Chapter 7: CRUD Database Operations

By Tim Condon

第5章“Fluent and Persisting Models”解释了模型的概念以及如何使用Fluent将它们存储在数据库中。本章重点介绍如何与数据库中的模型进行交互。您将了解CRUD操作及其与REST API的关系。您还将了解如何利用Fluent对模型执行复杂查询。

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

CRUD and REST

CRUD操作 - 创建，检索，更新，删除 - 形成持久存储的四个基本功能。通过这些，您可以执行应用程序所需的大多数操作。实际上您在第5章中已实现了第一个功能“创建”。

RESTful API为客户端提供了一种在应用程序中调用CRUD函数的方法。通常，您拥有模型的资源URL。对于TIL应用程序，这是缩略词资源：**http://localhost:8080/api/acronyms**。然后，您可以在此资源上定义路由，并与适当的HTTP请求方法配对，以执行CRUD操作。例如：

- **GET http://localhost:8080/api/acronyms/**: 得到所有的缩略词.
- **POST http://localhost:8080/api/acronyms**: 创建一个新的缩略词.
- **GET http://localhost:8080/api/acronyms/1**: 获得ID为1的缩略词.

- **PUT http://localhost:8080/api/acronyms/1**: 更新ID为1缩略词.
- **DELETE http://localhost:8080/api/acronyms/1**: 删除ID为1的缩略词.

Create

在第5章“Fluent and Persisting Models”中，您实现了缩略词的创建路由。您可以继续执行项目，也可以在本章的starter文件夹中打开TILApp。 回顾一下， 您在**routes.swift**中已创建了一个新的路由处理程序：

```
// 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/**注册新路由，接受POST请求并返回Future <Acronym>。

2. 将请求的JSON解码为缩略词。这很简单，因为Acronym遵循Content协议。decode(_:) 返回Future；使用flatMap(to:)在解码完成时提取缩略词。

3. 使用Fluent保存模型。保存完成后，它将模型作为Future返回 - 在本例中为Future <Acronym>。

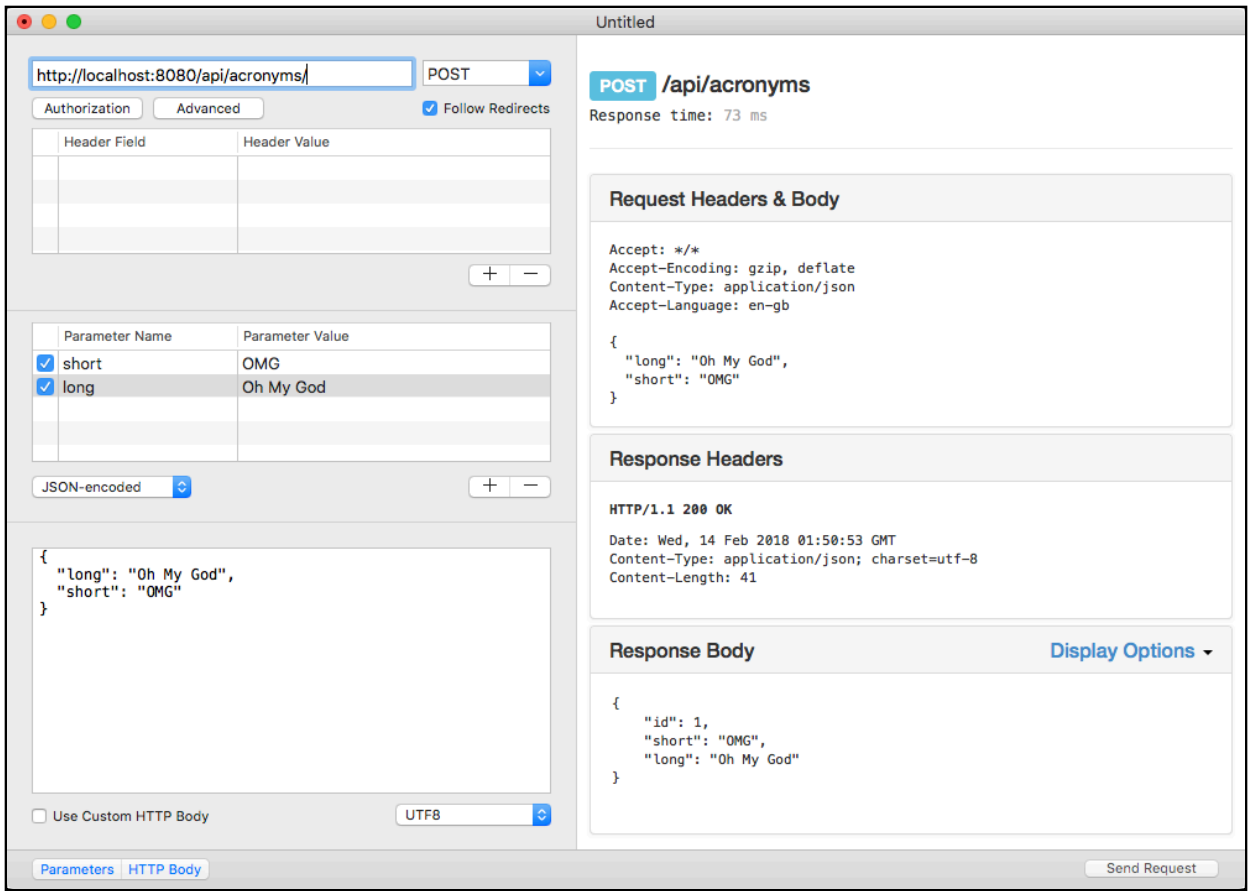
构建并运行应用程序，然后打开RESTed。配置请求如下：

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

发送请求，您将看到包含创建的缩略词的响应：



Retrieve

对于TILApp，检索包含两个单独的操作：检索所有缩略词和检索单个特定缩略词。Fluent使这两项任务变得简单。

检索所有缩略词

要检索所有缩略词，请为**/api/acronyms/**的GET请求创建路由处理程序。打开**routes.swift**并在routes(_:)末尾添加以下内容：

```
// 1
router.get("api", "acronyms") { req -> Future<[Acronym]> in
  // 2
  return Acronym.query(on: req).all()
}
```

这是它的作用：

1. 为请求注册一个新的路由处理程序，返回Future<[Acronym]>，这是future的缩略词数组。

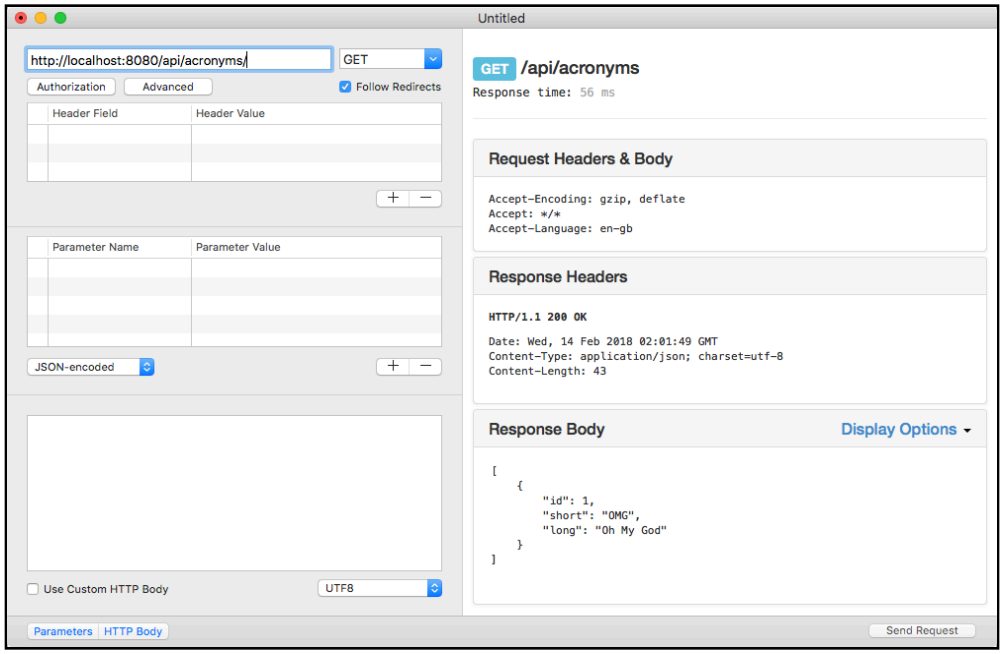
2. 执行查询以获取所有缩略词。

Fluent为模型添加了功能方法，以便能够对模型执行查询。您必须为查询提供一个DatabaseConnectable参数。这几乎总是请求并提供一个线程来执行工作。 all()返回数据库中该类型的所有模型。这相当于SQL查询 SELECT * FROM Acronyms;。

构建并运行您的应用程序，然后在RESTed中创建一个新请求。配置请求如下：

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

发送请求以查看数据库中已有的缩略词：



检索单个缩略词

Vapor强大的参数类型安全性扩展到遵循Parameter协议的模型。要使其为Acronym工作， 请打开**Acronym.swift**并在文件末尾添加以下内容：

```
extension Acronym: Parameter {}
```

要获得单个缩略词，您需要一个新的路由处理程序。打开**routes.swift**并在routes(_:)末尾添加以下内容：

```
// 1
router.get("api", "acronyms", Acronym.parameter) {
  req -> Future<Acronym> in
  // 2
  return try req.parameters.next(Acronym.self)
}
```

这是它的作用：

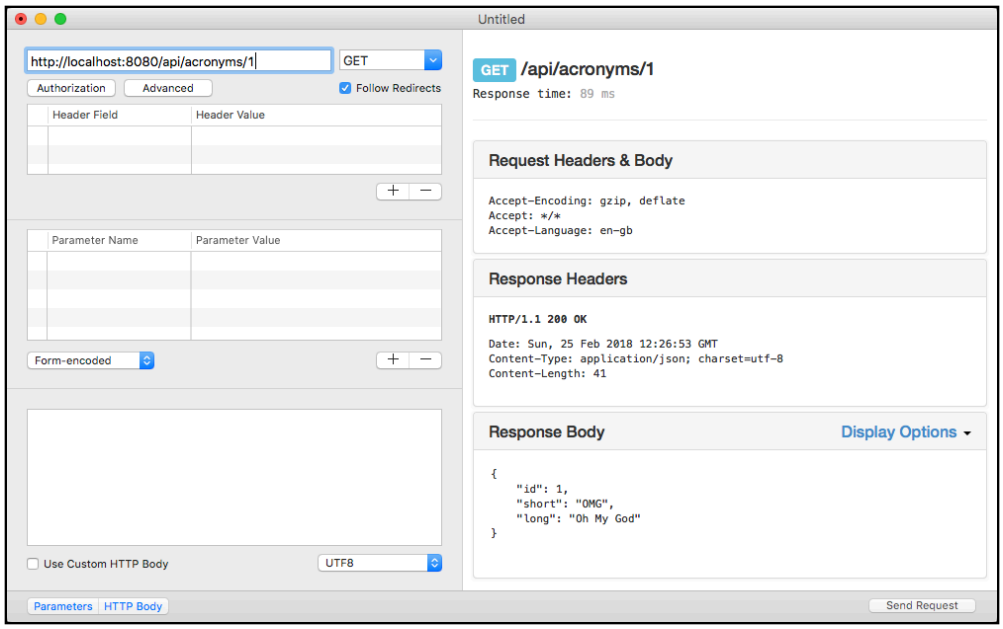
1. 在**/api/acronyms/<ID>**注册路由以处理GET请求。该路由将缩略词的id属性作为最后的路径段。这将返回Future <Acronym>。

2. 使用parameters从请求中提取缩略词。此计算属性执行从数据库获取缩略词所需的所有工作。它还处理错误情况，当缩略词不存在或ID类型错误时，例如，当ID为UUID时却传递了整数。

构建并运行您的应用程序，然后在RESTed中创建一个新请求。配置请求如下：

- **URL**: http://localhost:8080/api/acronyms/1 (1是创建的第一个缩略词的ID)
- **method**: GET

发送请求，您将收到第一个缩略词作为回复：



Update

在RESTful API中，对单个资源的更新使用PUT请求以及包含新信息的请求数据。

在routes(_:)的末尾添加以下内容以注册新的路由处理程序：

```
// 1
router.put("api", "acronyms", Acronym.parameter) {
  req -> Future<Acronym> in
  // 2
  return try flatMap(
    to: Acronym.self,
    req.parameters.next(Acronym.self),
    req.content.decode(Acronym.self)) {
      acronym, updatedAcronym in
      // 3
      acronym.short = updatedAcronym.short
      acronym.long = updatedAcronym.long

      // 4
      return acronym.save(on: req)
  }
}
```

这是它的作用：

1. 将PUT请求的路由注册到返回Future<Acronym>的**/api/acronyms/<ID>**。

2. 使用flatMap(to:_:_:)，flatMap的双重future形式，等待parameter提取和content解码完成。 这提供了数据库的缩略词和请求正文传给闭包的缩略词。

3. 使用新值更新缩写词的属性。

4. 保存缩略词并返回结果。

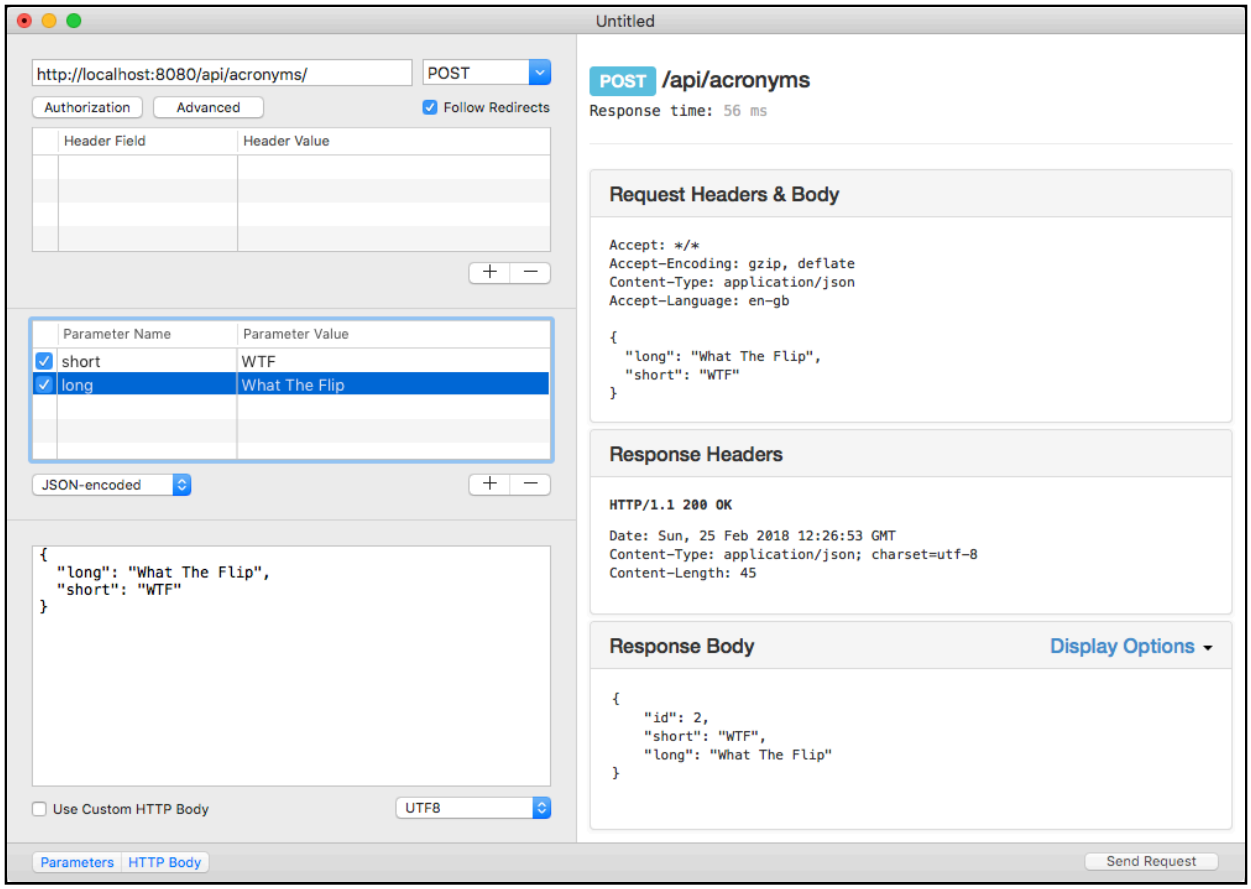
构建并运行应用程序，然后使用RESTed创建一个新的缩略词。配置请求如下：

- **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**: WTF
- **long**: What The Flip

发送请求，您将看到包含创建的缩略词的响应：



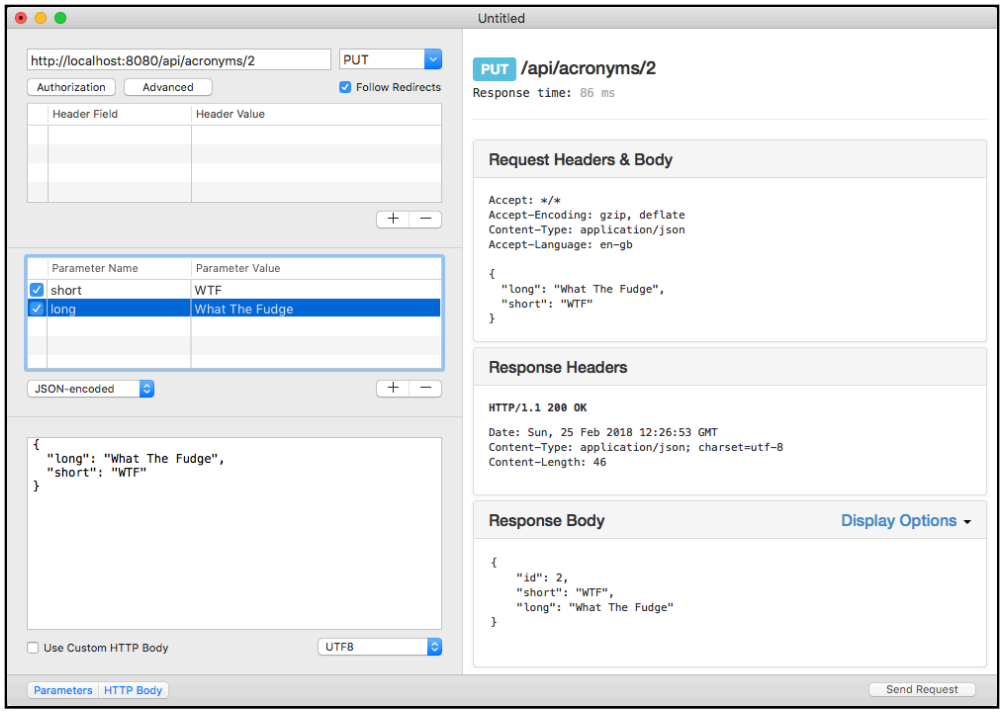
事实证明，WTF的意义实际上并不是“What The Flip”，所以它需要更新。在RESTed中更改请求，如下所示：

- **URL**: http://localhost:8080/api/acronyms/<ID>

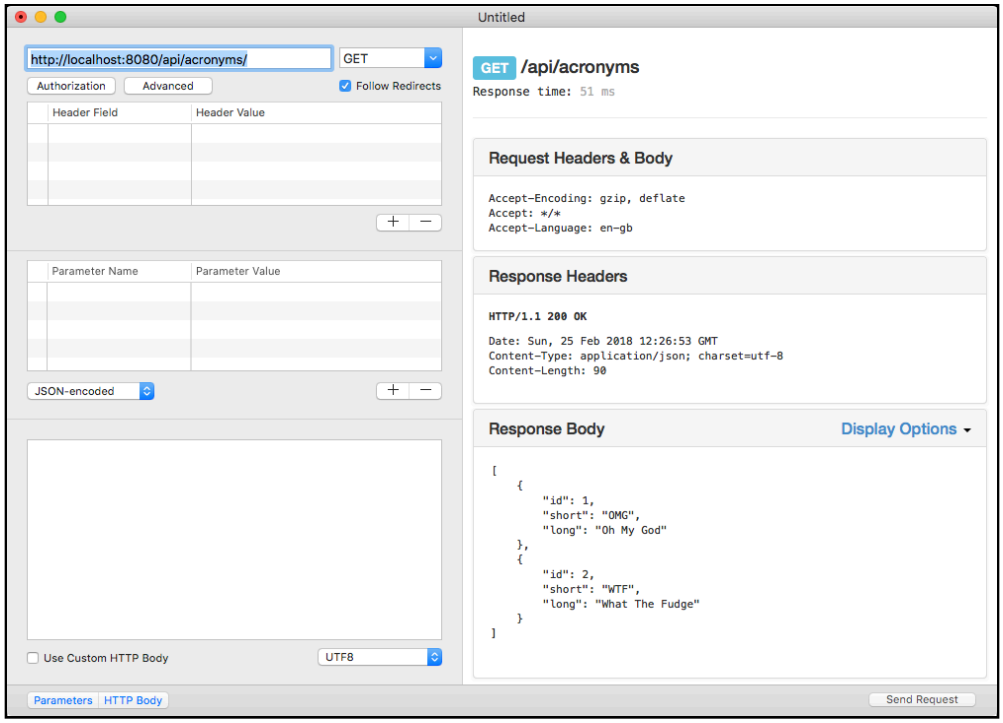
> 使用创建请求路由返回的ID。

- **method**: PUT
- **long**: What The Fudge

发送请求。您将在响应中收到更新的缩略词：



要确保更新成功，请在RESTed中发送请求以获取所有缩略词。在返回响应中您将看到已更新的缩略词：



Delete

要删除RESTful API中的模型，请向资源发送DELETE请求。将以下内容添加到routes(_:)的末尾以创建新的路由处理程序：

```
// 1
router.delete("api", "acronyms", Acronym.parameter) {
  req -> Future<HTTPStatus> in
  // 2
  return try req.parameters.next(Acronym.self)
    // 3
    .delete(on: req)
    // 4
    .transform(to: .noContent)
}
```

这是它的作用：

1. 将DELETE请求的路由注册到**/api/acronyms/<ID>**，返回Future <HTTPStatus>。

2. 从请求的parameters中提取要删除的缩略词。

3. 使用delete(on:)删除缩略词。 Fluent允许您直接在Future上调用delete(on:)，而不是要求您解包返回的Future的包装类。这有助于整理代码并减少嵌套。 Fluent为删除，更新，创建和保存提供了便利方法。

4. 将结果转换为**204 No Content**响应。这告诉客户端请求已成功完成，但没有内容可以返回。

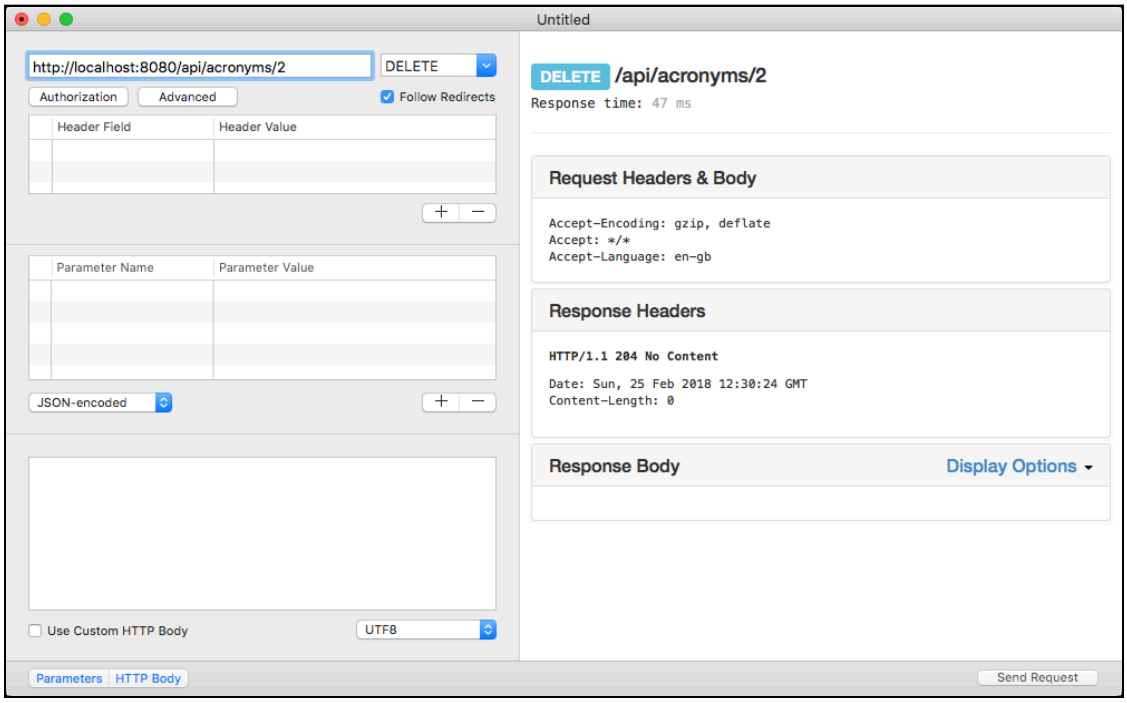
构建并运行应用程序。 “WTF”的缩写是有点不合适，所以删除它。在RESTed中配 置新请求，如下所示：

- **URL**: http://localhost:8080/api/acronyms/<ID>

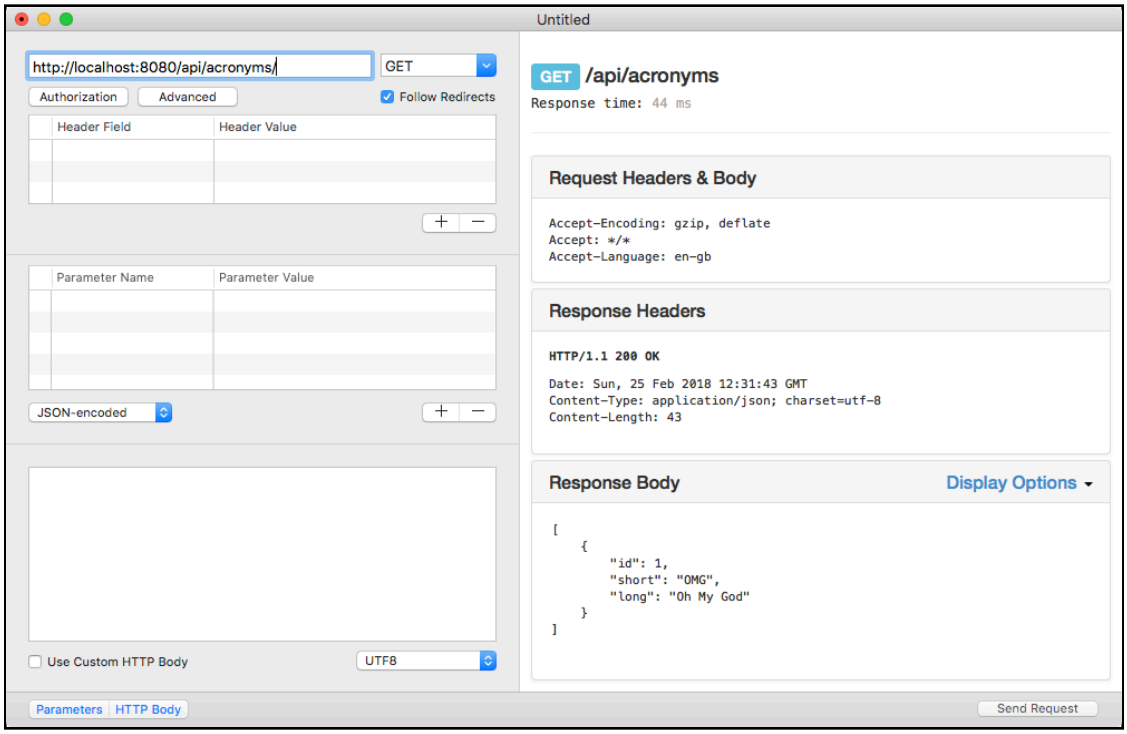
> 使用上一个请求中WTF缩略词的ID

- **method**: DELETE

发送请求;您将收到**204 No Content**回复。



发送获取所有缩略词的请求，您将看到WTF缩略词不再出现在数据库中了。



Fluent 查询

您已经看到Fluent如何轻松地进行基本的CRUD操作。它可以轻松地执行更强大的查询。

Filter

搜索功能是应用程序中的常见功能。如果您想搜索数据库中的所有缩略词，Fluent可以轻松完成。在**routes.swift**顶部的import Vapor语句下面添加以下内容：

```
import Fluent
```

接下来，在routes(_:)末尾添加一个新的路由处理程序用于搜索：

```
// 1
router.get("api", "acronyms", "search") {
  req -> Future<[Acronym]> in
  // 2
  guard
    let searchTerm = req.query[String.self, at: "term"] else {
      throw Abort(.badRequest)
  }
  // 3
  return Acronym.query(on: req)
    .filter(\.short == searchTerm)
    .all()
}
```

这是搜索缩略词的原因：

1. 为**/api/acronyms/search**注册一个新的路由处理程序，返回Future <[Acronym]>。

2. 从URL查询字符串中检索搜索词。如果失败，抛出400 Bad请求错误。

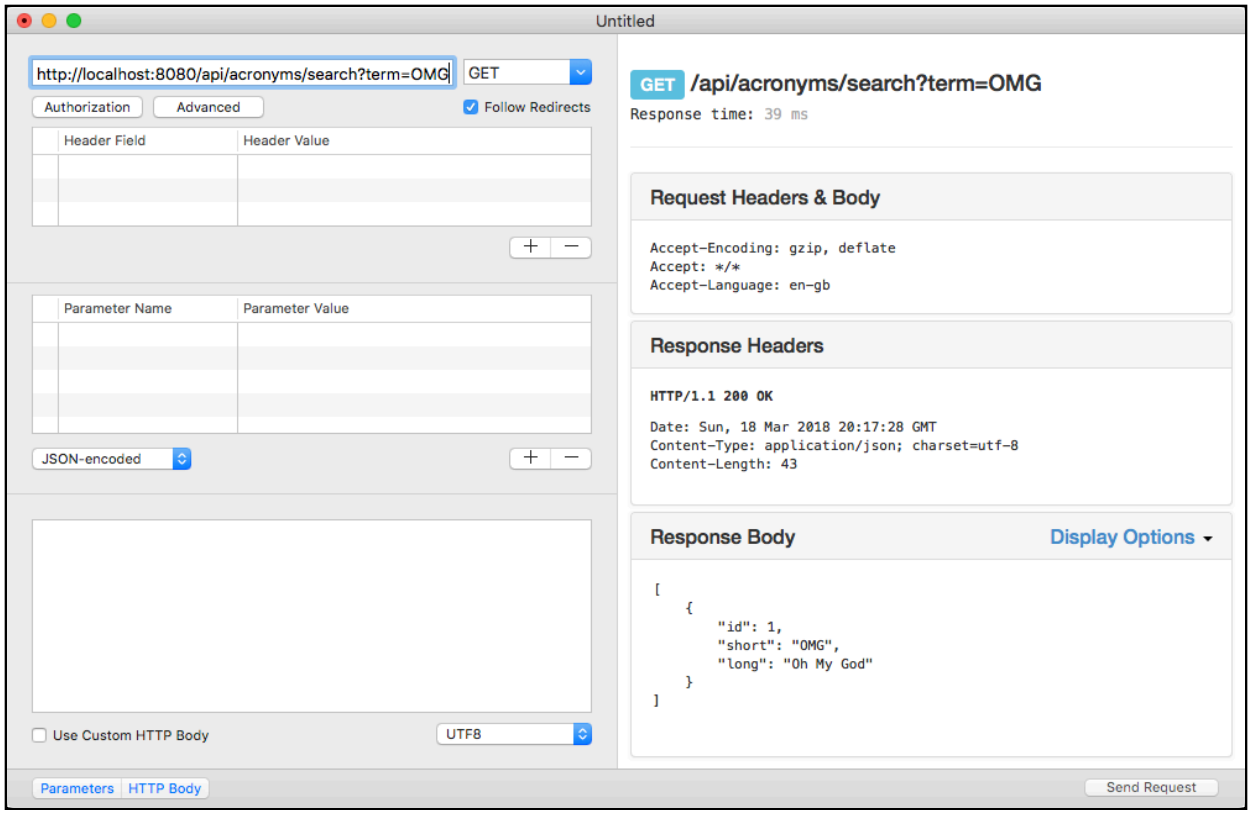
> 注意：URL中的查询字符串允许客户端将不适合在路径中的信息传递给服务器。例如，通常用于定义搜索结果的页码。

3. 使用filter(_:)查找short属性与searchTerm匹配的所有缩略词。因为这使用了key paths，编译器可以对属性和过滤条件强制执行类型安全。这可以防止因指定无效列名或无效类型而导致的运行时问题。

构建并运行您的应用程序，然后在RESTed中创建一个新请求。配置请求如下：

- **URL**: http://localhost:8080/api/acronyms/search?term=OMG
- **method**: GET

发送请求，您将看到返回的OMG缩略词及其含义：



如果要搜索多个字段（例如short字段和long字段），则需要更改查询。你不能链filter(_ :)函数，因为它只匹配short和long属性相同的缩略词。相反，您必须使用**filter group**。用以下内容替换return Acronym.query(on: req).filter(\.short == searchTerm).all()：

```
// 1
return Acronym.query(on: req).group(.or) { or in
  // 2
  or.filter(\.short == searchTerm)
```

```
    // 3
    or.filter(\.long == searchTerm)
  // 4
  }.all()
```

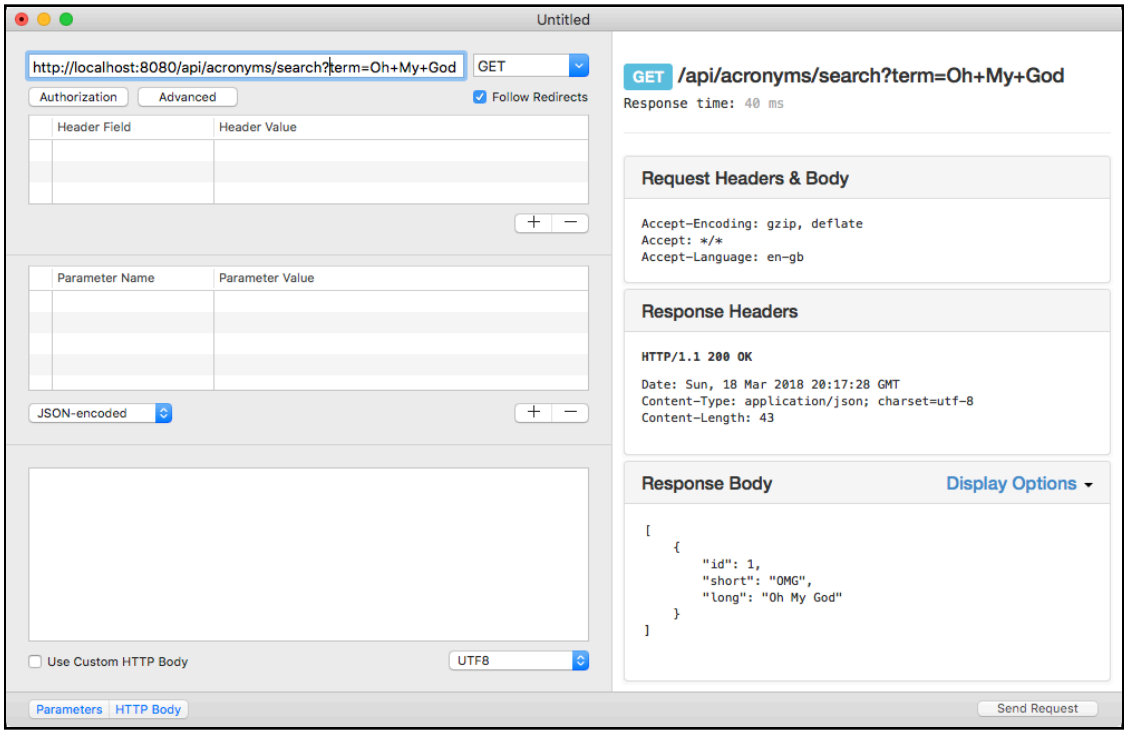
这是它的作用：

1. 使用.or关系创建filter group。
2. 向group中添加过滤器以过滤其short属性与搜索项匹配的缩略词。
3. 向group中添加过滤器以过滤其long属性与搜索项匹配的缩略词。
4. 返回所有结果。

这将返回与第一个过滤器或第二个过滤器匹配的所有缩略词。构建并运行应用程序并回到RESTed。从上面重新发送请求，您仍会看到相同的结果。

将URL更改为 **http://localhost:8080/api/acronyms/search?term=Oh+My+God** 并发送请求。你会得到OMG的缩略词作为回应：

> 注意：URL中的空格必须以URL-encoded为**%20**或+才能有效。



First result

有时，应用程序只需要查询的第一个结果。为此创建特定处理程序可确保数据库仅返回一个结果，而不是将所有结果加载到内存中。在routes(_:)末尾创建一个新的路由处理程序以返回第一个缩写词：

```
// 1
router.get("api", "acronyms", "first") {
  req -> Future<Acronym> in
  // 2
  return Acronym.query(on: req)
    .first()
    .unwrap(or: Abort(.notFound))
}
```

这是这个函数的作用：

1. 为 **/api/acronyms/first** 注册一个新的HTTP GET路由，返回Future <Acronym>。

2. 执行查询以获得第一个缩略词。 first()返回一个可选项，因为数据库中可能没有缩略词。使用unwrap(or:)确保存在缩略词或抛出404 Not Found错误。

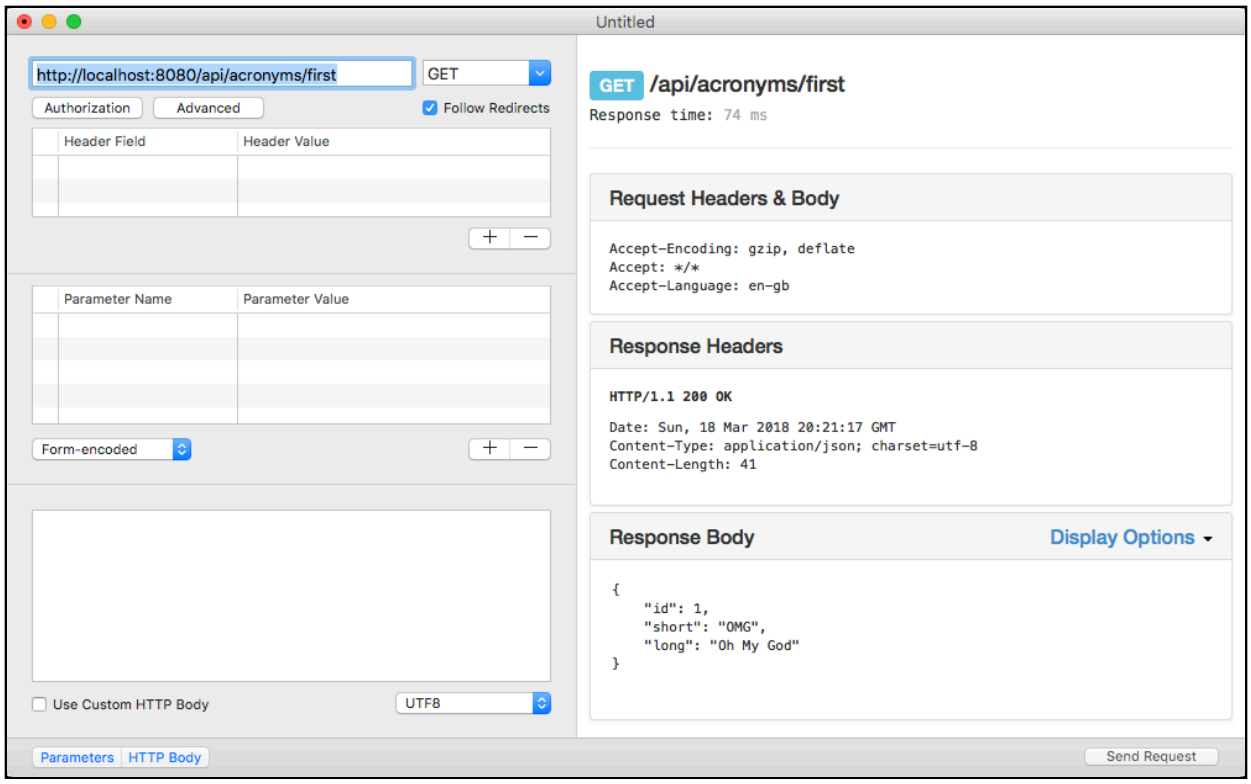
您还可以将.first()应用于任何查询，例如过滤器的结果。构建并运行应用程序，然后打开RESTed。创建新的缩略词：

- **short**: IKR
- **long**: I Know Right

现在创建一个新的RESTed请求，配置为：

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

发送请求，您将看到您创建的第一个缩略词：



Sorting results

应用程序通常需要在返回查询结果之前对其进行排序。因此，Fluent提供了排序功能。在routes(_:)函数末尾写一个新的路由处理程序，返回所有的缩略词，按其short属性升序排序：

```
// 1
router.get("api", "acronyms", "sorted") {
  req -> Future<[Acronym]> in
  // 2
  return Acronym.query(on: req)
    .sort(\.short, .ascending)
    .all()
}
```

这是如何实现的：

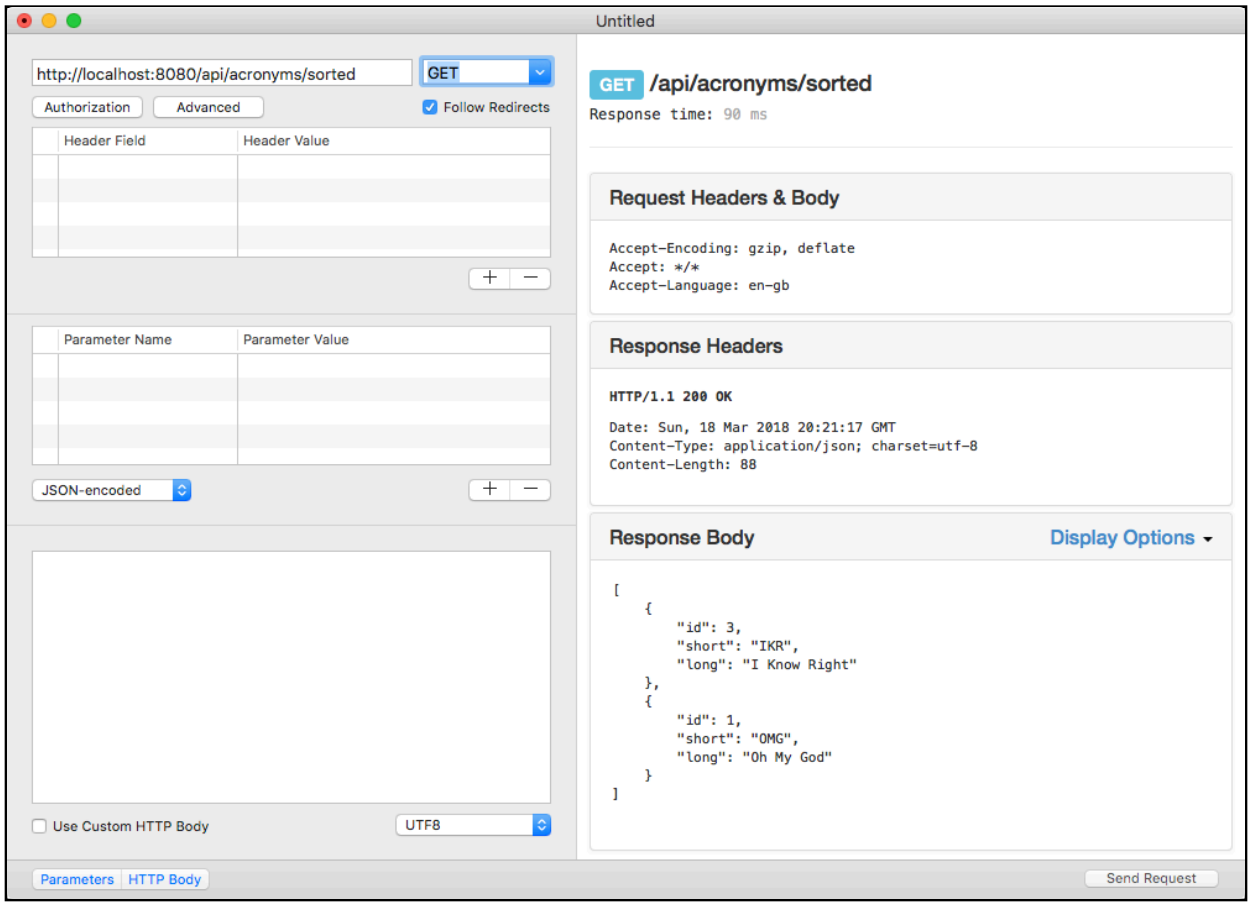
1. 为**/api/acronyms/sorted**注册一个新的HTTP GET路由，返回Future <[Acronym]>。

2. 为Acronym创建查询并使用sort(_:_:)执行排序。此函数采用字段的key path进行排序和排序方向。最后使用all()返回查询的所有结果。

构建并运行应用程序，然后在RESTed中创建一个新请求：

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

发送请求，您将看到按其short属性的字母顺序排序的缩略词：



然后去哪儿?

您现在知道如何使用Fluent执行不同的CRUD操作和高级查询。在这个阶段，**routes.swift**与本章中的所有代码混杂在一起。下一章将介绍如何使用controllers更好地组织代码。