5. 端点（Endpoint）

Phoenix应用程序启动 `HelloWeb.Endpoint` 作为管理进程。默认情况下，Endpoint被添加到 `lib/hello/application.ex` 管理树作为管理进程。每个请求都在应用程序的Endpoint开始和结束它的生命周期。Endpoint在调用Router之前启动web服务并通过一些定义好的plug转换请求。

```
defmodule Hello.Application do
  use Application
  def start(_type, _args) do
    ...

    children = [
      HelloWeb.Endpoint
    ]

    opts = [strategy: :one_for_one, name: Hello.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

Endpoint Contents

Endpoint聚合了通用功能，并充当对应用程序所有HTTP请求的入口和出口。Endpoint持有对进入应用程序的所有请求都通用的plug。

让我们看一下Up and Running这篇教程里为 `Hello` 应用生成的Endpoint。

```
defmodule HelloWeb.Endpoint do
  ...
end
```

Endpoint模块内部的第一个调用是带有 `otp_app` 的 `use Phoenix.Endpoint` 宏。 `otp_app` 用于配置。这在 `HelloWeb.Endpoint` 模块上定义了几个函数，包括了在监视树中被调用的 `start_link` 函数。

```
use Phoenix.Endpoint, otp_app: :hello
```

接下来，Endpoint在"/socket" URI上声明一个socket。"/socket"请求将由应用程序其他地方声明的 `HelloWeb.UserSocket` 模块处理。在这里，我们只是声明这种连接将存在。

```
socket "/socket", HelloWeb.UserSocket,
  websocket: true,
  longpoll: false
```

接下来是一系列与我们应用程序中的所有请求相关的plug。我们可以自定义某些功能，例如，开启 `gzip: true` 在部署到生产环境时使用gzip压缩静态文件。

静态文件在我们请求的任何部分把它发送到router之前，都是从 `priv/static` 提供的。

```
plug Plug.Static,
  at: "/",
  from: :hello,
  gzip: false,
  only: ~w(css fonts images js favicon.ico robots.txt)
```

如果启用了代码重新加载，那么当服务器上的代码更改时，socket将用于通知浏览器需要重新加载页面。在开发环境中，默认情况下启用此功能。使用 `config :hello, HelloWeb.Endpoint, code_reloader: true` 进行配置。

```
if code_reloading? do
  socket "/phoenix/live_reload/socket", Phoenix.LiveReloader.Socket
  plug Phoenix.LiveReloader
  plug Phoenix.CodeReloader
end
```

Plug.RequestId为每个请求生成一个唯一的ID，Plug.Telemetry添加了检测点，因此Phoenix默认可以记录请求路径，状态码和请求时间。

```
plug Plug.RequestId
plug Plug.Telemetry, event_prefix: [:phoenix, :endpoint]
```

Plug.Session处理session cookies和session存储。

```
plug Plug.Session, @session_options
```

默认情况下，Endpoint中的最后一个plug是router。router将路径匹配到特定的控制器动作或plug。router在Routing Guide中有介绍。

```
plug HelloWeb.Router
```

可以自定义Endpoint以添加额外的plug，以允许HTTP basic authentication，CORS，subdomain路由等。

管理树不同部分（例如Ecto Repo）的故障不会立即影响主应用程序。因此，管理员可以在发生意外故障后分别重新启动这些进程。一个应用程序也可能有多个Endpoint，每个Endpoint都有自己的管理树。

Endpoint模块中定义了许多函数，用于路径助手、频道订阅和广播、检测和endpoint配置。这些都涵盖在 `Phoenix.Endpoint` 的 Endpoint API docs 文档中。

Using SSL

为了使应用程序能通过SSL来处理请求，我们需要添加一些配置和两个环境变量。为了使SSL真正起作用，我们需要来自证书颁发机构的密钥文件和证书文件。我们需要的环境变量正是这两个文件的路径。

该配置为我们的endpoint包含了一个新的 `https:` key，该key的值是由密钥文件的路径以及cert（pem）文件的路径组成的关键字列表。如果添加 `otp_app:` key，它的值是我们应用程序名称，Plug将开始在我们应用程序的根目录中查找它们。然后，我们可以将这些文件放在 `priv` 目录中，并将路径设置为 `priv/our_keyfile.key` 和 `priv/our_cert.crt` 。

这是一个 `config/prod.exs` 的配置例子。

```
use Mix.Config
```

```
config :hello, HelloWeb.Endpoint,
 http: [port: {:system, "PORT"}],
 url: [host: "example.com"],
 cache_static_manifest: "priv/static/cache_manifest.json",
 https: [
  port: 443,
  cipher_suite: :strong,
  otp_app: :hello,
  keyfile: System.get_env("SOME_APP_SSL_KEY_PATH"),
  certfile: System.get_env("SOME_APP_SSL_CERT_PATH"),
  # OPTIONAL Key for intermediate certificates:
  cacertfile: System.get_env("INTERMEDIATE_CERTFILE_PATH")
 ]
```

没有 `otp_app:` key，我们需要为这些文件提供它们在文件系统上的绝对路径，以便Plug可以找到它们。

```
Path.expand("../../../some/path/to/ssl/key.pem", __DIR__)
```

`https:` key下的选项将被传递到Plug适配器（通常是 `Plug.Cowboy` ），依次使用 `Plug.SSL` 来选择TLS socket选项。请参阅Plug.SSL.configure/1文档，以获取有关可用选项及其默认值的更多信息。Plug HTTPS Guide和Erlang/OTP ssl 文档同样也提供有价值的信息。

SSL in Development

如果您想在开发环境中使用HTTPS，则可以通过运行 `mix phx.gen.cert` 来生成自签名证书。这需要Erlang/OTP 20或更高版本。

使用您的自签名证书，您在 `config/dev.exs` 中的开发环境配置可以更新以运行HTTPS endpoint：

```
config :my_app, MyApp.Endpoint,
  ...
 https: [
  port: 4001,
  cipher_suite: :strong,
```

```
    keyfile: "priv/cert/selfsigned_key.pem",
    certfile: "priv/cert/selfsigned.pem"
  ]
```

这可以替换您的 `http` 配置，您也可以在不同的端口上运行HTTP和HTTPS服务器。

Force SSL

在许多情况下，您需要通过将HTTP重定向到HTTPS来强制所有传入请求使用SSL。这可以通过在您的endpoint配置中设置 `:force_ssl` 选项来完成。它需要一个被转发到 `Plug.SSL` 的选项列表。默认情况下，它在HTTPS请求中设置"strict-transport-security"请求头，从而强制浏览器始终使用HTTPS。如果发送了不安全的（HTTP）请求，它将使用 `:url` 配置中指定的 `:host` 重定向到HTTPS版本。例如：

```
config :my_app, MyApp.Endpoint,
  force_ssl: [rewrite_on: [:x_forwarded_proto]]
```

要动态重定向当前请求到 `host` ，请将 `:force_ssl` 配置中的 `:host` 设置为 `nil` 。

```
config :my_app, MyApp.Endpoint,
  force_ssl: [rewrite_on: [:x_forwarded_proto], host: nil]
```

在这些示例中， `rewrite_on:` key指定应用程序前的反向代理或负载均衡使用的HTTP header，以指示是通过HTTP还是HTTPS接收到的请求。有关将TLS卸载到外部元素（尤其是与安全的cookie有关）的影响的更多信息，请参阅Plug HTTPS Guide。请记住，在Phoenix应用中，那篇文档中传递给 `Plug.SSL` 的选项应该使用 `force_ssl:` endpoint选项来设置。

HSTS

HSTS或"strict-transport-security"是一种允许网站将其声明为只能通过安全链接（HTTPS）访问的机制。引入它是为了防止中间人剥夺SSL/TLS的攻击。除非链接使用SSL/TLS，否则它将导致Web浏览器从HTTP重定向到HTTPS并拒绝连接。

使用 `force_ssl: :hsts` 来设置 `Strict-Transport-Security` 头，它设置了一个最大的使用期限来定义该策略的有效时长。对于标准情况，现代Web浏览器将通过从HTTP重定向到HTTPS来对此作出响应，但它确实有其他后果。定义了HSTS的RFC6797还指定**浏览器应跟踪主机的策略并应用它直到它过期**。它还指定根据该策略**假定除80以外的任何端口上的通信都是加密的**。

如果您在本地主机上访问应用程序，例如https://localhost:4000，这可能导致意外的行为。因为从那时开始，除了将被重定向443端口的80端口，来自本地主机的通信和转发将被加密。这可能会中断您计算机上正在运行的任何其他本地服务或代理的通信。除非对通信进行加密，否则本地主机上的其他应用程序或代理将拒绝工作。

如果您无意中为本地主机打开了HSTS，则可能需要在浏览器上重置缓存，然后才能接受来自本地主机的任何HTTP通信。对于Chrome，您需要执行 `Empty Cache and Hard Reload` ，从开发者工具面板中单击并按住重新加载图标时出现的重新加载菜单就会出现。对于Safari，您需要清除缓存，从 `~/Library/Cookies/HSTS.plist` 删除条目（或完全删除该文件），然后重新启动Safari。或者您可以设置 `force_ssl` 中应该的 `:expires` 选项为 `0` ，该选项将使关闭HSTS的条目过期。有关HSTS选项的更多信息，请访问Plug.SSL。