Chapter 5: Fluent & Persisting Models

By Tim Condon

在第2章“Hello，Vapor！”中，您学习了创建Vapor应用程序的基础知识，包括如何创建路由。本章介绍如何使用Fluent在Vapor应用中保存数据。

Fluent

Fluent是Vapor的ORM或**object relational mapping**工具。它是Vapor应用程序和数据库之间的抽象层，旨在简化数据库的工作。使用诸如Fluent之类的ORM具有许多好处。

最大的好处是您不必直接使用数据库！直接与数据库交互时，将数据库查询编写为字符串。这些不是类型安全的，从Swift中使用可能会很痛苦。

Fluent给您的益处是即使在同一个应用程序中，您也可以使用任何数据库引擎。最后，您不需要知道如何编写查询，因为您可以以“Swifty”方式与您的**models**交互。

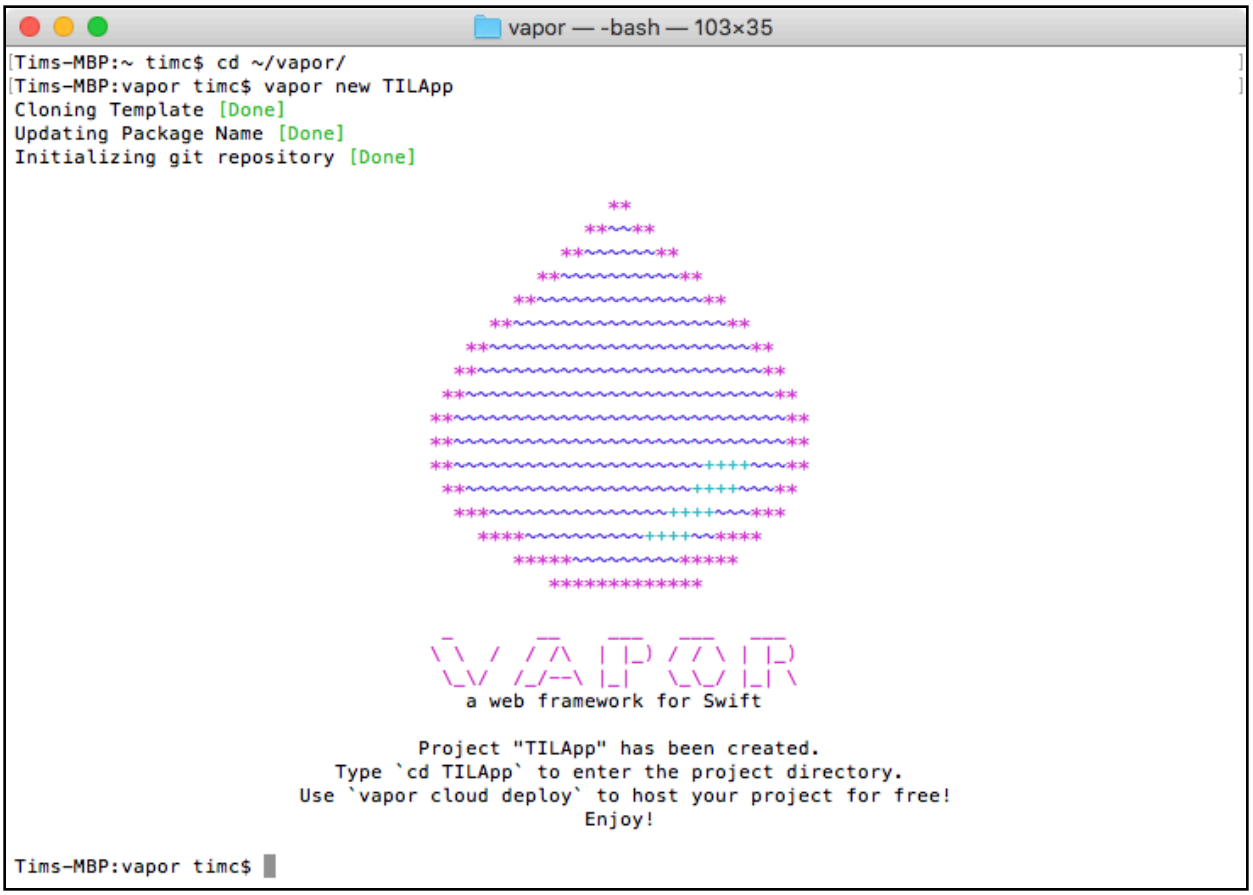
Models是数据的Swift表示，并在整个Fluent中使用。**Models**是您在数据库中保存和访问的对象，例如用户资料信息。 Fluent在与数据库交互时返回并使用类型安全的models，为您提供编译时安全性。

缩略词(Acronyms)

在接下来的几章中，您将构建一个复杂的“Today I Learned”应用程序，它可以保存不同的缩略词及其含义。首先使用Vapor Toolbox创建一个新项目。在终端中，输入以下命令：

```
cd ~/vapor
vapor new TILApp
```

第一个命令将您带入主目录中名为**vapor**的目录，并假定您已完成第2章“Hello Vapor！”中的步骤。第二个命令使用默认模板创建一个名为**TILApp**的新Vapor 3项目。



该模板提供models和controllers的示例文件。你将构建自己的，所以删除示例。在终端中，输入：

```
cd TILApp
rm -rf Sources/App/Models/*
rm -rf Sources/App/Controllers/*
```

由于Xcode项目在使用Vapor时是可丢弃的 - 它们完全是可选的 - 最佳做法是在Xcode之外创建项目文件。这使得Vapor Toolbox使用的Swift Package Manager可以确保它们链接到正确的目标。
创建一个文件来保存Acronym模型：

```
touch Sources/App/Models/Acronym.swift
```

此命令在**App**模块的**Models**目录中创建一个名为**Acronym.swift**的Swift文件。现在生成您的Xcode项目：

```
vapor xcode -y
```

打开**configure.swift**，找到**Configure migrations**组并删除以下行：

```
migrations.add(model: Todo.self, database: .sqlite)
```

接下来，打开**routes.swift**并删除以下行：

```
// Example of configuring a controller
let todoController = TodoController()
router.get("todos", use: todoController.index)
router.post("todos", use: todoController.create)
router.delete("todos", Todo.parameter,
              use: todoController.delete)
```

这将删除对模板的示例model和controller的剩余引用。

打开**Acronym.swift**并添加以下内容以创建缩略词的基本模型：

```
import Vapor
import FluentSQLite

final class Acronym: Codable {
  var id: Int?
  var short: String
  var long: String

  init(short: String, long: String) {
    self.short = short
    self.long = long
  }
}
```

该模型包含两个String属性来保存缩略词及其定义。它还包含一个可选的id属性，用于存储模型的ID（如果已设置）。

所有Fluent模型必须遵循Codable协议。在可能的情况下将类标记为final也是一种很好的做法，因为它提供了性能优势。保存缩略词时，ID由数据库设置。

接下来使Acronym遵循Fluent的Model协议。在文件末尾添加以下内容：

```
extension Acronym: Model {
  // 1
  typealias Database = SQLiteDatabase
  // 2
  typealias ID = Int
  // 3
  public static var idKey: IDKey = \Acronym.id
}
```

这是它的作用：

1. 告诉Fluent用于此模型的数据库。该模板已配置为使用SQLite。
2. 告诉Fluent ID的类型。
3. 告诉Fluent模型ID属性的key path。

使用SQLiteModel可以进一步改进此代码。更换：

```
extension Acronym: Model {
  typealias Database = SQLiteDatabase
  typealias ID = Int
  public static var idKey: IDKey = \Acronym.id
}
```

用以下内容：

```
extension Acronym: SQLiteModel {}
```

Fluent为每个数据库provider提供Model帮助协议，因此您不必指定数据库或ID类型或key。 SQLiteModel协议必须具有名称为id的可选Int类型的ID，但对于ID为UUID或String的模型，有SQLiteUUIDModel和SQLiteStringModel协议。如果要自定义ID属性名称，则必须遵循标准Model协议。

要将模型保存在数据库中，必须为其创建表。 Fluent通过**migration**完成此操作。Migrations允许您对数据库进行可靠，可测试，可重现的更改。它们通常用于为模型创建**database schema**或表描述。它们还用于将数据输入数据库中，或在模型保存后对其进行更改。

在**Acronym.swift**的末尾添加以下内容，使模型遵循Migration协议：

```
extension Acronym: Migration {}
```

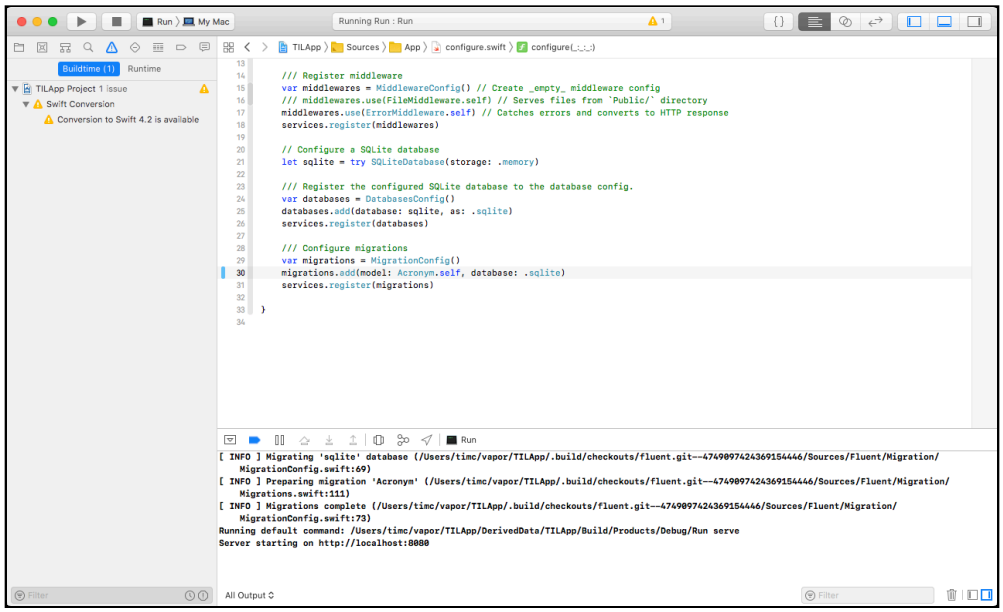
这就是你需要做的一切！通过Codable，Fluent可以为您的模型推断出架构。对于基本模型，您可以使用Migration的默认实现。如果您以后需要更改模型或执行更复杂的操作（例如将属性标记为唯一），则可能需要实现自己的migrations。您将在后面的章节中了解更多信息。

现在，Acronym遵循Migration协议，您可以告诉Fluent在应用程序启动时创建表。打开**configure.swift**并找到标记为 // Configure migrations的部分。在services.register(migrations)之前添加以下内容：

```
migrations.add(model: Acronym.self, database: .sqlite)
```

Fluent支持在单个应用程序中混合多个数据库，因此您可以指定保存每个模型的数据库。Migrations只运行一次；一旦他们在数据库中运行，他们就再也不会被执行了。记住这一点非常重要，因为如果更改模型，Fluent将不会尝试重新创建表。

将active scheme设置为以**My Mac**作为**Run**目标。构建并运行。检查控制台并查看migrations是否已运行。您应该看到类似于下面的控制台输出的内容：



保存 models

当您的应用程序用户输入新的缩略词时，您需要一种方法来保存它。在Swift 4和 Vapor 3中，Codable使这个变得微不足道。 Vapor提供了Content，一种以Codable为基础的包装类，允许您在各种格式之间转换模型和其他数据。这在Vapor中广泛使用，你会在整本书中看到它。

打开**Acronym.swift**并将以下内容添加到文件末尾，以使Acronym遵循Content协议：

```
extension Acronym: Content {}
```

由于Acronym已遵循Codable协议，因此您无需添加任何其他内容。要创建缩略词，用户的浏览器会发送一个POST请求，其中包含类似于以下内容的JSON：

```
{
  "short": "OMG",
  "long": "Oh My God"
}
```

您需要一个路由来处理POS请求并保存新的缩略词。打开**routes.swift**并将以下内容添加到routes(_:)末尾：

```
// 1
router.post("api", "acronyms") { req -> Future<Acronym> in
  // 2
  return try req.content.decode(Acronym.self)
    .flatMap(to: Acronym.self) { acronym in
    // 3
    return acronym.save(on: req)
  }
}
```

这是它的作用：

1. 在**/api/acronyms**注册一条接受POST请求并返回Future<Acronym>的新路由。它会在保存后返回缩略词。

2. 使用Codable将请求的JSON解码为Acronym模型。 这将返回Future<Acronym>，因此它使用flatMap(to:)在解码完成时提取acronym。请注意， 这与第2章“Hello Vapor！”中解码数据的方式不同。 在这个路由处理程序中， 您自己在请求上调用decode(_:)。然后，您才解包解码结果， 因为decode(_:)返回了Future <Acronym>。

3. 使用Fluent保存模型。这会返回Future <Acronym>，因为它会在保存后返回模型。

Fluent和Vapor集成使用Codable使这一点变得简单。由于Acronym遵循Content协议，因此可以在JSON和Model之间轻松转换。这允许Vapor在响应中可以毫不费力的将模型作为JSON返回。构建并运行应用程序以试用它。一个很好的测试工具是**RESTed**，可以从Mac App Store免费下载。其他工具如Paw和Postman也适用。

在RESTed中，按如下方式配置请求：

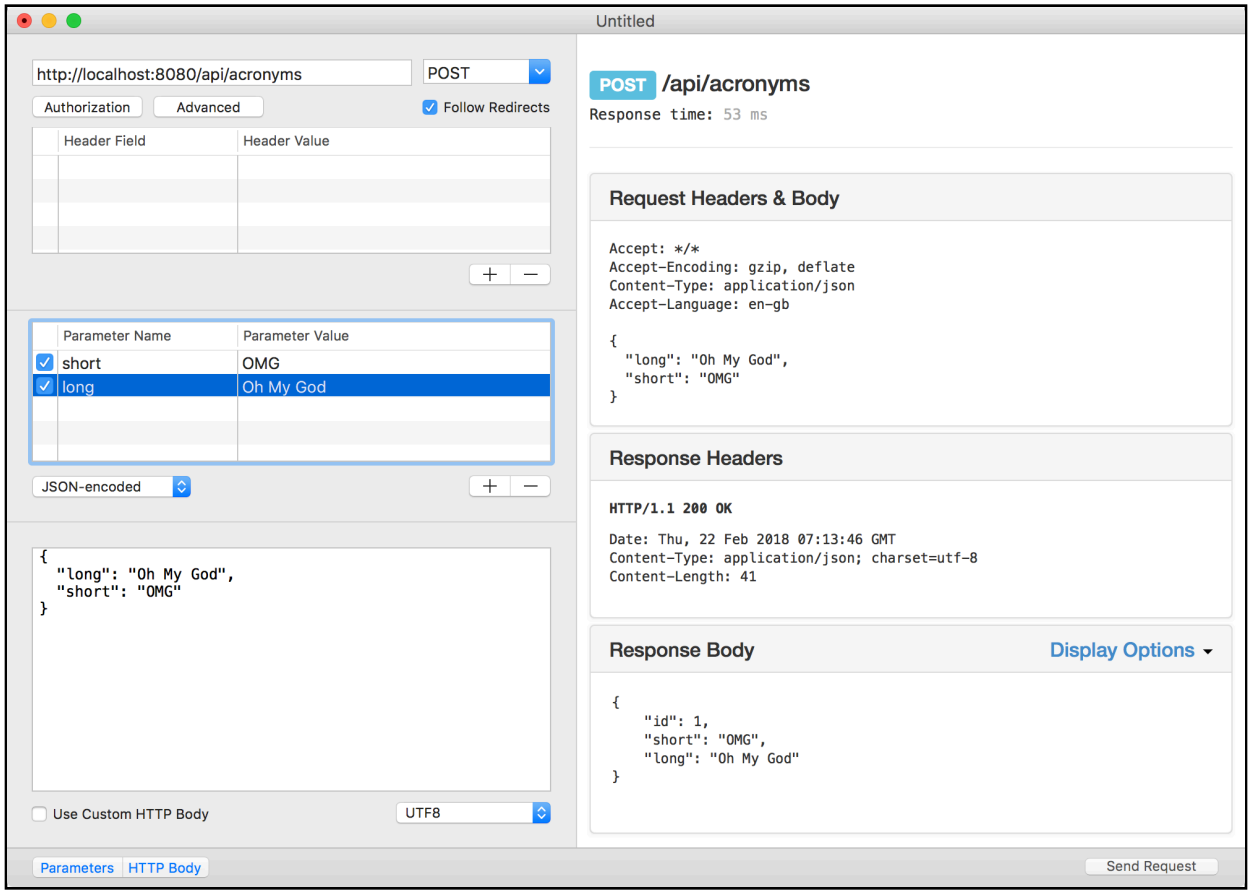
- **URL**: http://localhost:8080/api/acronyms
- **method**: POST
- **Parameter encoding**: JSON-encoded

添加两个带有名称和值的参数：

- **short**: OMG
- **long**: Oh My God

将参数编码设置为**JSON-encoded**可确保将数据作为JSON发送。重要的是要注意这也将Content-Type标头设置为application/json，它告诉Vapor请求包含JSON。如果您使用其他客户端发送请求，您可能需要手动设置。

单击“**Send Request**”，您将看到响应中提供的缩略词。id字段将具有一个值，因为它现在已保存在数据库中：



然后去哪儿?

本章向您介绍了Fluent，以及如何在Vapor中创建模型并将其保存在数据库中。接下来的章节将基于此应用程序构建，以创建功能齐全的TIL应用程序。