Nginx 极简教程

这是一个 Nginx 极简教程，目的在于帮助新手快速入门 Nginx

目　录

致谢

当前文档 《Nginx 极简教程》 由 进击的皇虫 使用 书栈网(BookStack.CN) 进行构建，生成于 2020-05-02。

书栈网仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈网难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到书栈网，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到书栈网获取最新的文档，以跟上知识更新换代的步伐。

内容来源：Zhang Peng https://github.com/dunwu/nginx-tutorial

文档地址：http://www.bookstack.cn/books/dunwu-nginx-tutorial

书栈官网：https://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

nginx-tutorial

> **nginx-tutorial** 是一个 Nginx 极简教程。

开始阅读

Nginx 教程

> 项目同步维护在 github | gitee
>
> 电子书 | 电子书（国内）

- Nginx 快速教程
- Nginx 运维
- Nginx 配置
- Nginx 问题

Nginx 简介

Ngnix 特点

- 模块化设计：良好的扩展性，可以通过模块方式进行功能扩展。
- 高可靠性：主控进程和 worker 是同步实现的，一个 worker 出现问题，会立刻启动另一个 worker。
- 内存消耗低：一万个长连接（keep-alive），仅消耗 2.5MB 内存。
- 支持热部署：不用停止服务器，实现更新配置文件，更换日志文件、更新服务器程序版本。
- 并发能力强：官方数据每秒支持 5 万并发；
- 功能丰富：优秀的反向代理功能和灵活的负载均衡策略

Nginx 功能

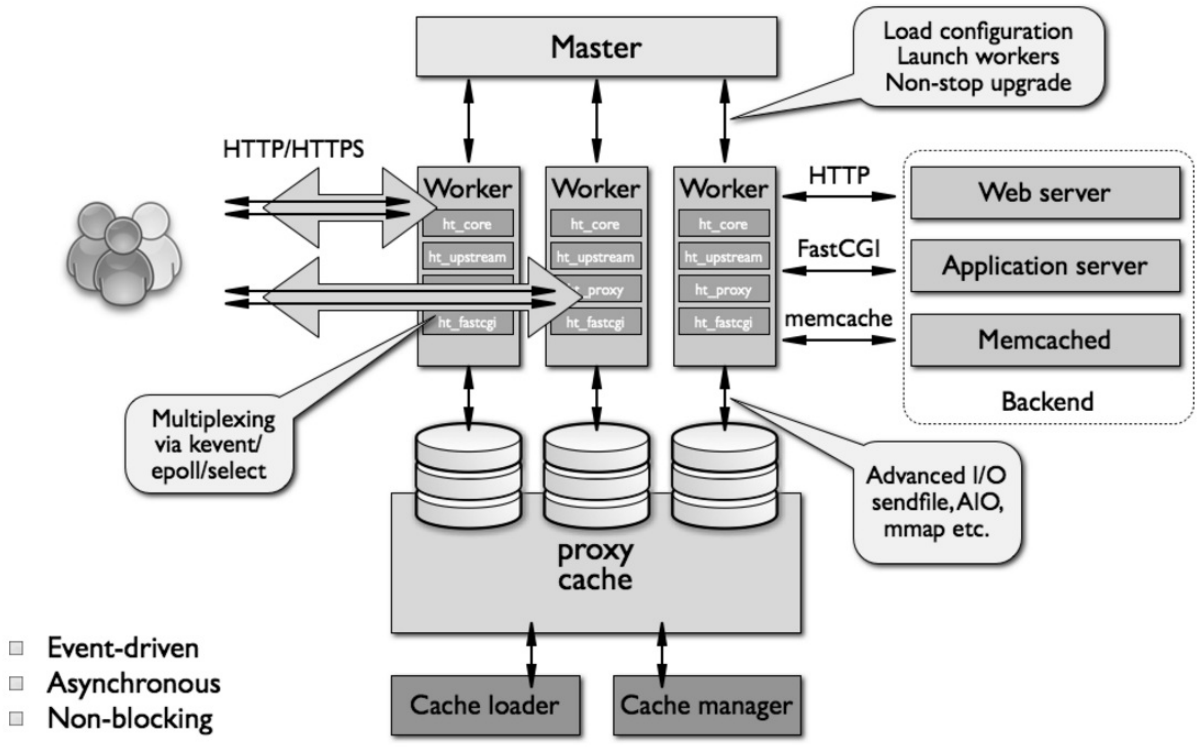
- 支持静态资源的 web 服务器。
- http,smtp,pop3 协议的反向代理服务器、缓存、负载均衡；
- 支持 FASTCGI（fpm）
- 支持模块化，过滤器（让文本可以实现压缩，节约带宽），ssl 及图像大小调整。
- 内置的健康检查功能
- 基于名称和 ip 的虚拟主机
- 定制访问日志
- 支持平滑升级
- 支持 KEEPALIVE
- 支持 url rewrite
- 支持路径别名
- 支持基于 IP 和用户名的访问控制。
- 支持传输速率限制，支持并发数限制。

Nginx 性能

Nginx 的高并发，官方测试支持 5 万并发连接。实际生产环境能到 2-3 万并发连接数。10000 个非活跃的 HTTP keep-alive 连接仅占用约 2.5MB 内存。三万并发连接下，10 个 Nginx 进程，消耗内存 150M。淘宝 tengine 团队测试结果是“24G 内存机器上，处理并发请求可达 200 万”。

Ngnix 架构

主从模式



Nginx 采用一主多从的主从架构。

但是这里 master 是使用 root 身份启动的，因为 nginx 要工作在 80 端口。而只有管理员才有权限启动小于低于 1023 的端口。master 主要是负责的作用只是启动 worker，加载配置文件，负责系统的平滑升级。其它的工作是交给 worker。那么当 worker 被启动之后，也只是负责一些 web 最简单的工作，而其他的工作都是有 worker 中调用的模块来实现的。

模块之间是以流水线的方式实现功能的。流水线，指的是一个用户请求，由多个模块组合各自的功能依次实现完成的。比如：第一个模块只负责分析请求首部，第二个模块只负责查找数据，第三个模块只负责压缩数据，依次完成各自工作。来实现整个工作的完成。

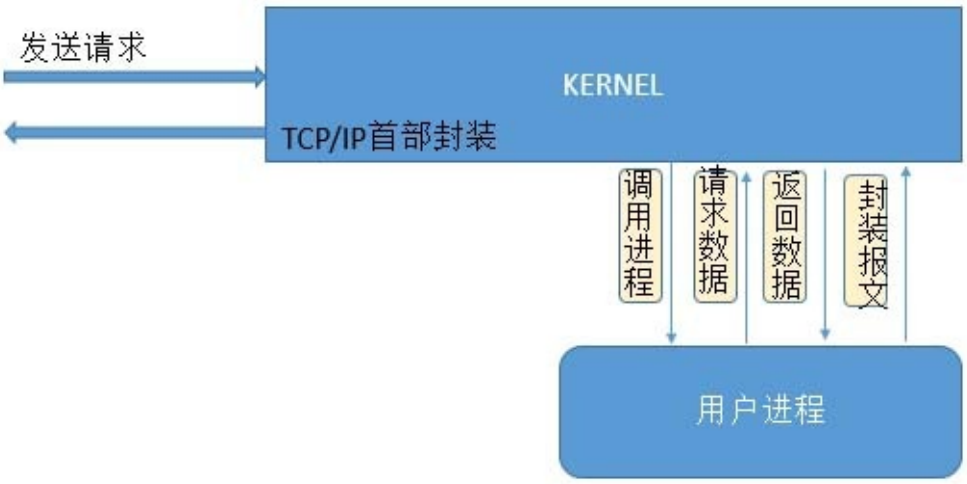
他们是如何实现热部署的呢？其实是这样的，我们前面说 master 不负责具体的工作，而是调用 worker 工作，他只是负责读取配置文件，因此当一个模块修改或者配置文件发生变化，是由 master 进行读取，因此此时不会影响到 worker 工作。在 master 进行读取配置文件之后，不会立即的把修改的配置文件告知 worker。而是让被修改的 worker 继续使用老的配置文件工作，当 worker 工作完毕之后，直接当掉这个子进程，更换新的子进程，使用新的规则。

sendfile 机制

Nginx 支持 **sendfile** 机制。

所谓 Sendfile 机制，是指：用户将请求发给内核，内核根据用户的请求调用相应用户进程，进程在处理时需要资源。此时再把请求发给内核（进程没有直接 IO 的能力），由内核加载数据。内核查找到

数据之后，会把数据复制给用户进程，由用户进程对数据进行封装，之后交给内核，内核在进行 tcp/ip 首部的封装，最后再发给客户端。这个功能用户进程只是发生了一个封装报文的过程，却要绕一大圈。因此 nginx 引入了 sendfile 机制，使得内核在接受到数据之后，不再依靠用户进程给予封装，而是自己查找自己封装，减少了一个很长一段时间的浪费，这是一个提升性能的核心点。



以上内容摘自网友发布的文章，简单一句话是资源的处理，直接通过内核层进行数据传递，避免了数据传递到应用层，应用层再传递到内核层的开销。

目前高并发的处理，一般都采用 sendfile 模式。通过直接操作内核层数据，减少应用与内核层数据传递。

I/O 复用机制

Nginx 通信模型采用 **I/O** 复用机制。

开发模型: epoll 和 kqueue。

支持的事件机制: kqueue、epoll、rt signals、/dev/poll 、event ports、select 以及 poll。

支持的 kqueue 特性包括 EV_CLEAR、EV_DISABLE、NOTE_LOWAT、EV_EOF，可用数据的数量，错误代码.

支持 sendfile、sendfile64 和 sendfilev;文件 AIO; DIRECTIO;支持 Accept-filters 和 TCP_DEFER_ACCEP.

以上概念较多，大家自行百度或谷歌，知识领域是网络通信（BIO,NIO,AIO）和多线程方面的知识。

Nginx 负载均衡

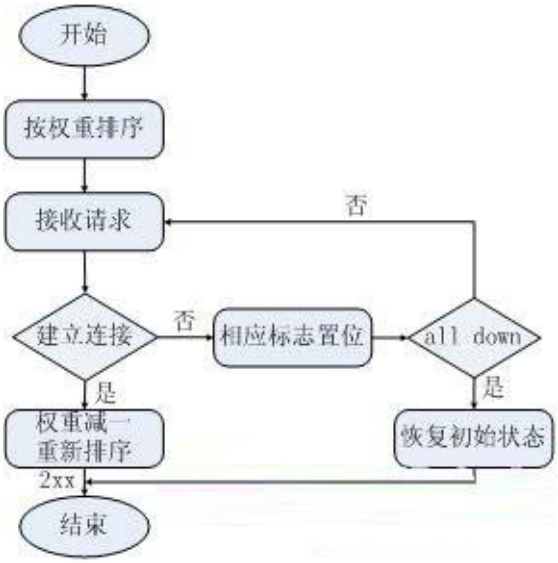
nginx 的负载均衡策略可以划分为两大类：内置策略和扩展策略。内置策略包含加权轮询和 ip

hash，在默认情况下这两种策略会编译进 nginx 内核，只需在 nginx 配置中指明参数即可。扩展策略有很多，如 fair、通用 hash、consistent hash 等，默认不编译进 nginx 内核。由于在 nginx 版本升级中负载均衡的代码没有本质性的变化，因此下面将以 nginx1.0.15 稳定版为例，从源码角度分析各个策略。

加权轮询

Nginx 支持加权轮询（**Weighted Round Robin**）负载均衡。

轮询的原理很简单，首先我们介绍一下轮询的基本流程。如下是处理一次请求的流程图：



图中有两点需要注意，第一，如果可以把加权轮询算法分为先深搜索和先广搜索，那么 nginx 采用的是先深搜索算法，即将首先将请求都分给高权重的机器，直到该机器的权值降到了比其他机器低，才开始将请求分给下一个高权重的机器；第二，当所有后端机器都 down 掉时，nginx 会立即将所有机器的标志位清成初始状态，以避免造成所有的机器都处在 timeout 的状态，从而导致整个前端被夯住。

Ip Hash

Nginx 支持 **Ip Hash** 负载均衡。

通过 Ip Hash 这种负载均衡策略，可以实现会话粘滞。

Fair

fair 策略是扩展策略，默认不被编译进 nginx 内核。其原理是根据后端服务器的响应时间判断负载情况，从中选出负载最轻的机器进行分流。这种策略具有很强的自适应性，但是实际的网络环境往往不是那么简单，因此要慎用。

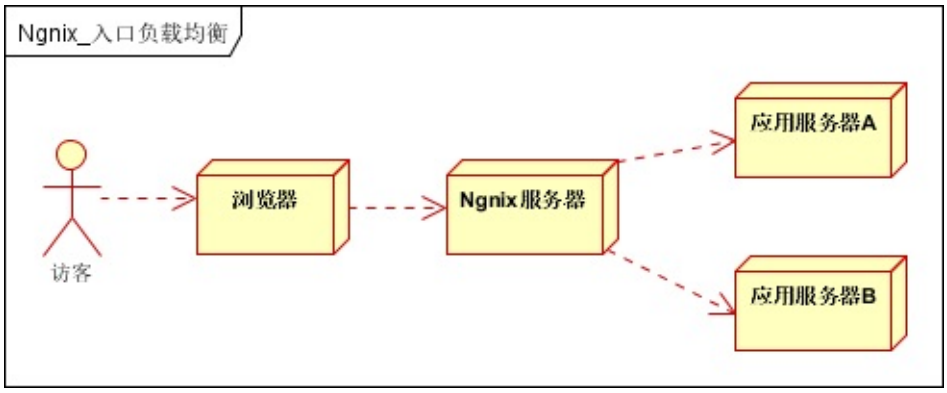
通用 Hash、一致性 Hash

这两种也是扩展策略，在具体的实现上有些差别，通用 hash 比较简单，可以以 nginx 内置的变量为 key 进行 hash，一致性 hash 采用了 nginx 内置的一致性 hash 环，可以支持 memcache。

Nginx 场景

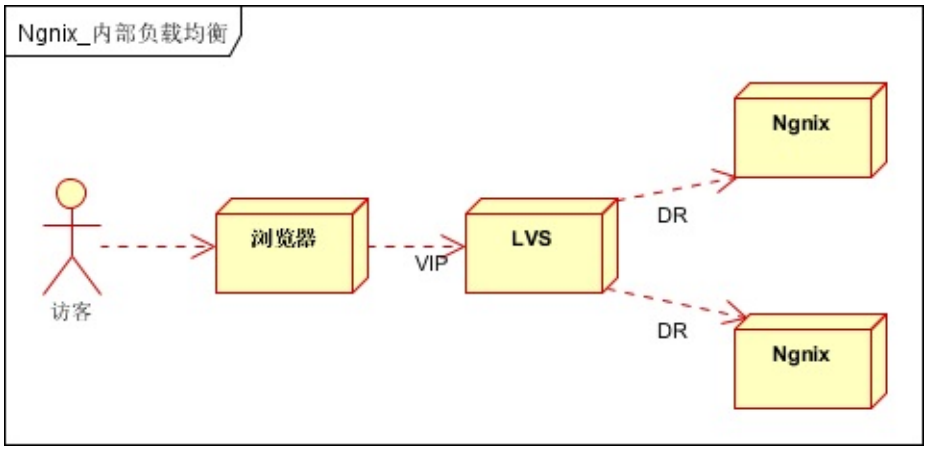
Ngnix 一般作为入口负载均衡或内部负载均衡，结合反向代理服务器使用。以下架构示例，仅供参考，具体使用根据场景而定。

入口负载均衡架构



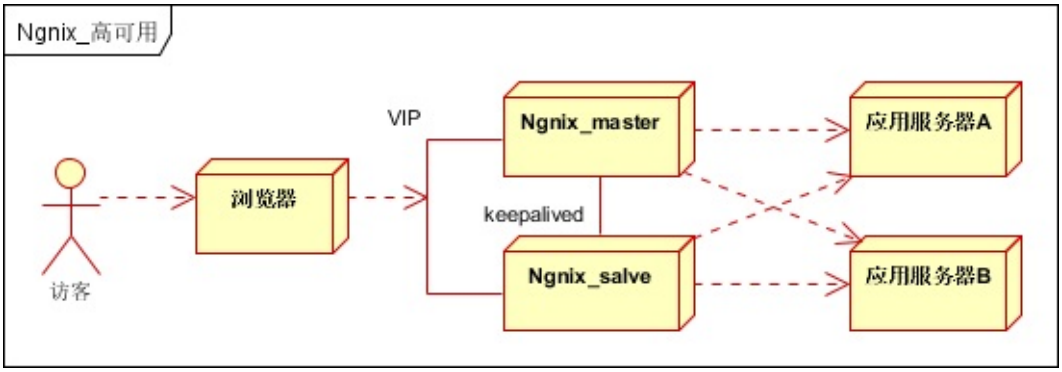
Ngnix 服务器在用户访问的最前端。根据用户请求再转发到具体的应用服务器或二级负载均衡服务器（LVS）

内部负载均衡架构



LVS 作为入口负载均衡，将请求转发到二级 Ngnix 服务器，Ngnix 再根据请求转发到具体的应用服务器。

Ngnix 高可用



分布式系统中，应用只部署一台服务器会存在单点故障，负载均衡同样有类似的问题。一般可采用主备或负载均衡设备集群的方式节约单点故障或高并发请求分流。

Ngnix 高可用，至少包含两个 Ngnix 服务器，一台主服务器，一台备服务器，之间使用 Keepalived 做健康监控和故障检测。开放 VIP 端口，通过防火墙进行外部映射。

DNS 解析公网的 IP 实际为 VIP。

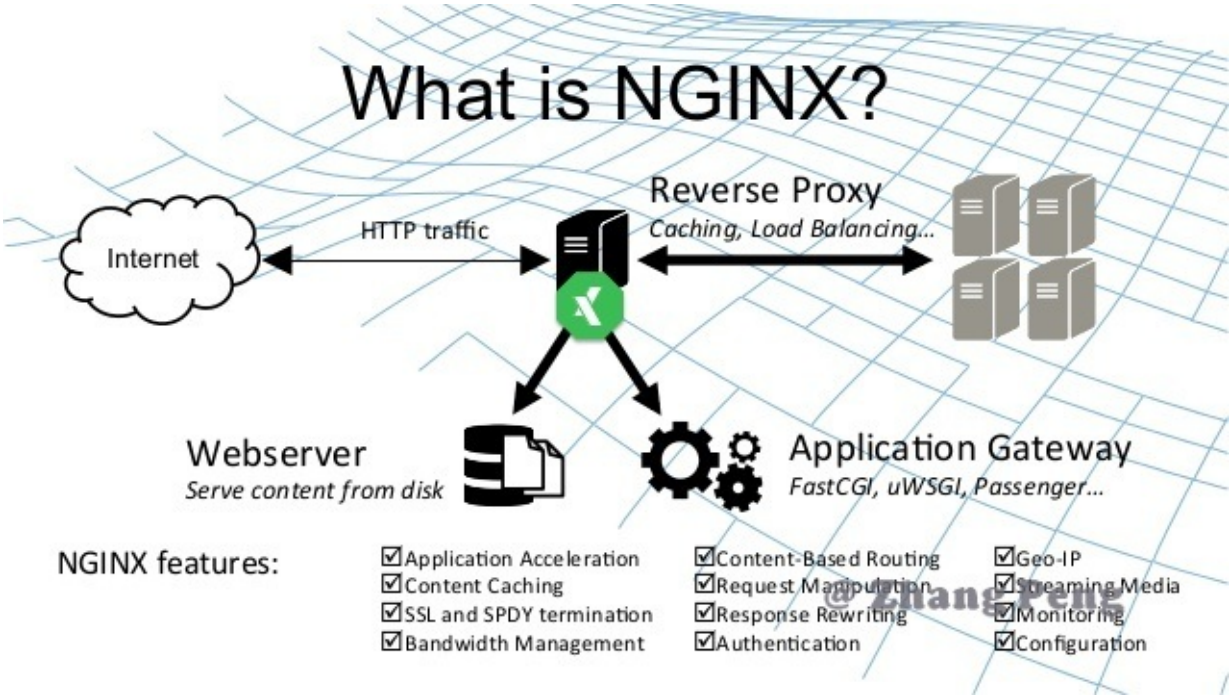
Nginx 极简教程

> 本项目是一个 Nginx 极简教程，目的在于帮助新手快速入门 Nginx。
>
> **examples** 目录中的示例模拟了工作中的一些常用实战场景，并且都可以通过脚本一键式启动，让您可以快速看到演示效果。

一、Nginx 简介

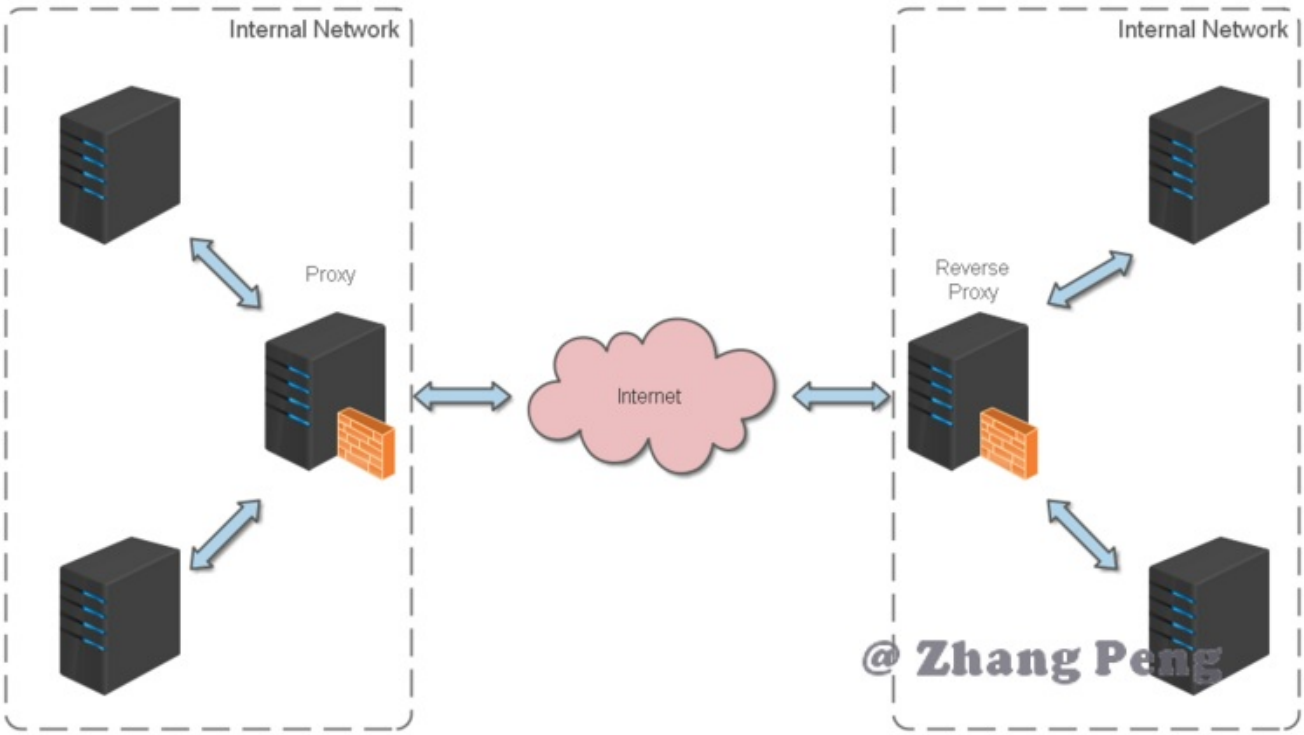
什么是 **Nginx?**

Nginx (engine x) 是一款轻量级的 Web 服务器 、反向代理服务器及电子邮件（IMAP/POP3）代理服务器。



什么是反向代理？

反向代理（Reverse Proxy）方式是指以代理服务器来接受 internet 上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给 internet 上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。


Internal Network
Proxy
Internet
Internal Network
Reverse
Proxy
@ Zhang Peng

二、Nginx 入门

> 详细安装方法请参考：Nginx 运维

nginx 的使用比较简单，就是几条命令。

常用到的命令如下：

```
1.  nginx -s stop       快速关闭Nginx，可能不保存相关信息，并迅速终止web服务。
2.  nginx -s quit       平稳关闭Nginx，保存相关信息，有安排的结束web服务。
3.  nginx -s reload     因改变了Nginx相关配置，需要重新加载配置而重载。
4.  nginx -s reopen     重新打开日志文件。
5.  nginx -c filename   为 Nginx 指定一个配置文件，来代替缺省的。
    nginx -t            不运行，仅仅测试配置文件。nginx 将检查配置文件的语法的正确性，并尝试
6.  打开配置文件中所引用到的文件。
7.  nginx -v            显示 nginx 的版本。
8.  nginx -V            显示 nginx 的版本，编译器版本和配置参数。
```

如果不想每次都敲命令，可以在 nginx 安装目录下新添一个启动批处理文件**startup.bat**，双击即可运行。内容如下：

```
1.  @echo off
2.  rem 如果启动前已经启动nginx并记录下pid文件，会kill指定进程
3.  nginx.exe -s stop
4.
5.  rem 测试配置文件语法正确性
6.  nginx.exe -t -c conf/nginx.conf
7.
8.  rem 显示版本信息
9.  nginx.exe -v
10.
11. rem 按照指定配置去启动nginx
12. nginx.exe -c conf/nginx.conf
```

如果是运行在 Linux 下，写一个 shell 脚本，大同小异。

三、Nginx 实战

我始终认为，各种开发工具的配置还是结合实战来讲述，会让人更易理解。

Http 反向代理

我们先实现一个小目标：不考虑复杂的配置，仅仅是完成一个 http 反向代理。

`nginx.conf` 配置文件如下：

> 注：***conf/nginx.conf*** 是 ***nginx*** 的默认配置文件。你也可以使用 ***nginx -c*** 指定你的配置文件

```
#运行用户
#user somebody;

#启动进程,通常设置成和cpu的数量相等
worker_processes  1;

#全局错误日志
error_log  D:/Tools/nginx-1.10.1/logs/error.log;
error_log  D:/Tools/nginx-1.10.1/logs/notice.log  notice;
error_log  D:/Tools/nginx-1.10.1/logs/info.log  info;

#PID文件，记录当前启动的nginx的进程ID
pid        D:/Tools/nginx-1.10.1/logs/nginx.pid;

#工作模式及连接数上限
events {
    worker_connections 1024;    #单个后台worker process进程的最大并发链接数
}

#设定http服务器，利用它的反向代理功能提供负载均衡支持
http {
    #设定mime类型(邮件支持类型),类型由mime.types文件定义
    include       D:/Tools/nginx-1.10.1/conf/mime.types;
    default_type  application/octet-stream;

    #设定日志
    log_format  main  '[$remote_addr] - [$remote_user] [$time_local] "$request"
'
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log    D:/Tools/nginx-1.10.1/logs/access.log main;
    rewrite_log     on;

```

```
        #sendfile 指令指定 nginx 是否调用 sendfile 函数（zero copy 方式）来输出文件，对于普通应用，
        #必须设为 on,如果用来进行下载等应用磁盘IO重负载应用，可设置为 off，以平衡磁盘与网络I/O处理速度，降低系统的uptime.
        sendfile        on;
        #tcp_nopush     on;

        #连接超时时间
        keepalive_timeout  120;
        tcp_nodelay        on;

        #gzip压缩开关
        #gzip  on;

        #设定实际的服务器列表
        upstream zp_server1{
            server 127.0.0.1:8089;
        }

        #HTTP服务器
        server {
            #监听80端口，80端口是知名端口号，用于HTTP协议
            listen       80;

            #定义使用www.xx.com访问
            server_name  www.helloworld.com;

            #首页
            index index.html

            #指向webapp的目录
            root D:\01_Workspace\Project\github\zp\SpringNotes\spring-security\spring-shiro\src\main\webapp;

            #编码格式
            charset utf-8;

            #代理配置参数
            proxy_connect_timeout 180;
            proxy_send_timeout 180;
            proxy_read_timeout 180;
            proxy_set_header Host $host;
            proxy_set_header X-Forwarder-For $remote_addr;
```

```
74.
75.         #反向代理的路径（和upstream绑定），location 后面设置映射的路径
76.         location / {
77.             proxy_pass http://zp_server1;
78.         }
79.
80.         #静态文件，nginx自己处理
81.         location ~ ^/(images|javascript|js|css|flash|media|static)/ {
                root D:\01_Workspace\Project\github\zp\SpringNotes\spring-security\sp
82. shiro\src\main\webapp\views;
                #过期30天，静态文件不怎么更新，过期可以设大一点，如果频繁更新，则可以设置得小一
83. 点。
84.             expires 30d;
85.         }
86.
87.         #设定查看Nginx状态的地址
88.         location /NginxStatus {
89.             stub_status           on;
90.             access_log            on;
91.             auth_basic            "NginxStatus";
92.             auth_basic_user_file  conf/htpasswd;
93.         }
94.
95.         #禁止访问 .htxxx 文件
96.         location ~ /\.ht {
97.             deny all;
98.         }
99.
100.        #错误处理页面（可选择性配置）
101.        #error_page   404              /404.html;
102.        #error_page   500 502 503 504  /50x.html;
103.        #location = /50x.html {
104.        #    root   html;
105.        #}
106.    }
107. }
```

好了，让我们来试试吧：

1. 启动 webapp，注意启动绑定的端口要和 nginx 中的 `upstream` 设置的端口保持一致。
2. 更改 host：在 C:\Windows\System32\drivers\etc 目录下的 host 文件中添加一条 DNS 记录

```
127.0.0.1 www.helloworld.com
```

1. 启动前文中 startup.bat 的命令
2. 在浏览器中访问 www.helloworld.com，不出意外，已经可以访问了。

Https 反向代理

一些对安全性要求比较高的站点，可能会使用 HTTPS（一种使用 ssl 通信标准的安全 HTTP 协议）。

这里不科普 HTTP 协议和 SSL 标准。但是，使用 nginx 配置 https 需要知道几点：

- HTTPS 的固定端口号是 443，不同于 HTTP 的 80 端口
- SSL 标准需要引入安全证书，所以在 nginx.conf 中你需要指定证书和它对应的 key

其他和 http 反向代理基本一样，只是在 `Server` 部分配置有些不同。

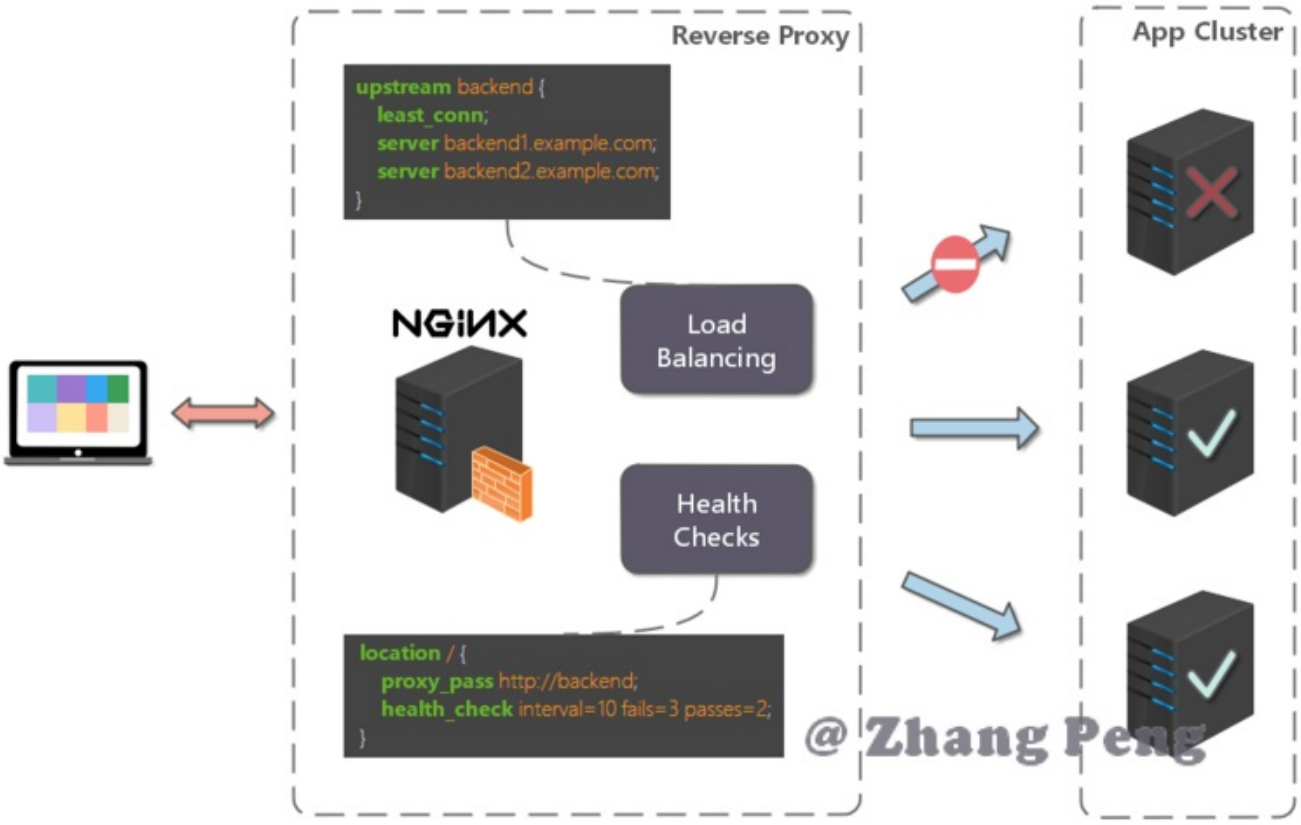
```
1.    #HTTP服务器
2.    server {
3.        #监听443端口。443为知名端口号，主要用于HTTPS协议
4.        listen       443 ssl;
5.
6.        #定义使用www.xx.com访问
7.        server_name  www.helloworld.com;
8.
9.        #ssl证书文件位置(常见证书文件格式为：crt/pem)
10.       ssl_certificate      cert.pem;
11.       #ssl证书key位置
12.       ssl_certificate_key  cert.key;
13.
14.       #ssl配置参数（选择性配置）
15.       ssl_session_cache    shared:SSL:1m;
16.       ssl_session_timeout  5m;
17.       #数字签名，此处使用MD5
18.       ssl_ciphers  HIGH:!aNULL:!MD5;
19.       ssl_prefer_server_ciphers  on;
20.
21.       location / {
22.           root   /root;
23.           index  index.html index.htm;
24.       }
25.   }
```

负载均衡

前面的例子中，代理仅仅指向一个服务器。

但是，网站在实际运营过程中，大部分都是以集群的方式运行，这时需要使用负载均衡来分流。

nginx 也可以实现简单的负载均衡功能。



假设这样一个应用场景：将应用部署在 192.168.1.11:80、192.168.1.12:80、192.168.1.13:80 三台 linux 环境的服务器上。网站域名叫 www.helloworld.com，公网 IP 为 192.168.1.11。在公网 IP 所在的服务器上部署 nginx，对所有请求做负载均衡处理（下面例子中使用的是加权轮询策略）。

nginx.conf 配置如下：

```
1.  http {
2.       #设定mime类型,类型由mime.type文件定义
3.      include       /etc/nginx/mime.types;
4.      default_type  application/octet-stream;
5.      #设定日志格式
6.      access_log    /var/log/nginx/access.log;
7.
```

```
    #设定负载均衡的服务器列表
    upstream load_balance_server {
        #weigth参数表示权值，权值越高被分配到的几率越大
        server 192.168.1.11:80   weight=5;
        server 192.168.1.12:80   weight=1;
        server 192.168.1.13:80   weight=6;
    }

   #HTTP服务器
    server {
        #侦听80端口
        listen       80;

        #定义使用www.xx.com访问
        server_name  www.helloworld.com;

        #对所有请求进行负载均衡请求
        location / {
            root        /root;                #定义服务器的默认网站根目录位置
            index       index.html index.htm;  #定义首页索引文件的名称
            proxy_pass  http://load_balance_server ;#请求转向load_balance_server 定义的服务器列表

            #以下是一些反向代理的配置(可选择性配置)
            #proxy_redirect off;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            #后端的Web服务器可以通过X-Forwarded-For获取用户真实IP
            proxy_set_header X-Forwarded-For $remote_addr;
            proxy_connect_timeout 90;          #nginx跟后端服务器连接超时时间(代理连接超时)
            proxy_send_timeout 90;             #后端服务器数据回传时间(代理发送超时)
            proxy_read_timeout 90;             #连接成功后，后端服务器响应时间(代理接收超时)
            proxy_buffer_size 4k;              #设置代理服务器（nginx）保存用户头信息的缓冲区大小
            proxy_buffers 4 32k;               #proxy_buffers缓冲区，网页平均在32k以下的话，这样设置
            proxy_busy_buffers_size 64k;       #高负荷下缓冲大小 （proxy_buffers*2）
            proxy_temp_file_write_size 64k;    #设定缓存文件夹大小，大于这个值，将从upstream服务器传

```

```
            client_max_body_size 10m;           #允许客户端请求的最大单文件字节数
            client_body_buffer_size 128k;       #缓冲区代理缓冲用户端请求的最大字节数
        }
    }
}
```

负载均衡策略

Nginx 提供了多种负载均衡策略，让我们来一一了解一下：

负载均衡策略在各种分布式系统中基本上原理一致，对于原理有兴趣，不妨参考 负载均衡

轮询

```
upstream bck_testing_01 {
  # 默认所有服务器权重为 1
  server 192.168.250.220:8080
  server 192.168.250.221:8080
  server 192.168.250.222:8080
}
```

加权轮询

```
upstream bck_testing_01 {
  server 192.168.250.220:8080   weight=3
  server 192.168.250.221:8080             # default weight=1
  server 192.168.250.222:8080             # default weight=1
}
```

最少连接

```
upstream bck_testing_01 {
  least_conn;

  # with default weight for all (weight=1)
  server 192.168.250.220:8080
  server 192.168.250.221:8080
  server 192.168.250.222:8080
}
```

加权最少连接

```
upstream bck_testing_01 {
```

```
    least_conn;

    server 192.168.250.220:8080   weight=3
    server 192.168.250.221:8080              # default weight=1
    server 192.168.250.222:8080              # default weight=1
}
```

IP Hash

```
upstream bck_testing_01 {

    ip_hash;

    # with default weight for all (weight=1)
    server 192.168.250.220:8080
    server 192.168.250.221:8080
    server 192.168.250.222:8080

}
```

普通 Hash

```
upstream bck_testing_01 {

    hash $request_uri;

    # with default weight for all (weight=1)
    server 192.168.250.220:8080
    server 192.168.250.221:8080
    server 192.168.250.222:8080

}
```

网站有多个 webapp 的配置

当一个网站功能越来越丰富时，往往需要将一些功能相对独立的模块剥离出来，独立维护。这样的话，通常，会有多个 webapp。

举个例子：假如 www.helloworld.com 站点有好几个 webapp，finance（金融）、product（产品）、admin（用户中心）。访问这些应用的方式通过上下文(context)来进行区分：

www.helloworld.com/finance/

www.helloworld.com/product/

www.helloworld.com/admin/

我们知道，http 的默认端口号是 80，如果在一台服务器上同时启动这 3 个 webapp 应用，都用 80 端口，肯定是不成的。所以，这三个应用需要分别绑定不同的端口号。

那么，问题来了，用户在实际访问 www.helloworld.com 站点时，访问不同 webapp，总不会还带着对应的端口号去访问吧。所以，你再次需要用到反向代理来做处理。

配置也不难，来看看怎么做吧：

```
http {
    #此处省略一些基本配置

    upstream product_server{
        server www.helloworld.com:8081;
    }

    upstream admin_server{
        server www.helloworld.com:8082;
    }

    upstream finance_server{
        server www.helloworld.com:8083;
    }

    server {
        #此处省略一些基本配置
        #默认指向product的server
        location / {
            proxy_pass http://product_server;
        }
```

```
22.
23.         location /product/{
24.             proxy_pass http://product_server;
25.         }
26.
27.         location /admin/ {
28.             proxy_pass http://admin_server;
29.         }
30.
31.         location /finance/ {
32.             proxy_pass http://finance_server;
33.         }
34.     }
35. }
```

静态站点

有时候，我们需要配置静态站点(即 html 文件和一堆静态资源)。

举例来说：如果所有的静态资源都放在了 `/app/dist` 目录下，我们只需要在 `nginx.conf` 中指定首页以及这个站点的 host 即可。

配置如下：

```
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    gzip on;
    gzip_types text/plain application/x-javascript text/css application/xml
text/javascript application/javascript image/jpeg image/gif image/png;
    gzip_vary on;

    server {
        listen       80;
        server_name  static.zp.cn;

        location / {
            root /app/dist;
            index index.html;
            #转发任何请求到 index.html
        }
    }
}
```

然后，添加 HOST：

```
127.0.0.1 static.zp.cn
```

此时，在本地浏览器访问 `static.zp.cn` ，就可以访问静态站点了。

搭建文件服务器

有时候，团队需要归档一些数据或资料，那么文件服务器必不可少。使用 Nginx 可以非常快速便捷的搭建一个简易的文件服务。

Nginx 中的配置要点：

- 将 autoindex 开启可以显示目录，默认不开启。
- 将 autoindex_exact_size 开启可以显示文件的大小。
- 将 autoindex_localtime 开启可以显示文件的修改时间。
- root 用来设置开放为文件服务的根路径。
- charset 设置为 `charset utf-8,gbk;`，可以避免中文乱码问题（windows 服务器下设置后，依然乱码，本人暂时没有找到解决方法）。

一个最简化的配置如下：

```
autoindex on;# 显示目录
autoindex_exact_size on;# 显示文件大小
autoindex_localtime on;# 显示文件时间

server {
    charset      utf-8,gbk; # windows 服务器下设置后，依然乱码，暂时无解
    listen       9050 default_server;
    listen       [::]:9050 default_server;
    server_name  _;
    root         /share/fs;
}
```

解决跨域

web 领域开发中，经常采用前后端分离模式。这种模式下，前端和后端分别是独立的 web 应用程序，例如：后端是 Java 程序，前端是 React 或 Vue 应用。

各自独立的 web app 在互相访问时，势必存在跨域问题。解决跨域问题一般有两种思路：

1. **CORS**

在后端服务器设置 HTTP 响应头，把你需要允许访问的域名加入 `Access-Control-Allow-Origin` 中。

1. **jsonp**

把后端根据请求，构造 json 数据，并返回，前端用 jsonp 跨域。

这两种思路，本文不展开讨论。

需要说明的是，nginx 根据第一种思路，也提供了一种解决跨域的解决方案。

举例：www.helloworld.com 网站是由一个前端 app ，一个后端 app 组成的。前端端口号为 9000， 后端端口号为 8080。

前端和后端如果使用 http 进行交互时，请求会被拒绝，因为存在跨域问题。来看看，nginx 是怎么解决的吧：

首先，在 enable-cors.conf 文件中设置 cors ：

```
# allow origin list
set $ACAO '*';

# set single origin
if ($http_origin ~* (www.helloworld.com)$) {
  set $ACAO $http_origin;
}

if ($cors = "trueget") {
    add_header 'Access-Control-Allow-Origin' "$http_origin";
    add_header 'Access-Control-Allow-Credentials' 'true';
    add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
    add_header 'Access-Control-Allow-Headers' 'DNT,X-Mx-ReqToken,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type';
}
```

```
15.
16.  if ($request_method = 'OPTIONS') {
17.    set $cors "${cors}options";
18.  }
19.
20.  if ($request_method = 'GET') {
21.    set $cors "${cors}get";
22.  }
23.
24.  if ($request_method = 'POST') {
25.    set $cors "${cors}post";
26.  }
```

接下来，在你的服务器中 `include enable-cors.conf` 来引入跨域配置：

```
1.  # ----------------------------------------------------
2.  # 此文件为项目 nginx 配置片段
3.  # 可以直接在 nginx config 中 include（推荐）
4.  # 或者 copy 到现有 nginx 中，自行配置
5.  # www.helloworld.com 域名需配合 dns hosts 进行配置
6.  # 其中，api 开启了 cors，需配合本目录下另一份配置文件
7.  # ----------------------------------------------------
8.  upstream front_server{
9.    server www.helloworld.com:9000;
10. }
11. upstream api_server{
12.   server www.helloworld.com:8080;
13. }
14.
15. server {
16.   listen       80;
17.   server_name  www.helloworld.com;
18.
19.   location ~ ^/api/ {
20.     include enable-cors.conf;
21.     proxy_pass http://api_server;
22.     rewrite "^/api/(.*)$" /$1 break;
23.   }
24.
25.   location ~ ^/ {
26.     proxy_pass http://front_server;
27.   }
```

```
28. }
```

到此，就完成了。

资源

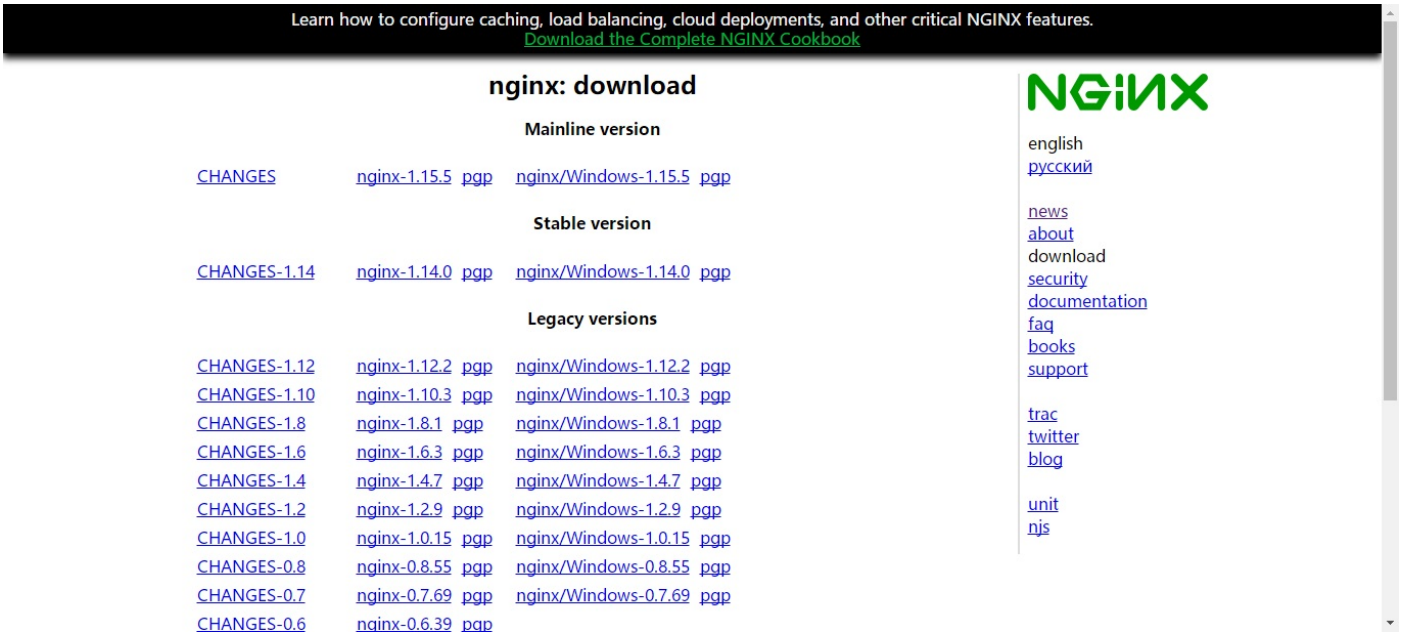
- Nginx 的中文维基
- Nginx 开发从入门到精通
- nginx-admins-handbook
- nginxconfig.io - 一款 Nginx 配置生成器

Nginx 运维

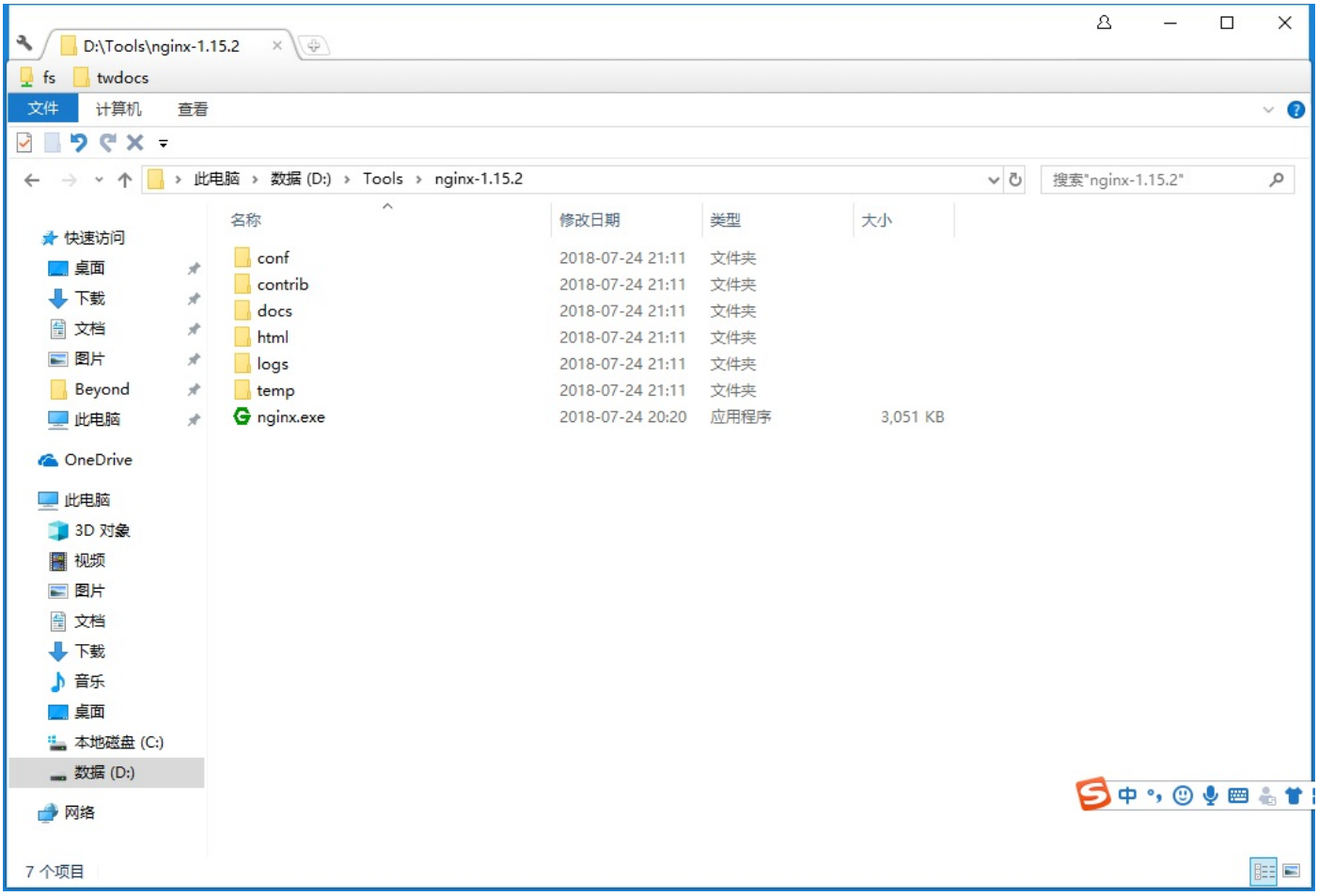
一、普通安装

Windows 安装

（1）进入官方下载地址，选择合适版本（nginx/Windows-xxx）。



（2）解压到本地



（3）启动

下面以 C 盘根目录为例说明下：

```
1.  cd C:
2.  cd C:\nginx-0.8.54 start nginx
```

> 注：Nginx / Win32 是运行在一个控制台程序，而非 windows 服务方式的。服务器方式目前还是开发尝试中。

Linux 安装

rpm 包方式（推荐）

（1）进入下载页面，选择合适版本下载。

```
    $ wget http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-
1.  0.el7.ngx.noarch.rpm
```

（2）安装 nginx rpm 包

nginx rpm 包实际上安装的是 nginx 的 yum 源。

```
$ rpm -ivh nginx-*.rpm
```

（3）正式安装 rpm 包

```
$ yum install nginx
```

（4）关闭防火墙

```
$ firewall-cmd --zone=public --add-port=80/tcp --permanent
$ firewall-cmd --reload
```

源码编译方式

安装编译工具及库

Nginx 源码的编译依赖于 gcc 以及一些库文件，所以必须提前安装。

```
$ yum -y install make zlib zlib-devel gcc-c++ libtool  openssl openssl-devel
```

Nginx 依赖 pcre 库，安装步骤如下：

（1）下载解压到本地

进入pcre 官网下载页面，选择合适的版本下载。

我选择的是 8.35 版本：

```
    wget -O /opt/pcre/pcre-8.35.tar.gz
1.  http://downloads.sourceforge.net/project/pcre/pcre/8.35/pcre-8.35.tar.gz
2.  cd /opt/pcre
3.  tar zxvf pcre-8.35.tar.gz
```

（2）编译安装

执行以下命令：

```
cd /opt/pcre/pcre-8.35
./configure
make && make install
```

（3）检验是否安装成功

执行 `pcre-config --version` 命令。

编译安装 Nginx

安装步骤如下：

（1）下载解压到本地

进入官网下载地址：http://nginx.org/en/download.html ，选择合适的版本下载。

我选择的是 1.12.2 版本：http://downloads.sourceforge.net/project/pcre/pcre/8.35/pcre-8.35.tar.gz

```
    wget -O /opt/nginx/nginx-1.12.2.tar.gz http://nginx.org/download/nginx-
1.  1.12.2.tar.gz
2.  cd /opt/nginx
3.  tar zxvf nginx-1.12.2.tar.gz
```

（2）编译安装

执行以下命令：

```
1.  cd /opt/nginx/nginx-1.12.2
    ./configure --with-http_stub_status_module --with-http_ssl_module --with-
2.  pcre=/opt/pcre/pcre-8.35
3.  make && make install
```

（3）关闭防火墙

```
1.  $ firewall-cmd --zone=public --add-port=80/tcp --permanent
2.  $ firewall-cmd --reload
```

（4） 启动 Nginx

安装成功后，直接执行 `nginx` 命令即可启动 nginx。

启动后，访问站点：

Linux 开机自启动

Centos7 以上是用 Systemd 进行系统初始化的，Systemd 是 Linux 系统中最新的初始化系统（init），它主要的设计目标是克服 sysvinit 固有的缺点，提高系统的启动速度。Systemd 服务文件以 .service 结尾。

rpm 包方式

如果是通过 rpm 包安装的，会自动创建 nginx.service 文件。

直接用命令：

```
1.  $ systemctl enable nginx.service
```

设置开机启动即可。

源码编译方式

如果采用源码编译方式，需要手动创建 nginx.service 文件。

二、Docker 安装

- 官网镜像：https://hub.docker.com/_/nginx/
- 下载镜像：`docker pull nginx`
- 启动容器：`docker run --name my-nginx -p 80:80 -v /data/docker/nginx/logs:/var/log/nginx -v /data/docker/nginx/conf/nginx.conf:/etc/nginx/nginx.conf:ro -d nginx`
- 重新加载配置（目前测试无效，只能重启服务）：`docker exec -it my-nginx nginx -s reload`
- 停止服务：`docker exec -it my-nginx nginx -s stop` 或者：`docker stop my-nginx`
- 重新启动服务：`docker restart my-nginx`

三、脚本

> CentOS7 环境安装脚本：软件运维配置脚本集合

安装说明

- 采用编译方式安装 Nginx，并将其注册为 systemd 服务
- 安装路径为：`/usr/local/nginx`
- 默认下载安装 `1.16.0` 版本

使用方法

- 默认安装 - 执行以下任意命令即可：

```
curl -o- https://gitee.com/turnon/linux-tutorial/raw/master/codes/linux/soft/nginx-install.sh | bash
wget -qO- https://gitee.com/turnon/linux-tutorial/raw/master/codes/linux/soft/nginx-install.sh | bash
```

- 自定义安装 - 下载脚本到本地，并按照以下格式执行：

```
sh nginx-install.sh [version]
```

参考资料

- http://www.dohooe.com/2016/03/03/352.html?utm_source=tuicool&utm_medium=referral
- nginx+keepalived实现nginx双主高可用的负载均衡

Nginx 配置

> Nginx 的默认配置文件为 `nginx.conf` 。
>
> - `nginx -c xxx.conf` - 以指定的文件作为配置文件，启动 Nginx。

配置文件实例

以下为一个 `nginx.conf` 配置文件实例：

```
#定义 nginx 运行的用户和用户组
user www www;

#nginx 进程数，建议设置为等于 CPU 总核心数。
worker_processes 8;

#nginx 默认没有开启利用多核 CPU，通过增加 worker_cpu_affinity 配置参数来充分利用多核 CPU 以下是 8 核的配置参数
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]
error_log /var/log/nginx/error.log info;

#进程文件
pid /var/run/nginx.pid;

#一个 nginx 进程打开的最多文件描述符数目，理论值应该是最多打开文件数（系统的值 ulimit -n）与 nginx 进程数相除，但是 nginx 分配请求并不均匀，所以建议与 ulimit -n 的值保持一致。
worker_rlimit_nofile 65535;

#工作模式与连接数上限
events
{
    #参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll 模型是 Linux 2.6 以上版本内核中的高性能网络 I/O 模型，如果跑在 FreeBSD 上面，就用 kqueue 模型。
    #epoll 是多路复用 IO(I/O Multiplexing) 中的一种方式，但是仅用于 linux2.6 以上内核，可以大大提高 nginx 的性能
    use epoll;


##################################################################################
    #单个后台 worker process 进程的最大并发链接数
    #事件模块指令，定义 nginx 每个进程最大连接数，默认 1024。最大客户连接数由 worker_processes 和 worker_connections 决定
    #即 max_client=worker_processes*worker_connections，在作为反向代理时：max_client=worker_processes*worker_connections / 4
```

```
    worker_connections 65535;

################################################################################
}

#设定 http 服务器
http {
    include mime.types; #文件扩展名与文件类型映射表
    default_type application/octet-stream; #默认文件类型
    #charset utf-8; #默认编码

    server_names_hash_bucket_size 128; #服务器名字的 hash 表大小
    client_header_buffer_size 32k; #上传文件大小限制
    large_client_header_buffers 4 64k; #设定请求缓
    client_max_body_size 8m; #设定请求缓
    sendfile on; #开启高效文件传输模式，sendfile 指令指定 nginx 是否调用 sendfile 函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘 IO 重负载应用，可设置为 off，以平衡磁盘与网络 I/O 处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成 off。
    autoindex on; #开启目录列表访问，合适下载服务器，默认关闭。
    tcp_nopush on; #防止网络阻塞
    tcp_nodelay on; #防止网络阻塞

    ##连接客户端超时时间各种参数设置##
    keepalive_timeout  120;          #单位是秒，客户端连接时时间，超时之后服务器端自动关闭该连接 如果 nginx 守护进程在这个等待的时间里，一直没有收到浏览发过来 http 请求，则关闭这个 http 连接
    client_header_timeout 10;        #客户端请求头的超时时间
    client_body_timeout 10;          #客户端请求主体超时时间
    reset_timedout_connection on;    #告诉 nginx 关闭不响应的客户端连接。这将会释放那个客户端所占有的内存空间
    send_timeout 10;                 #客户端响应超时时间，在两次客户端读取操作之间。如果在这段时间内，客户端没有读取任何数据，nginx 就会关闭连接
    ##################################

    #FastCGI 相关参数是为了改善网站的性能：减少资源占用，提高访问速度。下面参数看字面意思都能理解。
    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
    fastcgi_buffer_size 64k;
    fastcgi_buffers 4 64k;
    fastcgi_busy_buffers_size 128k;
    fastcgi_temp_file_write_size 128k;
```

```
65.
66.     ###作为代理缓存服务器设置#######
67.     ###先写到 temp 再移动到 cache
        #proxy_cache_path /var/tmp/nginx/proxy_cache levels=1:2
68. keys_zone=cache_one:512m inactive=10m max_size=64m;
69.     ###以上 proxy_temp 和 proxy_cache 需要在同一个分区中
        ###levels=1:2 表示缓存级别，表示缓存目录的第一级目录是 1 个字符，第二级目录是 2 个字符
70. keys_zone=cache_one:128m 缓存空间起名为 cache_one 大小为 512m
        ###max_size=64m 表示单个文件超过 128m 就不缓存了  inactive=10m 表示缓存的数据，10
71. 分钟内没有被访问过就删除
72.     #########end####################
73.
74.     #####对传输文件压缩###########
75.     #gzip 模块设置
76.     gzip on; #开启 gzip 压缩输出
77.     gzip_min_length 1k; #最小压缩文件大小
78.     gzip_buffers 4 16k; #压缩缓冲区
79.     gzip_http_version 1.0; #压缩版本（默认 1.1，前端如果是 squid2.5 请使用 1.0）
        gzip_comp_level 2; #压缩等级，gzip 压缩比，1 为最小，处理最快；9 为压缩比最大，处理
80. 最慢，传输速度最快，也最消耗 CPU；
81.     gzip_types text/plain application/x-javascript text/css application/xml;
        #压缩类型，默认就已经包含 text/html，所以下面就不用再写了，写上去也不会有问题，但是会有
82. 一个 warn。
83.     gzip_vary on;
84.     ##############################
85.
86.     #limit_zone crawler $binary_remote_addr 10m; #开启限制 IP 连接数的时候需要使用
87.
88.     upstream blog.ha97.com {
            #upstream 的负载均衡，weight 是权重，可以根据机器配置定义权重。weigth 参数表示权
89. 值，权值越高被分配到的几率越大。
90.         server 192.168.80.121:80 weight=3;
91.         server 192.168.80.122:80 weight=2;
92.         server 192.168.80.123:80 weight=3;
93.     }
94.
95.     #虚拟主机的配置
96.     server {
97.         #监听端口
98.         listen 80;
99.
100.        ############https##################
101.        #listen 443 ssl;
```

```
102.          #ssl_certificate /opt/https/xxxxxx.crt;
103.          #ssl_certificate_key /opt/https/xxxxxx.key;
104.          #ssl_protocols SSLv3 TLSv1;
105.          #ssl_ciphers HIGH:!ADH:!EXPORT57:RC4+RSA:+MEDIUM;
106.          #ssl_prefer_server_ciphers on;
107.          #ssl_session_cache shared:SSL:2m;
108.          #ssl_session_timeout 5m;
109.          ##################################end
110.
111.          #域名可以有多个，用空格隔开
112.          server_name www.ha97.com ha97.com;
113.          index index.html index.htm index.php;
114.          root /data/www/ha97;
115.          location ~ .*.(php|php5)?$ {
116.              fastcgi_pass 127.0.0.1:9000;
117.              fastcgi_index index.php;
118.              include fastcgi.conf;
119.          }
120.
121.          #图片缓存时间设置
122.          location ~ .*.(gif|jpg|jpeg|png|bmp|swf)$ {
123.              expires 10d;
124.          }
125.
126.          #JS 和 CSS 缓存时间设置
127.          location ~ .*.(js|css)?$ {
128.              expires 1h;
129.          }
130.
131.          #日志格式设定
              log_format access '$remote_addr - $remote_user [$time_local] "$request"
    ' '$status $body_bytes_sent "$http_referer" ' '"$http_user_agent"
132.  $http_x_forwarded_for';
133.
134.          #定义本虚拟主机的访问日志
135.          access_log /var/log/nginx/ha97access.log access;
136.
137.          #对 "/" 启用反向代理
138.          location / {
139.              proxy_pass http://127.0.0.1:88;
140.              proxy_redirect off;
141.              proxy_set_header X-Real-IP $remote_addr;
142.              #后端的 Web 服务器可以通过 X-Forwarded-For 获取用户真实 IP
```

```
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            #以下是一些反向代理的配置，可选。
            proxy_set_header Host $host;
            client_max_body_size 10m; #允许客户端请求的最大单文件字节数
            client_body_buffer_size 128k; #缓冲区代理缓冲用户端请求的最大字节数，

            ##代理设置 以下设置是 nginx 和后端服务器之间通讯的设置##
            proxy_connect_timeout 90; #nginx 跟后端服务器连接超时时间（代理连接超时）
            proxy_send_timeout 90; #后端服务器数据回传时间（代理发送超时）
            proxy_read_timeout 90; #连接成功后，后端服务器响应时间（代理接收超时）
            proxy_buffering on;    #该指令开启从后端被代理服务器的响应内容缓冲 此参数开启后 proxy_buffers 和 proxy_busy_buffers_size 参数才会起作用
            proxy_buffer_size 4k;  #设置代理服务器（nginx）保存用户头信息的缓冲区大小
            proxy_buffers 4 32k;   #proxy_buffers 缓冲区，网页平均在 32k 以下的设置
            proxy_busy_buffers_size 64k; #高负荷下缓冲大小（proxy_buffers*2）
            proxy_max_temp_file_size 2048m; #默认 1024m，该指令用于设置当网页内容大于 proxy_buffers 时，临时文件大小的最大值。如果文件大于这个值，它将从 upstream 服务器同步地传递请求，而不是缓冲到磁盘
            proxy_temp_file_write_size 512k; 这是当被代理服务器的响应过大时 nginx 一次性写入临时文件的数据量。
            proxy_temp_path  /var/tmp/nginx/proxy_temp;    ##定义缓冲存储目录，之前必须要先手动创建此目录
            proxy_headers_hash_max_size 51200;
            proxy_headers_hash_bucket_size 6400;
            ##########################################################
        }

        #设定查看 nginx 状态的地址
        location /nginxStatus {
            stub_status on;
            access_log on;
            auth_basic "nginxStatus";
            auth_basic_user_file conf/htpasswd;
            #htpasswd 文件的内容可以用 apache 提供的 htpasswd 工具来产生。
        }

        #本地动静分离反向代理配置
        #所有 jsp 的页面均交由 tomcat 或 resin 处理
        location ~ .(jsp|jspx|do)?$ {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass http://127.0.0.1:8080;
```

```
        }

        #所有静态文件由 nginx 直接读取不经过 tomcat 或 resin
        location ~ .*.
(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|pdf|xls|mp3|wm
        { expires 15d; }

        location ~ .*.(js|css)?$
        { expires 1h; }
    }
}
```

基本规则

管理 Nginx 配置

> 随着 Nginx 配置的增长，您有必要组织、管理配置内容。
>
> 当您的 Nginx 配置增加时，组织配置的需求也会增加。 井井有条的代码是：
>
> - 易于理解
> - 易于维护
> - 易于使用
>
> 使用 `include` 指令可将常用服务器配置移动到单独的文件中，并将特定代码附加到全局配置，上下文等中。
>
> 我总是尝试在配置树的根目录中保留多个目录。 这些目录存储所有附加到主文件的配置文件。 我更喜欢以下结构：
>
> - `html` - 用于默认静态文件，例如 全局 5xx 错误页面
> - `master` - 用于主要配置，例如 ACL，侦听指令和域
>   - `_acls` - 用于访问控制列表，例如 地理或地图模块
>   - `_basic` - 用于速率限制规则，重定向映射或代理参数
>   - `_listen` - 用于所有侦听指令； 还存储 SSL 配置
>   - `_server` - 用于域（localhost）配置； 还存储所有后端定义
> - `modules` - 用于动态加载到 Nginx 中的模块
> - `snippets` - 用于 Nginx 别名，配置模板
>
> 如果有必要，我会将其中一些附加到具有 `server` 指令的文件中。

示例：

```
## Store this configuration in https.conf for example:
listen 10.240.20.2:443 ssl;

ssl_certificate
/etc/nginx/master/_server/example.com/certs/nginx_example.com_bundle.crt;
ssl_certificate_key
/etc/nginx/master/_server/example.com/certs/example.com.key;

## Include this file to the server section:
server {

  include /etc/nginx/master/_listen/10.240.20.2/https.conf;

```

```
    ## And other:
    include /etc/nginx/master/_static/errors.conf;
    include /etc/nginx/master/_server/_helpers/global.conf;

    ...

    server_name domain.com www.domain.com;

    ...
```

重加载 Nginx 配置

示例：

```
## 1)
systemctl reload nginx

## 2)
service nginx reload

## 3)
/etc/init.d/nginx reload

## 4)
/usr/sbin/nginx -s reload

## 5)
kill -HUP $(cat /var/run/nginx.pid)
## or
kill -HUP $(pgrep -f "nginx: master")

## 6)
/usr/sbin/nginx -g 'daemon on; master_process on;' -s reload
```

监听 80 和 443 端口

> 如果您使用完全相同的配置为 HTTP 和 HTTPS 提供服务（单个服务器同时处理 HTTP 和 HTTPS 请求），Nginx 足够智能，可以忽略通过端口 80 加载的 SSL 指令。
>
> Nginx 的最佳实践是使用单独的服务器进行这样的重定向（不与您的主要配置的服务器共享），对所有内容进行硬编码，并且完全不使用正则表达式。

> 我不喜欢复制规则，但是单独的监听指令无疑可以帮助您维护和修改配置。
>
> 如果将多个域固定到一个 IP 地址，则很有用。 这使您可以将一个侦听指令（例如，如果将其保留在配置文件中）附加到多个域配置。
>
> 如果您使用的是 HTTPS，则可能还需要对域进行硬编码，因为您必须预先知道要提供的证书。

示例：

```
## For HTTP:
server {

  listen 10.240.20.2:80;

  ...

}

## For HTTPS:
server {

  listen 10.240.20.2:443 ssl;

  ...

}
```

显示指定监听的地址和端口

Nginx 的 listen 指令用于监听指定的 IP 地址和端口号，配置形式为： `listen <address>:<port>` 。若 IP 地址或端口缺失，Nginx 会以默认值来替换。

而且，仅当需要区分与 listen 指令中的同一级别匹配的服务器块时，才会评估 server_name 指令。

示例：

```
server {

  ## This block will be processed:
  listen 192.168.252.10;  ## --> 192.168.252.10:80

  ...

```

```
}

server {

  listen 80;  ## --> *:80 --> 0.0.0.0:80
  server_name api.random.com;

  ...

}
```

防止使用未定义的服务器名称处理请求

> Nginx 应该阻止使用未定义的服务器名称（也使用 IP 地址）处理请求。它可以防止配置错误，例如流量转发到不正确的后端。通过创建默认虚拟虚拟主机可以轻松解决该问题，该虚拟虚拟主机可以捕获带有无法识别的主机标头的所有请求。
>
> 如果没有一个 listen 指令具有 default_server 参数，则具有 address:port 对的第一台服务器将是该对的默认服务器（这意味着 Nginx 始终具有默认服务器）。
>
> 如果有人使用 IP 地址而不是服务器名称发出请求，则主机请求标头字段将包含 IP 地址，并且可以使用 IP 地址作为服务器名称来处理请求。
>
> 在现代版本的 Nginx 中，不需要服务器名称_。如果找不到具有匹配的 listen 和 server_name 的服务器，Nginx 将使用默认服务器。如果您的配置分散在多个文件中，则评估顺序将不明确，因此您需要显式标记默认服务器。
>
> Nginx 使用 Host 标头进行 server_name 匹配。它不使用 TLS SNI。这意味着对于 SSL 服务器，Nginx 必须能够接受 SSL 连接，这归结为具有证书/密钥。证书/密钥可以是任意值，例如自签名。

示例：

```
## Place it at the beginning of the configuration file to prevent mistakes:
server {

  ## For ssl option remember about SSL parameters (private key, certs, cipher suites, etc.);
  ## add default_server to your listen directive in the server that you want to act as the default:
  listen 10.240.20.2:443 default_server ssl;

  ## We catch:
  ##   - invalid domain names
  ##   - requests without the "Host" header
  ##   - and all others (also due to the above setting)
```

```
    ##   - default_server in server_name directive is not required - I add this
12. for a better understanding and I think it's an unwritten standard
    ## ...but you should know that it's irrelevant, really, you can put in
13. everything there.
14.   server_name _ "" default_server;
15.
16.   ...
17.
18.   return 444;
19.
20.   ## We can also serve:
21.   ## location / {
22.
23.     ## static file (error page):
24.     ##   root /etc/nginx/error-pages/404;
25.     ## or redirect:
26.     ##   return 301 https://badssl.com;
27.
28.     ## return 444;
29.
30.   ## }
31.
32. }
33.
34. server {
35.
36.   listen 10.240.20.2:443 ssl;
37.
38.   server_name domain.com;
39.
40.   ...
41.
42. }
43.
44. server {
45.
46.   listen 10.240.20.2:443 ssl;
47.
48.   server_name domain.org;
49.
50.   ...
51.
52. }
```

不要在 listen 或 upstream 中使用 hostname

> 通常，在 listen 或上游指令中使用主机名是一种不好的做法。
>
> 在最坏的情况下，Nginx 将无法绑定到所需的 TCP 套接字，这将完全阻止 Nginx 启动。
>
> 最好和更安全的方法是知道需要绑定的 IP 地址，并使用该地址代替主机名。 这也可以防止 Nginx 查找地址并消除对外部和内部解析器的依赖。
>
> 在 server_name 指令中使用\$ hostname（计算机的主机名）变量也是不当行为的示例（类似于使用主机名标签）。
>
> 我认为也有必要设置 IP 地址和端口号对，以防止可能难以调试的软错误。

示例：

错误配置

```
upstream {

  server http://x-9s-web01-prod:8080;

}

server {

  listen rev-proxy-prod:80;

  ...

}
```

正确配置

```
upstream {

  server http://192.168.252.200:8080;

}

server {

  listen 10.10.100.20:80;
```

```
10.
11.    ...
12.
13. }
```

指令中只配置一个 SSL

> 此规则使调试和维护更加容易。
>
> 请记住，无论 SSL 参数如何，您都可以在同一监听指令（IP 地址）上使用多个 SSL 证书。
>
> 我认为要在多个 HTTPS 服务器之间共享一个 IP 地址，您应该使用一个 SSL 配置（例如协议，密码，曲线）。这是为了防止错误和配置不匹配。
>
> 还请记住有关默认服务器的配置。这很重要，因为如果所有 listen 指令都没有 default_server 参数，则配置中的第一台服务器将是默认服务器。因此，您应该只使用一个 SSL 设置，并且在同一 IP 地址上使用多个名称。
>
> 从 Nginx 文档中：
>
> 这是由 SSL 协议行为引起的。在浏览器发送 HTTP 请求之前，已建立 SSL 连接，nginx 不知道所请求服务器的名称。因此，它可能仅提供默认服务器的证书。
>
> 还要看看这个：
>
> TLS 服务器名称指示扩展名（SNI，RFC 6066）是在单个 IP 地址上运行多个 HTTPS 服务器的更通用的解决方案，它允许浏览器在 SSL 握手期间传递请求的服务器名称，因此，服务器将知道哪个用于连接的证书。
>
> 另一个好主意是将常用服务器设置移到单独的文件（即 common / example.com.conf）中，然后将其包含在单独的服务器块中。

示例：

```
1.  ## Store this configuration in e.g. https.conf:
2.  listen 192.168.252.10:443 default_server ssl http2;
3.
4.  ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
    ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES256-
5.  SHA384:ECDHE-RSA-AES256-SHA384";
6.
7.  ssl_prefer_server_ciphers on;
8.
9.  ssl_ecdh_curve secp521r1:secp384r1;
10.
11. ...
```

```
## Include this file to the server context (attach domain-a.com for specific listen directive):
server {

  include             /etc/nginx/https.conf;

  server_name         domain-a.com;

  ssl_certificate     domain-a.com.crt;
  ssl_certificate_key domain-a.com.key;

  ...

}

## Include this file to the server context (attach domain-b.com for specific listen directive):
server {

  include             /etc/nginx/https.conf;

  server_name         domain-b.com;

  ssl_certificate     domain-b.com.crt;
  ssl_certificate_key domain-b.com.key;

  ...

}
```

使用 geo/map 模块替代 allow/deny

> 使用地图或地理模块（其中之一）可以防止用户滥用您的服务器。这样就可以创建变量，其值取决于客户端 IP 地址。
>
> 由于仅在使用变量时才对其进行求值，因此甚至仅存在大量已声明的变量。地理位置变量不会为请求处理带来任何额外费用。
>
> 这些指令提供了阻止无效访问者的完美方法，例如使用 ngx_http_geoip_module。例如，geo 模块非常适合有条件地允许或拒绝 IP。
>
> geo 模块（注意：不要将此模块误认为是 GeoIP）在加载配置时会构建内存基数树。这与路由中使用的数据结构相同，并且查找速度非常快。如果每个网络有许多唯一值，那么较长的加载时间是由在数组中搜索数据重复项

> 引起的。否则，可能是由于插入基数树引起的。
>
> 我将两个模块都用于大型列表。您应该考虑一下，因为此规则要求使用多个 `if` 条件。我认为，对于简单的列表，毕竟允许/拒绝指令是更好的解决方案。看下面的例子：

```
## Allow/deny:
location /internal {

  include acls/internal.conf;
  allow   192.168.240.0/24;
  deny    all;

  ...

## vs geo/map:
location /internal {

  if ($globals_internal_map_acl) {
    set $pass 1;
  }

  if ($pass = 1) {
    proxy_pass http://localhost:80;
  }

  if ($pass != 1) {
    return 403;
  }

  ...

}
```

示例：

```
## Map module:
map $remote_addr $globals_internal_map_acl {

  ## Status code:
  ##  - 0 = false
  ##  - 1 = true
  default 0;

```

```
    ### INTERNAL ###
    10.255.10.0/24 1;
    10.255.20.0/24 1;
    10.255.30.0/24 1;
    192.168.0.0/16 1;

}

## Geo module:
geo $globals_internal_geo_acl {

    ## Status code:
    ##  - 0 = false
    ##  - 1 = true
    default 0;

    ### INTERNAL ###
    10.255.10.0/24 1;
    10.255.20.0/24 1;
    10.255.30.0/24 1;
    192.168.0.0/16 1;

}
```

Map 所有事物

> 使用地图管理大量重定向，并使用它们来自定义键/值对。
>
> map 指令可映射字符串，因此可以表示例如 192.168.144.0/24 作为正则表达式，并继续使用 map 指令。
>
> Map 模块提供了一种更优雅的解决方案，用于清晰地解析大量正则表达式，例如 用户代理，引荐来源。
>
> 您还可以对地图使用 include 指令，这样配置文件看起来会很漂亮。

示例：

```
map $http_user_agent $device_redirect {

    default "desktop";

    ~(?i)ip(hone|od) "mobile";
    ~(?i)android.*(mobile|mini) "mobile";
    ~Mobile.+Firefox "mobile";
```

```
    ~^HTC "mobile";
    ~Fennec "mobile";
    ~IEMobile "mobile";
    ~BB10 "mobile";
    ~SymbianOS.*AppleWebKit "mobile";
    ~Opera\sMobi "mobile";

}

## Turn on in a specific context (e.g. location):
if ($device_redirect = "mobile") {

    return 301 https://m.domain.com$request_uri;

}
```

为所有未匹配的路径设置根路径

> 为请求设置服务器指令内部的全局根路径。 它为未定义的位置指定根路径。
>
> 根据官方文档:
>
> 如果您在每个位置块中添加一个根路径，则不匹配的位置块将没有根路径。因此，重要的是，根指令必须在您的位置块之前发生，然后根目录指令可以在需要时覆盖该指令。

示例:

```
server {

    server_name domain.com;

    root /var/www/domain.com/public;

    location / {

        ...

    }

    location /api {

        ...

```

```
    }

    location /static {

      root /var/www/domain.com/static;

      ...

    }

}
```

使用 return 指令进行 URL 重定向（301、302）

> 这是一个简单的规则。 您应该使用服务器块和 return 语句，因为它们比评估 RegEx 更快。
>
> 因为 Nginx 停止处理请求（而不必处理正则表达式），所以它更加简单快捷。

示例

```
server {

    server_name www.example.com;

    ## return    301 https://$host$request_uri;
    return      301 $scheme://www.example.com$request_uri;

}
```

配置日志轮换策略

> 日志文件为您提供有关服务器活动和性能以及可能出现的任何问题的反馈。 它们记录了有关请求和 Nginx 内部的详细信息。 不幸的是，日志使用了更多的磁盘空间。
>
> 您应该定义一个过程，该过程将定期存档当前日志文件并启动一个新日志文件，重命名并有选择地压缩当前日志文件，删除旧日志文件，并强制日志记录系统开始使用新日志文件。
>
> 我认为最好的工具是 logrotate。 如果我想自动管理日志，也想睡个好觉，那么我会在任何地方使用它。 这是一个旋转日志的简单程序，使用 crontab 可以工作。 它是计划的工作，而不是守护程序，因此无需重新加载其配置。

示例：

- 手动旋转

```
## Check manually (all log files):
logrotate -dv /etc/logrotate.conf

## Check manually with force rotation (specific log file):
logrotate -dv --force /etc/logrotate.d/nginx
```

- 自动旋转

```
cat > /etc/logrotate.d/nginx << __EOF__
/var/log/nginx/*.log {
  daily
  missingok
  rotate 14
  compress
  delaycompress
  notifempty
  create 0640 nginx nginx
  sharedscripts
  prerotate
    if [ -d /etc/logrotate.d/httpd-prerotate ]; then \
      run-parts /etc/logrotate.d/httpd-prerotate; \
    fi \
  endscript
  postrotate
    ## test ! -f /var/run/nginx.pid || kill -USR1 `cat /var/run/nginx.pid`
    invoke-rc.d nginx reload >/dev/null 2>&1
  endscript
}

/var/log/nginx/localhost/*.log {
  daily
  missingok
  rotate 14
  compress
  delaycompress
  notifempty
  create 0640 nginx nginx
  sharedscripts
  prerotate
    if [ -d /etc/logrotate.d/httpd-prerotate ]; then \
      run-parts /etc/logrotate.d/httpd-prerotate; \
    fi \
```

```
    endscript
    postrotate
      ## test ! -f /var/run/nginx.pid || kill -USR1 `cat /var/run/nginx.pid`
      invoke-rc.d nginx reload >/dev/null 2>&1
    endscript
}

/var/log/nginx/domains/example.com/*.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 nginx nginx
    sharedscripts
    prerotate
      if [ -d /etc/logrotate.d/httpd-prerotate ]; then \
        run-parts /etc/logrotate.d/httpd-prerotate; \
      fi \
    endscript
    postrotate
      ## test ! -f /var/run/nginx.pid || kill -USR1 `cat /var/run/nginx.pid`
      invoke-rc.d nginx reload >/dev/null 2>&1
    endscript
}
__EOF__
```

不要重复索引指令，只能在 http 块中使用

> 一次使用 index 指令。 它只需要在您的 http 上下文中发生，并将在下面继承。
>
> 我认为我们在复制相同规则时应格外小心。 但是，当然，规则的重复有时是可以的，或者不一定是大麻烦。

示例：

错误配置

```
http {

  ...

  index index.php index.htm index.html;
```

```
  server {
    server_name www.example.com;
    location / {
      index index.php index.html index.$geo.html;
      ...
    }
  }
  server {
    server_name www.example.com;
    location / {
      index index.php index.htm index.html;
      ...
    }
    location /data {
      index index.php;
      ...
    }
    ...
}
```

正确配置

```
http {
```

```
 2.
 3.    ...
 4.
 5.    index index.php index.htm index.html index.$geo.html;
 6.
 7.    server {
 8.
 9.      server_name www.example.com;
10.
11.      location / {
12.
13.        ...
14.
15.      }
16.
17.    }
18.
19.    server {
20.
21.      server_name www.example.com;
22.
23.      location / {
24.
25.        ...
26.
27.      }
28.
29.      location /data {
30.
31.        ...
32.
33.      }
34.
35.      ...
36.
37.  }
```

Debugging

使用自定义日志格式

> 您可以在 Nginx 配置中作为变量访问的任何内容都可以记录，包括非标准的 HTTP 标头等。因此，这是一种针对特定情况创建自己的日志格式的简单方法。
>
> 这对于调试特定的 `location` 指令非常有帮助。

示例：

```
## Default main log format from the Nginx repository:
log_format main
                '$remote_addr - $remote_user [$time_local] "$request" '
                '$status $body_bytes_sent "$http_referer" '
                '"$http_user_agent" "$http_x_forwarded_for"';

## Extended main log format:
log_format main-level-0
                '$remote_addr - $remote_user [$time_local] '
                '"$request_method $scheme://$host$request_uri '
                '$server_protocol" $status $body_bytes_sent '
                '"$http_referer" "$http_user_agent" '
                '$request_time';

## Debug log formats:
log_format debug-level-0
                '$remote_addr - $remote_user [$time_local] '
                '"$request_method $scheme://$host$request_uri '
                '$server_protocol" $status $body_bytes_sent '
                '$request_id $pid $msec $request_time '
                '$upstream_connect_time $upstream_header_time '
                '$upstream_response_time "$request_filename" '
                '$request_completion';
```

使用调试模式来跟踪意外行为

> 通常，`error_log` 指令是在 `main` 中指定的，但是也可以在 `server` 或 `location` 块中指定，全局设置将被覆盖，并且这个 `error_log` 指令将设置其自己的日志文件路径和日志记录级别。
>
> 如果要记录 `ngx_http_rewrite_module` (at the notice level)，应该在

> `http` 、 `server` 或 `location` 块中开启 `rewrite_log on;` 。
>
> 注意：
>
> - 永远不要将调试日志记录留在生产环境中的文件上
> - 不要忘记在流量非常高的站点上恢复 `error_log` 的调试级别
> - 必须使用日志回滚政策

示例：

- 将 debug 信息写入文件

```
## Turn on in a specific context, e.g.:
##   - global    - for global logging
##   - http      - for http and all locations logging
##   - location  - for specific location
error_log /var/log/nginx/error-debug.log debug;
```

- 将 debug 信息写入内存

```
error_log memory:32m debug;
```

- IP 地址/范围的调试日志：

```
events {

  debug_connection    192.168.252.15/32;
  debug_connection    10.10.10.0/24;

}
```

- 为不同服务器设置不同 Debug 配置

```
error_log /var/log/nginx/debug.log debug;

...

http {

  server {

    ## To enable debugging:
    error_log /var/log/nginx/domain.com/domain.com-debug.log debug;
```

```
    ## To disable debugging:
    error_log /var/log/nginx/domain.com/domain.com-debug.log;

    ...

  }

}
```

核心转储

> 核心转储基本上是程序崩溃时内存的快照。
>
> Nginx 是一个非常稳定的守护程序，但是有时可能会发生正在运行的 Nginx 进程独特终止的情况。
>
> 如果要保存内存转储，它可以确保应启用两个重要的指令，但是，为了正确处理内存转储，需要做一些事情。有关它的完整信息，请参见转储进程的内存（来自本手册）。
>
> 当您的 Nginx 实例收到意外错误或崩溃时，应始终启用核心转储。

示例：

```
worker_rlimit_core    500m;
worker_rlimit_nofile  65535;
working_directory     /var/dump/nginx;
```

性能

工作进程数

> `worker_processes` - 用于设置 Nginx 的工作进程数。

- worker_processes 的默认值为 1。
- 设置 worker_processes 的安全做法是将其设为 auto，则启动 Nginx 时会自动分配工作进程数。当然，也可以显示的设置一个工作进程数值。
- 一般一个进程足够了，你可以把连接数设得很大。（worker_processes: 1，worker_connections: 10,000）如果有 SSL、gzip 这些比较消耗 CPU 的工作，而且是多核 CPU 的话，可以设为和 CPU 的数量一样。或者要处理很多很多的小文件，而且文件总大小比内存大很多的时候，也可以把进程数增加，以充分利用 IO 带宽（主要似乎是 IO 操作有 block）

示例：

```
## The safest way:
worker_processes auto;

## VCPU = 4 , expr $(nproc --all) - 1
worker_processes 3;
```

最大连接数

> `worker_connections` - 单个 Nginx 工作进程允许同时建立的外部连接的数量。数字越大，能同时处理的连接越多。

`worker_connections` 不是随便设置的，而是与两个指标有重要关联：

- 内存
  - 每个连接数分别对应一个 read_event、一个 write_event 事件，一个连接数大概占用 232 字节，2 个事件总占用 96 字节，那么一个连接总共占用 328 字节，通过数学公式可以算出 100000 个连接数大概会占用 31M = 100000 * 328 / 1024 / 1024，当然这只是 nginx 启动时，worker_connections 连接数所占用的 nginx。
- 操作系统级别”进程最大可打开文件数“。
  - 进程最大可打开文件数受限于操作系统，可通过 `ulimit -n` 命令查询，以前是 1024，现在是 65535。
  - nginx 提供了 worker_rlimit_nofile 指令，这是除了 ulimit 的一种设置可用的描述符的方式。 该指令与使用 ulimit 对用户的设置是同样的效果。此指令的值将覆盖

ulimit 的值，如：worker_rlimit_nofile 20960; 设置 ulimits：ulimit -SHn 65535

使用 HTTP/2

> HTTP / 2 将使我们的应用程序更快，更简单且更可靠。 HTTP / 2 的主要目标是通过启用完整的请求和响应多路复用来减少延迟，通过有效压缩 HTTP 标头字段来最小化协议开销，并增加对请求优先级和服务器推送的支持。
>
> HTTP / 2 与 HTTP / 1.1 向后兼容，因此有可能完全忽略它，并且一切都会像以前一样继续工作，因为如果不支持 HTTP / 2 的客户端永远不会向服务器请求 HTTP / 2 通讯升级：它们之间的通讯将完全是 HTTP1 / 1。
>
> 请注意，HTTP / 2 在单个 TCP 连接中多路复用许多请求。 通常，当使用 HTTP / 2 时，将与服务器建立单个 TCP 连接。
>
> 您还应该包括 ssl 参数，这是必需的，因为浏览器不支持未经加密的 HTTP / 2。
>
> HTTP / 2 对旧的和不安全的密码有一个非常大的黑名单，因此您应该避免使用它们。

示例：

```
server {

  listen 10.240.20.2:443 ssl http2;

  ...
```

维护 SSL 会话

> 客户端每次发出请求时都进行新的 SSL 握手的需求。默认情况下，内置会话缓存并不是最佳选择，因为它只能由一个工作进程使用，并且可能导致内存碎片，最好使用共享缓存。
>
> 使用 `ssl_session_cache` 时，通过 SSL 保持连接的性能可能会大大提高。10M 的值是一个很好的起点（1MB 共享缓存可以容纳大约 4,000 个会话）。通过共享，所有工作进程之间共享一个缓存（可以在多个虚拟服务器中使用相同名称的缓存）。
>
> 但是，大多数服务器不清除会话或票证密钥，因此增加了服务器受到损害将泄漏先前（和将来）连接中的数据的风险。

示例：

```
ssl_session_cache   shared:NGX_SSL_CACHE:10m;
ssl_session_timeout 12h;
ssl_session_tickets off;
```

```
4.  ssl_buffer_size     1400;
```

尽可能在 server_name 指令中使用确切名称

> 确切名称，以星号开头的通配符名称和以星号结尾的通配符名称存储在绑定到侦听端口的三个哈希表中。
>
> 首先搜索确切名称哈希表。 如果未找到名称，则搜索具有以星号开头的通配符名称的哈希表。 如果未在此处找到名称，则搜索带有通配符名称以星号结尾的哈希表。 搜索通配符名称哈希表比搜索精确名称哈希表要慢，因为名称是按域部分搜索的。
>
> 正则表达式是按顺序测试的，因此是最慢的方法，并且不可缩放。由于这些原因，最好在可能的地方使用确切的名称。

示例：

```
1.  ## It is more efficient to define them explicitly:
2.  server {
3.
4.      listen       192.168.252.10:80;
5.
6.      server_name  example.org  www.example.org  *.example.org;
7.
8.      ...
9.
10. }
11.
12. ## Than to use the simplified form:
13. server {
14.
15.     listen       192.168.252.10:80;
16.
17.     server_name  .example.org;
18.
19.     ...
20.
21. }
```

避免使用 `if` 检查 `server_name`

> 当 Nginx 收到请求时，无论请求的是哪个子域，无论是 www.example.com 还是普通的 example.com，如果始终对 if 指令进行评估。 由于您是在请求 Nginx 检查每个请求的 Host 标头。 效率极低。
>
> 而是使用两个服务器指令，如下面的示例。 这种方法降低了 Nginx 的处理要求。

示例：

错误配置

```
server {

  server_name                domain.com www.domain.com;

  if ($host = www.domain.com) {

    return                   301 https://domain.com$request_uri;

  }

  server_name                domain.com;

  ...

}
```

正确配置

```
server {

    server_name              www.domain.com;

    return                   301 $scheme://domain.com$request_uri;

    ## If you force your web traffic to use HTTPS:
    ##                        301 https://domain.com$request_uri;

    ...

}

server {

    listen                   192.168.252.10:80;

    server_name              domain.com;

    ...
```

```
21.
22. }
```

使用 `$request_uri` 来避免使用正则表达式

> 使用内置变量 `$request_uri` ，我们可以完全避免进行任何捕获或匹配。默认情况下，正则表达式的代价较高，并且会降低性能。
>
> 此规则用于解决将 URL 不变地传递到新主机，确保仅通过现有 URI 进行返回的效率更高。

示例：

错误配置

```
1. ## 1)
2. rewrite ^/(.*)$ https://example.com/$1 permanent;
3.
4. ## 2)
5. rewrite ^ https://example.com$request_uri? permanent;
```

正确配置

```
1. return 301 https://example.com$request_uri;
```

使用 `try_files` 指令确认文件是否存在

> `try_files` is definitely a very useful thing. You can use `try_files` directive to check a file exists in a specified order.
>
> You should use `try_files` instead of `if` directive. It’s definitely better way than using `if` for this action because `if` directive is extremely inefficient since it is evaluated every time for every request.
>
> The advantage of using `try_files` is that the behavior switches immediately with one command. I think the code is more readable also.
>
> `try_files` allows you:
>
> - to check if the file exists from a predefined list
> - to check if the file exists from a specified directory
> - to use an internal redirect if none of the files are found

示例：

错误配置

```
    ...

    root /var/www/example.com;

    location /images {

      if (-f $request_filename) {

        expires 30d;
        break;

      }

    ...

}
```

正确配置

```
    ...

    root /var/www/example.com;

    location /images {

      try_files $uri =404;

    ...

}
```

使用 return 代替 rewrite 来做重定向

> 您应该使用服务器块和 return 语句，因为它们比通过位置块评估 RegEx 更简单，更快捷。 该指令停止处理，并将指定的代码返回给客户端。

示例：

错误配置

```
server {

  ...

  if ($host = api.domain.com) {

    rewrite     ^/(.*)$ http://example.com/$1 permanent;

  }

  ...
```

正确配置

```
server {

  ...

  if ($host = api.domain.com) {

    return      403;

    ## or other examples:
    ##   return    301 https://domain.com$request_uri;
    ##   return    301 $scheme://$host$request_uri;

  }

  ...
```

开启 PCRE JIT 来加速正则表达式处理

> 允许使用 JIT 的正则表达式来加速他们的处理。
>
> 通过与 PCRE 库编译 Nginx 的，你可以用你的 location 块进行复杂的操作和使用功能强大的 return 和 rewrite。
>
> PCRE JIT 可以显著加快正则表达式的处理。 Nginx 的与 pcre_jit 比没有更快的幅度。
>
> 如果你试图在使用 pcre_jit;没有可用的 JIT，或者 Nginx 的与现有 JIT，但当前加载 PCRE 库编译不支持 JIT，将配置解析时发出警告。
>
> 当您编译使用 NGNIX 配置 PCRE 库时，才需要—with-PCRE-JIT 时（./configure —with-PCRE =）。

> 当使用系统 PCRE 库 JIT 是否被支持依赖于库是如何被编译。
>
> 从 Nginx 的文档:
>
> JIT 正在从与—enable-JIT 配置参数内置 8.20 版本开始 PCRE 库提供。当 PCRE 库与 nginx 的内置（—with-PCRE =）时，JIT 支持经由—with-PCRE-JIT 配置参数使能。

示例:

```
## In global context:
pcre_jit on;
```

进行精确的位置匹配以加快选择过程

> 精确的位置匹配通常用于通过立即结束算法的执行来加快选择过程。

示例:

```
## Matches the query / only and stops searching:
location = / {

  ...

}

## Matches the query /v9 only and stops searching:
location = /v9 {

  ...

}

...

## Matches any query due to the fact that all queries begin at /,
## but regular expressions and any longer conventional blocks will be matched
at first place:
location / {

  ...

}
```

使用 `limit_conn` 改善对下载速度的限制

> Nginx provides two directives to limiting download speed:
>
> Nginx 提供了两个指令来限制下载速度:
>
> - `limit_rate_after` - 设置 limit_rate 指令生效之前传输的数据量
> - `limit_rate` - 允许您限制单个客户端连接的传输速率
>
> 此解决方案限制了每个连接的 Nginx 下载速度，因此，如果一个用户打开多个（例如） 视频文件，则可以下载 `X * 连接到视频文件的次数` 。

示例:

```
1.  ## Create limit connection zone:
2.  limit_conn_zone $binary_remote_addr zone=conn_for_remote_addr:1m;
3.
4.  ## Add rules to limiting the download speed:
5.  limit_rate_after 1m;  ## run at maximum speed for the first 1 megabyte
6.  limit_rate 250k;      ## and set rate limit after 1 megabyte
7.
8.  ## Enable queue:
9.  location /videos {
10.
11.   ## Max amount of data by one client: 10 megabytes (limit_rate_after * 10)
12.   limit_conn conn_for_remote_addr 10;
13.
14.   ...
```

Hardening

In this chapter I will talk about some of the Nginx hardening approaches and security standards.

:beginner: Always keep Nginx up-to-date

Rationale

> Nginx is a very secure and stable but vulnerabilities in the main binary itself do pop up from time to time. It’s the main reason for keep Nginx up-to-date as hard as you can.
>
> A very safe way to plan the update is once a new stable version is released but for me the most common way to handle Nginx updates is to wait a few weeks after the stable release.
>
> Before update/upgrade Nginx remember about do it on the testing environment.
>
> Most modern GNU/Linux distros will not push the latest version of Nginx into their default package lists so maybe you should consider install it from sources.

External resources

- Installing from prebuilt packages (from this handbook)
- Installing from source (from this handbook)

:beginner: Run as an unprivileged user

Rationale

> There is no real difference in security just by changing the process owner name. On the other hand in security, the principle of least privilege states that an entity should be given no more permission than necessary to accomplish its goals within a given system. This way only master process runs as root.
>
> This is the default Nginx behaviour, but remember to check it.

Example

```
## Edit nginx.conf:
user nginx;

## Set owner and group for root (app, default) directory:
chown -R nginx:nginx /var/www/domain.com
```

External resources

- Why does nginx starts process as root?

:beginner: Disable unnecessary modules

Rationale

> It is recommended to disable any modules which are not required as this will minimise the risk of any potential attacks by limiting the operations allowed by the web server.
>
> The best way to unload unused modules is use the `configure` option during installation. If you have static linking a shared module you should re-compile Nginx.
>
> Use only high quality modules and remember about that:
>
> *Unfortunately, many third-party modules use blocking calls, and users (and sometimes even the developers of the modules) aren't aware of the drawbacks. Blocking operations can ruin Nginx performance and must be avoided at all costs.*

Example

```
## 1) During installation:
./configure --without-http_autoindex_module

## 2) Comment modules in the configuration file e.g. modules.conf:
## load_module                 /usr/share/nginx/modules/ndk_http_module.so;
## load_module
/usr/share/nginx/modules/ngx_http_auth_pam_module.so;
## load_module
/usr/share/nginx/modules/ngx_http_cache_purge_module.so;
## load_module
/usr/share/nginx/modules/ngx_http_dav_ext_module.so;
load_module                  /usr/share/nginx/modules/ngx_http_echo_module.so;
## load_module
/usr/share/nginx/modules/ngx_http_fancyindex_module.so;
load_module
/usr/share/nginx/modules/ngx_http_geoip_module.so;
load_module
/usr/share/nginx/modules/ngx_http_headers_more_filter_module.so;
## load_module
/usr/share/nginx/modules/ngx_http_image_filter_module.so;
## load_module                 /usr/share/nginx/modules/ngx_http_lua_module.so;
load_module                  /usr/share/nginx/modules/ngx_http_perl_module.so;
## load_module                 /usr/share/nginx/modules/ngx_mail_module.so;
## load_module                 /usr/share/nginx/modules/ngx_nchan_module.so;
## load_module                 /usr/share/nginx/modules/ngx_stream_module.so;
```

External resources

- nginx-modules

:beginner: Protect sensitive resources

Rationale

> Hidden directories and files should never be web accessible - sometimes critical data are published during application deploy. If you use control version system you should defninitely drop the access to the critical hidden directories like a `.git` or `.svn` to prevent expose source code of your application.
>
> Sensitive resources contains items that abusers can use to fully recreate the source code used by the site and look for bugs, vulnerabilities, and exposed passwords.

Example

```
if ($request_uri ~ "/\.git") {

  return 403;

}

## or
location ~ /\.git {

  deny all;

}

## or
location ~* ^.*(\.(?:git|svn|htaccess))$ {

  return 403;

}

## or all . directories/files excepted .well-known
location ~ /\.(?!well-known\/) {

  deny all;

}
```

External resources

- Hidden directories and files as a source of sensitive information about web application

:beginner: Hide Nginx version number

Rationale

> Disclosing the version of Nginx running can be undesirable, particularly in environments sensitive to information disclosure.

But the “Official Apache Documentation (Apache Core Features)” (yep, it’s not a joke…) say:

> *Setting ServerTokens to less than minimal is not recommended because it makes it more difficult to debug interoperational problems. Also note that disabling the Server: header does nothing at all to make your server more secure. The idea of “security through obscurity” is a myth and leads to a false sense of safety.*

Example

```
1.  server_tokens off;
```

External resources

- Remove Version from Server Header Banner in nginx
- Reduce or remove server headers

:beginner: Hide Nginx server signature

Rationale

> One of the easiest first steps to undertake, is to prevent the web server from showing its used software via the server header. Certainly, there are several reasons why you would like to change the server header. It could be security, it could be redundant systems, load balancers etc.
>
> In my opinion there is no real reason or need to show this much information about your server. It is easy to look up particular vulnerabilities once you know the version number.
>
> You should compile Nginx from sources with `ngx_headers_more` to used

> `more_set_headers` directive or use a `nginx-remove-server-header.patch`.

Example

```
more_set_headers "Server: Unknown";
```

External resources

- Shhh… don’t let your response headers talk too loudly
- How to change (hide) the Nginx Server Signature?

:beginner: Hide upstream proxy headers

Rationale

> Securing a server goes far beyond not showing what’s running but I think less is more is better.
>
> When Nginx is used to proxy requests to an upstream server (such as a PHP-FPM instance), it can be beneficial to hide certain headers sent in the upstream response (e.g. the version of PHP running).

Example

```
proxy_hide_header X-Powered-By;
proxy_hide_header X-AspNetMvc-Version;
proxy_hide_header X-AspNet-Version;
proxy_hide_header X-Drupal-Cache;
```

External resources

- Remove insecure http headers

:beginner: Force all connections over TLS

Rationale

> TLS provides two main services. For one, it validates the identity of the server that the user is connecting to for the user. It also protects the transmission of sensitive information from the user to the server.
>
> In my opinion you should always use HTTPS instead of HTTP to protect your website, even if it doesn’t handle sensitive communications. The application can have many

> sensitive places that should be protected.
>
> Always put login page, registration forms, all subsequent authenticated pages, contact forms, and payment details forms in HTTPS to prevent injection and sniffing. Them must be accessed only over TLS to ensure your traffic is secure.
>
> If page is available over TLS, it must be composed completely of content which is transmitted over TLS. Requesting subresources using the insecure HTTP protocol weakens the security of the entire page and HTTPS protocol. Modern browsers should blocked or report all active mixed content delivered via HTTP on pages by default.
>
> Also remember to implement the HTTP Strict Transport Security (HSTS).
>
> We have currently the first free and open CA - Let's Encrypt - so generating and implementing certificates has never been so easy. It was created to provide free and easy-to-use TLS and SSL certificates.

Example

- force all traffic to use TLS:

```
server {

  listen 10.240.20.2:80;

  server_name domain.com;

  return 301 https://$host$request_uri;

}

server {

  listen 10.240.20.2:443 ssl;

  server_name domain.com;

  ...

}
```

- force e.g. login page to use TLS:

```
server {
```

```
2.
3.    listen 10.240.20.2:80;
4.
5.    server_name domain.com;
6.
7.    ...
8.
9.    location ^~ /login {
10.
11.     return 301 https://domain.com$request_uri;
12.
13.   }
14.
15. }
```

External resources

- Should we force user to HTTPS on website?
- Force a user to HTTPS
- Let’s Encrypt Documentation

:beginner: Use only the latest supported OpenSSL version

Rationale

> Before start see Release Strategy Policies and Changelog on the OpenSSL website.
>
> Criteria for choosing OpenSSL version can vary and it depends all on your use.
>
> The latest versions of the major OpenSSL library are (may be changed):
>
> - the next version of OpenSSL will be 3.0.0
> - version 1.1.1 will be supported until 2023-09-11 (LTS)
>   - last minor version: 1.1.1c (May 23, 2019)
> - version 1.1.0 will be supported until 2019-09-11
>   - last minor version: 1.1.0k (May 28, 2018)
> - version 1.0.2 will be supported until 2019-12-31 (LTS)
>   - last minor version: 1.0.2s (May 28, 2018)
> - any other versions are no longer supported
>
> In my opinion the only safe way is based on the up-to-date and still supported version of the OpenSSL. And what’s more, I recommend to hang on to the latest versions (e.g. 1.1.1).

> If your system repositories do not have the newest OpenSSL, you can do the compilation process (see OpenSSL sub-section).

External resources

- OpenSSL Official Website
- OpenSSL Official Blog
- OpenSSL Official Newslog

:beginner: Use min. 2048-bit private keys

Rationale

> Advisories recommend 2048 for now. Security experts are projecting that 2048 bits will be sufficient for commercial use until around the year 2030 (as per NIST).
>
> The latest version of FIPS-186 also say the U.S. Federal Government generate (and use) digital signatures with 1024, 2048, or 3072 bit key lengths.
>
> Generally there is no compelling reason to choose 4096 bit keys over 2048 provided you use sane expiration intervals.
>
> If you want to get **A+ with 100%s on SSL Lab** (for Key Exchange) you should definitely use 4096 bit private keys. That's the main reason why you should use them.
>
> Longer keys take more time to generate and require more CPU and power when used for encrypting and decrypting, also the SSL handshake at the start of each connection will be slower. It also has a small impact on the client side (e.g. browsers).
>
> You can test above on your server with `openssl speed rsa` but remember: in OpenSSL speed tests you see difference on block cipher speed, while in real life most cpu time is spent on asymmetric algorithms during ssl handshake. On the other hand, modern processors are capable of executing at least 1k of RSA 1024-bit signs per second on a single core, so this isn't usually an issue.
>
> Use of alternative solution: ECC Certificate Signing Request (CSR) - `ECDSA` certificates contain an `ECC` public key. `ECC` keys are better than `RSA & DSA` keys in that the `ECC` algorithm is harder to break.

The "SSL/TLS Deployment Best Practices" book say:

> *The cryptographic handshake, which is used to establish secure connections, is an operation whose cost is highly influenced by private key size. Using a key that is too short is insecure, but using a key that is too long will result in "too much" security and slow operation. For most web sites, using RSA keys stronger than 2048*

> *bits and ECDSA keys stronger than 256 bits is a waste of CPU power and might impair user experience. Similarly, there is little benefit to increasing the strength of the ephemeral key exchange beyond 2048 bits for DHE and 256 bits for ECDHE.*

Konstantin Ryabitsev (Reddit):

> *Generally speaking, if we ever find ourselves in a world where 2048-bit keys are no longer good enough, it won't be because of improvements in brute-force capabilities of current computers, but because RSA will be made obsolete as a technology due to revolutionary computing advances. If that ever happens, 3072 or 4096 bits won't make much of a difference anyway. This is why anything above 2048 bits is generally regarded as a sort of feel-good hedging theatre.*

My recommendation:

> Use 2048-bit key instead of 4096-bit at this moment.

Example

```
### Example (RSA):
( _fd="domain.com.key" ; _len="2048" ; openssl genrsa -out ${_fd} ${_len} )

## Let's Encrypt:
certbot certonly -d domain.com -d www.domain.com --rsa-key-size 2048

### Example (ECC):
## _curve: prime256v1, secp521r1, secp384r1
( _fd="domain.com.key" ; _fd_csr="domain.com.csr" ; _curve="prime256v1" ; \
openssl ecparam -out ${_fd} -name ${_curve} -genkey ; \
openssl req -new -key ${_fd} -out ${_fd_csr} -sha256 )

## Let's Encrypt (from above):
certbot --csr ${_fd_csr} -[other-args]
```

For `x25519` :

```
( _fd="private.key" ; _curve="x25519" ; \
openssl genpkey -algorithm ${_curve} -out ${_fd} )
```

:arrow_right: ssllabs score: **100%**

```
( _fd="domain.com.key" ; _len="2048" ; openssl genrsa -out ${_fd} ${_len} )
```

```
2.
3.  ## Let's Encrypt:
4.  certbot certonly -d domain.com -d www.domain.com
```

:arrow_right: ssllabs score: **90%**

External resources

- Key Management Guidelines by NIST
- Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
- FIPS PUB 186-4 - Digital Signature Standard (DSS) [pdf]
- Cryptographic Key Length Recommendations
- So you’re making an RSA key for an HTTPS certificate. What key size do you use?
- RSA Key Sizes: 2048 or 4096 bits?
- Create a self-signed ECC certificate

:beginner: Keep only TLS 1.3 and TLS 1.2

Rationale

> It is recommended to run TLS 1.2/1.3 and fully disable SSLv2, SSLv3, TLS 1.0 and TLS 1.1 that have protocol weaknesses and uses older cipher suites (do not provide any modern ciper modes).
>
> TLS 1.0 and TLS 1.1 must not be used (see Deprecating TLSv1.0 and TLSv1.1) and were superceded by TLS 1.2, which has now itself been superceded by TLS 1.3. They are also actively being deprecated in accordance with guidance from government agencies (e.g. NIST SP 80052r2) and industry consortia such as the Payment Card Industry Association (PCI) [PCI-TLS1].
>
> TLS 1.2 and TLS 1.3 are both without security issues. Only these versions provides modern cryptographic algorithms. TLS 1.3 is a new TLS version that will power a faster and more secure web for the next few years. What’s more, TLS 1.3 comes without a ton of stuff (was removed): renegotiation, compression, and many legacy algorithms: `DSA`, `RC4`, `SHA1`, `MD5`, `CBC` MAC-then-Encrypt ciphers. TLS 1.0 and TLS 1.1 protocols will be removed from browsers at the beginning of 2020.
>
> TLS 1.2 does require careful configuration to ensure obsolete cipher suites with identified vulnerabilities are not used in conjunction with it. TLS 1.3 removes the need to make these decisions. TLS 1.3 version also improves TLS 1.2 security, privace and performance issues.

> Before enabling specific protocol version, you should check which ciphers are supported by the protocol. So if you turn on TLS 1.2 and TLS 1.3 both remember about the correct (and strong) ciphers to handle them. Otherwise, they will not be anyway works without supported ciphers (no TLS handshake will succeed).
>
> I think the best way to deploy secure configuration is: enable TLS 1.2 without any `CBC` Ciphers (is safe enough) only TLS 1.3 is safer because of its handling improvement and the exclusion of everything that went obsolete since TLS 1.2 came up.
>
> If you told Nginx to use TLS 1.3, it will use TLS 1.3 only where is available. Nginx supports TLS 1.3 since version 1.13.0 (released in April 2017), when built against OpenSSL 1.1.1 or more.
>
> For TLS 1.3, think about using `ssl_early_data` to allow TLS 1.3 0-RTT handshakes.

My recommendation:

> Use only TLSv1.3 and TLSv1.2.

Example

TLS 1.3 + 1.2:

```
1.  ssl_protocols TLSv1.3 TLSv1.2;
```

TLS 1.2:

```
1.  ssl_protocols TLSv1.2;
```

:arrow_right: ssllabs score: **100%**

TLS 1.3 + 1.2 + 1.1:

```
1.  ssl_protocols TLSv1.3 TLSv1.2 TLSv1.1;
```

TLS 1.2 + 1.1:

```
1.  ssl_protocols TLSv1.2 TLSv1.1;
```

:arrow_right: ssllabs score: **95%**

External resources

- The Transport Layer Security (TLS) Protocol Version 1.2
- The Transport Layer Security (TLS) Protocol Version 1.3
- TLS1.2 - Every byte explained and reproduced
- TLS1.3 - Every byte explained and reproduced
- TLS1.3 - OpenSSLWiki
- TLS v1.2 handshake overview
- An Overview of TLS 1.3 - Faster and More Secure
- A Detailed Look at RFC 8446 (a.k.a. TLS 1.3)
- Differences between TLS 1.2 and TLS 1.3
- TLS 1.3 in a nutshell
- TLS 1.3 is here to stay
- How to enable TLS 1.3 on Nginx
- How to deploy modern TLS in 2019?
- Deploying TLS 1.3: the great, the good and the bad
- Downgrade Attack on TLS 1.3 and Vulnerabilities in Major TLS Libraries
- Phase two of our TLS 1.0 and 1.1 deprecation plan
- Deprecating TLS 1.0 and 1.1 - Enhancing Security for Everyone
- TLS/SSL Explained – Examples of a TLS Vulnerability and Attack, Final Part
- This POODLE bites: exploiting the SSL 3.0 fallback
- Are You Ready for 30 June 2018? Saying Goodbye to SSL/early TLS
- Deprecating TLSv1.0 and TLSv1.1

:beginner: Use only strong ciphers

Rationale

> This parameter changes quite often, the recommended configuration for today may be out of date tomorrow.
>
> To check ciphers supported by OpenSSL on your server: `openssl ciphers -s -v`, `openssl ciphers -s -v ECDHE` or `openssl ciphers -s -v DHE`.
>
> For more security use only strong and not vulnerable cipher suites. Place `ECDHE` and `DHE` suites at the top of your list. The order is important because `ECDHE` suites are faster, you want to use them whenever clients supports them. Ephemeral `DHE/ECDHE` are recommended and support Perfect Forward Secrecy.
>
> For backward compatibility software components you should use less restrictive ciphers. Not only that you have to enable at least one special `AES128` cipher for HTTP/2 support regarding to RFC7540: TLS 1.2 Cipher Suites, you also have to allow `prime256` elliptic curves which reduces the score for key exchange by

> another 10% even if a secure server preferred order is set.
>
> Also modern cipher suites (e.g. from Mozilla recommendations) suffers from compatibility troubles mainly because drops `SHA-1`. But be careful if you want to use ciphers with `HMAC-SHA-1` - there's a perfectly good explanation why.
>
> If you want to get **A+ with 100%s on SSL Lab** (for Cipher Strength) you should definitely disable `128-bit` ciphers. That's the main reason why you should not use them.
>
> In my opinion `128-bit` symmetric encryption doesn't less secure. Moreover, there are about 30% faster and still secure. For example TLS 1.3 use `TLS_AES_128_GCM_SHA256 (0x1301)` (for TLS-compliant applications).
>
> It is not possible to control ciphers for TLS 1.3 without support from client to use new API for TLS 1.3 cipher suites. Nginx isn't able to influence that so at this moment it's always on (also if you disable potentially weak cipher from Nginx). On the other hand the ciphers in TLSv1.3 have been restricted to only a handful of completely secure ciphers by leading crypto experts.
>
> For TLS 1.2 you should consider disable weak ciphers without forward secrecy like ciphers with `CBC` algorithm. Using them also reduces the final grade because they don't use ephemeral keys. In my opinion you should use ciphers with `AEAD` (TLS 1.3 supports only these suites) encryption because they don't have any known weaknesses.
>
> Recently new vulnerabilities like Zombie POODLE, GOLDENDOODLE, 0-Length OpenSSL and Sleeping POODLE were published for websites that use `CBC` (Cipher Block Chaining) block cipher modes. These vulnerabilities are applicable only if the server uses TLS 1.2 or TLS 1.1 or TLS 1.0 with `CBC` cipher modes. Look at Zombie POODLE, GOLDENDOODLE, & How TLSv1.3 Can Save Us All presentation from Black Hat Asia 2019.
>
> Disable TLS cipher modes (all ciphers that start with `TLS_RSA_WITH_*`) that use RSA encryption because they are vulnerable to ROBOT attack. Not all servers that support RSA key exchange are vulnerable, but it is recommended to disable RSA key exchange ciphers as it does not support forward secrecy.
>
> You should also absolutely disable weak ciphers regardless of the TLS version do you use, like those with `DSS`, `DSA`, `DES/3DES`, `RC4`, `MD5`, `SHA1`, `null`, anon in the name.
>
> We have a nice online tool for testing compatibility cipher suites with user agents: CryptCheck. I think it will be very helpful for you.

My recommendation:

> Use only TLSv1.3 and TLSv1.2 with below cipher suites:

```
    ssl_ciphers "TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-256-GCM-SHA384:TLS13-AES-
    128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-
    RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
    POLY1305:DHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
    AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256";
```

Example

Cipher suites for TLS 1.3:

```
    ssl_ciphers "TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-256-GCM-SHA384";
```

Cipher suites for TLS 1.2:

```
    ssl_ciphers "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
    ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES256-SHA384";
```

:arrow_right: ssllabs score: **100%**

Cipher suites for TLS 1.3:

```
    ssl_ciphers "TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-256-GCM-SHA384:TLS13-AES-
    128-GCM-SHA256";
```

Cipher suites for TLS 1.2:

```
    ## 1)
    ssl_ciphers "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
    ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES256-SHA384";

    ## 2)
    ssl_ciphers "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
    AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
    POLY1305:DHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
    AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256";

    ## 3)
    ssl_ciphers "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
    ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-
    SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-
    SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256";

    ## 4)
```

```
ssl_ciphers "EECDH+CHACHA20:EDH+AESGCM:AES256+EECDH:AES256+EDH";
```

Cipher suites for TLS 1.1 + 1.2:

```
## 1)
ssl_ciphers "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256";

## 2)
ssl_ciphers "ECDHE-ECDSA-CHACHA20-POLY1305:ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:!AES256-GCM-SHA256:!AES256-GCM-SHA128:!aNULL:!MD5";
```

:arrow_right: ssllabs score: **90%**

This will also give a baseline for comparison with Mozilla SSL Configuration Generator:

- Modern profile with OpenSSL 1.1.0b (TLSv1.2)

```
ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256';
```

- Intermediate profile with OpenSSL 1.1.0b (TLSv1, TLSv1.1 and TLSv1.2)

```
ssl_ciphers 'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS';
```

External resources

- RFC 7525 - TLS Recommendations
- TLS Cipher Suites

- SSL/TLS: How to choose your cipher suite
- HTTP/2 and ECDSA Cipher Suites
- Which SSL/TLS Protocol Versions and Cipher Suites Should I Use?
- Recommendations for a cipher string by OWASP
- Recommendations for TLS/SSL Cipher Hardening by Acunetix
- Mozilla’s Modern compatibility suite
- Why use Ephemeral Diffie-Hellman
- Cipher Suite Breakdown
- Zombie POODLE and GOLDENDOODLE Vulnerabilities
- OpenSSL IANA Mapping
- Goodbye TLS_RSA

:beginner: Use more secure ECDH Curve

Rationale

> In my opinion your main source of knowledge should be The SafeCurves web site. This site reports security assessments of various specific curves.
>
> For a SSL server certificate, an “elliptic curve” certificate will be used only with digital signatures (`ECDSA` algorithm). Nginx provides directive to specifies a curve for `ECDHE` ciphers.
>
> `x25519` is a more secure (also with SafeCurves requirements) but slightly less compatible option. I think to maximise interoperability with existing browsers and servers, stick to `P-256 prime256v1` and `P-384 secp384r1` curves. Of course there’s tons of different opinions about `P-256` and `P-384` curves.
>
> NSA Suite B says that NSA uses curves `P-256` and `P-384` (in OpenSSL, they are designated as, respectively, `prime256v1` and `secp384r1`). There is nothing wrong with `P-521` , except that it is, in practice, useless. Arguably, `P-384` is also useless, because the more efficient `P-256` curve already provides security that cannot be broken through accumulation of computing power.
>
> Bernstein and Lange believe that the NIST curves are not optimal and there are better (more secure) curves that work just as fast, e.g. `x25519` .
>
> Keep an eye also on this:
>
> *Secure implementations of the standard curves are theoretically possible but very hard.*
>
> The SafeCurves say:
>
> - `NIST P-224` , `NIST P-256` and `NIST P-384` are UNSAFE

> From the curves described here only `x25519` is a curve meets all SafeCurves requirements.
>
> I think you can use `P-256` to minimise trouble. If you feel that your manhood is threatened by using a 256-bit curve where a 384-bit curve is available, then use `P-384` : it will increases your computational and network costs.
>
> If you use TLS 1.3 you should enable `prime256v1` signature algorithm. Without this SSL Lab reports `TLS_AES_128_GCM_SHA256 (0x1301)` signature as weak.
>
> If you do not set `ssl_ecdh_curve` , then Nginx will use its default settings, e.g. Chrome will prefer `x25519` , but it is **not recommended** because you can not control default settings (seems to be `P-256`) from the Nginx.
>
> Explicitly set `ssl_ecdh_curve X25519:prime256v1:secp521r1:secp384r1;` **decreases the Key Exchange SSL Labs rating**.
>
> Definitely do not use the `secp112r1` , `secp112r2` , `secp128r1` , `secp128r2` , `secp160k1` , `secp160r1` , `secp160r2` , `secp192k1` curves. They have a too small size for security application according to NIST recommendation.

My recommendation:

> Use only TLSv1.3 and TLSv1.2 and only strong ciphers with above curves:

```
ssl_ecdh_curve X25519:secp521r1:secp384r1:prime256v1;
```

Example

Curves for TLS 1.2:

```
ssl_ecdh_curve secp521r1:secp384r1:prime256v1;
```

:arrow_right: ssllabs score: **100%**

```
## Alternative (this one doesn’t affect compatibility, by the way; it’s just a question of the preferred order).

## This setup downgrade Key Exchange score but is recommended for TLS 1.2 + 1.3:
ssl_ecdh_curve X25519:secp521r1:secp384r1:prime256v1;
```

External resources

- Elliptic Curves for Security
- Standards for Efficient Cryptography Group
- SafeCurves: choosing safe curves for elliptic-curve cryptography
- A note on high-security general-purpose elliptic curves
- P-521 is pretty nice prime
- Safe ECC curves for HTTPS are coming sooner than you think
- Cryptographic Key Length Recommendations
- Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPST-001)>)
- Elliptic Curve performance: NIST vs Brainpool
- Which elliptic curve should I use?
- Elliptic Curve Cryptography for those who are afraid of maths

:beginner: Use strong Key Exchange with Perfect Forward Secrecy

Rationale

> To use a signature based authentication you need some kind of DH exchange (fixed or ephemeral/temporary), to exchange the session key. If you use it, Nginx will use the default Ephemeral Diffie-Hellman (`DHE`) paramaters to define how performs the Diffie-Hellman (DH) key-exchange. This uses a weak key (by default: `1024 bit`) that gets lower scores.
>
> You should always use the Elliptic Curve Diffie Hellman Ephemeral (`ECDHE`). Due to increasing concern about pervasive surveillance, key exchanges that provide Forward Secrecy are recommended, see for example RFC 7525.
>
> For greater compatibility but still for security in key exchange, you should prefer the latter E (ephemeral) over the former E (EC). There is recommended configuration: `ECDHE` > `DHE` (with min. `2048 bit` size) > `ECDH` . With this if the initial handshake fails, another handshake will be initiated using `DHE` .
>
> `DHE` is slower than `ECDHE` . If you are concerned about performance, prioritize `ECDHE-ECDSA` over `DHE` . OWASP estimates that the TLS handshake with `DHE` hinders the CPU by a factor of 2.4 compared to `ECDHE` .
>
> Diffie-Hellman requires some set-up parameters to begin with. Parameters from `ssl_dhparam` (which are generated with `openssl dhparam ...`) define how OpenSSL performs the Diffie-Hellman (DH) key-exchange. They include a field prime `p` and a generator `g` . The purpose of the availability to customize these parameter is to allow everyone to use own parameters for this. This can be used to prevent being affected from the Logjam attack.

> Modern clients prefer `ECDHE` instead other variants and if your Nginx accepts this preference then the handshake will not use the DH param at all since it will not do a `DHE` key exchange but an `ECDHE` key exchange. Thus, if no plain `DH/DHE` ciphers are configured at your server but only Eliptic curve DH (e.g. `ECDHE`) then you don't need to set your own `ssl_dhparam` directive. Enabling `DHE` requires us to take care of our DH primes (a.k.a. `dhparams`) and to trust in `DHE` .
>
> Elliptic curve Diffie-Hellman is a modified Diffie-Hellman exchange which uses Elliptic curve cryptography instead of the traditional RSA-style large primes. So while I'm not sure what parameters it may need (if any), I don't think it needs the kind you're generating (`ECDH` is based on curves, not primes, so I don't think the traditional DH params will do you any good).
>
> Cipher suites using `DHE` key exchange in OpenSSL require `tmp_DH` parameters, which the `ssl_dhparam` directive provides. The same is true for `DH_anon` key exchange, but in practice nobody uses those. The OpenSSL wiki page for Diffie Hellman Parameters it says: *To use perfect forward secrecy cipher suites, you must set up Diffie-Hellman parameters (on the server side).* Look also at SSL_CTX_set_tmp_dh_callback.
>
> If you use `ECDH/ECDHE` key exchange please see Use more secure ECDH Curve rule.
>
> Default key size in OpenSSL is `1024 bits` - it's vulnerable and breakable. For the best security configuration use your own DH Group (min. `2048 bit`) or use known safe ones pre-defined DH groups (it's recommended) from the Mozilla.
>
> The `2048 bit` is generally expected to be safe and is already very far into the "cannot break it zone". However years ago people expected 1024 bit to be safe so if you are after long term resistance You would go up to `4096 bit` (for both RSA keys and DH parameters). It's also important if you want to get 100% on Key Exchange of the SSL Labs test.
>
> You should remember that the `4096 bit` modulus will make DH computations slower and won't actually improve security.

There is good explanation about DH parameters recommended size:

> *Current recommendations from various bodies (including NIST) call for a `2048-bit` modulus for DH. Known DH-breaking algorithms would have a cost so ludicrously high that they could not be run to completion with known Earth-based technology. See this site for pointers on that subject.*
>
> *You don't want to overdo the size because the computational usage cost rises relatively sharply with prime size (somewhere between quadratic and cubic, depending on some implementation details) but a `2048-bit` DH ought to be fine (a basic low-end PC can do several hundreds of `2048-bit` DH per second).*

Look also at this answer by Matt Palmer:

> *Indeed, characterising `2048 bit` DH parameters as "weak as hell" is quite misleading. There are no known feasible cryptographic attacks against arbitrary strong 2048 bit DH groups. To protect against future disclosure of a session key due to breaking DH, sure, you want your DH parameters to be as long as is practical, but since `1024 bit` DH is only just getting feasible, `2048 bits` should be OK for most purposes for a while yet.*

My recommendation:

> If you use only TLS 1.3 - `ssl_dhparam` is not required (not used). Also, if you use `ECDHE/ECDH` - `ssl_dhparam` is not required (not used). If you use `DHE/DH` - `ssl_dhparam` with DH parameters is required (min. `2048 bit`). By default no parameters are set, and therefore `DHE` ciphers will not be used.

Example

```
## To generate a DH parameters:
openssl dhparam -out /etc/nginx/ssl/dhparam_4096.pem 4096

## To produce "DSA-like" DH parameters:
openssl dhparam -dsaparam -out /etc/nginx/ssl/dhparam_4096.pem 4096

## Nginx configuration only for DH/DHE:
ssl_dhparam /etc/nginx/ssl/dhparams_4096.pem;
```

:arrow_right: ssllabs score: **100%**

```
## To generate a DH parameters:
openssl dhparam -out /etc/nginx/ssl/dhparam_2048.pem 2048

## To produce "DSA-like" DH parameters:
openssl dhparam -dsaparam -out /etc/nginx/ssl/dhparam_2048.pem 2048

## Nginx configuration only for DH/DHE:
ssl_dhparam /etc/nginx/ssl/dhparam_2048.pem;
```

:arrow_right: ssllabs score: **90%**

External resources

- Weak Diffie-Hellman and the Logjam Attack

- Guide to Deploying Diffie-Hellman for TLS
- Pre-defined DHE groups
- Instructs OpenSSL to produce “DSA-like” DH parameters
- OpenSSL generate different types of self signed certificate
- Public Diffie-Hellman Parameter Service/Tool
- Vincent Bernat’s SSL/TLS & Perfect Forward Secrecy
- RSA and ECDSA performance
- SSL/TLS: How to choose your cipher suite
- Diffie-Hellman and its TLS/SSL usage

:beginner: Prevent Replay Attacks on Zero Round-Trip Time

Rationale

> This rules is only important for TLS 1.3. By default enabling TLS 1.3 will not enable 0-RTT support. After all, you should be fully aware of all the potential exposure factors and related risks with the use of this option.
>
> 0-RTT Handshakes is part of the replacement of TLS Session Resumption and was inspired by the QUIC Protocol.
>
> 0-RTT creates a significant security risk. With 0-RTT, a threat actor can intercept an encrypted client message and resend it to the server, tricking the server into improperly extending trust to the threat actor and thus potentially granting the threat actor access to sensitive data.
>
> On the other hand, including 0-RTT (Zero Round Trip Time Resumption) results in a significant increase in efficiency and connection times. TLS 1.3 has a faster handshake that completes in 1-RTT. Additionally, it has a particular session resumption mode where, under certain conditions, it is possible to send data to the server on the first flight (0-RTT).
>
> For example, Cloudflare only supports 0-RTT for GET requests with no query parameters in an attempt to limit the attack surface. Moreover, in order to improve identify connection resumption attempts, they relay this information to the origin by adding an extra header to 0-RTT requests. This header uniquely identifies the request, so if one gets repeated, the origin will know it’s a replay attack (the application needs to track values received from that and reject duplicates on non-idempotent endpoints).
>
> To protect against such attacks at the application layer, the `$ssl_early_data` variable should be used. You’ll also need to ensure that the `Early-Data` header is passed to your application. `$ssl_early_data` returns 1 if TLS 1.3 early data is used and the handshake is not complete.

> However, as part of the upgrade, you should disable 0-RTT until you can audit your application for this class of vulnerability.
>
> In order to send early-data, client and server must support PSK exchange mode (session cookies).
>
> In addition, I would like to recommend this great discussion about TLS 1.3 and 0-RTT.

If you are unsure to enable 0-RTT, look what Cloudflare say about it:

> *Generally speaking, 0-RTT is safe for most web sites and applications. If your web application does strange things and you're concerned about its replay safety, consider not using 0-RTT until you can be certain that there are no negative effects. […] TLS 1.3 is a big step forward for web performance and security. By combining TLS 1.3 with 0-RTT, the performance gains are even more dramatic.*

Example

Test 0-RTT with OpenSSL:

```
## 1)
_host="example.com"

cat > req.in << __EOF__
HEAD / HTTP/1.1
Host: $_host
Connection: close
__EOF__
## or:
## echo -e "GET / HTTP/1.1\r\nHost: $_host\r\nConnection: close\r\n\r\n" > req.in

openssl s_client -connect ${_host}:443 -tls1_3 -sess_out session.pem -ign_eof < req.in
openssl s_client -connect ${_host}:443 -tls1_3 -sess_in session.pem -early_data req.in

## 2)
python -m sslyze --early_data "$_host"
```

Enable 0-RTT with `$ssl_early_data` variable:

```
server {
```

```
2.
3.    ...
4.
5.    ssl_protocols   TLSv1.2 TLSv1.3;
6.    ## To enable 0-RTT (TLS 1.3):
7.    ssl_early_data  on;
8.
9.    location / {
10.
11.     proxy_pass       http://backend_x20;
12.     ## It protect against such attacks at the application layer:
13.     proxy_set_header Early-Data $ssl_early_data;
14.
15.   }
16.
17.   ...
18.
19. }
```

External resources

- Security Review of TLS1.3 0-RTT
- Introducing Zero Round Trip Time Resumption (0-RTT)
- What Application Developers Need To Know About TLS Early Data (0RTT)
- Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates
- 0-RTT and Anti-Replay
- Using Early Data in HTTP (2017)
- Using Early Data in HTTP (2018)
- 0-RTT Handshakes

:beginner: Defend against the BEAST attack

Rationale

> Generally the BEAST attack relies on a weakness in the way `CBC` mode is used in SSL/TLS.
>
> More specifically, to successfully perform the BEAST attack, there are some conditions which needs to be met:
>
> - vulnerable version of SSL must be used using a block cipher (`CBC` in particular)

> - JavaScript or a Java applet injection - should be in the same origin of the web site
> - data sniffing of the network connection must be possible
>
> To prevent possible use BEAST attacks you should enable server-side protection, which causes the server ciphers should be preferred over the client ciphers, and completely excluded TLS 1.0 from your protocol stack.

Example

```
1.  ssl_prefer_server_ciphers on;
```

External resources

- An Illustrated Guide to the BEAST Attack
- Is BEAST still a threat?
- Beat the BEAST with TLS 1.1/1.2 and More
- Use only strong ciphers (from this handbook)

:beginner: Mitigation of CRIME/BREACH attacks

Rationale

> Disable HTTP compression or compress only zero sensitive content.
>
> You should probably never use TLS compression. Some user agents (at least Chrome) will disable it anyways. Disabling SSL/TLS compression stops the attack very effectively. A deployment of HTTP/2 over TLS 1.2 must disable TLS compression (please see RFC 7540: 9.2. Use of TLS Features).
>
> CRIME exploits SSL/TLS compression which is disabled since nginx 1.3.2. BREACH exploits HTTP compression
>
> Some attacks are possible (e.g. the real BREACH attack is a complicated) because of gzip (HTTP compression not TLS compression) being enabled on SSL requests. In most cases, the best action is to simply disable gzip for SSL.
>
> Compression is not the only requirement for the attack to be done so using it does not mean that the attack will succeed. Generally you should consider whether having an accidental performance drop on HTTPS sites is better than HTTPS sites being accidentally vulnerable.
>
> You shouldn’t use HTTP compression on private responses when using TLS.
>
> I would gonna to prioritise security over performance but compression can be (I think) okay to HTTP compress publicly available static content like css or js and

> HTML content with zero sensitive info (like an “About Us” page).
>
> Remember: by default, Nginx doesn’t compress image files using its per-request gzip module.
>
> Gzip static module is better, for 2 reasons:
>
> - you don’t have to gzip for each request
> - you can use a higher gzip level
>
> You should put the `gzip_static on;` inside the blocks that configure static files, but if you’re only running one site, it’s safe to just put it in the http block.

Example

```
## Disable dynamic HTTP compression:
gzip off;

## Enable dynamic HTTP compression for specific location context:
location / {

  gzip on;

  ...

}

## Enable static gzip compression:
location ^~ /assets/ {

  gzip_static on;

  ...

}
```

External resources

- Is HTTP compression safe?
- HTTP compression continues to put encrypted communications at risk
- SSL/TLS attacks: Part 2 – CRIME Attack
- Defending against the BREACH Attack
- To avoid BREACH, can we use gzip on non-token responses?

- Don’t Worry About BREACH
- Module ngx_http_gzip_static_module
- Offline Compression with Nginx

:beginner: HTTP Strict Transport Security

Rationale

> Generally HSTS is a way for websites to tell browsers that the connection should only ever be encrypted. This prevents MITM attacks, downgrade attacks, sending plain text cookies and session ids.
>
> The header indicates for how long a browser should unconditionally refuse to take part in unsecured HTTP connection for a specific domain.
>
> When a browser knows that a domain has enabled HSTS, it does two things:
>
> - always uses an `https://` connection, even when clicking on an `http://` link or after typing a domain into the location bar without specifying a protocol
> - removes the ability for users to click through warnings about invalid certificates
>
> I recommend to set the `max-age` to a big value like `31536000` (12 months) or `63072000` (24 months).
>
> There are a few simple best practices for HSTS (from The Importance of a Proper HTTP Strict Transport Security Implementation on Your Web Server):
>
> - The strongest protection is to ensure that all requested resources use only TLS with a well-formed HSTS header. Qualys recommends providing an HSTS header on all HTTPS resources in the target domain
> - It is advisable to assign the max-age directive’s value to be greater than `10368000` seconds (120 days) and ideally to `31536000` (one year). Websites should aim to ramp up the max-age value to ensure heightened security for a long duration for the current domain and/or subdomains
> - RFC 6797, section 14.4 advocates that a web application must aim to add the `includeSubDomain` directive in the policy definition whenever possible. The directive’s presence ensures the HSTS policy is applied to the domain of the issuing host and all of its subdomains, e.g. `example.com` and `www.example.com`
> - The application should never send an HSTS header over a plaintext HTTP header, as doing so makes the connection vulnerable to SSL stripping attacks
> - It is not recommended to provide an HSTS policy via the http-equiv attribute

> of a meta tag. According to HSTS RFC 6797, user agents don't heed `http-equiv="Strict-Transport-Security"` attribute on `<meta>` elements on the received content`
>
> To meet the HSTS preload list standard a root domain needs to return a `strict-transport-security` header that includes both the `includeSubDomains` and `preload` directives and has a minimum `max-age` of one year. Your site must also serve a valid SSL certificate on the root domain and all subdomains, as well as redirect all HTTP requests to HTTPS on the same host.
>
> You had better be pretty sure that your website is indeed all HTTPS before you turn this on because HSTS adds complexity to your rollback strategy. Google recommend enabling HSTS this way:
>
> 1. Roll out your HTTPS pages without HSTS first
> 2. Start sending HSTS headers with a short `max-age` . Monitor your traffic both from users and other clients, and also dependents' performance, such as ads
> 3. Slowly increase the HSTS `max-age`
> 4. If HSTS doesn't affect your users and search engines negatively, you can, if you wish, ask your site to be added to the HSTS preload list used by most major browsers

Example

```
    add_header Strict-Transport-Security "max-age=63072000; includeSubdomains"
1.  always;
```

:arrow_right: ssllabs score: **A+**

External resources

- Strict-Transport-Security
- Security HTTP Headers - Strict-Transport-Security
- HTTP Strict Transport Security
- HTTP Strict Transport Security Cheat Sheet
- HSTS Cheat Sheet
- HSTS Preload and Subdomains
- HTTP Strict Transport Security (HSTS) and Nginx
- Is HSTS as a proper substitute for HTTP-to-HTTPS redirects?
- How to configure HSTS on www and other subdomains
- HSTS: Is includeSubDomains on main domain sufficient?
- The HSTS preload list eligibility
- Check HSTS preload status and eligibility
- HSTS Deployment Recommendations

- How does HSTS handle mixed content?

:beginner: Reduce XSS risks (Content-Security-Policy)

Rationale

> CSP reduce the risk and impact of XSS attacks in modern browsers.
>
> Whitelisting known-good resource origins, refusing to execute potentially dangerous inline scripts, and banning the use of eval are all effective mechanisms for mitigating cross-site scripting attacks.
>
> The inclusion of CSP policies significantly impedes successful XSS attacks, UI Redressing (Clickjacking), malicious use of frames or CSS injections.
>
> CSP is a good defence-in-depth measure to make exploitation of an accidental lapse in that less likely.
>
> The default policy that starts building a header is: block everything. By modifying the CSP value, the programmer loosens restrictions for specific groups of resources (e.g. separately for scripts, images, etc.).
>
> Before enable this header you should discuss with developers about it. They probably going to have to update your application to remove any inline script and style, and make some additional modifications there.
>
> Strict policies will significantly increase security, and higher code quality will reduce the overall number of errors. CSP can never replace secure code - new restrictions help reduce the effects of attacks (such as XSS), but they are not mechanisms to prevent them!
>
> You should always validate CSP before implement: CSP Evaluator and Content Security Policy (CSP) Validator.
>
> For generate a policy: https://report-uri.com/home/generate. Remember, however, that these types of tools may become outdated or have errors.

Example

```
## This policy allows images, scripts, AJAX, and CSS from the same origin, and
1. does not allow any other resources to load:
add_header Content-Security-Policy "default-src 'none'; script-src 'self';
2. connect-src 'self'; img-src 'self'; style-src 'self';" always;
```

External resources

- Content Security Policy (CSP) Quick Reference Guide
- Content Security Policy Cheat Sheet – OWASP
- Content Security Policy – OWASP
- Content Security Policy - An Introduction - Scott Helme
- CSP Cheat Sheet - Scott Helme
- Security HTTP Headers - Content-Security-Policy
- CSP Evaluator
- Content Security Policy (CSP) Validator
- Can I Use CSP
- CSP Is Dead, Long Live CSP!

:beginner: Control the behaviour of the Referer header (Referrer-Policy)

Rationale

> Determine what information is sent along with the requests.

Example

```
1.  add_header Referrer-Policy "no-referrer";
```

External resources

- A new security header: Referrer Policy
- Security HTTP Headers - Referrer-Policy

:beginner: Provide clickjacking protection (X-Frame-Options)

Rationale

> Helps to protect your visitors against clickjacking attacks. It is recommended that you use the `x-frame-options` header on pages which should not be allowed to render a page in a frame.

Example

```
1.  add_header X-Frame-Options "SAMEORIGIN" always;
```

External resources

- HTTP Header Field X-Frame-Options
- Clickjacking Defense Cheat Sheet
- Security HTTP Headers - X-Frame-Options
- X-Frame-Options - Scott Helme

:beginner: Prevent some categories of XSS attacks (X-XSS-Protection)

Rationale

> Enable the cross-site scripting (XSS) filter built into modern web browsers.

Example

```
1.  add_header X-XSS-Protection "1; mode=block" always;
```

External resources

- XSS (Cross Site Scripting) Prevention Cheat Sheet_Prevention_Cheat_Sheet>)
- DOM based XSS Prevention Cheat Sheet
- X-XSS-Protection HTTP Header
- Security HTTP Headers - X-XSS-Protection

:beginner: Prevent Sniff Mimetype middleware (X-Content-Type-Options)

Rationale

> It prevents the browser from doing MIME-type sniffing (prevents “mime” based attacks).

Example

```
1.  add_header X-Content-Type-Options "nosniff" always;
```

External resources

- X-Content-Type-Options HTTP Header

- Security HTTP Headers - X-Content-Type-Options
- X-Content-Type-Options - Scott Helme

:beginner: Deny the use of browser features (Feature-Policy)

Rationale

> This header protects your site from third parties using APIs that have security and privacy implications, and also from your own team adding outdated APIs or poorly optimised images.

Example

```
    add_header Feature-Policy "geolocation 'none'; midi 'none'; notifications
    'none'; push 'none'; sync-xhr 'none'; microphone 'none'; camera 'none';
    magnetometer 'none'; gyroscope 'none'; speaker 'none'; vibrate 'none';
1.  fullscreen 'none'; payment 'none'; usb 'none';";
```

External resources

- Feature Policy Explainer
- Policy Controlled Features
- Security HTTP Headers - Feature-Policy

:beginner: Reject unsafe HTTP methods

Rationale

> Set of methods support by a resource. An ordinary web server supports the `HEAD`, `GET` and `POST` methods to retrieve static and dynamic content. Other (e.g. `OPTIONS`, `TRACE`) methods should not be supported on public web servers, as they increase the attack surface.

Example

```
1.  add_header Allow "GET, POST, HEAD" always;
2.
3.  if ($request_method !~ ^(GET|POST|HEAD)$) {
4.
5.    return 405;
6.
```

```
}
```

External resources

- Vulnerability name: Unsafe HTTP methods

:beginner: Prevent caching of sensitive data

Rationale

> This policy should be implemented by the application architect, however, I know from experience that this does not always happen.
>
> Don’ to cache or persist sensitive data. As browsers have different default behaviour for caching HTTPS content, pages containing sensitive information should include a `Cache-Control` header to ensure that the contents are not cached.
>
> One option is to add anticaching headers to relevant HTTP/1.1 and HTTP/2 responses, e.g. `Cache-Control: no-cache, no-store` and `Expires: 0` .
>
> To cover various browser implementations the full set of headers to prevent content being cached should be:
>
> `Cache-Control: no-cache, no-store, private, must-revalidate, max-age=0, no-transform` > `Pragma: no-cache` > `Expires: 0`

Example

```
location /api {

  expires 0;
  add_header Cache-Control "no-cache, no-store";

}
```

External resources

- RFC 2616 - Hypertext Transfer Protocol (HTTP/1.1): Standards Track
- RFC 7234 - Hypertext Transfer Protocol (HTTP/1.1): Caching
- HTTP Cache Headers - A Complete Guide
- Caching best practices & max-age gotchas
- Increasing Application Performance with HTTP Cache Headers
- HTTP Caching

:beginner: Control Buffer Overflow attacks

Rationale

> Buffer overflow attacks are made possible by writing data to a buffer and exceeding that buffers' boundary and overwriting memory fragments of a process. To prevent this in Nginx we can set buffer size limitations for all clients.

Example

```
client_body_buffer_size 100k;
client_header_buffer_size 1k;
client_max_body_size 100k;
large_client_header_buffers 2 1k;
```

External resources

- SCG WS nginx

:beginner: Mitigating Slow HTTP DoS attacks (Closing Slow Connections)

Rationale

> Close connections that are writing data too infrequently, which can represent an attempt to keep connections open as long as possible.
>
> You can close connections that are writing data too infrequently, which can represent an attempt to keep connections open as long as possible (thus reducing the server's ability to accept new connections).

Example

```
client_body_timeout 10s;
client_header_timeout 10s;
keepalive_timeout 5s 5s;
send_timeout 10s;
```

External resources

- Mitigating DDoS Attacks with Nginx and Nginx Plus
- SCG WS nginx

- How to Protect Against Slow HTTP Attacks
- Effectively Using and Detecting The Slowloris HTTP DoS Tool

反向代理

使用与后端协议兼容的 pass 指令

> All `proxy_*` directives are related to the backends that use the specific backend protocol.
>
> You should use `proxy_pass` only for HTTP servers working on the backend layer (set also the `http://` protocol before referencing the HTTP backend) and other `*_pass` directives only for non-HTTP backend servers (like a uWSGI or FastCGI).
>
> Directives such as `uwsgi_pass`, `fastcgi_pass`, or `scgi_pass` are designed specifically for non-HTTP apps and you should use them instead of the `proxy_pass` (non-HTTP talking).
>
> For example: `uwsgi_pass` uses an uwsgi protocol. `proxy_pass` uses normal HTTP to talking with uWSGI server. uWSGI docs claims that uwsgi protocol is better, faster and can benefit from all of uWSGI special features. You can send to uWSGI information what type of data you are sending and what uWSGI plugin should be invoked to generate response. With http (`proxy_pass`) you won't get that.

示例：

错误配置

```
server {

  location /app/ {

    ## For this, you should use uwsgi_pass directive.
    proxy_pass      192.168.154.102:4000;         ## backend layer: uWSGI
Python app.

  }

  ...

}
```

正确配置

```
server {

```

```
  location /app/ {

    proxy_pass      http://192.168.154.102:80;    ## backend layer: OpenResty as a front for app.

  }

  location /app/v3 {

    uwsgi_pass      192.168.154.102:8080;         ## backend layer: uWSGI Python app.

  }

  location /app/v4 {

    fastcgi_pass    192.168.154.102:8081;         ## backend layer: php-fpm app.

  }
  ...

}
```

小心 `proxy_pass` 指令中的斜杠

> 注意尾随斜杠，因为 Nginx 会逐字替换部分，并且您可能会得到一些奇怪的 URL。
>
> 如果 proxy_pass 不带 URI 使用（即 server:port 之后没有路径），Nginx 会将原始请求中的 URI 与所有双斜杠 `../` 完全一样。
>
> `proxy_pass` 中的 URI 就像别名指令一样，意味着 Nginx 将用 `proxy_pass` 指令中的 URI 替换与位置前缀匹配的部分（我故意将其与位置前缀相同），因此 URI 将与请求的相同，但被规范化（没有小写斜杠和其他所有内容） 员工）。

示例：

```
location = /a {

  proxy_pass http://127.0.0.1:8080/a;

  ...

```

```
}

location ^~ /a/ {

  proxy_pass http://127.0.0.1:8080/a/;

  ...

}
```

仅使用 `$host` 变量设置和传递 Host 头

> 几乎应该始终将 `$host` 用作传入的主机变量，因为无论用户代理如何行为，它都是保证具有某种意义的唯一变量，除非您特别需要其他变量之一的语义。
>
> 变量 `$host` 是请求行或http头中的主机名。 变量 `$server_name` 是我们当前所在的服务器块的名称。
>
> 区别:
>
> - `$host` 包含“按此优先顺序：请求行中的主机名，或“主机”请求标头字段中的主机名，或与请求匹配的服务器名”
> - 如果请求中包含HTTP主机标头字段，则 `$http_host` 包含该内容（始终等于HTTP_HOST请求标头）
> - `$server_name` contains the `server_name` of the virtual host which processed the request, as it was defined in the Nginx configuration. If a server contains multiple server names, only the first one will be present in this variable
>
> `http_host` , moreover, is better than `$host:$server_port` because it uses the port as present in the URL, unlike `$server_port` which uses the port that Nginx listens on.

示例:

```
proxy_set_header    Host    $host;
```

正确设置 `X-Forwarded-For` 头的值

Rationale

> In the light of the latest httpoxy vulnerabilities, there is really a need for a full example, how to use `HTTP_X_FORWARDED_FOR` properly. In short, the load balancer sets the ‘most recent’ part of the header. In my opinion, for security reasons, the proxy servers must be specified by the administrator manually.

> `X-Forwarded-For` is the custom HTTP header that carries along the original IP address of a client so the app at the other end knows what it is. Otherwise it would only see the proxy IP address, and that makes some apps angry.
>
> The `X-Forwarded-For` depends on the proxy server, which should actually pass the IP address of the client connecting to it. Where a connection passes through a chain of proxy servers, `X-Forwarded-For` can give a comma-separated list of IP addresses with the first being the furthest downstream (that is, the user). Because of this, servers behind proxy servers need to know which of them are trustworthy.
>
> The proxy used can set this header to anything it wants to, and therefore you can't trust its value. Most proxies do set the correct value though. This header is mostly used by caching proxies, and in those cases you're in control of the proxy and can thus verify that is gives you the correct information. In all other cases its value should be considered untrustworthy.
>
> Some systems also use `X-Forwarded-For` to enforce access control. A good number of applications rely on knowing the actual IP address of a client to help prevent fraud and enable access.
>
> Value of the `X-Forwarded-For` header field can be set at the client's side - this can also be termed as `X-Forwarded-For` spoofing. However, when the web request is made via a proxy server, the proxy server modifies the `X-Forwarded-For` field by appending the IP address of the client (user). This will result in 2 comma separated IP addresses in the `X-Forwarded-For` field.
>
> A reverse proxy is not source IP address transparent. This is a pain when you need the client source IP address to be correct in the logs of the backend servers. I think the best solution of this problem is configure the load balancer to add/modify an `X-Forwarded-For` header with the source IP of the client and forward it to the backend in the correct form.
>
> Unfortunately, on the proxy side we are not able to solve this problem (all solutions can be spoofable), it is important that this header is correctly interpreted by application servers. Doing so ensures that the apps or downstream services have accurate information on which to make their decisions, including those regarding access and authorization.

There is also an interesing idea what to do in this situation:

> *To prevent this we must distrust that header by default and follow the IP address breadcrumbs backwards from our server. First we need to make sure the `REMOTE_ADDR` is someone we trust to have appended a proper value to the end of `X-Forwarded-For`. If so then we need to make sure we trust the `X-Forwarded-For` IP to have appended the proper IP before it, so on and so forth. Until, finally we get to an IP we don't trust and at that point we have to assume that's the IP of our user.* - it comes from Proxies & IP Spoofing by Xiao Yu.

Example

```
## The whole purpose that it exists is to do the appending behavior:
proxy_set_header    X-Forwarded-For    $proxy_add_x_forwarded_for;
## Above is equivalent for this:
proxy_set_header    X-Forwarded-For    $http_x_forwarded_for,$remote_addr;
## The following is also equivalent for above but in this example we use
http_realip_module:
proxy_set_header    X-Forwarded-For    "$http_x_forwarded_for,
$realip_remote_addr";
```

External resources

- Prevent X-Forwarded-For Spoofing or Manipulation
- Bypass IP blocks with the X-Forwarded-For header
- Forwarded header

不要在反向代理后面使用带有 `$scheme` 的 `X-Forwarded-Proto`

> 反向代理可以设置 `X-Forwarded-Proto` ，以告知应用程序它是HTTPS还是HTTP甚至是无效名称。 schema 变量仅在需要的时候才会被评估（仅用于当前请求）。
>
> 如果设置了 $schema 变量且沿途遇上多个代理，则会导致变形。例如：如果客户端转到 https://example.com，则代理将方案值存储为HTTPS。 如果代理与下一级代理之间的通信是通过HTTP进行的，则后端会将方案视为HTTP。

示例：

```
## 1) client <-> proxy <-> backend
proxy_set_header    X-Forwarded-Proto  $scheme;

## 2) client <-> proxy <-> proxy <-> backend
## proxy_set_header  X-Forwarded-Proto  https;
proxy_set_header    X-Forwarded-Proto  $proxy_x_forwarded_proto;
```

始终将 Host, X-Real-IP 和 X-Forwarded 标头传递给后端

Rationale

> When using Nginx as a reverse proxy you may want to pass through some information of the remote client to your backend web server. I think it’s good practices because gives you more control of forwarded headers.

> It’s very important for servers behind proxy because it allow to interpret the client correctly. Proxies are the “eyes” of such servers, they should not allow a curved perception of reality. If not all requests are passed through a proxy, as a result, requests received directly from clients may contain e.g. inaccurate IP addresses in headers.
>
> `X-Forwarded` headers are also important for statistics or filtering. Other example could be access control rules on your app, because without these headers filtering mechanism may not working properly.
>
> If you use a front-end service like Apache or whatever else as the front-end to your APIs, you will need these headers to understand what IP or hostname was used to connect to the API.
>
> Forwarding these headers is also important if you use the https protocol (it has become a standard nowadays).
>
> However, I would not rely on either the presence of all `X-Forwarded` headers, or the validity of their data.

Example

```
location / {

  proxy_pass          http://bk_upstream_01;

  ## The following headers also should pass to the backend:
  ##   - Host - host name from the request line, or host name from the Host request header field, or the server name matching a request
  ## proxy_set_header  Host              $host:$server_port;
  ## proxy_set_header  Host              $http_host;
  proxy_set_header    Host              $host;

  ##   - X-Real-IP - forwards the real visitor remote IP address to the proxied server
  proxy_set_header    X-Real-IP         $remote_addr;

  ## X-Forwarded headers stack:
  ##   - X-Forwarded-For - mark origin IP of client connecting to server through proxy
  ## proxy_set_header  X-Forwarded-For    $remote_addr;
  ## proxy_set_header  X-Forwarded-For    $http_x_forwarded_for,$remote_addr;
  ## proxy_set_header  X-Forwarded-For    "$http_x_forwarded_for, $realip_remote_addr";
  proxy_set_header    X-Forwarded-For    $proxy_add_x_forwarded_for;
```

```
20.
      ##   - X-Forwarded-Host - mark origin host of client connecting to server
21. through proxy
22.   ## proxy_set_header  X-Forwarded-Host   $host:443;
23.   proxy_set_header    X-Forwarded-Host   $host:$server_port;
24.
25.   ##   - X-Forwarded-Server - the hostname of the proxy server
26.   proxy_set_header    X-Forwarded-Server $host;
27.
28.   ##   - X-Forwarded-Port - defines the original port requested by the client
29.   ## proxy_set_header  X-Forwarded-Port   443;
30.   proxy_set_header    X-Forwarded-Port   $server_port;
31.
      ##   - X-Forwarded-Proto - mark protocol of client connecting to server
32. through proxy
33.   ## proxy_set_header  X-Forwarded-Proto  https;
34.   ## proxy_set_header  X-Forwarded-Proto  $proxy_x_forwarded_proto;
35.   proxy_set_header    X-Forwarded-Proto  $scheme;
36.
37. }
```

prefix 使用不带 `X-` 前缀的自定义头

Rationale

> Internet Engineering Task Force released a new RFC (RFC-6648), recommending deprecation of `X-` prefix.
>
> The `X-` in front of a header name customarily has denoted it as experimental/non-standard/vendor-specific. Once it’s a standard part of HTTP, it’ll lose the prefix.
>
> If it’s possible for new custom header to be standardized, use a non-used and meaningful header name.
>
> The use of custom headers with `X-` prefix is not forbidden but discouraged. In other words, you can keep using `X-` prefixed headers, but it’s not recommended and you may not document them as if they are public standard.

Example

Not recommended configuration:

```
1.  add_header X-Backend-Server $hostname;
```

Recommended configuration:

```
1.  add_header Backend-Server   $hostname;
```

External resources

- Use of the “X-“ Prefix in Application Protocols
- Custom HTTP headers : naming conventions

负载均衡

负载平衡是一种有用的机制，可将传入的流量分布在几个有能力的服务器之间。

健康检查

> 健康监控对于所有类型的负载平衡都非常重要，主要是为了业务连续性。 被动检查会按照客户端的请求监视通过 Nginx 的连接失败或超时。
>
> 默认情况下启用此功能，但是此处提到的参数允许您调整其行为。 默认值为： `max_fails = 1` 和 `fail_timeout = 10s` 。

示例：

```
upstream backend {

  server bk01_node:80 max_fails=3 fail_timeout=5s;
  server bk02_node:80 max_fails=3 fail_timeout=5s;

}
```

down 参数

> 有时我们需要关闭后端，例如 在维护时。 我认为良好的解决方案是使用 down 参数将服务器标记为永久不可用，即使停机时间很短也是如此。
>
> 如果您使用 IP 哈希负载平衡技术，那也很重要。 如果其中一台服务器需要临时删除，则应使用此参数进行标记，以保留客户端 IP 地址的当前哈希值。
>
> 注释对于真正永久禁用服务器或要出于历史目的而保留信息非常有用。
>
> Nginx 还提供了一个备份参数，将该服务器标记为备份服务器。 当主服务器不可用时，将传递请求。 仅当我确定后端将在维护时正常工作时，我才很少将此选项用于上述目的。

示例

```
upstream backend {

  server bk01_node:80 max_fails=3 fail_timeout=5s down;
  server bk02_node:80 max_fails=3 fail_timeout=5s;

}
```

安全

防盗链

```
location ~* \.(gif|jpg|png)$ {
    # 只允许 192.168.0.1 请求资源
    valid_referers none blocked 192.168.0.1;
    if ($invalid_referer) {
       rewrite ^/ http://$host/logo.png;
    }
}
```

参考资料

- nginx 这一篇就够了

Nginx 问题集

Nginx 出现大量 TIME_WAIT

检测TIME_WAIT状态的语句

```
$ netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
SYN_RECV 7
ESTABLISHED 756
FIN_WAIT1 21
SYN_SENT 3
TIME_WAIT 2000
```

状态解析:

- `CLOSED` - 无连接是活动的或正在进行
- `LISTEN` - 服务器在等待进入呼叫
- `SYN_RECV` - 一个连接请求已经到达，等待确认
- `SYN_SENT` - 应用已经开始，打开一个连接
- `ESTABLISHED` - 正常数据传输状态
- `FIN_WAIT1` - 应用说它已经完成
- `FIN_WAIT2` - 另一边已同意释放
- `ITMED_WAIT` - 等待所有分组死掉
- `CLOSING` - 两边同时尝试关闭
- `TIME_WAIT` - 另一边已初始化一个释放
- `LAST_ACK` - 等待所有分组死掉

解决方法

执行 `vim /etc/sysctl.conf` ，并添加下面字段

```
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 30
```

执行 `/sbin/sysctl -p` 让修改生效。

上传文件大小限制

问题现象

显示错误信息：**413 Request Entity Too Large**。

意思是请求的内容过大，浏览器不能正确显示。常见的情况是发送 `POST` 请求来上传大文件。

解决方法

- 可以在 `http` 模块中设置：`client_max_body_size 20m;`
- 可以在 `server` 模块中设置：`client_max_body_size 20m;`
- 可以在 `location` 模块中设置：`client_max_body_size 20m;`

三者区别是：

- 如果文大小限制设置在 `http` 模块中，则对所有 Nginx 收到的请求。
- 如果文大小限制设置在 `server` 模块中，则只对该 `server` 收到的请求生效。
- 如果文大小限制设置在 `location` 模块中，则只对匹配了 `location` 路由规则的请求生效。

请求时间限制

问题现象

请求时间较长，链接被重置页面刷新。常见的情况是：上传、下载大文件。

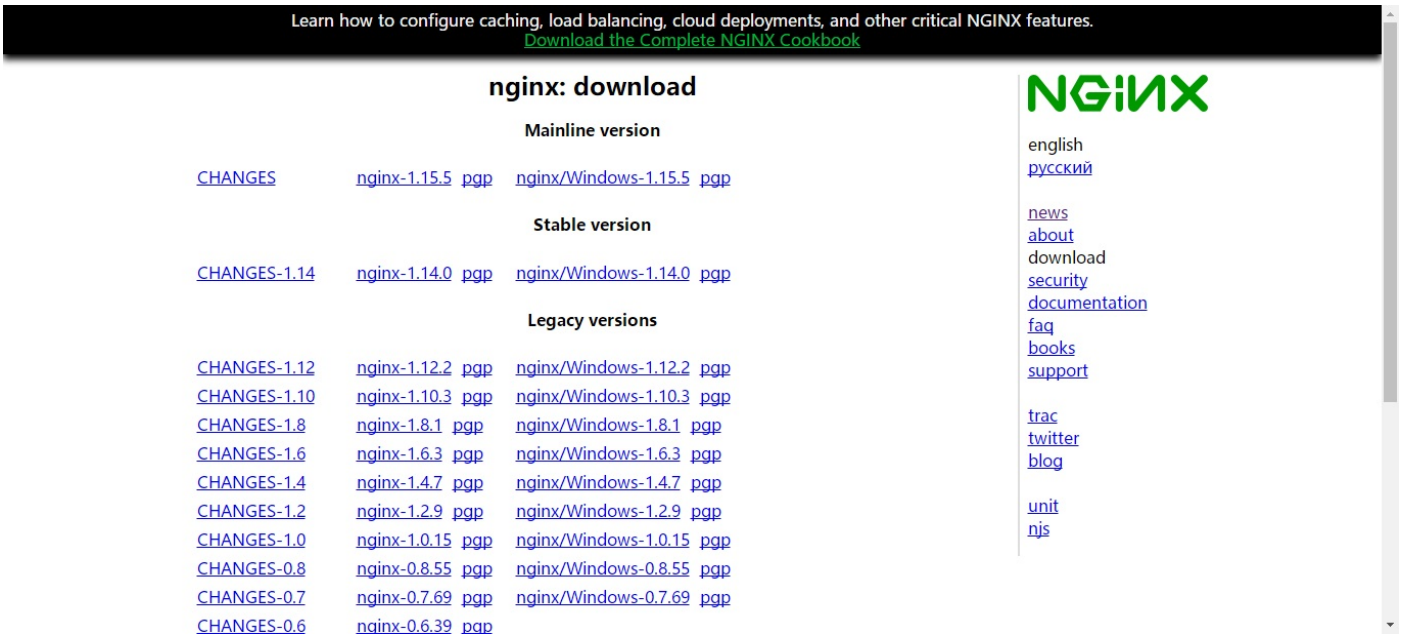
解决方法

修改超时时间

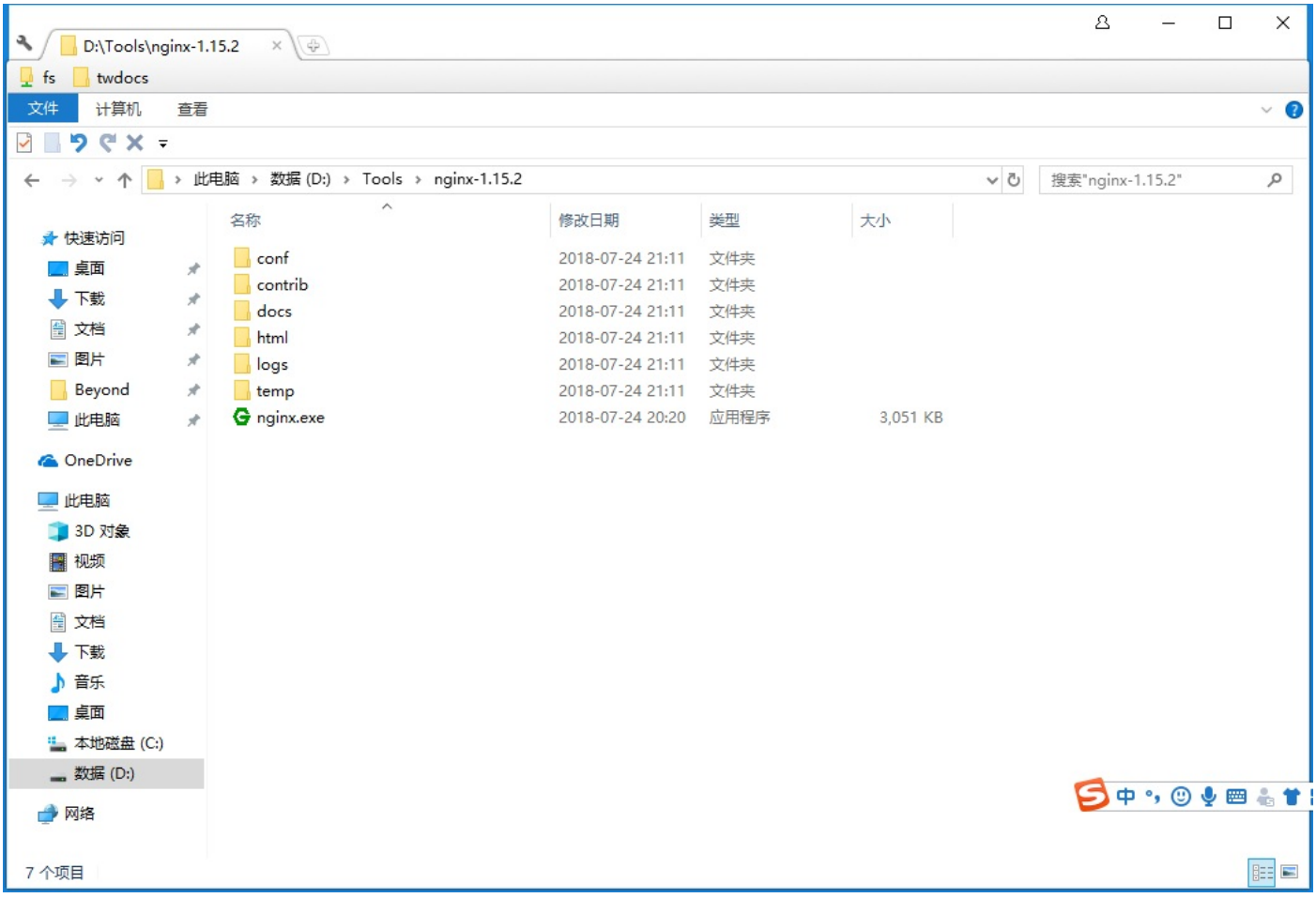
Nginx 安装

Windows 安装

（1）进入官方下载地址，选择合适版本（nginx/Windows-xxx）。



（2）解压到本地



（3）启动

下面以 C 盘根目录为例说明下：

```
1.  cd C:
2.  cd C:\nginx-0.8.54 start nginx
```

> 注：Nginx / Win32 是运行在一个控制台程序，而非 windows 服务方式的。服务器方式目前还是开发尝试中。

Linux 安装

rpm 包方式（推荐）

（1）进入下载页面，选择合适版本下载。

```
$ wget http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-
0.el7.ngx.noarch.rpm
```

（2）安装 nginx rpm 包

nginx rpm 包实际上安装的是 nginx 的 yum 源。

```
$ rpm -ivh nginx-*.rpm
```

（3）正式安装 rpm 包

```
$ yum install nginx
```

（4）关闭防火墙

```
$ firewall-cmd --zone=public --add-port=80/tcp --permanent
$ firewall-cmd --reload
```

源码编译方式

安装编译工具及库文件

Nginx 源码的编译依赖于 gcc 以及一些库文件，所以必须提前安装。

```
$ yum -y install make zlib zlib-devel gcc-c++ libtool  openssl openssl-devel
```

Nginx 依赖 pcre 库，安装步骤如下：

（1）下载解压到本地

进入pcre 官网下载页面，选择合适的版本下载。

我选择的是 8.35 版本：

```
    wget -O /opt/pcre/pcre-8.35.tar.gz
1.  http://downloads.sourceforge.net/project/pcre/pcre/8.35/pcre-8.35.tar.gz
2.  cd /opt/pcre
3.  tar zxvf pcre-8.35.tar.gz
```

（2）编译安装

执行以下命令：

```
1.  cd /opt/pcre/pcre-8.35
2.  ./configure
3.  make && make install
```

（3）检验是否安装成功

执行 `pcre-config --version` 命令。

安装 Nginx

安装步骤如下：

（1）下载解压到本地

进入官网下载地址：http://nginx.org/en/download.html ，选择合适的版本下载。

我选择的是 1.12.2 版本：http://downloads.sourceforge.net/project/pcre/pcre/8.35/pcre-8.35.tar.gz

```
    wget -O /opt/nginx/nginx-1.12.2.tar.gz http://nginx.org/download/nginx-
1.  1.12.2.tar.gz
2.  cd /opt/nginx
3.  tar zxvf nginx-1.12.2.tar.gz
```

（2）编译安装

执行以下命令：

```
1.  cd /opt/nginx/nginx-1.12.2
    ./configure --with-http_stub_status_module --with-http_ssl_module --with-
2.  pcre=/opt/pcre/pcre-8.35
```

（3）关闭防火墙

```
$ firewall-cmd --zone=public --add-port=80/tcp --permanent
$ firewall-cmd --reload
```

（4） 启动 Nginx

安装成功后，直接执行 `nginx` 命令即可启动 nginx。

启动后，访问站点：

Linux 开机自启动

Centos7 以上是用 Systemd 进行系统初始化的，Systemd 是 Linux 系统中最新的初始化系统（init），它主要的设计目标是克服 sysvinit 固有的缺点，提高系统的启动速度。Systemd 服务文件以 .service 结尾。

rpm 包方式

如果是通过 rpm 包安装的，会自动创建 nginx.service 文件。

直接用命令:

```
1.  $ systemctl enable nginx.service
```

设置开机启动即可。

源码编译方式

如果采用源码编译方式，需要手动创建 nginx.service 文件。

脚本

> CentOS7 环境安装脚本：软件运维配置脚本集合

安装说明

- 采用编译方式安装 Nginx，并将其注册为 systemd 服务
- 安装路径为：`/usr/local/nginx`
- 默认下载安装 `1.16.0` 版本

使用方法

- 默认安装 - 执行以下任意命令即可：

```
curl -o- https://gitee.com/turnon/linux-tutorial/raw/master/codes/linux/soft/ngin
install.sh | bash
wget -qO- https://gitee.com/turnon/linux-tutorial/raw/master/codes/linux/soft/ngi
install.sh | bash
```

- 自定义安装 - 下载脚本到本地，并按照以下格式执行：

```
sh nginx-install.sh [version]
```

参考资料

- http://www.dohooe.com/2016/03/03/352.html?utm_source=tuicool&utm_medium=referral

Nginx 示例教程

教程说明

环境要求

- Maven
- JDK8
- Nginx-1.14.0（内置）

javaapp

我写了一个嵌入式 Tomcat 的 Java 服务，代码在 javaapp 目录，基于 maven 管理。这个服务可以通过在启动时指定 `-Dtomcat.connector.port` 和 `-Dtomcat.context.path` 来分别指定服务启动时的端口号和 context。这样可以很方便的模拟多个服务器的场景。

例如：

```
java -Dtomcat.connector.port=9030 -Dtomcat.context.path=/app -cp
"JavaWebApp/WEB-INF/classes;JavaWebApp/WEB-INF/lib/*" io.github.dunwu.app.Main
```

- `io.github.dunwu.app.Main` 是这个 Java 服务的启动类。
- `JavaWebApp/WEB-INF/classes;JavaWebApp/WEB-INF/lib/*` 是 class 路径和 lib 路径，必须指定，否则无法识别启动类。

如上的配置参数，可以启动一个端口号为 9030，上下文为 `/app` 的服务。访问路径为：http://localhost:9030/app。

reactadmin

一个简单的 React 应用。用于演示静态站点场景。

reactapp

一个简单的 React 应用，生产环境时，会访问后台 API。用于演示前后端分离的应用场景。

nginx-1.14.0

nginx-1.14.0 是 Nginx 的 windows 环境的 1.14.0 官方版本。之所以把它完整的放入本项目中也是为了方便演示。

我添加了两个 bat 脚本，可以启动和关闭 nginx 服务。

- nginx-start.bat
- nginx-stop.bat

在 Nginx 默认配置文件 nginx.conf 中我通过配置 `include demos/*.conf;` 将 Nginx/demos/nginx-1.14.0/conf/demos 目录中所有 Nginx 配置示例都引入。

scripts

`scripts` 中包含了运行示例的启动脚本。目前只支持 windows 下运行，当然想基于此教程改造为在 Linux 下运行也不难，将 nginx-1.14.0 替换为 Linux 版本，bat 脚本修改为 shell 即可。

运行步骤：

1. 首先必须执行 build-javaapp.bat 构建 javaapp
2. 想运行哪个 demo，就执行对应的 demoxx-start.bat 脚本。

添加 hosts

因为示例中使用的不是公网域名，域名服务器不能识别。所以，要演示示例，还需要修改本地 hosts。

- windows 的 host 路径一般在：`C:\Windows\System32\drivers\etc\hosts`
- linxu 的 host 路径一般在：`/etc/hosts`

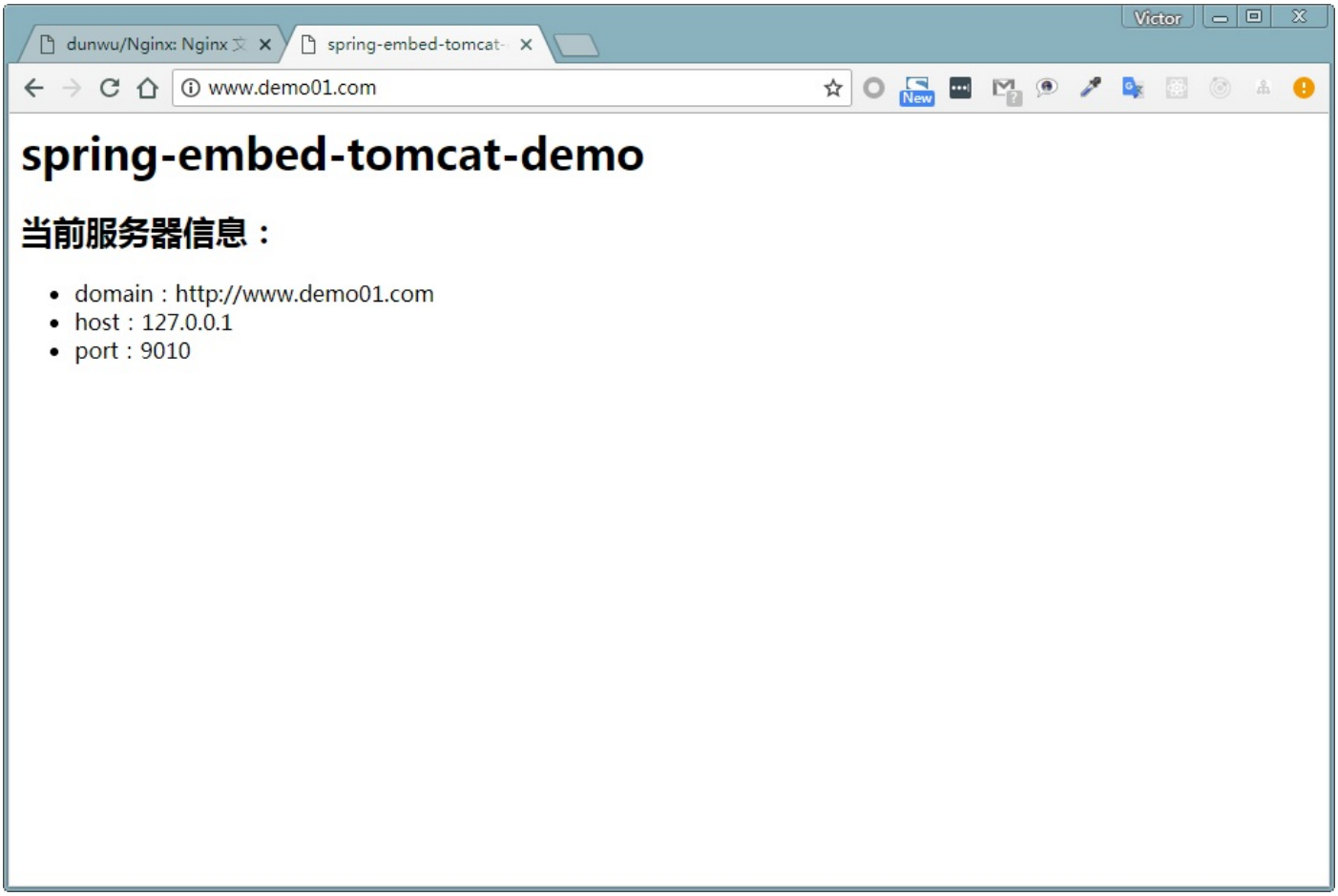
示例说明

Demo01 - 简单的反向代理示例

本示例启动一个 JavaApp，访问地址为：localhost:9010。 在 Nginx 中配置它的反向代理 host 为 www.demo01.com。Nginx 配置文件：demo01.conf

运行步骤：

1. 执行 demo01-start.bat 脚本。
2. 配置 hosts：`127.0.0.1 www.demo01.com`
3. 在浏览器中访问：www.demo01.com



Demo02 - 负载均衡示例

本示例启动三个 JavaApp，访问地址分别为：

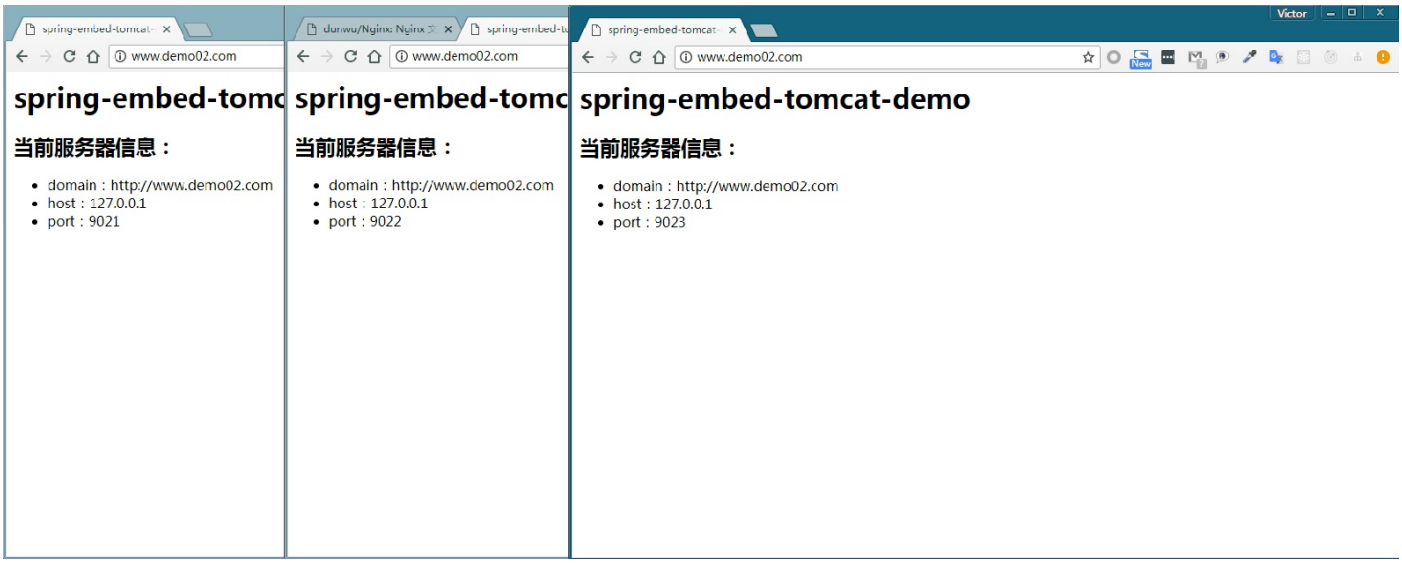
- localhost:9021
- localhost:9022
- localhost:9023

在 Nginx 中统一配置它们的反向代理 host 为 www.demo02.com，并设置相应权重，以便做负载均衡。Nginx 配置文件：demo02.conf

运行步骤：

1. 执行 demo02-start.bat 脚本。
2. 配置 hosts：`127.0.0.1 www.demo02.com`
3. 在浏览器中访问：www.demo02.com

如图所示：三次访问的端口号各不相同，说明三个服务器各自均有不同机率（基于权重）被访问。



Demo03 - 多 webapp 示例

当一个网站功能越来越丰富时，往往需要将一些功能相对独立的模块剥离出来，独立维护。这样的话，通常，会有多个 webapp。

http 的默认端口号是 80，如果在一台服务器上同时启动这 3 个 webapp 应用，都用 80 端口，肯定是不成的。所以，这三个应用需要分别绑定不同的端口号。

本示例启动三个 JavaApp，访问地址分别为：

- localhost:9030/
- localhost:9031/product
- localhost:9032/user

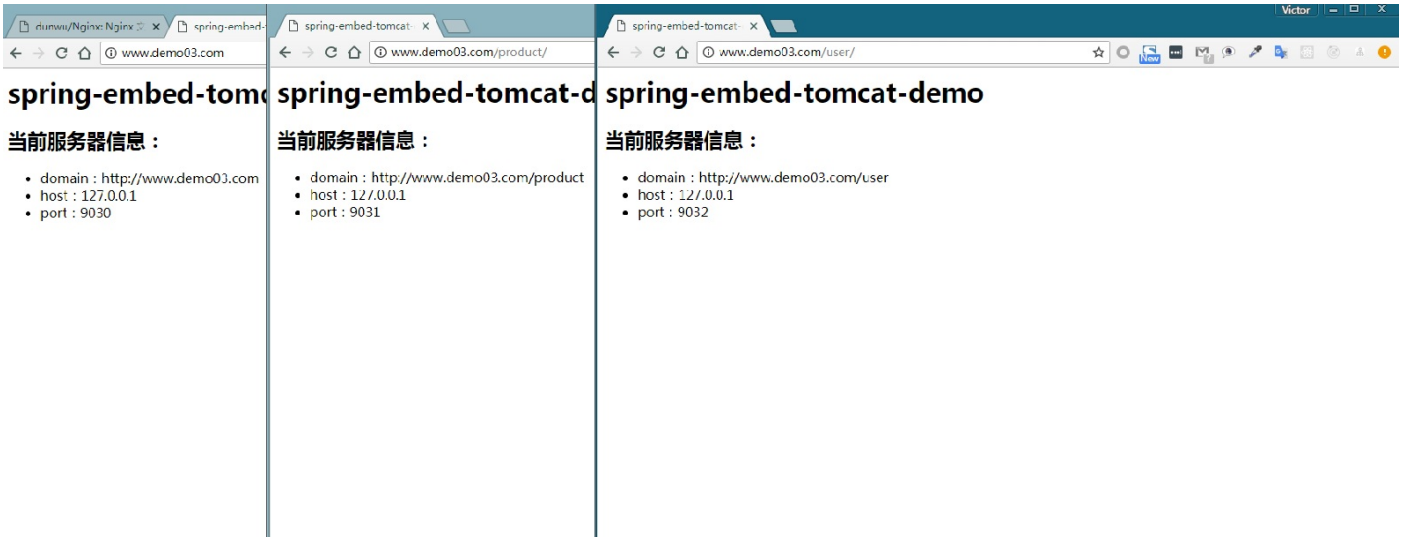
Nginx 中的配置要点就是为每个 server 配置一个 upstream。并在 server 配置下的 location 中指定 context 对应的 upstream。

Nginx 配置文件：demo03.conf

运行步骤：

1. 执行 demo03-start.bat 脚本。
2. 配置 hosts: `127.0.0.1 www.demo03.com`
3. 在浏览器中访问: www.demo03.com

如图所示：三次访问的 context 和端口号各不相同。说明 Nginx 根据不同的 context 将请求分发到指定的服务器上。



Demo04 - 前后端分离示例

做 web 开发，常常会把前后端分离，减少耦合。那么，前后端之间如何通信呢？可以使用 Nginx 来代理。

本例中，后端是一个 java web 项目；前端是一个 react 项目。

Nginx 中的配置要点

指定 root 参数为 react 项目构建后的静态文件存储路径。 指定后端应用的访问地址

```
location ~ ^/api/ {
    proxy_pass http://backend;
    rewrite "^/api/(.*)$" /$1 break;
}
```

Nginx 配置文件: demo04.conf

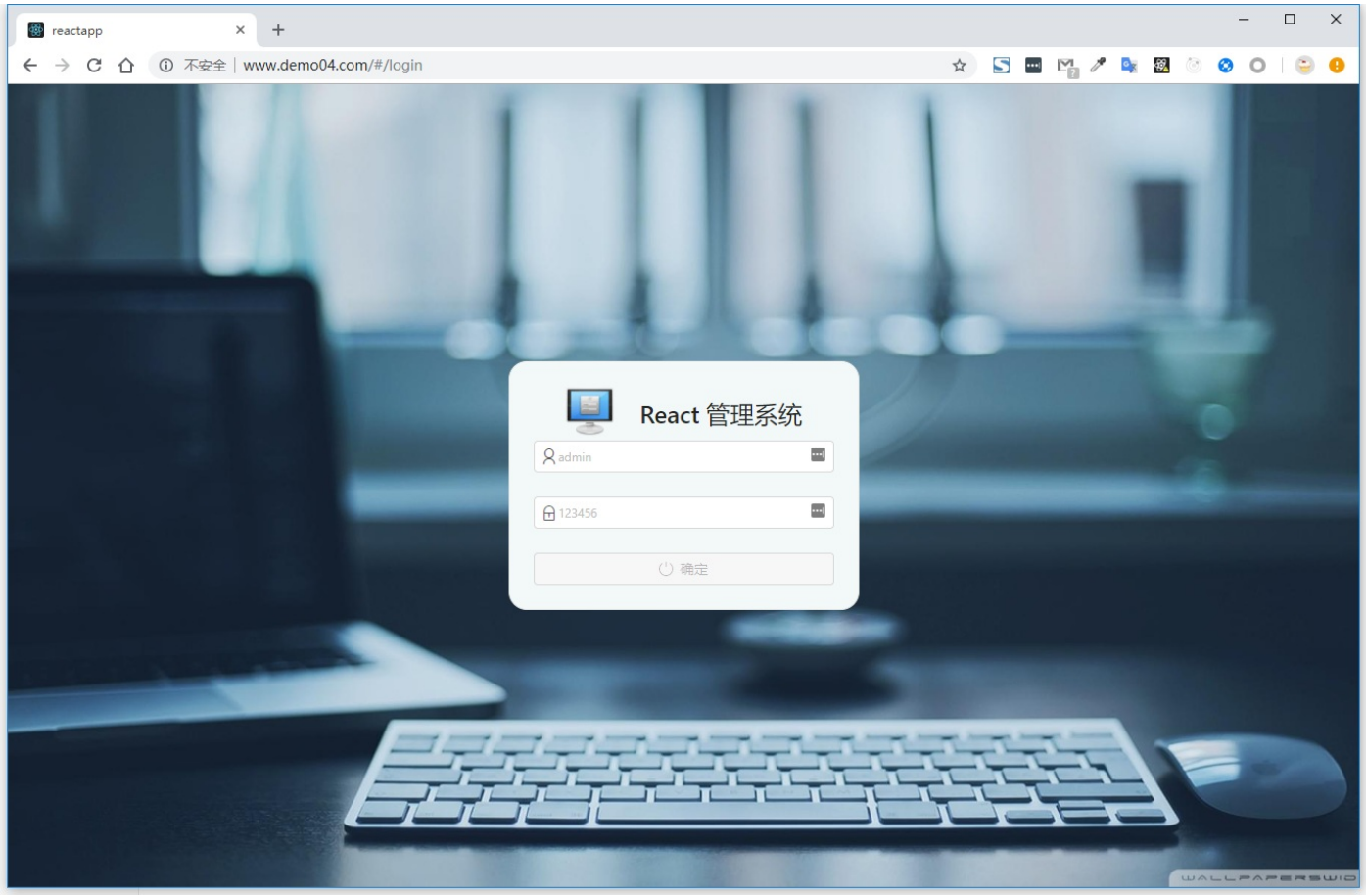
运行准备：由于我的配置中设置 root 的路径为我自己机器中的绝对路径，所以，各位在运行本例的时候要根据自己的实际情况替换。

运行步骤：

1. 执行 demo04-start.bat 脚本。

2. 配置 hosts: `127.0.0.1 www.demo04.com`
3. 在浏览器中访问: www.demo04.com

效果图:



按 F12 打开浏览器控制台，输入用户名/密码（admin/123456）执行登录操作。如下图所示，可以看到登录后的访问请求被转发到了 Nginx 配置的服务器地址。

Demo05 - 配置文件服务器示例

有时候，团队需要归档一些数据或资料，那么文件服务器必不可少。使用 Nginx 可以非常快速便捷的搭建一个简易的文件服务。

Nginx 中的配置要点:

- 将 autoindex 开启可以显示目录，默认不开启。
- 将 autoindex_exact_size 开启可以显示文件的大小。
- 将 autoindex_localtime 开启可以显示文件的修改时间。
- root 用来设置开放为文件服务的根路径。
- charset 设置为 `charset utf-8,gbk;`，可以避免中文乱码问题（windows 服务器下设置

后，依然乱码，本人暂时没有找到解决方法）。

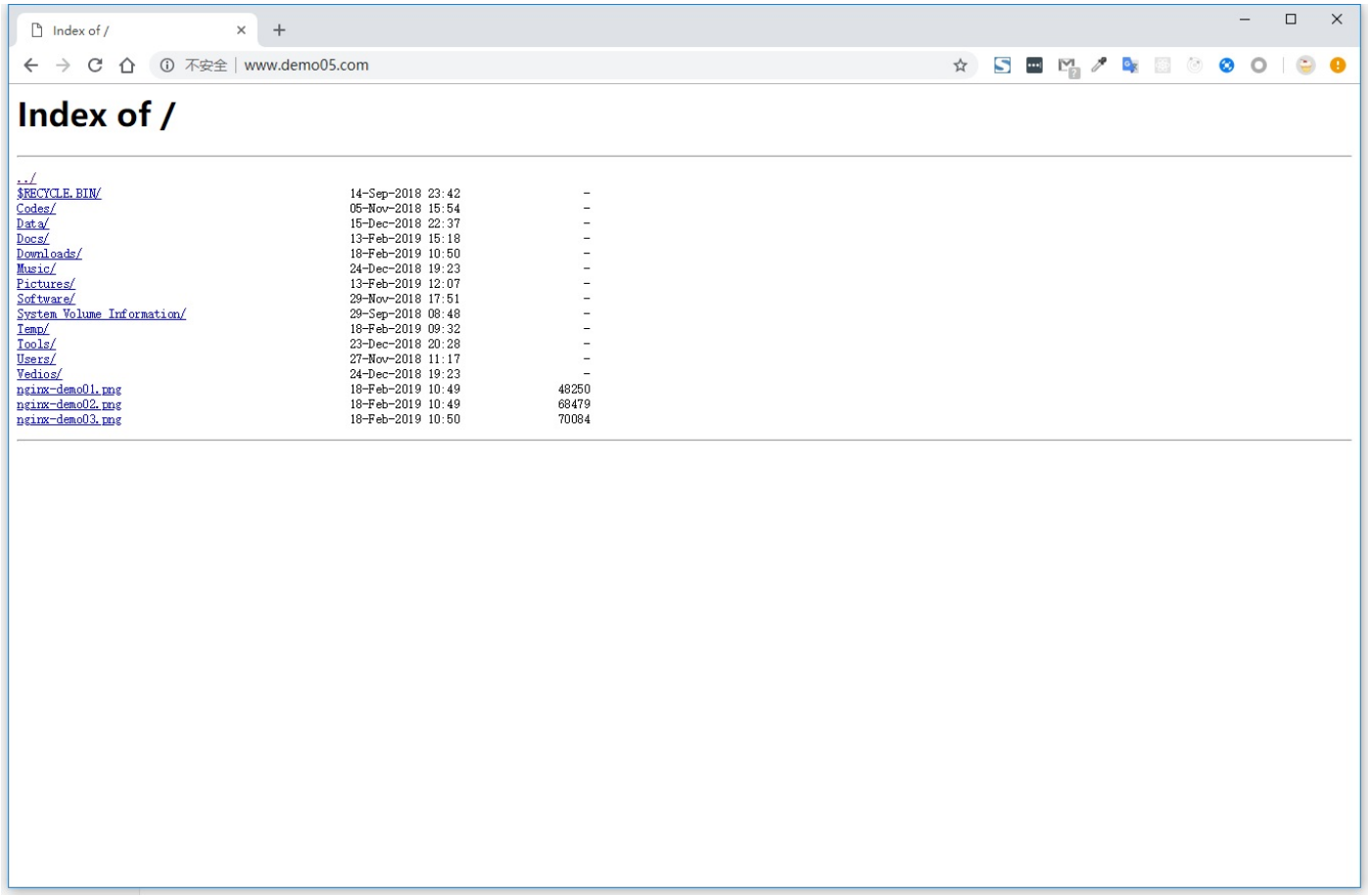
Nginx 配置文件：demo05.conf

运行准备：由于我的配置中设置 root 的路径为我自己机器中的绝对路径，所以，各位在运行本例的时候要根据自己的实际情况替换。

运行步骤：

1. 执行 demo05-start.bat 脚本。
2. 配置 hosts：`127.0.0.1 www.demo05.com`
3. 在浏览器中访问：www.demo05.com

效果图如下：



Demo06 - 静态站点示例

帮助文档、博客、用户手册等等，往往是由 html、js、css、图片等静态资源组成的静态站点型的网站。这时，可以使用 Nginx 快速搭建一个静态访问站点。

Nginx 中的配置要点：

- 设置 location ，指定支持访问的静态资源类型。如：`location \~* ^.+\.(jpg|jpeg|gif|png|ico|css|js|pdf|txt)`

- root 用来设置开放为文件服务的根路径。
- index 用来设置首页。

运行步骤：

1. 执行 build-reactadmin.bat 脚本编译构建一个 React 静态站点项目。
2. 执行 demo06-start.bat 脚本。
3. 配置 hosts: `127.0.0.1 www.demo06.com`
4. 在浏览器中访问：www.demo06.com

效果图如下：