8. 模板（Templates）

模板听起来像是：我们向其中传递数据以形成完整HTTP响应的文件。对于Web应用程序，这些响应通常是完整的HTML文档。对于API，它们通常是JSON或XML。模板文件中的大多数代码通常是标记，但是Phoenix还将有一部分Elixir代码可以编译和求值。Phoenix模板是预编译的，这使得它们非常快。

EEx是Phoenix中的默认模板系统，它与Ruby中的ERB非常相似。它实际上是Elixir本身的一部分，Phoenix使用EEx模板在生成新应用程序时创建了router和主应用程序视图之类的文件。

正如我们在View Guide中所了解的那样，默认情况下，模板位于 `lib/hello_web/templates` 目录中，组织成以试图命名的目录。每个目录都有其自己的视图模块来渲染其中的模板。

例子（Examples）

我们已经看到了使用模板的几种方法，特别是在Adding Pages Guide和Views Guide中。我们可能在这里涵盖了相同的内容，但是我们肯定会添加一些新信息。

hello_web.ex

Phoenix生成一个 `lib/hello_web.ex` 文件，该文件可用于对常见的imports和aliases进行分组。所有在 `view` 块中的声明都会应用与您所有的模板。

让我们对应用程序进行一些补充，以便我们可以做一些试验。

首先，让我们在 `lib/hello_web/router.ex` 中定义一个新路由。

```
defmodule HelloWeb.Router do
  ...

  scope "/", HelloWeb do
    pipe_through :browser

    get "/", PageController, :index
    get "/test", PageController, :test
```

```
  end

  # Other scopes may use custom stacks.
  # scope "/api", Hello do
  #   pipe_through :api
  # end
end
```

现在，让我们定义在路由中指定的控制器动作。我们将在 `lib/hello_web/controllers/page_controller.ex` 文件中添加一个 `test/2` 动作。

```
defmodule HelloWeb.PageController do
  ...

  def test(conn, _params) do
    render(conn, "test.html")
  end
end
```

我们将创建一个函数，该函数告诉我们哪个控制器和动作正在处理我们的请求。

要做到这一点，我们需要在 `lib/hello_web.ex` 中从 `Phoenix.Controller` 导入 `action_name/1` 和 `controller_module/1` 功能。

```
  def view do
    quote do
      use Phoenix.View, root: "lib/hello_web/templates",
                        namespace: HelloWeb

      # Import convenience functions from controllers
      import Phoenix.Controller, only: [get_flash: 1, get_flash: 2, view_module: 1,
                                        action_name: 1, controller_module: 1]

      ...
    end
  end
```

接下来，让我们在 `lib/hello_web/views/page_view.ex` 底部定义一个 `handler_info/1` 函数，该函数使用

我们刚导入的 `controller_module/1` 和 `action_name/1` 函数。我们还将定义一个 `connection_keys/1` 函数稍后使用。

```
defmodule HelloWeb.PageView do
  use HelloWeb, :view

  def handler_info(conn) do
    "Request Handled By: #{controller_module(conn)}.#{action_name(conn)}"
  end

  def connection_keys(conn) do
    conn
    |> Map.from_struct()
    |> Map.keys()
  end
end
```

我们有一个路由。我们创建了一个新的控制器动作。我们已经对应用程序主视图进行了修改。现在，我们需要的是一个新模板，以显示从 `handler_info/1` 中获取的字符串。让我们创建一个新的 `lib/hello_web/templates/page/test.html.eex` 。

```
<div class="phx-hero">
  <p><%= handler_info(@conn) %></p>
</div>
```

请注意， `@conn` 通过 `assigns` 映射可自由地在模板中使用。

如果我们访问localhost:4000/test，我们将看到页面是由 `Elixir.HelloWeb.PageController.test` 发出的。

我们可以在 `lib/hello_web/views` 中的任意一个视图中定义函数。在单个视图中定义的函数仅可用于该视图渲染的模板。例如，上述的 `handler_info` 函数仅适用于 `lib/hello_web/templates/page` 模板。

显示列表（Displaying Lists）

到目前为止，我们仅在模板中显示了单一种类值——此处为字符串，其他指南中为整数。我们

将如何显示列表的所有元素？

答案是我们可以使用Elixir的列表推导方法。

现在我们有了一个对模板可见的函数，该函数在 `conn` 结构中返回一个keys列表，我们要做的就是稍微修改 `lib/hello_web/templates/page/test.html.eex` 模板以显示它们。

我们可以像这样添加一个header和一个列表推导。

```
<div class="phx-hero">
  <p><%= handler_info(@conn) %></p>

  <h3>Keys for the conn Struct</h3>

  <%= for key <- connection_keys(@conn) do %>
    <p><%= key %></p>
  <% end %>
</div>
```

我们使用 `connection_keys` 函数返回的keys列表作为迭代的源列表。请注意，在列表推导的第一行和用于显示key的哪一行，我们都需要 `<%=` 中的 `=` 。没有它们，实际上什么也不会示。

当我们再次访问localhost:4000/test时，我们会看到所有的key都显示出来了。

在模板中渲染模板（Render templates within templates）

在上面的列表推导示例中，实际显示值的部分非常简单。

```
<p><%= key %></p>
```

我们可以将其保留在原处。但是，这种显示代码通常会更复杂一些，并且将其置于列表推导的中间位置会使我们的模板难以阅读。

简单的解决方案是使用另一个模板！模板只是函数调用，因此，就像常规代码一样，由小的、目的明确的函数组成更大的模板可以使设计更清晰。这就是我们已经看到的渲染链的简单延续。布局是将常规模板渲染到其中的模板。常规模板可能还具有其他模板。

让我们将此显示代码片段转换为自己的模板。让我们在 `lib/hello_web/templates/page/key.html.eex` 上创建一个新的模板文件，像这样。

```
<p><%= @key %></p>
```

我们需要更改此处的 `key` 为 `@key` ，因为这是一个新模板，而不是列表推导的一部分。我们将数据传递到模板的方式是通过 `assigns` 映射，而我们从 `assigns` 映射中取出值的方式是在keys前面加上 `@` 。 `@` 实际上是一个转换 `@key` 为 `Map.get(assigns, :key)` 的宏。

现在我们有了一个模板，我们只需在 `test.html.eex` 模板的列表推导中渲染它即可。

```
<div class="phx-hero">
  <p><%= handler_info(@conn) %></p>

  <h3>Keys for the conn Struct</h3>

  <%= for key <- connection_keys(@conn) do %>
    <%= render("key.html", key: key) %>
  <% end %>
</div>
```

让我们再次看看localhost:4000/test。该页面应看起来像以前一样。

跨视图共享模板（Shared Templates Across Views）

通常，我们发现小块数据需要在应用程序的不同部分中以相同的方式渲染。最好将这些模板移到它们自己的共享目录中，以指示它们应该在应用程序中的任何位置都可用。

让我们将模板移到共享视图中。

`key.html.eex` 当前由 `HelloWeb.PageView` 模块渲染，但是我们使用render调用，它假定当前schema就是我们要渲染的对象。我们可以使它明确，然后像这样重写它：

```
<div class="phx-hero">
  ...

  <%= for key <- connection_keys(@conn) do %>
```

```
    <%= render(HelloWeb.PageView, "key.html", key: key) %>
  <% end %>
</div>
```

由于我们希望它位于一个新 `lib/hello_web/templates/shared` 目录中，因此我们需要一个新的单独视图以在该目录中渲染 `lib/hello_web/views/shared_view.ex` 模板。

```
defmodule HelloWeb.SharedView do
  use HelloWeb, :view
end
```

现在我们可以移动 `key.html.eex` 从 `lib/hello_web/templates/page` 目录到 `lib/hello_web/templates/shared` 目录。一旦发生这种情况，我们可以将 `lib/hello_web/templates/page/test.html.eex` 中的 `render` 调用更改为使用新的 `HelloWeb.SharedView` 。

```
<%= for key <- connection_keys(@conn) do %>
  <%= render(HelloWeb.SharedView, "key.html", key: key) %>
<% end %>
```

再次回到localhost:4000/test。该页面应看起来跟以前一样。