = false

A and B →false
A or B →true
not A →false

A = false
B = true

A and B →false
A or B →true
not A →true
```


其他运算符

..
#

连接两个字符串
一元预算法，返回字符串或表的长度

例如：

```
> "HELLO " .. "WORLD"    →HELLO WORLD
> #"HELLO"               →5
```

全局变量&局部变量

在Lua语言中，全局变量无须声明即可使用。在默认情况下，变量总是认为是全局的，如果未提前赋值，默认为nil：

要想声明一个局部变量，需要使用local来声明

Lua数据类型

Lua有8个数据类型

nil(空, 无效值)
boolean(布尔, true/false)
number(数值)
string(字符串)
function(函数)
table (表)
thread(线程)
userdata (用户数据)

可以使用type函数测试给定变量或者的类型：

```
print(type(nil))           →nil
print(type(true))          → boolean
print(type(1.1*1.1))       → number
print(type("Hello world")) → string
print(type(io.stdin))       →userdata
print(type(print))         → function
print(type(type))          →function
print(type{})              →table
print(type(type(X)))       → string
```

nil

nil是一种只有一个nil值的类型，它的作用可以用来与其他所有值进行区分，也可以当想要移除一个变量时，只需要将该变量名赋值为nil，垃圾回收就会释放该变量所占用的内存。

boolean

boolean类型具有两个值，true和false。boolean类型一般被用来做条件判断的真与假。在Lua语言中，只会将false和nil视为假，其他的都视为真，特别是在条件检测中0和空字符串都会认为是真，这个和我们熟悉的大多数语言不太一样。

number

在Lua5.3版本开始，Lua语言为数值格式提供了两种选择：integer（整型）和float（双精度浮点型）[和其他语言不太一样，float不代表单精度类型]。

数值常量的表示方式：

```
>4           →4
>0.4         →0.4
>4.75e-3     →0.00475
>4.75e3      →4750
```

不管是整型还是双精度浮点型，使用type()函数来取其类型，都会返回的是number

```
>type(3)      →number
>type(3.3)    →number
```

所以它们之间是可以相互转换的，同时，具有相同算术值的整型值和浮点型值在Lua语言中是相等的

string

Lua语言中的字符串即可以表示单个字符，也可以表示一整本书籍。在Lua语言中，操作100K或者1M个字母组成的字符串的程序很常见。

可以使用单引号或双引号来声明字符串

```
>a = "hello"
>b = 'world'
>print(a)   →hello
>print(b)   →world
```

如果声明的字符串比较长或者有多行，则可以使用如下方式进行声明

```
html = [[
<html>
<head>
<title>Lua-string</title>
</head>
<body>
<a href="http://www.lua.org">Lua</a>
</body>
</html>
]]
```

table

table是Lua语言中最主要和强大的数据结构。使用表， Lua 语言可以以一种简单、统一且高效的方式表示数组、集合、记录和其他很多数据结构。 Lua语言中的表本质上是一种辅助数组。这种数组比Java中的数组更加灵活，可以使用数值做索引，也可以使用字符串或其他任意类型的值作索引(除nil外)。

创建表的最简单方式：

```
> a = {}
```

创建数组：

我们都知道数组就是相同数据类型的元素按照一定顺序排列的集合，那么使用table如何创建一个数组呢？

```
>arr = {"TOM","JERRY","ROSE"}
```

要想获取数组中的值，我们可以通过如下内容来获取：

```
print(arr[0])      nil
print(arr[1])      TOM
print(arr[2])      JERRY
print(arr[3])      ROSE
```

从上面的结果可以看出来，数组的下标默认是从1开始的。所以上述创建数组，也可以通过如下方式来创建

```
>arr = {}
>arr[1] = "TOM"
>arr[2] = "JERRY"
>arr[3] = "ROSE"
```

上面我们说过了，表的索引即可以是数字，也可以是字符串等其他的内容，所以我们可以将索引更改为字符串来创建

```
>arr = {}  
>arr["X"] = 10  
>arr["Y"] = 20  
>arr["Z"] = 30
```

当然，如果想要获取这些数组中的值，可以使用下面的方式

```
方式一  
>print(arr["X"])  
>print(arr["Y"])  
>print(arr["Z"])  
方式二  
>print(arr.X)  
>print(arr.Y)  
>print(arr.Z)
```

当前table的灵活不逊于此，还有更灵活的声明方式

```
>arr = {"TOM",X=10,"JERRY",Y=20,"ROSE",Z=30}
```

如何获取上面的值？

```
TOM : arr[1]  
10 : arr["X"] | arr.X  
JERRY: arr[2]  
20 : arr["Y"] | arr.Y  
ROESE?
```

function

在 Lua语言中，函数（ Function ）是对语句和表达式进行抽象的主要方式。

定义函数的语法为：

```
function functionName(params)  
  
end
```

函数被调用的时候，传入的参数个数与定义函数时使用的参数个数不一致的时候，Lua 语言会通过 抛弃多余参数和将不足的参数设为 nil 的方式来调整参数的个数。

```
function f(a,b)
print(a,b)
end

f()      → nil nil
f(2)     → 2 nil
f(2,6)   → 2 6
f(2.6.8) → 2 6 (8被丢弃)
```

可变量参数函数

```
function add(...)
    a,b,c=...
    print(a)
    print(b)
    print(c)
end

add(1,2,3) → 1 2 3
```

函数返回值可以有多个，这点和Java不太一样

```
function f(a,b)
return a,b
end

x,y=f(11,22) → x=11,y=22
```

thread

thread翻译过来是线程的意思，在Lua中，thread用来表示执行的独立线路，用来执行协同程序。

userdata

userdata是一种用户自定义数据，用于表示一种由应用程序或C/C++语言库所创建的类型。

Lua控制结构

Lua 语言提供了一组精简且常用的控制结构，包括用于条件执行的if 以及用于循环的while、 repeat 和 for。所有的控制结构语法上都有一个显式的终结符： end 用于终结if、 for 及 while 结构， until 用于终结 repeat 结构。

if then elseif else

if语句先测试其条件，并根据条件是否满足执行相应的 then 部分或 else 部分。else 部分是可选的。

```
function testif(a)
  if a>0 then
    print("a是正数")
  end
end

function testif(a)
  if a>0 then
    print("a是正数")
  else
    print("a是负数")
  end
end
```

如果要编写嵌套的 if 语句，可以使用 elseif。它类似于在 else 后面紧跟一个if。根据传入的年龄返回不同的结果，如

```
age ≤ 18 青少年,
age > 18 , age ≤ 45 青年
age > 45 , age ≤ 60 中年人
age > 60 老年人

function show(age)
  if age ≤ 18 then
    return "青少年"
  elseif age > 18 and age ≤ 45 then
    return "青年"
  elseif age > 45 and age ≤ 60 then
    return "中年人"
  elseif age > 60 then
    return "老年人"
  end
end
```

while循环

顾名思义，当条件为真时 while 循环会重复执行其循环体。Lua 语言先测试 while 语句的条件，若条件为假则循环结束；否则，Lua 会执行循环体并不断地重复这个过程。

语法：

```
while 条件 do
    循环体
end
```

例子:实现数组的循环

```
function testWhile()
    local i = 1
    while i ≤ 10 do
        print(i)
        i=i+1
    end
end
```

repeat循环

顾名思义， repeat-until语句会重复执行其循环体直到条件为真时结束。 由于条件测试在循环体之后执行，所以循环体至少会执行一次。

语法

```
repeat
    循环体
until 条件
function testRepeat()
    local i = 10
    repeat
        print(i)
        i=i-1
    until i < 1
end

--[ 变量定义 --]
a = 10
--[ 执行循环 --]
repeat
    print("a的值为:", a)
    a = a + 1
until( a > 15 )
```

for循环

数值型for循环

语法

```
for param=exp1,exp2,exp3 do
    循环体
end
```

param的值从exp1变化到exp2之前的每次循环会执行 循环体，并在每次循环结束后将步长(step)exp3增加到param上。exp3可选，如果不设置默认为1

```
for i = 1,100,10 do
    print(i)
end
```

泛型for循环

泛型for循环通过一个迭代器函数来遍历所有值，类似于java中的foreach语句。

语法

```
for i,v in ipairs(x) do
    循环体
end
```

i是数组索引值，v是对应索引的数组元素值，ipairs是Lua提供的一个迭代器函数，用来迭代数组，x是要遍历的数组。

例如：

```
arr = {"TOME","JERRY","ROWS","LUCY"}
for i,v in ipairs(arr) do
    print(i,v)
end
```

上述实例输出的结果为

```
1   TOM
2   JERRY
3   ROWS
4   LUCY
```

但是如果将arr的值进行修改为

```
arr = {"TOME","JERRY","ROWS",x="JACK","LUCY"}
```

同样的代码在执行的时候，就只能看到和之前一样的结果，而其中的x为JACK就无法遍历出来，缺失了数据，如果解决呢？

我们可以将迭代器函数变成pairs,如


```
for i,v in pairs(arr) do
    print(i,v)
end
```

上述实例就输出的结果为

```
1  TOM
2  JERRY
3  ROWS
4  LUCY
x  JACK
```

ngx_lua模块概念

淘宝开发的ngx_lua模块通过将lua解释器集成进Nginx，可以采用lua脚本实现业务逻辑，由于lua的紧凑、快速以及内建协程，所以在保证高并发服务能力的同时极大地降低了业务逻辑实现成本。

安装

方式一：lua-nginx-module

1. LuaJIT是采用C语言编写的Lua代表的解释器。

官网地址为：<http://luajit.org/>

在官网上找到对应的下载地址：<http://luajit.org/download/LuaJIT-2.0.5.tar.gz>

在centos上使用wget来下载：wget <http://luajit.org/download/LuaJIT-2.0.5.tar.gz>

将下载的资源进行解压：tar -zxvf LuaJIT-2.0.5.tar.gz

进入解压的目录：cd LuaJIT-2.0.5

执行编译和安装：make && make install

1. 下载lua-nginx-module

下载地址：<https://github.com/openresty/lua-nginx-module/archive/v0.10.16rc4.tar.gz>

在centos上使用wget来下载：wget <https://github.com/openresty/lua-nginx-module/archive/v0.10.16rc4.tar.gz>

将下载的资源进行解压: `tar -zxf lua-nginx-module-0.10.16rc4.tar.gz`

更改目录名: `mv lua-nginx-module-0.10.16rc4 lua-nginx-module`

导入环境变量, 告诉Nginx去哪里找luajit

```
export LUAJIT_LIB=/usr/local/lib
export LUAJIT_INC=/usr/local/include/luajit-2.0
```

进入Nginx的目录执行如下命令:

```
./configure --prefix=/usr/local/nginx --add-module=../lua-nginx-
module
make && make install
```

注意事项:

(1) 如果启动Nginx出现如下错误:

解决方案:

设置软链接, 使用如下命令

```
ln -s /usr/local/lib/libluajit-5.1.so.2 /lib64/libluajit-5.1.so.2
```

(2) 如果启动Nginx出现以下错误信息

分析原因: 因为lua-nginx-module是来自openresty, 错误中提示的resty.core是openresty的核心模块, 对其下的很多函数进行了优化等工作。以前的版本默认不会把该模块编译进去, 所以需要使用的話, 我们得手动安装, 或者禁用就可以。但是最新的lua-nginx-module模块已经强制性安装了该模块, 所以此处因为缺少resty模块导致的报错信息。

解决方案有两个: 一种是下载对应的模块, 另一种则是禁用掉resty模块, 禁用的方式为:

```
http{
    lua_load_resty_core off;
}
```

1. 测试

在nginx.conf下配置如下内容:

```
location /lua{
    default_type 'text/html';
    content_by_lua 'ngx.say("<h1>HELLO,LUA</h1>")';
}
```

配置成功后，启动nginx，通过浏览器进行访问，如果获取到如下结果，则证明安装成功。

方式二:OpenResty

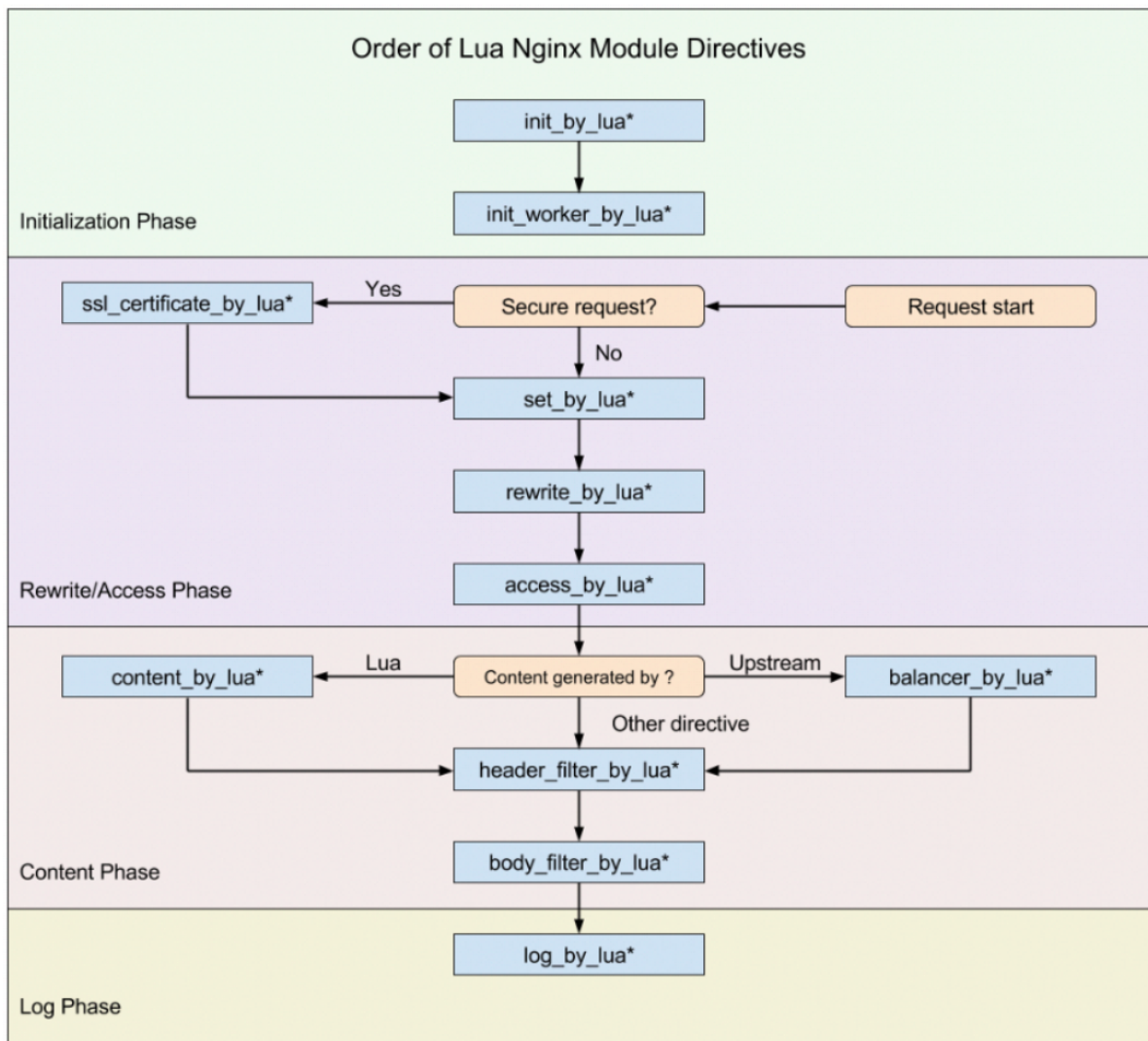
概述

前面我们提到过，OpenResty是由淘宝工程师开发的，所以其官方网站(<http://openresty.org/>)我们读起来是非常的方便。OpenResty是一个基于Nginx与 Lua 的高性能 Web 平台，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。所以本身OpenResty内部就已经集成了Nginx和Lua，所以我们使用起来会更加方便。

安装

- (1) 下载OpenResty: <https://openresty.org/download/openresty-1.15.8.2.tar.gz>
 - (2) 使用wget下载: `wget https://openresty.org/download/openresty-1.15.8.2.tar.gz`
 - (3) 解压缩: `tar -zxf openresty-1.15.8.2.tar.gz`
 - (4) 进入OpenResty目录: `cd openresty-1.15.8.2`
 - (5) 执行命令: `./configure`
 - (6) 执行命令: `make && make install`
 - (7) 进入OpenResty的目录，找到nginx: `cd /usr/local/openresty/nginx/`
 - (8) 在conf目录下的nginx.conf添加如下内容
- ```
location /lua{
 default_type 'text/html';
 content_by_lua 'ngx.say("<h1>HELLO,OpenResty</h1>")';
}
```
- (9) 在sbin目录下启动nginx
  - (10) 通过浏览器访问测试

# Lua的使用



先来解释下\*的作用

- \*: 无, 即 `xxx_by_lua`, 指令后面跟的是 lua 指令
- \*:\_file, 即 `xxx_by_lua_file`, 指令后面跟的是 lua 文件
- \*:\_block, 即 `xxx_by_lua_block`, 在 0.9.17 版后替换 `init_by_lua_file`

## init\_by\_lua\*

该指令在每次 Nginx 重新加载配置时执行, 可以用来完成一些耗时模块的加载, 或者初始化一些全局配置。

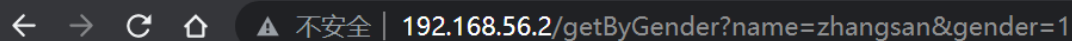
## init\_worker\_by\_lua\*

该指令用于启动一些定时任务, 如心跳检查、定时拉取服务器配置等。

## set\_by\_lua\*

该指令只要用来做变量赋值，这个指令一次只能返回一个值，并将结果赋值给Nginx中指定的变量。

```
http://192.168.56.2/getByGender?name=zhangsan&gender=1
location /getByGender {
 default_type text/html;
 # 声明一个变量
 set_by_lua $param "
 -- 获取请求url上面对应参数的值
 local uri_args = ngx.req.get_uri_args()
 local name = uri_args['name']
 local gender = uri_args['gender']
 if gender == '1' then
 return name..'先生'
 elseif gender == '0' then
 return name..'女士'
 else
 return name;
 end
 ";
 return 200 $param;
}
```



zhangsan先生

## rewrite\_by\_lua\*

该指令用于执行内部URL重写或者外部重定向，典型的如伪静态化URL重写，本阶段在rewrite处理阶段的最后默认执行。

## access\_by\_lua\*

该指令用于访问控制。例如，如果只允许内网IP访问。

## content\_by\_lua\*

该指令是应用最多的指令，大部分任务是在这个阶段完成的，其他的过程往往为这个阶段准备数据，正式处理基本都在本阶段。

```
location /lua {
 default_type text/html;
 content_by_lua 'ngx.say("<h1>Hello OpenResty</h1>")';
}
```

## header\_filter\_by\_lua\*

该指令用于设置应答消息的头部信息。

## body\_filter\_by\_lua\*

该指令是对响应数据进行过滤，如截断、替换。

## log\_by\_lua\*

该指令用于在log请求处理阶段，用Lua代码处理日志，但并不替换原有log处理。

## balancer\_by\_lua\*

该指令主要的作用是用来实现上游服务器的负载均衡器算法

## ssl\_certificate\_by\_\*

该指令作用在Nginx和下游服务开始一个SSL握手操作时将允许本配置项的Lua代码。

# ngx\_lua操作Redis

Redis在系统中经常作为数据缓存、内存数据库使用，在大型系统中扮演着非常重要的作用。在Nginx核心系统中，Redis是常备组件。Nginx支持3种方法访问Redis，分别是HttpRedis模块、HttpRedis2Module、lua-resty-redis库。这三种方式中HttpRedis模块提供的指令少，功能单一，适合做简单缓存，HttpRedis2Module模块比HttpRedis模块操作更灵活，功能更强大。而Lua-resty-redis库是OpenResty提供的一个操作Redis的接口库，可根据自己的业务情况做一些逻辑处理，适合做复杂的业务逻辑。所以本次课程将主要以Lua-resty-redis来进行讲解。

## 步骤二:准备对应的API

lua-resty-redis提供了访问Redis的详细API，包括创建对接、连接、操作、数据处理等。这些API基本上与Redis的操作一一对应。

(1) redis = require "resty.redis"

(2) new

语法: redis,err = redis:new(),创建一个Redis对象。

(3) connect

语法: ok,err=redis:connect(host,port[,options\_table]),设置连接Redis的连接信息。

ok:连接成功返回 1, 连接失败返回nil

err:返回对应的错误信息

(4) set\_timeout

语法: redis:set\_timeout(time) , 设置请求操作Redis的超时时间。

(5) close

语法: ok,err = redis:close(),关闭当前连接，成功返回1，失败返回nil和错误信息

(6) redis命令对应的方法

在lua-resty-redis中，所有的Redis命令都有自己的方法，方法名字和命令名字相同，只是全部为小写。

## 步骤三:效果实现

```
location / {
 default_type "text/html";
 content_by_lua_block{
 local redis = require "resty.redis" -- 引入Redis
 local redisObj = redis:new() --创建Redis对象
 redisObj:set_timeout(1000) --设置超时数据为1s
 local ok,err = redisObj:connect("192.168.200.1",6379) --设置redis连接信息
 if not ok then --判断是否连接成功
 ngx.say("failed to connection redis",err)
 return
 end
 ok,err = redisObj:set("username","TOM")--存入数据
 if not ok then --判断是否存入成功
```

```

 ngx.say("failed to set username",err)
 return
 end
 local res,err = redisObj:get("username") --从redis中获取数据
 ngx.say(res) --将数据写会消息体中
 redisObj:close()
}

}

location /testRedis {
 default_type text/html;
 content_by_lua_block{
 local redis = require "resty.redis"
 local redisObj = redis:new()
 -- 超时时间单位毫秒
 redisObj:set_timeout(1000)
 local ok,err = redisObj:connect("192.168.56.2",6379)
 if not ok then
 ngx.say("<h1>redis连接失败!!! </h1>",err)
 return
 end

 ok,err = redisObj:set("username","ROSE")

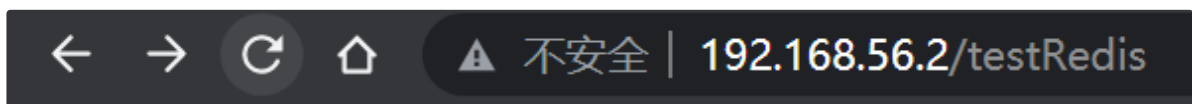
 if not ok then
 ngx.say("<h1>存入数据失败</h1>",err)
 return
 end

 local res,err = redisObj:get("username")
 ngx.say(res)

 redisObj:close()
 }
}

```

步骤四:运行测试效果



ROSE



# ngx\_lua操作MySQL

MySQL是一个使用广泛的关系型数据库。在ngx\_lua中, MySQL有两种访问模式, 分别是使

(1) 用ngx\_lua模块和lua-resty-mysql模块: 这两个模块是安装OpenResty时默认安装的。

(2) 使用drizzle\_nginx\_module(HttpDrizzleModule)模块: 需要单独安装, 这个库现不在OpenResty中。

## lua-resty-mysql

lua-resty-mysql是OpenResty开发的模块, 使用灵活、功能强大, 适合复杂的业务场景, 同时支持存储过程的访问。

## 使用lua-resty-mysql实现数据库的查询

步骤一:

准备MYSQL

```
host: 172.21.55.149
port: 3306
username: root
password: 123456
```

创建一个数据库表及表中的数据。

```
create database nginx_db;

use nginx_db;

create table users(
 id int primary key auto_increment,
 username varchar(30),
 birthday date,
 salary double
);

insert into users(id, username, birthday, salary)
values(null, "TOM", "1988-11-11", 10000.0);
insert into users(id, username, birthday, salary)
values(null, "JERRY", "1989-11-11", 20000.0);
insert into users(id, username, birthday, salary)
values(null, "ROWS", "1990-11-11", 30000.0);
insert into users(id, username, birthday, salary)
values(null, "LUCY", "1991-11-11", 40000.0);
```

```
insert into users(id,username,birthday,salary)
values(null,"JACK","1992-11-11",50000.0);
```

数据库连接四要素：

```
driverClass=com.mysql.jdbc.Driver
url=jdbc:mysql://192.168.200.111:3306/nginx_db
username=root
password=123456
```

步骤二:API学习

(1) 引入"resty.mysql"模块  
local mysql = require "resty.mysql"

(2) new  
创建一个MySQL连接对象，遇到错误时，db为nil，err为错误描述信息  
语法：db,err = mysql:new()

(3) connect  
尝试连接到一个MySQL服务器  
语法：ok,err=db:connect(options),options是一个参数的Lua表结构，里面包含数据库连接的相关信息  
host:服务器主机名或IP地址  
port:服务器监听端口，默认为3306  
user:登录的用户名  
password:登录密码  
database:使用的数据库名

(4) set\_timeout  
设置子请求的超时时间(ms)，包括connect方法  
语法：db:set\_timeout(time)

(5) close  
关闭当前MySQL连接并返回状态。如果成功，则返回1；如果出现任何错误，则将返回nil和错误描述。  
语法：db:close()

(6) send\_query  
异步向远程MySQL发送一个查询。如果成功则返回成功发送的字节数；如果错误，则返回nil和错误描述  
语法：bytes,err=db:send\_query(sql)

(7) read\_result  
从MySQL服务器返回结果中读取一行数据。res返回一个描述OK包或结果集包的Lua表，语法：  
res, err, errcode, sqlstate = db:read\_result()  
res, err, errcode, sqlstate = db:read\_result(rows) :rows指定返回结果集的最大值，默认为4  
如果是查询，则返回一个容纳多行的数组。每行是一个数据列的key-value对，如

```
{
```

```
{id=1,username="TOM",birthday="1988-11-11",salary=10000.0},
{id=2,username="JERRY",birthday="1989-11-11",salary=20000.0}
}
```

如果是增删改，则返回类上如下数据

```
{
 insert_id = 0,
 server_status=2,
 warning_count=1,
 affected_rows=2,
 message=nil
}
```

返回值:

res:操作的结果集

err:错误信息

errcode:MySQL的错误码，比如1064

sqlstate:返回由5个字符组成的标准SQL错误码，比如42000

### 步骤三:效果实现

```
location /{
 content_by_lua_block{
 local mysql = require "resty.mysql"
 local db = mysql:new()
 local ok,err = db:connect{
 host="192.168.200.111",
 port=3306,
 user="root",
 password="123456",
 database="nginx_db"
 }
 db:set_timeout(1000)

 db:send_query("select * from users where id =1")
 local res,err,errcode,sqlstate = db:read_result()

 ngx.say(res[1].id.."","..res[1].username.."","..res[1].birthday.."","
 ..res[1].salary)
 db:close()
 }
}
```

```
location /testMysql {
 default_type 'text/html';
 content_by_lua_block{
 local mysql = require "resty.mysql"
```

```

 local db = mysql:new()
 local ok,err = db:connect{
 host="192.168.56.2",
 port=3306,
 user="root",
 password="123456",
 database="nginx_db"
 }

 if not ok then
 ngx.say("<h1>连接数据库失败</h1>",err)
 end

 db:set_timeout(1000)

 db:send_query("select * from users where id = 2")
 local res,err,errcode,sqlstate = db:read_result()
 ngx.say(res[1].id.." " ..res[1].username.." ")
 }

```

问题:

- 1.如何获取返回数据的内容
  - 2.如何实现查询多条数据
- ```

local res,err,errcode,sqlstate = db:read_result()
for i,v in pairs(res) do
    ngx.say(v.id.." ",v.username)
end

```
- 3.如何实现数据库的增删改操作

使用lua-cjson处理查询结果

通过上述的案例学习，read_result()得到的结果res都是table类型，要想在页面上展示，就必须知道table的具体数据结构才能进行遍历获取。处理起来比较麻烦，接下来我们介绍一种简单方式cjson，使用它就可以将table类型的数据转换成json字符串，把json字符串展示在页面上即可。具体如何使用？

步骤一：引入cjson

```

local cjson = require "cjson"

```

步骤二：调用cjson的encode方法进行类型转换

```
cjson.encode(res)
```

步骤三:使用

```
location /{
    content_by_lua_block{

        local mysql = require "resty.mysql"
        local cjson = require "cjson"

        local db = mysql:new()

        local ok,err = db:connect{
            host="192.168.200.111",
            port=3306,
            user="root",
            password="123456",
            database="nginx_db"
        }
        db:set_timeout(1000)

        --db:send_query("select * from users where id = 2")
        db:send_query("select * from users")
        local res,err,errcode,sqlstate = db:read_result()
        ngx.say(cjson.encode(res))
        --[[for i,v in ipairs(res) do

            ngx.say(v.id.." ", "..v.username.." ", "..v.birthday.." ", "..v.salary)
            end --]]
        db:close()
    }
}
```

```
[{"birthday": "1988-11-11", "username": "TOM", "id": 1, "salary": 10000}, {"birthday": "1989-11-11", "username": "JERRY", "id": 2, "salary": 20000}, {"birthday": "1990-11-11", "username": "ROWS", "id": 3, "salary": 30000}, {"birthday": "1991-11-11", "username": "LUCY", "id": 4, "salary": 40000}, {"birthday": "1992-11-11", "username": "用户", "id": 5, "salary": 50000}]
```

```
# http块
init_by_lua_block{
    cjson = require "cjson"
}
```

lua-resty-mysql实现数据库的增删改

优化send_query和read_result

本方法是send_query和read_result组合的快捷方法。

语法：

```
res, err, errcode, sqlstate = db:query(sql[,rows])
```

有了该API，上面的代码我们就可以进行对应的优化，如下：

```
location /testMysqlUpdate {
    default_type 'text/html';
    content_by_lua_block{
        -- local cJSON = require "cjson"
        local mysql = require "resty.mysql"
        local db = mysql:new()
        local ok,err = db:connect{
            host="192.168.56.2",
            port=3306,
            user="root",
            password="123456",
            database="nginx_db"
        }

        if not ok then
            ngx.say("<h1>连接数据库失败</h1>",err)
        end

        db:set_timeout(1000)

        -- db:send_query("select * from users")
        -- local res,err,errcode,sqlstate =
        db:read_result()

        -- local sql = "insert into
        users(id,username,birthday,salary) values(null,'zhangsan','2020-
        11-11',32222.0)"
        -- local sql = "update users set username='lisi'
        where id = 6"
        local sql = "delete from users where id = 6"
        local res,err,errcode,sqlstate = db:query(sql)
        -- 转化为json
        ngx.say(cjson.encode(res))
        db:close()
    }
}
```

```
# 增加
{"affected_rows":1,"insert_id":6,"server_status":2,"warning_count":0}
# 修改
{"server_status":2,"warning_count":0,"message":"Rows matched: 1
Changed: 1 Warnings: 0","affected_rows":1,"insert_id":0}
# 删除
{"warning_count":0,"affected_rows":1,"insert_id":0,"server_status":2}
```

综合小案例

使用ngx_lua模块完成Redis缓存预热。

分析：

- (1) 先得有一张表(users)
- (2) 浏览器输入如下地址

```
http://191.168.200.133?username=TOM
```

- (3) 从表中查询出符合条件的记录，此时获取的结果为table类型
- (4) 使用cjson将table数据转换成json字符串
- (5) 将查询的结果数据存入Redis中

```
init_by_lua_block{
    redis = require "resty.redis"
    mysql = require "resty.mysql"
    cjson = require "cjson"
}

location /mysqlAndRedisCache{
    default_type 'text/html';
    content_by_lua_block{
        local param = ngx.req.get_uri_args()["username"]
        local db = mysql:new()
        local ok,err = db:connect{
            host = "192.168.56.2",
            port = 3306,
            user = "root",
            password = "123456",
            database = "nginx_db"
```

```

    }
    if not ok then
        ngx.say("<h1>连接数据库出错</h1>",err)
        return
    end

    db:set_timeout(1000)
    local sql = ""
    if not param then
        sql = "select * from users"
    else
        -- 查询时插入单引号
        sql = "select * from users where username
=" .. "'" .. param .. "'"
    end

    local res,err,errcode,sqlstate = db:query(sql)
    if not res then
        ngx.say("<h1>查询失败 无法存入Redis</h1>")
        return
    end

    local rd = redis:new()
    ok,err = rd:connect("192.168.56.2",6379)
    if not ok then
        ngx.say("<h1>Redis连接出错</h1>")
        return
    end
    rd:set_timeout(1000)

    for i,v in ipairs(res) do
        rd:set("user" .. v.username,cjson.encode(v))
    end
    ngx.say("<h1>SUCCESS</h1>")
    rd:close()
    db:close()
}
}

```

所有配置文件

```

#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;

events {
    worker_connections 1024;
}

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;
    # 初始化全局变量
    init_by_lua_block{
        cJSON = require "cjson"
        redis = require "resty.redis"
        mysql = require "resty.mysql"
    }
    server {
        listen 80;
        server_name localhost;

        charset utf-8;

        #access_log logs/host.access.log main;

        location /getByGender {
            default_type text/html;
            # 声明一个变量
            set_by_lua $param "
```

```

        -- 获取请求url上面对应参数的值
        local uri_args = ngx.req.get_uri_args()
        local name = uri_args['name']
        local gender = uri_args['gender']
        if gender == '1' then
            return name..'先生'
        elseif gender == '0' then
            return name..'女士'
        else
            return name;
        end
    ";
    return 200 $param;
}

location /getJSON {
    default_type 'text/html';
    set_by_lua $name "
        return 'Hello'
    ";
    return 200 $name;
}

location /lua {
    default_type text/html;
    content_by_lua 'ngx.say("<h1>Hello OpenResty</h1>")';
}

location /testRedis {
    default_type text/html;
    content_by_lua_block{
        local redis = require "resty.redis"
        local redisObj = redis:new()
        -- 超时时间单位毫秒
        redisObj:set_timeout(1000)
        local ok,err =
redisObj:connect("192.168.56.2",6379)
        if not ok then
            ngx.say("<h1>redis连接失败!!! </h1>",err)
            return
        end

        ok,err = redisObj:set("username","ROSE")

        if not ok then
            ngx.say("<h1>存入数据失败</h1>",err)
            return
        end
    }
}

```

```

        end

        local res,err = redisObj:get("username")
        ngx.say(res)

        redisObj:close()
    }
}

location /testMysql {
    default_type 'text/html';
    content_by_lua_block{
        -- local cJSON = require "cjson"
        local mysql = require "resty.mysql"
        local db = mysql:new()
        local ok,err = db:connect{
            host="192.168.56.2",
            port=3306,
            user="root",
            password="123456",
            database="nginx_db"
        }

        if not ok then
            ngx.say("<h1>连接数据库失败</h1>",err)
            end

            db:set_timeout(1000)

            db:send_query("select * from users")
            local res,err,errcode,sqlstate = db:read_result()
            --[[
            for i,v in pairs(res) do
                ngx.say(v.id.." "..v.username)
            end
            --]]
            -- 转化为json
            ngx.say(cjson.encode(res))
            db:close()
        }
    }

    location /testMysqlUpdate {
        default_type 'text/html';
        content_by_lua_block{
            -- local cJSON = require "cjson"
            local mysql = require "resty.mysql"

```

```

        local db = mysql:new()
        local ok,err = db:connect{
            host="192.168.56.2",
            port=3306,
            user="root",
            password="123456",
            database="nginx_db"
        }

        if not ok then
            ngx.say("<h1>连接数据库失败</h1>",err)
        end

        db:set_timeout(1000)

        -- db:send_query("select * from users")
        -- local res,err,errcode,sqlstate =
db:read_result()

        -- local sql = "insert into
users(id,username,birthday,salary) values(null,'zhangsan','2020-
11-11',32222.0)"
        -- local sql = "update users set username='lisi'
where id = 6"

        local sql = "delete from users where id = 6"
        local res,err,errcode,sqlstate = db:query(sql)
        -- 转化为json
        ngx.say(cjson.encode(res))
        db:close()
    }
}

location /mysqlAndRedisCache{
    default_type 'text/html';
    content_by_lua_block{
        local param = ngx.req.get_uri_args()["username"]
        local db = mysql:new()
        local ok,err = db:connect{
            host = "192.168.56.2",
            port = 3306,
            user = "root",
            password = "123456",
            database = "nginx_db"
        }
        if not ok then
            ngx.say("<h1>连接数据库出错</h1>",err)
            return
        end
    }
}

```

```

        end

        db:set_timeout(1000)
        local sql = ""
        if not param then
            sql = "select * from users"
        else
            -- 查询时插入单引号
            sql = "select * from users where username
=" .. "'" .. param .. "'"
        end

        local res,err,errcode,sqlstate = db:query(sql)
        if not res then
            ngx.say("<h1>查询失败 无法存入Redis</h1>")
            return
        end

        local rd = redis:new()
        ok,err = rd:connect("192.168.56.2",6379)
        if not ok then
            ngx.say("<h1>Redis连接出错</h1>")
            return
        end
        rd:set_timeout(1000)

        for i,v in ipairs(res) do
            rd:set("user" .. v.username,cjson.encode(v))
        end
        ngx.say("<h1>SUCCESS</h1>")
        rd:close()
        db:close()
    }
}

#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;
#    }
#}

}
```