Nginx 相关

Nginx 命令

- nginx 启动

```
指令:  nginx程序 -c nginx配置文件
如:  /usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf
```

- nginx重启

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

./nginx -s reload
```

- nginx停止

```
./nginx -s stop
./nginx -s quit  # quit是优雅的关闭，Nginx在退出前已经接受的请求会被完成响应，而stop是直接停止

系统杀死nginx:
pkill -9 nginx
```

- 重新打开日志

```
./nginx -s reopen
```

- nginx检查配置文件

```
检测配置文件是否正确,在启动之前检测哈，否者错误启动会很麻烦
/usr/local/nginx/sbin/nginx -t

/usr/local/nginx/sbin/nginx -t /usr/local/nginx/conf/nginx.conf
```

Nginx信号控制

nginx 支持2种进程模式：Single和Master-Worker；Single是单进程，一般不适用，Master-Worker是主进程和工作进程模型运行，主进程对工作进程管理。

```
[ops@10-222-12-98 sbin]$ sudo  ./nginx
[ops@10-222-12-98 sbin]$ ps aux|grep nginx
root      13096  0.0  0.0  20536   612 ?        Ss   18:30   0:00 nginx: master process ./nginx
nobody    13097  0.0  0.0  20980  1568 ?        S    18:30   0:00 nginx: worker process
ops       13219  0.0  0.0 112708   980 pts/0    R+   19:06   0:00 grep --color=auto nginx
```

nginx允许我们通过信号控制主进程，用信号的方式可以达到不影响现有连接的目的

信号类型

```
INT, TERM     快速关闭信号
QUIT     从容关闭信号
HUP      从容重启信号，一般用于修改配置文件后，重启
USR1     重读日志，一般用于日志切割
USR2     平滑升级信号
WINCH    从容关闭旧进程

具体语法:
    kill -信号选项   nginx的主进程号
如:  kill -INT 26661
```

nginx 平滑升级

如果我们在不影响现有业务服务的情况下，不想停止nginx开启的服务，但又想对nginx进行升级，这时候就需要使用平滑升级。

平滑升级步骤

- 下载高版本nginx： wget -P /opt/softwares http://nginx.org/download/nginx-1.13.2.tar.gz
- 执行指令

```
./configure

make # 但是不要make install (make后会生成一个obj文件，里面的nginx就是高版本的nginx，我们需要覆盖

 # 先备份低版本的nginx
cp /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.old

 # 覆盖低版本的nginx
sudo cp -rfp /opt/softwares/nginx-1.13.2/objs/nginx /usr/local/nginx/sbin/nginx

 # 平滑升级
kill -USR2 'cat /usr/local/nginx/logs/nginx.pid'
```

Nginx 的配置文件

位置：/usr/local/nginx/conf/nginx.conf

配置文件详解：

```
#定义Nginx运行的用户和用户组,定义运行的权限
user www www;

#nginx进程数，建议设置为等于CPU总核心数。
worker_processes 8;
```

```
#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]
error_log /usr/local/nginx/logs/error.log info;

#进程pid文件
pid /usr/local/nginx/logs/nginx.pid;

#指定进程可以打开的最大描述符：数目
#工作模式与连接数上限
#这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。
#现在在linux 2.6内核下开启文件打开数为65535，worker_rlimit_nofile就相应应该填写65535。
#这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。
worker_rlimit_nofile 65535;


events
{
    #参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型
    #是Linux 2.6以上版本内核中的高性能网络I/O模型，linux建议epoll，如果跑在FreeBSD上面，就用kqueue模型。
    #补充说明：
    #与apache相类，nginx针对不同的操作系统，有不同的事件模型
    #A）标准事件模型
    #Select、poll属于标准事件模型，如果当前系统不存在更有效的方法，nginx会选择select或poll
    #B）高效事件模型
    #Kqueue：使用于FreeBSD 4.1+，OpenBSD 2.9+，NetBSD 2.0 和 MacOS X.使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。
    #Epoll：使用于Linux内核2.6版本及以后的系统。
    #/dev/poll：使用于Solaris 7 11/99+，HP/UX 11.22+ (eventport)，IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。
    #Eventport：使用于Solaris 10。 为了防止出现内核崩溃的问题， 有必要安装安全补丁。
    use epoll;

    #单个进程最大连接数（最大连接数=连接数*进程数）
    #根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把cpu跑到100%就行。每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为。
    worker_connections 65535;

    #keepalive超时时间。
    keepalive_timeout 60;

    #客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。
    #分页大小可以用命令getconf PAGESIZE 取得。
    #[root@web001 ~]# getconf PAGESIZE
    #4096
    #但也有client_header_buffer_size超过4k的情况，但是client_header_buffer_size该值必须设置为“系统分页大小”的整倍数。
    client_header_buffer_size 4k;

    #这个将为打开文件指定缓存，默认是没有启用的，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没被请求后删除缓存。
```

```
    open_file_cache max=65535 inactive=60s;

    #这个是指多长时间检查一次缓存的有效信息。
    #语法:open_file_cache_valid time 默认值:open_file_cache_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open_file_cache中缓存项目的有效信息.
    open_file_cache_valid 80s;

    #open_file_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，它将被移除。
    #语法:open_file_cache_min_uses number 默认值:open_file_cache_min_uses 1 使用字段:http, server, location  这个指令指定了在open_file_cache指令无效的参数中一定的时间范围内可以使用的最小文件数，如果使用更大的值,文件描述符在cache中总是打开状态.
    open_file_cache_min_uses 1;

    #语法:open_file_cache_errors on | off 默认值:open_file_cache_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误.
    open_file_cache_errors on;
}



#设定http服务器，利用它的反向代理功能提供负载均衡支持
http
{
    #文件扩展名与文件类型映射表
    include mime.types;

    #默认文件类型
    default_type application/octet-stream;

    #默认编码
    #charset utf-8;

    #服务器名字的hash表大小
    #保存服务器名字的hash表是由指令server_names_hash_max_size 和server_names_hash_bucket_size所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.
    server_names_hash_bucket_size 128;

    #客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。
    client_header_buffer_size 32k;

    #客户请求头缓冲大小。nginx默认会用client_header_buffer_size这个buffer来读取header值，如果header过大，它会使用large_client_header_buffers来读取。
    large_client_header_buffers 4 64k;

    #设定通过nginx上传文件的大小
    client_max_body_size 8m;
```

```
    #开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。
    #sendfile指令指定 nginx 是否调用sendfile 函数（zero copy 方式）来输出文件，对于普通应用，必须设为on。如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络IO处理速度，降低系统uptime。
    sendfile on;

    #开启目录列表访问，合适下载服务器，默认关闭。
    autoindex on;

    #此选项允许或禁止使用socke的TCP_CORK的选项，此选项仅在使用sendfile的时候使用
    tcp_nopush on;

    tcp_nodelay on;

    #长连接超时时间，单位是秒
    keepalive_timeout 120;

    #FastCGI相关参数是为了改善网站的性能：减少资源占用，提高访问速度。下面参数看字面意思都能理解。
    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
    fastcgi_buffer_size 64k;
    fastcgi_buffers 4 64k;
    fastcgi_busy_buffers_size 128k;
    fastcgi_temp_file_write_size 128k;

    #gzip模块设置
    gzip on; #开启gzip压缩输出
    gzip_min_length 1k;    #最小压缩文件大小
    gzip_buffers 4 16k;    #压缩缓冲区
    gzip_http_version 1.0;    #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）
    gzip_comp_level 2;    #压缩等级
    gzip_types text/plain application/x-javascript text/css application/xml;    #压缩类型，默认就已经包含textml，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。
    gzip_vary on;

    #开启限制IP连接数的时候需要使用
    #limit_zone crawler $binary_remote_addr 10m;



    #负载均衡配置
    upstream piao.jd.com {

        #upstream的负载均衡，weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。
        server 192.168.80.121:80 weight=3;
        server 192.168.80.122:80 weight=2;
        server 192.168.80.123:80 weight=3;

        #nginx的upstream目前支持4种方式的分配
        #1、轮询（默认）
```

```
        #每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。
        #2、weight
        #指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。
        #例如:
        #upstream bakend {
        #    server 192.168.0.14 weight=10;
        #    server 192.168.0.15 weight=10;
        #}
        #2、ip_hash
        #每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。
        #例如:
        #upstream bakend {
        #    ip_hash;
        #    server 192.168.0.14:88;
        #    server 192.168.0.15:80;
        #}
        #3、fair（第三方）
        #按后端服务器的响应时间来分配请求，响应时间短的优先分配。
        #upstream backend {
        #    server server1;
        #    server server2;
        #    fair;
        #}
        #4、url_hash（第三方）
        #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，后端服务器为缓存时比较有效。
        #例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method是使用的
hash算法
        #upstream backend {
        #    server squid1:3128;
        #    server squid2:3128;
        #    hash $request_uri;
        #    hash_method crc32;
        #}

        #tips:
        #upstream bakend{#定义负载均衡设备的Ip及设备状态}{
        #    ip_hash;
        #    server 127.0.0.1:9090 down;
        #    server 127.0.0.1:8080 weight=2;
        #    server 127.0.0.1:6060;
        #    server 127.0.0.1:7070 backup;
        #}
        #在需要使用负载均衡的server中增加 proxy_pass http://bakend/;

        #每个设备的状态设置为:
        #1.down表示单前的server暂时不参与负载
        #2.weight为weight越大，负载的权重就越大。
        #3.max_fails：允许请求失败的次数默认为1.当超过最大次数时，返回proxy_next_upstream模块定义的错
误
        #4.fail_timeout:max_fails次失败后，暂停的时间。
        #5.backup：  其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

        #nginx支持同时设置多组的负载均衡，用来给不用的server来使用。
```

```
        #client_body_in_file_only设置为On 可以讲client post过来的数据记录到文件中用来做debug
        #client_body_temp_path设置记录文件的目录 可以设置最多3层目录
        #location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡
    }



    #虚拟主机的配置
    server
    {
        #监听端口
        listen 80;

        #域名可以有多个，用空格隔开
        server_name www.jd.com jd.com;
        index index.html index.htm index.php;
        root /data/www/jd;

        #对******进行负载均衡
        location ~ .*.(php|php5)?$
        {
            fastcgi_pass 127.0.0.1:9000;
            fastcgi_index index.php;
            include fastcgi.conf;
        }

        #图片缓存时间设置
        location ~ .*.(gif|jpg|jpeg|png|bmp|swf)$
        {
            expires 10d;
        }

        #JS和CSS缓存时间设置
        location ~ .*.(js|css)?$
        {
            expires 1h;
        }

        #日志格式设定
        #$remote_addr与$http_x_forwarded_for用以记录客户端的ip地址;
        #$remote_user: 用来记录客户端用户名称;
        #$time_local:  用来记录访问时间与时区;
        #$request:  用来记录请求的url与http协议;
        #$status:  用来记录请求状态；成功是200,
        #$body_bytes_sent : 记录发送给客户端文件主体内容大小;
        #$http_referer: 用来记录从那个页面链接访问过来的;
        #$http_user_agent: 记录客户浏览器的相关信息;
        #通常web服务器放在反向代理的后面，这样就不能获取到客户的IP地址了，通过$remote_add拿到的IP地址是反向代理服务器的iP地址。
        #反向代理服务器在转发请求的http头信息中，可以增加x_forwarded_for信息，用以记录原有客户端的IP地址和原来客户端的请求的服务器地址。
        log_format access '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
```

```
        '"$http_user_agent" $http_x_forwarded_for';

        #定义本虚拟主机的访问日志
        access_log  /usr/local/nginx/logs/host.access.log  main;
        access_log  /usr/local/nginx/logs/host.access.404.log  log404;

        #对 "/" 启用反向代理
        location / {
            proxy_pass http://127.0.0.1:88;
            proxy_redirect off;
            proxy_set_header X-Real-IP $remote_addr;

            #后端的Web服务器可以通过X-Forwarded-For获取用户真实IP
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

            #以下是一些反向代理的配置，可选。
            proxy_set_header Host $host;

            #允许客户端请求的最大单文件字节数
            client_max_body_size 10m;

            #缓冲区代理缓冲用户端请求的最大字节数，
            #如果把它设置为比较大的数值，例如256k，那么，无论使用firefox还是IE浏览器，来提交任意小于256k的图片，都很正常。如果注释该指令，使用默认的client_body_buffer_size设置，也就是操作系统页面大小的两倍，8k或者16k，问题就出现了。
            #无论使用firefox4.0还是IE8.0，提交一个比较大，200k左右的图片，都返回500 Internal Server Error错误
            client_body_buffer_size 128k;

            #表示使nginx阻止HTTP应答代码为400或者更高的应答。
            proxy_intercept_errors on;

            #后端服务器连接的超时时间_发起握手等候响应超时时间
            #nginx跟后端服务器连接超时时间(代理连接超时)
            proxy_connect_timeout 90;

            #后端服务器数据回传时间(代理发送超时)
            #后端服务器数据回传时间_就是在规定时间之内后端服务器必须传完所有的数据
            proxy_send_timeout 90;

            #连接成功后，后端服务器响应时间(代理接收超时)
            #连接成功后_等候后端服务器响应时间_其实已经进入后端的排队之中等候处理（也可以说是后端服务器处理请求的时间)
            proxy_read_timeout 90;

            #设置代理服务器（nginx）保存用户头信息的缓冲区大小
            #设置从被代理服务器读取的第一部分应答的缓冲区大小，通常情况下这部分应答中包含一个小的应答头，默认情况下这个值的大小为指令proxy_buffers中指定的一个缓冲区的大小，不过可以将其设置为更小
            proxy_buffer_size 4k;

            #proxy_buffers缓冲区，网页平均在32k以下的设置
            #设置用于读取应答（来自被代理服务器）的缓冲区数目和大小，默认情况也为分页大小，根据操作系统的不同可能是4k或者8k
```

```
            proxy_buffers 4 32k;

            #高负荷下缓冲大小（proxy_buffers*2）
            proxy_busy_buffers_size 64k;

            #设置在写入proxy_temp_path时数据的大小，预防一个工作进程在传递文件时阻塞太长
            #设定缓存文件夹大小，大于这个值，将从upstream服务器传
            proxy_temp_file_write_size 64k;
        }


        #设定查看Nginx状态的地址
        location /NginxStatus {
            stub_status on;
            access_log on;
            auth_basic "NginxStatus";
            auth_basic_user_file confpasswd;
            #htpasswd文件的内容可以用apache提供的htpasswd工具来产生。
        }

        #本地动静分离反向代理配置
        #所有jsp的页面均交由tomcat或resin处理
        location ~ .(jsp|jspx|do)?$ {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass http://127.0.0.1:8080;
        }

        #所有静态文件由nginx直接读取不经过tomcat或resin
        location ~ .*.(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|
        pdf|xls|mp3|wma)$
        {
            expires 15d;
        }

        location ~ .*.(js|css)?$
        {
            expires 1h;
        }
    }
}
```

配置conf文件

使用notepad++ 的插件功能方便的连接linux服务器，修改配置文件，所谓的虚拟主机是通过在nginx的配置文件中，添加多个server模块，分别可以使用不同的server_name来区分不同的目标虚拟主机服务（或者端口）。

nginx.conf配置文件

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


    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;

    #gzip  on;
        #虚拟主机1
    server {
        listen       80;
        server_name  www.server1.com;
        charset utf-8;

        location / {
            root   html/server1;
            index  index.html index.htm;
        }
    }

        #虚拟主机2
        server {
        listen       80;
        server_name  www.server2.com;  # 在对应的域名解析中对应一个目标ip，但是域名不同
        charset utf-8;

        location / {
            root   html/server2;
            index  index.html index.htm;
        }
    }
}
```

同时修改本地host文件，映射文件

```
47.95.217.144   www.server1.com
47.95.217.144   www.server2.com
```

重启nginx，同时修改html/index.html,分别映射到不同的虚拟服务器中

Nginx日志以及切割

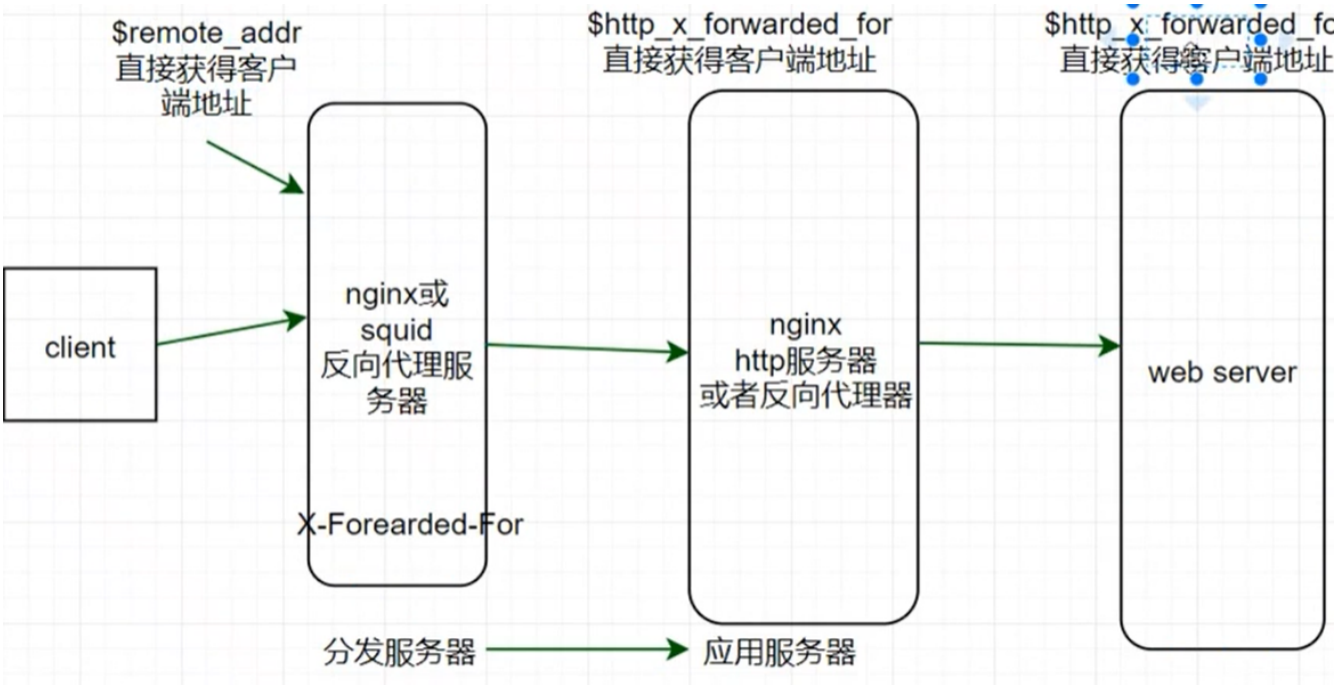
在http模块（表示此时ngnix作为http服务器）中的日志设置代码如下：

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

相应的main后面跟的是配置的变量：

```
参数                        说明                                          示例
$remote_addr                客户端地址                                    211.28.65.253
$remote_user                客户端用户名称                                --
$time_local                 访问时间和时区                                18/Jul/2012:17:00:01 +0800
$request                    请求的URI和HTTP协议                           "GET /article-10000.html HTTP/1.1"
$http_host                  请求地址，即浏览器中你输入的地址（IP或域名）      www.wang.com 192.168.100.100
$status                     HTTP请求状态                                  200
$upstream_status            upstream状态                                  200
$body_bytes_sent            发送给客户端文件内容大小                        1547
$http_referer               url跳转来源                                   https://www.baidu.com/
$http_user_agent            用户终端浏览器等信息                           "Mozilla/4.0 (compatible; MSIE 8.0;
Windows NT 5.1; Trident/4.0; SV1; GTB7.0; .NET4.0C;
$ssl_protocol               SSL协议版本                                   TLSv1
$ssl_cipher                 交换数据中的算法                               RC4-SHA
$upstream_addr              后台upstream的地址，即真正提供服务的主机地址     10.10.10.100:80
$request_time               整个请求的总时间                               0.205
$upstream_response_time     请求过程中，upstream响应时间                    0.002
$http_x_forwarded_for       是反向代理服务器转发客户端地址的参数
```

解释http_x_forward_for参数：



Nginx的location详解

- 语法规则：location [=|~|~*|^~] /uri/ {...}

  构成：

  指令 前缀 uri

  location [=|~|~*|^~] /uri

- location区分普通匹配和正则匹配：
  - 用前缀 "~" 和"~*"修饰的为正则匹配

    - "~*" 前缀表示不区分大小写的正则匹配
    - "~" 前缀表示区分大小写的正则匹配
    - 除上面修饰的前缀 ("="和"^~",或没有前缀修饰） 都为普通匹配
    - = 前缀表示精确匹配
    - ^~ 前缀表示uri以某个常规字符串开头，可以理解为url的普通匹配
- 在多个location情况下，是按照什么原则进行匹配?

```
普通匹配优先级高于正则匹配
```

- 匹配原则

  普通匹配 最大前缀匹配原则，如下:

```
server{
    location /prefix/{
        #规则A
    }
    location /prefix/mid/{
        #规则B
    }
}

如果请求为：/prefix/mid/t.html；此请求匹配到的是规则B
```

  正则匹配 为顺序匹配，如下:

```
server{
    location ~ \.(gif|jpg|png|js|css)$ {
        # 规则C
    }
    location ~* \.png$ {
        # 规则D
    }
}

请求http://localhost/1.png,匹配的规则是C，因为C在最前
```

- 如果location有普通匹配也有正则匹配，那匹配的原则为

```
1、location = /url   # =开头表示精确匹配，只有完全匹配上才能生效
2、location ^~ /uri  # ^~开头对url路径进行前缀匹配，并且在正则之前
3、location ~ pattern  # ~开头表示区分大小写的正则匹配
4、location ~ pattern  # ~开头不表示区分大小写的正则匹配
5、location /uri  # 不带任何修饰符，也表示前缀匹配，但是在正则之后
6、location /  # 通用匹配，任何未匹配到的请求都会到

例子：有如下的匹配规则（修改：default_type  text/html，否则下列会执行下载，因为默认映射成文件）;

location = / {
    return 200 '规则A'; #200是状态，后面是内容
}
```

```
location = /login {
    return 200 '规则B';
}
location ^~ /static/ {
    return 200 '规则C';
}
location ~ \.(gif|jpg|png|js|css)$ {
    return 200 '规则D'
}
location ~* \.js$ {
    return 200 '规则E';
}
location ~ \.html$ {
        root html/server1;  # 可直接访问静态文件
}
location / {
    return 200 '规则F';
}
```

Nginx的echo模块安装

查看nginx已安装的模块

```
./nginx -V
nginx version: nginx/1.13.2
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-39) (GCC)
configure arguments: --prefix=/usr/local/nginx
```

安装步骤

- 下载需要的echo模块

  ```
  wget -P /opt/softwares/nginx-tools/ https://github.com/openresty/echo-nginx-module/archive/v0.61.tar.gz

  指定下载后，解压
  tar -zxvf v0.61.tar.gz
  ```

- 重新编译nginx，也就是说重新解压之前的压缩包，重新编译安装echo-nginx模块

  进入nginx源文件，重新编译,需要删除之前的解压文件

```
[root@MyHost echo-nginx-module-0.61]# cd /usr/local/nginx/
[root@MyHost nginx]# ls
client_body_temp  conf  fastcgi_temp  html  logs  proxy_temp  sbin  scgi_temp
uwsgi_temp
[root@MyHost nginx]# mkdir moudles
创建一个目录方便以后的模块文件存储，在编译的时候指向这里的模块
[root@MyHost softwares]# cp -rf nginx-tools/echo-nginx-module-0.61/
/usr/local/nginx/moudles/


./configure --add-module=/usr/local/nginx/moudles/echo-nginx-module-0.61  # 安装echo模块

make # 开始编译，但别安装 （make install 会直接覆盖）
```

- 平滑升级nginx

```
注意先备份老的nginx
mv /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.old

复制新nginx
[root@MyHost nginx-1.13.2]# cp /opt/softwares/nginx-1.13.2/objs/nginx
/usr/local/nginx/sbin/

在源代码中平滑升级
[root@MyHost nginx-1.13.2]# make upgrade
[root@MyHost nginx-1.13.2]# make clean
```

- 试试echo模块

  在nginx.conf中修改location参数

```
    server {
        listen       80;
        server_name  localhost;
        charset utf-8;

        #location / {
        #    root   html/server1;
        #    index  index.html index.htm;
        #}

                location /echo-hello {
                        echo "hello-world";
                }
    }

浏览器访问：http://47.95.217.144/echo-hello；会出现hello-word字样
```

echo模块的简单使用

- 内部路由跳转

```
location /a {
        echo_exec /b;
}
location /b {
        echo "This is B";
}
location /c {
        set $foo 'hello world'; # 自定义变量
        echo "$request_uri"; # 显示nginx的全局变量内容
        echo $foo;
}
```

我们在浏览器中访问：http://47.95.217.144/a；会出现b的样式

Nginx内部变量

```
$arg_PARAMETER 客户端GET请求中PARAMETER 字段的值

$args 客户端请求中的参数

$binary_remote_addr 远程地址的二进制表示

$body_bytes_sent 已发送的消息体字节数

$content_length HTTP请求信息中content-length的字段

$content_type 请求信息中content-type字段

$cookie_COOKIE 客户端请求中COOKIE头域的值

$document_root 针对当前请求的根路径设置值
```

$ document_uri　　与$uri相同

$host 请求信息中的host头域，如果请求中没有Host行，则等于设置的服务器名

$http_HEADER HTTP请求信息里的HEADER地段

$ http_host 与$host相同，但是如果请求信息中没有host行，则可能不同客户端cookie信息

$http_cookie 客户端cookie信息

$http_referer 客户端是从哪一个地址跳转过来的

$http_user_agent 客户端代理信息，也就是你客户端浏览器

$http_via 最后一个访问服务器的IP

$http_x_forwarded_for 相当于访问网路访问的路径

$is_args 如果有args的值，则等于"?"，否则为空

$limit_rate 对连接速率的限制

$nginx_version 当前Nginx的版本

$pid 当前Nginx服务器的进程的进程ID

$ query_string 与$args相同

$remote_addr 客户端IP地址

$remote_port 客户端的端口

$remote_user 客户端的用户名，用于 auth basic module验证

$request 客户端请求

$request_body 客户端发送的报文体

$request_body_file 发送后端服务器的本地临时缓存文件的名称

$request_filename 当前请求的文件路径名，由root或alias指令与URI请求生成

$request_method 请求后端数据的方法，例如"GET",“POST”

$request_uri 请求的URI，带参数，不包含主机名

$ scheme 所用的协议，如http或者HTTPS，比如
rewrite^(.+)$scheme://mysite.namescheme://mysite.namescheme://mysite.nameredirect

$sent_http_cache_control 对应http请求头中的Cache-Control，需要打开chrome浏览器，右键检查，选中network，点中其中一个请求的资源

$sent_http_connection 对应http请求中的Connection

$sent_http_content_type 对应http请求中的Content-Type

$sent_last_modified 对应请求中的Last-Modified

$server_addr 服务端的地址

$server_port 请求到达服务器端口号

$server_protocol 请求协议的版本号，HTTP1.0/HTTP1.1

$uri 请求的不带请求参数的URI，可能和最初的值有不同，比如经过重定向之类的