Nginx笔记

本文档为GeekHour的《30分钟Nginx入门教程》的配套笔记文档，转载请注明出处。
视频地址 ↓
[Bilibili]
[YouTube]

本文档所有内容均来自于Nginx官网，仅供学习使用。
nginx admin guide
nginx documentation

1. Nginx简介

Nginx是目前最流行的Web服务器，
最开始是由一个叫做igor的俄罗斯的程序员开发的，
2019年3月11日被美国的F5公司以6.7亿美元的价格收购，
现在Nginx是F5公司旗下的一款产品了。

2. Nginx的版本

Nginx开源版本主要分为两种，一种是稳定版，一种是主线版。

- 主线版（mainline）：主线版是最新的版本，功能会比较多，会包含一些正在开发中的体验性模块功能，但是也可能会有一些新的bug。
- 稳定版（Stable）：稳定版是经过长时间测试的版本，不会有太多的bug，也不会包含一些新的功能。

3. Nginx安装(预编译二进制包)

这种方式比通过源码编译安装的方式要简单快捷得多，只需要输入一条install命令就可以了。

不同Linux发行版的安装方式略有不同，下面分别介绍一下。

3.1 CentOS/RHEL/Oracle Linux/AlmaLinux/Rocky Linux repository.

CentOS系Linux发行版可以使用yum来安装。

```
# 1. 安装EPEL仓库
sudo yum install epel-release

# 2. 更新repo
sudo yum update

# 3. 安装nginx
sudo yum install nginx

# 4. 验证安装
sudo nginx -V
```

也可以通过Nginx的官方仓库来安装，这样可以保证安装的是最新的版本。

1. 安装前置依赖

```
sudo yum install yum-utils
```

2. 添加nginx仓库

```
sudo vi /etc/yum.repos.d/nginx.repo
```

3. 添加以下内容

```
[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
```

上面的 `stable` 和 `mainline` 就分别表示稳定版和主线版，可以根据自己的需要来选择。

4. 更新repo

```
sudo yum update
```

5. 安装nginx

```
sudo yum install nginx
```

6. 验证安装
   除了使用 `sudo nginx -V` 之外，还可以使用下面的方式来验证：

```
# 启动Nginx
sudo nginx

curl -I 127.0.0.1
```

如果能够看到类似下面的输出，就表示安装成功了：

```
HTTP/1.1 200 OK
Server: nginx/1.25.1
```

3.2 Debian/Ubuntu repository

Debian、Ubuntu系列的Linux发行版可以使用apt来安装。

```
# 1. 更新仓库信息
sudo apt-get update

# 2. 安装nginx
sudo apt-get install nginx

# 3. 验证安装
sudo nginx -V
```

同样也可以从Nginx官方仓库来安装。

```
# 1. 安装前置依赖
sudo apt install curl gnupg2 ca-certificates lsb-release debian-archive-
keyring

# 2. 导入官方Nginx签名密钥
curl https://nginx.org/keys/nginx_signing.key | gpg --dearmor \
    | sudo tee /usr/share/keyrings/nginx-archive-keyring.gpg >/dev/null

# 3. 验证下载的文件中包含了正确的密钥
gpg --dry-run --quiet --no-keyring --import --import-options import-show
/usr/share/keyrings/nginx-archive-keyring.gpg

# 4. 设置稳定版或者主线版的Nginx包
# 稳定版
echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg] \
http://nginx.org/packages/debian `lsb_release -cs` nginx" \
    | sudo tee /etc/apt/sources.list.d/nginx.list

# 主线版
echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg] \
```

```
    http://nginx.org/packages/mainline/debian `lsb_release -cs` nginx" \
        | sudo tee /etc/apt/sources.list.d/nginx.list

    # 5. 设置仓库优先级，优先使用Nginx官方仓库
    echo -e "Package: *\nPin: origin nginx.org\nPin: release o=nginx\nPin-
    Priority: 900\n" \
        | sudo tee /etc/apt/preferences.d/99nginx

    # 6. 安装nginx
    sudo apt update
    sudo apt install nginx

    # 7. 验证安装
    sudo nginx
    curl -I 127.0.0.1
```

3.3 从源码编译安装

从源码编译安装的方式可以让我们自定义Nginx的安装目录、模块等，但是安装过程比较繁琐，需要安装一些依赖库。

3.3.1 安装PCRE库

PCRE是Perl Compatible Regular Expressions的缩写，是一个Perl库，包括perl兼容的正则表达式库。

```
    wget github.com/PCRE2Project/pcre2/releases/download/pcre2-10.42/pcre2-
    10.42.tar.gz
    tar -zxf pcre2-10.42.tar.gz
    cd pcre2-10.42
    ./configure
    make
    sudo make install
```

3.3.2 安装zlib库

zlib是一个数据压缩库，用于Nginx的gzip模块。

```
wget http://zlib.net/zlib-1.2.13.tar.gz
tar -zxf zlib-1.2.13.tar.gz
cd zlib-1.2.13
./configure
make
sudo make install
```

3.3.3 安装OpenSSL库

OpenSSL是一个强大的安全套接字层密码库，用于Nginx的SSL模块。

```
wget http://www.openssl.org/source/openssl-1.1.1t.tar.gz
tar -zxf openssl-1.1.1t.tar.gz
cd openssl-1.1.1t
./Configure darwin64-x86_64-cc --prefix=/usr
make
sudo make install
```

3.3.4 下载Nginx源码

下载主线版的Nginx源码：

```
wget https://nginx.org/download/nginx-1.23.4.tar.gz
tar zxf nginx-1.23.4.tar.gz
cd nginx-1.23.4
```

下载稳定版的Nginx源码：

```
wget https://nginx.org/download/nginx-1.24.0.tar.gz
tar zxf nginx-1.24.0.tar.gz
cd nginx-1.24.0
```

3.3.5 配置编译选项

编译选项可以通过 `./configure --help` 来查看。
下面是一个官网的例子：

```
./configure
--sbin-path=/usr/local/nginx/nginx
--conf-path=/usr/local/nginx/nginx.conf
--pid-path=/usr/local/nginx/nginx.pid
--with-pcre=../pcre2-10.42
--with-zlib=../zlib-1.2.13
--with-http_ssl_module
--with-stream
--with-mail=dynamic
--add-module=/usr/build/nginx-rtmp-module
--add-dynamic-module=/usr/build/3party_module
```

参数（Parameter）	说明（Description）
--prefix=	指定安装目录
--sbin-path=	指定Nginx可执行文件
--conf-path=	指定配置文件位置
--pid-path=	指定pid文件位置
--error-log-path=	指定错误日志文件
--http-log-path=	指定HTTP日志文件
--user=	指定运行Nginx的用户
--group=	指定运行Nginx的组
--with-pcre=	指定PCRE库的位置
--with-pcre-jit	开启PCRE的JIT（Just-in-time compilation）支持
--with-zlib=	指定zlib库的位置

4. Nginx的配置文件

Nginx的配置文件是 `nginx.conf`，一般位于 `/etc/nginx/nginx.conf`。
可以使用 `nginx -t` 来查看配置文件的位置和检查配置文件是否正确。

4.1 配置文件的结构

Nginx的配置文件是由一系列的指令组成的，每个指令都是由一个指令名和一个或者多个参数组成的。
指令和参数之间使用空格来分隔，指令以分号 `;` 结尾，参数可以使用单引号或者双引号来包裹。

配置文件分为以下几个部分：

```
# 全局块
worker_processes  1;
```

```
events {
    # events块
}
http {
    # http块
    server {
        # server块
        location / {
            # location块
        }
    }
}
```

4.1.1 全局块

全局块是配置文件的第一个块，也是配置文件的主体部分，主要用来设置一些影响Nginx服务器整体运行的配置指令，主要包括配置运行Nginx服务器的用户（组）、允许生成的worker process数、进程PID存放路径、日志存放路径和类型以及配置文件引入等。

```
# 指定运行Nginx服务器的用户，只能在全局块配置
# 将user指令注释掉，或者配置成nobody的话所有用户都可以运行
# user [user] [group]
# user nobody nobody;
user  nginx;

# 指定生成的worker进程的数量，也可使用自动模式，只能在全局块配置
worker_processes  1;

# 错误日志存放路径和类型
error_log  /var/log/nginx/error.log warn;

# 进程PID存放路径
pid        /var/run/nginx.pid;
```

4.1.2 events块

```
events {
    # 指定使用哪种网络IO模型，只能在events块中进行配置
    # use epoll

    # 每个worker process允许的最大连接数
    worker_connections  1024;

}
```

4.1.3 http块

http块是配置文件的主要部分，包括http全局块和server块。

```
http {
    # nginx 可以使用include指令引入其他配置文件
    include       /etc/nginx/mime.types;

    # 默认类型，如果请求的URL没有包含文件类型，会使用默认类型
    default_type  application/octet-stream;  # 默认类型

    # 开启高效文件传输模式
    sendfile        on;

    # 连接超时时间
    keepalive_timeout  65;

    # access_log 日志存放路径和类型
    # 格式为：access_log  <path> [format [buffer=size] [gzip[=level]]
[flush=time] [if=condition]];
    access_log  /var/log/nginx/access.log  main;

    # 定义日志格式
    log_format  main  '$remote_addr - $remote_user [$time_local]
"$request" '
```

```
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    # 设置sendfile最大传输片段大小，默认为0，表示不限制
    # sendfile_max_chunk 1m;

    # 每个连接的请求次数
    # keepalive_requests 100;

    # keepalive超时时间
    keepalive_timeout  65;

    # 开启gzip压缩
    # gzip  on;

    # 开启gzip压缩的最小文件大小
    # gzip_min_length 1k;

    # gzip压缩级别，1-9，级别越高压缩率越高，但是消耗CPU资源也越多
    # gzip_comp_level 2;

    # gzip压缩文件类型
    # gzip_types text/plain application/javascript application/x-
javascript text/css application/xml text/javascript application/x-httpd-
php image/jpeg image/gif image/png;

    # upstream指令用于定义一组服务器，一般用来配置反向代理和负载均衡
    upstream www.example.com {
        # ip_hash指令用于设置负载均衡的方式，ip_hash表示使用客户端的IP进行hash，
这样可以保证同一个客户端的请求每次都会分配到同一个服务器，解决了session共享的问题
        ip_hash;
        # weight 用于设置权重，权重越高被分配到的几率越大
        server 192.168.50.11:80 weight=3;
        server 192.168.50.12:80;
        server 192.168.50.13:80;
    }
```

```
    server {
        # 参考server块的配置
    }
}
```

4.1.4 server块

server块是配置虚拟主机的，一个http块可以包含多个server块，每个server块就是一个虚拟主机。

```
server {
    # 监听IP和端口
    # listen的格式为:
    # listen  [ip]:port [default_server] [ssl] [http2] [spdy]
[proxy_protocol] [setfib=number] [fastopen=number] [backlog=number];
    # listen指令非常灵活，可以指定多个IP和端口，也可以使用通配符
    # 下面是几个实际的例子:
    # listen 127.0.0.1:80;  # 监听来自127.0.0.1的80端口的请求
    # listen 80;  # 监听来自所有IP的80端口的请求
    # listen *:80;  # 监听来自所有IP的80端口的请求，同上
    # listen 127.0.0.1;  # 监听来自来自127.0.0.1的80端口，默认端口为80

    listen       80;

    # server_name 用来指定虚拟主机的域名，可以使用精确匹配、通配符匹配和正则匹配等
方式
    # server_name  example.org  www.example.org; # 精确匹配
    # server_name  *.example.org;                # 通配符匹配
    # server_name  ~^www\d+\.example\.net$;      # 正则匹配
    server_name  localhost;

    # location块用来配置请求的路由，一个server块可以包含多个location块，每个
location块就是一个请求路由
    # location块的格式是:
    # location [=|~|~*|^~] /uri/ { ... }
    # = 表示精确匹配，只有完全匹配上才能生效
```

```
        # ~ 表示区分大小写的正则匹配
        # ~* 表示不区分大小写的正则匹配
        # ^~ 表示普通字符匹配，如果匹配成功，则不再匹配其他location
        # /uri/ 表示请求的URI，可以是字符串，也可以是正则表达式
        # { ... } 表示location块的配置内容
        location / {
            # root指令用于指定请求的根目录，可以是绝对路径，也可以是相对路径
            root   /usr/share/nginx/html;  # 根目录
            # index指令用于指定默认文件，如果请求的是目录，则会在目录下查找默认文件
            index  index.html index.htm;  # 默认文件
        }
        # 下面是一些location的示例：
        location = / { # 精确匹配请求
            root   /usr/share/nginx/html;
            index  index.html index.htm;
        }
        location ^~ /images/ { # 匹配以/images/开头的请求
            root   /usr/share/nginx/html;
        }
        location ~* \.(gif|jpg|jpeg)$ { # 匹配以gif、jpg或者jpeg结尾的请求
            root   /usr/share/nginx/html;
        }
        location !~ \.(gif|jpg|jpeg)$ { # 不匹配以gif、jpg或者jpeg结尾的请求
            root   /usr/share/nginx/html;
        }
        location !~* \.(gif|jpg|jpeg)$ { # 不匹配以gif、jpg或者jpeg结尾的请求
            root   /usr/share/nginx/html;
        }

        # error_page 用于指定错误页面，可以指定多个，按照优先级从高到低依次查找
        error_page   500 502 503 504  /50x.html;  # 错误页面
        location = /50x.html {
            root   /usr/share/nginx/html;
        }
    }
```

5. Nginx的常用命令

```
nginx                # 启动Nginx
nginx -c filename    # 指定配置文件
nginx -V             # 查看Nginx的版本和编译参数等信息
nginx -t             # 检查配置文件是否正确，也可用来定位配置文件的位置
nginx -s quit        # 优雅停止Nginx
nginx -s stop        # 快速停止Nginx
nginx -s reload      # 重新加载配置文件
nginx -s reopen      # 重新打开日志文件
```

6. Nginx的常用模块

模块名（Module Name）	描述（Description）
http_access_module	接受或者拒绝特定的客户端请求
http_auth_basic_module	HTTP基本认证，使用用户名和密码来限制对资源的访问
http_autoindex_module	自动索引，用于显示目录列表
http_browser_module	从 `User-Agent` 请求头中获取和识别客户端浏览器
http_charset_module	添加特定的字符集到 `Content-Type` 响应头中
http_empty_gif_module	返回一个1像素的透明GIF图片
http_fastcgi_module	FastCGI支持
http_geo_module	从IP地址中获取地理位置信息
http_gzip_module	Gzip压缩支持
http_limit_conn_module	限制并发连接数
http_limit_req_module	限制请求速率
http_map_module	从变量中获取值
http_memcached_module	Memcached支持
http_proxy_module	反向代理支持

http_referer_module	防盗链
http_rewrite_module	URL重写
http_scgi_module	转发请求到SCGI服务器
http_ssi_module	处理和支持SSI（Server Side Includes）
http_split_clients_module	根据客户端IP地址或者其他变量将客户端分配到组中，一般用于A/B测试
http_upstream_hash_module	启用一致性哈希负载均衡
http_upstream_ip_hash_module	启用IP哈希负载均衡
http_upstream_keepalive_module	启用长连接负载均衡
http_upstream_least_conn_module	启用最少连接负载均衡
http_upstream_zone_module	启用共享内存负载均衡
http_userid_module	为客户端设置一个唯一的ID（UID、cookie）
http_uwsgi_module	转发请求到uWSGI服务器，一般用于Python应用