

# Nginx

## Nginx

### Nginx 介绍

Nginx 的最重要的几个使用场景:

### 相关概念

简单请求和非简单请求

浏览器处理简单请求和非简单请求

简单请求

非简单请求

跨域

正向代理和反向代理

正向代理

反向代理

负载均衡

动静分离


### Nginx 操作常用命令

windows

linux

### Nginx 配置语法

全局变量

 源文件地址: [Nginx 从入门到实践, 万字详解! - NGINX开源社区](#)

## Nginx 介绍

---

传统的 Web 服务器，每个客户端连接作为一个单独的进程或线程处理，需在切换任务时将 CPU 切换到新的任务并创建一个新的运行时上下文，消耗额外的内存和 CPU 时间，当并发请求增加时，服务器响应变慢，从而对性能产生负面影响。



Nginx 是 **开源**、**高性能**、**高可靠** 的 **Web** 和反向代理服务器，而且支持 **热部署**，几乎可以做到 7 \* 24 小时不间断运行，即使运行几个月也不需要重新启动，还能在不间断服务的情况下对软件版本进行热更新。性能是 Nginx 最重要的考量，其占用内存少、并发能力强、能支持高达 **5w** 个并发连接数，最重要的是，Nginx 是 **免费的并可以商业化**，配置使用也比较简单。

## Nginx 的最重要的几个使用场景：

- 1 静态资源服务，通过本地文件系统提供服务；
- 2 反向代理服务，延伸出包括缓存、负载均衡等；
- 3 API 服务，OpenResty；

对于前端来说 **Node.js** 不陌生了，**Nginx** 和 **Node.js** 的很多理念类似，**HTTP** 服务器、事件驱动、异步非阻塞等，且 **Nginx** 的大部分功能使用 **Node.js** 也可以实现，但 **Nginx** 和 **Node.js** 并不冲突，都有自己擅长的领域。

**Nginx** 擅长于底层服务器端资源的处理（静态资源处理转发、反向代理，负载均衡等），

**Node.js** 更擅长上层具体业务逻辑的处理，两者可以完美组合，共同助力前端开发。

## ✂ 相关概念

---

### 简单请求和非简单请求

首先我们来了解一下简单请求和非简单请求，如果同时满足下面两个条件，就属于简单请求：

- 1 请求方法是 HEAD 、 GET 、 POST 三种之一；
- 2 HTTP 头信息不超过 右边着几个字段： Accept 、 Accept-Language 、 Content-Language 、 Last-Event-ID 、 Content-Type 只限于 三个值 application/x-www-form-urlencoded 、 multipart/form-data 、 text/plain ；

凡是不同时满足这两个条件的，都属于非简单请求。

## 浏览器处理简单请求和非简单请求

### H5 简单请求

对于简单请求，浏览器会在头信息中增加 Origin 字段后直接发出，Origin 字段用来说明，本次请求来自的哪个源（协议+域名+端口）。

如果服务器发现 Origin 指定的源不在许可范围内，服务器会返回一个正常的 HTTP 回应，浏览器取到回应之后发现回应的头信息中没有包含 Access-Control-Allow-Origin 字段，就抛出一个错误给 XHR 的 error 事件；

如果服务器发现 Origin 指定的域名在许可范围内，服务器返回的响应会多出几个 Access-Control- 开头的头信息字段。

### H5 非简单请求

非简单请求是那种对服务器有特殊要求的请求，比如请求方法是 PUT 或 DELETE，或 Content-Type 值为 application/json。浏览器会在正式通信之前，发送一次 HTTP 预检 OPTIONS 请求，先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些 HTTP 请求方法和头信息字段。只有得到肯定答复，浏览器才会发出正式的 XHR 请求，否则报错。

## 跨域

在浏览器上当前访问的网站向另一个网站发送请求获取数据的过程就是跨域请求。

跨域是浏览器的 **同源策略** 决定的，是一个重要的 **浏览器安全策略**，用于限制一个 **origin** 的文档或者它加载的脚本与另一个源的资源进行交互，它能够帮助阻隔恶意文档，减少可能被攻击的媒介，可以使用 **CORS 配置** 解除这个限制。

关于跨域网上已经有很多解释，这里就不啰嗦，也可以直接看 **MDN** 的 <**浏览器的同源策略**> 文档进一步了解，这里就列举几个同源和不同元的例子，相信程序员都能看得懂。

```
1  # 同源的例子
2  http://example.com/app1/index.html  # 只是路径不同
3  http://example.com/app2/index.html
4
5  http://Example.com:80  # 只是大小写差异
6  http://example.com
7
8  # 不同源的例子
9  http://example.com/app1  # 协议不同
10 https://example.com/app2
11
12 http://example.com  # host 不同
13 http://www.example.com
14 http://myapp.example.com
15
16 http://example.com  # 端口不同
17 http://example.com:8080
18
```

## 正向代理和反向代理

**反向代理**（Reverse Proxy）对应的是 **正向代理**（Forward Proxy），他们的区别：

### 正向代理

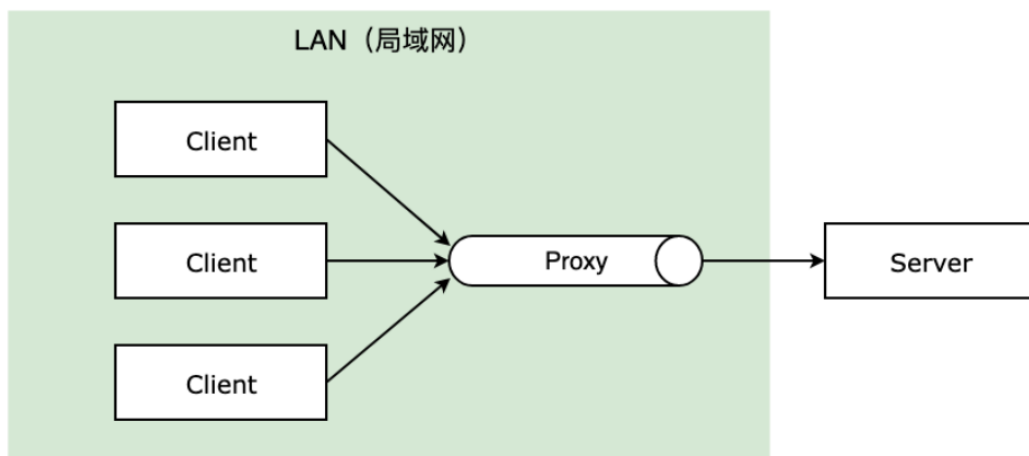
一般的访问流程是 **客户端** 直接向 **目标服务器** 发送请求并获取内容，

使用正向代理后，**客户端** 改为向 **代理服务器** 发送请求，并 **指定目标服务器**（原始服务器），然后由 **代理服务器** 和 **原始服务器** 通信，转交请求并获得的内容，再返回给 **客户端**。正向代理隐藏了真实的客户端，为客户端收发请求，使真实客户端对服务器不可见；

i

举个具体的例子 🌩️，你的浏览器无法直接访问谷歌，这时候可以通过一个代理服务器来帮助你访问谷歌，那么这个服务器就叫正向代理。

## 正向代理



## 反向代理

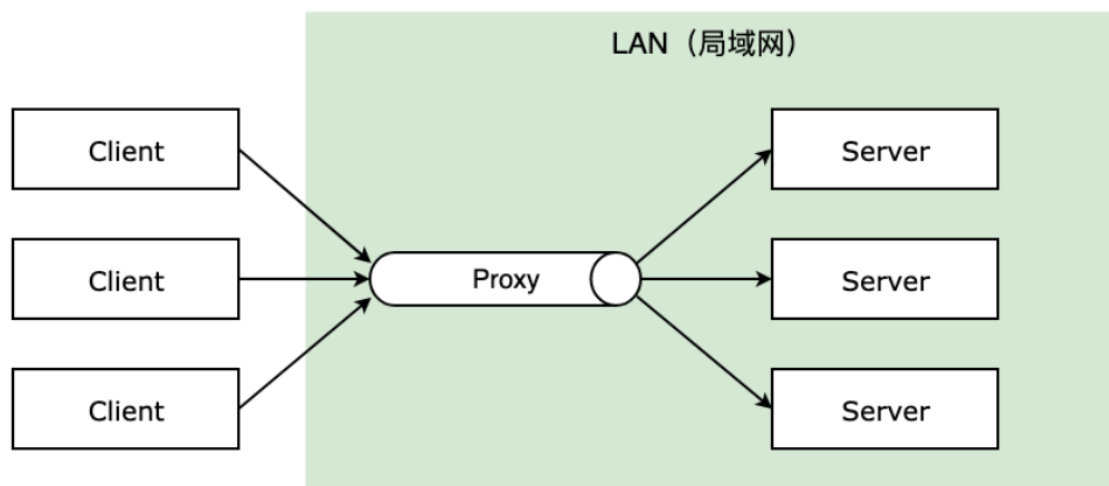
与一般访问流程相比，

使用 **反向代理** 后，**直接收到请求的服务器** 是 **代理服务器**，然后将请求转发给 **内部网络上** 真正进行处理的 **服务器**，得到的结果返回给 **客户端**。反向代理隐藏了真实的服务器，为 **服务器** 收发请求，使 **真实服务器** 对 **客户端** 不可见。一般在处理跨域请求的时候比较常用。现在基本上所有的大型网站都设置了反向代理。

i

举个具体的例子 🌩️，去饭店吃饭，可以点川菜、粤菜、江浙菜，饭店也分别有三个菜系的厨师 👨🍳，但是你作为顾客不用管哪个厨师给你做的菜，只用点菜即可，小二将你菜单中的菜分配给不同的厨师来具体处理，那么这个小二就是反向代理服务器。

## 反向代理



简单的说，一般给客户端做代理的都是正向代理，给服务器做代理的就是反向代理。

## 负载均衡

一般情况下，客户端发送多个请求到服务器，服务器处理请求，其中一部分可能要操作一些资源比如数据库、静态资源等，服务器处理完毕后，再将结果返回给客户端。

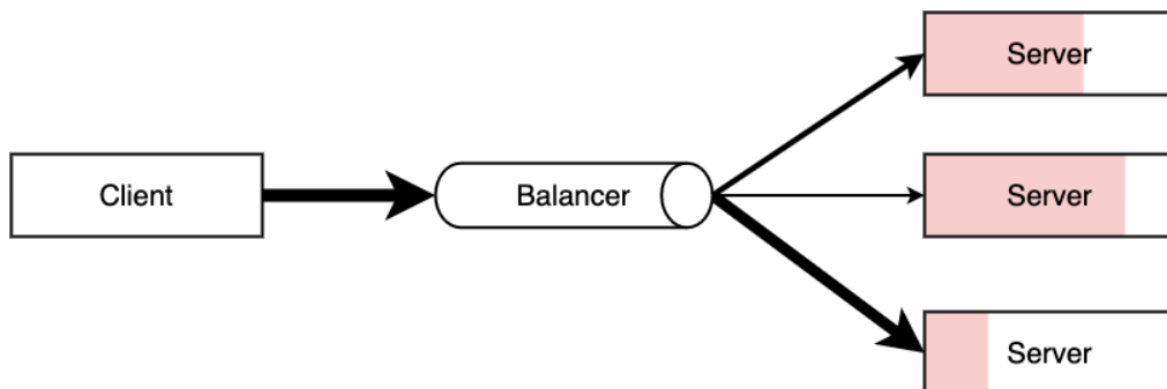
这种模式对于早期的系统来说，功能要求不复杂，且并发请求相对较少的情况下还能胜任，成本也低。随着信息数量不断增长，访问量和数据量飞速增长，以及系统业务复杂度持续增加，这种做法已无法满足要求，并发量特别大时，服务器容易崩。

很明显这是由于服务器性能的瓶颈造成的问题，除了堆机器之外，最重要的做法就是负载均衡。

请求爆发式增长的情况下，单个机器性能再强劲也无法满足要求了，这个时候集群的概念产生了，单个服务器解决不了的问题，可以使用多个服务器，然后将请求分发到各个服务器上，将负载分发到不同的服务器，这就是负载均衡，核心是「分摊压力」。Nginx 实现负载均衡，一般来说指的是将请求转发给服务器集群。



举个具体的例子 🚶，晚高峰乘坐地铁的时候，进站口经常会有地铁工作人员大喇叭“请走 B 口，B 口人少车空...”，这个工作人员的作用就是负载均衡。



## 动静分离

一般来说，都需要将 **动态资源** 和 **静态资源** 分开，由于 **Nginx** 的 **高并发** 和 **静态资源缓存** 等特性，经常将 **静态资源** 部署在 **Nginx** 上。如果请求的是 **静态资源**，直接到 **静态资源目录** 获取资源，如果是 **动态资源** 的请求，则利用 **反向代理** 的原理，把请求转发给 **对应后台应用** 去处理，从而实现 **动静分离**。

使用 **前后端分离** 后，可以很大程度 **提升静态资源的访问速度**，即使动态服务不可用，静态资源的访问也不会受到影响

## ✧ Nginx 操作常用命令

### windows

**Nginx** 的命令在 **控制台** 中输入 **nginx -h** 就可以看到完整的命令，这里列举几个常用的命令：

```
1  # 向主进程发送信号，重新加载配置文件，热重启
2  nginx -s reload
3  # 重启 Nginx
4  nginx -s reopen
5  # 快速关闭
6  nginx -s stop
7  # 等待工作进程处理完成后关闭
8  nginx -s quit
9  # 查看当前 Nginx 最终的配置
10 nginx -T
11 # 检查配置是否有问题，如果已经在配置目录，则不需要-c
12 nginx -t -c <配置路径>
```

`systemctl` 是 `Linux` 系统应用管理工具 `systemd` 的 **主命令**，用于 **管理系统**，我们也可以用它对 `Nginx` 进行管理，相关命令如下：

```
1 systemctl start nginx      # 启动 Nginx
2 systemctl stop nginx       # 停止 Nginx
3 systemctl restart nginx    # 重启 Nginx
4 systemctl reload nginx     # 重新加载 Nginx, 用于修改配置后
5 systemctl enable nginx     # 设置开机启动 Nginx
6 systemctl disable nginx    # 关闭开机启动 Nginx
7 systemctl status nginx     # 查看 Nginx 运行状态
```

## ✧ Nginx 配置语法

就跟前面文件作用讲解的图所示，`Nginx` 的主配置文件是 `/etc/nginx/nginx.conf`

你可以使用 `cat -n nginx.conf` 来查看配置。

`nginx.conf` 结构图可以这样概括：

```
1 main          # 全局配置，对全局生效
2 |─ events     # 配置影响 Nginx 服务器或与用户的网络连接
3 |─ http       # 配置代理，缓存，日志定义等绝大多数功能和第三方模块的配置
4 |   |─ upstream # 配置后端服务器具体地址，负载均衡配置不可或缺的部分
5 |   |─ server   # 配置虚拟主机的相关参数，一个 http 块中可以有多
server 块
6 |   |─ server
7 |   |   |─ location # server 块可以包含多个 location 块，location
指令用于匹配 uri
8 |   |   |─ location
9 |   |   |─ ...
10 |   |   |─ ...
11 |   |   |─ ...
```

一个 `Nginx` 配置文件的结构就像 `nginx.conf` 显示的那样，配置文件的语法规则：

- 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;

Nginx 的典型配置:

```
1 user nginx; # 运行用户, 默认即是nginx, 可以不
   进行设置
2 worker_processes 1; # Nginx 进程数, 一般设置为和 CPU
   核数一样
3 error_log /var/log/nginx/error.log warn; # Nginx 的错误日志存放目
   录
4 pid /var/run/nginx.pid; # Nginx 服务启动时的 pid 存放位
   置
5
6 events {
7     use epoll; # 使用epoll的I/O模型(如果你不知道Nginx该使用哪种轮询
   方法, 会自动选择一个最适合你操作系统的)
8     worker_connections 1024; # 每个进程允许最大并发数
9 }
10
11 http { # 配置使用最频繁的部分, 代理、缓存、日志定义等绝大多数功能和第三方
   模块的配置都在这里设置
12     # 设置日志模式
13     log_format main '$remote_addr - $remote_user [$time_local]
   "$request" '
14                     '$status $body_bytes_sent "$http_referer" '
15                     '"$http_user_agent"
   "$http_x_forwarded_for"';
16
17     access_log /var/log/nginx/access.log main; # Nginx访问日志
   存放位置
18
19     sendfile on; # 开启高效传输模式
20     tcp_nopush on; # 减少网络报文段的数量
21     tcp_nodelay on;
22     keepalive_timeout 65; # 保持连接的时间, 也叫超时时间, 单位秒
23     types_hash_max_size 2048;
24
25     include /etc/nginx/mime.types; # 文件扩展名与
   类型映射表
26     default_type application/octet-stream; # 默认文件类型
27
```

```

28     include /etc/nginx/conf.d/*.conf;    # 加载子配置项
29
30     server {
31         listen      80;                # 配置监听的端口
32         server_name localhost;        # 配置的域名
33
34         location / {
35             root     /usr/share/nginx/html; # 网站根目录
36             index    index.html index.htm;  # 默认首页文件
37             deny     172.168.22.11;        # 禁止访问的ip地址, 可以为all
38             allow    172.168.33.44;        # 允许访问的ip地址, 可以为all
39         }
40
41         error_page 500 502 503 504 /50x.html; # 默认50x对应的访问页面
42         error_page 400 404 error.html;        # 同上
43     }
44 }

```

**server** 块可以包含多个 **location** 块，**location 指令** 用于 **匹配 uri**，语法：

```

1 location [ = | ~ | ~* | ^~ ] uri {
2     ...
3 }

```

指令后面：

- 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 **location** 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。

## 全局变量

**Nginx** 有一些常用的 **全局变量**，你可以在配置的任何位置使用它们，如下表：

全局变量名	功能
<code>\$host</code>	请求信息中的 <code>Host</code> ，如果请求中没有 <code>Host</code> 行，则等于设置的服务器名，不包含端口
<code>\$request_method</code>	客户端请求类型，如 <code>GET</code> 、 <code>POST</code>
<code>\$remote_addr</code>	客户端的 <code>IP</code> 地址
<code>\$args</code>	请求中的参数
<code>\$arg_PARAMETER</code>	<code>GET</code> 请求中变量名 <code>PARAMETER</code> 参数的值，例如： <code>\$http_user_agent</code> (User-Agent 值), <code>\$http_referer</code> ...
<code>\$content_length</code>	请求头中的 <code>Content-length</code> 字段
<code>\$http_user_agent</code>	客户端agent信息
<code>\$http_cookie</code>	客户端cookie信息
<code>\$remote_addr</code>	客户端的IP地址
<code>\$remote_port</code>	客户端的端口
<code>\$http_user_agent</code>	客户端agent信息
<code>\$server_protocol</code>	请求使用的协议，如 <code>HTTP/1.0</code> 、 <code>HTTP/1.1</code>
<code>\$server_addr</code>	服务器地址
<code>\$server_name</code>	服务器名称
<code>\$server_port</code>	服务器的端口号
<code>\$scheme</code>	HTTP 方法（如http，https）

还有更多的 **内置预定义变量**，可以直接搜索关键字「**nginx内置预定义变量**」可以看到一堆博客写这个，这些变量都可以在配置文件中直接使用。