3. 路由（Routing）

Routers是Phoenix应用程序的主要集线器。它们将HTTP请求匹配到控制器动作，连接实时通道处理程序，并定义了一系列管道转换以将中间件的作用域限定为路由集。

Phoenix生成的router文件 `lib/hello_web/router.ex` 看起来像这样：

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :fetch_flash
    plug :protect_from_forgery
    plug :put_secure_browser_headers
  end

  pipeline :api do
    plug :accepts, ["json"]
  end

  scope "/", HelloWeb do
    pipe_through :browser

    get "/", PageController, :index
  end

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

router和controller模块的名称都将以您为应用程序提供的名称开头，而不是 `HelloWeb` 。

该模块的第一行 `use HelloWeb, :router` 只是使Phoenix router函数在我们的特定router中可用。

Scopes在本指南中有自己的部分，因此我们不会花费时间在 `scope "/", HelloWeb do` 这

里。 `pipe_through :browser` 这行将在本指南的Pipeline部分中得到全面处理。现在，您只需要知道pipelines允许将一组中间件转换应用于不同的路由集。

但是，在scope块内，我们有第一个实际的路由：

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

`get` 是Phoenix宏，它扩展为定义 `match/5` 函数的一个子句。它对应于HTTP动词GET。其他HTTP动词也存在类似的宏，包括
POST，PUT，PATCH，DELETE，OPTIONS，CONNECT，TRACE和HEAD。

这些宏的第一个参数是path。在这里，它是应用程序的根， `/` 。接下来的两个参数是我们要处理此请求的控制器和动作。这些宏还可以采用其他选项，我们将在本指南的其余部分中看到这些选项。

如果这是我们router模块中的唯一路由，则 `match/5` 函数的子句在宏扩展后如下所示：

```
def match(:get, "/", PageController, :index, [])
```

`match/5` 函数体建立连接并调用匹配的控制器动作。

当我们添加更多路由时，match函数的更多子句将添加到我们的router模块中。它们的行为将与Elixir中的任何其他多子句函数一样。它们将从顶部开始按顺序尝试，并且将执行匹配给定参数（动词和路径）的第一个子句。找到匹配项后，搜索将停止，并且不会尝试其他子句。

这意味着可以根据HTTP动词和路径创建一个永远不匹配的路由，而与控制器和动作无关。

如果确实创建了模凌两可的路由，router仍会编译，但会收到警告。让我们看看实际情况。

在该router `scope "/", HelloWeb do` 语句的底部定义此路由。

```
get "/", RootController, :index
```

然后在项目的根目录下运行 `mix compile` 。

检查路由（Examining Routes）

Phoenix提供了一个很好的工具来研究应用程序中的路由，即mix 任务 `phx.routes` 。

让我们看看它是如何工作的。前往新生成的Phoenix应用程序的根目录并运行 `mix phx.routes` 。（如果尚未这样做，则需要在运行 `routes` 任务前先运行 `mix do deps.get, compile` ）您应该看到类似以下内容，这些内容是根据我们目前拥有的唯一路由生成的：

```
$ mix phx.routes
page_path  GET  /  HelloWeb.PageController :index
```

输出的内容告诉我们，对于应用程序的根节点，任何HTTP GET请求将被 `HelloWeb.PageController` 的 `index` 动作处理。

`page_path` 是Phoenix称为路径助手的一个例子，我们将很快讨论它们。

资源（Resources）

除了HTTP动词 `get` ， `post` 和 `put` 之外，路由器还支持其他宏。其中最重要的是 `resources` ，它扩展为 `match/5` 函数的八个子句。

让我们在 `lib/hello_web/router.ex` 中添加一个像这样的资源：

```
scope "/", HelloWeb do
  pipe_through :browser

  get "/", PageController, :index
  resources "/users", UserController
end
```

出于这个目的，我们实际上没有 `HelloWeb.UserController` 并不重要。

然后前往项目的根目录并运行 `mix phx.routes`

您应该看到类似以下的内容：

```
user_path  GET     /users          HelloWeb.UserController :index
user_path  GET     /users/:id/edit  HelloWeb.UserController :edit
user_path  GET     /users/new       HelloWeb.UserController :new
user_path  GET     /users/:id       HelloWeb.UserController :show
user_path  POST    /users           HelloWeb.UserController :create
user_path  PATCH   /users/:id       HelloWeb.UserController :update
           PUT     /users/:id       HelloWeb.UserController :update
user_path  DELETE  /users/:id       HelloWeb.UserController :delete
```

当然，您的项目名称将替换 `HelloWeb` 。

这是HTTP动词，路径和控制器动作的标准矩阵。让我们以稍微不同的顺序分别来看它们。

- 对 `/users` 的GET请求将调用 `index` 动作来显示所有用户。
- 对 `/users/:id` 的GET请求将使用一个id调用 `show` 动作，以显示由该ID标识的单个用户。
- 对 `/users/new` 的GET请求将调用 `new` 动作以呈现用于创建新用户的表单。
- 对 `/users` 的POST请求将调用 `create` 动作以将新用户保存到数据存储中。
- 对 `/users/:id/edit` 的GET请求将使用一个id调用 `edit` 动作，以从数据存储中检索单个用户，并以表格形式显示信息以进行编辑。
- 对 `/users/:id` 的PATCH请求将使用一个id调用 `update` 动作，以将更新后的用户保存到数据存储中。
- 对 `/users/:id` 的PUT请求同样将使用一个id调用 `update` 动作，以将更新后的用户保存到数据存储中。
- 对 `/users/:id` 的DELETE请求使用一个id调用 `delete` 动作，以从数据存储中删除单个用户。

如果我们觉得不需要所有这些路由，可以使用 `:only` 和 `:except` 选项进行选择性的操作。

假设我们有一个只读的帖子资源。我们可以这样定义它：

```
resources "/posts", PostController, only: [:index, :show]
```

运行 `mix phx.routes` 显示我们只有定义了index和show动作的路由。

```
post_path  GET     /posts     HelloWeb.PostController :index
post_path  GET     /posts/:id  HelloWeb.PostController :show
```

同样，如果我们拥有一个评论资源，并且不想提供删除的路由，则可以定义这样的路由。

```
resources "/comments", CommentController, except: [:delete]
```

现在运行 `mix phx.routes` 显示我们有除了删除动作的DELETE请求外的所有路由。

```
comment_path  GET    /comments          HelloWeb.CommentController :index
comment_path  GET    /comments/:id/edit  HelloWeb.CommentController :edit
comment_path  GET    /comments/new       HelloWeb.CommentController :new
comment_path  GET    /comments/:id       HelloWeb.CommentController :show
comment_path  POST   /comments          HelloWeb.CommentController :create
comment_path  PATCH  /comments/:id       HelloWeb.CommentController :update
              PUT    /comments/:id       HelloWeb.CommentController :update
```

`Phoenix.Router.resources/4` 宏描述了定制资源路由的其他选项。

转发（Forward）

`Phoenix.Router.forward/4` 宏可以用于将以特定路径开始的所有请求发送到特定的plug。假设我们有一部分系统负责在后台运行jobs（甚至可以是一个单独的应用程序或库），它可以具有自己的web界面来检查jobs的状态。我们可以使用以下命令转发到该admin界面：

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  ...

  scope "/", HelloWeb do
    ...
  end

  forward "/jobs", BackgroundJob.Plug
end
```

这意味着所有以 `/jobs` 开头的路由都将被发送到 `HelloWeb.BackgroundJob.Plug` 模块。

我们甚至可以在pipeline中使用 `forward/4` 宏。如果我们想确保用户通过身份验证并拥有管理员身份才能查看jobs页面，则可以在router中使用以下内容。

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  ...

  scope "/" do
    pipe_through [:authenticate_user, :ensure_admin]
    forward "/jobs", BackgroundJob.Plug
  end
end
```

这意味着 `authenticate_user` 和 `ensure_admin` pipelines中的plug将在 `BackgroundJob.Plug` 允许它们发送适当的响应并调用 `halt()` 之前被调用。

传递给Plug `init/1` 回调的 `opts` 可以作为第三个参数传递。例如，也许后台job页面可让您设置要在页面上显示的应用程序名称。这可以通过下面来实现：

```
forward "/jobs", BackgroundJob.Plug, name: "Hello Phoenix"
```

可以传递第四个参数 `router_opts` 。这些选项在 `Phoenix.Router.scope/2` 文档中有概述。

尽管可以转发到任何模块plug，但不建议转发到另一个endpoint。这是因为您的应用程序定义的plug和转发的endpoint将被调用两次，这可能会导致错误。

编写一个实际的后台job worker不在本指南的讨论范围内。但是，为了方便起见并允许您测试上面的代码，以下是 `BackgroundJob.Plug` 的实现，您可以将其复制到应用程序中的 `lib/plugs/background_job_plug.ex` 里：

```
defmodule HelloWeb.BackgroundJob.Plug do
  def init(opts), do: opts
  def call(conn, opts) do
    conn
    |> Plug.Conn.assign(:name, Keyword.get(opts, :name, "Background Job"))
    |> HelloWeb.BackgroundJob.Router.call(opts)
```

```
  end
end

defmodule HelloWeb.BackgroundJob.Router do
  use Plug.Router

  plug :match
  plug :dispatch

  get "/", do: send_resp(conn, 200, "Welcome to #{conn.assigns.name}")
  get "/active", do: send_resp(conn, 200, "5 Active Jobs")
  get "/pending", do: send_resp(conn, 200, "3 Pending Jobs")
  match _, do: send_resp(conn, 404, "Not found")
end
```

路径助手（Path Helpers）

Path helpers是为单个应用程序动态定义于 `Router.Helpers` 模块的功能。对我们来说，它是 `HelloWeb.Router.Helpers` 。它们的名称是从路由定义所使用的控制器的名字中衍生出来的。我们的控制器是 `HelloWeb.PageController` ，而 `page_path` 是一个会返回应用程序根目录路径的函数。

百闻不如一见。在项目的根目录下运行 `iex -S mix` 。当我们在router helpers上调用 `page_path` 函数，以 `Endpoint` 或connection和action作为参数，它会返回路径。

```
iex> HelloWeb.Router.Helpers.page_path(HelloWeb.Endpoint, :index)
"/"
```

这很有用，因为我们可以在模板中使用 `page_path` 函数链接到应用程序的根目录。然后，我们可以在模板中使用该助手：

```
<a href="<%= Routes.page_path(@conn, :index) %>">To the Welcome Page!</a>
```

之所以可以使用 `Routes.page_path` 而不是全名 `HelloWeb.Router.Helpers.page_path` ，是因为在 `view/0` 中(`lib/hello_web.ex`) 默认定义 `Routes` 作为 `HelloWeb.Router.Helpers` 的别名，并通

过 `use HelloWeb, :view` 可以将其提供给我们的模板。当然，我们可以改用 `HelloWeb.Router.Helpers.page_path(@conn, :index)` ，但是为了简洁起见，约定使用别名版本（请注意，别名仅自动设置以用于视图，控制器和模板中，在这些之外您需要使用全名或在模块定义中自己指定： `alias HelloWeb.Router.Helpers, as: Routes` ）。请参阅View Guide以获取更多信息。

如果我们需要在router中改变路由的路径，这将会带来巨大的回报。由于path helper是基于路由动态构建的，所以在模板中任何对 `page_path` 的调用都能正常工作。

有关路径助手的更多信息（More on Path Helpers）

当我们为user resource运行 `phx.routes` 任务时，它列出了 `user_path` 作为每一行输出的path helper函数。这是每个动作的含义：

```
iex> alias HelloWeb.Router.Helpers, as: Routes
iex> alias HelloWeb.Endpoint
iex> Routes.user_path(Endpoint, :index)
"/users"

iex> Routes.user_path(Endpoint, :show, 17)
"/users/17"

iex> Routes.user_path(Endpoint, :new)
"/users/new"

iex> Routes.user_path(Endpoint, :create)
"/users"

iex> Routes.user_path(Endpoint, :edit, 37)
"/users/37/edit"

iex> Routes.user_path(Endpoint, :update, 37)
"/users/37"

iex> Routes.user_path(Endpoint, :delete, 17)
"/users/17"
```

如果path中有查询字符串呢？通过添加键值对作为第四个可选参数，path helpers将会以查询字符串返回这些键值对。

```
iex> Routes.user_path(Endpoint, :show, 17, admin: true, active: false)
"/users/17?admin=true&active=false"
```

如果我们需要一个完整的url而不是path呢？只需替换 `_path` 为 `_url` ：

```
iex(3)> Routes.user_url(Endpoint, :index)
"http://localhost:4000/users"
```

`_url` 函数将从每个环境设置的配置参数中获取构建完整URL所需的host，port，proxy port和SSL信息。我们将在它自己的指南中更详细地讨论配置。现在，您可以在自己的项目中查看 `config/dev.exs` 文件以查看这些值。

尽可能地传递 `conn` 来代替 `Endpoint` 。

嵌套资源（Nested Resources）

在Phoenix router中也可以嵌套resources。假设我们还有一个 `posts` 资源，它与 `users` 有着多对一的关系。也就是说，一个用户可以创建许多帖子，而单个帖子仅属于一个用户。我们可以在 `lib/hello_web/router.ex` 添加一个这样的嵌套路由来表示：

```
resources "/users", UserController do
  resources "/posts", PostController
end
```

当我们运行 `mix phx.routes` 时，除了上面看到的 `users` 路由外，我们还获得了以下一组路由：

```
...
user_post_path  GET     /users/:user_id/posts          HelloWeb.PostController :index
user_post_path  GET     /users/:user_id/posts/:id/edit HelloWeb.PostController :edit
user_post_path  GET     /users/:user_id/posts/new      HelloWeb.PostController :new
user_post_path  GET     /users/:user_id/posts/:id      HelloWeb.PostController :show
user_post_path  POST    /users/:user_id/posts          HelloWeb.PostController :create
user_post_path  PATCH   /users/:user_id/posts/:id      HelloWeb.PostController :update
                PUT     /users/:user_id/posts/:id      HelloWeb.PostController :update
user_post_path  DELETE  /users/:user_id/posts/:id      HelloWeb.PostController :delete
```

我们看到这些路由中的每一个都将posts限制到了一个用户ID。我们调用 `PostController` `index` 动作，但还会传递一个user_id。这意味着我们只显示某个用户的所有帖子。相同的限制应用于所有这些路由。

当我们为嵌套路由调用path helper函数时，我们需要按路由定义中的顺序传递这些ID。对于下面的 `show` 路由， `42` 是 `user_id` ， `17` 是 `post_id` 。在开始之前记得给我们的 `HelloWeb.Endpoint` 加上别名。

```
iex> alias HelloWeb.Endpoint
iex> HelloWeb.Router.Helpers.user_post_path(Endpoint, :show, 42, 17)
"/users/42/posts/17"
```

同样，如果我们在函数调用的末尾添加一个键/值对，则会将其添加到查询字符串中。

```
iex> HelloWeb.Router.Helpers.user_post_path(Endpoint, :index, 42, active: true)
"/users/42/posts?active=true"
```

如果我们像以前一样给 `Helpers` 模块加了别名（它仅自动为视图，模板和控制器设置别名，因为我们在 `iex` 里面，我们需要自己设置），我们可以改为：

```
iex> alias HelloWeb.Router.Helpers, as: Routes
iex> alias HelloWeb.Endpoint
iex> Routes.user_post_path(Endpoint, :index, 42, active: true)
"/users/42/posts?active=true"
```

作用域路由（Scoped Routes）

Scopes可以将路由分组到有着相同path前缀和一组作用域内的plug中间件里。我们可能想对admin功能，APIs，尤其是对版本化的APIs进行此操作。假设我们有一个用户在站点上生成reviews，并且这些reviews首先需要得到管理员的批准。这些资源的语义完全不同，它们可能不能共享同一个控制器。Scopes使我们能够分隔这些路由。

面向用户的reviews path看起来像是标准资源。

```
/reviews
/reviews/1234
/reviews/1234/edit
...
```

管理员的review path可以以 `/admin` 开头。

```
/admin/reviews
/admin/reviews/1234
/admin/reviews/1234/edit
...
```

我们使用作用域路由来完成此操作，该路由将path选项设置为 `/admin` 这样。现在，我们不要将此scope嵌套在任何其他scopes内（例如，在新应用程序中为我们提供的 `scope "/", HelloWeb do` ）。

```
scope "/admin" do
  pipe_through :browser

  resources "/reviews", HelloWeb.Admin.ReviewController
end
```

还要注意，当前定义此scope的方式，我们需要完全修饰控制器名称为 `HelloWeb.Admin.ReviewController` 。我们将在稍后修复这个问题。

再次运行 `mix phx.routes` ，除了前面的一组路由外，我们还得到以下内容：

```
...
review_path  GET     /admin/reviews          HelloWeb.Admin.ReviewController :index
review_path  GET     /admin/reviews/:id/edit HelloWeb.Admin.ReviewController :edit
review_path  GET     /admin/reviews/new      HelloWeb.Admin.ReviewController :new
review_path  GET     /admin/reviews/:id      HelloWeb.Admin.ReviewController :show
review_path  POST    /admin/reviews          HelloWeb.Admin.ReviewController :create
review_path  PATCH   /admin/reviews/:id      HelloWeb.Admin.ReviewController :update
             PUT     /admin/reviews/:id      HelloWeb.Admin.ReviewController :update
review_path  DELETE  /admin/reviews/:id      HelloWeb.Admin.ReviewController :delete
```

看起来不错，但是这里有问题。记住，我们既要用户面对的reviews路由，也要管理员的 `/admin/reviews` 。如果我们现在将用户面对的reviews包括在router中，如下所示：

```
scope "/", HelloWeb do
  pipe_through :browser
  ...
  resources "/reviews", ReviewController
  ...
end

scope "/admin" do
  resources "/reviews", HelloWeb.Admin.ReviewController
end
```

然后运行 `mix phx.routes` ，我们得到以下输出：

```
...
review_path  GET     /reviews               HelloWeb.ReviewController :index
review_path  GET     /reviews/:id/edit      HelloWeb.ReviewController :edit
review_path  GET     /reviews/new           HelloWeb.ReviewController :new
review_path  GET     /reviews/:id           HelloWeb.ReviewController :show
review_path  POST    /reviews               HelloWeb.ReviewController :create
review_path  PATCH   /reviews/:id           HelloWeb.ReviewController :update
             PUT     /reviews/:id           HelloWeb.ReviewController :update
review_path  DELETE  /reviews/:id           HelloWeb.ReviewController :delete
...
review_path  GET     /admin/reviews         HelloWeb.Admin.ReviewController :index
review_path  GET     /admin/reviews/:id/edit  HelloWeb.Admin.ReviewController :edit
review_path  GET     /admin/reviews/new     HelloWeb.Admin.ReviewController :new
review_path  GET     /admin/reviews/:id     HelloWeb.Admin.ReviewController :show
review_path  POST    /admin/reviews         HelloWeb.Admin.ReviewController :create
review_path  PATCH   /admin/reviews/:id     HelloWeb.Admin.ReviewController :update
             PUT     /admin/reviews/:id     HelloWeb.Admin.ReviewController :update
review_path  DELETE  /admin/reviews/:id     HelloWeb.Admin.ReviewController :delete
```

除了每行开头的 `review_path` path helper，我们实际获得的所有路由都看起来正确。我们为用户面对的review路由和管理员的使用了同一个helper，这是不对的。我们可以通过添加一个 `as: :admin` 选项到admin scope来解决这个问题。

```
scope "/", HelloWeb do
  pipe_through :browser
  ...
  resources "/reviews", ReviewController
  ...
end

scope "/admin", as: :admin do
  resources "/reviews", HelloWeb.Admin.ReviewController
end
```

`mix phx.routes` 现在向我们展示了我们所需要的结果。

```
...
      review_path  GET     /reviews                 HelloWeb.ReviewController :index
      review_path  GET     /reviews/:id/edit        HelloWeb.ReviewController :edit
      review_path  GET     /reviews/new             HelloWeb.ReviewController :new
      review_path  GET     /reviews/:id             HelloWeb.ReviewController :show
      review_path  POST    /reviews                 HelloWeb.ReviewController :create
      review_path  PATCH   /reviews/:id             HelloWeb.ReviewController :update
                   PUT     /reviews/:id             HelloWeb.ReviewController :update
      review_path  DELETE  /reviews/:id             HelloWeb.ReviewController :delete
...
admin_review_path  GET     /admin/reviews           HelloWeb.Admin.ReviewController :index
admin_review_path  GET     /admin/reviews/:id/edit  HelloWeb.Admin.ReviewController :edit
admin_review_path  GET     /admin/reviews/new       HelloWeb.Admin.ReviewController :new
admin_review_path  GET     /admin/reviews/:id       HelloWeb.Admin.ReviewController :show
admin_review_path  POST    /admin/reviews           HelloWeb.Admin.ReviewController :create
admin_review_path  PATCH   /admin/reviews/:id       HelloWeb.Admin.ReviewController :update
                   PUT     /admin/reviews/:id       HelloWeb.Admin.ReviewController :update
admin_review_path  DELETE  /admin/reviews/:id       HelloWeb.Admin.ReviewController :delete
```

path helper现在也会像我们预期的那样运作。运行 `iex -S mix` 并尝试一下。

```
iex(1)> HelloWeb.Router.Helpers.review_path(HelloWeb.Endpoint, :index)
"/reviews"

iex(2)> HelloWeb.Router.Helpers.admin_review_path(HelloWeb.Endpoint, :show, 1234)
"/admin/reviews/1234"
```

如果我们有一些资源都需要管理员处理呢？我们可以将它们全部放在同一个scope内，如下所示：

```
scope "/admin", as: :admin do
  pipe_through :browser

  resources "/images",  HelloWeb.Admin.ImageController
  resources "/reviews", HelloWeb.Admin.ReviewController
  resources "/users",   HelloWeb.Admin.UserController
end
```

这里就是 `mix phx.routes` 将会告诉我们的：

```
...
 admin_image_path  GET     /admin/images          HelloWeb.Admin.ImageController :index
 admin_image_path  GET     /admin/images/:id/edit HelloWeb.Admin.ImageController :edit
 admin_image_path  GET     /admin/images/new      HelloWeb.Admin.ImageController :new
 admin_image_path  GET     /admin/images/:id      HelloWeb.Admin.ImageController :show
 admin_image_path  POST    /admin/images          HelloWeb.Admin.ImageController :create
 admin_image_path  PATCH   /admin/images/:id      HelloWeb.Admin.ImageController :update
                   PUT     /admin/images/:id      HelloWeb.Admin.ImageController :update
 admin_image_path  DELETE  /admin/images/:id      HelloWeb.Admin.ImageController :delete
admin_review_path  GET     /admin/reviews         HelloWeb.Admin.ReviewController :index
admin_review_path  GET     /admin/reviews/:id/edit HelloWeb.Admin.ReviewController :edit
admin_review_path  GET     /admin/reviews/new     HelloWeb.Admin.ReviewController :new
admin_review_path  GET     /admin/reviews/:id     HelloWeb.Admin.ReviewController :show
admin_review_path  POST    /admin/reviews         HelloWeb.Admin.ReviewController :create
admin_review_path  PATCH   /admin/reviews/:id     HelloWeb.Admin.ReviewController :update
                   PUT     /admin/reviews/:id     HelloWeb.Admin.ReviewController :update
admin_review_path  DELETE  /admin/reviews/:id     HelloWeb.Admin.ReviewController :delete
 admin_user_path  GET     /admin/users           HelloWeb.Admin.UserController :index
 admin_user_path  GET     /admin/users/:id/edit  HelloWeb.Admin.UserController :edit
 admin_user_path  GET     /admin/users/new       HelloWeb.Admin.UserController :new
 admin_user_path  GET     /admin/users/:id       HelloWeb.Admin.UserController :show
 admin_user_path  POST    /admin/users           HelloWeb.Admin.UserController :create
 admin_user_path  PATCH   /admin/users/:id       HelloWeb.Admin.UserController :update
                  PUT     /admin/users/:id       HelloWeb.Admin.UserController :update
 admin_user_path  DELETE  /admin/users/:id       HelloWeb.Admin.UserController :delete
```

这很好，正是我们想要的，但是我们可以做得更好。请注意，每个资源的控制器前都要加

上 `HelloWeb.Admin` ，这很乏味且容易出错。假设每个控制器的名称都以 `HelloWeb.Admin` 开头，那么我们可以添加一个 `HelloWeb.Admin` 选项到我们的scope声明中，就在scope path后面。这样我们所有的路由都将具有正确完整的控制器名。

```
scope "/admin", HelloWeb.Admin, as: :admin do
  pipe_through :browser

  resources "/images",  ImageController
  resources "/reviews", ReviewController
  resources "/users",   UserController
end
```

现在再次运行 `mix phx.routes` ，您会看到与上面分别修饰每个控制器名称时相同的结果。

这不仅适用于嵌套路由，我们甚至可以将应用程序的所有路由嵌套在一个作用域中，该作用域的别名就是我们Phoenix应用的名字，这样可以消除控制器名称中重复的应用名。

Phoenix已经在为新应用程序生成的router中为我们做到了（请参阅本节的开头）。注意此处 `HelloWeb` 在 `scope` 声明中的使用：

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  scope "/", HelloWeb do
    pipe_through :browser

    get "/images", ImageController, :index
    resources "/reviews", ReviewController
    resources "/users",   UserController
  end
end
```

再次运行 `mix phx.routes` ，结果表明我们所有的控制器现在都具有正确的，完全修饰的名称。

```
image_path  GET     /images           HelloWeb.ImageController :index
review_path  GET     /reviews          HelloWeb.ReviewController :index
review_path  GET     /reviews/:id/edit  HelloWeb.ReviewController :edit
review_path  GET     /reviews/new      HelloWeb.ReviewController :new
```

```
review_path  GET     /reviews/:id       HelloWeb.ReviewController :show
review_path  POST    /reviews           HelloWeb.ReviewController :create
review_path  PATCH   /reviews/:id       HelloWeb.ReviewController :update
             PUT     /reviews/:id       HelloWeb.ReviewController :update
review_path  DELETE  /reviews/:id       HelloWeb.ReviewController :delete
 user_path  GET     /users             HelloWeb.UserController :index
 user_path  GET     /users/:id/edit    HelloWeb.UserController :edit
 user_path  GET     /users/new         HelloWeb.UserController :new
 user_path  GET     /users/:id         HelloWeb.UserController :show
 user_path  POST    /users             HelloWeb.UserController :create
 user_path  PATCH   /users/:id         HelloWeb.UserController :update
            PUT     /users/:id         HelloWeb.UserController :update
 user_path  DELETE  /users/:id         HelloWeb.UserController :delete
```

尽管从技术上讲，作用域也可以嵌套（就像资源一样），但通常不建议使用嵌套作用域，因为它有时会使我们的代码混乱不清。话虽如此，假设我们有一个版本化的API，包含为images，reviews和用户定义的resources。从技术上讲，我们可以为版本化的API这样设置路由：

```
scope "/api", HelloWeb.Api, as: :api do
 pipe_through :api

 scope "/v1", V1, as: :v1 do
  resources "/images",  ImageController
  resources "/reviews", ReviewController
  resources "/users",   UserController
 end
end
```

`mix phx.routes` 表面我们得到了我们想要的路由。

```
 api_v1_image_path  GET     /api/v1/images          HelloWeb.Api.V1.ImageController :index
 api_v1_image_path  GET     /api/v1/images/:id/edit HelloWeb.Api.V1.ImageController :edit
 api_v1_image_path  GET     /api/v1/images/new      HelloWeb.Api.V1.ImageController :new
 api_v1_image_path  GET     /api/v1/images/:id      HelloWeb.Api.V1.ImageController :show
 api_v1_image_path  POST    /api/v1/images          HelloWeb.Api.V1.ImageController :create
 api_v1_image_path  PATCH   /api/v1/images/:id      HelloWeb.Api.V1.ImageController :update
                    PUT     /api/v1/images/:id      HelloWeb.Api.V1.ImageController :update
 api_v1_image_path  DELETE  /api/v1/images/:id      HelloWeb.Api.V1.ImageController :delete
```

```
api_v1_review_path  GET     /api/v1/reviews          HelloWeb.Api.V1.ReviewController :index
api_v1_review_path  GET     /api/v1/reviews/:id/edit HelloWeb.Api.V1.ReviewController :edit
api_v1_review_path  GET     /api/v1/reviews/new      HelloWeb.Api.V1.ReviewController :new
api_v1_review_path  GET     /api/v1/reviews/:id      HelloWeb.Api.V1.ReviewController :show
api_v1_review_path  POST    /api/v1/reviews          HelloWeb.Api.V1.ReviewController :create
api_v1_review_path  PATCH   /api/v1/reviews/:id      HelloWeb.Api.V1.ReviewController :update
                    PUT     /api/v1/reviews/:id      HelloWeb.Api.V1.ReviewController :update
api_v1_review_path  DELETE  /api/v1/reviews/:id      HelloWeb.Api.V1.ReviewController :delete
 api_v1_user_path  GET     /api/v1/users            HelloWeb.Api.V1.UserController :index
 api_v1_user_path  GET     /api/v1/users/:id/edit   HelloWeb.Api.V1.UserController :edit
 api_v1_user_path  GET     /api/v1/users/new        HelloWeb.Api.V1.UserController :new
 api_v1_user_path  GET     /api/v1/users/:id        HelloWeb.Api.V1.UserController :show
 api_v1_user_path  POST    /api/v1/users            HelloWeb.Api.V1.UserController :create
 api_v1_user_path  PATCH   /api/v1/users/:id        HelloWeb.Api.V1.UserController :update
                   PUT     /api/v1/users/:id        HelloWeb.Api.V1.UserController :update
 api_v1_user_path  DELETE  /api/v1/users/:id        HelloWeb.Api.V1.UserController :delete
```

有趣的是，我们可以让多个scopes具有相同的path，只要我们注意不要重复路由。如果我们重复了路由，将会得到类似的警告。

```
warning: this clause cannot match because a previous clause at line 16 always matches
```

这个路由为同一个path定义了两个scopes，依旧完全正常。

```
defmodule HelloWeb.Router do
  use Phoenix.Router
  ...
  scope "/", HelloWeb do
    pipe_through :browser

    resources "/users", UserController
  end

  scope "/", AnotherAppWeb do
    pipe_through :browser

    resources "/posts", PostController
  end
  ...
end
```

当我们运行 `mix phx.routes` 时，我们看到以下输出：

```
user_path  GET     /users          HelloWeb.UserController :index
user_path  GET     /users/:id/edit  HelloWeb.UserController :edit
user_path  GET     /users/new       HelloWeb.UserController :new
user_path  GET     /users/:id       HelloWeb.UserController :show
user_path  POST    /users           HelloWeb.UserController :create
user_path  PATCH   /users/:id       HelloWeb.UserController :update
           PUT     /users/:id       HelloWeb.UserController :update
user_path  DELETE  /users/:id       HelloWeb.UserController :delete
post_path  GET     /posts           AnotherAppWeb.PostController :index
post_path  GET     /posts/:id/edit  AnotherAppWeb.PostController :edit
post_path  GET     /posts/new       AnotherAppWeb.PostController :new
post_path  GET     /posts/:id       AnotherAppWeb.PostController :show
post_path  POST    /posts           AnotherAppWeb.PostController :create
post_path  PATCH   /posts/:id       AnotherAppWeb.PostController :update
           PUT     /posts/:id       AnotherAppWeb.PostController :update
post_path  DELETE  /posts/:id       AnotherAppWeb.PostController :delete
```

管道（Pipelines）

距离在router中看到的第一行— `pipe_through :browser` 已经过了很久，我们还没谈论到。现在让我们处理它。

还记得Overview Guide中，我们曾把plug描述为按预先确定的顺序堆叠并执行吗？就像管道一样。我们将进一步研究这些plug堆栈是如何在router中工作的。

Pipelines就是plug按特定顺序堆叠在一起，然后起个名字。它们允许我们自定义与请求处理相关的行为和转换。Phoenix为我们提供了一些用于常见任务的默认pipelines。反过来，我们可以定制它们，并创建新的pipeline来满足我们的需求。

新生成的Phoenix应用程序定义了两个pipeline，叫做 `:browser` 和 `:api` 。我们将会介绍这些内容，但是首先我们需要讨论一下Endpoint plug中的plug堆栈。

端点Plugs（The Endpoint Plugs）

Endpoints组织了所有请求都要用到的plugs，并在将它们与底层的 `:browser` 、 `:api` 和自定义管道分配到router之前应用它们。默认的Endpoint plug做了非常多的工作。下面按顺序介绍它们。

- Plug.Static—服务静态资源。由于这个plug位于logger之前，所以静态资源服务是没有登记的。
- Phoenix.CodeReloader—为web目录中的所有入口开启了代码重载。它直接在Phoenix应用程序中配置。
- Plug.RequestId—为每个请求生成唯一的请求ID。
- Plug.Logger—记录传入的请求。
- Plug.Parsers—在已知解析器可用时解析请求的正文。默认情况下，解析器解析urlencoded，multipart和json（包括 `jason` ）。当不能解析请求内容类型时，请求主体将保持不变。
- Plug.MethodOverride—使用有效 `_method` 参数将POST请求的方法转换为PUT，PATCH或DELETE。
- Plug.Head—将HEAD请求转换为GET请求并剥离response body。
- Plug.Session—设置会话管理。请注意， `fetch_session/2` 在使用session之前，仍必须显式被调用，因为这个plug仅设置如何获取会话。
- Plug.Router-将router放进请求周期。

`:browser` 和 `:api` 管道

Phoenix定义了另外两个默认的管道， `:browser` 和 `:api` 。假定我们已经在一个封闭的scope中使用它们调用了 `pipe_through/1` ，则router会在匹配到一个路由之后调用它们。

顾名思义， `:browser` 管道是为准备向浏览器渲染请求的路由提供的。 `:api` 管道是为准备给API产生数据的路由提供的。

`:browser` 管道有五个plug： `plug :accepts, ["html"]` 定义了请求的格式或将要接受的格式， `:fetch_session` 会自然地获取session数据，并让其在连接中可用， `:fetch_flash` 会检索任何设置好的flash信息，以及 `:protect_from_forgery` 与 `:put_secure_browser_headers` ，可保护表单提交免受跨站点伪造。

当前， `:api` 管道仅定义了 `plug :accepts, ["json"]` 。

router在scope内定义的路由上调用pipeline。如果scope未定义，router将会在其所有路由上调用pipeline。尽管不鼓励嵌套scope（请参见上文），但是如果我们在一个嵌套scope内调用 `pipe_through` ，router将从父作用域调用所有的 `pipe_through` ，然后是嵌套作用域。

多说无益，直接上代码。

下面是新生成的Phoenix应用程序的另一个router。这次取消了api scope的注释，并添加了一条路由。

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :fetch_flash
    plug :protect_from_forgery
    plug :put_secure_browser_headers
  end

  pipeline :api do
    plug :accepts, ["json"]
  end

  scope "/", HelloWeb do
    pipe_through :browser

    get "/", PageController, :index
  end

  # Other scopes may use custom stacks.
  scope "/api", HelloWeb do
```

```
    pipe_through :api

    resources "/reviews", ReviewController
  end
end
```

当服务器接受一个请求时，该请求总是会先经过Endpoint上的中的plugs，然后它会试图匹配path和HTTP verb。

假设该请求匹配到了我们的第一个路由：对 `/` 的GET。router会先将请求送至 `:browser` 管道—它会获取session数据，获取flash并执行防伪造保护。然后再将请求分派给 `PageController` `index` 动作。

相反，如果请求匹配到了 `resources/2` 宏定义的任何路由，router会将其送至 `:api` 管道—当前它什么都不会做—在将请求分派到 `HelloWeb.ReviewController` 中正确的动作之前。

如果我们知道我们的应用仅为浏览器渲染视图，则可以通过删除 `api` 以及scopes来简化router。

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :fetch_flash
    plug :protect_from_forgery
    plug :put_secure_browser_headers
  end

  pipe_through :browser

  get "/", HelloWeb.PageController, :index

  resources "/reviews", HelloWeb.ReviewController
end
```

删除了所有scope意味着router强制对其所有路由调用 `:browser` 管道。

让我们进一步思考一下。如果我们需要同时通过 `:browser` 和一个或多个自定义管道输送请求，该怎么办？我们简单地通过 `pipe_through` 管道列表，Phoenix将按顺序调用它们。

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :fetch_flash
    plug :protect_from_forgery
    plug :put_secure_browser_headers
  end
  ...

  scope "/reviews" do
    pipe_through [:browser, :review_checks, :other_great_stuff]

    resources "/", HelloWeb.ReviewController
  end
end
```

这是另一个示例，其中两个scopes有着不同的管道：

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :fetch_flash
    plug :protect_from_forgery
    plug :put_secure_browser_headers
  end
  ...

  scope "/", HelloWeb do
    pipe_through :browser

    resources "/posts", PostController
```

```
  end

  scope "/reviews", HelloWeb do
    pipe_through [:browser, :review_checks]

    resources "/", ReviewController
  end
end
```

通常，管道的scoping规则和你预期一样。在此示例中，所有路由都将会经过 `:browser` 管道。但是，只有 `reviews` 资源路由会经过 `:review_checks` 管道。由于我们是以 `pipe_through [:browser, :review_checks]` 来声明管道的，因此Phoenix会按顺序调用列表中的每个管道。

创建新管道

Phoenix允许我们在router中的任何位置创建自己的自定义管道。为此，我们使用以下参数调用 `pipeline/2` 宏：一个代表管道名的原子，一个包含所有所需plug的块。

```
defmodule HelloWeb.Router do
  use HelloWeb, :router

  pipeline :browser do
    plug :accepts, ["html"]
    plug :fetch_session
    plug :fetch_flash
    plug :protect_from_forgery
    plug :put_secure_browser_headers
  end

  pipeline :review_checks do
    plug :ensure_authenticated_user
    plug :ensure_user_owns_review
  end

  scope "/reviews", HelloWeb do
    pipe_through :review_checks

    resources "/", ReviewController
```

```
  end
end
```

频道路由（Channel Routes）

频道是Phoenix框架中非常令人兴奋的实时组件。频道会处理给定主题的socket广播的传入和传出消息。因此，频道路由需要按socket和主题匹配请求，以便分配到正确的频道。（关于频道及其行为的详细说明，请参阅Channel Guide）

我们在 `lib/hello_web/endpoint.ex` 中的endpoint挂载了socket handlers。Socket handlers会处理认证回调和频道路由。

```
defmodule HelloWeb.Endpoint do
  use Phoenix.Endpoint, otp_app: :hello

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

默认情况下，在endpoint中调用 `Phoenix.Endpoint.socket/3` 时，它同时支持websocket和longpoll。这里我们指定可以通过WebSocket连接建立传入的socket连接。

接下来，我们需要打开 `lib/hello_web/channels/user_socket.ex` 文件并使用 `channel/3` 宏来定义我们的通道路由。路由会将一个主题模式与一个通道匹配以处理事件。如果我们有一个名为 `RoomChannel` 的通道模块和一个名为 `"rooms:*"` 的主题，那么执行此操作的代码可以这样写。

```
defmodule HelloWeb.UserSocket do
  use Phoenix.Socket

  channel "rooms:*", HelloWeb.RoomChannel
  ...
end
```

主题只是字符串标识符。我们在这里使用的形式是一种约定，它使我们可以在同一字符串中定义主题和子主题—`“topic：subtopic”`。`*`是一个通配符，它允许我们匹配任何子话题，所以`"rooms:lobby"`和`"rooms:kitchen"`会都匹配该路由。

每个socket可以为多个频道处理请求。

```
channel "rooms:*", HelloWeb.RoomChannel
channel "foods:*", HelloWeb.FoodChannel
```

我们可以把多个socket handlers挂载到endpoint中：

```
socket "/socket", HelloWeb.UserSocket
socket "/admin-socket", HelloWeb.AdminSocket
```

总结

路由是一个很大的话题，在这里我们花了许多笔墨。本指南的重点是：

- 以HTTP verb名为开头的路由会展开成match函数的一个子句。
- 以“resources”为开头的路由会展开为match函数的8个子句。
- Resources可以通过使用`only:`或`except:`选项来限制match函数子句的数量。
- 任何路由都可以嵌套。
- 任何路由都可以被scoped到一个给定的path。
- 在scope中使用`as:`选项可以减少重复。
- 对scoped路由使用helper选项可消除无法获取的paths。