安装

安装依赖

安装 nginx 之前，确保系统已经安装 gcc、openssl-devel、pcre-devel 和 zlib-devel 软件库

- gcc 可以通过光盘直接选择安装
- openssl-devel、zlib-devel 可以通过光盘直接选择安装，https 时使用
- pcre-devel 安装 pcre 库是为了使 nginx 支持 HTTP Rewrite 模块

下载

nginx 下载

编译安装

通过上面的下载页下载最新的稳定版

```
#wget http://nginx.org/download/nginx-1.8.0.tar.gz
#tar xzvf nginx-1.8.0.tar.gz
#cd nginx-1.8.0
#./configure --prefix=/opt/X_nginx/nginx --with-http_ssl_module
#make && sudo make install
复制代码
```

> - --prefix=/opt/X_nginx/nginx 安装目录
> - --with-http_ssl_module 添加 https 支持

编译时将 ssl 模块静态编译

```
./configure  --prefix=/opt/X_nginx/nginx \
             --with-openssl=../openssl-1.0.2l \
             --with-zlib=../zlib-1.2.11 \
             --with-pcre=../pcre-8.41 \
             --with-http_ssl_module
复制代码
```

nginx 服务架构

模块化结构

> nginx 服务器的开发`完全`遵循模块化设计思想

模块化开发

1. 单一职责原则，一个模块只负责一个功能
2. 将程序分解，自顶向下，逐步求精
3. 高内聚，低耦合

nginx 的模块化结构

- 核心模块：nginx 最基本最核心的服务，如进程管理、权限控制、日志记录；
- 标准 HTTP 模块：nginx 服务器的标准 HTTP 功能；
- 可选 HTTP 模块：处理特殊的 HTTP 请求
- 邮件服务模块：邮件服务
- 第三方模块：作为扩展，完成特殊功能

nginx 的模块清单

- 核心模块
  - ngx_core
  - ngx_errlog
  - ngx_conf
  - ngx_events
  - ngx_event_core
  - ngx_epll
  - ngx_regex
- 标准 HTTP 模块
  - ngx_http
  - ngx_http_core #配置端口，URI 分析，服务器相应错误处理，别名控制 (alias) 等
  - ngx_http_log #自定义 access 日志
  - ngx_http_upstream #定义一组服务器，可以接受来自 proxy, Fastcgi,Memcache 的重定向；主要用作负载均衡
  - ngx_http_static
  - ngx_http_autoindex #自动生成目录列表
  - ngx_http_index #处理以`/`结尾的请求，如果没有找到 index 页，则看是否开启了`random_index`；如开启，则用之，否则用 autoindex
  - ngx_http_auth_basic #基于 http 的身份认证 (auth_basic)
  - ngx_http_access #基于 IP 地址的访问控制 (deny,allow)
  - ngx_http_limit_conn #限制来自客户端的连接的响应和处理速率
  - ngx_http_limit_req #限制来自客户端的请求的响应和处理速率
  - ngx_http_geo
  - ngx_http_map #创建任意的键值对变量
  - ngx_http_split_clients
  - ngx_http_referer #过滤 HTTP 头中 Referer 为空的对象
  - ngx_http_rewrite #通过正则表达式重定向请求
  - ngx_http_proxy
  - ngx_http_fastcgi #支持 fastcgi
  - ngx_http_uwsgi
  - ngx_http_scgi
  - ngx_http_memcached
  - ngx_http_empty_gif #从内存创建一个 1×1 的透明 gif 图片，可以快速调用
  - ngx_http_browser #解析 http 请求头部的 User-Agent 值

    - ngx_http_charset #指定网页编码
    - ngx_http_upstream_ip_hash
    - ngx_http_upstream_least_conn
    - ngx_http_upstream_keepalive
    - ngx_http_write_filter
    - ngx_http_header_filter
    - ngx_http_chunked_filter
    - ngx_http_range_header
    - ngx_http_gzip_filter
    - ngx_http_postpone_filter
    - ngx_http_ssi_filter
    - ngx_http_charset_filter
    - ngx_http_userid_filter
    - ngx_http_headers_filter #设置 http 响应头
    - ngx_http_copy_filter
    - ngx_http_range_body_filter
    - ngx_http_not_modified_filter
- 可选 HTTP 模块
    - ngx_http_addition #在响应请求的页面开始或者结尾添加文本信息
    - ngx_http_degradation #在低内存的情况下允许服务器返回 444 或者 204 错误
    - ngx_http_perl
    - ngx_http_flv #支持将 Flash 多媒体信息按照流文件传输，可以根据客户端指定的开始位置返回 Flash
    - ngx_http_geoip #支持解析基于 GeoIP 数据库的客户端请求
    - ngx_google_perftools
    - ngx_http_gzip #gzip 压缩请求的响应
    - ngx_http_gzip_static #搜索并使用预压缩的以.gz 为后缀的文件代替一般文件响应客户端请求
    - ngx_http_image_filter #支持改变 png，jpeg，gif 图片的尺寸和旋转方向
    - ngx_http_mp4 #支持.mp4,.m4v,.m4a 等多媒体信息按照流文件传输，常与 ngx_http_flv 一起使用
    - ngx_http_random_index #当收到 / 结尾的请求时，在指定目录下随机选择一个文件作为 index
    - ngx_http_secure_link #支持对请求链接的有效性检查
    - ngx_http_ssl #支持 https
    - ngx_http_stub_status
    - ngx_http_sub_module #使用指定的字符串替换响应中的信息
    - ngx_http_dav #支持 HTTP 和 WebDAV 协议中的 PUT/DELETE/MKCOL/COPY/MOVE 方法
    - ngx_http_xslt #将 XML 响应信息使用 XSLT 进行转换
- 邮件服务模块
    - ngx_mail_core
    - ngx_mail_pop3
    - ngx_mail_imap
    - ngx_mail_smtp
    - ngx_mail_auth_http
    - ngx_mail_proxy
    - ngx_mail_ssl
- 第三方模块
    - echo-nginx-module #支持在 nginx 配置文件中使用 echo/sleep/time/exec 等类 Shell 命令
    - memc-nginx-module
    - rds-json-nginx-module #使 nginx 支持 json 数据的处理

    - lua-nginx-module

nginx 的 web 请求处理机制

作为服务器软件，必须具备并行处理多个客户端的请求的能力， 工作方式主要以下 3 种：

- 多进程 (Apache)
    - 优点：设计和实现简单；子进程独立
    - 缺点：生成一个子进程要内存复制，在资源和时间上造成额外开销
- 多线程 (IIS)
    - 优点：开销小
    - 缺点：开发者自己要对内存进行管理；线程之间会相互影响
- 异步方式 (nginx)

经常说道异步非阻塞这个概念， 包含两层含义：

通信模式： + 同步：发送方发送完请求后，等待并接受对方的回应后，再发送下个请求 + 异步：发送方发送完请求后，不必等待，直接发送下个请求

nginx 配置文件实例

```
#定义 nginx 运行的用户和用户组
user www www;

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

#nginx 默认没有开启利用多核 CPU，通过增加 worker_cpu_affinity 配置参数来充分利用多核 CPU 以下是 8 核的配置参数
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

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

#进程文件
pid /var/run/nginx.pid;

#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（系统的值 ulimit -n）与 nginx 进程数相除，但是 nginx 分配请求并不均匀，所以建议与 ulimit -n 的值保持一致。
worker_rlimit_nofile 65535;

#工作模式与连接数上限
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，如果跑在 FreeBSD 上面，就用 kqueue 模型。
    #epoll 是多路复用 IO(I/O Multiplexing) 中的一种方式，但是仅用于 linux2.6 以上内核，可以大大提高 nginx 的性能
    use epoll;
```

```
    ##############################################################################
    #单个后台 worker process 进程的最大并发链接数
    #事件模块指令，定义 nginx 每个进程最大连接数，默认 1024。最大客户连接数由 worker_processes 和
worker_connections 决定
    #即 max_client=worker_processes*worker_connections, 在作为反向代理时：
max_client=worker_processes*worker_connections / 4
    worker_connections 65535;
    ##############################################################################
}

#设定 http 服务器
http {
    include mime.types; #文件扩展名与文件类型映射表
    default_type application/octet-stream; #默认文件类型
    #charset utf-8; #默认编码

    server_names_hash_bucket_size 128; #服务器名字的 hash 表大小
    client_header_buffer_size 32k; #上传文件大小限制
    large_client_header_buffers 4 64k; #设定请求缓
    client_max_body_size 8m; #设定请求缓
    sendfile on; #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。
    autoindex on; #开启目录列表访问，合适下载服务器，默认关闭。
    tcp_nopush on; #防止网络阻塞
    tcp_nodelay on; #防止网络阻塞

    ##连接客户端超时时间各种参数设置##
    keepalive_timeout  120;          #单位是秒，客户端连接时时间，超时之后服务器端自动关闭该连接 如
果 nginx 守护进程在这个等待的时间里，一直没有收到浏览发过来 http 请求，则关闭这个 http 连接
    client_header_timeout 10;        #客户端请求头的超时时间
    client_body_timeout 10;          #客户端请求主体超时时间
    reset_timedout_connection on;    #告诉 nginx 关闭不响应的客户端连接。这将会释放那个客户端所占有
的内存空间
    send_timeout 10;                 #客户端响应超时时间，在两次客户端读取操作之间。如果在这段时间
内，客户端没有读取任何数据，nginx 就会关闭连接
    ##################################

    #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;

    ###作为代理缓存服务器设置#######
    ###先写到 temp 再移动到 cache
    #proxy_cache_path /var/tmp/nginx/proxy_cache levels=1:2 keys_zone=cache_one:512m
inactive=10m max_size=64m;
    ###以上 proxy_temp 和 proxy_cache 需要在同一个分区中
```

```
    ###levels=1:2 表示缓存级别，表示缓存目录的第一级目录是 1 个字符，第二级目录是 2 个字符
keys_zone=cache_one:128m 缓存空间起名为 cache_one 大小为 512m
    ###max_size=64m 表示单个文件超过 128m 就不缓存了  inactive=10m 表示缓存的数据，10 分钟内没有被
访问过就删除
    #########end#####################

    #####对传输文件压缩############
    #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 压缩比，1 为最小，处理最快；9 为压缩比最大，处理最慢，传输速度
最快，也最消耗 CPU；
    gzip_types text/plain application/x-javascript text/css application/xml;
    #压缩类型，默认就已经包含 text/html，所以下面就不用再写了，写上去也不会有问题，但是会有一个 warn。
    gzip_vary on;
    ##############################

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

    upstream blog.ha97.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;
    }

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

        #############https##################
        #listen 443 ssl;
        #ssl_certificate /opt/https/xxxxxx.crt;
        #ssl_certificate_key /opt/https/xxxxxx.key;
        #ssl_protocols SSLv3 TLSv1;
        #ssl_ciphers HIGH:!ADH:!EXPORT57:RC4+RSA:+MEDIUM;
        #ssl_prefer_server_ciphers on;
        #ssl_session_cache shared:SSL:2m;
        #ssl_session_timeout 5m;
        ####################################end

        #域名可以有多个，用空格隔开
        server_name www.ha97.com ha97.com;
        index index.html index.htm index.php;
        root /data/www/ha97;
        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
        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 /var/log/nginx/ha97access.log access;

        #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;
            client_max_body_size 10m; #允许客户端请求的最大单文件字节数
            client_body_buffer_size 128k; #缓冲区代理缓冲用户端请求的最大字节数，

            ##代理设置 以下设置是 nginx 和后端服务器之间通讯的设置##
            proxy_connect_timeout 90; #nginx 跟后端服务器连接超时时间（代理连接超时）
            proxy_send_timeout 90; #后端服务器数据回传时间（代理发送超时）
            proxy_read_timeout 90; #连接成功后，后端服务器响应时间（代理接收超时）
            proxy_buffering on;     #该指令开启从后端被代理服务器的响应内容缓冲 此参数开启后
proxy_buffers 和 proxy_busy_buffers_size 参数才会起作用
            proxy_buffer_size 4k;  #设置代理服务器（nginx）保存用户头信息的缓冲区大小
            proxy_buffers 4 32k;   #proxy_buffers 缓冲区，网页平均在 32k 以下的设置
            proxy_busy_buffers_size 64k; #高负荷下缓冲大小（proxy_buffers*2）
            proxy_max_temp_file_size 2048m; #默认 1024m, 该指令用于设置当网页内容大于
proxy_buffers 时，临时文件大小的最大值。如果文件大于这个值，它将从 upstream 服务器同步地传递请求，而不
是缓冲到磁盘
            proxy_temp_file_write_size 512k; 这是当被代理服务器的响应过大时 nginx 一次性写入临时文
件的数据量。
            proxy_temp_path  /var/tmp/nginx/proxy_temp;     ##定义缓冲存储目录，之前必须要先手动创
建此目录
            proxy_headers_hash_max_size 51200;
            proxy_headers_hash_bucket_size 6400;
            ###########################################################
        }

        #设定查看 nginx 状态的地址
```

```
        location /nginxStatus {
            stub_status on;
            access_log on;
            auth_basic "nginxStatus";
            auth_basic_user_file conf/htpasswd;
            #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; }
    }
}
复制代码
```

nginx 服务器基础配置指令

nginx.conf 文件的结构

- Global: nginx 运行相关
- events: 与用户的网络连接相关
- http
  - http Global: 代理，缓存，日志，以及第三方模块的配置
  - server
    - server Global: 虚拟主机相关
    - location: 地址定向，数据缓存，应答控制，以及第三方模块的配置

> 所有的所有的所有的指令，都要以 ; 结尾

nginx 运行相关的 Global 部分

配置运行 nginx 服务器用户

user nobody nobody;

配置允许生成的 worker process 数

worker_processes auto; worker_processes 4;

> 这个数字，跟电脑 CPU 核数要保持一致

```
# grep ^proces /proc/cpuinfo
processor       : 0
processor       : 1
processor       : 2
processor       : 3
# grep ^proces /proc/cpuinfo | wc -l
4
复制代码
```

配置 nginx 进程 PID 存放路径

pid logs/nginx.pid;

> 这里面保存的就是一个数字，nginx master 进程的进程号

配置错误日志的存放路径

error_log logs/error.log; error_log logs/error.log error;

配置文件的引入

include mime.types; include fastcgi_params; include ../../conf/*.conf;

与用户的网络连接相关的 events

设置网络连接的序列化

accept_mutex on;

> 对多个 nginx 进程接收连接进行序列化，防止多个进程对连接的争抢（惊群）

设置是否允许同时接收多个网络连接

multi_accept off;

事件驱动模型的选择

use select|poll|kqueue|epoll|rtsig|/dev/poll|eventport

> 这个重点，后面再看

配置最大连接数

worker_connections 512;

http

http Global 代理 - 缓存 - 日志 - 第三方模块配置

定义 MIME-Type

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

自定义服务日志

access_log logs/access.log main; access_log off;

配置允许 sendfile 方式传输文件

sendfile off;

sendfile on; sendfile_max_chunk 128k;

> nginx 每个 worker process 每次调用 sendfile() 传输的数据量的最大值

Refer:

- Linux kenel sendfile 如何提升性能
- nginx sendifle tcp_nopush tcp_nodelay 参数解释

配置连接超时时间

> 与用户建立连接后，nginx 可以保持这些连接一段时间，默认 75s 下面的 65s 可以被 Mozilla/Konqueror 识别，是发给用户端的头部信息`Keep-Alive`值

keepalive_timeout 75s 65s;

单连接请求数上限

> 和用户端建立连接后，用户通过此连接发送请求；这条指令用于设置请求的上限数

keepalive_requests 100;

server

配置网络监听

listen *:80 | *:8000; # 监听所有的 80 和 8000 端口

listen 192.168.1.10:8000; listen 192.168.1.10; listen 8000; # 等同于 listen *:8000; listen 192.168.1.10 default_server backlog=511; # 该 ip 的连接请求默认由此虚拟主机处理；最多允许 1024 个网络连接同时处于挂起状态

基于名称的虚拟主机配置

server_name myserver.com www.myserver.com;

server_name .*myserver.com www.myserver.* myserver2.*; # 使用通配符

> 不允许的情况： server_name www.ab*d.com; # ``只允许出现在 www 和 com 的位置

server_name ~^www\d+.myserver.com$; # 使用正则

> nginx 的配置中，可以用正则的地方，都以`~`开头

> from nginx~0.7.40 开始，server_name 中的正则支持 字符串捕获功能（capture）

server_name ~^www.(.+).com$; # 当请求通过 www.myserver.com 请求时， myserver 就被记录到 `$1` 中，在本 server 的上下文中就可以使用

如果一个名称 被多个虚拟主机的 server_name 匹配成功，那这个请求到底交给谁处理呢？看优先级：

1. 准确匹配到 server_name
2. 通配符在开始时匹配到 server_name
3. 通配符在结尾时匹配到 server_name
4. 正则表达式匹配 server_name
5. 先到先得

配置 https 证书

原理

https 是在 http 和 TCP 中间加上一层加密层

> - 浏览器向服务端发送消息时：本质上是浏览器（客户端）使用服务端的公钥来加密信息，服务端使用自己的私钥解密，
> - 浏览器从服务端获取消息是：服务端使用自己私钥加密，浏览器（客户端）使用服务端的公钥来解密信息

在这个过程中，需要保证服务端给浏览器的公钥不是假冒的。证明服务端公钥信息的机构是 CA（数字认证中心）

可以理解为：如果想证明一个人的身份是真的，就得证明这个人的身份证是真的

数字证书

```
数字证书相当于物理世界中的身份证，
在网络中传递信息的双方互相不能见面，利用数字证书可确认双方身份，而不是他人冒充的。
这个数字证书由信任的第三方，即认证中心使用自己的私钥对 A 的公钥加密，加密后文件就是网络上的身份证了，即数字证书
复制代码
```

大致可以理解为如下

```
1．服务端将自己的公钥和其他信息（服务端数字证书），请求数字认证中心签名，数字认证中心使用自己的私钥在证书里加密（只有数字认证中心的公钥才能解开）
2．服务端将自己的证书（证书里面包括服务端的公钥）给浏览器
3．浏览器的“证书管理器”中有“受信任的根证书颁发机构”列表，客户端在接收到响应后，会在这个列表里查看是否存在解开该服务器数字证书的公钥。有两种错误情况：如果公钥在这个列表里，但是解码后的内容不匹配，说明证书被冒用；如果公钥不在这个列表里，说明这张证书不是受信任的机构所颁发，他的真实性无法确定
4．如果一切都没问题，浏览器就可以使用服务器的公钥对信息内容进行加密，然后与服务器交换信息（已加密）

+--------------+           +------------------+
|    服务端     |---------->| 数字认证中心（CA）|
+------+-------+    1    X +------------------+
       |                 / /
       |                / /
       |               / /
       |              / /
       |2         3 / / 4
       |            / /
       |           / /
       |          / /
```

```
       X        / /
+--------------+ /
|     浏览器     |X
+--------------+

只要证书（证书里有服务端的公钥）是可信的，公钥就是可信的。
复制代码
```

证书格式

Linux 下的工具们通常使用 base64 编码的文本格式，相关常用后缀如下

- 证书
  - .crt
  - .pem
  - .cer(IIS 等一些平台下，则习惯用 cer 作为证书文件的扩展名，二进制证书）
- 私钥：.key
- 证书请求：.csr
- 其他
  - .keystore java 密钥库（包括证书和私钥）

制作证书

```
1. 生成服务器端的私钥（key 文件）
$openssl genrsa  -out server.key 1024

2. 生成服务器端证书签名请求文件（csr 文件）;
$ openssl req -new -key server.key -out server.csr

...
Country Name:CN------------ 证书持有者所在国家
State or Province Name:BJ-- 证书持有者所在州或省份（可省略不填）
Locality Name:BJ----------- 证书持有者所在城市（可省略不填）
Organization Name:SC------- 证书持有者所属组织或公司
Organizational Unit Name:.- 证书持有者所属部门（可省略不填）
Common Name :ceshi.com----- 域名
Email Address:------------- 邮箱（可省略不填）

A challenge password:------ 直接回车
An optional company name:-- 直接回车


3. 生成证书文件（crt 文件）
$ openssl x509 -req -days 1000 -in server.csr -signkey server.key -out server.crt
复制代码
```

以上生成 server.crt server.key 文件即是用于 HTTPS 配置的证书和 key

如果想查看证书里面的内容，可以通过 $openssl x509 -in server.crt -text -noout 查看

配置 nginx

在 nginx 的 server 区域内添加如下

```
listen 443 ssl;
ssl_certificate /opt/https/server.crt;
ssl_certificate_key /opt/https/server.key;
ssl_protocols SSLv3 TLSv1;
ssl_ciphers HIGH:!ADH:!EXPORT57:RC4+RSA:+MEDIUM;
ssl_prefer_server_ciphers on;
ssl_session_cache shared:SSL:2m;
ssl_session_timeout 5m;
复制代码
```

基于 IP 的虚拟主机配置

> 基于 IP 的虚拟主机，需要将网卡设置为同时能够监听多个 IP 地址

```
ifconfig
# 查看到本机 IP 地址为 192.168.1.30
ifconfig eth1:0 192.168.1.31 netmask 255.255.255.0 up
ifconfig eth1:1 192.168.1.32 netmask 255.255.255.0 up
ifconfig
# 这时就看到 eth1 增加来 2 个别名， eth1:0 eth1:1

# 如果需要机器重启后仍保持这两个虚拟的 IP
echo "ifconfig eth1:0 192.168.1.31 netmask 255.255.255.0 up" >> /etc/rc.local
echo "ifconfig eth1:0 192.168.1.32 netmask 255.255.255.0 up" >> /etc/rc.local
复制代码
```

再来配置基于 IP 的虚拟主机

```
http {
    ...
    server {
     listen 80;
     server_name 192.168.1.31;
     ...
    }
    server {
     listen 80;
     server_name 192.168.1.32;
     ...
    }
}
复制代码
```

配置 location 块

> location 块的配置，应该是最常用的了

location [= | ~ | ~* | ^~] uri {...}

这里内容分 2 块，匹配方式和 uri， 其中 uri 又分为 标准 uri 和正则 uri

先不考虑 那 4 种匹配方式

1. nginx 首先会再 server 块的多个 location 中搜索是否有 `标准 uri` 和请求字符串匹配， 如果有，记录匹配度最高的一个；
2. 然后，再用 location 块中的 `正则 uri` 和请求字符串匹配， 当第一个 `正则 uri` 匹配成功，即停止搜索， 并使用该 location 块处理请求；
3. 如果，所有的 `正则 uri` 都匹配失败，就使用刚记录下的匹配度最高的一个 `标准 uri` 处理请求
4. 如果都失败了，那就失败喽

再看 4 种匹配方式：

- `=`: 用于 `标准 uri` 前，要求请求字符串与其严格匹配，成功则立即处理
- `^~`: 用于 `标准 uri` 前，并要求一旦匹配到，立即处理，不再去匹配其他的那些个 `正则 uri`
- `~`: 用于 `正则 uri` 前，表示 uri 包含正则表达式， 并区分大小写
- `~*`: 用于 `正则 uri` 前， 表示 uri 包含正则表达式， 不区分大小写

> ^~` 也是支持浏览器编码过的 URI 的匹配的哦， 如 `/html/%20/data` 可以成功匹配 `/html/ /data

[root] 配置请求的根目录

Web 服务器收到请求后，首先要在服务端指定的目录中寻找请求资源

```
root /var/www;
复制代码
```

root 后跟的指定目录是上级目录

该上级目录下要含有和 location 后指定名称的同名目录才行，末尾“/”加不加无所谓

```
location /c/ {
      root /a/
}
复制代码
```

访问站点 http://location/c 访问的就是 /a/c 目录下的站点信息。

[alias] 更改 location 的 URI

除了使用 root 指明处理请求的根目录，还可以使用 alias 改变 location 收到的 URI 的请求路径

```
location ~ ^/data/(.+\.(htm|html))$ {
    alias /locatinotest1/other/$1;
}
复制代码
```

alias 后跟的指定目录是准确的，并且末尾必须加“/”，否则找不到文件

```
location /c/ {
      alias /a/
}
复制代码
```

访问站点 http://location/c 访问的就是 /a/ 目录下的站点信息。

【注】一般情况下，在 location / 中配置 root，在 location /other 中配置 alias 是一个好习惯。

设置网站的默认首页

index 指令主要有 2 个作用：

- 对请求地址没有指明首页的，指定默认首页
- 对一个请求，根据请求内容而设置不同的首页，如下：

```
location ~ ^/data/(.+)/web/$ {
    index index.$1.html index.htm;
}
复制代码
```

设置网站的错误页面

error_page 404 /404.html; error_page 403 /forbidden.html; error_page 404 =301 /404.html;

```
location /404.html {
    root /myserver/errorpages/;
}
复制代码
```

基于 IP 配置 nginx 的访问权限

```
location / {
    deny 192.168.1.1;
    allow 192.168.1.0/24;
    allow 192.168.1.2/24;
    deny all;
}
复制代码
```

> 从 192.168.1.0 的用户时可以访问的，因为解析到 allow 那一行之后就停止解析了

基于密码配置 nginx 的访问权限

auth_basic "please login"; auth_basic_user_file /etc/nginx/conf/pass_file;

> 这里的 file 必须使用绝对路径，使用相对路径无效

```
# /usr/local/apache2/bin/htpasswd -c -d pass_file user_name
# 回车输入密码，-c 表示生成文件，-d 是以 crypt 加密。

name1:password1
name2:password2:comment
复制代码
```

> 经过 basic auth 认证之后没有过期时间，直到该页面关闭； 如果需要更多的控制，可以使用 HttpAuthDigestModule wiki.nginx.org/HttpAuthDig...

应用

架设简单文件服务器

将 /data/public/ 目录下的文件通过 nginx 提供给外部访问

```
#mkdir /data/public/
#chmod 777 /data/public/
复制代码
worker_processes 1;
error_log logs/error.log info;
events {
    use epoll;
}
http {
    server {
        # 监听 8080 端口
        listen 8080;
        location /share/ {
            # 打开自动列表功能，通常关闭
            autoindex on;
            # 将 /share/ 路径映射至 /data/public/，请保证 nginx 进程有权限访问 /data/public/
            alias /data/public/;
        }
    }
}
复制代码
```

nginx 正向代理

- 正向代理指代理客户端访问服务器的一个中介服务器，代理的对象是客户端。正向代理就是代理服务器替客户端去访问目标服务器
- 反向代理指代理后端服务器响应客户端请求的一个中介服务器，代理的对象是服务器。

1. 配置

代理服务器配置

nginx.conf

```
server{
    resolver x.x.x.x;
#       resolver 8.8.8.8;
    listen 82;
    location / {
            proxy_pass http://$http_host$request_uri;
    }
    access_log  /data/httplogs/proxy-$host-aceess.log;
}
复制代码
```

location 保持原样即可，根据自己的配置更改 listen port 和 dnf 即 resolver 验证： 在需要访问外网的机器上执行以下操作之一即可：

```
1. export http_proxy=http://yourproxyaddress：proxyport（建议）
2. vim ~/.bashrc
    export http_proxy=http://yourproxyaddress：proxyport
复制代码
```

2 不足 nginx 不支持 CONNECT 方法，不像我们平时用的 GET 或者 POST，可以选用 apache 或 squid 作为代替方案。

nginx 服务器基础配置实例

```
user nginx nginx;

worker_processes 3;

error_log logs/error.log;
pid myweb/nginx.pid;

events {
    use epoll;
    worker_connections 1024;
}

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

    sendfile on;

    keepalive_timeout 65;

    log_format access.log '$remote_addr [$time_local] "$request" "$http_user_agent"';

    server {
        listen 8081;
        server_name myServer1;

        access_log myweb/server1/log/access.log;
        error_page 404 /404.html;

        location /server1/location1 {
            root myweb;
            index index.svr1-loc1.htm;
        }

        location /server1/location2 {
            root myweb;
            index index.svr1-loc2.htm;
        }
```

```
    }

    server {
        listen 8082;
        server_name 192.168.0.254;

        auth_basic "please Login:";
        auth_basic_user_file /opt/X_nginx/nginx/myweb/user_passwd;

        access_log myweb/server2/log/access.log;
        error_page 404 /404.html;

        location /server2/location1 {
            root myweb;
            index index.svr2-loc1.htm;
        }

        location /svr2/loc2 {
            alias myweb/server2/location2/;
            index index.svr2-loc2.htm;
        }

        location = /404.html {
            root myweb/;
            index 404.html;
        }
    }
}
复制代码
#./sbin/nginx -c conf/nginx02.conf
nginx: [warn] the "user" directive makes sense only if the master process runs with super-
user privileges, ignored in /opt/X_nginx/nginx/conf/nginx02.conf:1
.
├── 404.html
├── server1
|   ├── location1
|   |   └── index.svr1-loc1.htm
|   ├── location2
|   |   └── index.svr1-loc2.htm
|   └── log
|       └── access.log
└── server2
    ├── location1
    |   └── index.svr2-loc1.htm
    ├── location2
    |   └── index.svr2-loc2.htm
    └── log
        └── access.log

8 directories, 7 files
复制代码
```

测试 myServer1 的访问

```
http://myserver1:8081/server1/location1/
this is server1/location1/index.svr1-loc1.htm

http://myserver1:8081/server1/location2/
this is server1/location1/index.svr1-loc2.htm
复制代码
```

测试 myServer2 的访问

```
http://192.168.0.254:8082/server2/location1/
this is server2/location1/index.svr2-loc1.htm

http://192.168.0.254:8082/svr2/loc2/
this is server2/location1/index.svr2-loc2.htm

http://192.168.0.254:8082/server2/location2/
404 404 404 404
复制代码
```

使用缓存

创建缓存目录

```
mkdir  /tmp/nginx_proxy_cache2
chmod 777 /tmp/nginx_proxy_cache2
复制代码
```

修改配置文件

```
# http 区域下添加缓存区配置
proxy_cache_path /tmp/nginx_proxy_cache2 levels=1 keys_zone=cache_one:512m inactive=60s
max_size=1000m;

# server 区域下添加缓存配置
#缓存相应的文件（静态文件）
location ~ \.(gif|jpg|png|htm|html|css|js|flv|ico|swf)(.*) {
     proxy_pass http://IP: 端口；#如果没有缓存则通过 proxy_pass 转向请求
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_cache cache_one;
     proxy_cache_valid 200 302 1h;            #对不同的 HTTP 状态码设置不同的缓存时间，h 小时，d
天数
     proxy_cache_valid 301 1d;
     proxy_cache_valid any 1m;
     expires 30d;
}
复制代码
```

使用 location 反向代理到已有网站

```
location ~/bianque/(.*)$ {
        proxy_pass http://127.0.0.1:8888/$1/?$args;
    }
复制代码
```

> - 加内置变量
> $args$是保障$nginx$正则捕获get请求时不丢失，如果只是$post$请求，‘
>   args`是非必须的
> - `$1` 取自正则表达式部分()里的内容

其他

ngx_http_sub_module 替换响应中内容

- ngx_http_sub_module nginx 用来替换响应内容的一个模块（应用：有些程序中写死了端口，可以通过此工具将页面中的端口替换为其他端口）

配置 http 强制跳转 https

在 nginx 配置文件中的 server 区域添加如下内容

```
if ($scheme = 'http') {
    rewrite ^(.*)$ https://$host$uri;
}
```