Nginx

2024年8月30日　　10:32

Nginx

一、简介

Nginx(engine x)是一个高性能的 HTTP 和反向代理服务器，同时也提供了IMAP/POP3/SMTP 服务。Nginx 是由伊戈尔·赛索耶夫为俄罗斯访问量第二的站点开发的。第一个公开版本 0.1.0发布于2004年10月4日。其将源代码以类 BSD 许可证的形式发布，因它的稳定性，丰富的功能集，简单的配置文件和低系统资源的消耗(占有内存少、并发能力强)闻名。百度、京东、新浪、网易、腾讯、淘宝等网站均使用 Nginx。

除了负载均衡，还能做动静分离。网站的一些图片等静态资源，可以放到Nginx中，由Nginx直接返回。没必要发送到后端，由后端进行返回，节省网络资源

二、安装

- **安装**

  sudo apt install nginx

- **查看是否启动**

  sudo systemctl status nginx

```
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2024-08-30 10:38:45 CST; 31s ago
       Docs: man:nginx(8)
    Process: 79797 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, statu>
    Process: 79798 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCE>
   Main PID: 79888 (nginx)
      Tasks: 3 (limit: 4554)
     Memory: 4.5M
        CPU: 43ms
     CGroup: /system.slice/nginx.service
             ├─79888 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
             ├─79891 "nginx: worker process" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ">
             └─79892 "nginx: worker process" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ">
```

- **配置防火墙（或直接关闭防火墙）**

  （1）运行 Nginx后，需要允许流量通过 HTTP(80)和 HTTPS(443)端口。启用'Nginx Ful’，它包含了这两个端口。

  sudo ufw allow 'Nginx Full’

  （2）验证状态

  sudo ufw status

  输出为 状态:不活动　　表示防火墙未开启

- **测试**

  在浏览器中打开 http://自己ip地址，　可以看到默认的Nginx的加载界面

三、配置文件结构目录

1、/etc/nginx/: 这是 Nginx 的主要配置目录。所有的全局配置文件都位于这单

（1） /etc/nginx/nginx.conf: 这是 Nginx 的主配置文件。它控制着 Nginx的全局设置。

（2）/etc/nginx/conf.d/: 用于存放用户自定义的配置文件。默认情况下nginx.conf会包含此目录下的所有配置。这使得管理单独的站点配置变得更加简单。

（3）/etc/nginx/sites-available/: 这个目录用于存放每个可用站点的配置文件。这些文件需要被链接到 sites-enabled 目录下，才能被 Nginx读取和使用。

（4）/etc/nginx/sites-enabled/:存放链接到 sites-available 中活跃站点的配置文件的符号链接。Nginx 将会读取并运行这里的配置。

（5）/etc/nginx/snippets/: 此目录用于存放可以在 Nginx 配置中重复使用的配置片段。

2、/var/log/nginx/: Nginx的日志目录。这里存放着访问日志和错误日志。

（1）access.log: 默认的访问日志文件，记录了所有访问服务器的请求。

（2）error.log:默认的错误日志文件，记录了所有的错误信息。

3、/usr/share/nginx/:存放静态文件和资源的目录。例如，html文件和图像文件等。

4、/var/www/htm!/: 默认的文档根目录。这是 Nginx 默认配置下，用于存放网站文件的地方。

5、/usr/sbin/nginx: Nginx 的可执行文件。用于启动、停止、重载 Nginx 配置等操作。

6、/ib/systemd/system/nginx.service: 如果你的 Ubuntu 使用 systemd 作为服务管理工具，那么 Nginx 的服务定义文件位于此处。该文件定义了如何启动 Nginx 服务，以及在什么情况下启动。

四、相关概念

4.1 MIME类型

多用途互联网邮件扩展类型(MultipurposeInternet MailExtensions)，在 HTTP 中用于描述消息内容的性质和格式。例如text/html 表示 HTML 文档，image/png 表示 PNG 格式的图片。

Nginx 使用哈希表来存储和查找 MIME 类型。types hash max size指令可以设置哈希表的最大大小，也就是可以存储的元素数量。增加这个属性的值可以减少哈希冲突的可能性，但是会消耗更多的内存。在默认情况下types hash max size可能已经足够使用，但是在某些特殊情况下如果有非常多的不同 MIME 类型需要处理，或者遇到由于哈希表大小限制导致的性能问题，可能需要调整这个值。

4.2 正向代理与反向代理

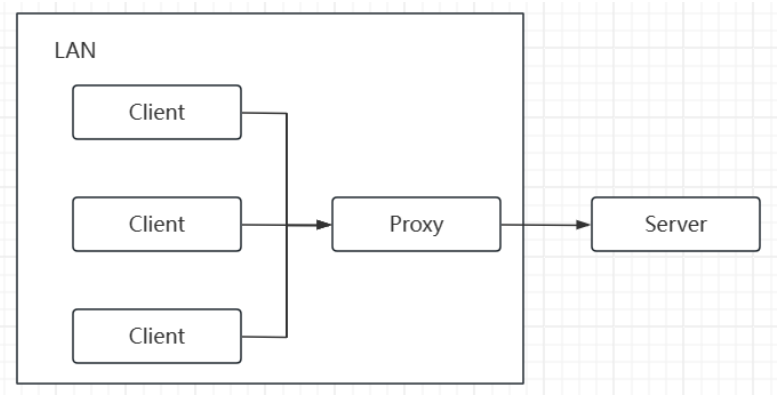
区分正向代理还是反向代理，就看代理服务器是和谁在一个局域网下

正向代理与客户端在一个局域网下

反向代理和服务器在一个局域网下

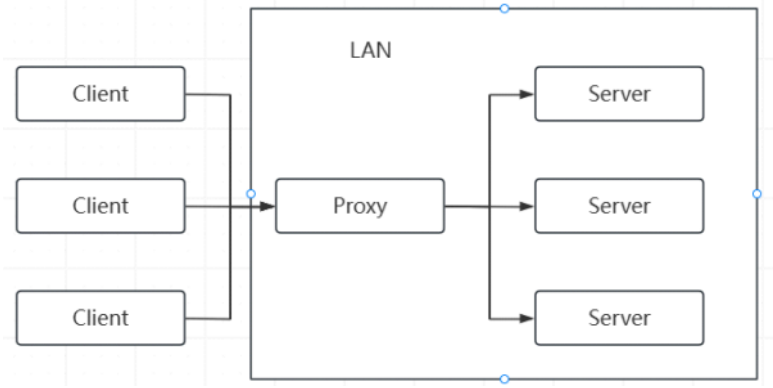
1、正向代理

顺着请求的方向进行的代理，即代理服务器是为发送请求方服务的，去请求目标服务器地址。举例:我们想要访问谷歌，但是由于某些原因导致我们无法直接访问谷歌，我们可以通过连接一台代理服务器，代理服务将我们的请求提交到谷歌，然后再将谷歌的相应返回给我们。



2、反向代理

跟正向代理相反，它是为目标服务器(也就是接收请求方)服务的，但请求流程还是相同的。举例:我们访问百度，百度的代理服务器对外的域名为 baidu.com。具体内部的服务器节点我们是不知道的。当我们访问百度时，它的代理服务器会将我们的请求转发到他们很多的服务器节点中的一个。



4.3 跨域

当一个请求 url 的协议、域名、端口三者之间的任意一个与当前页面url 不同即为跨域。

当前页面 url	被请求页面 url	是否跨域	原因
http://www.test.com	http://www.test.com/index.html	否	同源（协议、域名、端口号相同）
http://www.test.com	https://www.test.com/index.html	跨域	协议不同(http/https)
http://www.test.com	http://www.baidu.com/	跨域	主域名不同(test/baidu)
http://www.test.com	http://blog.test.com/	跨域	子域名不同(www/blog)
http://www.test.com:8080	http://www.test.com:7001/	跨域	端口号不同(8080/7001)

4.4 负载均衡

为了提高服务器的并发量，采用集群进行服务，部署多个服务器进行请求处理。那请求应该交给那个服务器进行处理呢？使用Nginx进行负载均衡。

负载均衡是指将网络或计算资源的工作负载(如数据流量、请求或计算任务)分布到多个服务器、设备或其他资源上，以确保这些资源能够有效地共享负载并提供高可用性和性能。通过负载均衡，可以避免单一服务器或设备因过载而导致性能下降或故障，从而提高系统的稳定性和可靠性。在服务器集群或云计算环境中，负载均衡通常通过专门的负载均衡器来实现，它可以根据预先设置的规则将请求分发到后端的多台服务器上以达到均衡负载、提高性能和可用性的目的。

常见的Nginx负载均衡方式

1、**轮询(round-robin)**:默认的负载均衡策略，即将请求依次分配到不同的后端服务器上。当请求分配到最后一个后端服务器时，统计数据清零，重新从第一个后端服务器开始分配。
就是依次分配到服务器

2、**IP 哈希(ip hash)**:根据客户端的IP 地址进行哈希计算，将同-客户端的请求分配到同一后端服务器上。这种方式可以保证同一客户端的所有请求都会被分配到同一后端服务器上，可以解决某些应用场景下的问题。（一直程度上解决session一致性问题）

3、**最少连接(least conn)**:将请求发送到当前连接数最少的后端服务器上。这种方式可以让负载均衡算法选择处理请求最快的服务器，提高系统响应速度。

4、**加权轮询(weight)**:根据服务器的权重进行请求分配，权重越高的服务器能够处理更多的请求。这种方式可以根据服务器的处理能力分配请求，提高整个系统的性能。

5、**加权最少连接(least conn+weight)**:将最少连接方式与加权方式结合使用，基于连接数和服务器权重进行请求的分配，能够选择最快的和处理能力最强的服务器。

4.5 动静分离

Nginx 动静分离，简单来说，就是动态请求和静态请求分开，也可以理解成使用 Nginx 处理静态页面，后端服务器(uWSGI)处理动态页面动静分离从目前实现角度来讲大致分为两种。

- 纯粹把静态文件独立成单独的域名，放在独立的服务器上(主流推崇的方案)
- 动态跟静态文件混合在一起发布，通过 Nginx 来分开

通过 location 指定不同的后缀名实现不同的请求转发，也可以通过expires(过期时间)参数设置，使浏览器缓存文件的过期时间，从而减少与服务器之前的请求和流量。

动静分离的目的就是为了加快网站的解析速度，可以把动态页面和静态页面交给不同的服务器来解析，降低原来单个服务器的压力。

五、nginx.conf配置文件

1、注释：以#开头

2、模块化配置：通过 include 指令将配置分散到多个文件中，便于管理

3、指令块：如 http、events，用花括号包围的部分，可以包含多个指令或者而嵌套更多的指令块

4、语法规则：

（1）配置文件由指令与指令块构成
（2）每条指令以;分号结尾，指令与参数间以空格符号分隔
（3）指令块以 {}大括号将多条指令组织在一起;
（4）include 语句允许组合多个配置文件以提升可维护性;
（5）使用 # 符号添加注释，提高可读性，
（6）使用$符号使用变量:
（7）部分指令的参数支持正则表达式:

```
#设置 worker 进程的用户，指 linux 中的用户，会涉及到 nginx 操作目录或文件的一些权限，默认为 nobody
user root;
```

```
# worker 进程工作数设置，一般来说 CPU有几个就设置几个，或者n-1个，也可以设置为 auto 以自适应核心数量
worker_processes auto;

# 设置 nginx 进程 pid
pid /run/nginx.pid;

#导入配置文件
include /etc/nginx/modules-enabled/*.conf;

#设置工作模式，配置与事件模型相关的参数
events{
        #每个 worker 允许连接的客户端最大连接数
        worker_connections 768;
        #multi_accept on;
}

#指令块，针对 http 网络传输的一些指令配置
http {
        ##
        # Basic Settings
        ##
        # sendfile 使用高效文件传输，提升传输性能。启用后才能使用tcp_nopush
        sendfile on;

        # 是指当数据表累计一定大小后才发送，提高了效率
        tcp_nopush on;

        # 设置类型哈希表的最大大小，这个指令影响 Nginx如何存储和查找 MIME 类型(多用途互联网邮件扩展类型)
        types_hash_max_size 2048;
        # 用于控制 Nginx是否在错误页面和 HTTP 响应头中发送 Nginx 的版本号
        # server_tokens off;

        # 用于设置值存储服务器名称的哈希表的桶大小。桶大小影响哈希表的效率和内存使用。增加桶大小会使用更多的内存，但可以提高查找速度
        # server_names_hash_bucket size 64;

        # 控制 Nginx 在自动生成的重定向相应中是否使用请求行中的主机名，如果Nginx 服务器配置了多个域名，并且不希望由于某些重定向导致使用错误的域名，可以考虑关闭这个功能，关闭后，Nginx将使用请求中的 Host 头部进行重定向
        # server_name_in_redirect off;

        # include 引入外部配置，提高可读性，避免单个配置文件过大
        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        # 设置 Nginx 服务器支持的 SSL/TLS 协议版本。在这个例子中，Nginx 被配#置为支持多个 SSL/TIS 版本，分别为 TLS 1.0、TLS 1.2、TLS 1.3。指定这些协议版本可以确保与客户端之间建立安全的加密连接。TLS1.3是最新的 TLS 版本，提供更强大的安全性和性能优化。SSLv3 由于存在严重的安全漏洞，如POODLE攻击(可以让攻击者获取 sSL 通信中的部分信息明文，如果将明文中的重要部分获取了，比如cookie，session，则信息的安全出现了隐患)。因此，为了保护服务器和用户数据安全，通常会禁用 SSLv3并选择更安全的TLS 协议版本。通过正确配置SSL/TLS 协议版本，并禁用不安全的协议，可以加强服务器的安全性，保护通信数据不受到潜在的安全威胁。

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TSv1.3; # DroppingSSLv3，ref:POODLE
```

```
        #控制 Nginx在 SSL/TLS 连接过程中优先选择使用哪种加密套件。当设置为 on 时，Nginx会优先选择服务器端设定的加密套件，而不是按照客户端的顺序来选择
        ssl_prefer_server_ciphers on;


        ##
        # Logging Settings
        ##

        # 日志路径
        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        ##
        # Gzip Settings
        #
        # gzip启用压缩，html/js/css压缩后传输会更快
        gzip on;

        # 告诉代理服务器应该在响应头中包含 vary 头信息，以便通知缓存服务器根并据不同的 Accept-Encoding 头信息来存储不同版本的页面
        # gip_vary on;

        # 表示需要对所有请求进行压缩
        # qzip_proxied any;

        # 指定压缩级别为6，级别越高压缩率越大，但同时也会增加CPU使用量
        # gzip_comp_level 6;


        # 指定用于压缩的内存缓冲区大小
        # gzip_buffers 16 8k;

        # 指定启用 Gzip 压缩的 HTTP 版本好
        # gzip_http_version 1.1;
        # 将指定哪些 MIME 类型的文件需要进行 Gzip 压缩
        # gzip types text/plain text/css application/jsonapplication/javascript text/xml application/xmlapplication/xml+rss text/javascript;


        # 设置客户端与服务器请求的超时时间，保证客户端多次请求的时候不会重复芬建立新的连接，节约资源消耗
        # keepalive-timeout 65

        ##
        # Virtual Host Configs
        ##

        # 包含的其他配置文件
        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

5.1 全局块

从配置文件开始到 events 块之间的内容，主要会设置一下影响 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的引入等。

5.2 events块

events 块涉及的指令主要影响 Nginx 服务器与用户的网络连接，常用的设置包括:是否开启对多 work process 下的网络连接进行序列化、是否允许同时接收多个网络连接，选取哪种事件驱动模型来处理连接请求，每个 work process 可以同时支持的最大连接数等。这部分对 Nginx 的性能影响较大，实际运用时应该灵活配置

5.3 http块

这部分是 Nginx 服务器配置中最频繁的部分，代理、缓存和日志定义等绝大多数功能和第三方模块的配置都在这里。需要注意的是:http 块也可以包括 http 全局块、server 块。反向代理、动静分离、负载均衡都是在这部分中配置。

每个 http 块可以包含多个 server 块，而每个 server块相当于一个虚拟主机。
每个 server块也分为全局 server块，以及可以同时包含多个location 块。

1、http 全局块:

包含了一些全局性配置指令如文件引入、MIME-TYPE 定义、Gzip 压缩、日志自定义、保持连接超时时间、单链接请求数上限等。

2、server块:

用于配置一个虚拟主机(或服务器块)，定义了 Nginx 如何处理特定的网站、应用程序或域名。每个 server 块通常包含了一组指令，用于配置该虚拟主机的行为、路由规则、日志设置等。通过配置不同的 server块，可以根据需求为每个虚拟主机定制化的配置，实现不同网站或应用程序之间的隔离和定制化需求

(1) 全局 server块:最常见的配置是本虚拟主机的监听配置和本虚拟主机的名称或 IP 配置
(2) location 块:一个server 块可以配置多个 location 块。
①作用：用于定义不同 URL路径的处理规则，对特定的请求进行处理。地址定向、数据缓存和应答控制等功能，还有许多第三方模块的配置也在这里
②语法:

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

1) = :精确匹配路径，用于不含正则表达式的 uri 前，如果匹配成功，不再进行后续的查找;
2) ^~:用于不含正则表达式的 uri 前，表示如果该符号后面的字符是最佳匹配，采用该规则，不再进行后续的查找;
3) ~:表示用该符号后面的正则去匹配路径，区分大小写
4) ~*:表示用该符号后面的正则去匹配路径，不区分大小写。跟~优先级都比较低，如有多个位置的正则能匹配的话，则使用正则表达式最长的那个

如果 uri 包含正则表达式，则必须要有~或~*标志。

六、相关重要命令

使用systemctl命令管理Nginx服务

- **如果没有 systemctl 命令**

```
sudo apt update
sudo apt install systemd
```

- **启动 Nginx**

```
sudo systemctl start nginx
```

- **设置系统启动时自启动**

```
sudo systemctl enable nginx
```

- **停止 Nginx 服务**

```
sudo systemctl stop nginx
```

- **重启 Nginx 服务**
  sudo systemctl restart nginx

- **重载 Nginx 配置文件**
  sudo systemctl reload nginx

- **查看 Nginx 状态**
  sudo systemctl status nginx

七、nginx功能配置

7.1 反向代理

要将nginx配置为反向代理服务器需要的配置

1、适用场景：反向代理是工作中最常用的服务器功能，经常被用来解决跨域问题。

2、配置文件：

```
#当访问本地的 80 端口时，nginx服务器会自动转发请求到 8080端口，此处只是举个转发的例子，如果proxy_pass指定别的地址如(http://www.baidu.com),访问本地即访问百度首页
server{
    listen       80; # 监听的端口,IPV4
    listen       [::]:80 default server; # 监听的端口,IPV6
    server_name  localhost; # 域名/IP
    root         /usr/share/nginx/html;  # root文件的路径

    location / {
        #指定域名的跳转地址
        proxy pass http://localhost:8080;

    }

}
```

3、举例

监听 9001 端口，将访问不同路径的请求进行反向代理
把访问 http://127.0.0.1:9001/login 的请求转发到http://127.0.0.1:8080
把访问 http://127.0.0.1:9001/register 的请求转发到http://127.0.0.1:8081

```
server {
    listen 9001;
    server_name localhost;

    location ~ /login/ {
      proxy_pass http://127.0.0.1:8080;
    }

    location ~ /register/ {
      proxy_pass http://127.0.0.1:8081;
    }
}
```

7.1 负载均衡

要将nginx配置为负载均衡服务器需要的配置

适用场景:避免单一服务器或设备因过载而导致性能下降或故障，从而提高系统的稳定性和可靠性。

举例：
监听 80 端口，根据配置将请求转发至 8081和 8082端口，这两个端口运行了 uWSGI服务器。以下几种方式选择其一

```
# 轮询方式（默认）
upstream uwsgi_servers {
    server 127.0.0.1:8081;
    server 127.0.0.1:8082;
}

# weight 权重方式
upstream uwsgi_servers {
    server 127.0.0.1:8081 weight = 5;
    server 127.0.0.1:8082 weight = 1;
}

# ip_hash 方式
upstream uwsgi_servers {
    ip_hash;
    server 127.0.0.1:8081;
    server 127.0.0.1:8082;
}

# 最少连接
upstream uwsgi_servers {
    least_conn;
    server 127.0.0.1:8081;
    server 127.0.0.1:8082;
}

server {
        listen 80;

        server_name 123.123.123.123;

        location / {
            include uwsgi_params;
            uwsgi_pass uwsgi_servers;
        }

}
```

7.3 动静分离

当请求以/www/开始，则进入/www/data/目录下找资源，如果是以/image/开始，则进入/image/data/目录下找资源，同时在该路径下配置了个autoindexon，当访问/image/目录时，会列出该目录下的所有文件。

```
http {
    ……

    server {
        listen       80;
        server_name  IP;

        location /www/ {
            root   /data/;
            index  index.html index.htm; #表示当访问的URL对应的目录中没有在指定的默认文件时，会尝试访问index.html或index.htm文件
        }

        location /image/ {
            root   /data/;
            autoindex on; // 列出访问目录
        }
    }
}

```