nginx

什么是nginx

Nginx是一款轻量级的Web服务器、反向代理服务器、负载均衡器，具有高效处理大量并发连接的能力，适用于高并发场景。 Nginx的并发能力在同类型的服务器中表现优异，能够同时处理超过10000个连接。此外，Nginx还具备安全防护功能，可以抵御DDoS攻击和SQL注入攻击，并通过配置访问控制列表(ACL)限制特定IP地址的访问。

nginx的优点

高性能

- **高并发处理能力**：Nginx 使用事件驱动和异步非阻塞的处理方式，能够高效地处理大量的并发连接和请求。这使得 Nginx 在高并发场景下表现出色，非常适合高流量的网站和应用。
- **低资源消耗**：相比于传统的多进程或多线程服务器模型，Nginx 的事件驱动架构在处理大量并发连接时消耗的内存和 CPU 资源较少。这使得 Nginx 能够在有限的硬件资源下提供更高的性能。

稳定性

- **稳定性强**：Nginx 经过多年的开发和广泛的使用，已经非常稳定。它在各种复杂的生产环境中表现出色，能够长时间运行而不需要频繁重启。
- **容错性好**：Nginx 对错误和异常情况有很好的处理机制，能够有效地避免因单个请求或连接的问题导致整个服务器崩溃。

可扩展性

- **模块化设计**：Nginx 采用模块化的设计架构，提供了丰富的模块支持。开发者可以根据需要启用或禁用特定的模块，或者开发新的模块来扩展 Nginx 的功能。
- **插件生态系统**：Nginx 有一个活跃的社区和丰富的插件生态系统，提供了各种功能扩展，如缓存、压缩、安全、负载均衡等。

丰富的功能

- **反向代理和负载均衡**：Nginx 提供了强大的反向代理功能，可以将请求转发到后端的多个服务器上，实现负载均衡和高可用性。它支持多种负载均衡算法，如轮询、最少连接、IP哈希等。
- **静态文件服务**：Nginx 在提供静态文件服务方面非常高效，能够快速地将静态资源（如 HTML、CSS、JavaScript、图片等）发送给客户端。
- **缓存功能**：Nginx 支持多种缓存机制，可以缓存静态文件、动态内容、反向代理的响应等，从而提高响应速度和减轻后端服务器的负载。
- **SSL/TLS 支持**：Nginx 提供了全面的 SSL/TLS 支持，可以轻松地实现 HTTPS 加密通信，保护数据传输的安全性。
- **Web 应用服务器**：Nginx 可以作为 Web 应用服务器，支持与后端应用服务器（如 PHP-FPM、Node.js、Ruby on Rails 等）的集成，提供高性能的 Web 应用服务。

易于配置

- **配置文件简洁**：Nginx 的配置文件语法简洁明了，易于理解和编写。配置指令的结构清晰，便于管理和维护。
- **灵活的配置选项**：Nginx 提供了丰富的配置选项，可以灵活地根据具体需求进行配置和优化。

跨平台支持

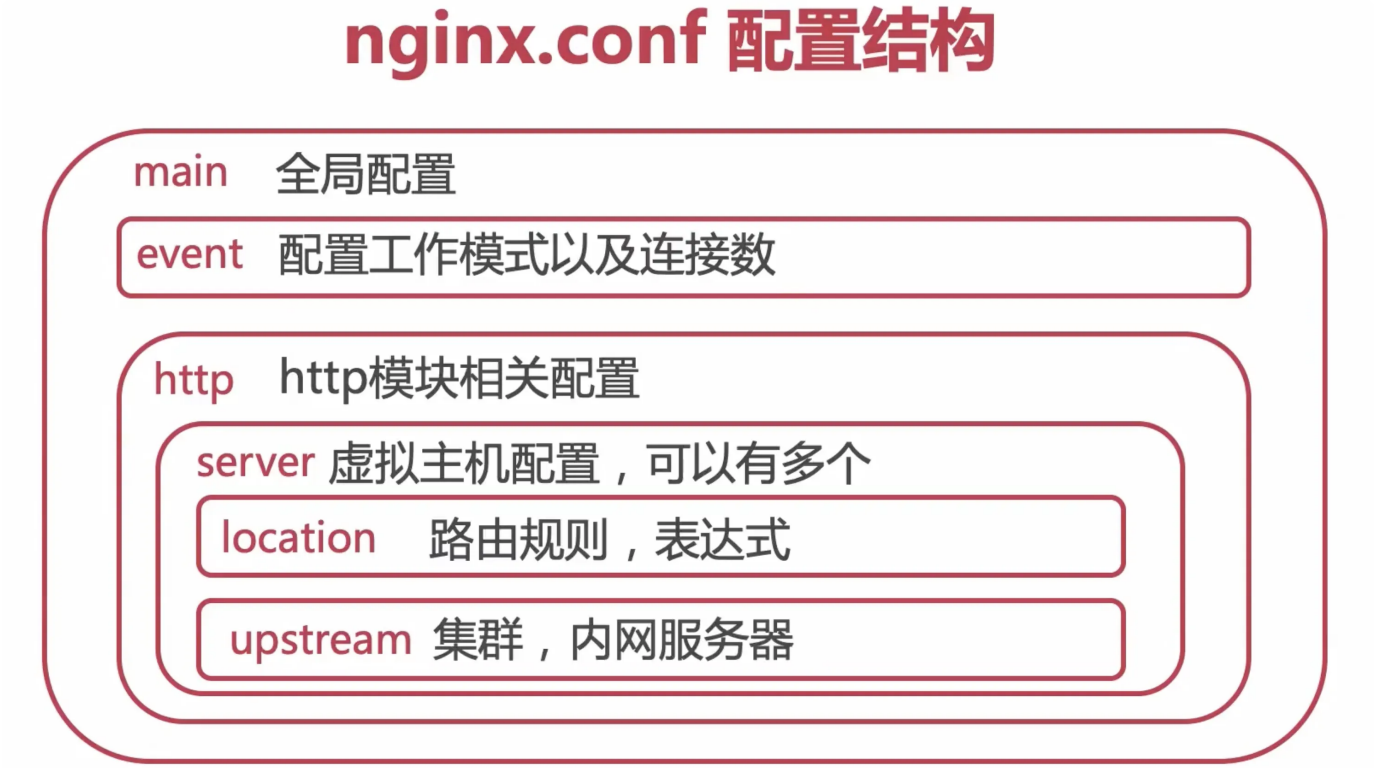
- **多平台兼容性**：Nginx 支持多种操作系统平台，包括 Linux、Unix、Windows 等。这使得 Nginx 可以在不同的服务器环境中部署和运行。

开源和免费

- **开源社区支持**：Nginx 是一个开源软件，拥有一个活跃的社区，社区成员不断贡献代码和改进，使得 Nginx 不断发展和完善。
- **免费使用**：Nginx 的开源版本是免费的，用户可以根据自己的需求自由地使用和修改，降低了使用成本。

配置文件框架

Nginx的配置文件主要由块（blocks）和指令（directives）组成。块用于组织和分组相关的配置，而指令用于指定具体的配置参数。



main块 用于进行nginx全局信息的配置

events块 用于定义Nginx的事件处理配置，主要影响客户端连接的处理。

http块 是Nginx配置文件中最重要的部分之一，它包含了大部分的配置指令和模块设置。

server块 定义了一个虚拟主机或服务器实例的配置。

location块 定义了URL匹配规则和处理请求的方式。

upstream块 用于定义一组后端服务器，通常用于负载均衡。

main块

```nginx
# 用户 用户组
user nginx nginx;
# 工作进程个数 参考CPU个数 不知道可以设置为auto
worker_processes 2;
# 日志及级别 notice info
error_log logs/error.log;
# 进程id存放的文件
pid logs/nginx.pid;
```

events块

```nginx
events {
  # 工作进程最大连接数
  worker_connections 65535;
}
```

http块

作为web服务器，http模块是nginx最核心的一个模块，配置项也是比较多的，项目中会设置到很多的实际业务场景，需要根据硬件信息进行适当的配置，常规情况下，使用默认配置即可！

```nginx
http {
    # 引用文件类型相关配置
    include mime.types;
    # 默认返回文件类型（当没有指定文件类型时会按二进制流返回触发浏览器下载）
    default_type application/octet-stream
    # 定义日志文件记录格式
    log_format bar '$remote_addr - $remote_user [$time_local] "$request"'
                   '$status $body_bytes_sent "$http_feferer"'
                   '"$http_x_forwarded_for"';
    log_format baz '$remote_addr - $remote_user [$time_local] "$request"'
                   '$status $body_bytes_sent "$http_feferer"'
                   '"$http_x_forwarded_for"';
    # 访问日志文件位置 及使用哪种日志格式化方案
    access_log logs/access.log bar;
    # 零拷贝
    sendfile on;
    # 减少网络开销
    tcp_nopush on;
    tcp_nodelay on;
    # 60秒内的请求复用一个TCP连接
    keepalive_timeout 60;
    # 当60秒内请求数超过100 那么需要重新建立TCP连接
    keepalive_requests 100;
    # 开启gzip压缩
    gzip on;
    # 开启文件压缩的阈值
    gzip_min_length 1000;
    # 被压缩文件的文件类型
    gzip_types text/plain application/javascript text/css;
    # 使用server虚拟主机的配置
    server { }
    #upstream指令可以定义一组服务器
    upstream xxx { }
}
```

server块

srever模块配置是http模块中的一个子模块，用来定义一个虚拟访问主机，也就是一个虚拟服务器的配置信息。

```nginx
server {
    # 监听端口
    listen 80;
    # 监听IP+端口
    # listen 127.0.0.1:8080;
    # 配置域名
    server_name localhost;
    # 指定字符集
    charset utf-8;
    # 访问日志
    access_log logs/host.access.log baz;
    # 错误页面
    error_page 404 /404.html;
    # 路由 资源定位
    location / { }
}
```

location模块

location模块是nginx配置中出现最多的一个配置，主要用于配置路由访问信息在路由访问信息配置中关联到反向代理、负载均衡等等各项功能。

```nginx
location / {
  root /www/wwwroot;
  index index.html;
  limit_rate 1k;
  limit_rate_after 40k;
}
location ~* \.(png|jpg|jpeg|gif|webp)$ {
  valid_referers none blocked www.bar.com;
  root /www/wwwroot/bar/images;
  if($invalid_referer) {
    # return 403;
    rewrite /(.*) /abc.jpg break;
  }
}
location /api {
   proxy_.pass http://ddosi.top; #反向代理配置,将请求转发到指定服务
}
```

nginx控制浏览器缓存

在 Nginx 中控制浏览器缓存主要通过设置 HTTP 响应头中的 `Cache-Control` 、 `Expires` 等字段来实现。以下是一些常用的配置方法：

Cache-Control

`Cache-Control` 是 HTTP/1.1 中用于控制缓存行为的字段，常用的指令包括：

- **max-age**：指定资源可以被缓存的时间长度，单位为秒。例如， `max-age=3600` 表示资源可以被缓存 1 小时。
- **no-cache**：指示浏览器在使用缓存前必须先向服务器验证资源是否更新。这并不是完全禁止缓存，而是要求在每次请求时都进行验证。
- **no-store**：完全禁止浏览器缓存该资源，每次请求都会从服务器获取最新内容。
- **public**：指示响应可以被任何中间缓存服务器（如 CDN）缓存，并且可以被其他用户共享。
- **private**：指示响应只能被单个用户的浏览器缓存，不能被其他用户共享。

例如，在 Nginx 配置文件中设置 `Cache-Control` ：

```nginx
location /images/ {
  expires 30d;
  add_header Cache-Control "public, max-age=2592000";
}
```

这里使用了 `expires` 指令来设置资源的过期时间，同时通过 `add_header` 添加了 `Cache-Control` 字段，表示图片资源可以被缓存 30 天，并且是公共缓存。

Expires

`Expires` 是 HTTP/1.0 中用于设置资源过期时间的字段，格式为 GMT 格式的日期时间字符串。例如：

```nginx
location /css/ {
    expires 1y;
}
```

这里设置 CSS 文件的过期时间为 1 年，浏览器会在一年内直接使用缓存的 CSS 文件，而不需要再次请求服务器。

Last-Modified 和 ETag

除了设置缓存时间，还可以通过 `Last-Modified` 和 `ETag` 来实现协商缓存：

- **Last-Modified**：表示资源最后修改的时间。当客户端请求资源时，如果提供了 `If-Modified-Since` 头，服务器会比较这个时间与资源的最后修改时间，如果资源没有更新，则返回 304 Not Modified 响应，客户端使用缓存。
- **ETag**：是资源的唯一标识符，通常由文件内容的哈希值生成。客户端请求时如果提供了 `If-None-Match` 头，服务器会比较这个 ETag，如果资源没有变化，则返回 304 Not Modified 响应。

Nginx 会自动为静态资源生成 `Last-Modified` 和 `ETag` 头，无需手动配置。

示例配置

以下是一个综合示例，展示了如何为不同类型的资源设置浏览器缓存：

```nginx
server {
    # 设置静态资源的缓存
    location ~* \.(jpg|jpeg|gif|png|ico)$ {
        expires 30d;
        add_header Cache-Control "public, max-age=2592000";
    }

    location ~* \.(css|js)$ {
        expires 1y;
        add_header Cache-Control "public, max-age=31536000";
    }

    # 设置 HTML 页面的缓存
    location / {
        expires 1h;
        add_header Cache-Control "public, max-age=3600";
    }
}
```

在这个配置中，图片资源缓存 30 天，CSS 和 JavaScript 文件缓存 1 年，HTML 页面缓存 1 小时。通过合理设置缓存策略，可以减少服务器的负载，提高网站的加载速度和用户体验。

nginx内置变量

请求相关的变量

- `$remote_addr`：客户端的 IP 地址。
- `$remote_port`：客户端的端口号。
- `$server_addr`：服务器的 IP 地址。
- `$server_port`：服务器的端口号。
- `$server_name`：服务器的名称，通常是配置中的 `server_name` 指令的值。
- `$host`：请求中的 `Host` 头部的值，如果请求中没有 `Host` 头部，则使用 `server_name`。
- `$http_host`：请求中的 `Host` 头部的值。
- `$uri`：当前请求的 URI（不包含参数）。
- `$request_uri`：原始请求的 URI，包括参数。
- `$args`：请求中的参数部分。
- `$query_string`：同 `$args`，请求中的参数部分。

- `$document_root`：当前请求的 `root` 指令的值。
- `$request_filename`：当前请求的文件路径，由 `root` 或 `alias` 指令和 URI 计算得出。
- `$scheme`：请求的协议，如 `http` 或 `https`。
- `$request_method`：请求的方法，如 `GET`、`POST` 等。
- `$request_body`：请求体的内容。
- `$content_length`：请求体的长度。
- `$content_type`：请求体的类型。
- `$is_args`：如果请求中有参数，则为 `?`，否则为空字符串。

响应相关的变量

- `$status`：当前请求的响应状态码。
- `$sent_http_content_type`：响应的 `Content-Type` 头部的值。
- `$sent_http_content_length`：响应的 `Content-Length` 头部的值。
- `$sent_http_location`：响应的 `Location` 头部的值。
- `$sent_http_last_modified`：响应的 `Last-Modified` 头部的值。
- `$sent_http_etag`：响应的 `ETag` 头部的值.

连接相关的变量

- `$connection`：当前连接的序列号。
- `$connection_requests`：当前连接的请求数。
- `$pipe`：如果请求是管道请求，则为 `p`，否则为 `.`。
- `$ssl_protocol`：使用的 SSL 协议版本。
- `$ssl_cipher`：使用的 SSL 加密套件。
- `$ssl_session_id`：SSL 会话 ID。
- `$ssl_session_reused`：如果 SSL 会话被重用，则为 `r`，否则为 `.`。

其他变量

- `$msec`：当前时间的时间戳（毫秒级）。
- `$time_iso8601`：当前时间的 ISO 8601 格式。
- `$time_local`：当前时间的本地格式。
- `$pid`：当前 Nginx 进程的 PID。
- `$pids`：当前 Nginx 进程的 PID 列表。
- `$server_protocol`：请求使用的协议版本，如 `HTTP/1.0`、`HTTP/1.1` 等。
- `$request`：完整的请求行，如 `GET / HTTP/1.1`。

- `$request_length`：请求的总长度。
- `$bytes_sent`：已发送给客户端的字节数。
- `$body_bytes_sent`：已发送给客户端的响应体字节数。
- `$connection_requests`：当前连接的请求数。
- `$pipe`：如果请求是管道请求，则为 `p`，否则为 `.`.
- `$limit_rate`：限制带宽的速率。
- `$proxy_protocol_addr`：如果使用了 PROXY 协议，则为客户端的真实 IP 地址.
- `$proxy_protocol_port`：如果使用了 PROXY 协议，则为客户端的真实端口号。