Chapter 8: Controllers

By Tim Condon

在前面的章节中，您已经在**routes.swift**中编写了所有路由处理程序。对于大型项目来说，这是不可持续的，因为文件很快变得太大而且杂乱无章。本章介绍了使用基本控制器和RESTful控制器来帮助管理路由和模型的控制器概念。

> 注意：本章要求您已设置并配置PostgreSQL。按照第6章“Configuring a Database”中的步骤，在Docker中设置PostgreSQL并配置Vapor应用程序。

Controllers

Vapor中的控制器与iOS中的控制器具有相似的用途。它们处理来自客户端的交互，例如请求，处理它们并返回响应。控制器提供了一种更好地组织代码的方法。在专用控制器中与模型进行所有交互是一种很好的做法。例如，在TIL应用程序中，缩略词控制器可以处理缩略词上的所有CRUD操作。

控制器还用于组织您的应用程序。例如，您可以使用一个控制器来管理旧版本的API，而使用另一个控制器来管理当前版本。这样可以明确分离代码中的职责并保持代码的可维护性。

控制器入门

创建一个新文件来保存缩略词控制器。在终端中，输入：

```
# 1
cd ~/vapor/TILApp/
# 2
touch Sources/App/Controllers/AcronymsController.swift
# 3
vapor xcode -y
```

这是它的作用：

1. 导航到TIL应用程序所在的目录。
2. 在**App**模块的**Controllers**目录中创建一个新文件**AcronymsController.swift**。
3. 重新生成**Xcode**项目以将新文件添加到**App** target。

Route collections

在控制器内部，您可以定义不同的路由处理程序。要访问这些路由，必须使用路由器注册这些处理程序。一种简单的方法是从**routes.swift**调用控制器内的函数。例如：

```
router.get(
  "api",
  "acronyms",
  use: acronymsController.getAllHandler)
```

此示例在acronymsController上调用getAlllHandler(_:)。这个调用就像你在第7章中编写的路由处理程序一样。但是，不是将闭包作为最后一个参数传递，而是传递要使用的函数。

这适用于小型应用程序。但是如果要注册大量路由，**routes.swift**将再次变得无法管理。控制器负责注册他们控制的路由是一种很好的做法。 Vapor提供RouteCollection协议来实现此功能。

在Xcode中打开**AcronymsController.swift**并添加以下内容以创建一个遵循RouteCollection协议的AcronymsController：

```
import Vapor
import Fluent

struct AcronymsController: RouteCollection {
  func boot(router: Router) throws {

  }
}
```

RouteCollection要求您实现boot(router:)来注册路由。在boot(router:)后面添加新的路由处理程序：

```
func getAllHandler(_ req: Request) throws -> Future<[Acronym]> {
  return Acronym.query(on: req).all()
}
```

处理程序的函数体与您之前编写的函数体相同，函数原型与您之前使用的闭包的原型相匹配。在 boot(router:)中注册路由：

```
router.get("api", "acronyms", use: getAllHandler)
```

这使得对**/api/acronyms**的GET请求调用getAllHandler(_ :)。你之前在**routes.swift**中写了相同的路由。现在，是时候删除它了。打开**routes.swift**并删除以下处理程序：

```
router.get("api", "acronyms") { req -> Future<Acronym> in
  return Acronym.query(on: req).all()
}
```

接下来，将以下内容添加到routes(_:)末尾：

```
// 1
let acronymsController = AcronymsController()
// 2
try router.register(collection: acronymsController)
```

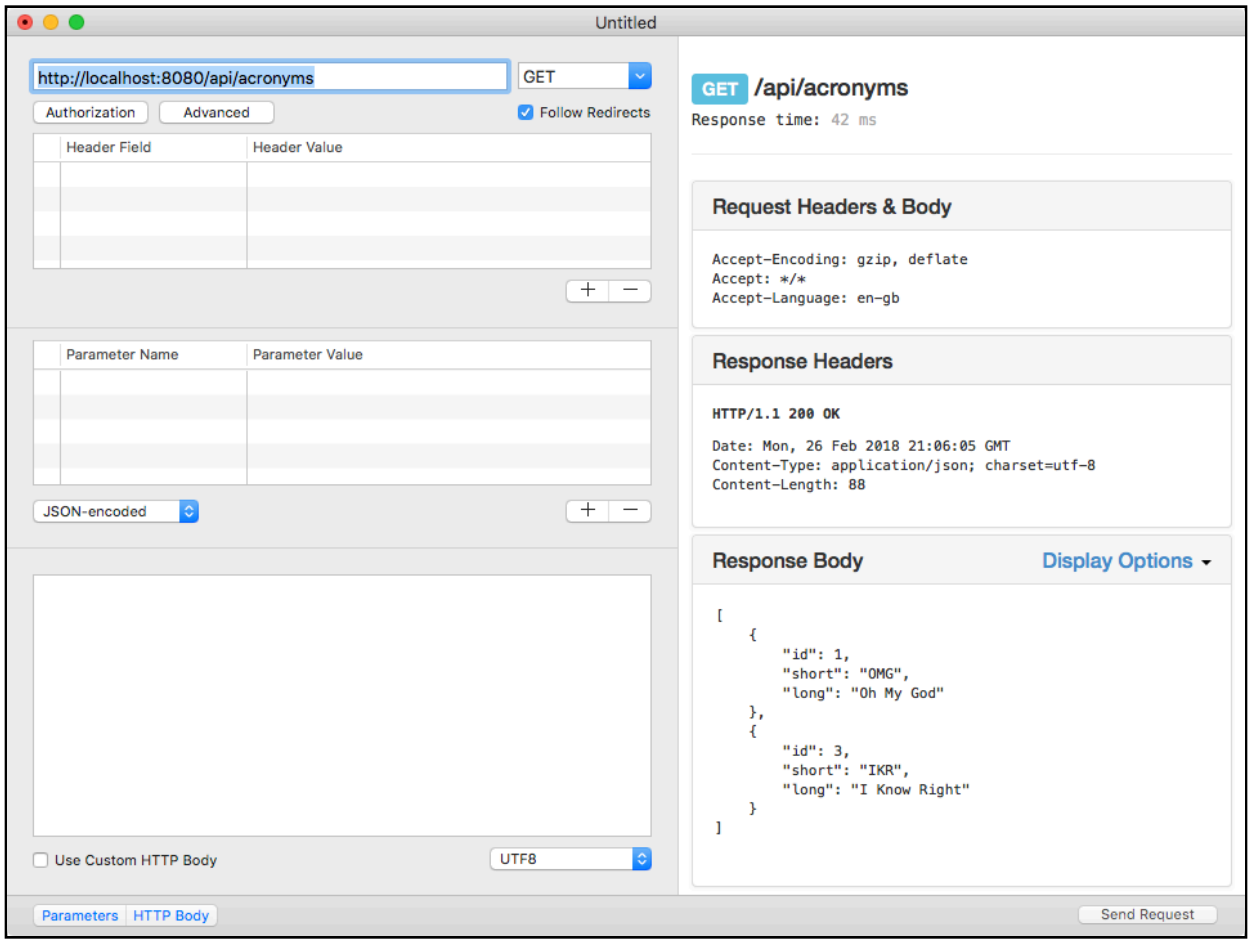
这是它的作用：

1. 创建一个新的AcronymsController实例。
2. 向路由器注册新类型，以确保控制器的路由得到注册。

构建并运行应用程序，然后在RESTed中创建新请求。配置请求如下：

- **URL**: http://localhost:8080/api/acronyms/
- **method**: GET

发送请求，您将获得数据库中现有的缩略词：



Route groups

在前面章节中为缩略词创建的所有REST路由都使用相同的初始路径，例如：

```
router.get(
  "api",
  "acronyms",
  Acronym.parameter
) { req -> Future<Acronym> in
  return try req.parameters.next(Acronym.self)
}
```

如果需要更改/**api/acronyms/**路径，则必须在多个位置更改路径。如果添加新路由，则必须记住添加路径的两个部分。 Vapor提供路由组(**route groups**)以简化此操作。打开**AcronymsController.swift**并在boot(router:)开头创建一个路由组：

```
let acronymsRoutes = router.grouped("api", "acronyms")
```

这将为路径 **/api/acronyms** 创建一个新的路由组。接下来，替换：

```
router.get("api", "acronyms", use: getAllHandler)
```

用以下内容：

```
acronymsRoutes.get(use: getAllHandler)
```

这样可以像以前一样工作，但大大简化了代码，使维护更容易。

接下来，打开**routes.swift**并删除剩余的缩略词路由：

- `router.post("api", "acronyms")` route handler.
- `router.get("api", "acronyms", Acronym.parameter)`
- `router.put("api", "acronyms", Acronym.parameter)`
- `router.delete("api", "acronyms", Acronym.parameter)`
- `router.get("api", "acronyms", "search")`
- `router.get("api", "acronyms", "first")`
- `router.get("api", "acronyms", "sorted")`

然后从模板中删除任何其他路由。 您应该只在routes(_:)中留下AcronymsController的注册。接下来，打开**AcronymsController.swift**并在boot(router:)末尾添加以下每一项来重新创建路由处理程序：

```
func createHandler(_ req: Request) throws -> Future<Acronym> {
  return try req
    .content
    .decode(Acronym.self)
    .flatMap(to: Acronym.self) { acronym in
    return acronym.save(on: req)
  }
}

func getHandler(_ req: Request) throws -> Future<Acronym> {
  return try req.parameters.next(Acronym.self)
}
```

```
func updateHandler(_ req: Request) throws -> Future<Acronym> {
  return try flatMap(
    to: Acronym.self,
    req.parameters.next(Acronym.self),
    req.content.decode(Acronym.self)
  ) { acronym, updatedAcronym in
    acronym.short = updatedAcronym.short
    acronym.long = updatedAcronym.long
    return acronym.save(on: req)
  }
}

func deleteHandler(_ req: Request)
  throws -> Future<HTTPStatus> {
  return try req
    .parameters
    .next(Acronym.self)
    .delete(on: req)
    .transform(to: .noContent)
}

func searchHandler(_ req: Request) throws -> Future<[Acronym]> {
  guard let searchTerm = req
    .query[String.self, at: "term"] else {
    throw Abort(.badRequest)
  }
  return Acronym.query(on: req).group(.or) { or in
    or.filter(\.short == searchTerm)
    or.filter(\.long == searchTerm)
  }.all()
}

func getFirstHandler(_ req: Request) throws -> Future<Acronym> {
  return Acronym.query(on: req)
    .first()
    .unwrap(or: Abort(.notFound))
}

func sortedHandler(_ req: Request) throws -> Future<[Acronym]> {
  return Acronym.query(on: req).sort(\.short, .ascending).all()
}
```

这些处理程序中的每一个都与您在第7章中创建的处理程序完全相同。 如果你需要他们做了什么的提醒，那就是你要留意的地方！

最后，使用路由组注册这些路由处理程序。将以下内容添加到boot(router:)底部：

```
// 1
acronymsRoutes.post(use: createHandler)
// 2
acronymsRoutes.get(Acronym.parameter, use: getHandler)
// 3
acronymsRoutes.put(Acronym.parameter, use: updateHandler)
// 4
acronymsRoutes.delete(Acronym.parameter, use: deleteHandler)
```

```
// 5
acronymsRoutes.get("search", use: searchHandler)
// 6
acronymsRoutes.get("first", use: getFirstHandler)
// 7
acronymsRoutes.get("sorted", use: sortedHandler)
```

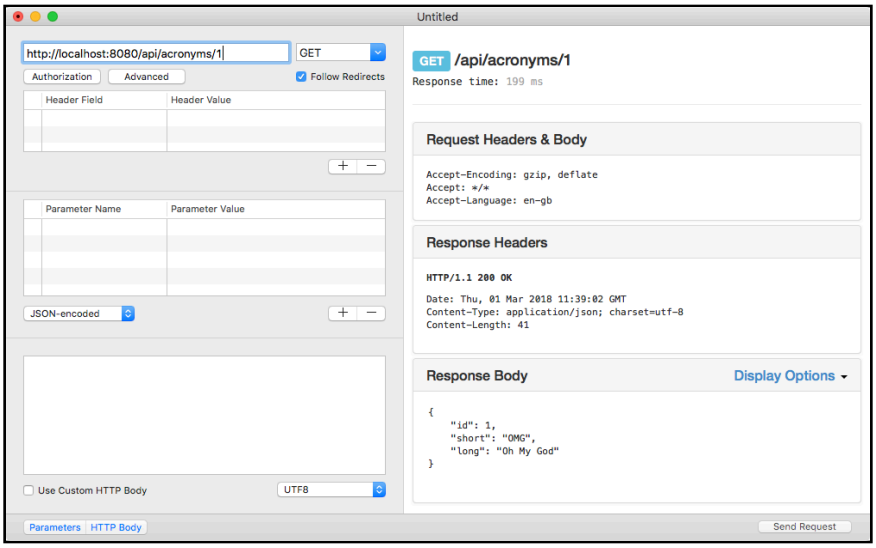
这是它的作用：

1. 注册createHandler(_:)以处理对 **/api/acronyms**的POST请求。
2. 注册getHandler(_:)以处理对 **/api/acronyms/<ACRONYM ID>**的GET请求。
3. 注册updateHandler(_:)以处理对 **/api/acronyms/<ACRONYM ID>**的PUT请求。
4. 注册deleteHandler(_:)以处理对 **/api/acronyms/<ACRONYM ID>**的 DELETE请求。
5. 注册searchHandler(_:)以处理对 **/api/acronyms/search**的GET请求。
6. 注册getFirstHandler(_:)以处理对 **/api/acronyms/first**的GET请求。
7. 注册sortedHandler(_:)以处理对 **/api/acronyms/sorted**的GET请求。

构建并运行应用程序，然后在RESTed中创建新请求。配置请求如下：

- **URL**: http://localhost:8080/api/acronyms/1
- **method**: GET

发送请求，您将看到使用新控制器响应的先前创建的缩略词：



接受POST数据

如第2章“Hello Vapor！”中所述，Vapor为PUT，POST和PATCH路由解码传入数据提供了辅助方法。这有助于删除一层嵌套。要使用此辅助方法，请打开**AcronymsController.swift**并替换createHandler(_:)用以下内容：

```
func createHandler(
  _ req: Request,
  acronym: Acronym
) throws -> Future<Acronym> {
  return acronym.save(on: req)
}
```

函数原型现在有一个Acronym作为参数。这是请求解码的缩略词，因此您不必自己解码数据。在boot(router:)里替换：

```
acronymsRoutes.post(use: createHandler)
```

用下面内容：

```
acronymsRoutes.post(Acronym.self, use: createHandler)
```

该辅助方法将类型作为第一个参数去解码。如果需要，您可以在use：参数之前提供任何路径组成部分。

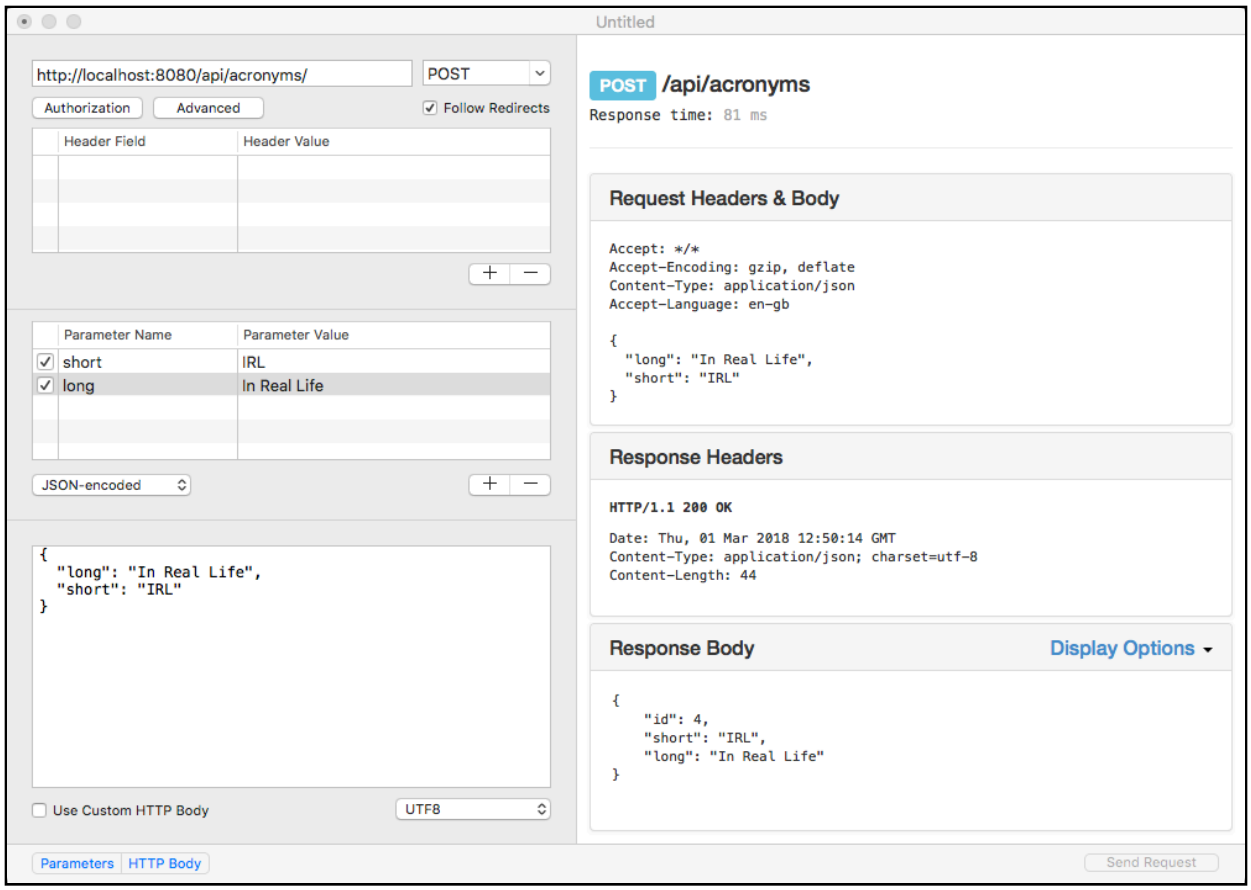
构建并运行应用程序，然后在RESTed中创建新请求。配置请求如下：

- **URL**: http://localhost:8080/api/acronyms/
- **method**: POST
- **Parameter encoding**: JSON-encoded

添加两个带有名称和值的参数：

- **short**: IRL
- **long**: In Real Life

发送请求，您将看到包含创建的缩略词的响应：



Fluent为Future<Model>类型提供了save(on:)，create(on:)，update(on:)和delete(on:)的便利方法。例如，如果您在保存模型之前不需要操作它，save(on:)就非常有用。

然后去哪儿?

本章介绍了控制器作为更好地组织代码的方法。它们有助于将路由处理程序划分为不同的责任区域。这允许应用程序以可维护的方式增长。接下来的章节将介绍如何将不同模型与Fluent中的关系结合起来。