

# Nginx

xbZhong

2025-10-12

[本页PDF](#)

[官方文档](#) [高阶指南](#)

## Nginx

一个高性能的开源web服务器，可以用作反向代理、负载均衡、Http缓存

### 简单控制命令

通过控制 `nginx.exe` 文件来启动 `nginx`

```
nginx -s signal
```

其中 `signal` 为：

- `stop` ：快速关机
- `quit` ：优雅关机
- `reload` ：重新加载配置文件
- `reopen` ：重新打开日志文件

查看 `nginx` 版本号

```
nginx -v
```

获取所有正在运行 `nginx` 进程的列表

```
ps aux | grep nginx
```

检查配置文件语法

```
nginx -t
```

### 配置文件

核心配置块

- 全局块：设置 `Nginx` 整体运行的配置指令
  - `user` ：设定用户组

- `worker_processes` : 设定处理请求的进程数量
- `error_log` : 错误日志存放路径, 后面跟日志级别
- `access_log` : 访问日志存放路径

```
1 user www-data www-data;
2 worker_processes auto;
3 worker_rlimit_nofile 65535;
4 pid /run/nginx.pid;
5
6 error_log /var/log/nginx/error.log warn;
```

- `events` : 调整 Nginx 如何进行网络连接
  - `worker_connections` : 每个进程能处理的最大连接数
  - `epoll` : 用 `epoll` 高效处理连接
  - `multi_accept` : 是否一次性接受所有新连接

```
1 events {
2     worker_connections 2048;
3     use epoll;
4     multi_accept on;
5 }
```

- `http` : 定义网站相关的全局配置
  - `include` : 加载其它配置文件, 可以是站点的配置文件
  - `default_type` : 默认响应类型
    - 如果 Nginx 无法识别客户端请求的文件类型, 就会使用默认响应类型进行返回
  - `keepalive_timeout` : 长连接超时时间
  - `gzip` : 开启响应压缩
    - 将文本类响应压缩后再发送给客户端, 减少传输时间

```
1 http {
2     include      /etc/nginx/mime.types;
3     default_type application/octet-stream;
4     keepalive_timeout 65;
5     gzip on;
6
7     # 后续包含 server 块
8 }
```

- `server` : 定义一个虚拟主机, 通过不同域名或者端口区分多个网站
  - `listen 80` : 监听80端口
  - `server_name` : 域名
  - `root` : 站点根目录

- `index` : 默认首页文件
- `error` : 自定义错误页面

```
1  server {
2      listen      80;
3      server_name example.com;
4      root        /var/www/html;
5      index       index.html;
6
7      # 后续包含 location 块
8  }
```

- `location` : 根据URI路径匹配不同的处理规则

- 匹配语法

- `location /path/` : 前缀匹配 (区分大小写)
    - `location ~ regex` : 正则匹配 (区分大小写)
    - `location = /path` : 精确匹配

- 核心指令

- `root` : 文件系统路径
    - `proxy_pass` : 反向代理
    - `try_files` : 按顺序试文件
    - `expires` : 缓存控制
      - `s` : 秒
      - `m` : 分钟
      - `h` : 小时
      - `d` : 天
      - `w` : 周
      - `m` : 月
      - `y` : 年

```
1  location / {
2      try_files $uri $uri/ /index.php?$query_string;
3  }
4
5  location ~ /\.php$ {
6      proxy_pass http://php_backend;
7  }
8
9  location ~* \.(jpg|png)$ {
10     root /data/media;
11     expires 7d;
12 }
```

- `proxy_set_header` : 配置请求头信息

- `Host` : 客户端原始请求域名
- `Referer` : Referer头
- `Cookie` : Cookie头
- `X-Real-IP` : 客户端真实IP
- `X-Forwarded-For` : 代理链

```
1 proxy_set_header Host $host;  
2 proxy_set_header X-Real-IP $remote_addr;  
3 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
4 proxy_set_header X-Forwarded-Proto $scheme;
```

### 其它注意事项

- `$` : 是 `Nginx` 内置变量, 用于动态获取请求或服务器的信息
  - `$uri` : 当前请求的 `uri` , 不包含查询参数
  - `$request_uri` : 完整的原始请求 `uri`
  - `$args` : 查询参数
  - `$host` : 请求的主机名
  - `$scheme` : 请求协议

### 配置文件架构

```

1  # 主上下文 (main context) 中的指令
2  user nobody; # 指定运行Nginx工作进程的系统用户 (这里是'nobody')
3
4  events {
5      # 连接处理相关的配置 (事件驱动模型参数)
6  }
7
8  http {
9      # HTTP协议相关的全局配置 (影响所有虚拟服务器)
10
11     server {
12         # 第一个HTTP虚拟服务器的配置
13         location /one {
14             # 处理以 '/one' 开头的URI请求的配置
15         }
16         location /two {
17             # 处理以 '/two' 开头的URI请求的配置
18         }
19     }
20
21     server {
22         # 第二个HTTP虚拟服务器的配置
23     }
24 }
25
26 stream {
27     # TCP/UDP协议相关的全局配置 (用于非HTTP流量, 如数据库连接、邮件服务等)
28     server {
29         # 第一个TCP虚拟服务器的配置
30     }
31 }

```

架构层级关系如下

```

1  main (全局)
2  └─ events (事件模型)
3  └─ http (HTTP服务)
4      └─ server (虚拟主机1)
5          └─ location /one (路径规则1)
6          └─ location /two (路径规则2)
7      └─ server (虚拟主机2)
8  └─ stream (TCP/UDP服务)
9      └─ server (TCP服务1)

```

# 负载均衡

## 常见负载均衡算法

- 轮询
- 加权轮询
- 最少连接数：长连接，占用服务器资源场景
- IP哈希：会话保持，同一用户访问同一后端
- 响应时间优先：需要第三方模块，按后端响应速度动态调整

## 配置流程

- 在 `http` 下定义 `upstream backend_servers`

```
1 http {
2     upstream backend_servers {
3         # 定义后端服务器列表
4         server 192.168.1.101:8080;
5         server 192.168.1.102:8080;
6         server 192.168.1.103:8080;
7     }
8 }
```

- 在 `server` 中配置代理，将请求转发到 `upstream` 组

```
1 server {
2     listen 80;
3     server_name example.com;
4
5     location / {
6         proxy_pass http://backend_servers; # 关键指令：指向 upstream 名称
7         proxy_set_header Host $host;
8     }
9 }
```

# 重定向

可以使用 `return` 或者 `rewrite` 进行重定向，可对 `server` 和 `location` 进行重定向

## return 语法

- `code` 是HTTP状态码
- `URL` 是重定向目标地址

```
return code [URL/text];
```

## rewrite 语法

- `regex` 是匹配URL的正则表达式
- `replacement` 是替换后的URL
- `flag` 可以是
  - `last` : 停止当前 `rewrite` 规则，**用修改后的URI重新匹配** `location`
  - `break` : 停止所有 `rewrite` 处理，**继续执行当前** `location` **的剩余指令**
  - `redirect` : 302临时重定向
  - `permanent` : 301永久重定向

```
rewrite regex replacement [flag];
```

## 跨域配置

**跨域**是浏览器的安全策略

- 浏览器会**阻止**前端 `Javascript` 代码直接访问不同源的API
- 不同源指的就是**协议、ip地址、端口号**任意一个不相同

### 基础配置

- `add_header 'Access-Control-Allow-Origin'` : 允许的域名
- `add_header 'Access-Control-Allow-Methods'` : 允许的HTTP方法
- `add_header 'Access-Control-Allow-Headers'` : 允许的请求头
- `add_header 'Access-Control-Allow-Credentials'` : 允许浏览器携带Cookie

```
1  server {
2      listen 80;
3      server_name api.example.com;
4
5      location / {
6          # 允许的域名（生产环境建议替换为具体域名）
7          add_header 'Access-Control-Allow-Origin' '*';
8
9          # 允许的 HTTP 方法
10         add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT,
DELETE';
11
12         # 允许的请求头
13         add_header 'Access-Control-Allow-Headers' 'Content-Type, Authorization,
X-Requested-With';
14
15         # 允许浏览器携带 Cookie（需配合具体域名，不能为 *）
16         add_header 'Access-Control-Allow-Credentials' 'true';
17
18         # 预检请求（OPTIONS）缓存时间
19         add_header 'Access-Control-Max-Age' 1728000;
20
21         # 正常请求转发到后端
22         proxy_pass http://backend_server;
23     }
24 }
```

## 常见状态码

### 1xx：信息类

- 100：客户端继续发送请求
- 101：协议切换

### 2xx：成功类

- 200：请求成功
- 201：Post创建成功
- 204：请求成功但无返回内容

### 3xx：重定向类

- 301：永久重定向
- 302/307：临时重定向
- 304：缓存有效

### 4xx：客户端错误

- 400：请求语法错误



- 401：未认证
- 403：无权限
- 404：资源不存在
- 405：请求方法不允许
- 429：请求过多，触发限流

#### **5xx：服务端错误**

- 500：服务器内部错误
- 502：网关错误
- 503：服务不可用
- 504：网关超时