



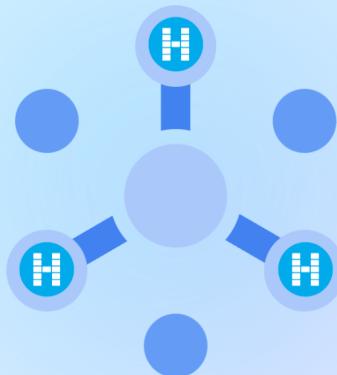
格言格语 活字格网关方案

开始学习 →

联系专家 🤖

Released under the MIT License.

Copyright © 2023-present 格言格语



格言格语 活字格集群负载均衡方案

开始学习 →

联系专家 🤖

Released under the MIT License.
Copyright © 2023-present 格言格语

跨域请求

跨域请求（Cross-Origin Request）是指浏览器从一个来源（域名、协议、端口）向另一个不同来源发送的请求。

NOTE

例如，页面在 `https://example.com` 上运行，但尝试向 `https://api.another.com` 发出请求，这就是跨域请求。

在活字格应用的实施场景中，经常会遇到跨系统的对接与集成。这时必然会存在在已有系统的页面上请求活字格应用的服务接口，或者在已有页面中内嵌活字格应用页面并免登录跳转，这些场景都是跨域请求的典型场景。

跨域限制

为了保证数据的安全，浏览器本身提供了 **同源策略（Same-Origin Policy）** 的安全机制，用来防止恶意网站访问另一个网站的敏感信息。

NOTE

如果没有同源策略，用户可能会在不知情的情况下，通过恶意网站向另一个站点发起请求，窃取数据或者造成其他危害。

同源策略的核心限制包括：

- 1. DOM 访问限制：**不同源的页面无法操作彼此的 DOM。
- 2. Cookie 和存储限制：**一个源不能读取另一个源的 Cookie、LocalStorage 等。
- 3. 网络请求限制：**跨域的 Ajax 请求会受到限制，无法正常获取响应内容。

由于同源策略的存在。当您使用 `iframe` 跨域集成活字格页面的时候，必然会出现如下现象：

尽管在您的 `iframe` 中可以查看到活字格的页面，但是经过活字格的授权后，您无法正确跳转到鉴权成功后的页面上。活字格的管理控制台中的应用日志也会报错如下信息：

```
1 Invalid request with no permission.
```

text

这是因为跨域集成触发了同源策略中 Cookie 禁止传递的限制。

解决方案

接口跨域请求

对于接口层面的跨域请求，您可以在服务器侧设置合适的 HTTP 响应头，允许特定来源的请求，也就是 CORS（跨域资源共享）。例如：

- `Access-Control-Allow-Origin`：指定允许的来源（可以是具体域名，也可以是 `*` 表示允许所有来源）。
- `Access-Control-Allow-Methods`：指定允许的 HTTP 方法（如 GET, POST, PUT）。
- `Access-Control-Allow-Headers`：指定允许的自定义请求头。

一个允许来自 `https://example.com` 的跨域请求示例：

```
1 Access-Control-Allow-Origin: https://example.com
2 Access-Control-Allow-Methods: GET, POST
3 Access-Control-Allow-Headers: Content-Type
```

http

TIP

您可以将 CORS 配置在活字格的管理控制台中，也可以配置在活字格的前置 Nginx 上。

- 活字格的管理控制台：

The screenshot shows the FGCLiveConsole management control panel. On the left sidebar, under the '设置' (Settings) section, the '安全设置' (Security Settings) option is selected. In the main content area, there are two sections: '所有应用的默认设置' (Default settings for all applications) and '所有应用的IP限制设置' (IP restriction settings for all applications). The '所有应用的默认设置' section contains two configuration boxes: 'Http Referer 允许列表' (Allow list for 'Referer' header) and 'iframe 跨域策略' (Cross-domain strategy for iframes). The 'Http Referer 允许列表' box contains the URL patterns 'https://www.example.com*' and 'http://localhost*'. The 'iframe 跨域策略' box contains the HTTP response headers 'Referrer-Policy:strict-origin-when-cross-origin' and 'X-Content-Type-Options:nosniff'. A red circle with the number '2' is placed over the 'iframe 跨域策略' box. The '所有应用的IP限制设置' section contains two configuration boxes: '可访问应用的IP地址列表(白名单)' (White list of IP addresses for accessible applications) and '不可访问应用的IP地址列表(黑名单)' (Black list of IP addresses for inaccessible applications). Both boxes allow for multiple IP addresses separated by newlines and support CIDR notation.

- Nginx:

```

1   location / {
2       add_header Access-Control-Allow-Origin *;
3       add_header Access-Control-Allow-Methods "GET, POST, OPTIONS, PUT, DELETE";
4       add_header Access-Control-Allow-Headers "Content-Type, Authorization";
5       if ($request_method = 'OPTIONS') {
6           add_header Access-Control-Allow-Origin *;
7           add_header Access-Control-Allow-Methods "GET, POST, OPTIONS, PUT, DELETE";
8           add_header Access-Control-Allow-Headers "Content-Type, Authorization";
9           return 204;
10      }
11
12      proxy_pass http://backend;
13  }

```

iframe 跨域集成

此类场景在浏览器的安全策略中属于高危行为。因此要求集成与被集成侧必须使用 HTTPS。此外，您仍然需要在活字格侧开启 `samesite`，才能保证 cookie 的正常传递。

- 活字格的应用必须为 **HTTPS** 且绑定认证证书。如果应用本身是 **HTTP**，使用 Nginx 代理为 **HTTPS**，会出现不可预估的问题。

- 集成的应用本身也必须为 **HTTPS** 且绑定认证证书。
- 活字格 **应用内部的高级设置** 中请开启 `samesite`。

NOTE

活字格管理控制台的 iframe 跨域策略默认允许在任意页面的 iframe 中显示。如果您有额外的安全需求，请自行调整策略。

- 如果您的活字格应用前置了 Nginx 网关，为确保数据传递正常，请在 Nginx 上配置 `samesite`：

1

```
proxy_cookie_path / ";" SameSite=None; Secure";
```

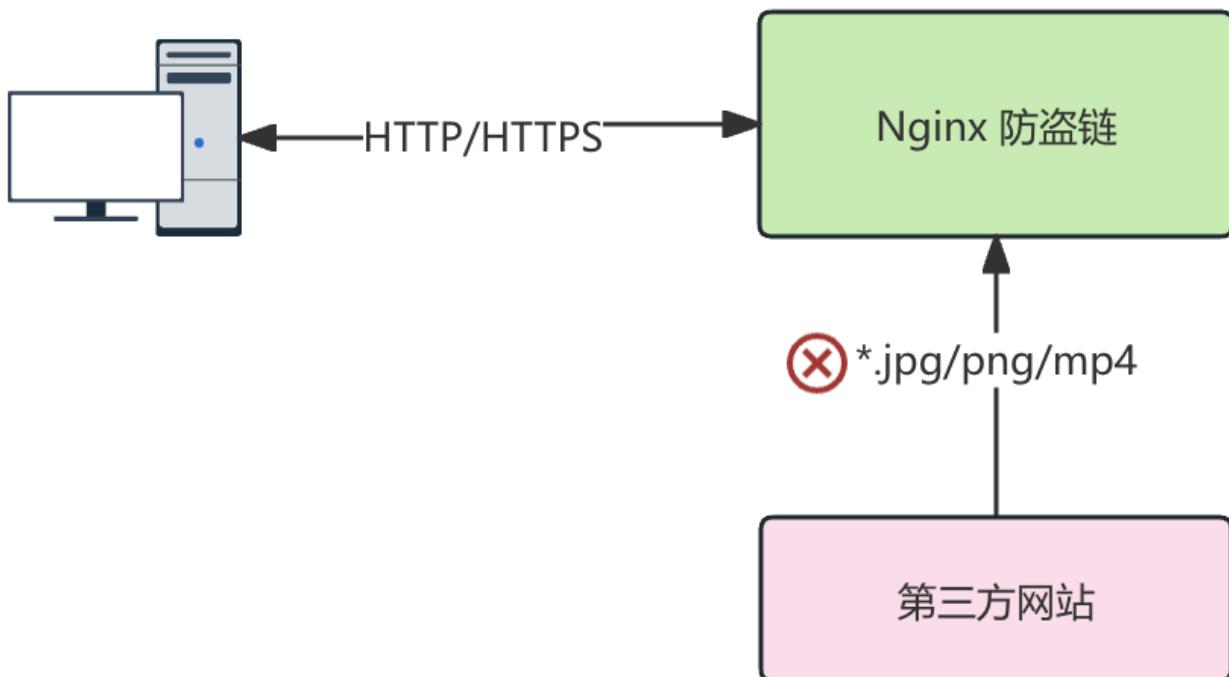
nginx

防盗链

防盗链是一个用于防止其他网站盗用自身网站资源的安全机制。

使用场景

如果有第三方的网站未经授权直接引用我们网站的图片、视频等资源，当用户访问这些第三方网站时，实际上加载的资源是来自于我们的网站，从而导致服务器的资源开销增加。



配置

在 Nginx 中，通过配置 `Referer` 请求头的验证，可以实现有效的防盗链。

```

1  server {
2      listen 80;
3      server_name example.com;
4

```

nginx

```

5   location /images/ {
6     valid_referers none blocked example.com *.example.com;
7
8     if ($invalid_referer) {
9       return 403; # 禁止访问
10    }
11
12    root /var/www/html;
13  }
14 }
```

上述示例实现了阻止非授权来源的图片访问。其中，授权列表通过 `valid_referers` 指令定义了允许的来源列表。如果请求的 `Referer` 无效，则直接 `403` 进制访问。如果您希望为非法请求返回一个替代的图片提示，可以使用 `rewrite` 指令重写资源地址。

```

1 server {
2   listen 80;
3   server_name example.com;
4
5   location /images/ {
6     valid_referers none blocked example.com *.example.com;
7
8     if ($invalid_referer) {
9       rewrite ^/images/.*$ /images/anti_hotlink.jpg last;
10    }
11
12    root /var/www/html;
13  }
14 }
```

nginx

HTTPS

HTTPS (HyperText Transfer Protocol Secure) 是基于 HTTP 的安全通信协议，通过 SSL/TLS (安全套接字层/传输层安全性协议) 对数据进行加密，确保数据在客户端和服务器之间的传输过程中安全。

相比于 HTTP，HTTPS 的数据传输都是经过加密的，这就保证了数据在客户端与服务器之间的传输过程中安全。此外，HTTPS 还具备身份验证、防篡改、防窃听等多个安全优势。因此，现代浏览器的很多涉及到安全的功能都强制要求网站是 HTTPS。

为活字格网站启用 HTTPS。

1. 获取数字认证证书。

认证证书确保通信的安全性和身份的可信性，因此，是 HTTPS 的必备文件。如何获取证书，可以参考[这里](#)。

2. 进入活字格管理控制台，「设置 → Htts 证书」。

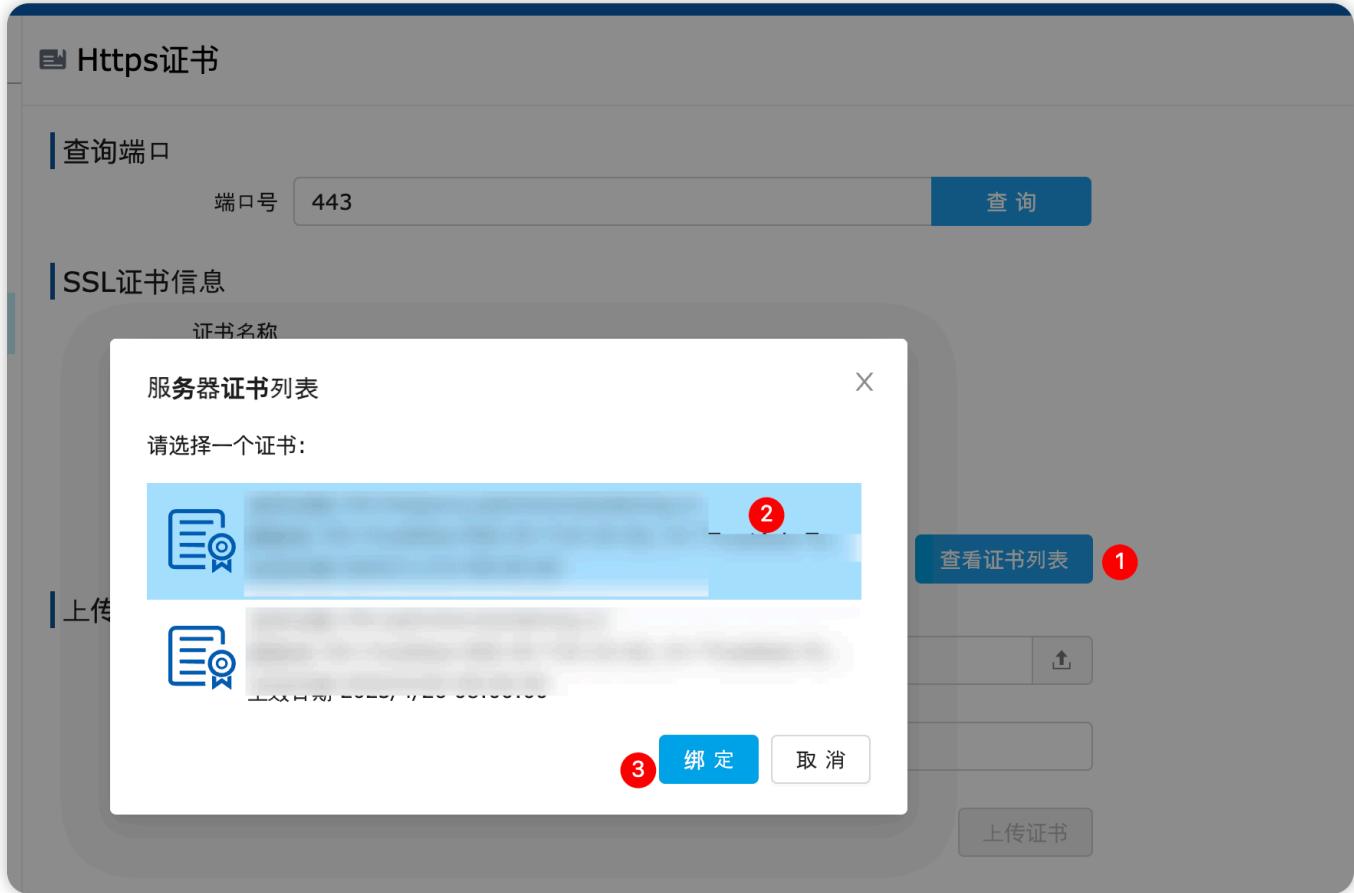
3. 在「上传证书到服务器」中进行证书的上传。

IMPORTANT

活字格管理控制台仅支持 `.pfx` 格式的证书。

4. 在「查询端口」处搜索您需要配置成 HTTPS 的应用端口号。

5. 为应用端口号绑定证书。「查看证书列表 → 选择证书，点击绑定」



使用 Nginx 代理 HTTPS

如果您的活字格应用有使用 Nginx 做反向代理，可以直接将证书绑定在 Nginx 上，这样无需为每个活字格应用都绑定证书，也可以实现 HTTPS 的访问。

```

1  server {
2      listen 80;
3      server_name example.com www.example.com;
4      return 301 https://$host$request_uri; # 强制 HTTP 跳转到 HTTPS
5  }
6
7  server {
8      listen 443 ssl;
9      server_name example.com www.example.com;
10
11     ssl_certificate /path/to/fullchain.pem; # 证书文件
12     ssl_certificate_key /path/to/privkey.pem; # 私钥文件
13
14

```

```
15      ssl_protocols TLSv1.2 TLSv1.3; # 启用安全的 TLS 协议
16      ssl_ciphers HIGH:!aNULL:!MD5; # 配置加密套件
17
18      root /var/www/html;
19      index index.html index.htm;
20
21      location / {
22          try_files $uri $uri/ =404;
23      }
24 }
```

其中，证书文件和私钥文件的格式可以是 `pem` 格式，也可以是 `crt` 与 `key` 的格式。

IMPORTANT

为了确保用户体验无感，强烈建议配置 80 端口的重定向，强制跳转至 HTTPS 端口。

简介

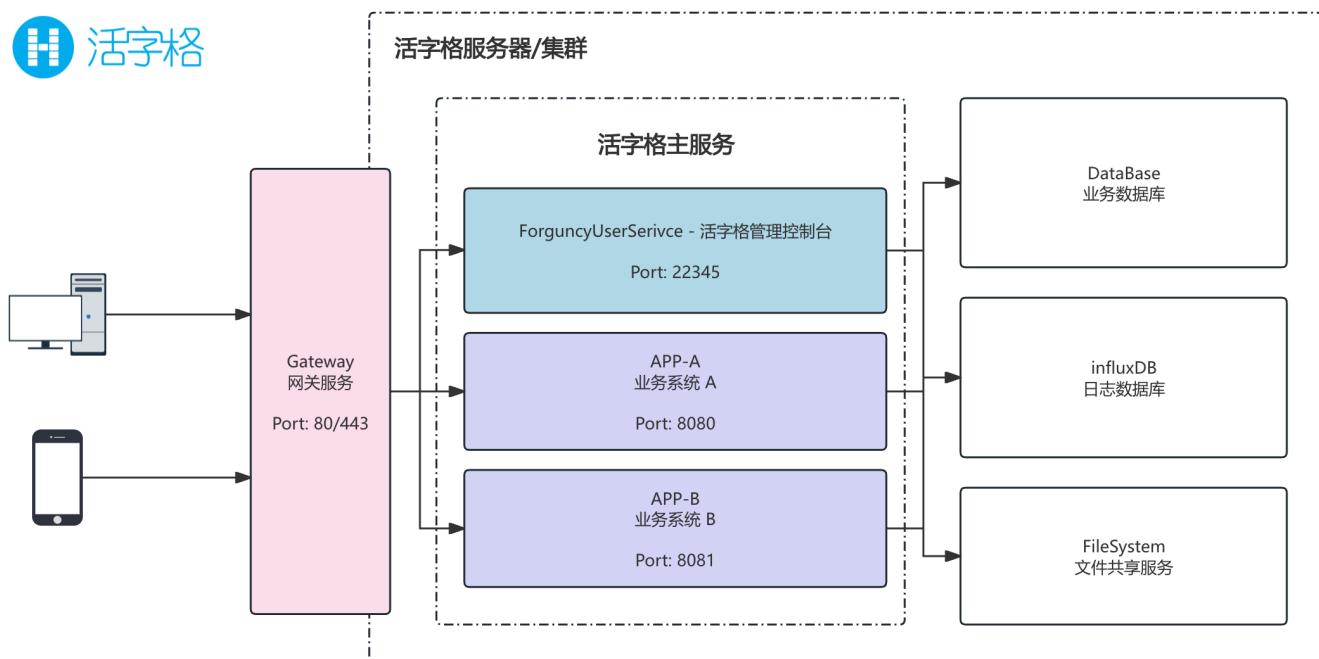
网关（Gateway）是计算机网络中的一个重要组件，作为不同网络或系统之间的桥梁。

在软件服务和分布式系统中，网关的含义更加具体，它通常指 API Gateway，即一种用于管理和控制客户端与后端服务交互的中间层。

- 在网络层面上，网关可以是一个硬件设备或软件，用于连接不同的网络协议或拓扑结构（如企业内网和互联网）。
- 在应用层面上，网关是一种服务，充当客户端请求与后端服务之间的中介，负责流量管理、安全保护、协议转换等功能。

架构

在经典的系统部署架构中，活字格服务管理器（安装在服务器上的服务，加载和运行使用设计器开发的应用）定位于应用服务器。而应用服务器在面向最终用户提供服务时，稳定性、安全性、高可用等都需要重点关注。因此，在应用服务器与最终用户之间，非常建议为您的活字格服务器架设一个网关服务。



通过网关服务，您可以实现：

- 提供统一的访问入口。终端用户只需访问网关，而无需关心后端服务的具体实现。这样后端的活字格服务可以随时变更或扩展。
- 所有的请求都会通过网关服务进入系统，方便对访问的请求和流量做相应的监控和处理。
- 只需在网关侧 [配置 HTTPS](#)，后端的所有服务便都可以对终端客户提供 HTTPS 服务了。避免了多处维护证书的繁琐。
- 通过访问不同的路径，将客户端的请求转发至不同的应用上。
- 集群场景下，可以在网关侧配置负载均衡，提升系统的性能和可靠性。
- 黑白名单的管控，可以有效放置来自外部的网络攻击。
- 更多网络场景在安全性与便利性的实践。

Nginx

[Nginx](#) 是一款高性能的 HTTP 服务器和反向代理服务器，广泛应用于静态资源托管、负载均衡和高并发环境中。它以事件驱动架构著称，能够高效处理大量并发请求，不仅适合作为反向代理隐藏后端服务，还能通过缓存和限流提升系统性能。由于其灵活的模块化设计，Nginx 还可以实现请求路由、访问控制和 SSL/TLS 加密等功能，是构建现代化 web 服务和分布式系统的核心组件之一。

在活字格的应用场景中，Nginx 也是您构建网关服务的首选之一。

安装

Nginx 提供了多平台的安装方式。

- Linux：基于您系统的版本，建议使用相应的包管理器进行下载安装。当然，如果您有指定版本号需求，Nginx 官方提供了相应的压缩包下载。
- Windows：Nginx 官方提供了.zip 包，在指定目录内解压即可。

安装完成后，需要将 Nginx 命令加入到环境变量中，并将服务设置为开机自启，避免每次手动启动服务。

使用

Nginx 的核心功能就是应对 HTTP 请求的处理。由于具体硬件、操作系统及应用场景的不同，需要 Nginx 在对 HTTP 请求的处理方法上进行不同的调整，为了应对这些差异，Nginx 提供了多种配置指令，让用户可以根据实际的软硬件及使用场景进行灵活配置。

因此，Nginx 使用的核心在于对 Nginx 配置文件的设置。根据您安装方式不同，Nginx 的配置文件目录也会有些不同。您可以进入到 Nginx 的安装目录下，寻找文件 `nginx.conf` 即可，这是 Nginx 唯一的默认配置入口。

TIP

建议您的 Nginx 配置遵循 Debian 的网页服务器配置文件标准：

- 创建的网站配置文件放置于 Nginx 安装目录下的 `sites-available` 文件夹中，并为需要启用的网站创建一份配置文件软连接到 `sites-enabled` 文件夹。
- `sites-available` 与 `sites-enabled` 可自行创建。

如果您的系统是非 Debian 系列的系统，您需要将 `nginx.conf` 配置文件中的内容修改为如下所示的 Debian 风格：

```
1 http {
2     # ...
3     include /etc/nginx/conf.d/*.conf;
4     include /etc/nginx/sites-enabled/*;
5 }
```

nginx

- `include` 为 Nginx 提供的配置指令，用于引入其他配置文件的内容。

为了便于维护，建议每个网站都可以在目录 `sites-available` 中创建对应的 server 配置文件，然后将配置软链至 `sites-enabled` 中。通过 `include` 指令将 `sites-enabled` 中的所有开启的配置文件引入到入口配置文件 `nginx.conf` 中。这样只需删除软链，就可以在不删除真实配置的情况下完成网站的快速上下线。

Nginx 提供了非常丰富的指令用于管控自身的各种能力以及相关模块的能力。您感兴趣的话可以参考官网文档，也可以访问[这里](#)以作参考。

可视化

如果您拥有多个 Nginx 服务器，或者希望通过可视化的方式对 Nginx 进行配置管理。您可以选择：

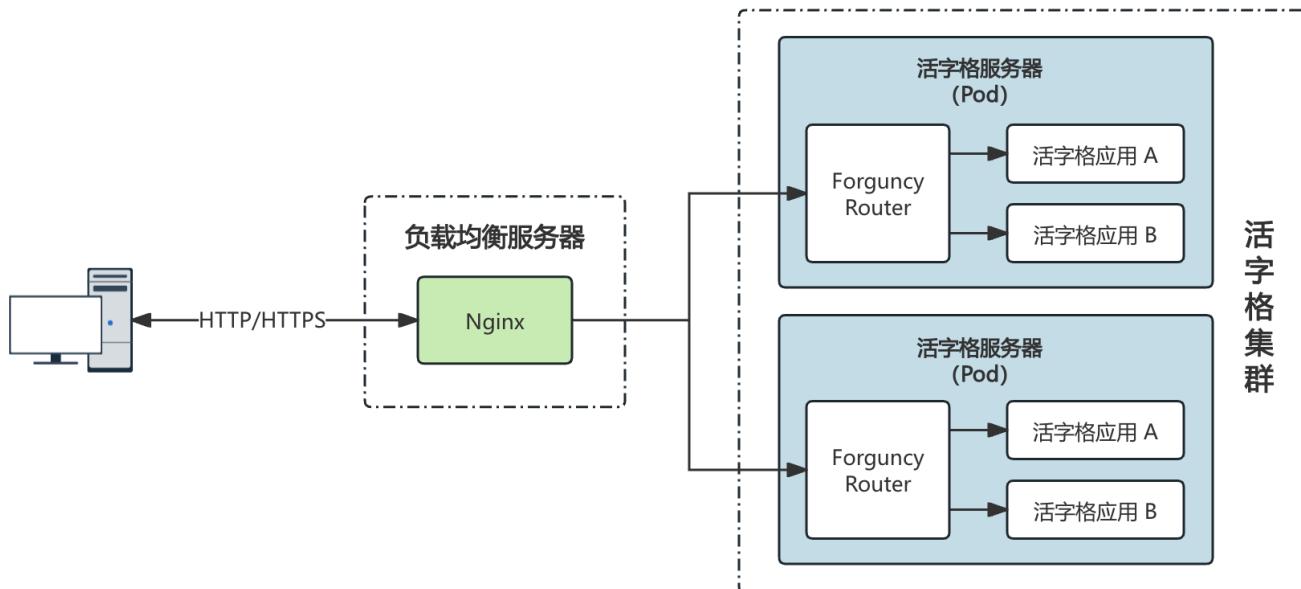
- [Nginx UI](#)
- [Nginx Proxy Manager](#)

负载均衡

将 Nginx 作为一个负载均衡器，以应对高并发的场景。

使用场景

用户访问 Nginx，Nginx 再将请求转发至后端的活字格多台服务器上，来提供活字格服务器高可用与高并发的能力。用于[活字格的集群场景](#)。



配置

Nginx 通过 `upstream` 指令来定义后端多台服务器 IP 地址和端口。之后在 `location` 中通过 `proxy_pass` 指令来指定 `upstream` 的名称。这样就可以将请求转发到定义的 `upstream` 这个组中的服务器。

```

1  upstream fgc_servers {
2      ip_hash;
3      server 192.168.1.100:31291;
4      server 192.168.1.101:31291;
5      server 192.168.1.102:31291;

```

nginx

```

6 }
7
8 server {
9   listen 80;
10  server_name _;
11  location /app {
12    proxy_pass http://fgc_servers/app;
13    proxy_set_header Host $host;
14    proxy_set_header X-Real-IP $remote_addr;
15  }
16 }

```

TIP

- `fgc_servers` 为活字格集群的节点组，其中每个 IP 都是节点的 IP，而不是 Pod 的 IP。
- `31291` 为集群的 fgc-service 对外暴露的服务端口，详情可参考[文档](#)。

Nginx 提供了丰富的流量分发算法。上述配置中高亮的 `ip_hash` 就是其中的一种。如果在 `upstream` 中不做任何配置，Nginx 会使用默认的轮询转发，即配置的多台 web 服务器，它会轮询的转发。

算法	配置指令示例	适用场景
轮询	无	后端服务器性能一致，简单场景。
加权轮询	在 <code>server</code> 后追加 <code>weight</code>	后端服务器性能不一致，需要按比例分发流量。
IP 哈希	<code>ip_hash;</code>	需要会话保持，例如用户登录。活字格集群推荐。
最少连接	<code>least_conn;</code>	请求处理时间不一致，适合动态负载分配。
一致性哈希	<code>hash \$request_uri consistent;</code>	缓存或会话场景，减少服务器变动带来的影响。
随机	<code>random two;</code>	简单的负载均衡，结合权重提供更灵活的分配。
最短响应时间	<code>least_time header;</code>	追求最快的响应速度，需要更精确的负载均衡。

基本命令

本节介绍 Nginx 使用的基本命令行命令。

Nginx 命令

1. 检查配置文件

```
1 nginx -t
```

bash

2. 热加载配置文件（无需重启服务）

```
1 nginx -s reload
```

bash

3. 强制关闭服务

```
1 nginx -s stop
```

bash

4. 优雅关闭服务（等待工作进程处理任务完成后关闭）

```
1 nginx -s quit
```

bash

其他命令

1. 查看进程号

```
1 ps -ef | grep nginx
```

bash

2.

配置指令

本节介绍的是 Nginx 配置中的指令说明。

NOTE

本节的配置样例是为了方便您理解 Nginx 配置指令的功能和使用方法，切勿将其直接用于实际生产环境。

核心进程指令

```

1  daemon on;                                nginx
# 以守护进程的方式运行Nginx
2  pid  logs/nginx.pid;                      # 主进程ID记录在logs/
3  user nobody nobody;                      # 工作进程运行用户为nobody
4  load_module "modules/ngx_http_xslt_filter_module.so"; # 加载动态模块ngx_http_xslt_filter_module.so
5  error_log  logs/error.log debug;          # 错误日志输出级别为debug
6  pcre_jit on;                            # 启用pcre_jit技术
7  thread_pool default threads=32 max_queue=65536; # 线程池的线程数为32,
8  timer_resolution 100ms;                  # 定时器周期为100毫秒
9  worker_priority -5;                     # 工作进程系统优先级为-5
10 worker_processes auto;                   # 工作进程数由Nginx自动设置
11 worker_cpu_affinity auto;                # 工作进程的CPU绑定由Nginx自动分配
12 worker_rlimit_nofile 65535;              # 所有工作进程的最大连接数
13 worker_shutdown_timeout 10s;              # 工作进程关闭等待时间
14 lock_file logs/nginx.lock;              # 互斥锁文件的位置是logs/nginx.lock
15
16 working_directory logs;                 # 工作进程工作目录是logs
17 debug_points stop;                      # 调试点模式为stop
18 worker_rlimit_core 800m;                # 崩溃文件大小为800MB
19
20 events {
21     worker_connections 65535;            # 每个工作进程的最大连接数
22     use epoll;                         # 指定事件模型为epoll
23     accept_mutex on;                  # 启用互斥锁模式的进程间通信
24     accept_mutex_delay 300ms;          # 互斥锁模式下进程等待时间
25     multi_accept on;                 # 启用支持多连接
26     worker_aio_requests 128;           # 完成异步操作最大数为128
27 }
```

```
28     debug_connection 192.0.2.0/24;          # 调试指定连接的IP地址
}

```

HTTP 指令

```
nginx
1 http {
2     resolver 192.168.2.11 valid=30s;      # 全局域名解析服务器为192.168.2.11, 30s
3     resolver_timeout 10s;                  # 域名解析超时时间为10s
4     variables_hash_max_size 1024;        # Nginx变量的hash表的大小为1024字节
5     variables_hash_bucket_size 64;       # Nginx变量的hash表的哈希桶的大小是64字节
6     types_hash_max_size 1024;           # MIME类型映射表哈希表的大小为1024字节
7     types_hash_bucket_size 64;         # MIME类型映射表哈希桶的大小是64字节
8
9     # 请求解析, HTTP全局有效
10    ignore_invalid_headers on;          # 忽略请求头中无效的属性名
11    underscores_in_headers on;        # 允许请求头的属性名中有下划线“_”
12    client_header_buffer_size 2k;      # 客户请求头缓冲区大小为2KB
13    large_client_header_buffers 4 16k;  # 超大客户请求头缓冲区大小为64KB
14    client_header_timeout 30s;        # 读取客户请求头的超时时间是30s
15    request_pool_size 4k;            # 请求池的大小是4K
16
17    merge_slashes on;                # 当URI中有连续的斜线时做合并处理
18    server_tokens off;              # 当返回错误信息时, 不显示Nginx服务的版本号
19    msie_padding on;                # 当客户端请求出错时, 在响应数据中添加注释
20
21    subrequest_output_buffer_size 8k; # 子请求响应报文缓冲区大小为8KB
22
23    lingering_close on;            # Nginx主动关闭连接时启用延迟关闭
24    lingering_time 60s;           # 延迟关闭的处理数据的最长时间是60s
25    lingering_timeout 5s;         # 延迟关闭的超时时间是5s
26    reset_timedout_connection on; # 当Nginx主动关闭连接而客户端无响应时, 在
27
28    log_not_found on;             # 将未找到文件的错误信息记录到日志中
29    log_subrequest on;            # 将子请求的访问日志记录到访问日志中
30    error_page 404 /404.html;    # 所有请求的404状态码返回404.html文件
31    error_page 500 502 503 504 /50x.html; # 所有请求的500、502、503、504状态码以
32
33    server {
```

```

54      # 监听本机的 8000 端口, 当前服务是 http 指令域的主服务, 开启 fastopen 功能并
55      listen *:8000 default_server fastopen=30 deferred reuseport backlog
56      server_name a.nginxbar.com b.nginxtest.net c.nginxbar.com a.nginxba
57      server_names_hash_max_size 1024; # 服务主机名哈希表大小为1024字节
58      server_names_hash_bucket_size 128;# 服务主机名哈希桶大小为128字节
59
60
61      # 保持链接配置
62      keepalive_disable msie6;          # 对MSIE6版本的客户端关闭保持连接机制
63      keepalive_requests 1000;         # 保持连接可复用的HTTP连接为1000个
64      keepalive_timeout 60s;           # 保持连接空置超时时间为60s
65      tcp_nodelay on;                # 当处于保持连接状态时, 以最快的方式发送数据
66
67
68      # 本地文件相关配置
69      root /data/website;            # 当前服务对应本地文件访问的根目录是/data/w
70      disable_symlinks off;          # 对本地文件路径中的符号链接不做检测
71
72      # 静态文件场景
73      location / {
74          server_name_in_redirect on; # 在重定向时, 拼接服务主机名
75          port_in_redirect on;       # 在重定向时, 拼接服务主机端口
76          if_modified_since exact;  # 当请求头中有if_modified_since属性时,
77                                  # 与被请求的本地文件修改时间做精确匹配处理
78          etag on;                  # 启用etag功能
79          msie_refresh on;          # 当客户端是msie时, 以添加HTML头信息的方式执行跳转
80          open_file_cache max=1000 inactive=20s;# 对被打开文件启用缓存支持, 缓
81          open_file_cache_errors on; # 对无法找到文件的错误元素也进行缓存
82          open_file_cache_min_uses 2; # 缓存中的元素至少要被访问两次才为活
83          open_file_cache_valid 60s; # 每60s对缓存元素与本地文件进行一次
84      }
85
86      # 上传接口的场景应用
87      location /upload {
88          alias /data/upload;          # 将upload的请求重定位到目录/data/u
89          limit_except GET {          # 对除GET以外的所有方法进行限制
90              allow 192.168.100.1;    # 允许192.168.100.1执行所有请求方法
91              deny all;               # 其他IP只允许执行GET方法
92          }
93          client_max_body_size 200m;   # 允许上传的最大文件大小是200MB
94          client_body_buffer_size 16k; # 上传缓冲区的大小是16KB
95          client_body_in_single_buffer on; # 上传文件完整地保存在临时文件中
96      }

```

```

77     client_body_in_file_only off;          # 不禁用上传缓冲区
78     client_body_temp_path /tmp/upload 1 2;# 设置请求体临时文件存储目录
79     client_body_timeout 120s;            # 请求体接收超时时间为120s
80   }
81
82   # 下载场景应用
83   location /download {
84     alias /data/upload;                # 将download的请求重定位到目录/data/
85     types { };
86     default_type application/octet-stream; # 设置当前目录所有文件默认MIME类型
87     try_files $uri @nofile;           # 当文件不存在时, 跳转到location @nofile
88     sendfile on;                   # 开启零复制文件传输功能
89     sendfile_max_chunk 1M;         # 每个sendfile调用的最大传输量为1MB
90     tcp_nopush on;                # 启用最小传输限制功能
91     aio on;                      # 启用异步传输
92     directio 5M;                 # 当文件大于5MB时以直接读取磁盘方式读取
93     directio_alignment 4096;       # 与磁盘的文件系统对齐
94     output_buffers 4 32k;          # 文件输出的缓冲区为128KB
95     limit_rate 1m;                # 限制下载速度为1MB
96     limit_rate_after 2m;          # 当客户端下载速度达到2MB时, 进入限速阶段
97     max_ranges 4096;              # 客户端执行范围读取的最大值是4096B
98     send_timeout 20s;             # 客户端引发传输超时时间为20s
99     postpone_output 2048;          # 当缓冲区的数据达到2048B时再向客户端发送
100    chunked_transfer_encoding on; # 启用分块传输标识
101  }
102
103  location @nofile {
104    index nofile.html;
105  }
106  location = /404.html {
107    internal;
108  }
109  location = /50x.html {
110    internal;
111  }
112}

```

日志

Nginx 日志分为两种主要类型：访问日志（access log）和错误日志（error log），它们记录了服务器的请求处理信息和运行状态。

位置

默认情况下，Nginx 日志文件存储在以下路径（可能因安装方式或配置不同而变化）：

- 访问日志（Access Log）：`/var/log/nginx/access.log`
- 错误日志（Error Log）：`/var/log/nginx/error.log`

可以通过配置文件查看或更改日志路径：

```
1 http {  
2     access_log /var/log/nginx/access.log;  
3     error_log /var/log/nginx/error.log;  
4 }
```

nginx

查看

- 如果需要实时查看日志，可以使用 tail 命令：

```
1 # 查看最近 20 行日志  
2 tail -n 20 /var/log/nginx/access.log  
3  
4 # 实时跟踪日志输出  
5 tail -f /var/log/nginx/access.log
```

bash

- 如果需要查看完整日志，可以使用 less 或者 cat 命令。

```

1 # 分页查看
2 less /var/log/nginx/access.log
3
4 # 全部显示
5 cat /var/log/nginx/access.log

```

bash

NOTE

如果日志文件内容较多，更推荐将日志复制到本地，使用文本编辑器打开检索。

分析

Nginx 日志中依照配置的格式，记录了请求所包含的信息。您可以通过如下命令统计常见的分析数据：

- 统计每个 IP 的请求数量：

```

1 awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -nr

```

bash

- 按状态码统计数量：

```

1 awk '{print $9}' /var/log/nginx/access.log | sort | uniq -c | sort -nr

```

bash

- 查看特定时间的日志：

```

1 grep "27/Dec/2024:14" /var/log/nginx/access.log

```

bash

- 过滤特定的状态码（如 404 错误）：

```

1 grep " 404 " /var/log/nginx/access.log

```

bash

管理

记录格式

Nginx 的日志记录格式可以自行配置，仅需修改配置文件即可：

```

1 # 访问日志格式配置
2 http {
3     log_format custom_format '$remote_addr - $remote_user [$time_local] '
4             '"$request" $status $body_bytes_sent '
5             '"$http_referer" "$http_user_agent"';
6
7     access_log /var/log/nginx/access.log custom_format;
8 }
9
10 # 错误日志级别设置
11 http {
12     error_log /var/log/nginx/error.log warn;
13 }
```

轮转压缩

`logrotate` 程序是一个日志文件管理工具。用于分割日志文件，删除旧的日志文件，并创建新的日志文件，起到“转储”作用。Linux 系统默认安装 `logrotate`。

使用 `logrotate` 管理 Nginx 日志文件，避免日志过大：

```

1 sudo vim /etc/logrotate.d/nginx
```

添加一下内容：

```

1 /var/log/nginx/*.log {
2     daily
3     missingok
4     rotate 7
5     compress
```

```
7     delaycompress
8     notifempty
9     create 0640 nginx adm
10    sharedscripts
11    postrotate
12        [ -s /run/nginx.pid ] && kill -USR1 `cat /run/nginx.pid`
13    endscript
14 }
```

上述配置表示每天轮转 Nginx 日志文件，最多保留 7 个，支持压缩旧日志并确保服务不中断，通过 `postrotate` 通知 Nginx 重新加载日志文件。

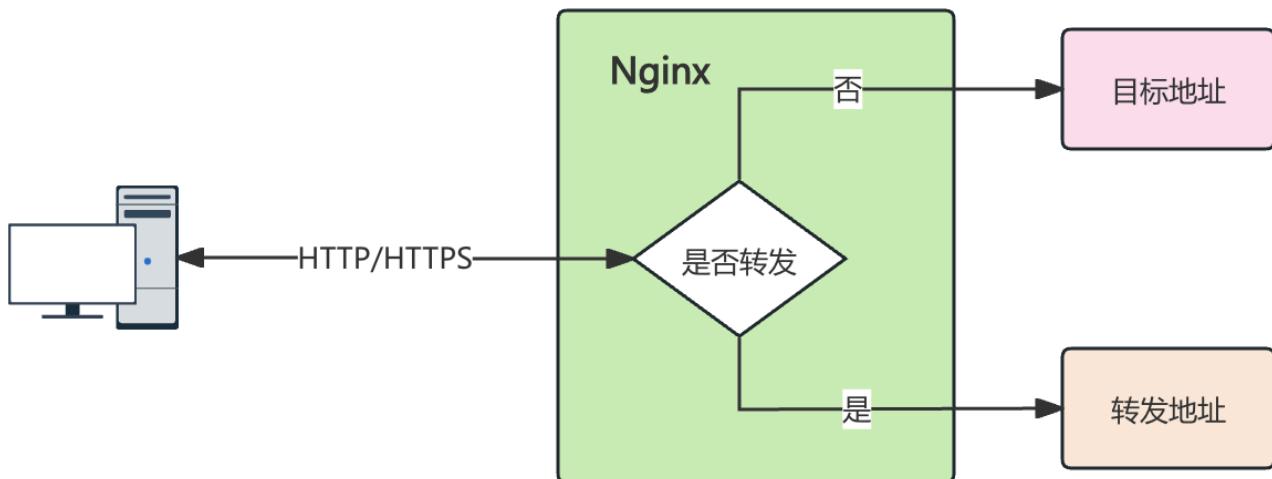
重定向

重定向（Redirect）是指服务器向客户端发送一个特殊的响应指令，告知浏览器或客户端应重新请求另一个 URL。

使用场景

通过重定向，用户的请求会被转发到指定的新地址。这是活字格应用长期维护中必然会遇到的场景，诸如：

- 网站迁移，希望老客户访问服务的地址不变，将旧地址的流量导入到新地址上，一般用在网站升级、域名更新等场景。
- HTTPS 强制跳转，强制用户访问 HTTPS 版本的站点，但是客户本身的访问行为无感。
- URL 规范化，避免 URL 重复索引，例如将 `www.example.com` 统一重定向至 `example.com`。
- 页面移动，将旧的页面地址重定向到新地址上，避免用户在访问时出现页面无法访问的错误。



配置

Nginx 提供 `rewrite` 指令实现重定向功能。

```

1 # HTTPS 强制跳转
2 server {
3     listen 80;
4     server_name example.com;
5     return 301 https://example.com$request_uri;
6 }
7
8 # 将旧站点转移到新站点上
9 server {
10    listen 80;
11    server_name oldsite.com;
12    return 301 https://newsite.com$request_uri;
13 }
14
15 # 将 www.example.com 统一为 example.com 访问
16 server {
17     server_name www.example.com;
18     return 301 https://example.com$request_uri;
19 }
```

其中：

- `301` 是 HTTP 请求中重定向的状态码，用于告知客户端该请求被重定向，资源永久移动到新地址。搜索引擎会更新索引，新地址会被缓存。
- `$request_uri` 为客户端请求的完整原始 URI。重定向时需要将客户端的请求路径和查询参数原封不动地转发到新地址。

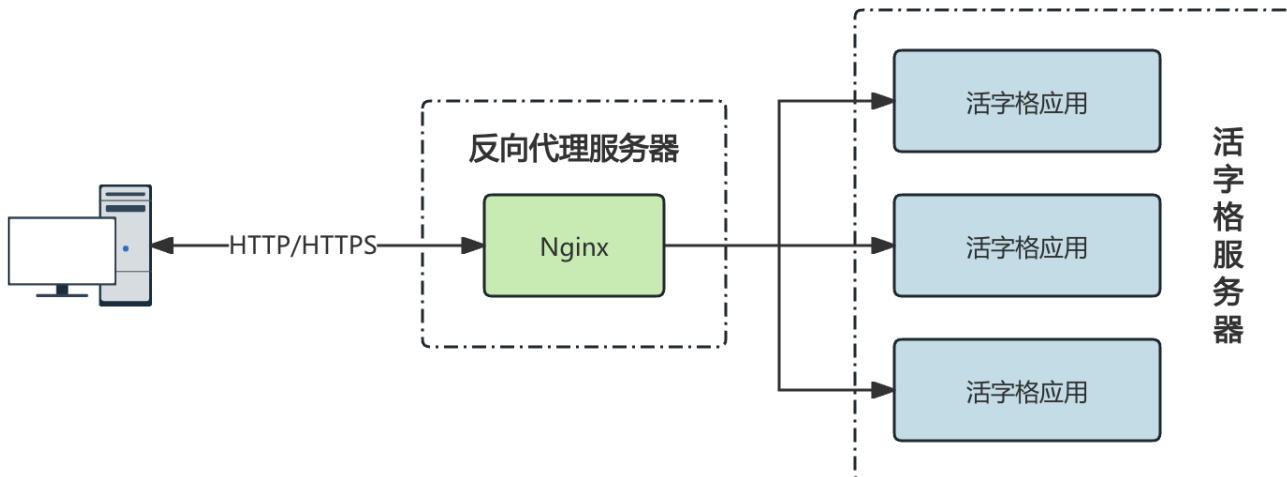
反向代理

使用反向代理，也是 Nginx 的一种常见的使用方法，也是活字格应用场景中使用 Nginx 的主要场景。

使用场景

Nginx 作为网站的访问入口，用户先访问 Nginx，Nginx 再将请求转发至活字格服务器，接收到活字格服务器的返回结果后后再响应给客户。

活字格支持在一台服务器上发布多个应用，使用反向代理的目的是为了隐藏后端的真实服务，统一的对外暴露访问，提高活字格服务器的安全性。



配置

反向代理的配置也很简单。只需要在 `location` 块中，通过 `proxy_pass` 指令来指定后端访问地址就可以了。

```

1  server {
2      listen 80;
3      server_name _;
4      location /UserService {

```

nginx

```

5     proxy_pass http://192.168.1.10:22345/UserService;
6     proxy_set_header Host $host;
7     proxy_set_header X-Real-IP $remote_addr;
8 }
9
10    location /app {
11        proxy_pass http://192.168.1.10:8080/app;
12        proxy_set_header Host $host;
13        proxy_set_header X-Real-IP $remote_addr;
14    }
15 }
```

上述配置针对于活字格的管理控制台服务和 app 应用做了两个反向代理。

- 假设活字格的服务地址是 `192.168.1.10`。
- 当用户在地址栏输入的地址是 `/UserService` 时，会被代理到活字格的管理控制台路径。
- 当用户在地址栏输入的地址是 `/app` 时，会被代理到活字格应用 app 上。这个应用发布的端口是 8080。

NOTE

在代理请求时更好地传递客户端的真实信息，确保后端服务器处理逻辑的准确性和日志记录的完整性。需要在 Nginx 上使用 `proxy_set_header` 指令显示的设置头信息：

- `Host`：识别请求是针对哪个主机名的。
- `X-Real-IP`：获取客户端的真实请求 IP 地址。

如果您需要处理多级代理的情况，还可以使用如下设置：

```
1 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

nginx

认证证书

认证证书（Digital Certificate）是一种电子文档，用于证明实体（如服务器、个人或组织）的身份，由权威机构（CA，即证书颁发机构）签发。认证证书广泛应用于 HTTPS、邮件加密、代码签名等场景，确保通信的安全性和身份的可信性。其中，用于 HTTPS 的证书被称为 SSL/TLS 证书。

公开认证证书

从某种程度来说，认证证书就是做担保的中间人，由颁发证书的机构来确保网站可信。因此，在公网上发布的证书必须是由权威机构进行签发。

各大云厂商都提供了证书的申请业务。证书的申请有免费与付费两种模式。

比较项	免费证书	付费证书
安全性	基础加密，适合简单 HTTPS 使用。	提供基础加密和企业验证，增强用户信任。
续期	有效期通常为 90 天，需要频繁续期。	有效期为 1-2 年，可选择自动续期服务。
身份验证	仅验证域名所有权。	可验证企业或组织身份（OV/EV）。
兼容性	兼容性稍弱，部分旧设备可能不支持。	更全面的兼容性。
技术支持	无技术支持，需自行解决问题。	提供专业支持和服务保障。

NOTE

一般来说，免费证书就能满足基本的 HTTPS 的使用场景。有效期较短的问题，也可以通过一些自动化工具（Let's Encrypt 的 [Cerbot](#)）实现自动续期。

自认证证书

在内网环境下，通过网络进行认证的公开认证可能会影响，因此，您可以考虑使用自认证的方式，自行签发证书，从而实现内网环境下的 HTTPS 请求。

IMPORTANT

确保已在服务器上安装 Nginx 服务。

OpenSSL

自签名证书需要依赖服务 [OpenSSL](#)，您可以通过包管理器直接进行安装。

生成自签名证书与私钥

1. 生成私钥文件：

```
1 openssl genrsa -out private.key 2048
```

bash

2. 生成证书签名请求 (CSR)：

```
1 openssl req -new -key private.key -out cert.csr -subj "/C=CN/ST=Beijing/L=B
```

bash

其中：

- `private.key` 是第一步生成的私钥。
- `-subj` 提供了证书的详细信息，避免交互式填写。当然，您也可以取消这个参数，通过交互的方式填写相关信息。
 - `/C=CN`：国家代码，例如中国为 CN，美国为 US。
 - `/ST=State`：州或省的名称，例如 Beijing。
 - `/L=City`：城市的名称，例如 Beijing。
 - `/O=Organization`：组织名称，例如 YourCompany。
 - `/OU=Unit`：组织部门，例如 IT Department。
 - `/CN=yourdomain.com`：通用名称，通常填写域名或 IP。对于内网，可以直接写内网域名或 IP 地址。

3. 生成自签名证书：

```
1 openssl x509 -req -days 365 -in cert.csr -signkey private.key -out cert.crt
```

bash

其中：

- `-x509`：指定生成的是一个自签名证书。
- `-nodes`：表示生成的私钥不需要加密。
- `-days`：指定证书的有效期（单位为 `天`），此处设置为一年，可自行调整。
- 此命令需要指定私钥与签名请求，也就是前两步生成的结果。

至此，我们已经获得了 Nginx 配置 HTTPS 所需要的 crt 与 key 文件。

4. 如果您需要为活字格应用直接配置证书，需要将证书和私钥合并生成 PFX 文件：

```
1     openssl pkcs12 -export -out certificate.pfx -inkey private.key -in cert.crt
```

bash

这一步会提示你输入导出密码，请记住这个密码。如果您不希望设置密码，也可以直接回车即可。

NOTE

自认证的证书是不会被浏览器信任的。因此客户端尽管可以正常访问绑定了自认证证书的 HTTPS 网站，但是会向客户提示证书不可信的警告。这是正常现象。客户端可以选择自行配置浏览器进行证书信任，从而规避该警告。

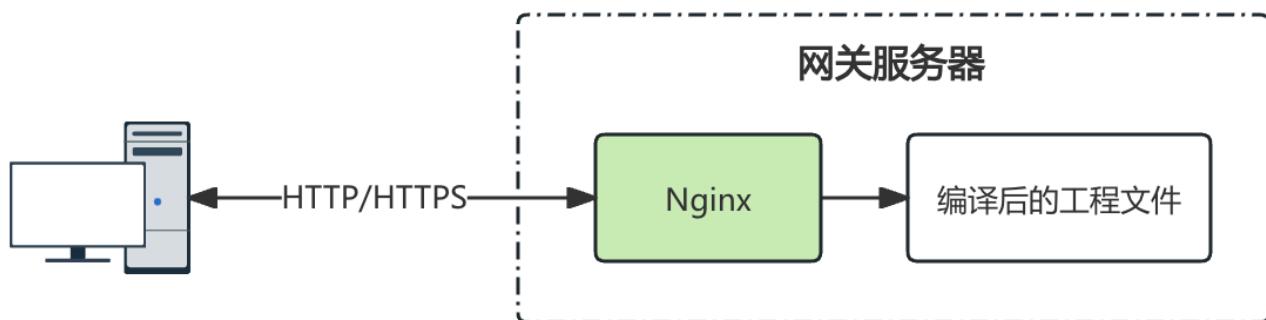
除非特别情况，生产环境下均建议使用受信任的 CA 机构颁发的证书。

Web 服务器

将 Nginx 作为一台 Web 服务器来提供静态网站服务，是 Nginx 最常见的使用方法。

使用场景

对于活字格的应用来说，活字格服务器程序本身就是一台 web 服务器，无需 Nginx 提供相应的静态网站服务。但是在实际的实施场景中，经常会遇到前端页面使用 Vue / React 等框架开发前端页面，活字格负责后端的业务服务。此时，前端页面需要构建一个 Web 服务才能为最终客户提供页面访问的能力。这时，Nginx 的价值就可以提现出来。事实上，这也是现在广大前端项目广泛使用 Nginx 的原因。



配置

Nginx 作为 Web 服务器的配置非常简单，只需要在 `location` 块中通过 `root` 指令来指定网站根目录即可。

```

1  server {
2      listen 80;
3      server_name _;
4      location / {
5          root /var/www/root;
6          index index.html index.htm;
7      }
8  }

```

nginx

```
    }
}
```

如果需要配置 HTTPS，只需要监听端口从 `80` 变更为 `443`，并配置 SSL 的两个参数指定证书和私钥就可以了。

```
nginx
1 server {
2     listen 443;
3     server_name _;
4     ssl_certificate /path/to/certificate.crt;
5     ssl_certificate_key /path/to/private-key.key;
6     location / {
7         root /var/www/root;
8         index index.html index.htm;
9     }
10 }
```

NOTE

SSL 证书可以通过各大云服务厂商进行申请。如果您是内网环境，也可以考虑使用[自认证证书](#)，自行构建证书和密钥。