iOS 5 Programming Cookbook

第十三章

Core Data

版本 1.0

翻译时间：2012-08-06

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun dymx101
Jimmylianf BeyondVincent

DevDiv 校对： laigb kyelup

DevDiv 编辑： BeyondVincent

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问 DevDiv 移动开发论坛 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3、xiebaochun、dymx101、jimmylianf 和 BeyondVincent 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译各章节汇总

更多学习资料

我们也为大家准备了 iOS6 新特征的相关内容，请参考下面的帖子：

iOS6 新特征：参考资料和示例汇总------持续更新中

目录

前言

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_前言

第 1 章　基础入门

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第一章_基础入门

第 2 章　使用控制器和视图

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(上)

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(下)

第 3 章　构造和使用 Table View

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第三章_构造和使用 Table View

第 4 章 Storyboards

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第四章_Storyboards

第 5 章　并发

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第五章_并发

第 6 章　定位核心与地图

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第六章_核心定位与地图

第 7 章　实现手势识别的功能

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第七章_实现手势识别

第 8 章　网络，JSON，XML 以及 Twitter

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第八章_网络，JSON，XML 以及 Twitter

第 9 章　音频和视频

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第九章_音频和视频

第 10 章 通讯录

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译 第十章 通讯录

第 11 章 照相机和图片库

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十一章_照相机和图片库

第 12 章 多任务

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十二章_多任务

第 13 章 Core Data

13.0. 介绍

Core Data 是 iOS SDK 里的一个很强大的框架，它允许程序员在一个面向对象的方法里储存和管理数据。传统地，程序员必须在硬盘上使用 Objective-C 的存取功能来储存数据，或者将数据写到文件里并手动来管理它们。使用 Core Data 的引导，程序员可以很轻松有效地通过面向对象的接口管理他们的数据。在这个章节将会学到怎样使用 Core Data 来创建程序的模式（在 Model-View-Controller 的软件构架）。

Core Data 是与一个程序员不可见的底层永久存储交互的。iOS 决定底层数据管理是如何执行的。程序员们必须知道它所提供的高层 API。但是理解 Core Data 的结构和其内部的工作原理是很重要的。那就让我们创建一个 Core Data 应用程序来加深理解吧。

Core Data 在一个 iOS 程序里需要一些设置。幸运的是，使用 Xcode 这个步骤就很简单了。你可以简单地创建一个 Core Data 应用程序，剩下的工作就交给 Xcode 来完成。

按照以下步骤在 Xcode 里创建一个使用 Core Data 的工程：

1、打开 Xcode（假如没有打开的话）。

2、在文件菜单栏里，选择→New Project….

3、在新建工程对话框里，确保 iOS 是主类，Application 是子类。然后在对话框的右边，，选择 Empty Application 并按 Next 按钮。如图 13-1.

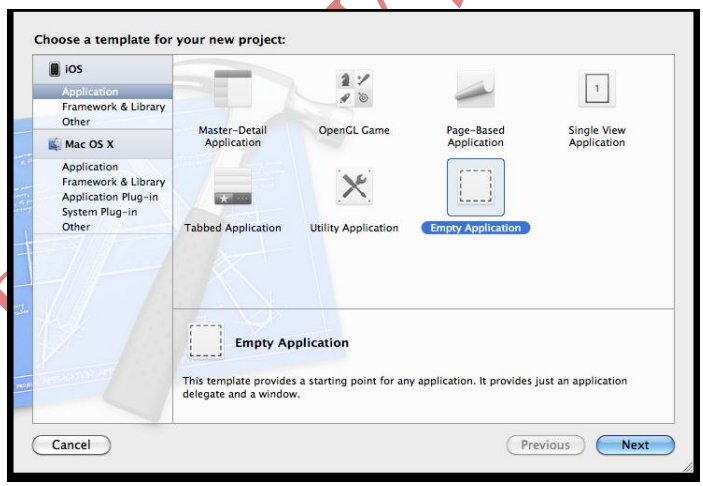

图 13-1 针对 Core Data 创建一个空的工程

4、现在为你的程序名选择引导到 Core Data 并确保 User Core Data 复选框被选中，如图 13-2.完成后按 Next 按钮。

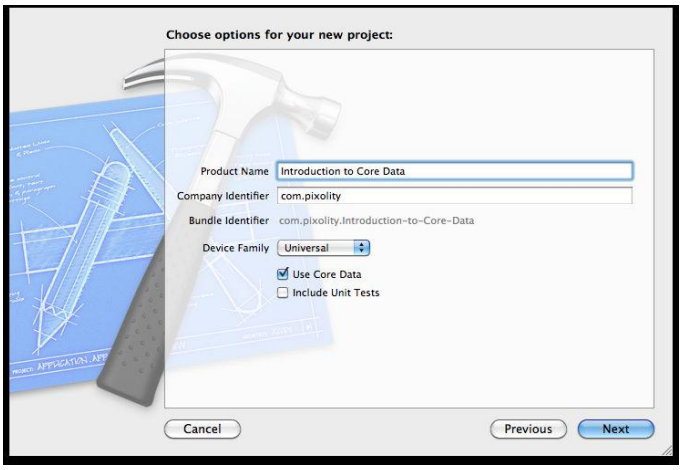

图 13-2 设置使用 Core Data

5、 现在选择你的程序要存储的位置。当你选择好了目标文件夹后，按下 Ccreat 按钮，如图 13-3 所示。

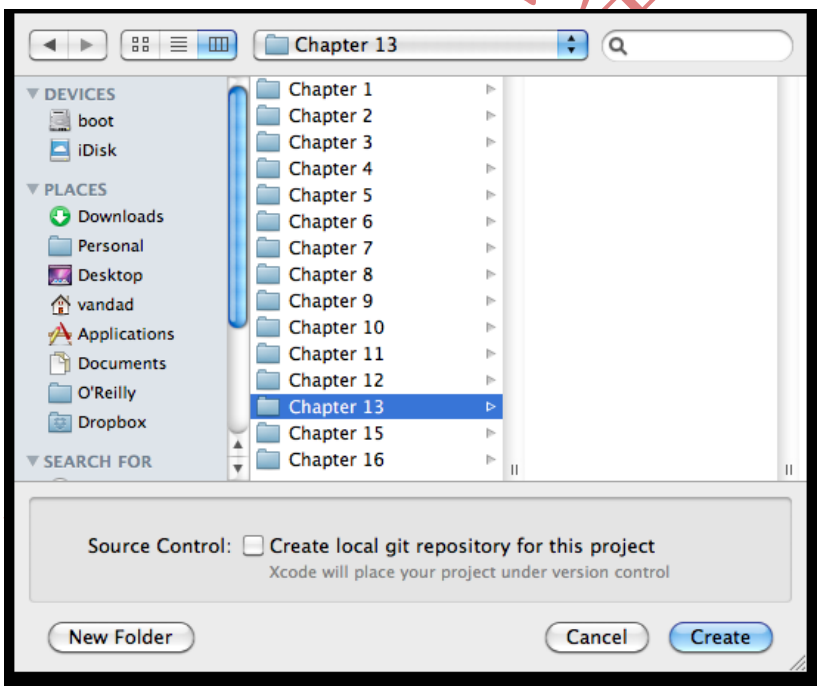

图 13-3 选择 Core Data 工程存储路径

现在在 Xcode 里，找到名为 Introduction_to_Core_DataAppDelegate.h 的文件。当我们的程序是公有时，这个文件是我们程序的共享委托。iPad 和 iPhone 程序的委托都将使用这个委托作为他们的父类。假如你已经看过了这个文件的内容，你会发现三个属性被添加到了程序委托的声明里。这些属性如下：

1、managedObjectContext（属于 NSManagedObjectContext 类型）。

2、managedObjectModel（属于 NSManagedObjectModel）。

3、persistentStoreCoordinator（属于 NSPersistentStoreCoordinator）。

我知道这个很新也许对你来说有点难，但是把这些新的概念和已经熟悉的数据库相比较，这些概念将会很容易被消化。

Persistent store coordinator

这是物理数据存储的物理文件和程序之间的联系的桥梁。这个桥梁将负责管理不同对象上下文。

Managed object model

这跟在一个数据库的框架的概念是一样的。他可以代表一个数据库的桌面或者在创建的数据库里的不同管理对象类型。

Managed object context

这是程序员和管理对象模式之间的桥梁。使用管理对象的上下文，你可以将一个新的列插入到一个新的表格，从一个确定的表格里读取某一列，等等。（其实，Core Data 不会使用表格这个概念，但是在这里我会使用 term 来表示，因为他和列和相似，将会更好的帮助你了解 Core Data 是怎样工作的。）

Managed object

这和表格里的列很相似。我们将管理对象插入到管理对象上下文并保存它。这样，我们在数据库里的表格里创建了一个新的列。

在 13.1 章节。我们会学到如何使用 Xcode 来创建一个 Core Data 模型。这是创建一个数据库框架的第一步。

13.1. 使用 Xcode 创建一个 Core Data Model

13.1.1. 问题

使用 Xcode 来可视化设计 iOS 程序的数据模型。

13.1.2. 方案

按照介绍章节的步骤去创建一个 Core Data 工程。然后使用程序里的 xcdatamodel 拓展找到这个文件并点击它打开数据编辑区，如图 13-4。

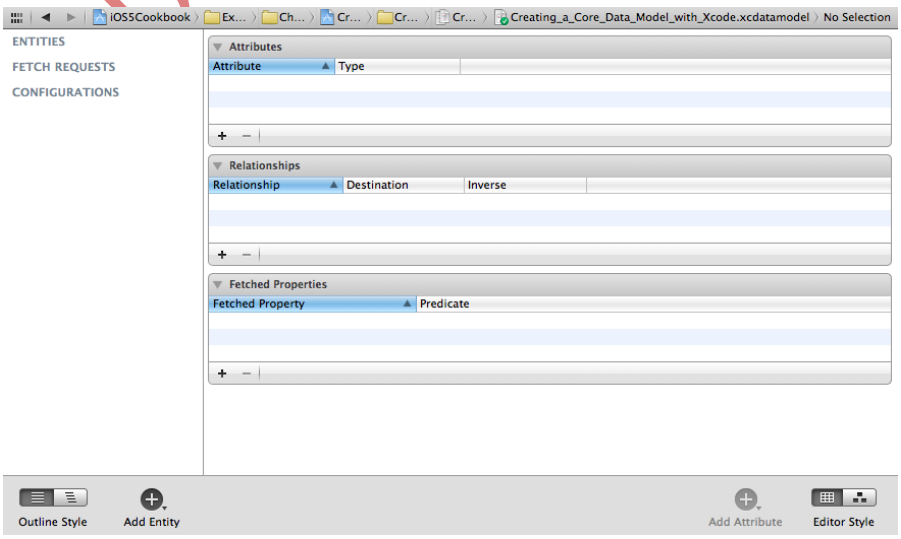

图 13-4 在 XCode 中可视化编辑数据

13.1.3. 讨论

Xcode 可视数据编剧器是一个神奇的工具，它允许程序员很轻松地设计程序的数据模型。在这个工具可以正常工作之前，你需要学习两个重要的定义：

Entity

这和数据库的表格相似。

Attribute

在 entity（实体）里定义一个栏。

当我们在我们的对象模型里生成代码时 Entities 将会变成对象。在 13.2 章节中有所介绍。现在，在这里，我们会专注于在这个工具里创建数据模型。

在编辑器里，在按钮里找到“+”按钮。用鼠标点住这个按钮不放，然后选择菜单栏里显示项里 Add Entity，如图 13.5 所示。

图 13-5 添加新的一条目到数据模型中

新的实体将会被创建并他将会处于可编辑名称的状态。将它命名为 Person，如图 13-6.

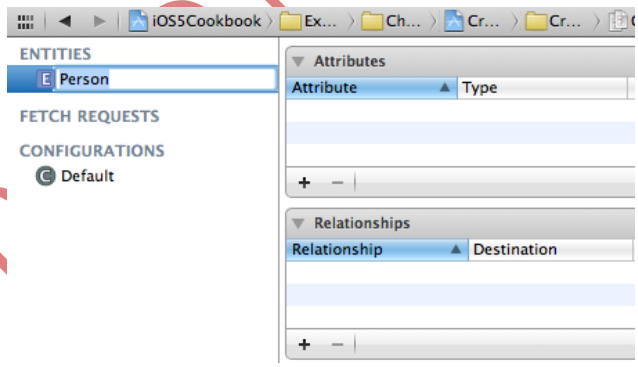

图 13-6 将新条目的名称修改为 Persion

选择 Person Entity 并给它创建下面的三个属性。在属性栏里选择加【+】按钮（如图 13-7）。

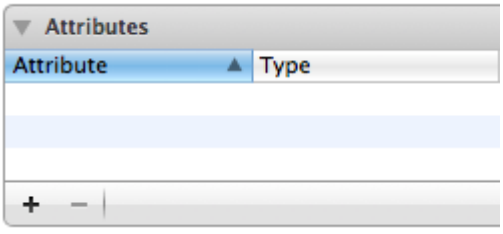

图 13-7 属性框

firstName（属于 String）
lastName（属于 String）
age(属性 Integer32)

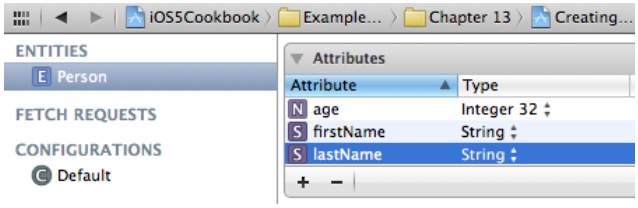

图 13-8 给 Person 条目添加三个属性

在数据编辑模式下，从 Xcode 菜单栏里选择 Utilities→Show Utilities。公共边设备边框将在 Xcode 的右边打开。 现在在顶部选择数据模式检查器按钮并按下刚才创建的 Person 实体。现在，Data Model 检查器会和 Person 实体项目关联起来，如图 13-9 所示。

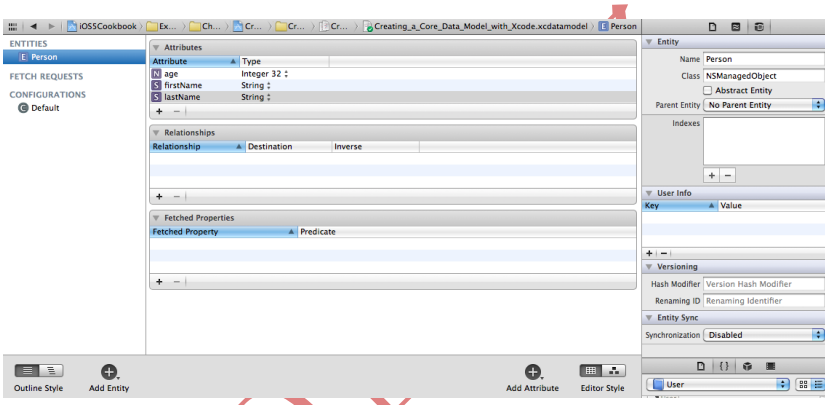

图 13-9 数据模型的 Inspector 显示在 XCode 窗口的右边

先在点击 firstname,lastName, 和 Person 实体 age 属性在 Data Model 检查器，确保 firstName 和 lastname 属性在没有点击 Optional 复选框时是不可选的，在点击了 Optional 复选框后是可选的。

现在在编辑器里的数据模型如图 13-10 所示。

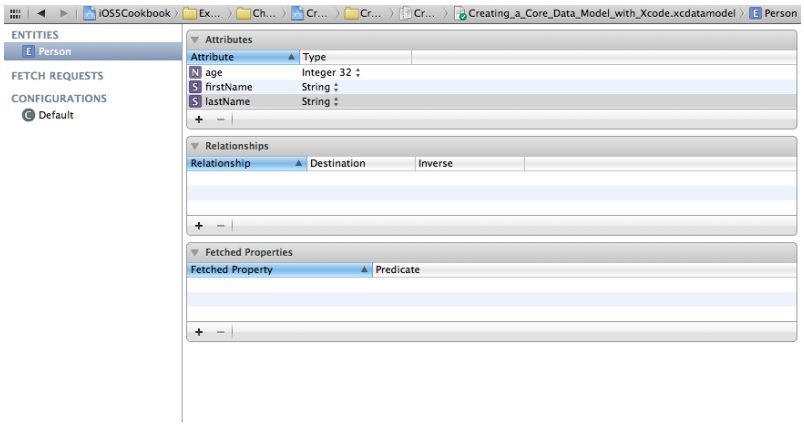

图 13-10 Persion 条目和三个属性

好的，我们已经完成了创建模型。选择文件→Save 确保改变后的内容。要学习如如何在管理对象里生成代码，请参考 13.2 章节。

13.2. 给 Core Data entities 生成类文件

13.2.1. 问题

根据 13.1 小节里的介绍想要知道如何根据对象模型来生成代码。

13.2.2. 方案

按照下面步骤：

1、在 Xcode，使用 xcdatamodel 扩展器找到要创建的程序的文文件。点击文件，并看着右边的编辑器。

2、选择我们之前创建的 Person 实体。

3、在 Xcode 里选择文件→New File。

4、在新建对话框里，确保你选择的主类是 iOS，Core Data 为子类。然后在右边的对话框里选择 NSManagedObject subclass 项并按 Next 按钮，如图 13-11 所示。

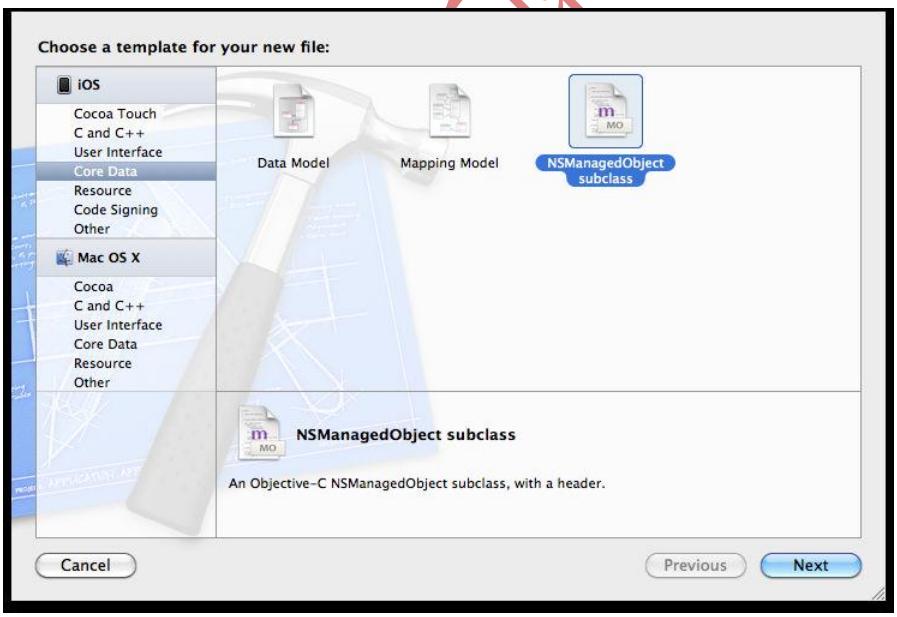

图 13-11 在 XCode 中创建一个 NSManagedObject 子类

5、现在选择工程文件保存的位置，然后按下 Create 按钮，如图 13-12。

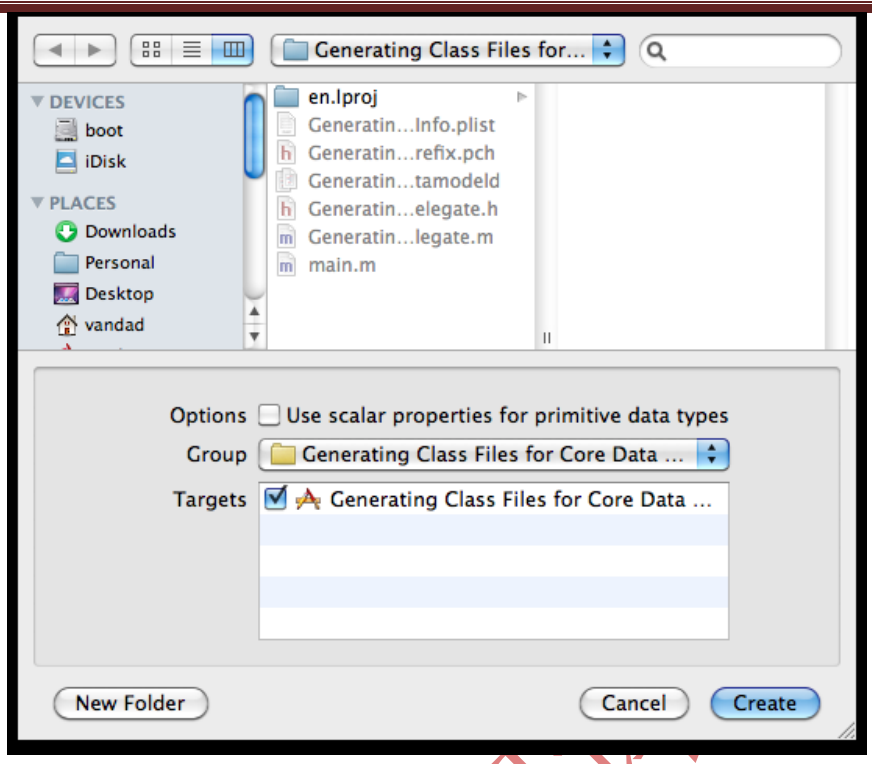

图 13-12 选择 Managed 对象的目标路径

现在你会在你的工程里看到两个新的文件，分别为 Person.h 和 Person.m。打开 Person.h 文件。将看到如下的内容：

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface Person : NSManagedObject {
@private
}
@property (nonatomic, retain) NSString * firstName;
@property (nonatomic, retain) NSString * lastName;
@property (nonatomic, retain) NSNumber * age;

@end
```

The *Person.m* file is implemented for us in this way:

```
#import "Person.h"
@implementation Person
@dynamic firstName;
@dynamic lastName;
@dynamic age;

@end
```

将我们的管理对象变为真实的定义并且编译它，在 13.3 章节我们将会学到如何实例化和保存 Person 类型的管路对象为程序的管理上下文。

13.2.3. 讨论

当你在 Xcode 使用编辑器创建你的数据模型时，也同时在进行创建它们之间的关系，实

体，属性，等。但是要在你的程序里使用你的数据模型，必须先为你的数据模型生成代码。假如你查看实体的.h 和.m 文件的话你会发现所有的属性都是静态定义的。

运行在程序的 Core Data 的代码是不可见的，并且刚开始这些代码也没有必要对成程序员可见。

你所需要知道的是一个 Person 实体具有三个属性 firstname,lastName 和 age。你可以将这些属性绑定一些数据，也可以从上下文里保存和下载这些属性，请参考 13.3 小节。

13.3. 使用 Core Data 创建和保存数据

13.3.1. 问题

你已经创建了一个管理对象并且你想实例化它然后将它插入到程序 Core Data 的上下文里。

13.3.2. 方案

按照 13.1 和 13.2 小节的介绍。现在 NSEntityDescription 的 insertnewObjectForEntityForName:inManageobjectContext:类方法的第一个属性创建一个新的对象。一旦这个新的实体被创建后，你可以通过它的属性修改它。完成之后，使用管理对象上下文的 save：的实例方法保存管理对象的上下文。

假设你已经在 Xcode 里创建了一个名为“Creating and saving Data Using Core Data”的程序，并通过以下步骤插入了一个管理对象到上下文：

1、找到名为“Create_and_Saving_data_Using_Core_DataAppDelegate.m 的文件。

2、导入 Person.h 文件到程序委托的执行文件。

Person 是我们在 13.1 小节创建的实体。

```
#import "Creating_and_Saving_Data_Using_Core_DataAppDelegate.h"
#import "Person.h"

@implementation Creating_and_Saving_Data_Using_Core_DataAppDelegate

@synthesize window = _window;
@synthesize managedObjectContext = __managedObjectContext;
@synthesize managedObjectModel = __managedObjectModel;
@synthesize persistentStoreCoordinator = __persistentStoreCoordinator;
...
```

3、在你共享程序委托的 application：didFinishLauchingWithOptions:的方法里写如下代码：

```
- (BOOL)                application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  self.window = [[UIWindow alloc] initWithFrame:
```

```
                [[UIScreen mainScreen] bounds]];

Person *newPerson = [NSEntityDescription
                        insertNewObjectForEntityForName:@"Person"
                        inManagedObjectContext:self.managedObjectContext];
if (newPerson != nil){

   newPerson.firstName = @"Anthony";
   newPerson.lastName = @"Robbins";
   newPerson.age = [NSNumber numberWithUnsignedInteger:51];

   NSError *savingError = nil;

   if ([self.managedObjectContext save:&savingError]){
      NSLog(@"Successfully saved the context.");
   } else {
      NSLog(@"Failed to save the context. Error = %@", savingError);
   }
} else {
      NSLog(@"Failed to create the new person.");
}

self.window.backgroundColor = [UIColor whiteColor];
   [self.window makeKeyAndVisible];
   return YES;
}
```

13.3.3. 讨论

前面的章节里介绍了如何使用 Xcode 的编辑器去创建实体和生成代码。接下来我们就要使用这些尸体并实例化它们。在这里，我们使用 NSEntityDescription 和并调用 insertNewObjectForEntityForname:inManagedObjectContext:类方法。他将会在已知的管理对象上下文里查找已知的实体（根据它的名字可断定为是 NSString 类型的）。假如这个实体被找出来了，那么这个方法将返回这个实体一个新实例。这就好像在数据库里(管理对象上下文)创建一个新栏（管理对象）。

尝试着在管理对象上下文里插入一个未知的实体将会产生一个 NSInternalInconsistencyException；类型的异常。

将一个新的实体插入到上下文后，我们必须保存上上下文。这将会所有没有储存的上下文数据放到永久储存器里。我们可以使用管理对象的 save:实例方法。假如 Bool 返回的数值是 yes,我们可以确定我们的上下文已经储存起来了。在 13.4 章节里我们会学到如何从内存里读取数据。

13.4. 从 Core Data 里读取数据

13.4.1. 问题

想要使用 Core Data 读取实体（桌面）的内容。

13.4.2. 方案

使用 NSFetchrequest 的一个实例：

```
- (BOOL) createNewPersonWithFirstName:(NSString *)paramFirstName
                             lastName:(NSString *)paramLastName
                                  age:(NSUInteger)paramAge{

BOOL result = NO;

if ([paramFirstName length] == 0 ||
   [paramLastName length] == 0){
NSLog(@"First and Last names are mandatory.");
return NO;
}
Person *newPerson = [NSEntityDescription
                     insertNewObjectForEntityForName:@"Person"
                     inManagedObjectContext:self.managedObjectContext];

 if (newPerson == nil){
    NSLog(@"Failed to create the new person.");
    return NO;
}
newPerson.firstName = paramFirstName;
newPerson.lastName = paramLastName;
newPerson.age = [NSNumber numberWithUnsignedInteger:paramAge];

NSError *savingError = nil;

    if ([self.managedObjectContext save:&savingError]){
       return YES;
    } else {
       NSLog(@"Failed to save the new person. Error = %@", savingError);
    }

    return result;
}
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

[self createNewPersonWithFirstName:@"Anthony"
                          lastName:@"Robbins"
                               age:51];

[self createNewPersonWithFirstName:@"Richard"
                          lastName:@"Branson"
                               age:61];

/* Create the fetch request first */
NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

/* Here is the entity whose contents we want to read */
NSEntityDescription *entity =
```

```
    [NSEntityDescription
     entityForName:@"Person"
     inManagedObjectContext:self.managedObjectContext];

     /* Tell the request that we want to read the
       contents of the Person entity */
     [fetchRequest setEntity:entity];

     NSError *requestError = nil;

     /* And execute the fetch request on the context */
     NSArray *persons =
     [self.managedObjectContext executeFetchRequest:fetchRequest
                                              error:&requestError];

     /* Make sure we get the array */
     if ([persons count] > 0){

        /* Go through the persons array one by one */
       NSUInteger counter = 1;
       for (Person *thisPerson in persons){

          NSLog(@"Person %lu First Name = %@",
                (unsigned long)counter,
                thisPerson.firstName);

          NSLog(@"Person %lu Last Name = %@",
                (unsigned long)counter,
                  thisPerson.lastName);

          NSLog(@"Person %lu Age = %ld",
                (unsigned long)counter,
                (unsigned long)[thisPerson.age unsignedIntegerValue]);
          counter++;
       }
     } else {
        NSLog(@"Could not find any Person entities in the context.");
     }

     self.window = [[UIWindow alloc] initWithFrame:
                    [[UIScreen mainScreen] bounds]];

     self.window.backgroundColor = [UIColor whiteColor];
     [self.window makeKeyAndVisible];
     return YES;
}
```

更多的关于获取请求，请参考本章节讨论部分。

13.4.3. 讨论

对于熟悉数据库专业术语的你来说，一个获取请求就像一个 DELECT 状态。在 SELECT 状态中，你可以确定某一行，在这种情况下，将返回这一行的数据。这个获取请求，我们做了同样的事情。我们确定了实体（桌面）和管理对象的上下文（数据库的底层）。我们也可以为我们读取的分类数据确定分类描述符。但是为了简单点只把重点放在读取数据上。

为了能够读取 Person 实体（在 13.1 小节创建并在 13.2 小节转换成代码）的内容。首先我们请求 NSEntityDescription 类来为名为 Person 的实体查找我们的管理对象上下文。一旦找到它，我们会告诉我们要从实体读取的获取请求。然后，剩下的工作就是像本章节里的方案部分里的计算获取请求。

NSManagedOnjectContext 的 executeFetchRequest：error：实例方法返回的值不是 nil（出现错误）就是 Person 管理对象的数据。假如没有发现已知实体的结果，返回的数据为空。

13.4.4. 参考：

13.1 小节和 13.2 小节

13.5. 从 Core Data 里删除数据

13.5.1. 问题

想要从一个管理对象上下文里删除一个管理对象。

13.5.2. 方案

使用 NSManagedObjectContext 的 deleteObject：实例方法：

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

[self createNewPersonWithFirstName:@"Anthony"
                          lastName:@"Robbins"
                               age:51];

[self createNewPersonWithFirstName:@"Richard"
                          lastName:@"Branson"
                               age:61];

/* Create the fetch request first */
NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

/* Here is the entity whose contents we want to read */
NSEntityDescription *entity =
[NSEntityDescription entityForName:@"Person"
                inManagedObjectContext:self.managedObjectContext];

/* Tell the request that we want to read the
contents of the Person entity */
[fetchRequest setEntity:entity];

NSError *requestError = nil;

/* And execute the fetch request on the context */
```

```
    NSArray *persons =
    [self.managedObjectContext executeFetchRequest:fetchRequest
                                              error:&requestError];

    /* Make sure we get the array */
    if ([persons count] > 0){
    /* Delete the last person in the array */
    Person *lastPerson = [persons lastObject];

    [self.managedObjectContext deleteObject:lastPerson];

    NSError *savingError = nil;
    if ([self.managedObjectContext save:&savingError]){
       NSLog(@"Successfully deleted the last person in the array.");
    } else {
       NSLog(@"Failed to delete the last person in the array.");
    }

    } else {
       NSLog(@"Could not find any Person entities in the context.");
    }

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

在这个例子里我们使用了 createNewPersonWithFirstName：lastName:age:这个方法（在 13.4 章节里介绍了。

13.5.3. 讨论

你可以使用 NSManagedObjectContext 的 deleteObject：的实例方法删除管理对象（在一个数据库里的表格记录）。

这个方法不会返回错误依然不会返回一个 Bool 值。所以你确实没有一个很好的方法知道对象是否已经删除或是否正在使用管理对象上下文。判断一个对象是否被删除的最好的方法是使用管理对象的 isDelated 方法。

有了这个知识，那就然我们改动一下前面章节的代码吧。如下：

```
- (BOOL) application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

  [self createNewPersonWithFirstName:@"Anthony"
                            lastName:@"Robbins"
                                 age:51];
    [self createNewPersonWithFirstName:@"Richard"
                              lastName:@"Branson"
                                   age:61];
```

```
 /* Create the fetch request first */
 NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

/* Here is the entity whose contents we want to read */
NSEntityDescription *entity =
[NSEntityDescription entityForName:@"Person"
            inManagedObjectContext:self.managedObjectContext];

/* Tell the request that we want to read the
contents of the Person entity */
[fetchRequest setEntity:entity];

NSError *requestError = nil;
/* And execute the fetch request on the context */
NSArray *persons =
[self.managedObjectContext executeFetchRequest:fetchRequest
                                         error:&requestError];

/* Make sure we get the array */
if ([persons count] > 0){

   /* Delete the last person in the array */
   Person *lastPerson = [persons lastObject];

    [self.managedObjectContext deleteObject:lastPerson];

    if ([lastPerson isDeleted]){
       NSLog(@"Successfully deleted the last person...");
       NSError *savingError = nil;

       if ([self.managedObjectContext save:&savingError]){
          NSLog(@"Successfully saved the context.");
       } else {
          NSLog(@"Failed to save the context.");
       }

    } else {
       NSLog(@"Failed to delete the last person.");
    }

    } else {
      NSLog(@"Could not find any Person entities in the context.");
    }

    self.window = [[UIWindow alloc] initWithFrame:
                   [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
     return YES;
}
```

一旦运行这个程序，你将得到如下的打印在控制台上结果：

```
Successfully deleted the last person...
Successfully saved the context.
```

13.6. 在 Core Data 给数据分类

13.6.1. 问题

想要在管理对象的上下文里分类管理对象。

13.6.2. 方案

给每个需要分类的实体属性创建 NSSortDescriptor 的实例。添加分类符号一个数组里并使用 setSortDescriptor：实例方法将数组签名到 NSFetchrequest 的一个实例。在示例代码里，Sorting_Data_in_Core_dataAppDelegate 是一个普通程序里德委托代表类。要知道如何创建 Person 实体，请参考 13.1 和 13.2 小节。

```
- (BOOL) application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    [self createNewPersonWithFirstName:@"Richard"
                               lastName:@"Branson"
                                    age:61];

    [self createNewPersonWithFirstName:@"Anthony"
                               lastName:@"Robbins"
                                    age:51];

     /* Create the fetch request first */
     NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

     /* Here is the entity whose contents we want to read */
     NSEntityDescription *entity =
     [NSEntityDescription entityForName:@"Person"
                 inManagedObjectContext:self.managedObjectContext];

     NSSortDescriptor *ageSort =
     [[NSSortDescriptor alloc] initWithKey:@"age"
                                 ascending:YES];

     NSSortDescriptor *firstNameSort =
     [[NSSortDescriptor alloc] initWithKey:@"firstName"
                                 ascending:YES];
     NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:
                                  ageSort,
                                  firstNameSort, nil];

     fetchRequest.sortDescriptors = sortDescriptors;

     /* Tell the request that we want to read the
       contents of the Person entity */
       [fetchRequest setEntity:entity];

       NSError *requestError = nil;
   /* And execute the fetch request on the context */
```

```
    NSArray *persons =
                    [self.managedObjectContext executeFetchRequest:fetchRequest
                                                              error:&requestError];
  for (Person *person in persons){

    NSLog(@"First Name = %@", person.firstName);
    NSLog(@"Last Name = %@", person.lastName);
    NSLog(@"Age = %lu", (unsigned long)[person.age unsignedIntegerValue]);

  }

  self.window = [[UIWindow alloc] initWithFrame:
                 [[UIScreen mainScreen] bounds]];

   self.window.backgroundColor = [UIColor whiteColor];
   [self.window makeKeyAndVisible];
   Return YES;
}
```

13.6.3. 讨论

一个 NSFetchrequest 的实例携带自身的 NSSortDescriptor 的实例数组。每个分类符都是在当前需要分类的实体定义属性的。例如，Person 实体具有 firstname,lastAnme,age 属性。假如我们想读取管理对象上下文的所有联系人并将他们按照从小到大的年龄分类，我们将会创建一个 NSSortDescriptor 的实例并使用 age 关键字并将它设置成 ascending：

```
NSSortDescriptor *ageSortDescriptor =
[[NSSortDescriptor alloc] initWithKey:@"age"
ascending:YES];
```

。
你可以签名多个分类符到一个获取请求

13.6.4. 参考

13．4 章节

13.7. 在桌面视图增强数据交互

13.7.1. 问题

在一个使用桌面视图来给用户呈现管理对象的程序里，你想要让获取和呈现数据比手动管理数据跟流畅些。

13.7.2. 方案

使用获取结果控制器，它是 NSFetchedResultsController。

13.7.3. 讨论

获取结果控制器和桌面视图的工作方式是一样的。都有 sections 和 rows。一个获取结果控制器能够从管理对象上下文读取管理对象并将他们分别放到 sections 和 rows 里。每个 section 都是一个组（假如你指定它），每个 Row 在一个 Section 里是一个管理对象。你可以很轻松地将数据以表格的形式呈现给用户。这里有一些为什么你会修改你的程序而是用获取结果控制器的原因：

1、 在一个管理对象上下文里创建了一个获取结果控制器后，任何改变（插入、删除、修改等）都会立即反应到获取结果控制器上。例如，你可以创建获取结果控制器来读取 Person 实体的管理对象。然后在程序的其他地方，插入一个新的 Person 管理对象到上下文获取结果控制器创建的同一个上下文）。新的管理对象将会立即在获取结果控制器里变成可访问的。这是不是很神奇呢！

2、 有了获取结果控制器你可以更加有效地管理储存器了。例如，你可以请求你的获取结果控制器来让管理对象的数值 N 保持在每个控制器实例。

3、 获取结果控制器很像桌面视图一样具有 sections 和 rows，像之前解释的那样。你可以使用获取结果控制器在程序 GUI 的桌面视图来呈现管理对象。

这里有一些获取结果控制器的重要属性和实例（全部都是 NSFetchedResultsController 类型的）：

Sections（属性，NSArray 类型）

一个获取结果控制器可以使用一个重要的方法将数据归类。NSFetchResultsController 类的指定初始化器接受通过 sectionnameKeyPath 参数群组的过滤器。Section 数组会包含每个类 section。在这个数组里的每个对象都要遵守 NSFetchedResultsSectionInfo 协议。

objectAtIndexPath：（实例方法，返回一个管理对象）。

带有获取结果控制器的获取对象可以使用它们的 section 和 row 索引来获得它。每个 section 包含了从 0 到 N-1 个行，N 是在 section 里的总项目条数。一个索引路径对象包括一个 section 和一个 row 索引，并且完美配合需要从获取结果控制器获取对象的信息。objectAtIndexPath：实例方法接受索引路径。每个索引路径都是 NSIndexPath 类型。如果你需要在一个获取结果控制器里使用一个管理对象构造一个桌面视图单元格，只要通过一个桌面视图的 tableView：cellForRowAtIndexPath：委托方法的 cellForRowAtIndexPath 参数里的索引路径对象即可。假如你想要在程序里的任何位置构造一个索引路径，请使用 NSIndexPath 的 indexPathForRow：inSection：类方法。

fetchRequest（属性，属于 NSFetchRequest 类型）

假如在程序的任何位置，你相信你需要为获取结果控制器改变获取请求对象，你可以使用 NSFetchedRequestsController 的一个实例的 fetchRequest 属性。这是很有用的，例如，假如你想要在调用和初始化你的获取结果控制器之后改变获取结果对象的分类描述符（参考 13.6 小节的分类描述符）。

为了证明获取结果控制器是有用的，我们需要创建一个程序，这个程序允许我们使用用户接口添加管理对象到管理对象上下文，也允许我们从管理对象上下文里删除管理对象。这里，我们需要两个视图控制器：

Persons List View Controller

这将回事我们的根视图控制器，并且在这里，会显示桌面视图，在视图里我们会成列管

理对象上下文的所有 Person 管理对象。请参考 13.1 小节查看如何创建 Person 管理对象的。在视图控制器的桌面视图允许用户通过点击导航条上的 Edit 按钮删除 Person 对象。我们还会在导航条上放置“+”按钮，允许用户往管理对象上下文里添加新的 Person 对象。

Add New Person View Controller

我们会使用视图控制器来允许用户添加新的 Person 对象到管理对象上下文。

只要按照以下步骤即可完成：

1、 先从一个空白的应用程序开始吧。打开 Xcode，选择 File→New→New Project…并选择 iOS 的一个普通的空白的程序（Core Data 是可用的，请参考 13.1 小节获取更多信息。）并将它命名为 Booting Data Access in Table Views。现在选择在 Xcode 里 Booting_Data_Access_in_Table_Views.xcdatamodeld 并且创建一样的 Person 实体（在 13.1 小节创建过的）并将它保存为 Person。

2、 你工程里，创建两个视图控制器，两个都是 UIViewController 的子类，分别命名为 PersonListViewController 和 AddPersonViewController。创建的这两个视图控制器时不带 Xib 文件的（我们不需要 Interface Builder,因为我们 UI 很简单）。

3、 现在打开程序委托的声明并定义一个新的 PersonListViewController 类型的属性并将它命名为 personListViewController。这将是我们要呈现给用户的根视图控制器。记住我们也需要一个导航控制器，所以让我们也将他定义为属性吧：

```
#import <UIKit/UIKit.h>

@class PersonListViewController;

@interface Boosting_Data_Access_in_Table_ViewsAppDelegate
        : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@property (nonatomic, strong)
  PersonListViewController *personListViewController;

@property (nonatomic, strong) UINavigationController *navigationController;

@property (readonly, strong, nonatomic)
  NSManagedObjectContext *managedObjectContext;

@property (readonly, strong, nonatomic)
  NSManagedObjectModel *managedObjectModel;

@property (readonly, strong, nonatomic)
  NSPersistentStoreCoordinator *persistentStoreCoordinator;

- (void)saveContext;
- (NSURL *)applicationDocumentsDirectory;

@end
```

4、现在让我们开始编译程序的委托并同步视图控制器和导航控制器属性，并确保我们已经将 PersonListViewController.h 头文件导入到程序的委托编译文件里，下面是我么要初始化这个对象的一个实例。

```
#import "Boosting_Data_Access_in_Table_ViewsAppDelegate.h"
#import "PersonListViewController.h"
```

```
@implementation Boosting_Data_Access_in_Table_ViewsAppDelegate

@synthesize window = _window;
@synthesize managedObjectContext = __managedObjectContext;
@synthesize managedObjectModel = __managedObjectModel;
@synthesize persistentStoreCoordinator = __persistentStoreCoordinator;
@synthesize personListViewController;
@synthesize navigationController;
...
```

5、当我们的程序加载后，我们想要呈现 Person List View Controller 到窗口，所以我们现在程序的委托 application：didFinishLaunchingWithOption：方法里做文章吧：

```
- (BOOL) application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

   self.personListViewController =
   [[PersonListViewController alloc]
      initWithNibName:nil
      bundle:nil];

   self.navigationController =
   [[UINavigationController alloc]
      initWithRootViewController:self.personListViewController];

   self.window = [[UIWindow alloc] initWithFrame:
    [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    self.window.rootViewController = self.navigationController;

   return YES;
}
```

6、当 Person List 视图控制器显示完后，我们想要在这放置桌面视图来将管理对象上下文的所有的 Person 实体显示出来。让我们现在创建一个桌面并将我们的视图控制器设置成视图委托和数据资源。我们也需要一个在导航条用“+”标记的添加按钮，这样当用户按钮这个按钮就可以添加新的 Person 视图控制器。我给它添加了两条属性，我们也需要在根视图控制器放置获取姐夫哦控制器。所以让我们在视图控制器里定义所有所需的属性吧：

```
#import <UIKit/UIKit.h>
#import <CoreData/CoreData.h>

@interface PersonListViewController : UIViewController
                    <UITableViewDelegate,
                     UITableViewDataSource,
                    NSFetchedResultsControllerDelegate>

@property (nonatomic, strong) UITableView *tableViewPersons;
@property (nonatomic, strong) UIBarButtonItem *barButtonAddPerson;
@property (nonatomic, strong) NSFetchedResultsController *personsFRC;

@end
```

7、按照惯例，让我们同时结合我们的属性：

```
#import "PersonListViewController.h"

@implementation PersonListViewController

@synthesize tableViewPersons;
@synthesize barButtonAddPerson;
@synthesize personsFRC;

...
```

8、接下来，我们需要需要开始执行我们的视图控制器。在 viewDidLoad 方法，让我们实例化我们 Add 导航条按钮。同样的，让我们实例化桌面视图并设置在导航条上的编辑按钮：

```
- (void)viewDidLoad{
    [super viewDidLoad];

    self.title = @"Persons";

    self.tableViewPersons =
    [[UITableView alloc] initWithFrame:self.view.bounds
                                 style:UITableViewStylePlain];

    self.tableViewPersons.delegate = self;
    self.tableViewPersons.dataSource = self;

  [self.view addSubview:self.tableViewPersons];
  self.barButtonAddPerson = [[UIBarButtonItem alloc]
                             initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
                             target:self
                             action:@selector(addNewPerson:)];

  [self.navigationItem setLeftBarButtonItem:[self editButtonItem]
                                   animated:NO];

    [self.navigationItem setRightBarButtonItem:self.barButtonAddPerson
                                      animated:NO];
}
- (void)viewDidUnload{
  [super viewDidUnload];
  self.barButtonAddPerson = nil;
  self.tableViewPersons = nil;
}
```

9、很显然，现在我们选择了根视图控制器变成桌面视图的委托和数据资源。现在，桌面上将会返回 0 个单元格。之后，当我们可能使用获取结果控制器从管理对象上下文读取 Person 实体，我们可以 Person 对象的可能的数量：

```
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{
  return 0;
}
```

```
- (UITableViewCell *)tableView:(UITableView *)tableView
                cellForRowAtIndexPath:(NSIndexPath *)indexPath{
  return nil;
  }
```

10、现在让我们将“+”放置到 Add Person 视图控制器，所以要确保你已经将 AddPersonViewController.h 头文件导入到 Person List 视图控制器的执行文件里：

```
- (void) addNewPerson:(id)paramSender{

  AddPersonViewController *controller = [[AddPersonViewController alloc]
                                         initWithNibName:nil
                                         bundle:nil];

  [self.navigationController pushViewController:controller
                                       animated:YES];
}
```

11、在 Add Person 视图控制器里，我们需要 4 个属性。三个文本框，一个用来显示 FirstName，第二个显示 lastName，最后一个显示年龄，第四个属性是 Add 导航按钮，我们将它放置在导航条上。当用户完成后将会按钮这个按钮来添加新的联系人到列表里：

```
#import <UIKit/UIKit.h>
#import <CoreData/CoreData.h>

@interface AddPersonViewController : UIViewController

@property (nonatomic, strong) UITextField *textFieldFirstName;
@property (nonatomic, strong) UITextField *textFieldLastName;
@property (nonatomic, strong) UITextField *textFieldAge;
@property (nonatomic, strong) UIBarButtonItem *barButtonAdd;
@end
```

12、接下来，结合我们的属性：

```
#import "AddPersonViewController.h"

@implementation AddPersonViewController

@synthesize textFieldFirstName;
@synthesize textFieldLastName;
@synthesize textFieldAge;
@synthesize barButtonAdd;

...
```

13、现在在我们的 Add Person 视图控制器的 viewDidLoad 方法里，让我们实例化这些属性并将他们放置到视图中并将条按钮放置到导航条上：

```
- (void)viewDidLoad{
   [super viewDidLoad];

   self.title = @"New Person";

   CGRect textFieldRect = CGRectMake(20.0f,
                                     20.0f,
```

```
                                        self.view.bounds.size.width - 40.0f,
                                        31.0f);

    self.textFieldFirstName = [[UITextField alloc] initWithFrame:textFieldRect];
    self.textFieldFirstName.placeholder = @"First Name";
    self.textFieldFirstName.borderStyle = UITextBorderStyleRoundedRect;
    self.textFieldFirstName.autoresizingMask = UIViewAutoresizingFlexibleWidth;
    self.textFieldFirstName.contentVerticalAlignment =
      UIControlContentVerticalAlignmentCenter;
    [self.view addSubview:self.textFieldFirstName];

    textFieldRect.origin.y += 37.0f;
    self.textFieldLastName = [[UITextField alloc] initWithFrame:textFieldRect];
    self.textFieldLastName.placeholder = @"Last Name";
    self.textFieldLastName.borderStyle = UITextBorderStyleRoundedRect;
    self.textFieldLastName.autoresizingMask = UIViewAutoresizingFlexibleWidth;
    self.textFieldLastName.contentVerticalAlignment =
      UIControlContentVerticalAlignmentCenter;
    [self.view addSubview:self.textFieldLastName];

    textFieldRect.origin.y += 37.0f;
    self.textFieldAge = [[UITextField alloc] initWithFrame:textFieldRect];
    self.textFieldAge.placeholder = @"Age";
    self.textFieldAge.borderStyle = UITextBorderStyleRoundedRect;
    self.textFieldAge.autoresizingMask = UIViewAutoresizingFlexibleWidth;
    self.textFieldAge.keyboardType = UIKeyboardTypeNumberPad;
    self.textFieldAge.contentVerticalAlignment =

      UIControlContentVerticalAlignmentCenter;
    [self.view addSubview:self.textFieldAge];

    self.barButtonAdd =
    [[UIBarButtonItem alloc] initWithTitle:@"Add"
                                     style:UIBarButtonItemStylePlain
                                    target:self
                                    action:@selector(createNewPerson:)];
    [self.navigationItem setRightBarButtonItem:self.barButtonAdd
                                     animated:NO];
}

    - (void)viewDidUnload{
        [super viewDidUnload];
        self.textFieldFirstName = nil;
        self.textFieldLastName = nil;
        self.textFieldAge = nil;
        self.barButtonAdd = nil;
}
```

14、你可以看到，Add 按钮现在已经超链接到了 Add Person 视图控制器的 crateNewPerson：方法，所以我们需要执行这个方法并取得文本框里的值，将它放置到一个新的 Person object。然后我们将需要保存这个对象到管对象的上下文并视图控制器弹出返回到 Person List 视图控制器（这时 Person View Controller 将需要显示新的的 Person 到列表里）。综上所述，我们要确保已经将 Booting_Data_Access_in_Table_ViewAppDelegate.h 头文件到 Add Person 视图控制器执行文件以至于我们呢能创建一个新的 person 并使用在程序里的委托管理对象上下文插入这个 person 到数据库里：

```
#import "AddPersonViewController.h"
#import "Person.h"
#import "Boosting_Data_Access_in_Table_ViewsAppDelegate.h"

@implementation AddPersonViewController

@synthesize textFieldFirstName;
@synthesize textFieldLastName;
@synthesize textFieldAge;
@synthesize barButtonAdd;

...
```

15、现在让我们执行在 Add Person 视图控制器的 createNewPerson：方法：

```
- (void) createNewPerson:(id)paramSender{

Boosting_Data_Access_in_Table_ViewsAppDelegate *appDelegate =
   (Boosting_Data_Access_in_Table_ViewsAppDelegate *)
      [[UIApplication sharedApplication] delegate];

NSManagedObjectContext *managedObjectContext =
   appDelegate.managedObjectContext;

Person *newPerson =
[NSEntityDescription insertNewObjectForEntityForName:@"Person"
                              inManagedObjectContext:managedObjectContext];

if (newPerson != nil){

   newPerson.firstName = self.textFieldFirstName.text;
   newPerson.lastName = self.textFieldLastName.text;
   newPerson.age = [NSNumber numberWithInteger:
                    [self.textFieldAge.text integerValue]];

   NSError *savingError = nil;

   if ([managedObjectContext save:&savingError]){
   [self.navigationController popViewControllerAnimated:YES];
} else {
   NSLog(@"Failed to save the managed object context.");
}

} else {
   NSLog(@"Failed to create the new person object.");
}
}
```

16、为了提供更佳的用户体验，当视图显示到屏幕上时，我们可以自动在 First Name 文本框里显示键盘：

```
- (void) viewDidAppear:(BOOL)paramAnimated{
    [super viewDidAppear:paramAnimated];
    [self.textFieldFirstName becomeFirstResponder];
}
```

17、我们完成了 Add Person 视图控制器。你可以自己先动手尝试着完成了。现在让我

们到 Person List 视图控制器并在初始化时实例化获取结果控制器
：

```
#import "PersonListViewController.h"
#import "AddPersonViewController.h"
#import "Boosting_Data_Access_in_Table_ViewsAppDelegate.h"
#import "Person.h"

@implementation PersonListViewController

@synthesize tableViewPersons;
@synthesize barButtonAddPerson;
@synthesize personsFRC;

- (NSManagedObjectContext *) managedObjectContext{

Boosting_Data_Access_in_Table_ViewsAppDelegate *appDelegate =
   (Boosting_Data_Access_in_Table_ViewsAppDelegate *)
      [[UIApplication sharedApplication] delegate];

NSManagedObjectContext *managedObjectContext =
   appDelegate.managedObjectContext;

return managedObjectContext;
}
- (id) initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil{

self = [super initWithNibName:nibNameOrNil
                       bundle:nibBundleOrNil];

if (self != nil){

/* Create the fetch request first */
NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

/* Here is the entity whose contents we want to read */
NSEntityDescription *entity =
[NSEntityDescription entityForName:@"Person"
            inManagedObjectContext:[self managedObjectContext]];
   NSSortDescriptor *ageSort =
   [[NSSortDescriptor alloc] initWithKey:@"age"
                                 ascending:YES];

   NSSortDescriptor *firstNameSort =
   [[NSSortDescriptor alloc] initWithKey:@"firstName"
                                ascending:YES];

   NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:
                               ageSort,
                               firstNameSort, nil];

   fetchRequest.sortDescriptors = sortDescriptors;

   /* Tell the request that we want to read the
   contents of the Person entity */
   [fetchRequest setEntity:entity];
```

```
    self.personsFRC =
    [[NSFetchedResultsController alloc]
        initWithFetchRequest:fetchRequest
        managedObjectContext:[self managedObjectContext]
        sectionNameKeyPath:nil
        cacheName:nil];

        self.personsFRC.delegate = self;
        NSError *fetchingError = nil;
        if ([self.personsFRC performFetch:&fetchingError]){
            NSLog(@"Successfully fetched.");
        } else {
            NSLog(@"Failed to fetch.");
        }
    }
    return self;
}
```

18、现在让我们继续并执行桌面视图委托的各种方法（我们已经在之前留给他们所需最小的执行空间了）。我们必须现在从获取结果控制器里读取获取管理对象并在桌面视图里显示 Person 管理对象：

```
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section{

id <NSFetchedResultsSectionInfo> sectionInfo = [self.personsFRC.sections
                                                    objectAtIndex:section];

    return [sectionInfo numberOfObjects];
}
- (UITableViewCell *)tableView:(UITableView *)tableView
            cellForRowAtIndexPath:(NSIndexPath *)indexPath{

    UITableViewCell *result = nil;

    static NSString *PersonTableViewCell = @"PersonTableViewCell";

    result = [tableView dequeueReusableCellWithIdentifier:PersonTableViewCell];

    if (result == nil){
        result =
        [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
                                    reuseIdentifier:PersonTableViewCell];

        result.selectionStyle = UITableViewCellSelectionStyleNone;
    }
        Person *person = [self.personsFRC objectAtIndexPath:indexPath];

        result.textLabel.text =
            [person.firstName stringByAppendingFormat:@" %@", person.lastName];

        result.detailTextLabel.text =
        [NSString stringWithFormat:@"Age: %lu",

            (unsigned long)[person.age unsignedIntegerValue]];

        return result;
    }
```

19、现在你可以运行这个程序并自己来测试它。现在在程序的唯一问题就是假如用户在 Add Person 视图控制器并添加一个新的 person 到管理对象上下文，当用户返回到 Person List 视图控制器时，新插入的新 person 将不会在列表里显示。这是因为我们获取结果控制器不知道新的对象在管理所对象上下文里。解决这个问题的方法是在 Person List 视图控制器里执行获取结果控制器对象的 controllerDidChangeContent：委托方法。当我们创建获取结果控制器时，我们将视图控制器变成获取结果控制器的委托，然后我们执行这个方法。当管理对象上下文的对象发生变化时将调用这个方法，一旦这个方法被调用时，我们可以继续并加载桌面视图：

```
- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller{
[self.tableViewPersons reloadData];
}
```

20、接下来在 Person List 视图控制器里要做的就是提交编辑：

```
- (void) tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
      forRowAtIndexPath:(NSIndexPath *)indexPath{

    Person *personToDelete = [self.personsFRC objectAtIndexPath:indexPath];

    /* Very important: we need to make sure we are not reloading the table view
    while deleting the managed object */
    self.personsFRC.delegate = nil;

    [[self managedObjectContext] deleteObject:personToDelete];

    if ([personToDelete isDeleted]){
      NSError *savingError = nil;
      if ([[self managedObjectContext] save:&savingError]){

        NSError *fetchingError = nil;
        if ([self.personsFRC performFetch:&fetchingError]){
          NSLog(@"Successfully fetched.");

            NSArray *rowsToDelete = [[NSArray alloc]
                                     initWithObjects:indexPath, nil];
          [tableViewPersons
          deleteRowsAtIndexPaths:rowsToDelete
          withRowAnimation:UITableViewRowAnimationAutomatic];

        } else {
          NSLog(@"Failed to fetch with error = %@", fetchingError);
        }
      } else {
        NSLog(@"Failed to save the context with error = %@", savingError);
      }
    }

    self.personsFRC.delegate = self;
}
- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView
          editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath{
return UITableViewCellEditingStyleDelete;
```

```
}
  - (void) setEditing:(BOOL)paramEditing
              animated:(BOOL)paramAnimated{

  [super setEditing:paramEditing
           animated:paramAnimated];

  if (paramEditing){
     [self.navigationItem setRightBarButtonItem:nil
                                       animated:YES];
  } else {
     [self.navigationItem setRightBarButtonItem:self.barButtonAddPerson
                                         animated:YES];
  }
  [self.tableViewPersons setEditing:paramEditing
                           animated:YES];
  }
```

现在全部完成了。你可以自己尝试着做一下。

13.8. 在 Core data 里执行 Relationships

13.8.1. 问题

想要将管理对象关联其它信息，例如，将 Person 和她住的 Home 关联起来。

13.8.2. 方案

在模型编辑器里使用逆向关系。

13.8.3. 讨论

在 Core Data 里的关系可以是一对一，逆向一对多，或者逆向多对多的关系。下面有每种关系类型的例子：

一对一关系：

例如，人和她的鼻子的关系。每个人都只有一个鼻子，每个鼻子只能属于某一个人的。

逆向一对多关系：

例如，员工和管理者的关系。员工只能有一个管理者，但是管理者可以有多个员工为他工作。这里，员工和管理者的关系是一对一的关系，但是反过来则是一对多的关系；请注意“逆向“这个词的意义。

多对多的关系：

例如，一个人和一辆车的关系。一辆车可以被很多人使用，一个人也可以使用很多的车。

在 Core Data 里，你可以创建一对一的关系，但是我强烈建议你不要这么做，因为回顾

一下之前讲过的列表的例子，人知道那个鼻子是她的，但是鼻子不知道他是属于谁的。在一个面向对象的编程语言中，例如 Objective-C，最好是创建逆向的关系，这样子元素可以参照父元素之间的关系。

下面就来创建一个逆向一对多关系的有利数据模型：

1、 在 Xcode 里，找到 xcdatamodel 文件，当你启动你的工程时它会为你创建这种关系模型，如之前的图 13-1 所示(参考 13.1 小节去创建这样的工程)。

2、 点击编辑器打开数据模型文件。

3、 将之前创建的实体都移除掉，选中它们在键盘上按 Delete 键即可移除。

4、 创建一个实体并命名为 Employ。为实体创建三个属性，命名为 firstName(tring 类型的) ，lastName(string 类型)，和 age(Integer 类型)，如图 13-13 所示。

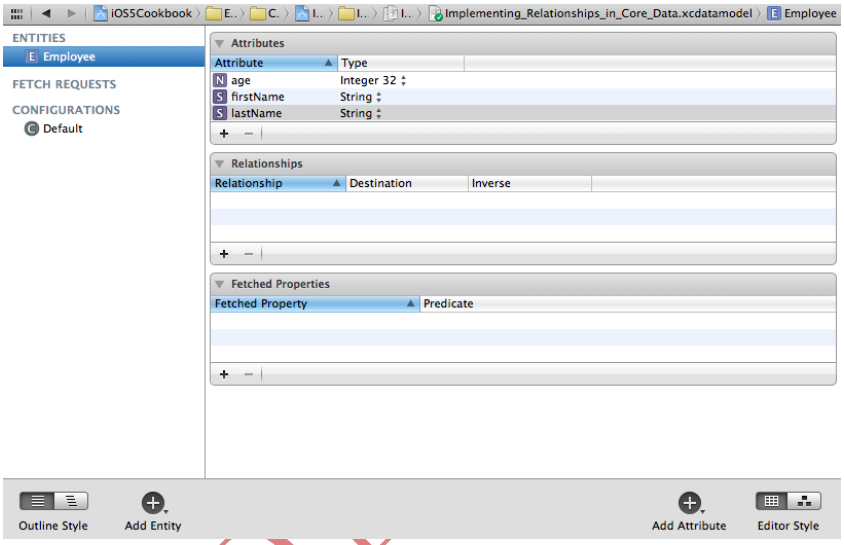

图 13-13 Employee 条目与三个属性

5、 创建另外一个实体并命名为 Manager，创建和 Employ 一样的属性（firstName(tring 类型的) ，lastName(string 类型)，和 age(Integer 类型)，如图 13-14 所示：

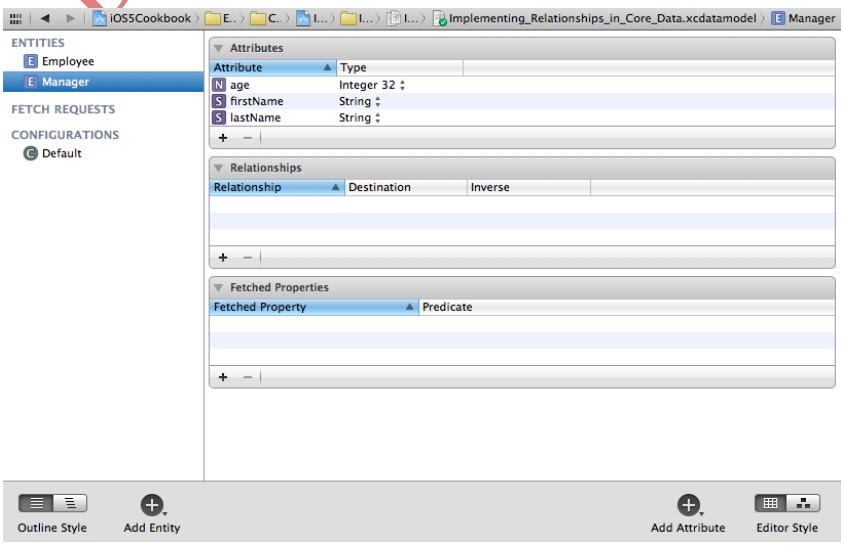

图 13-14 Manager 条目与三个属性

6、 给 manager 实体创建新的关系，通过选择列表里 manager 实体，然后按下 Relationships 复选框底部的“+”按钮（见图 13-15）。

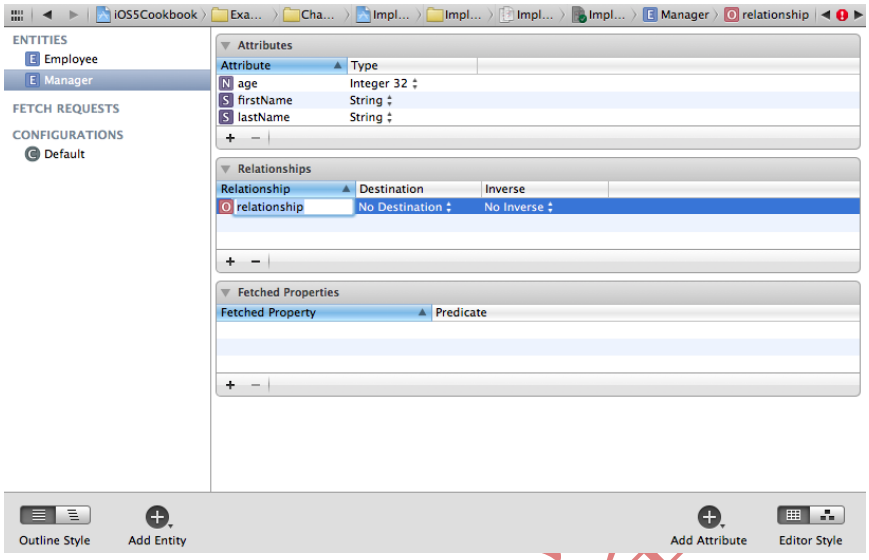

图 13-15 给 Manager 创建新的关系

7、 给新的关系的名称设置成 FKManagerToEmployees(见图 13-16)。

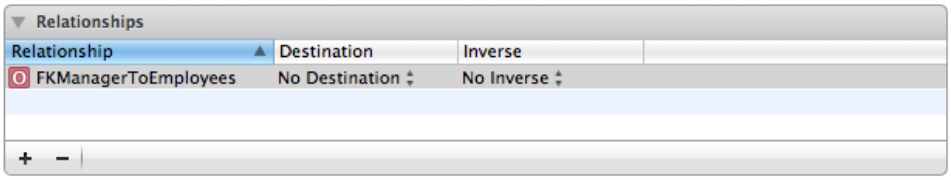

图 13-16 修改新关系名称

8、 选择 Employ 实体并为它创建一个新的关系。将此关系命名为 FKEmployeeToManager（见图 13-17）。

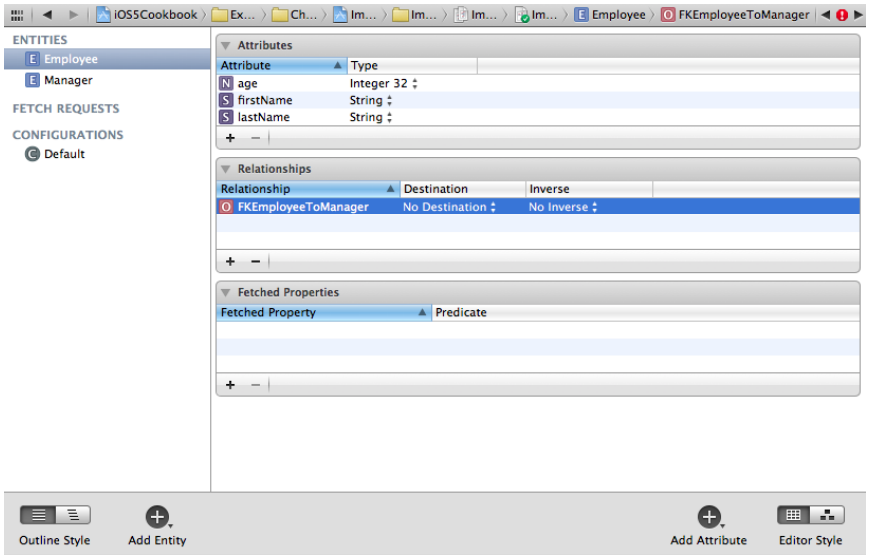

图 13-17 Changing the name of the new Employee to Manager relationship

9、 选择 Manager 实体，然后为 manager 选择 FKManagerToEmployees 关系。在 Relationships 复选框里，在 Destination 的下拉菜单里选择 Employee（因为我们想通过这种关系将 Manager 关联到 Employ 实体，将 Inverse 下拉框的值设置成 FKEmployeeToManager（因为 Employee 的 FKEmployeeToManager 关系将一个员工关联她的管理者上），并在 Data Model 观察器里点击 To-Many 的关系（见图 13.1）。结果如图 13-18 所示。

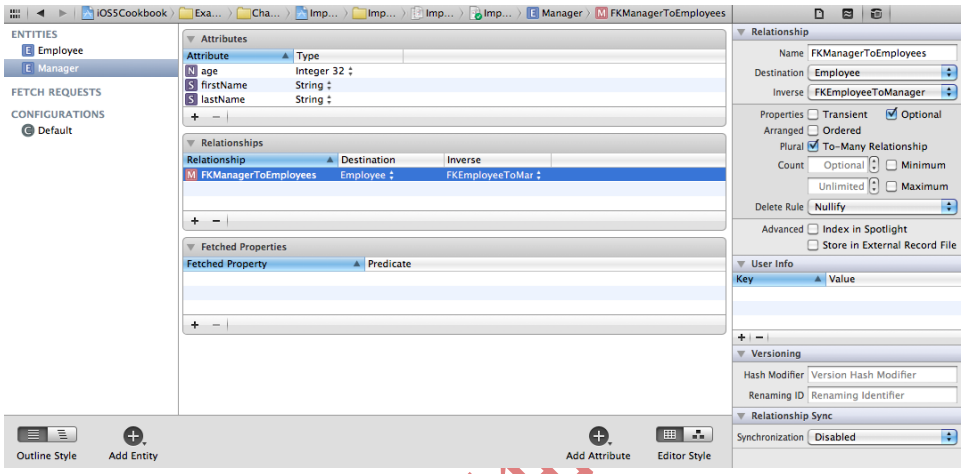

图 13-18 The Manager invese relationship established with employees

10、 选择 Employee 和 Manager 实体，选择 File→New→New File，并为你的模型创建管理对象类，如 13.2 小节的介绍。

创建完逆向的一对多的关系后，打开 Employee 实体的.h 文件：

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Manager;

@interface Employee : NSManagedObject {
@private
}
@property (nonatomic, retain) NSString * firstName;
@property (nonatomic, retain) NSString * lastName;
@property (nonatomic, retain) NSNumber * age;
@property (nonatomic, retain) Manager *FKEmployeeToManager;
@end
```

你会发现一个新的属性被添加到了这个文件里。这个属性的名字为 FKEmployeeToManager，它的类型为 Manger，意味着现在开始，假如我们要访问任何一个 Employee 类型的对象时，我们可以通过访问 FKEmployeeToManager 属性去访问指定的员工的 Manager 对象(假如有的话)。让我们看看 Manager 实体的.h 文件：

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
```

```
@class Employee;

@interface Manager : NSManagedObject {
@private
}
@property (nonatomic, retain) NSNumber * age;
@property (nonatomic, retain) NSString * firstName;
@property (nonatomic, retain) NSString * lastName;
@property (nonatomic, retain) NSSet *FKManagerToEmployees;
@end

@interface Manager (CoreDataGeneratedAccessors)

- (void)addFKManagerToEmployeesObject:(Employee *)value;
- (void)removeFKManagerToEmployeesObject:(Employee *)value;
- (void)addFKManagerToEmployees:(NSSet *)values;
- (void)removeFKManagerToEmployees:(NSSet *)values;
@end
```

FKmanagerToEmployees 属性也是为 Manager 实体创建的。这个对象是 NSSet 类型的。这就意味着 Manager 实体的任何实例的 FKManagerToEmployees 属性都可以包含 1 到 N 个 Employee 实体（一个一对多的关系：一个管理者，多个员工）。

另一个关系类型是多对多的关系。回顾一下 Manager 和 Employee 的关系，有一个多对多的关系，任何一个管理者都可以有 N 个员工并且一个员工可以有多个管理者。这样，我们可以按照创建一对多关系同样的方法，但是选择 Employ 实体后选择 FKEmployeeToManager 关系。将这个名字修改成 FKEmployeeToManagers 并点击 To-Many Relationship 复选框，如图 13-19 所示。现在的箭头是指向双向的了。

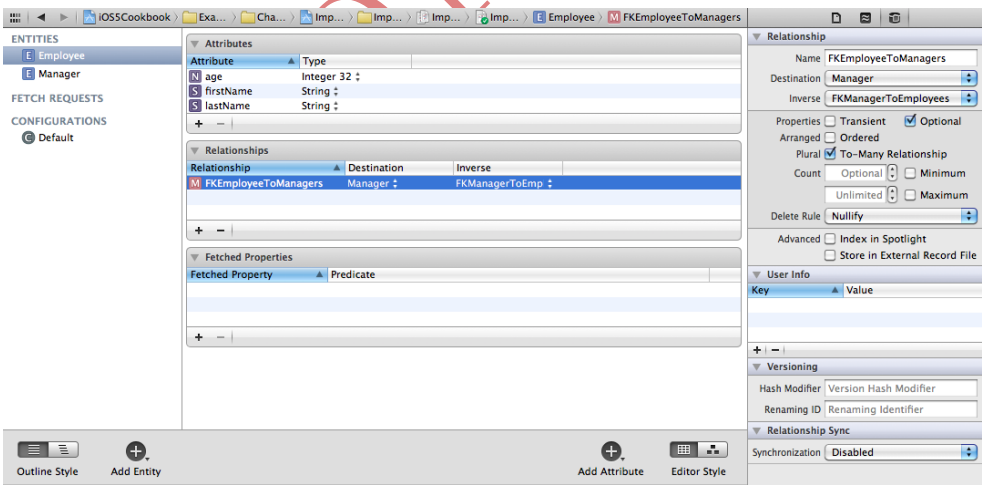

图 13-19 在 Manager 和 Employee 条目间创建更多关系

在你的代码里，对于一对多的关系，你可以简单的创建一个新的 Manager 管理对象（请参考 13.3 小节的学习如何将对象插入到一个管理对象上下文里），将它保存到管理对象上下文里，然后将创建一个几个 Employee 管理对象并也将她们并保存到上下文。现在将管理者和一个员工关联起来，将一个实例的 FKEmployeeToManager 属性值设置为 Manager 管理对象。Core Data 会为你创建它之间的关系。

点击这里访问：DevDiv.com 移动开发论坛