从零开始学iOS7开发系列3-我的地盘我做主-Cha20

说明:

本系列文章的原文及示例代码来自raywenderlich store中的iOS Apprentice 系列3教程，经过翻译和改编。

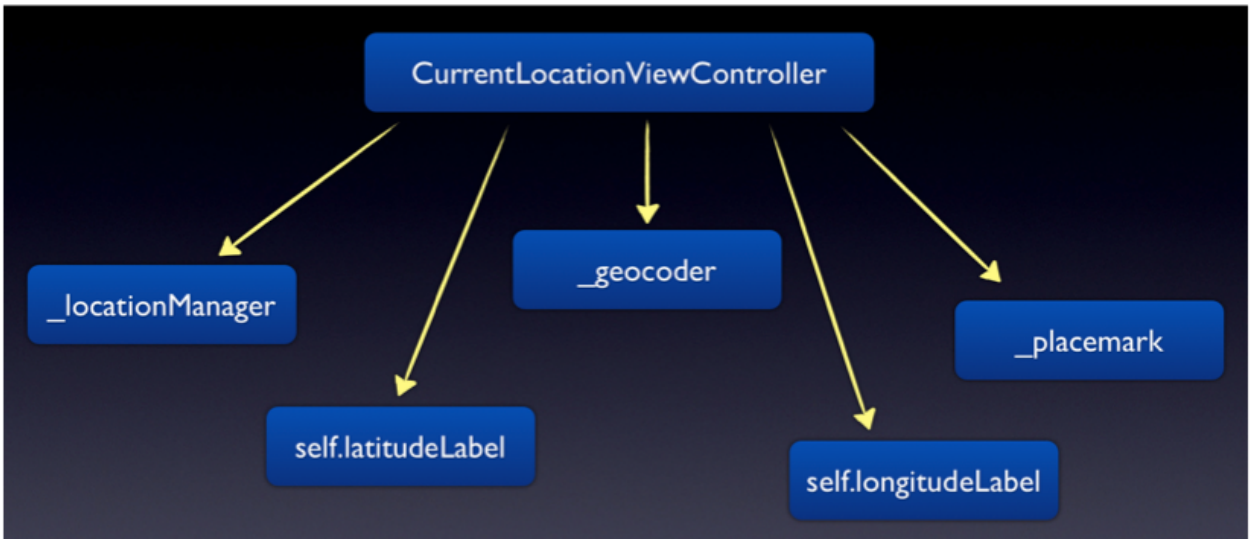
欢迎继续我们的学习。

首先让我们来点理论知识充电吧。

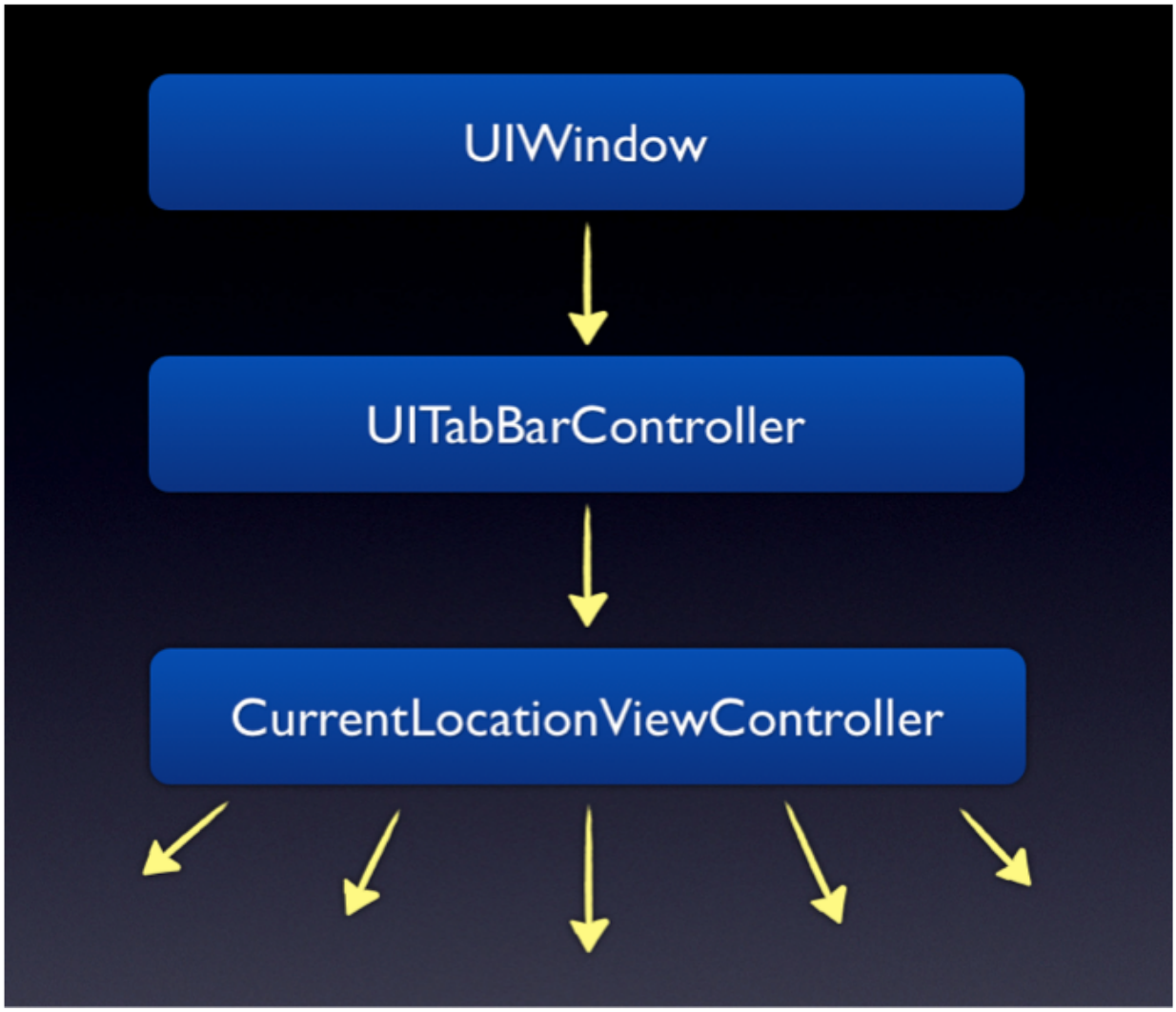
Object graph(对象图）

对象可不像修士，在深山之中云深不知处。我们的应用中有很多对象，它们需要在一起共同工作。对象之间的关系被称为object graph（对象图）。

比如在这款应用中，CurrentLocationViewController就和几个对象之间有一定的关联：



这些对象是它所拥有的实例变量和属性，或者换句话说是它所“拥有”的对象。但这只是个开始，CurrentLocationViewController本身被其它对象所拥有，比如它属于UITabBarController,而UITabBarController则属于UIWindow。



当然，到了这里，我们也仅仅看到了当前应用的对象图的一小部分。对象图显示了不同对象之间的所属关系。（不要把对象图和类层级图弄混了。类层级图显示的是对象的数据类型，而不是对象本身）。

对象的ownership（所属关系）在iOS开发中是非常重要的概念。我们需要建立一个清晰的图景，了解对象之间的所属关系，因为这些对象的生死存亡就取决于这一点！如果一个对象没有任何的owner（不被任何人需要的穷DS），那么它就会被立即deallocate（销毁！）。如果应用中还要用到这个对象，就会带来毁灭性的打击（crash!）。反过来也一样，如果某个对象有太多的owner(人人都想追的白富美），那么这个对象就会一直驻留在内存之中，这样就可能导致应用内存空间不足，直到崩溃。

每个对象都可能有超过一个owner。例如，当用户触碰Tag Location按钮时，CurrentLocationViewController会传递一个CLPlacemark对象给LocationDetailsViewController，那么视图控制器就会分享它的所有权。

我们在LocationDetailsViewController.h中使用以下语句对此进行了声明：

@property(nonatomic,strong) CLPlacemark *placemark;

通过上面的语句，LocationDetailsViewController就获得了一个新属性placemark，同时是一种strong关系。当我们把一个对象放到该属性变量时，Tag Location界面就成了该对象的共同拥有者。

对象关系的类型有两种：strong和weak。在strong强关系中，一个对象拥有另外一个对象的所有权，同时还可以和其它所有者分享。在weak弱关系中，不存在此类的ownership（所属关系）。

之前我们在outlet类型的属性变量中曾经使用过weak属性：

@property(nonatomic,weak) IBOutlet UILabel *latitudeLabel;

之所以outlet类型的属性属于weak，是因为视图控制器并非真的是它们的所有者，真实的情况是，outlet属性变量属于视图控制器的主视图。

weak弱类型在delegate中也经常用到：

@property (nonatomic, weak) id <IconPickerViewControllerDelegate> delegate;

上面的这行语句来自系列1教程中的IconPickerViewController类。图标选择器有一个代理对象，它的作用是监听在图标选择界面上可能发生的事件。

通常情况下一个对象不要拥有自己的代理，因为这样会导致ownership cycle（所属关系循环）-两个对象互相宣布自己拥有对方（虽然对两者的基情或者爱情非常赞赏，遗憾的是在代码世界中不存在婚姻系统~）。通过将delegate代理关系设置为weak类型，就可以避免这种情况的发生。

weak弱关系还可以产生很酷的副作用。比如我们对某个对象有一个weak弱引用，由于它之前的owner都已经放弃了对它的主权要求，该对象就被deallocate销毁了，此时我们的引用就会自动变为nil。这也是件好事情，因为如果不是这样的话，我们就会指向一个死对象，当我们试用使用它的时候就可能导致崩溃。（记住，在Objective-C中我们是可以向nil发送消息的，但可以这样做并不代表我们可以向根本不存在的对象发送消息）。

对象一生的故事

当我们创建了一个对象后，它会一直活着，直到不再有任何人需要它。在iOS中，使用一种所谓的retain count（引用计数）机制来记录这一切。当我们使用强关系将某个对象提供给别人的时候，retain count（引用计数）会增加，当该对象的某个owner停止使用它时，retain count（引用计数）会减少。因此，retain count（引用计数）代表某个对象owner的数量。weak弱关系不会影响retain count（引用计数）。

通常情况下我们不必担心retain count（引用计数）的问题，除了这一点：当retain count（引用计数）变成0的时候-比如当没有owner的时候-该对象就会被deallocate和delete（销毁并删除）。因此一个对象的生命长短取决于它在某个特定点上的owner数量。

下面是retain count（引用计数）机制的示例：

```
- (void)greetPerson:(NSString *)name {

NSString *text = [NSString stringWithFormat:
@"Hello, %@", name];
NSLog(@"The string is: %@", text);

}
```

通过上面的语句，我们创建了一个NSString对象，并将其保存在一个名为text的变量中。此时它的引用计数是1.这个字符串对象直到方法的结束都是有效的，因为text是一个本地变量，而本地变量只能活到方法结束的那一刻。到了那个时候，string字符串对象不再有任何的owner，它的引用计数会降到0，然后就会被无情的销毁。

再看下一个例子，_text是一个实例变量。

```
@implementation GreetingBot {
NSString *_text;
}
```

```
- (void)greetPerson:(NSString *)name {
    _text = [NSString stringWithFormat:@"Hello, %@", name];
    }

- (void)printText {
NSLog(@"The string is: %@", _text);
}
```

在这个例子中，当调用greetPerson方法的时候，会在_text实例变量中放入一个新的对象。当greetPerson结束的时候，字符串对象依然笑傲江湖，因为一个实例变量的生命周期不会局限在一个方法之中。我们可以在随后的printText方法中继续使用这个变量来显示字符串中的内容。

_text对象会活得很好，活得很久，直到：

1.GreetingBot这个类的实例变量本身被销毁。当这一幕发生时，会释放出它所拥有的所有对象。因为_text的字符串对象不存在其它的owner，此时也会被销毁。

想象一下你所生活的宇宙毁灭了会是怎样的情况？皮之不存毛将焉附？

2. 我们可以在_text中放入了一个新的数值，比如通过再次调用greetPerson方法来实现这一点。

3. 我们强制让_text = nil，这样做不会在变量中放入新的对象，但会强行释放已有的对象。

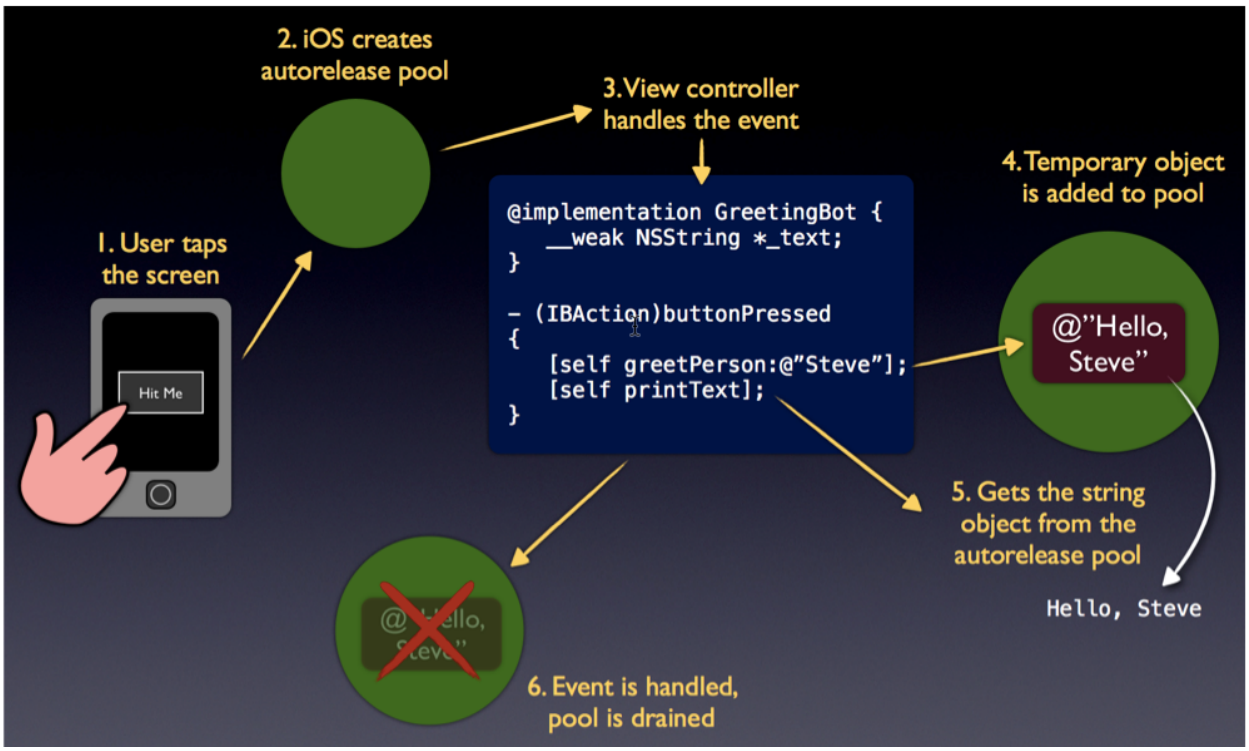
想象这样一个宇宙，有一种叫实例变量的生物，它们在亿万年的进化之中终于拥有了永恒的生命（得道？修仙？半人半机械？）。只要宇宙还在，实例变量这种生物就会快乐的活下去。但世界上没有任何幸福是永恒的，对象虽然拥有永恒的生命，但宇宙却不是（可能有亿年？百亿年？千亿年？），终有一天宇宙走到了它的终点。在那一刻，宇宙中的所有实例变量也香消玉殒。另外，实例变量虽然是永生的，但总有一些实例变量厌倦了永无休止的生命，选择自己干掉自己或者被别人干掉。

实例变量总是strong强关系的，除非我们使用__weak来特别说明。比如我们可以把字符串对象的声明改成下面的样子：

```
@implementation GreetingBot {
__weak NSString *_text;
}
```

当我们在greetPerson方法之后调用printText方法时，有可能会成功打印出所希望的字符串内容，也可能会打印出null，结果取决于我们调用它的时间。

当iOS开始处理一个新的事件（可能由用户触碰按钮，计时器或其它触发器引发）时，首先或创建一个autorelease pool（自动释放池）。这个释放池是所有不具备strong强关系的对象的临时owner。当事件结束时，自动释放池会被排空，并释放掉其中所有的临时对象。如果届时我们没有对这些对象抓住不放，这些临时对象就会被销毁。不过在此之前，也就是在当前的事件仍被处理的过程中，即便这些对象没有真正的owner，也仍然是有效的。



好吧，内存管理的这些东东听起来真的有点可怕。我们真的需要关注这些细节吗？当然不是。不过了解一点iOS内存管理的知识绝对是有好处的。只要记住一点，如果我们想在当前事件之外仍然保留一个对象，就必须使用strong强关系来保存它。如果不这样做，而这个对象又没有其它的owner,那么自动释放池就会被排空。

copy和assign属性

之前我们曾经使用weak和strong来声明变量：

@property (nonatomic, weak) IBOutlet UILabel *latitudeLabel;

@property (nonatomic, strong) CLPlacemark *placemark;

其实我们也可以使用assign或copy来声明变量。

此外，nonatomic关键字（或者是它的反义词atomic)用在multi-threading（多线程）代码中。关于多线程的问题超出了我们这个系列教程的范围。大家需要知道的是，多数情况下我们应该使用nonatomic。

如果某个变量属于基础数据类型，应该使用assign类型：
@property (nonatomic, assign) int someNumber;

我们也可以对结构体使用assign类型，因为它们同样不属于对象：

@property (nonatomic, assign) CLLocationCoordinate2D coordinate;

如果我们使用了错误的类型修饰词，那么Xcode就会毫不留情的给你错误信息：”Property with 'retain (or strong)' attribute must be of object type”。

当我们把某个属性声明成copy的时候，那么不管你要把属性给哪个对象，都会首先被复制，复制的对象会被保存，而非初始对象。

@property (nonatomic, copy) NSString *text;

这里同样建立了一个强关系，但却是使用了新复制的对象。

当我们把某个属性变量添加copy类型修饰词时，通常是用在字符串和array数组上，从而可以确保我们会获得一个独一无二的不会被改变的对象。尽管NSString和NSArray是immutable（不可变的），但在创建后仍然可以被改变，因为它们仍然有可变的子类。因此以下情况是可能发生的：

```
// on someObject:
@property (nonatomic, strong) NSString *text;
…

NSMutableString *m = @"Strawberry";
someObject.text = m;
// at this point, someObject.text is @"Strawberry"

[m appendString:@" and banana”];

// now both m and someObject.text are @"Strawberry and banana”
```

因为someObject的text属性是strong类型而非copy类型的，因此someObject.text和m都指向同一个对象。如果m从@“Strawberry”变成@“Strawberry and banana”，那么someObject.text也会发生同样的变化。以上代码是有效的，因为NSMutableString是NSString的子类，因此它可以被当做NSString来使用-但它同时也能做更多的事情，而someObject可能不需要这样的事情发生。如果我们把strong更改成copy，那么@“Strawberry”就会被拷贝到一个新的NSString（而不是一个可变的字符串！），那么对m所做的改变就不会影响someObject。

有时我们还会看到另外一个修饰词retain。在iOS5之前，我们使用retain对一个强关系进行手动内存管理。在下一部分关于Core Data的内容中，我们会使用Xcode为数据模型对象创建类，此时仍然会在属性定义中放入关键词retain。retain实际上是strong的同义词，它的工作原理也是完全相同的。

Manual memory management（手动内存管理）

在iOS5之前，开发者需要手动进行内存管理，包括对象ownership的获取和释放。这就是所谓的manual memory management（手动内存管理）,想到这个名词不禁让人菊花一紧！
虽然现在有了ARC（Automatic Reference Counting，自动引用计数）的概念，了解一点手动内存管理的知识还是有必要的。

如果我们在手动的retain和release对象中出了差错（这个是很常见的事情），那么要吗我们会得到死对象，要吗会得到拥有永恒生命的对象。后面这种情况被称为memory leak（内存泄露）。应用中很可能会“泄露”不能被回收再利用的内存。如果在应用中有太多的内存泄露，那么到某个时候可用内存就会消耗殆尽，直到彻底崩溃。

虽然内存泄露的情况比较少发生，但仍然有可能会出现。如果我们对对象的保持超出了所需，那么应用就是在占用它本应释放的内存。记住我们使用strong关系创建的对象会永远停留在内存中，除非我们使用其它对象将其替代，比如将其指针设置为nil,或者是销毁owner对象。

重要提示：
如果我们不打算继续使用一个对象，就需要将其指针设置为nil。通过这种操作，可以销毁对象（如果你是唯一的owner)，并返回它所占用的内存空间。但如果你一直保留该对象，那么系统就永远无法回收再利用它所占用的内存，这样就制造了内存泄露。

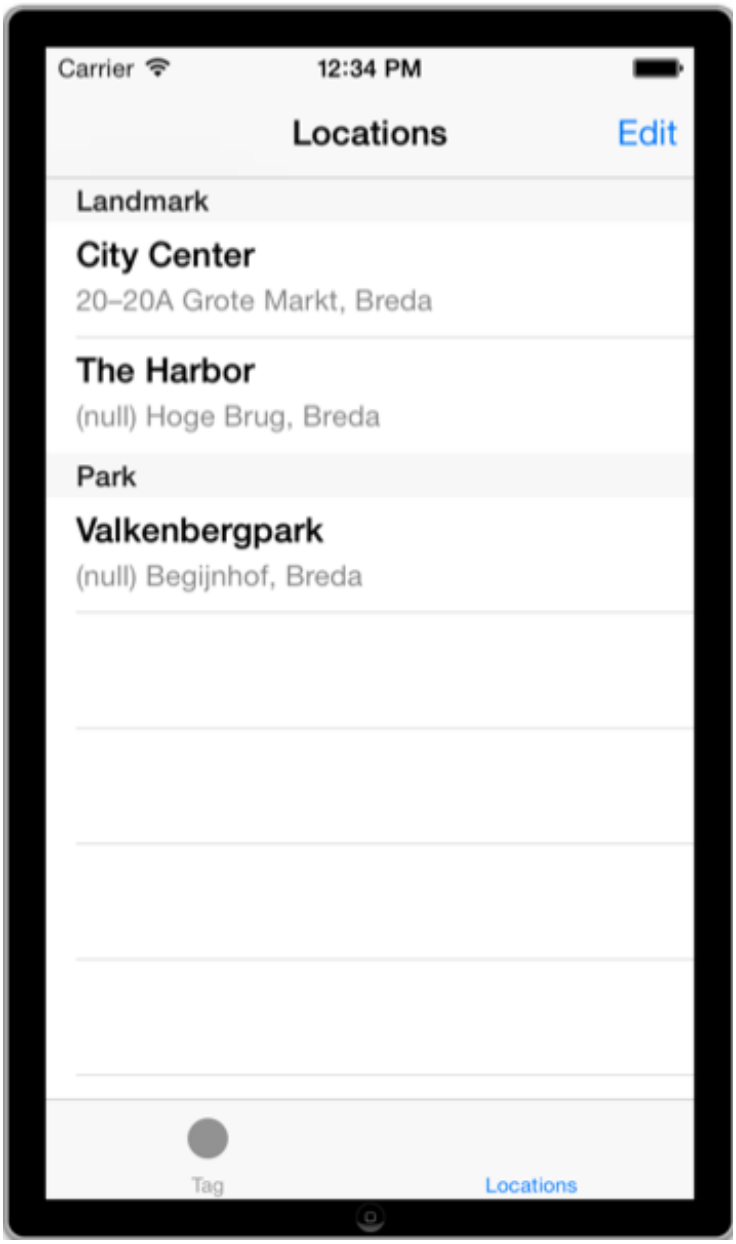
虽然在ARC时代我们无需再操心手动内存管理的事情，但仍然需要考虑对象之间的关系（它们是strong或weak），当我们使用完某个对象时必须将指针设置为nil，以免它永远停驻在内存之中。

在下一个系列的教程中，我们会简单解释下如何进行手动内存管理。因为有时候我们会需要处理一些史前时代的旧代码，或者需要阅读一些远古时代的图书和示例代码。

使用Core Data来保存地理位置信息

到目前为止，我们的应用已经可以获取用户当前位置所对应的GPS坐标信息。同时我们还创建了一个界面，用户可以在这里”tag”（标记）位置，其中可以输入描述信息，并选择一种分类。随后，我们还将允许用户选择一个照片。而下一个要实现的特性则是让应用记住用户曾经标记过的位置信息，并在列表中将其展示。

Locations界面的效果将会如下所示：



我们应该想办法将所获取的地理位置信息进行持久化保存，这样即便应用被强关也会被记住。之前我们创建了一个遵循NSCoding协议的数据模型对象，并将其使用NSKeyedArchiver保存在.plist文件中。这种方式当然是可行的，不过在这里我们将要学习一个新的框架：Core Data。

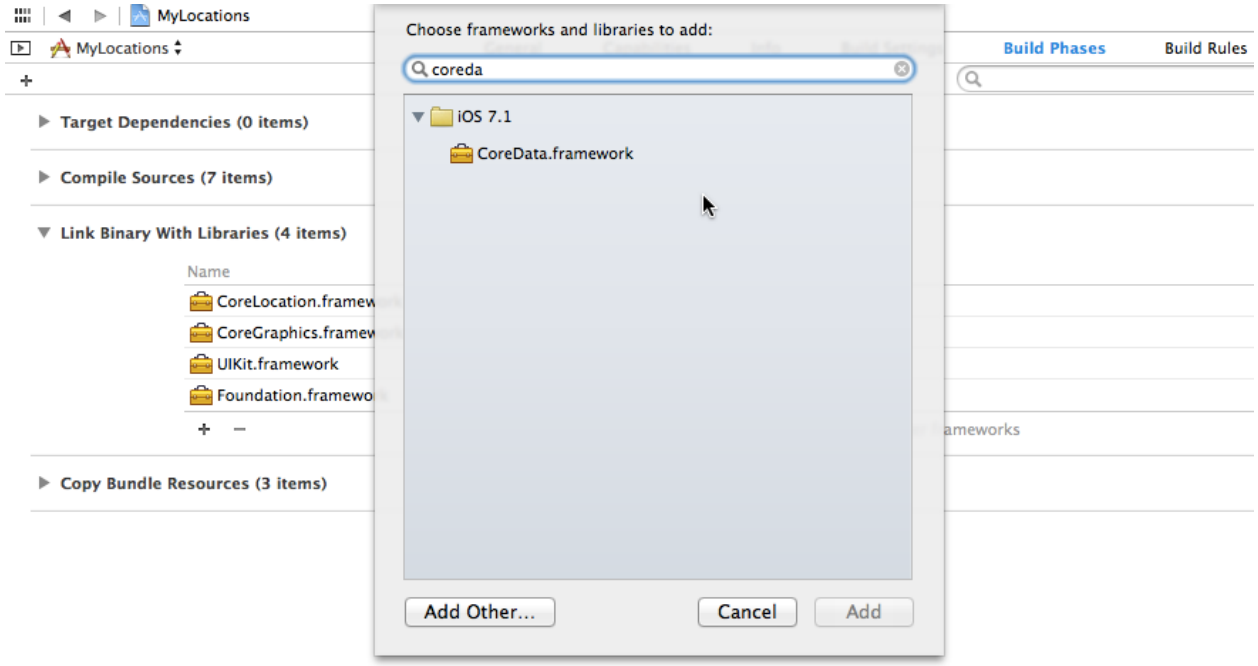
Core Data是iOS应用中的一个对象持久化保存框架。
如果你直接看官方的文档，可能会被吓退三千里不知所终。这就好比一个后天二重的修士突然接触到元婴期老怪，不吓死也得吓尿。其实没那么夸张，Core Data的工作原理是很简单的。我们刚刚了解过，如果对象没有被其它owner对象引用，就会被销毁。此外，当应用被关停的时候，所有对象都会被销毁。有了Core Data之后，我们就可以将某些对象指定为不朽级别的，这样它们就会被保存在data store(数据存储）中。即便某个managed object的所有引用都消失，实例被销毁，它的数据仍然会被保存在Core Data中，我们可以随时重新获取。

如果你之前有接触过数据库的概念，那么可能会把Core Data看成是一个数据库，但这样想其实是有些误导的。在很多方面这二者都有共性，但Core Data是关于如何保存对象的，而不是数据库中

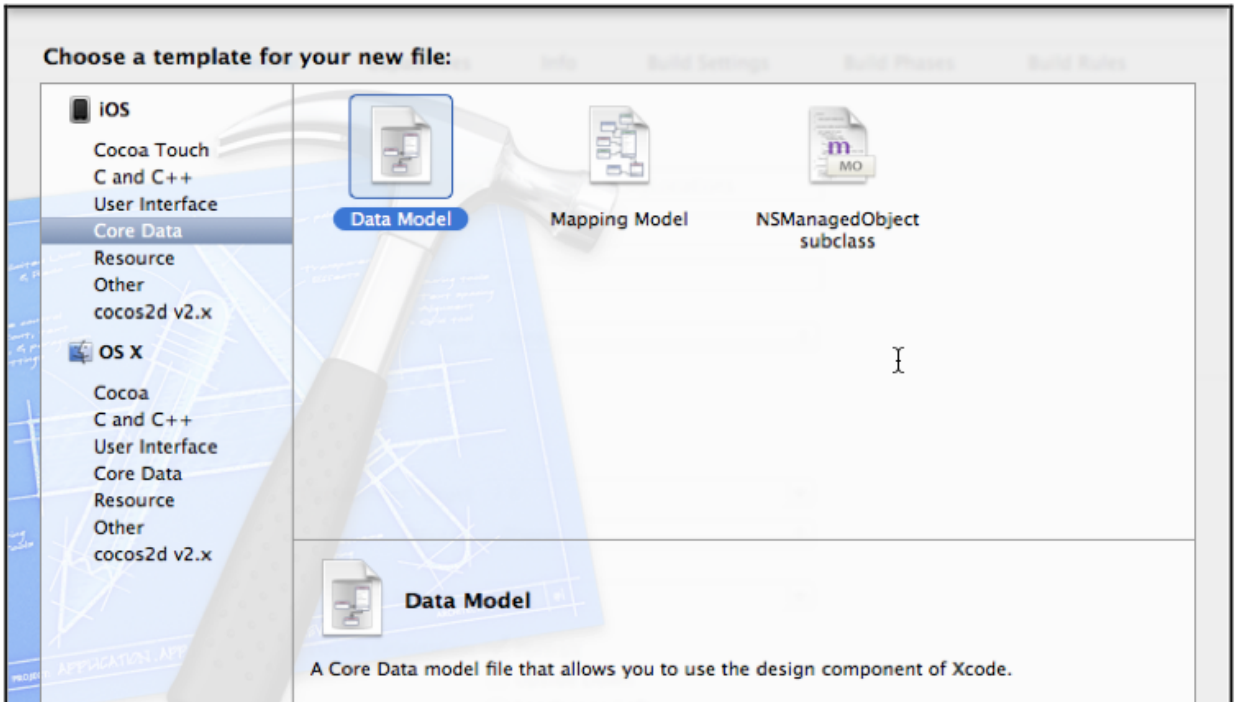
的关系表。它只是保存特定对象中数据的另外一种方式，即便这些对象被销毁或是应用被终止也是如此。

好了，接下来就让我们小试牛刀，看看Core Data的威力吧~

在Xcode中，进入Build Phasesg界面，切换到Linked Framework and Libraries部分，在项目中添加CoreData.framework。



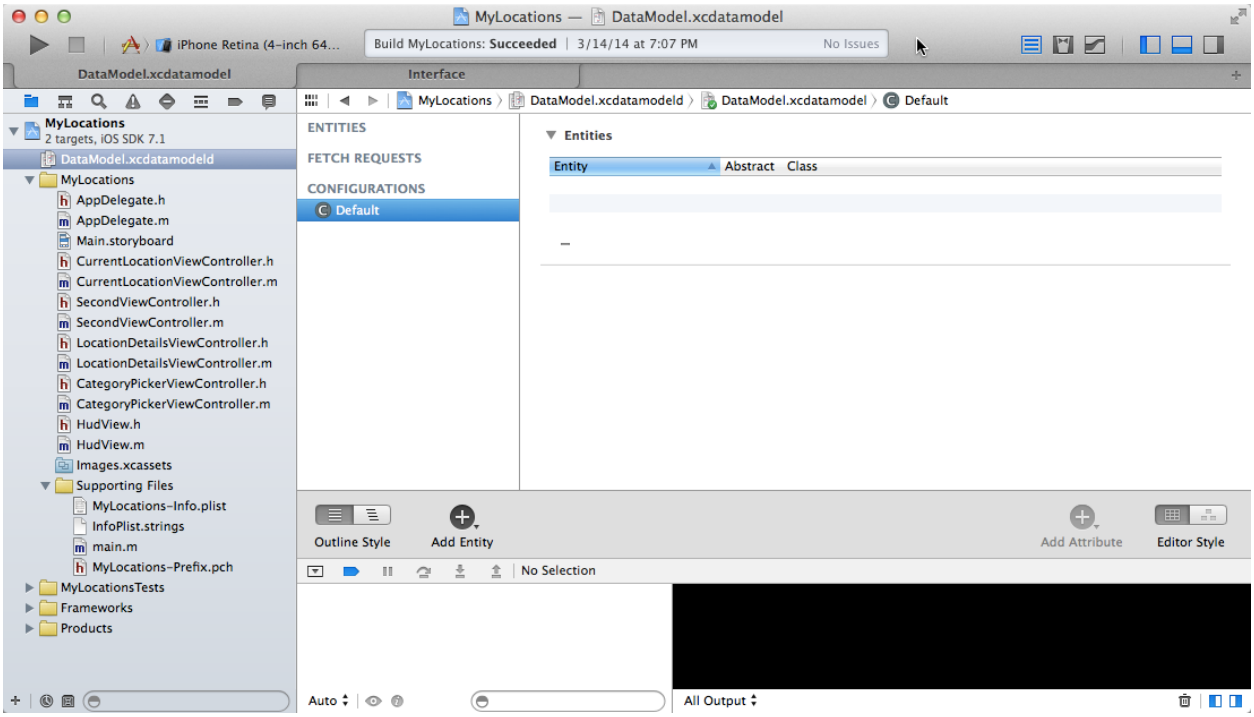
使用Core Data需要用到数据模型。它是一个特殊的文件，我们需要将其添加到项目中，用来描述想要保存的对象。这些对象被称为managed objects，即便我们显示将其删除，也仍然会在数据存储中保留其中的数据。
在项目中添加一个新的文件，选择Core Data部分的Data Model模板：



将其命名为DataModel。

此时会在项目中添加一个新的文件，DataModel.xcdatamodeld。

点击DataModel.xcdatamodeld打开Data Model 编辑器：



对某个希望使用Core Data来管理的对象，我们都需要添加一个entity（实体）。一个entity用来描述对象所拥有的数据字段。换句话说，它和一个类的作用是相同的，只不过仅限于在Core Data的数据存储中使用。（如果你之前接触过SQL数据库的概念，那么可以把entity看做一个table数据表）。

在我们这个应用中只有一个entity，就是Location，其中会存储用户所标记的地理位置的所有属性。每个Location中都会保存以下的数据：

1.经度和纬度
2.placemark（街道地址）
3.地理位置被标记的日期
4.描述
5.类型

上面这些内容都是Tag Location界面中的项目，照片除外。Photos相对数值来说太大，可能会占用若干MB的存储空间。即便Core Data可以处理大的blobs类型的数据，但通常也会将照片保存在应用的Document文件夹的单独文件下。关于这一点后来会详细说明。

好了，今天的学习就到此结束吧，下一课的内容中会详细说明如何添加entity，并在随后的课程中讲解Core Data的更多知识，敬请期待~

今日福利，小清新美女一枚