Nginx的使用

参考链接：https://www.docs4dev.com/docs/zh/nginx/current/reference/install.html

1 简介

https://www.nginx.com/resources/wiki/

> NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption.
>
> NGINX is one of a handful of servers written to address the C10K problem. Unlike traditional servers, NGINX doesn't rely on threads to handle requests. Instead it uses a much more scalable event-driven (asynchronous) architecture. This architecture uses small, but more importantly, predictable amounts of memory under load. Even if you don't expect to handle thousands of simultaneous requests, you can still benefit from NGINX's high-performance and small memory footprint. NGINX scales in all directions: from the smallest VPS all the way up to large clusters of servers.

NGINX是一个免费的，开源的高性能HTTP服务器和**反向代理**，以及IMAP / POP3代理服务器。NGINX以其高性能，稳定性，丰富的功能集，简单的配置和低资源消耗而闻名。

NGINX是为解决C10K问题而编写的少数服务器之一。与传统服务器不同，NGINX不依赖线程来处理请求。相反，它使用更具可伸缩性的事件驱动（异步）体系结构。这种架构在负载下使用较小但更重要的是可预测的内存量。即使您不希望同时处理成千上万的请求，您仍然可以从NGINX的高性能和小内存占用中受益。NGINX可以全方位扩展：从最小的VPS一直到大型服务器集群。

2 安装Nginx

2.1 安装

这里我安装的是 nginx-1.18.0 版本，其他版本查看官方地址http://nginx.org/en/download.html

```
# 切换到/usr/local目录
cd /usr/local/
# 下载Nginx包
wget http://nginx.org/download/nginx-1.18.0.tar.gz
# 解压安装包
tar -zxvf nginx-1.18.0.tar.gz
# 进入安装包目录
cd nginx-1.18.0
# 编译并安装
make && make install
```

2.2 启动或重启Nginx

```
# 启动Nignx命令
/usr/local/nginx/sbin/nginx
# 重启Nginx命令
/usr/local/nginx/sbin/nginx -s reload
```

2.3 防火墙设置

```
# 开放访问端口
firewall-cmd --permanent --add-port=80/tcp
# 重新加载防火墙规则
firewall-cmd --reload
# 查看开放的所有防火墙端口
firewall-cmd --list-port
```

3 配置文件的结构

Nginx 由受配置文件中指定的指令控制的模块组成。伪指令分为简单伪指令和块伪指令。一个简单的指令由名称和参数组成，这些名称和参数之间用空格分隔，并以分号(`;`)结尾。块指令的结构与简单指令的结构相同，但它以分号结尾，而不是以分号结尾的一组附加指令(`{`和`}`)包围。如果块指令在括号内可以有其他指令，则称为上下文(示例：events，http，server和location)。

放置在任何上下文外部的配置文件中的指令都被视为在main上下文中。 `events`和`http`指令位于`main`上下文中，`server`在`http`中，而`location`在`server`中。

`#`符号后的其余行被视为注释。

3.1 提供静态内容

Web 服务器的一项重要任务是分发文件(例如图像或静态 HTML 页面)。您将实现一个示例，其中将根据请求从不同的本地目录提供文件：`/data/www`(可能包含 HTML 文件)和`/data/images`(包含图像)。这将需要编辑配置文件，并在带有两个location块的http块内设置一个server块。

首先，创建`/data/www`目录，并将包含任何文本内容的`index.html`文件放入其中，并创建`/data/images`目录，并将一些图像放入其中。

接下来，打开配置文件。默认配置文件已经包含`server`块的几个示例，大部分已注释掉。现在，注释掉所有此类块并开始一个新的`server`块：

```
http {
    server {
    }
}
```

通常，配置文件可按其listen到server names的端口包括几个`server`块distinguished。一旦 Nginx 决定了哪个`server`处理请求，它就会针对`server`块中定义的`location`伪指令的参数测试在请求 Headers 中指定的 URI。

将以下`location`块添加到`server`块：

```
location / {
    root /data/www;
}
```

与来自请求的 URI 相比，此`location`块指定“`/`”前缀。对于匹配的请求，会将 URI 添加到root指令中指定的路径，即`/data/www`，以形成本地文件系统上所请求文件的路径。如果有多个匹配的`location`块，nginx 将选择前缀最长的那个。上面的`location`块提供了最短的前缀(长度为 1)，因此，仅当所有其他`location`块均未提供匹配项时，才会使用该块。

接下来，添加第二个`location`块：

```
location /images/ {
    root /data;
}
```

匹配以`/images/`开头的请求(`location /`也匹配此类请求，但前缀较短)。

`server`块的最终配置应如下所示：

```
server {
    location / {
        root /data/www;
    }

    location /images/ {
        root /data;
    }
}
```

这已经是服务器的工作配置，可以在标准端口 80 上侦听，并且可以在本地计算机`http://localhost/`上访问。响应以`/images/`开头的 URI 请求，服务器将从`/data/images`目录发送文件。例如，响应`http://localhost/images/example.png`请求，nginx 将发送`/data/images/example.png`文件。如果该文件不存在，nginx 将发送一个指示 404 错误的响应。 URI 不以`/images/`开头的请求将被映射到`/data/www`目录。例如，响应`http://localhost/some/example.html`请求，nginx 将发送`/data/www/some/example.html`文件。

要应用新配置，请启动 nginx(如果尚未启动)，或者通过执行以下命令将`reload`signal 发送到 nginx 的主进程：

```
nginx -s reload
```

注意:

如果某些操作无法按预期工作，则可以尝试在目录`/usr/local/nginx/logs`或`/var/log/nginx`的`access.log`和`error.log`文件中查找原因。

3.2 设置简单的代理服务器

nginx 的一种常用用法是将其设置为代理服务器，这意味着服务器可以接收请求，将请求传递给代理服务器，从请求中获取响应并将其发送给客户端。

我们将配置一个基本的代理服务器，该服务器为图像请求和本地目录中的文件提供服务，并将所有其他请求发送到代理服务器。在此示例中，两个服务器都将在单个 nginx 实例上定义。

首先，通过向 nginx 的配置文件中添加一个`server`1 块来定义代理服务器，其内容如下：

```
server {
    listen 8080;
    root /data/up1;

    location / {
    }
}
```

这将是一个简单的服务器，它侦听端口 8080(以前，由于使用了标准端口 80，因此未指定`listen`指令)，并将所有请求映射到本地文件系统上的`/data/up1`目录。创建此目录并将`index.html`文件放入其中。请注意，`root`指令位于`server`上下文中。当选择用于服务请求的`location`块不包含自己的`root`指令时，将使用此类`root`指令。

接下来，使用上一部分中的服务器配置并对其进行修改以使其成为代理服务器配置。在第一个`location`块中，将proxy_pass指令与参数中指定的代理服务器的协议，名称和端口(在本例中为`http://localhost:8080`)放在一起：

```
server {
    location / {
        proxy_pass http://localhost:8080;
    }

    location /images/ {
        root /data;
    }
}
```

我们将修改第二个`location`块，该块当前将带有`/images/`前缀的请求映射到`/data/images`目录下的文件，以使其与具有典型文件 extensions 的图像的请求匹配。修改后的`location`块如下所示：

```
location ~ \.(gif|jpg|png)$ {
    root /data/images;
}
```

该参数是一个正则表达式，匹配所有以`.gif`，`.jpg`或`.png`结尾的 URI。正则表达式应以`~`开头。相应的请求将被映射到`/data/images`目录。

当 nginx 选择一个`location`块来服务请求时，它首先检查指定前缀的location指令，记住最长的`location`，然后检查正则表达式。如果与正则表达式匹配，则 nginx 会选择此`location`，否则，它将选择先前记住的那个。

代理服务器的最终配置如下所示：

```
server {
    location / {
        proxy_pass http://localhost:8080/;
    }

    location ~ \.(gif|jpg|png)$ {
        root /data/images;
    }
}
```

该服务器将过滤以`.gif`，`.jpg`或`.png`结尾的请求，并将它们映射到`/data/images`目录(通过将 URI 添加到`root`指令的参数)，并将所有其他请求传递到上面配置的代理服务器。

要应用新配置，请按照前几节中的说明将`reload`signal 发送到 nginx。

许多more指令可用于进一步配置代理连接。

3.3 设置 FastCGI 代理

nginx 可用于将请求路由到 FastCGI 服务器，该服务器运行使用各种框架和编程语言(例如 PHP)构建的应用程序。

与 FastCGI 服务器一起使用的最基本的 Nginx 配置包括使用fastcgi_pass指令而不是`proxy_pass`指令和fastcgi_param指令来设置传递给 FastCGI 服务器的参数。假设在`localhost:9000`上可访问 FastCGI 服务器。以上一节中的代理配置为基础，将`proxy_pass`指令替换为`fastcgi_pass`指令，并将参数更改为`localhost:9000`。在 PHP 中，`SCRIPT_FILENAME`参数用于确定脚本名称，而`QUERY_STRING`参数用于传递请求参数。结果配置为：

```
server {
    location / {
        fastcgi_pass  localhost:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param QUERY_STRING    $query_string;
    }

    location ~ \.(gif|jpg|png)$ {
        root /data/images;
    }
}
```

这将设置一个服务器，该服务器将通过 FastCGI 协议将除静态图像请求以外的所有请求路由到在`localhost:9000`上运行的代理服务器。

4 使用 Nginx 作为 HTTP 负载均衡器

跨多个应用程序实例的负载平衡是一种用于优化资源利用率，最大化吞吐量，减少延迟和确保容错配置的常用技术。

可以将 nginx 用作非常有效的 HTTP 负载平衡器，以将流量分配到多个应用程序服务器，并使用 nginx 改善 Web 应用程序的性能，可伸缩性和可靠性。

4.1 负载均衡方法

nginx 支持以下负载平衡机制(或方法)：

- 循环-对应用服务器的请求以循环方式分发，
- 最少连接-将下一个请求分配给活动连接数量最少的服务器，
- ip-hash —哈希函数用于确定应为下一个请求选择哪个服务器(基于客户端的 IP 地址)。

4.2 默认负载平衡配置

使用 nginx 进行负载平衡的最简单配置如下所示：

```
http {
    upstream myapp1 {
        server srv1.example.com;
        server srv2.example.com;
        server srv3.example.com;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://myapp1;
        }
    }
}
```

在上面的示例中，同一应用程序的 3 个实例在 srv1-srv3 上运行。如果未特别配置负载平衡方法，则默认为循环。所有请求均为proxied到服务器组 myapp1，并且 nginx 应用 HTTP 负载平衡来分发请求。

nginx 中的反向代理实现包括 HTTP，HTTPS，FastCGI，uwsgi，SCGI，memcached 和 gRPC 的负载平衡。

要为 HTTPS(而非 HTTP)配置负载平衡，只需使用“ https”作为协议。

为 FastCGI，uwsgi，SCGI，memcached 或 gRPC 设置负载平衡时，请分别使用fastcgi_pass，uwsgi_pass，scgi_pass，memcached_pass和grpc_pass指令。

4.3 最少连接的负载平衡

另一个负载平衡原则是连接最少的。最少连接允许在某些请求需要较长时间才能完成的情况下更公平地控制应用程序实例上的负载。

使用最少连接的负载平衡，nginx 将尝试不使繁忙的应用程序服务器因过多的请求而过载，而是将新的请求分配给不太繁忙的服务器。

当least_conn指令用作服务器组配置的一部分时，将激活 nginx 中的最少连接负载均衡：

```
upstream myapp1 {
        least_conn;
        server srv1.example.com;
        server srv2.example.com;
        server srv3.example.com;
    }
```

4.4 Session persistence

请注意，使用循环或最少连接的负载平衡时，每个后续客户端的请求都可能分配到不同的服务器。无法保证将同一客户端始终定向到同一服务器。

如果需要将客户端绑定到特定的应用程序服务器(换句话说，就始终尝试选择特定服务器而言，使客户端的会话为“粘性”或“持久性”)，则可以使用 ip-hash 负载平衡机制用过的。

使用 ip-hash，客户端的 IP 地址用作哈希密钥，以确定应为客户端的请求选择服务器组中的哪个服务器。此方法可确保将来自同一客户端的请求始终定向到同一服务器，除非该服务器不可用。

要配置 ip-hash 负载平衡，只需将ip_hash指令添加到服务器(上游)组配置中：

```
upstream myapp1 {
    ip_hash;
    server srv1.example.com;
    server srv2.example.com;
    server srv3.example.com;
}
```

4.5 加权负载均衡

还可以通过使用服务器权重来进一步影响 nginx 负载平衡算法。

在上面的示例中，未配置服务器权重，这意味着所有指定的服务器都被视为对特定负载平衡方法具有同等资格。

特别是对于循环，这还意味着在服务器之间的请求分配大致相等-前提是存在足够的请求，并且当请求以统一的方式处理并足够快地完成时。

当为服务器指定weight参数时，权重将作为负载平衡决策的一部分。

```
upstream myapp1 {
        server srv1.example.com weight=3;
        server srv2.example.com;
        server srv3.example.com;
    }
```

使用此配置，每 5 个新请求将按以下方式分布在应用程序实例中：3 个请求将定向到 srv1，一个请求将定向到 srv2，另一个将定向到 srv3.

类似地，在最新版本的 nginx 中，可以使用权重最少的连接和 ip-hash 负载平衡。

4.6 Health checks

nginx 中的反向代理实现包括带内(或被动)服务器运行状况检查。如果来自特定服务器的响应失败并出现错误，nginx 会将其标记为失败服务器，并尝试避免为一段时间的后续入站请求选择该服务器。

max_fails伪指令设置应在fail_timeout期间进行的与服务器通信的连续失败尝试次数。默认情况下，max_fails设置为 1.当它设置为 0 时，将禁用此服务器的运行状况检查。 fail_timeout参数还定义服务器将被标记为故障的时间。在服务器发生故障后间隔fail_timeout之后，nginx 将开始使用实时客户端的请求来正常探测服务器。如果探测成功，则将服务器标记为活动服务器。

4.7 Further reading

另外，在 nginx 中还有更多指令和参数可以控制服务器负载平衡，例如proxy_next_upstream，backup，down和keepalive。有关更多信息，请检查我们的reference documentation。

最后但并非最不重要的一点是，应用程序负载平衡，应用程序运行状况检查，activity monitoring和on-the-fly reconfiguration服务器组是我们的 NGINX Plus 付费订阅的一部分。

以下文章更详细地介绍了 NGINX Plus 的负载平衡：

- 使用 NGINX 和 NGINX Plus 进行负载平衡

- 使用 NGINX 和 NGINX Plus 进行负载平衡第 2 部分

5 配置 HTTPS 服务器

要配置 HTTPS 服务器，必须在server块中的listening sockets上启用`ssl`参数，并且应指定server certificate和private key文件的位置：

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

服务器证书是公共实体。它被发送到连接到服务器的每个客户端。私钥是一个安全实体，应存储在访问受限的文件中，但是，nginx 的主进程必须可以读取它。私钥也可以与证书存储在同一文件中：

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

在这种情况下，文件访问权限也应受到限制。尽管证书和密钥存储在一个文件中，但是只有证书被发送到客户端。

指令ssl_protocols和ssl_ciphers可用于限制连接，使其仅包括 SSL/TLS 的强版本和密码。默认情况下，nginx 使用“`ssl_protocols TLSv1 TLSv1.1 TLSv1.2`”和“`ssl_ciphers HIGH:!aNULL:!MD5`”，因此通常不需要显式配置它们。请注意，这些指令的默认值是changed数倍。

5.1 HTTPS 服务器优化

SSL 操作会消耗额外的 CPU 资源。在 multiprocessing 器系统上，应运行多个worker processes，不少于可用的 CPU 内核数。最耗 CPU 的操作是 SSL 握手。有两种方法可以最大程度地减少每个客户端的这些操作数量：第一种方法是通过启用keepalive连接通过一个连接发送多个请求，第二种方法是重用 SSL 会话参数，以避免并行和后续连接的 SSL 握手。会话存储在工作程序之间共享的 SSL 会话缓存中，并由ssl_session_cache指令配置。一兆字节的缓存包含大约 4000 个会话。默认的缓存超时为 5 分钟。可以通过使用ssl_session_timeout指令来增加它。这是针对具有 10 MB 共享会话缓存的多核系统进行优化的示例配置：

```
worker_processes auto;

http {
    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 10m;

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

5.2 SSL 证书链

一些浏览器可能会抱怨由知名证书颁发机构签署的证书，而其他浏览器可能会毫无问题地接受该证书。发生这种情况的原因是，颁发机构已使用中间证书对服务器证书进行了签名，该中间证书在随特定浏览器分发的众所周知的可信证书颁发机构的证书库中不存在。在这种情况下，授权机构将提供一束链接的证书，这些证书应与已签名的服务器证书串联在一起。服务器证书必须出现在组合文件中的链接证书之前：

```
$ cat www.example.com.crt bundle.crt > www.example.com.chained.crt
```

结果文件应在ssl_certificate指令中使用：

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

如果服务器证书和分发包的连接顺序错误，nginx 将无法启动，并显示错误消息：

```
SSL_CTX_use_PrivateKey_file(" ... /www.example.com.key") failed
   (SSL: error:0B080074:x509 certificate routines:
    X509_check_private_key:key values mismatch)
```

因为 nginx 尝试将私有密钥与 Binding 包的第一个证书一起使用，而不是服务器证书。

浏览器通常存储接收到的并由受信任的 Authority 机构签名的中间证书，因此活跃使用的浏览器可能已经具有所需的中间证书，并且可能不会抱怨没有链式 Binding 包发送的证书。为了确保服务器发送完整的证书链，可以使用`openssl`命令行 Util，例如：

```
$ openssl s_client -connect www.godaddy.com:443
...
Certificate chain
 0 s:/C=US/ST=Arizona/L=Scottsdale/1.3.6.1.4.1.311.60.2.1.3=US
     /1.3.6.1.4.1.311.60.2.1.2=AZ/O=GoDaddy.com, Inc
     /OU=MIS Department/CN=www.GoDaddy.com
     /serialNumber=0796928-7/2.5.4.15=V1.0, Clause 5.(b)
   i:/C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc.
     /OU=http://certificates.godaddy.com/repository
     /CN=Go Daddy Secure Certification Authority
     /serialNumber=07969287
 1 s:/C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc.
     /OU=http://certificates.godaddy.com/repository
     /CN=Go Daddy Secure Certification Authority
     /serialNumber=07969287
   i:/C=US/O=The Go Daddy Group, Inc.
     /OU=Go Daddy Class 2 Certification Authority
```

```
 2 s:/C=US/O=The Go Daddy Group, Inc.
     /OU=Go Daddy Class 2 Certification Authority
   i:/L=ValiCert Validation Network/O=ValiCert, Inc.
     /OU=ValiCert Class 2 Policy Validation Authority
     /CN=http://www.valicert.com//emailAddress=info@valicert.com
...
```

注意:

使用SNI测试配置时，重要的是指定`-servername`选项，因为`openssl`默认情况下不使用 SNI。

在此示例中，`www.GoDaddy.com`服务器证书#0 的主题(" s")由本身是证书#1 主题的发行者(" i")签名，证书本身由作为主题的发行者签名的证书#1 证书#2，由著名发行人 ValiCert，Inc.签名，证书存储在浏览器的内置证书库(位于 Jack 所建房屋中)中。

如果尚未添加证书 Binding 包，则仅显示服务器证书#0.

5.3 单个 HTTP/HTTPS 服务器

可以配置一个同时处理 HTTP 和 HTTPS 请求的服务器：

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

注意:

如上所示，在 0.7.14 之前，无法为各个侦听套接字选择性地启用 SSL。仅可以使用ssl指令为整个服务器启用 SSL，从而无法设置单个 HTTP/HTTPS 服务器。添加了listen指令的`ssl`参数来解决此问题。因此不建议在现代版本中使用ssl指令。

5.4 基于名称的 HTTPS 服务器

配置两个或多个侦听单个 IP 地址的 HTTPS 服务器时，会出现一个常见问题：

```
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;
    ...
}
```

使用此配置，浏览器将接收默认服务器的证书，即`www.example.com`，而与请求的服务器名称无关。这是由 SSL 协议行为引起的。在浏览器发送 HTTP 请求之前，已构建 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。

解决此问题的最古老，最可靠的方法是为每个 HTTPS 服务器分配一个单独的 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;
    ...
}
```

5.5 具有多个名称的 SSL 证书

还有其他方法可以在多个 HTTPS 服务器之间共享一个 IP 地址。但是，它们都有缺点。一种方法是在 SubjectAltName 证书字段中使用具有多个名称的证书，例如`www.example.com`和`www.example.org`。但是，SubjectAltName 字段的长度是有限的。

另一种方法是使用带有通配符名称的证书，例如`*.example.org`。通配符证书可保护指定域的所有子域，但只能在一个级别上。该证书与`www.example.org`匹配，但与`example.org`和`www.sub.example.org`不匹配。这两种方法也可以结合使用。证书可以在 SubjectAltName 字段中包含确切的名称和通配符名称，例如`example.org`和`*.example.org`。

最好将带有多个名称的证书文件及其私钥文件放在配置的 http 级别，以在所有服务器中继承其单个内存副本：

```
ssl_certificate     common.crt;
ssl_certificate_key common.key;

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

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

5.6 服务器名称指示

在单个 IP 地址上运行多个 HTTPS 服务器的更通用的解决方案是TLS 服务器名称指示扩展(SNI，RFC 6066)，它允许浏览器在 SSL 握手期间传递请求的服务器名称，因此，服务器将知道应使用哪个证书。用于连接。目前，大多数现代浏览器都使用supported SNI，尽管某些老客户或特殊客户可能不会使用 SNI。

Note

SNI 中只能传递域名，但是，如果请求中包含文字 IP 地址，则某些浏览器可能会错误地将服务器的 IP 地址作为其名称传递。一个不应该依靠这一点。

为了在 nginx 中使用 SNI，必须在已构建 nginx 二进制文件的 OpenSSL 库以及在运行时将其动态链接到的库中都支持它。如果 OpenSSL 使用配置选项“ --enable-tlsext”构建，则从 0.9.8f 版本开始支持 SNI。从 OpenSSL 0.9.8j 开始，默认情况下启用此选项。如果 nginx 是使用 SNI 支持构建的，那么当使用“ -V”开关运行时，nginx 将显示此信息：

```
$ nginx -V
...
TLS SNI support enabled
...
```

但是，如果启用了 SNI 的 nginx 动态链接到不支持 SNI 的 OpenSSL 库，则 nginx 将显示警告：

```
nginx was built with SNI support, however, now it is linked
dynamically to an OpenSSL library which has no tlsext support,
therefore SNI is not available
```

5.7 Compatibility

- 从 0.8.21 和 0.7.62 开始，“-V”开关显示了 SNI 支持状态。
- 自 0.7.14 开始支持listen指令的`ssl`参数。在 0.8.21 之前，只能与`default`参数一起指定。
- 自 0.5.23 起已支持 SNI。
- 从 0.5.6 开始，支持共享 SSL 会话缓存。
- 版本 1.9.1 和更高版本：默认 SSL 协议是 TLSv1，TLSv1.1 和 TLSv1.2(如果 OpenSSL 库支持)。
- 版本 0.7.65、0.8.19 和更高版本：默认 SSL 协议为 SSLv3，TLSv1，TLSv1.1 和 TLSv1.2(如果 OpenSSL 库支持)。
- 版本 0.7.64、0.8.18 和更早版本：默认 SSL 协议为 SSLv2，SSLv3 和 TLSv1.
- 版本 1.0.5 和更高版本：默认 SSL 密码为“ `HIGH:!aNULL:!MD5` ”。
- 版本 0.7.65、0.8.20 及更高版本：默认 SSL 密码为“ `HIGH:!ADH:!MD5` ”。
- 版本 0.8.19：默认 SSL 密码为“ `ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM` ”。
- 0.7.64、0.8.18 及更早版本：默认 SSL 密码为“ `ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP` ”。