Chapter 10: Sibling Relationships

By Tim Condon

在第9章“Parent Child Relationships”中，您学习了如何使用Fluent构建模型之间的父子关系。本章介绍如何实现其他类型的关系：兄弟关系。您将学习如何在Vapor中对它们进行建模以及如何在路由中使用它们。

> 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。

兄弟关系

兄弟关系描述了将两个模型相互链接的关系。它们也被称为多对多关系。与父子关系不同，兄弟关系中的模型之间没有约束。

例如，如果您对宠物和玩具之间的关系进行建模，则宠物可以拥有一个或多个玩具，一个玩具可以被一个或多个宠物使用。在TIL应用程序中，您将能够对缩略词进行分类。缩略词可以是一个或多个类别的一部分，类别可以包含一个或多个首字母缩略词。

创建类别

为Category和CategoriesController创建新文件。在终端中，键入：

```
# 1
cd ~/vapor/TILApp
# 2
touch Sources/App/Models/Category.swift
# 3
touch Sources/App/Controllers/CategoriesController.swift
# 4
vapor xcode -y
```

这是它的作用：

1. 切换到TILApp的项目目录。
2. 创建一个新文件**Category.swift**。
3. 创建一个新文件**CategoriesController.swift**。
4. 重新生成Xcode项目并打开它。

类别模型

在Xcode中，打开**Category.swift**并为类别创建基本模型：

```
import Vapor
import FluentPostgreSQL

final class Category: Codable {
  var id: Int?
  var name: String

  init(name: String) {
    self.name = name
  }
}
```

该模型包含一个String属性来保存类别的名称。该模型还包含一个可选的id属性，用于存储模型设置时的ID。通过在类下面添加以下扩展，使Category模型遵循Fluent的PostgreSQLModel、Content、Migration和Parameter协议：

```
extension Category: PostgreSQLModel {}
extension Category: Content {}
extension Category: Migration {}
extension Category: Parameter {}
```

最后，打开**configure.swift**并在migration.add(model: Acronym.self, database: .psql)之后将Category模型添加到migration列表：

```
migrations.add(model: Category.self, database: .psql)
```

这会将新模型添加到MigrationConfig，以便Fluent在下一个应用程序启动时在数据库中创建表。

类别控制器

打开**CategoriesController.swift**并创建一个新的控制器，用以创建和检索类别：

```
import Vapor

// 1
struct CategoriesController: RouteCollection {
  // 2
  func boot(router: Router) throws {
    // 3
    let categoriesRoute = router.grouped("api", "categories")
    // 4
    categoriesRoute.post(Category.self, use: createHandler)
    categoriesRoute.get(use: getAllHandler)
    categoriesRoute.get(Category.parameter, use: getHandler)
  }

  // 5
  func createHandler(
    _ req: Request,
    category: Category
  ) throws -> Future<Category> {
    // 6
    return category.save(on: req)
  }

  // 7
  func getAllHandler(
    _ req: Request
  ) throws -> Future<[Category]> {
    // 8
    return Category.query(on: req).all()
  }

  // 9
  func getHandler(_ req: Request) throws -> Future<Category> {
    // 10
    return try req.parameters.next(Category.self)
  }
}
```

这是控制器的作用：

1. 定义遵循RouteCollection协议的新CategoriesController类型。
2. 根据RouteCollection的要求实现boot(router:)。这是您注册路由处理程序的地方。
3. 为路径 **/api/categories**创建新的路由组。
4. 将路由处理程序注册到其路由。
5. 定义创建类别的createHandler(_:category:)。
6. 保存请求中的已解码类别。
7. 定义返回所有类别的getAllHandler(_:)。
8. 执行Fluent查询以从数据库中检索所有类别。
9. 定义返回单个类别的getHandler(_:)。
10. 返回从请求的parameters中提取的类别。

最后，打开**routes.swift**并通过在routes(_:)末尾添加以下内容来注册控制器：

```
// 1
let categoriesController = CategoriesController()
// 2
try router.register(collection: categoriesController)
```

这是它的作用：

1. 创建CategoriesController实例。
2. 向路由器注册新实例以连接路由。

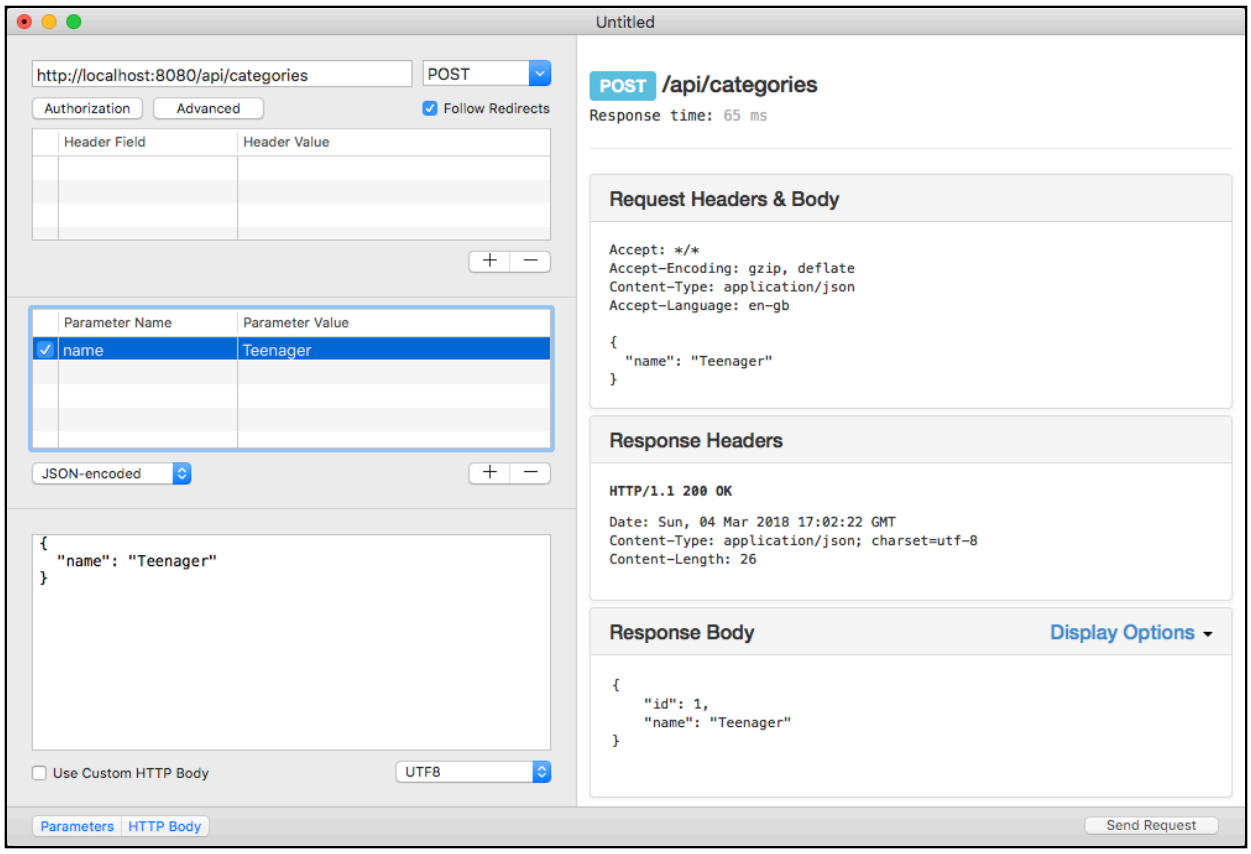
构建并运行应用程序，然后在RESTed中创建新请求。配置请求如下：

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

一个名称和值的参数：

- **name**: Teenager

发送请求，您将在响应中看到已保存的类别：



创建Pivot

在第9章“Parent Child Relationships”中，您在缩略词中添加了对用户的引用，以建立缩略词和用户之间的关系。但是，您不能对此类兄弟关系建模，因为查询效率太低。如果您在类别中有一组缩略词，要搜索某个缩略词所有类别，您必须检查每个类别。如果您在缩略词中有一组类别，要搜索某个类别中的所有缩略词，您必须检查每个缩略词。你需要一个单独的模型来保持这种关系。在Fluent中，这就是一个中间表（**pivot**）。

pivot是Fluent中包含关系的另一种模型类型。在终端中，创建此新模型文件：

```
touch Sources/App/Models/AcronymCategoryPivot.swift
vapor xcode -y
```

AcronymCategoryPivot.swift将包含用于管理兄弟关系的pivot模型。

打开**AcronymCategoryPivot.swift**并添加以下内容以创建pivot：

```
import FluentPostgreSQL
import Foundation

// 1
final class AcronymCategoryPivot: PostgreSQLUUIDPivot {
  // 2
  var id: UUID?
  // 3
  var acronymID: Acronym.ID
  var categoryID: Category.ID

  // 4
  typealias Left = Acronym
  typealias Right = Category
  // 5
  static let leftIDKey: LeftIDKey = \.acronymID
  static let rightIDKey: RightIDKey = \.categoryID

  // 6
  init(_ acronym: Acronym, _ category: Category) throws {
    self.acronymID = try acronym.requireID()
    self.categoryID = try category.requireID()
  }
}

// 7
extension AcronymCategoryPivot: Migration {}
extension AcronymCategoryPivot: ModifiablePivot {}
```

这是这个模型的作用：

1. 定义遵循PostgreSQLUUIDPivot协议的新对象AcronymCategoryPivot。这是一个基于Fluent 的Pivot协议的辅助协议。

2. 定义模型的id。请注意，这是一个UUID类型，因此您必须在文件中导入Foundation模块。

3. 定义两个属性以链接到Acronym和Category的ID。这就是维持这种关系的原因。

4. 定义Pivot所需的Left和Right类型。这告诉Fluent此关系中的两个模型是什么。

5. 告诉Fluent此关系每一侧的两个ID属性的key path。

6. 根据ModifiablePivot的要求实现可抛出异常的初始化函数。

7. 遵循Migration协议以便Fluent可以设置表。

8. 遵循ModifiablePivot协议。这允许您使用Vapor提供的语法糖添加和删除关系。

最后，打开**configure.swift**，并在migrations.add(model: Category.self, database: .psql)之后将AcronymCategoryPivot模型添加到migration列表中：

```
migrations.add(
  model: AcronymCategoryPivot.self,
  database: .psql)
```

这会将新的pivot模型添加到MigrationConfig，以便Fluent在下一个应用程序启动时准备好数据库中的表。

要实际创建两个模型之间的关系，您需要使用pivot。 Fluent提供了创建和删除关系的便利方法。首先，打开**Acronym.swift**并在包含user计算属性的扩展中添加新的计算属性：

```
// 1
var categories: Siblings<Acronym,
                         Category,
                         AcronymCategoryPivot> {
  // 2
  return siblings()
}
```

这是它的作用：

1. 向Acronym添加计算属性以获取Acronym的类别。这将返回Fluent的Sibling泛型类型。它返回使用AcronymCategoryPivot保存的、属于Category类型的Acronym的兄弟类。

2. 使用Fluent的siblings()函数检索所有类别。Fluent处理其他一切。

打开**AcronymsController.swift**并在getUserHandler(_:)下面添加以下路由处理程序用于设置缩略词和类别之间的关系：

```
// 1
func addCategoriesHandler(
  _ req: Request
) throws -> Future<HTTPStatus> {
  // 2
  return try flatMap(
    to: HTTPStatus.self,
    req.parameters.next(Acronym.self),
    req.parameters.next(Category.self)) { acronym, category in
      // 3
      return acronym.categories
        .attach(category, on: req)
```

```
            .transform(to: .created)
    }
}
```

这是路由处理程序的作用：

1. 定义一个返回Future<HTTPStatus>的新路由处理程序addCategoriesHandler(_:)。

2. 使用flatMap(to:_:_:)从请求的parameters中提取缩略词和类别。

3. 使用attach(_:on:)来设置缩略词和类别之间的关系。这将创建一个pivot模型并将其保存在数据库中。将结果转换为**201 Created**作为响应。

在boot(router:)底部注册此路由处理程序：

```
acronymsRoutes.post(
  Acronym.parameter,
  "categories",
  Category.parameter,
  use: addCategoriesHandler)
```

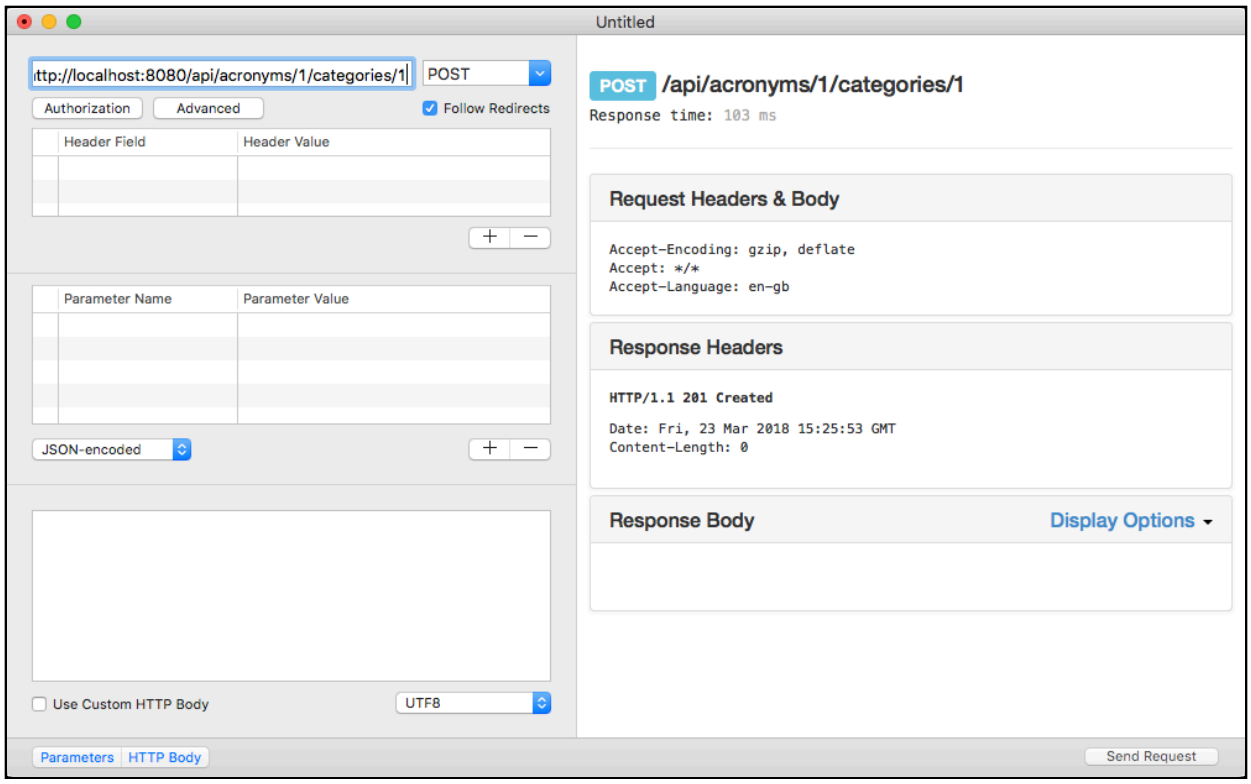
这会将**/api/acronyms/<ACRONYM_ID>/categories/<CATEGORY_ID>**的HTTP POST请求路由到addCategoriesHandler(_:)。

构建并运行应用程序并启动RESTed。如果数据库中没有任何缩略词，请立即创建一个。然后，创建一个配置如下的新请求：

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

这将在ID为1的缩略词和ID为1的类别之间创建一个兄弟关系，这是您在本章前面创建的。

单击发送请求，您将看到**201 Created**的响应：



查询关系

缩略语和类别现在是兄弟关系相关联。 但是如果您看不到这些关系， 这就不太有用了！ Fluent提供了一些方法以允许您查询这些关系。在上面您已经使用了一个来创建关系。

缩略词的类别

打开**AcronymsController.swift**并在addCategoriesHandler(:_)之后添加一个新的路由处理程序：

```
// 1
func getCategoriesHandler(
  _ req: Request
) throws -> Future<[Category]> {
  // 2
  return try req.parameters.next(Acronym.self)
    .flatMap(to: [Category].self) { acronym in
      // 3
      try acronym.categories.query(on: req).all()
  }
}
```

这是它的作用：

1. 定义返回Future<[Category]>的路由处理程序getCategoriesHandler(_:)。
2. 从请求的parameters中提取缩略词并解包返回的future。
3. 使用新的计算属性来获取类别。然后使用Fluent查询返回所有类别。

在boot(router:)底部注册此路由处理程序

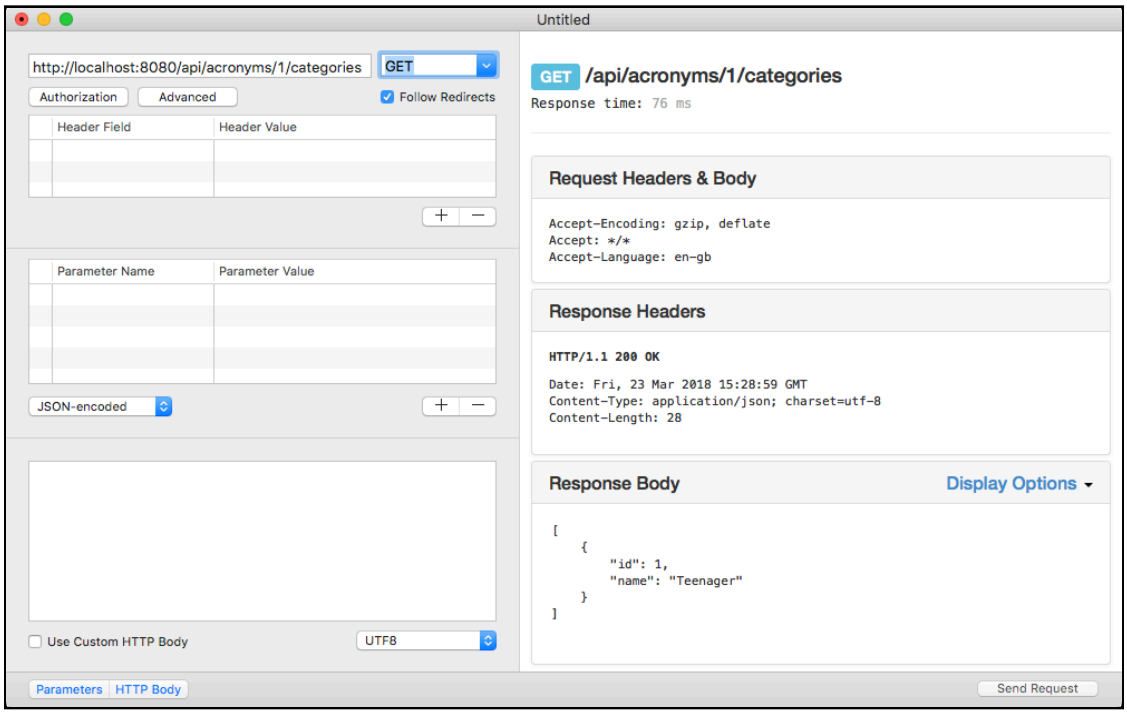
```
acronymsRoutes.get(
  Acronym.parameter,
  "categories",
  use: getCategoriesHandler)
```

这会将**/api/acronyms/<ACRONYM_ID>/categories**的HTTP GET请求路由到getCategoriesHandler(:_)。

构建并运行应用程序并启动RESTed。使用以下属性创建请求：

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

发送请求，您将收到缩略词的类别数组：



类别的缩略词

打开**Category.swift**并在文件底部添加一个扩展以获取该类别的缩略词：

```
extension Category {
  // 1
  var acronyms: Siblings<Category,
                         Acronym,
                         AcronymCategoryPivot> {
    // 2
    return siblings()
  }
}
```

这是它的作用：

1. 向Category添加计算属性以获取其缩略词。 这将返回Fluent的Sibling泛型类型。它返回使用AcronymCategoryPivot保存的、属于Acronym类型的Category的兄弟类。

2. 使用Fluent的siblings()函数检索所有缩略词。Fluent处理其他一切。

打开**CategoriesController.swift**并在getHandler(_:)之后添加一个新的路由处理程序：

```
// 1
func getAcronymsHandler(
  _ req: Request) throws -> Future<[Acronym]> {
  // 2
  return try req.parameters.next(Category.self)
    .flatMap(to: [Acronym].self) { category in
      // 3
      try category.acronyms.query(on: req).all()
  }
}
```

这是它的作用：

1. 定义一个返回Future<[Acronym]>的新的路由处理程序getAcronymsHandler(_:)。

2. 从请求的parameters中提取类别并解包返回的future。

3. 使用新的计算属性来获取缩略词。然后，使用Fluent查询返回所有缩略词。

在boot(router:)底部注册此路由处理程序：

```
categoriesRoute.get(
  Category.parameter,
```

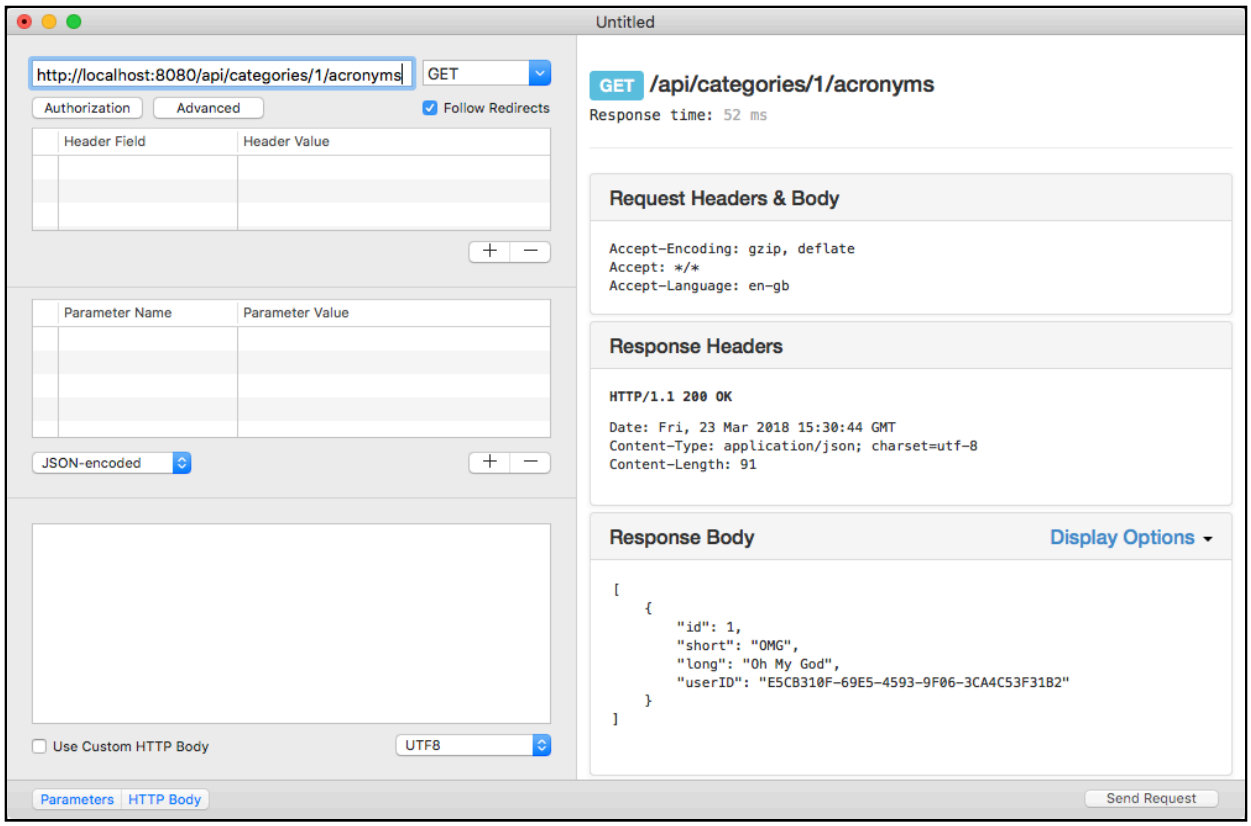
```
    "acronyms",
    use: getAcronymsHandler)
```

这会将**/api/categories/<CATEGORY_ID>/acronyms**的HTTP GET请求路由到getAcronymsHandler(_:)。

构建并运行应用程序并启动RESTed。创建一个请求如下：

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

发送请求，您将收到该类别中的一系列缩略词：



删除关系

删除缩略词和类别之间的关系与添加关系非常相似。打开**AcronymsController.swift**并在getCategoriesHandler(req:)下面添加以下内容：

```
// 1
func removeCategoriesHandler(
  _ req: Request) throws -> Future<HTTPStatus> {
```

```
    // 2
    return try flatMap(
      to: HTTPStatus.self,
      req.parameters.next(Acronym.self),
      req.parameters.next(Category.self)
    ) { acronym, category in
        // 3
        return acronym.categories
          .detach(category, on: req)
          .transform(to: .noContent)
  }
}
```

这是新路由处理程序的作用：

1. 定义一个新的返回Future <HTTPStatus>的路由处理程序removeCategoriesHandler(_:)。

2. 使用flatMap(to:_:_:)从请求的parameters中提取缩略词和类别。

3. 使用detach(_:on:)删除缩略词和类别之间的关系。这将在数据库中找到pivot模型并将其删除。将结果转换为**204 No Content**以做响应。

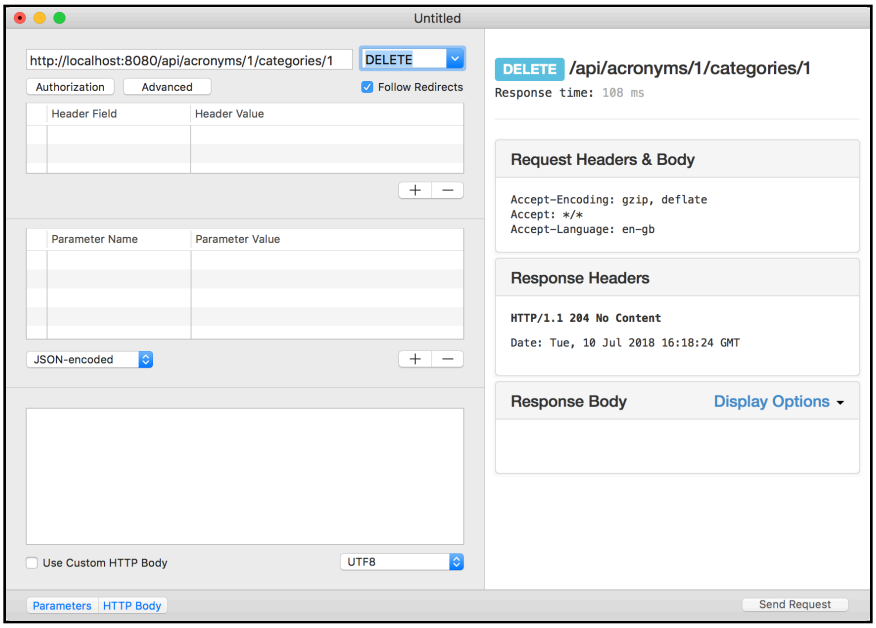
最后，在boot(router:)底部注册路由：

```
acronymsRoutes.delete(
  Acronym.parameter,
  "categories",
  Category.parameter,
  use: removeCategoriesHandler)
```

这会将**/api/acronyms/<ACRONYM_ID>/categories/<CATEGORY_ID>**的HTTP DELETE请求路由到removeCategoriesHandler(_:)。构建并运行应用程序并启动RESTed。使用以下属性创建请求：

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

发送请求，您将收到**204 No Content**的响应：



如果您再次发送请求以获取缩略词的类别，您将收到一个空数组。

外键约束

与第9章“Parent Child Relationships”一样，最好将外键约束与兄弟关系一起使用。当前的AcronymCategoryPivot不会检查缩略词和类别的ID。此时，您可以删除仍由pivot链接的缩略词和类别，并且关系将保留，而不会标记错误。

打开**AcronymCategoryPivot.swift**并替换当前的Migration遵循扩展。在新migration中，将外键约束添加到pivot：

```
// 1
extension AcronymCategoryPivot: Migration {
  // 2
  static func prepare(
    on connection: PostgreSQLConnection
  ) -> Future<Void> {
    // 3
    return Database.create(self, on: connection) { builder in
      // 4
      try addProperties(to: builder)
      // 5
      builder.reference(
        from: \.acronymID,
```

```
          to: \Acronym.id,
          onDelete: .cascade)
        // 6
        builder.reference(
          from: \.categoryID,
          to: \Category.id,
          onDelete: .cascade)
    }
  }
}
```

以下是新migration的作用：

1. 使AcronymCategoryPivot遵循Migration协议。
2. 按照Migration的定义实现prepare(on:)。这会覆盖默认实现。
3. 在数据库中为AcronymCategoryPivot创建表。
4. 使用addProperties(to:)将所有字段添加到数据库。
5. 在AcronymCategoryPivot的acronymID属性和Acronym的id属性之间添加引用。这将设置外键约束。删除缩略词时，.cascade设置级联模式引用操作。这意味着将自动删除关系，而不是抛出错误。
6. 在AcronymCategoryPivot的categoryID属性和Category的id属性之间添加引用。这将设置外键约束。删除类别时，还要设置要删除的模式引用操作。

在Xcode中停止应用程序。由于migration已更改，因此您需要重置数据库，以便Fluent运行新的migration。

在终端中，键入：

```
# 1
docker stop postgres
# 2
docker rm postgres
# 3
docker run --name postgres -e POSTGRES_DB=vapor \
  -e POSTGRES_USER=vapor -e POSTGRES_PASSWORD=password \
  -p 5432:5432 -d postgres
```

这是它的作用：

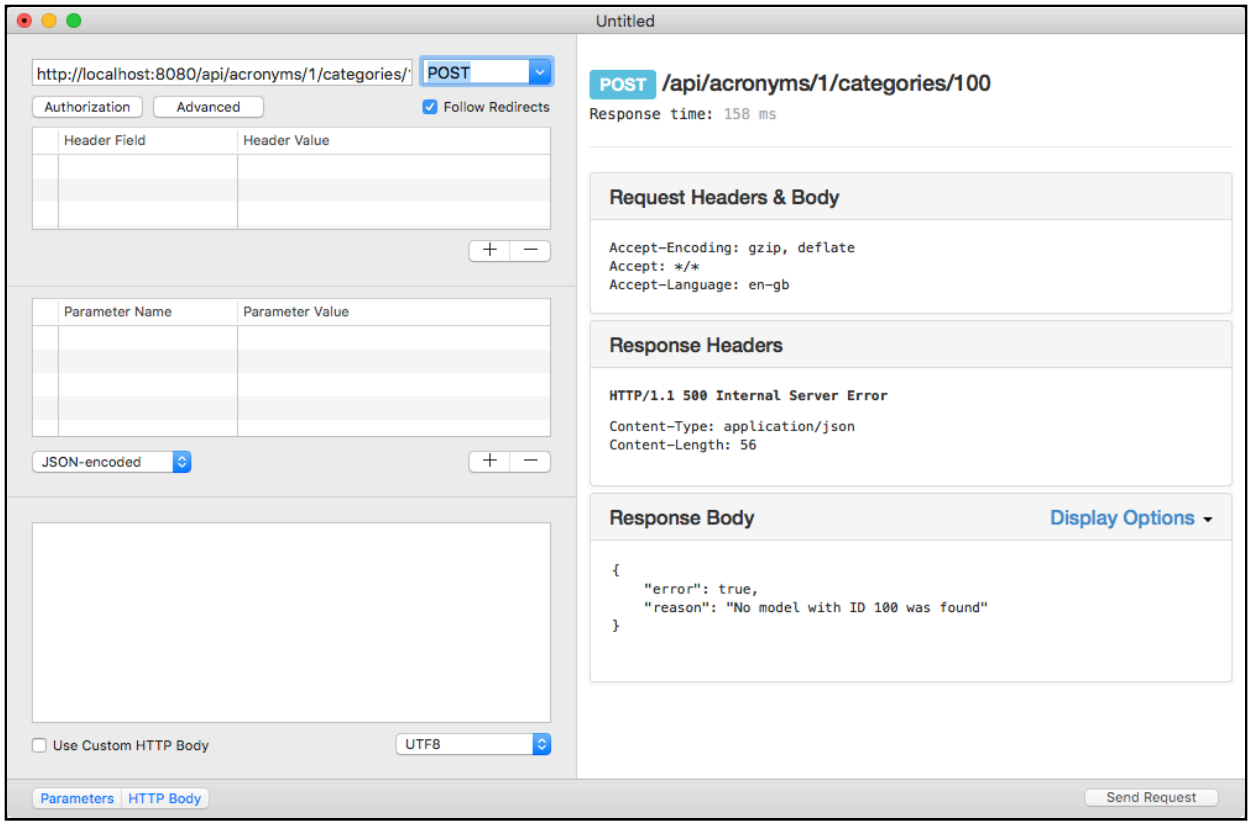
1. 停止运行名为postgres的Docker容器。这是当前运行数据库的容器。

2. 移除名为postgres的Docker容器以删除任何现有数据。

3. 启动一个运行PostgreSQL的新Docker容器。有关更多信息，请参阅第6章“Configuring a Database”。

构建并运行应用程序。在RESTed中创建用户，缩略词和类别。然后，将URL设置为**http://localhost:8080/api/acronyms/1/categories/100**以及**POST**方法。

发送请求，您将收到错误，因为没有ID为**100**的类别：



将URL设置为**http://localhost:8080/api/acronyms/1/categories/1**，并发送请求以将该缩略词添加到类别中。您将在响应标头中看到**201 Created**状态。

然后去哪儿?

在本章中，您学习了如何使用Fluent在Vapor中实现兄弟关系。在本节中，您学习了如何使用Fluent建模所有类型的关系并执行高级查询。 TIL API功能齐全，可供客户使用。

在下一章中，您将学习如何为应用程序编写测试以确保代码正确。然后，本书的下一部分将向您展示如何创建功能强大的客户端以与API进行交互 - 无论是在iOS上还是在Web上。