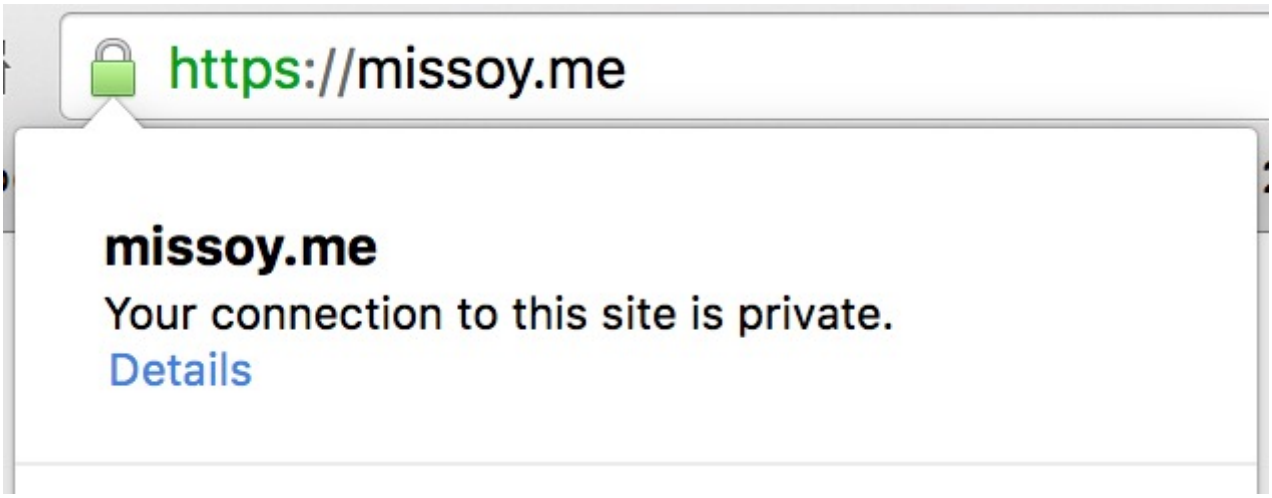
Nginx 配置 HTTPS 服务器

by Mihan (https://github.com/mihanX) on 2016-08-16

Chrome 浏览器地址栏标志着 HTTPS 的绿色小锁头从心理层面上可以给用户专业安全的心理暗示，本文简单总结一下如何在 Nginx 配置 HTTPS 服务器，让自己站点上『绿锁』。



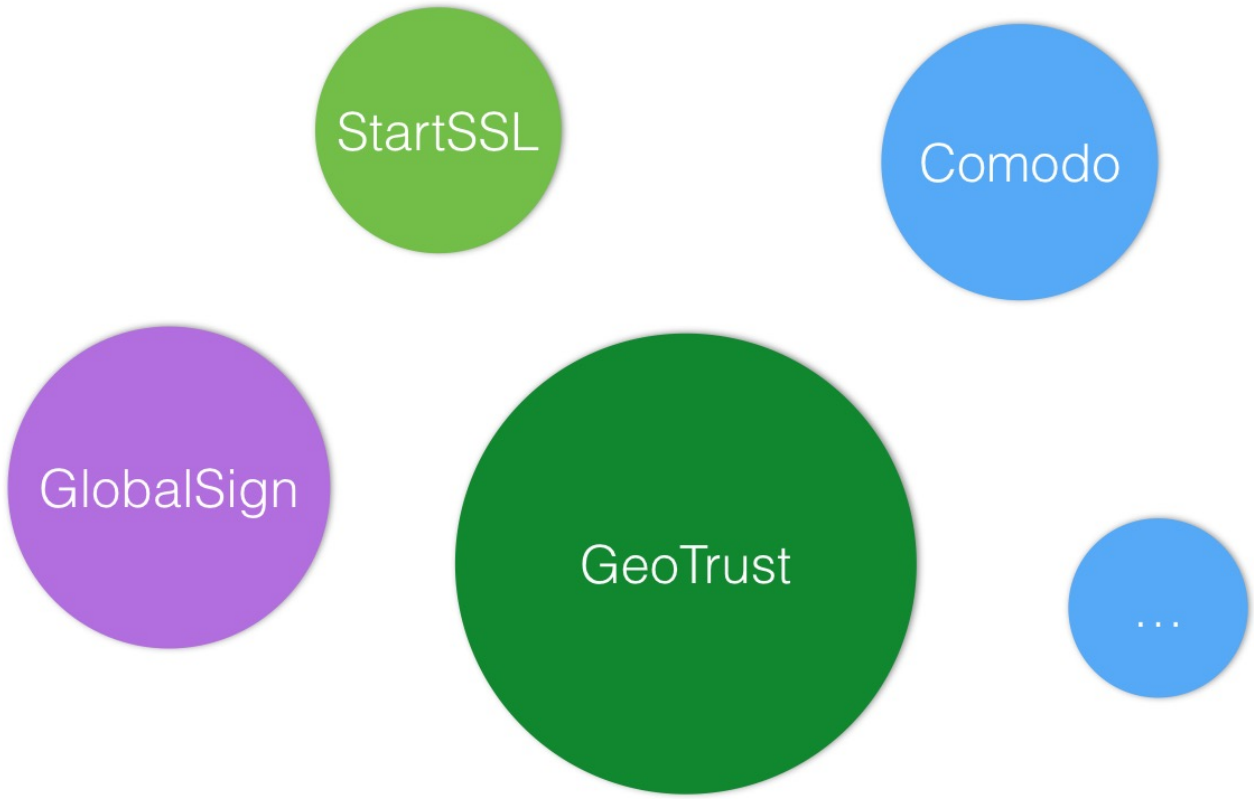
Nginx 配置 HTTPS 并不复杂，主要有两个步骤：签署第三方可信任的 SSL 证书 和 配置 HTTPS

签署第三方可信任的 SSL 证书

关于 SSL 证书

有关 SSL 的介绍可以参阅维基百科的传输层安全协议和阮一峰先生的《SSL/TLS协议运行机制的概述》(http://www.ruanyifeng.com/blog/2014/02/ssl_tls.html)。

SSL 证书主要有两个功能：加密和身份证明，通常需要购买，也有免费的，通过第三方 SSL 证书机构颁发，常见可靠的第三方 SSL 证书颁发机构有下面几个：

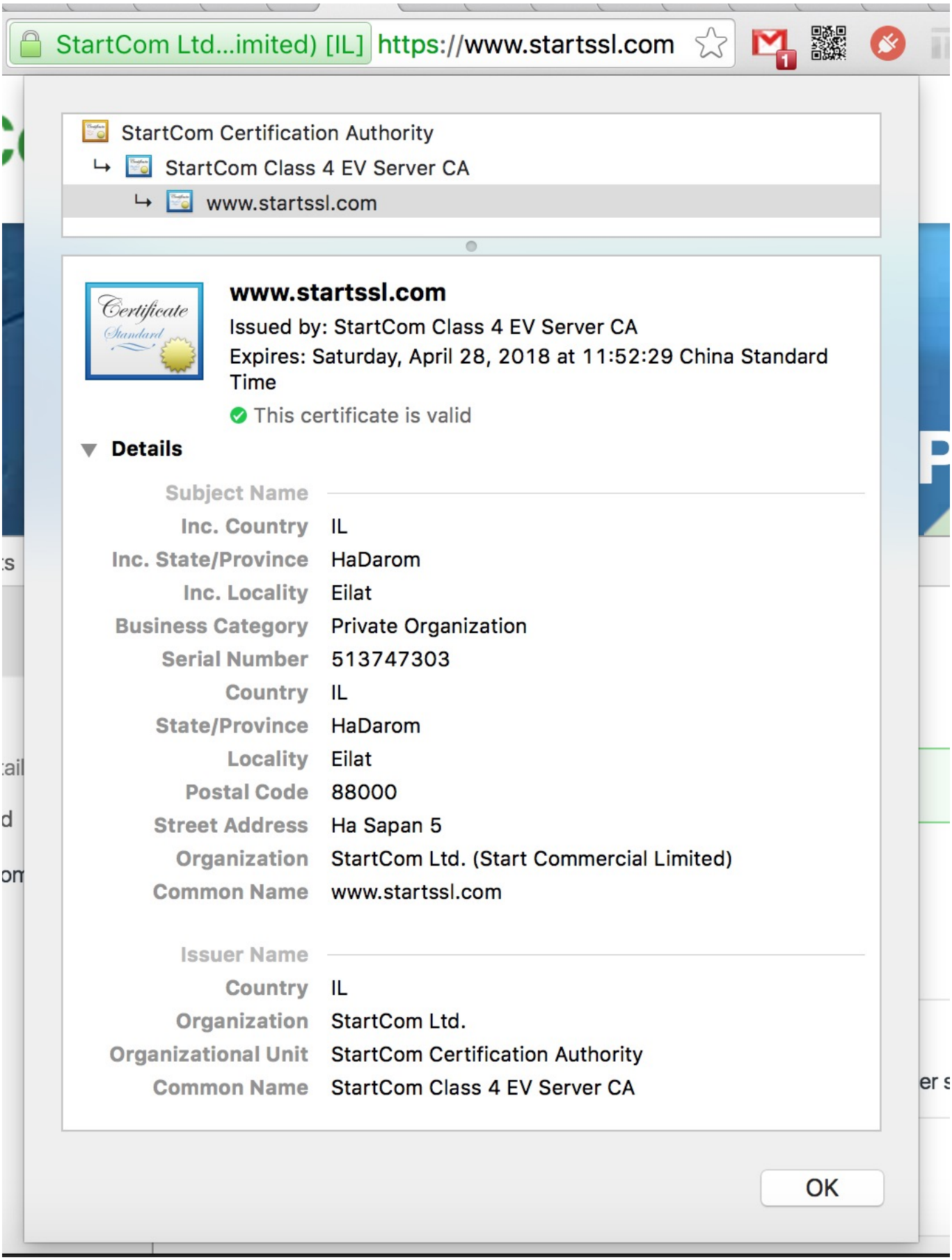


StartCom (https://www.startssl.com/) 机构上的 SSL 证书有以下几种：

- 企业级别：EV(Extended Validation)、OV(Organization Validation)
- 个人级别：IV(Identity Validation)、DV（Domain Validation）

其中 EV、OV、IV 需要付费

免费的证书安全认证级别一般比较低，不显示单位名称，不能证明网站的真实身份，仅起到加密传输信息的作用，适合个人网站或非电商网站。由于此类只验证域名所有权的低端 SSL 证书已经被国外各种欺诈网站滥用，因此强烈推荐部署验证单位信息并显示单位名称的 OV SSL 证书或申请最高信任级别的、显示绿色地址栏、直接在地址栏显示单位名称的 EV SSL 证书，就好像 StarCom 的地址栏一样：



更多关于购买 SSL 证书的介绍：SSL 证书服务，大家用哪家的? 、DV免费SSL证书 (http://freessl.wosign.com/freessl)

使用 OpenSSL 生成 SSL Key 和 CSR 文件

配置 HTTPS 要用到私钥 example.key 文件和 example.crt 证书文件，申请证书文件的时候要用到 example.csr 文件， OpenSSL 命令可以生成 example.key 文件和 example.csr 证书文件。

- CSR：Cerificate Signing Request，证书签署请求文件，里面包含申请者的 DN（Distinguished Name，标识名）和公钥信息，在第三方证书颁发机构签署证书的时候需要提供。证书颁发机构拿到 CSR 后使用其根证书私钥对证书进行加密并生成 CRT 证书文件，里面包含证书加密信息以及申请者的 DN 及公钥信息
- Key：证书申请者私钥文件，和证书里面的公钥配对使用，在 HTTPS 『握手』通讯过程需要使用私钥去解密客戶端发來的经过证书公钥加密的随机数信息，是 HTTPS 加密通讯过程非常重要的文件，在配置 HTTPS 的時候要用到

使用 OpenSSl 命令可以在系统当前目录生成 example.key 和 example.csr 文件：

```
1  openssl req -new -newkey rsa:2048 -sha256 -nodes -out example_com.csr -keyout example_com.
```

下面是上述命令相关字段含义：

- C：Country ，单位所在国家，为两位数的国家缩写，如： CN 就是中国
- ST 字段： State/Province ，单位所在州或省
- L 字段： Locality ，单位所在城市 / 或县区
- O 字段： Organization ，此网站的单位名称;
- OU 字段： Organization Unit，下属部门名称;也常常用于显示其他证书相关信息，如证书类型，证书产品名称或身份验证类型或验证内容等;
- CN 字段： Common Name ，网站的域名;

生成 csr 文件后，提供给 CA 机构，签署成功后，就会得到一個 example.crt 证书文件，SSL 证书文件获得后，就可以在 Nginx 配置文件里配置 HTTPS 了。

配置 HTTPS

基础配置

要开启 HTTPS 服务，在配置文件信息块(server block)，必须使用监听命令 `listen` 的 ssl 参数和定义服务器证书文件和私钥文件，如下所示：

```
1  server {
```

```
server {
    #ssl参数
    listen               443 ssl;
    server_name          example.com;
    #证书文件
    ssl_certificate      example.com.crt;
    #私钥文件
    ssl_certificate_key example.com.key;
    ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers          HIGH:!aNULL:!MD5;
    #...
}
```

证书文件会作为公用实体發送到每台连接到服务器的客戶端，私钥文件作为安全实体，应该被存放在具有一定权限限制的目录文件，并保证 Nginx 主进程有存取权限。

私钥文件也有可能会和证书文件同放在一個文件中，如下面情況：

```
ssl_certificate     www.example.com.cert;
ssl_certificate_key www.example.com.cert;
```

这种情況下，证书文件的的读取权限也应该加以限制，仅管证书和私钥存放在同一个文件里，但是只有证书会被发送到客戶端

命令 `ssl_protocols` 和 `ssl_ciphers` 可以用来限制连接只包含 SSL/TLS 的加強版本和算法，默认值如下：

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers HIGH:!aNULL:!MD5;
```

由于这两个命令的默认值已经好几次发生了改变，因此不建议显性定义，除非有需要额外定义的值，如定义 D–H 算法：

```
#使用DH文件
ssl_dhparam /etc/ssl/certs/dhparam.pem;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
#定义算法
ssl_ciphers "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 EECDH+ECDSA+SHA256 EE
#...
```

HTTPS服务器优化

减少 CPU 运算量

SSL 的运行计算需要消耗额外的 CPU 资源，一般多核处理器系统会运行多个工作进程(worker processes (http://nginx.org/en/docs/ngx_core_module.html#worker_processes))，进程的数量不会少于可用的 CPU 核数。SSL 通讯过程中『握手』阶段的运算最占用 CPU 资源，有两个方法可以减少每台客户端的运算量：

- 激活 keepalive (http://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_timeout) 长连接，一个连接发送更多个请求
- 复用 SSL 会话参数，在并行并发的连接数中避免进行多次 SSL『握手』

这些会话会存储在一个 SSL 会话缓存里面，通过命令 ssl_session_cache 配置，可以使缓存在机器间共享，然后利用客戶端在『握手』阶段使用的 `seesion id` 去查询服务端的 session cathe(如果服务端设置有的话)，简化『握手』阶段。

1M 的会话缓存大概包含 4000 個会话，默认的缓存超时时间为 5 分钟，可以通过使用 ssl_session_timeout 命令设置缓存超时时间。下面是一個拥有 10M 共享会话缓存的多核系统优化配置例子：

```
worker_processes auto;

http {
    #配置共享会话缓存大小
    ssl_session_cache    shared:SSL:10m;
    #配置会话超时时间
    ssl_session_timeout 10m;

    server {
        listen              443 ssl;
        server_name         www.example.com;
        #设置长连接
        keepalive_timeout   70;

        ssl_certificate     www.example.com.crt;
        ssl_certificate_key www.example.com.key;
        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers         HIGH:!aNULL:!MD5;
        #...
```

使用 HSTS 策略强制浏览器使用 HTTPS 连接

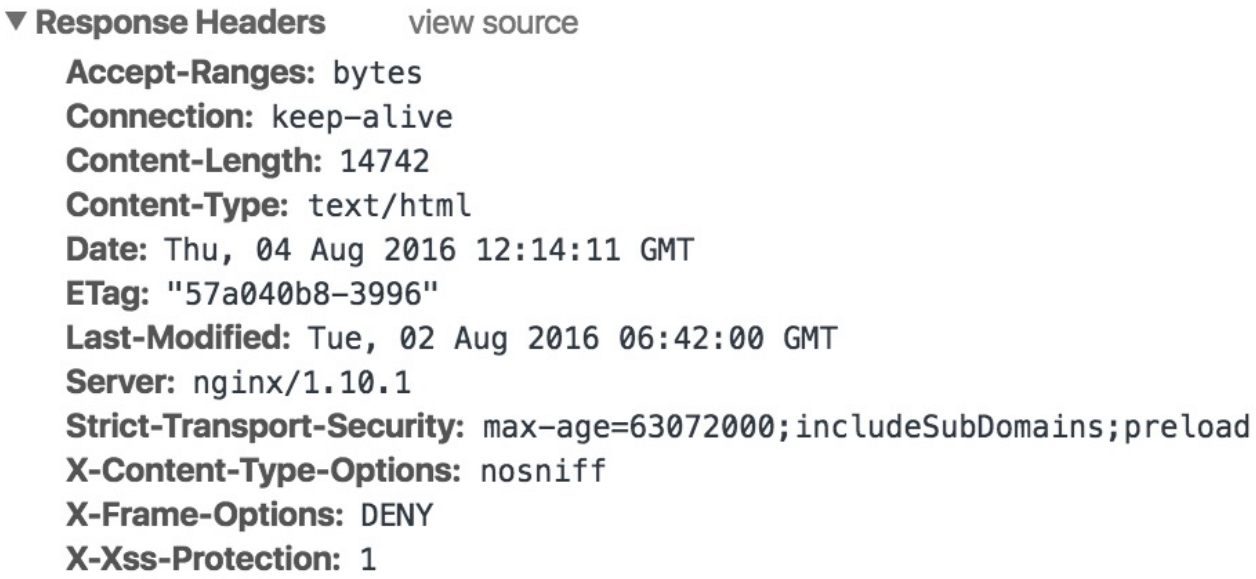
HSTS — HTTP Strict Transport Security，HTTP严格传输安全。它允许一个 HTTPS 网站要求浏览器总是通过 HTTPS 来访问，这使得攻击者在用戶与服务器通讯过程中拦截、篡改信息以及冒充身份变得更为困难。

只要在 Nginx 配置文件加上以下头信息就可以了：

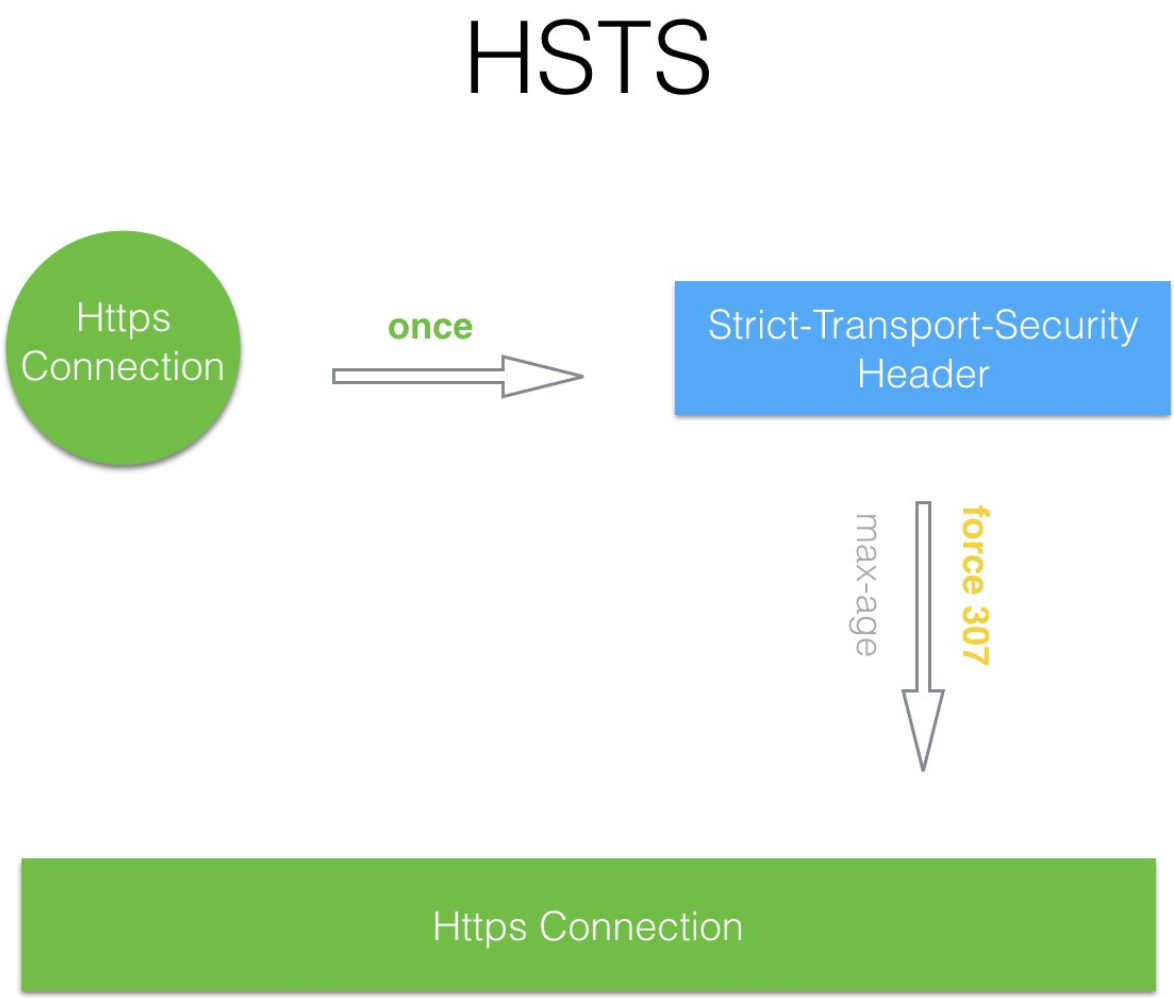
```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains;preload" always;
```

- max-age：设置单位时间内强制使用 HTTPS 连接
- includeSubDomains：可选，所有子域同时生效
- preload：可选，**非规范值**，用于定义使用『HSTS 预加载列表』
- always：可选，保证所有响应都发送此响应头，包括各种内置错误响应

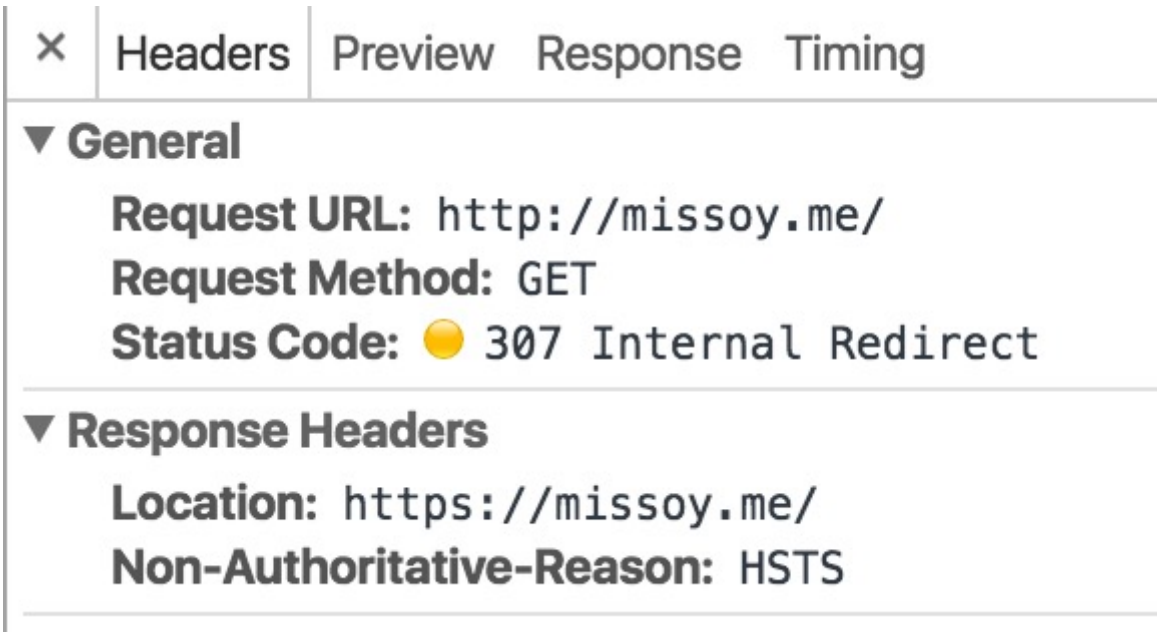
当用户进行 HTTPS 连接的时候，服务器会发送一个 Strict-Transport-Security 响应头：



浏览器在获取该响应头后，在 `max-age` 的时间内，如果遇到 HTTP 连接，就会通过 307 跳转强制使用 HTTPS 进行连接，并忽略其它的跳转设置（如 301 重定向跳转）：



307 跳转 Non-Authoritative-Reason 响应头



Google HSTS 预加载列表（HSTS Preload List）

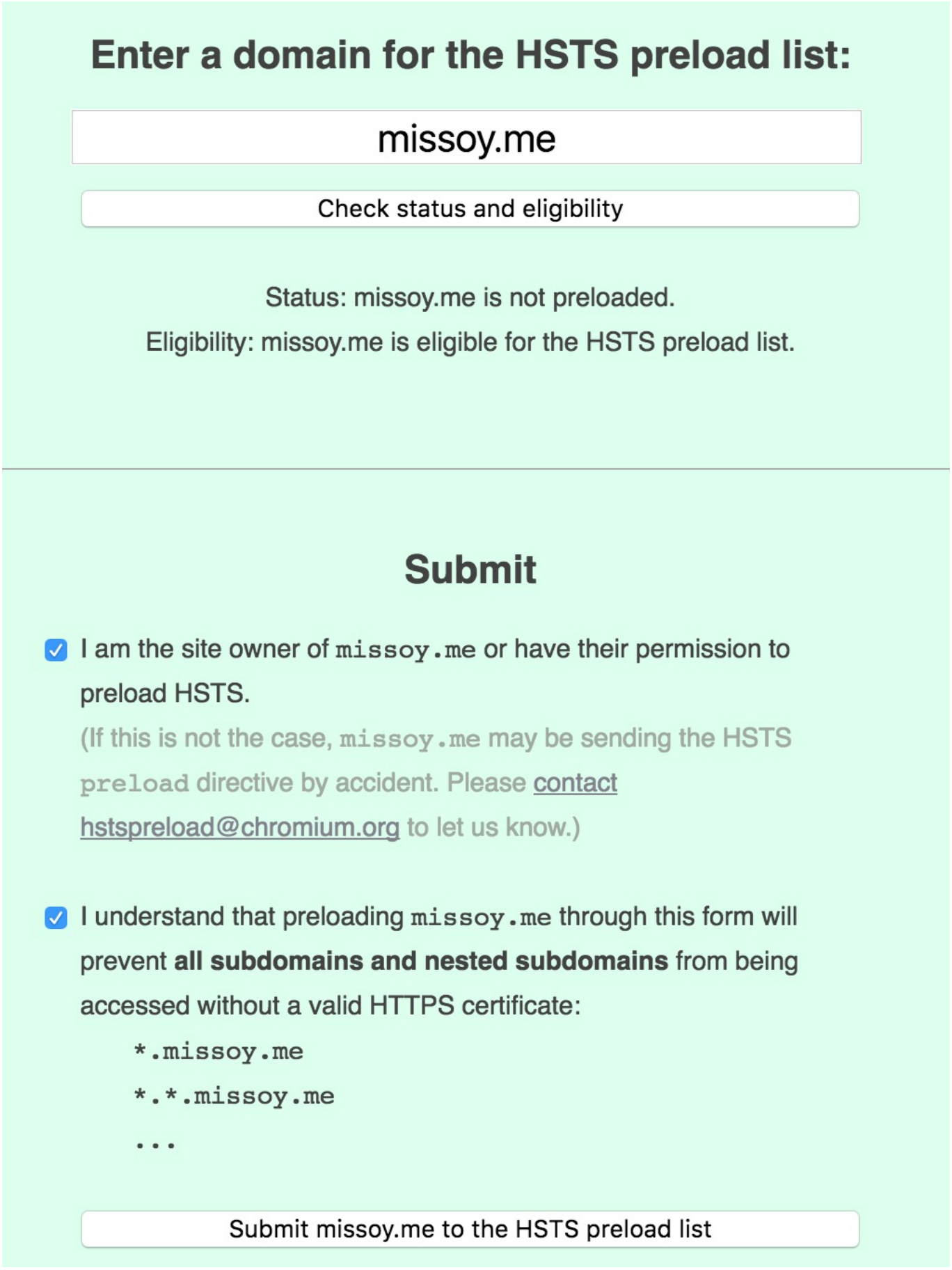
由于 HSTS 需要用戶经过一次安全的 HTTPS 连接后才会在 max-age 的时间內生效，因此 HSTS 策略并不能完美防止 HTTP 会话劫持（HTTP session hijacking）(https://github.com/astaxie/build-web-application-with-golang/blob/master/zh/06.4.md)，在下面这些情况下还是存在被劫持的可能：

- 从未访问过的网站
- 近期重装过操作系統
- 近期重装过浏览器
- 使用新的浏览器
- 使用了新的设备（如手机）
- 删除了浏览器缓存
- 近期沒有打开过网站且 max-age 过期

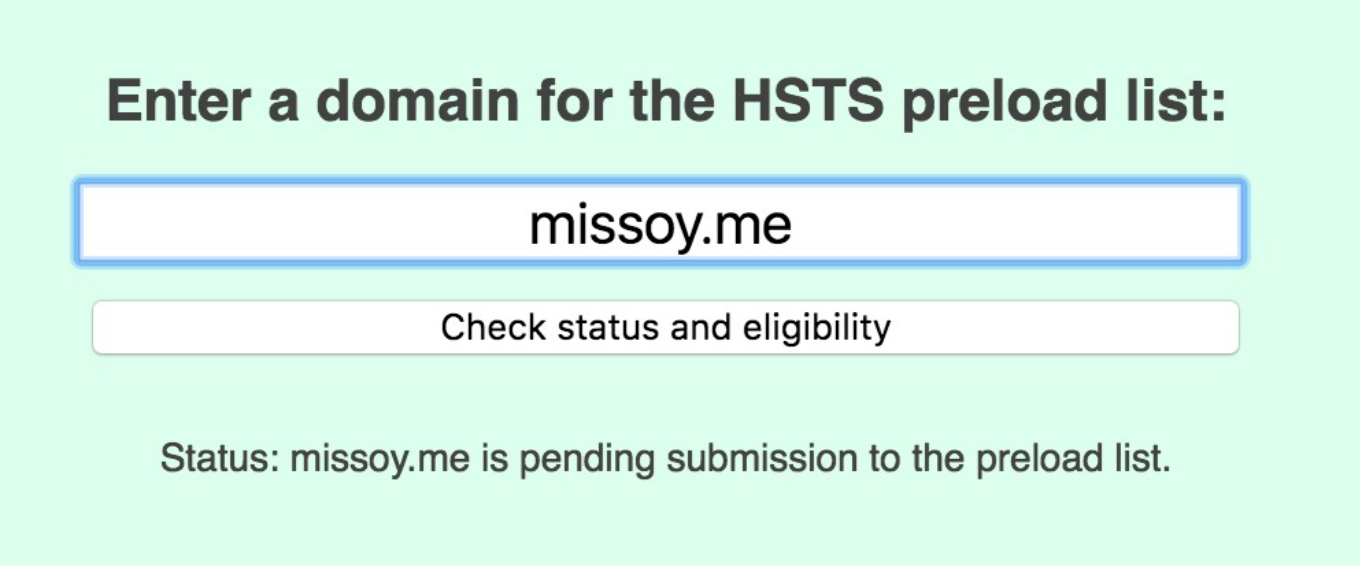
针对这种情况，Google 维护了一份『HSTS 预加载列表』，列表里包含了使用了 HSTS 的站点主域名和子域名，可以通过以下页面申请加入：

https://hstspreload.appspot.com/. (https://hstspreload.appspot.com/.)

申请的时候会先验证站点是否符合资格，一般会检验待验证的站点主域和子域是否能通过 HTTPS 连接、HTTPS 和 HTTP 配置是否有 STS Header 等信息，通过验证后，会让你确认一些限制信息，如下图：



当确认提交后，就会显示处理状态：



申请通过后，列表内的站点名会被写进主流的浏览器，当浏览器更新版本后，只要打开列表内的站点，浏览器会拒绝所有 HTTP 连接而自动使用 HTTPS，即使关闭了 HSTS 设置。

可以在下面两个连接分别查找 Chrome 和 Firfox 的『HSTS 预加载列表』内容：

The Chromium Projects – HTTP Strict Transport Security (https://www.chromium.org/hsts)

Firefox HSTS preload list – nsSTSPreloadList.inc (https://dxr.mozilla.org/comm-central/source/mozilla/security/manager/ssl/nsSTSPreloadList.inc)

需要注意的是：

- 一旦把自己的站点名加入『HSTS 预加载列表』，将很难彻底从列表中移除，因为不能保证其它浏览器可以及时移除，即使 Chrome 提供有便捷的移除方法，也是要通过出邮件联系，注明移除原因，并等到最新的浏览器版本更新发布才有机会（用戶不一定会及时更新）
- 所有不具备有效证书的子域或內嵌子域的访问将会被阻止

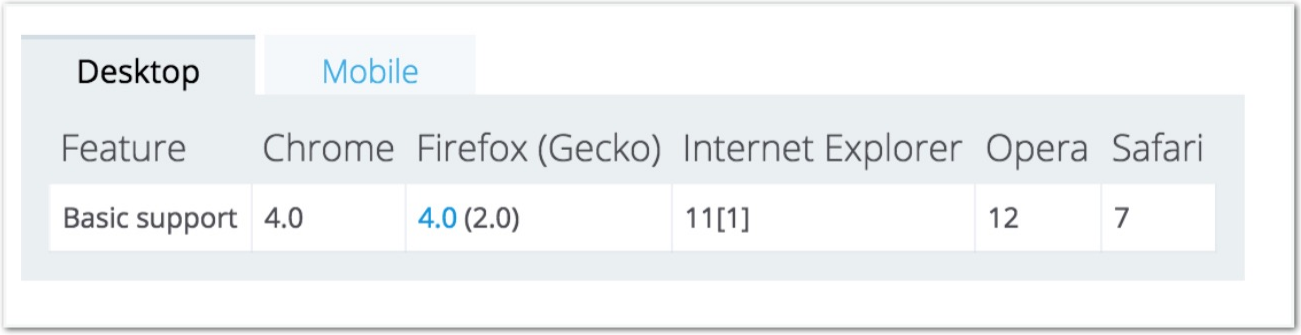
因此，如果自己站点子域名变化比较多，又沒有泛域证书，又沒法确定全站是否能应用 HTTPS 的朋友，就要谨慎申请了。

更多关于 HSTS 配置可参考：

《HTTP Strict Transport Security (HSTS) and NGINX》 (https://www.nginx.com/blog/http-strict-transport-security-hsts-and-nginx/)

MDN的《HTTP Strict Transport Security》 (https://developer.mozilla.org/en-US/docs/Web/Security/HTTP_strict_transport_security#Preloading_Strict_Transport_Security)

浏览器兼容性



Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	4.0	4.0 (2.0)	11[1]	12	7

Desktop Mobile

Feature	Chrome for Android	Firefox Mobile (Gecko)	IE Mobile	Opera Mobile	Safari Mobile
Basic support	18	4.0 (2.0)	No support	No support	8.4

加强 HTTPS 安全性

HTTPS 基础配置采取的默认加密算法是 SHA-1，这个算法非常脆弱，安全性在逐年降低，在 2014 年的时候， Google 官方博客就宣布在 Chrome 浏览器中逐渐降低 SHA-1 证书的安全指示，会从 2015 年起使用 SHA-2 签名的证书，可参阅 Rabbit_Run (http://www.freebuf.com/author/rabbit_run) 在 2014 年发表的文章：《为什么Google急着杀死加密算法SHA-1》 (http://www.freebuf.com/news/topnews/44288.html)

为此，主流的 HTTPS 配置方案应该避免 SHA-1，可以使用 迪菲-赫尔曼密钥交换（D-H，Diffie—Hellman key exchange） (https://zh.wikipedia.org/wiki/%E8%BF%AA%E8%8F%B2-%E8%B5%AB%E7%88%BE%E6%9B%BC%E5%AF%86%E9%91%B0%E4%BA%A4%E6%8F%9B) 方案。

首先在目录 `/etc/ssl/certs` 运行以下代码生成 `dhparam.pem` 文件：

```
openssl dhparam -out dhparam.pem 2048
```

然后加入 Nginx 配置：

```
#优先采取服务器算法
ssl_prefer_server_ciphers on;
#使用DH文件
ssl_dhparam /etc/ssl/certs/dhparam.pem;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
```

```
#定义算法
ssl_ciphers "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 EECDH+ECDSA+SHA256 EE
```

如果服务器夠強大，可以使用更为复杂的 4096 位进行加密。

一般情况下还应该加上以下几个增强安全性的命令：

```
#减少点击劫持
add_header X-Frame-Options DENY;
#禁止服务器自动解析资源类型
add_header X-Content-Type-Options nosniff;
#防XSS攻击
add_header X-Xss-Protection 1;
```

这几个安全命令在 Jerry Qu (https://imququ.com/) 大神的文章《一些安全相关的HTTP响应头》(https://imququ.com/post/web-security-and-response-header.html) 有详细的介紹。

优化后的综合配置

```
worker_processes auto;

http {

    #配置共享会话缓存大小，视站点访问情况设定
    ssl_session_cache    shared:SSL:10m;
    #配置会话超时时间
    ssl_session_timeout 10m;

    server {
        listen              443 ssl;
        server_name         www.example.com;

        #设置长连接
        keepalive_timeout   70;

        #HSTS策略
        add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preloa

        #证书文件
        ssl_certificate     www.example.com.crt;
        #私钥文件
        ssl_certificate_key www.example.com.key;

        #优先采取服务器算法
        ssl_prefer_server_ciphers on;
        #使用DH文件
        ssl_dhparam /etc/ssl/certs/dhparam.pem;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        #定义算法
```

```
        ssl_ciphers "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 EECDH+ECDSA+
        #减少点击劫持
        add_header X-Frame-Options DENY;
        #禁止服务器自动解析资源类型
        add_header X-Content-Type-Options nosniff;
        #防XSS攻擊
        add_header X-Xss-Protection 1;
        #...
```

HTTP/HTTPS混合服务器配置

可以同时配置 HTTP 和 HTTPS 服务器：

```
server {
    listen              80;
    listen              443 ssl;
    server_name         www.example.com;
    ssl_certificate     www.example.com.crt;
    ssl_certificate_key www.example.com.key;
    #...
}
```

在 0.7.14 版本之前，在独立的 server 端口中是不能选择性开启 SSL 的。如上面的例子，SSL 只能通过使用 ssl 命令为单个 server 端口开启

```
server {
    listen              443;
    server_name         www.example.com;
    ssl_certificate     www.example.com.crt;
    ssl_certificate_key www.example.com.key;
    #ssl命令开启 https
    ssl on;
    #...
}
```

因此没有辦法设置 HTTP/HTTPS 混合服务器。于是 Nginx 新增了监听命令 listen参数 `ssl` 來解决这个问题，Nginx 現代版本的 `ssl` 命令并不推荐使用

基于服务器名称（name-based）的 HTTPS 服务器

一个常见的问题就是当使用同一个 IP 地址去配置两个或更多的 HTTPS 服务器的时候，出现证书不匹配的情况：

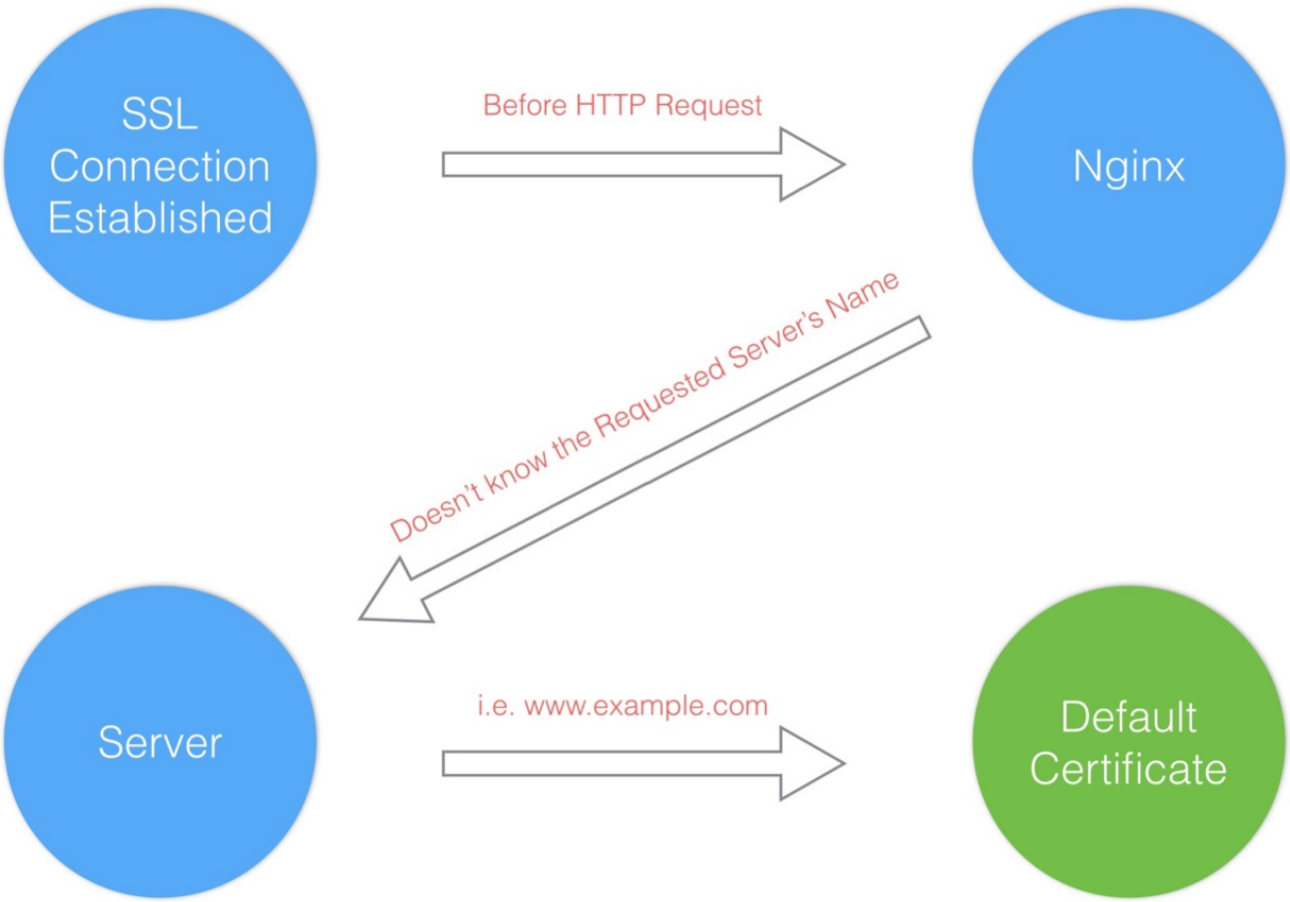
```
server {
```

```
    listen          443 ssl;
    server_name     www.example.com;
    ssl_certificate www.example.com.crt;
    #...
}

server {
    listen          443 ssl;
    server_name     www.example.org;
    ssl_certificate www.example.org.crt;
    #...
}
```

这种情况下浏览器会获取默认的服务器证书（如上面例子的 www.example.com.crt）而忽视请求的服务器名，如输入网址： `www.example.org` ，服务器会发送 `www.example.com.crt` 的证书到客戶端，而不是 `www.exaple.org.crt` 。

这是因为 SSL 协议行为所致，SSL 连接在浏览器发送 HTTP 请求之前就被建立，Nginx 并不知道被请求的服务器名字，因此 Nginx 只会提供默认的服务器证书。



解决这个问题最原始最有效的方法就是为每个 HTTPS 服务器分配独立的 IP 地址：

```
server {
    listen          192.168.1.1:443 ssl;
```

```
    server_name     www.example.com;
    ssl_certificate www.example.com.crt;
    #...
}

server {
    listen          192.168.1.2:443 ssl;
    server_name     www.example.org;
    ssl_certificate www.example.org.crt;
    #...
}
```

更多解决方案

除此之外，官方还介绍了两个方法：泛域证书和域名指示（SNI）

其实 OpenSSL 在 0.9.8f版本就支持 SNI 了，只要在安装的时候加上 `--enable-tlsext` 选项就可以。到了 0.9.8j版本，这个选项在安装的时候会默认启用。如果创建 Nginx 的时候支持 SNI，可以在 Nginx 版本信息查到以下的字段：

```
TLS SNI support enabled
```

因此，如果较新版本的 Nginx 使用默认的 OpenSSL 库，是不存在使用 HTTPS 同时支持基于名字的虚拟主机的时候同 IP 不同域名证书不匹配的问题。

注意：即使新版本的 Nginx 在创建时支持了 SNI，如果 Nginx 动态加载不支持 SNI 的 OpenSSL 库的话，SNI 扩展将不可用

有兴趣的朋友可以看下：

An SSL certificate with several names && Server Name Indication (http://nginx.org/en/docs/http/configuring_https_servers.html)

总结

OK，我们简单总结一下在 Nginx 下配置 HTTPS 的关键要点：

- 获得 SSL 证书
  - 通过 OpenSSL 命令获得 example.key 和 example.csr 文件
  - 提供 example.csr 文件给第三方可靠证书颁发机构，选择适合的安全级别证书并签署，获得 example.crt 文件

- 通过 listen 命令 SSL 参数以及引用 example.key 和 example.crt 文件完成 HTTPS 基础配置
- HTTPS优化
  - 减少 CPU 运算量
    - 使用 keepalive 长连接
    - 复用 SSL 会话参数
  - 使用 HSTS 策略强制浏览器使用 HTTPS 连接
    - 添加 Strict-Transport-Security 头部信息
    - 使用 HSTS 预加载列表（HSTS Preload List）
  - 加强 HTTPS 安全性
    - 使用迪菲-赫尔曼密钥交换（D-H，Diffie—Hellman key exchange）方案
    - 添加 X-Frame-Options 头部信息，减少点击劫持
    - 添加 X-Content-Type-Options 头部信息，禁止服务器自动解析资源类型
    - 添加 X-Xss-Protection 头部信息，防XSS攻击
- HTTP/HTTPS混合服务器配置
- 基于服务器名称（name-based）的 HTTPS 服务器
  - 为每个 HTTPS 服务器分配独立的 IP 地址
  - 泛域证书
  - 域名标识（SNI）

其实简单的个人博客，如果没有敏感数据交互的话，使用 http 协议通讯，一般都夠用了，页面速度还会更快，但正如文章开头所说，戴上『绿锁』，更专业更安全~~有兴趣的同学可以去深入了解折腾下：）