

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。

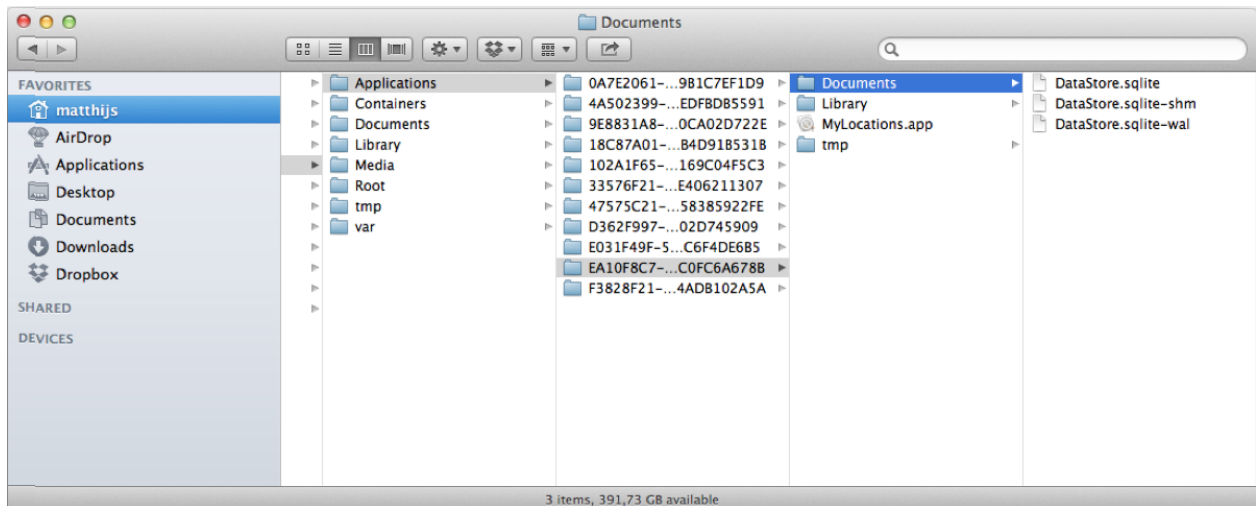
版权归作者所有，本系列教程仅供学习参考使用，感兴趣的朋友建议购买原英文教程教程(The iOS Apprentice Second Edition: Learn iPhone and iPad Programming via Tutorials!)

购买链接:

<http://www.raywenderlich.com/store>

欢迎继续我们的学习。

当我们初始化持久化保存的coordinator时，也给数据库文件指定了一个路径。该文件的名称是DataStore.sqlite，它依然位于应用的Documents文件夹。当我们从Finder中切换到~/Library/Application Support/iPhone Simulator，然后找到包含了MyLocations应用的文件夹



DataStore.sqlite-shm和-wal文件也是数据存储的一部分。

Tip:

如果在Finder中找不到Library文件夹，可以打开Terminal，然后键入以下命令：

```
chflags nohidden ~/Library
```

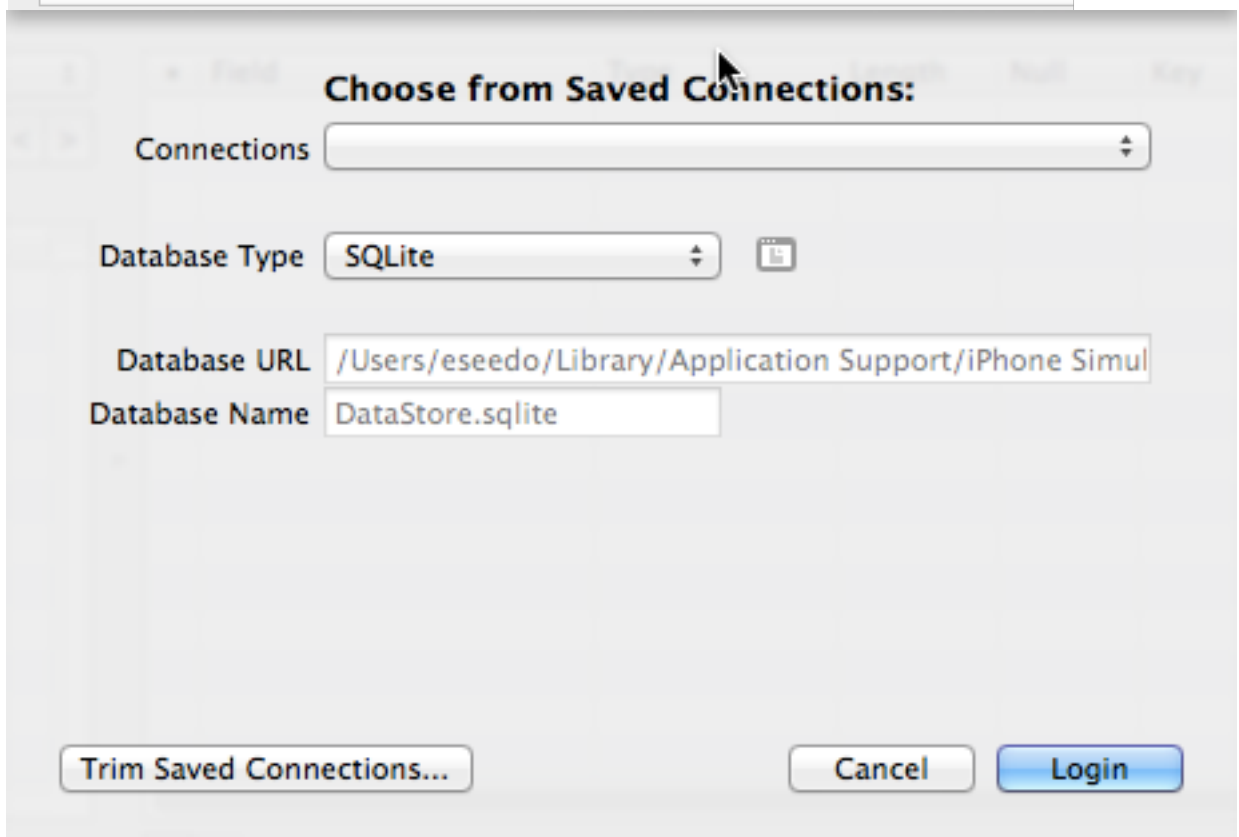
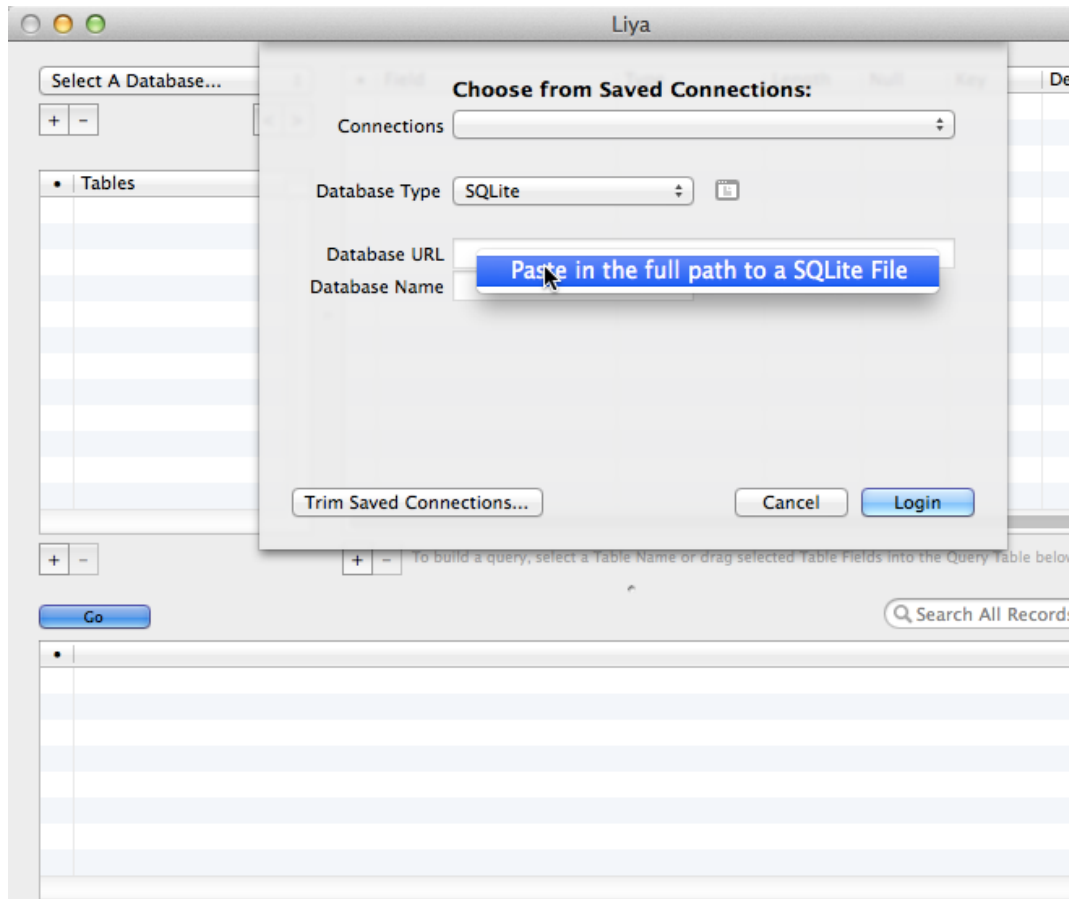
此外，也可以在Finder中打开Go菜单，然后按住Alt/Option在下拉菜单中显示Library文件夹。

此时数据库中仍然是空的，因为我们还没有在其中保存任何对象，不过我们还是可以看看其中的内容。

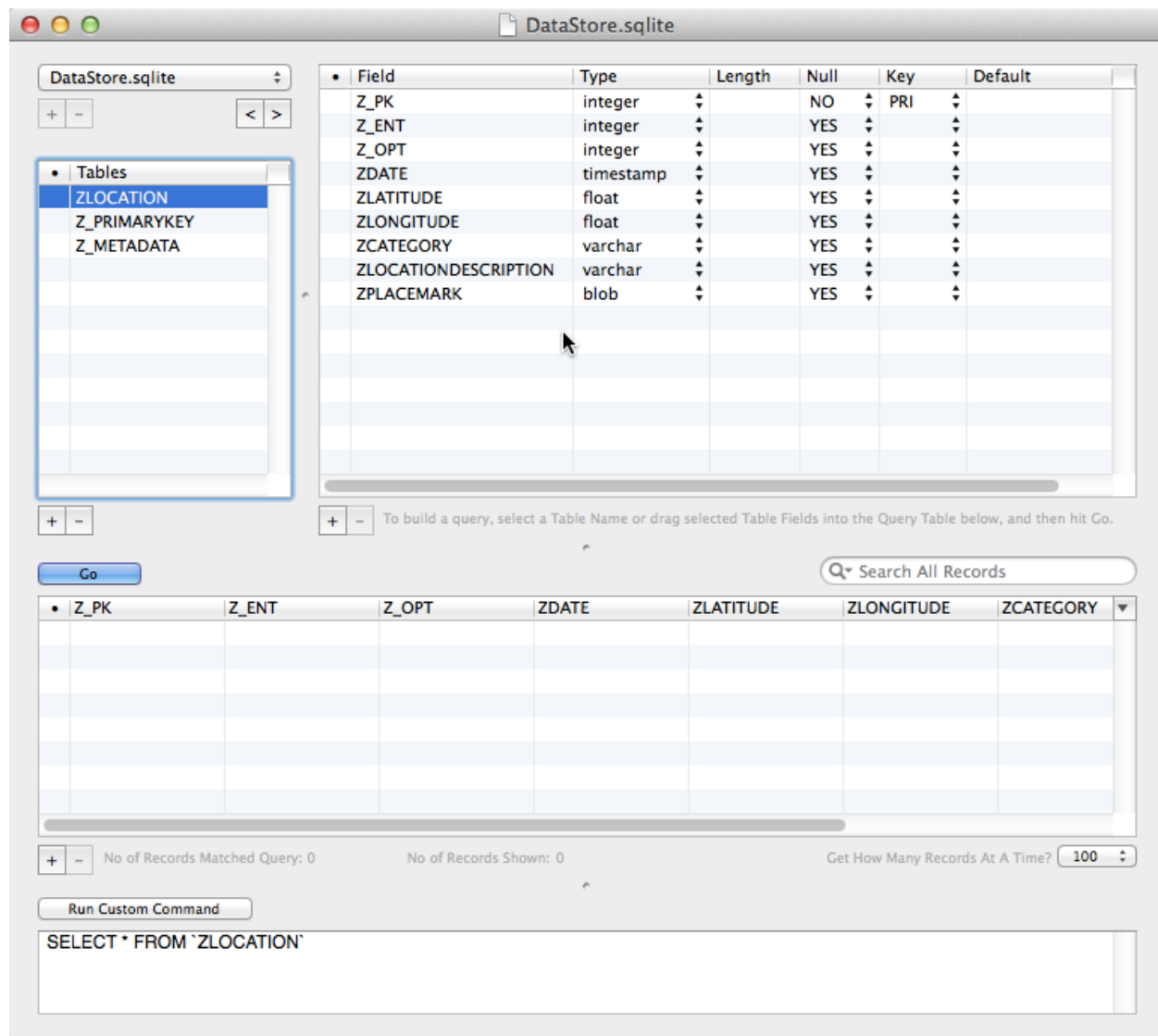
我们可以通过一些方便（免费!!!）的图形化界面工具来查看SQLite数据库。这里我们可以先用Liya来查看数据存储文件。可以从Mac App Store下载，也可以从这里下载：

<http://www.cutedgesystems.com/software/liya/>

启动Liya，它会要求提供一个数据库连接。在Database Type下面选择SQLite。Database URL就是电脑商保存DataStore.sqlite文件的文件夹。只需要把iPhone Simulator/.../Documents文件夹从Finder中拖到这个文本域就可以了。Database Name处应该填上DataStore.sqlite。（注意：Database URL部分不要包含了路径中的DataStore.sqlite部分）。



点击Login继续，此时界面如下所示：



其中ZLOCATION这个表用来保存Location相关的对象。虽然当前它的内容还是空的，但我们可以从右侧看到对应的列名：ZDATE,ZLATITUDE，等等。Core Data添加了它自己的列和表（使用Z_前缀）。当然，我们不会用这个工具来手动修改数据库的内容，不过有时候使用这样的可视化工具可以方便查看事情的进展情况。后面当我们添加了新的Location对象时还会使用Liya来查看。

注意：

Liya只是可选工具之一，还可以使用SQLiteStudio来查看<http://sqlitestudio.pl>。当然，你可以在Mac App Store中使用sqlite来搜索更多付费或免费的工具。

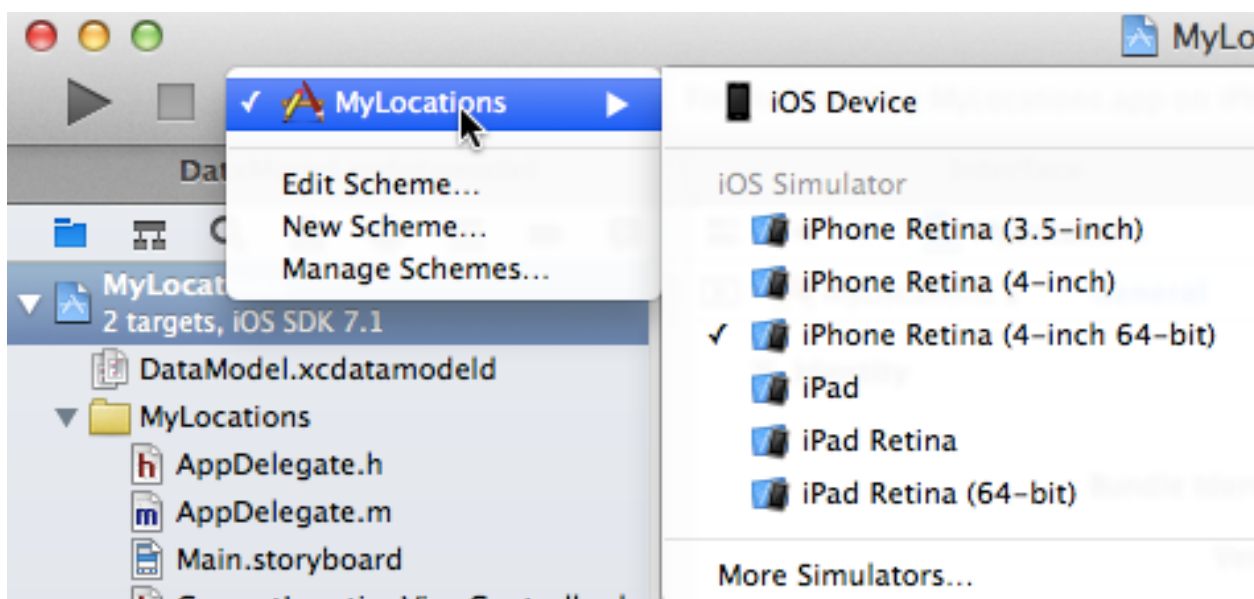
还有另外一种有用的工具可以帮助诊断使用Core Data所遇到的问题。通过在应用上设置一个特殊的flag标志，就可以看到Core Data和数据存储对话所使用的SQL语句。即便你完全看不懂SQL语

言，这些信息仍然有用。至少我们可以知道Core Data是否在处理一些事情。为了启用这一工具，我们必须编辑项目的scheme。

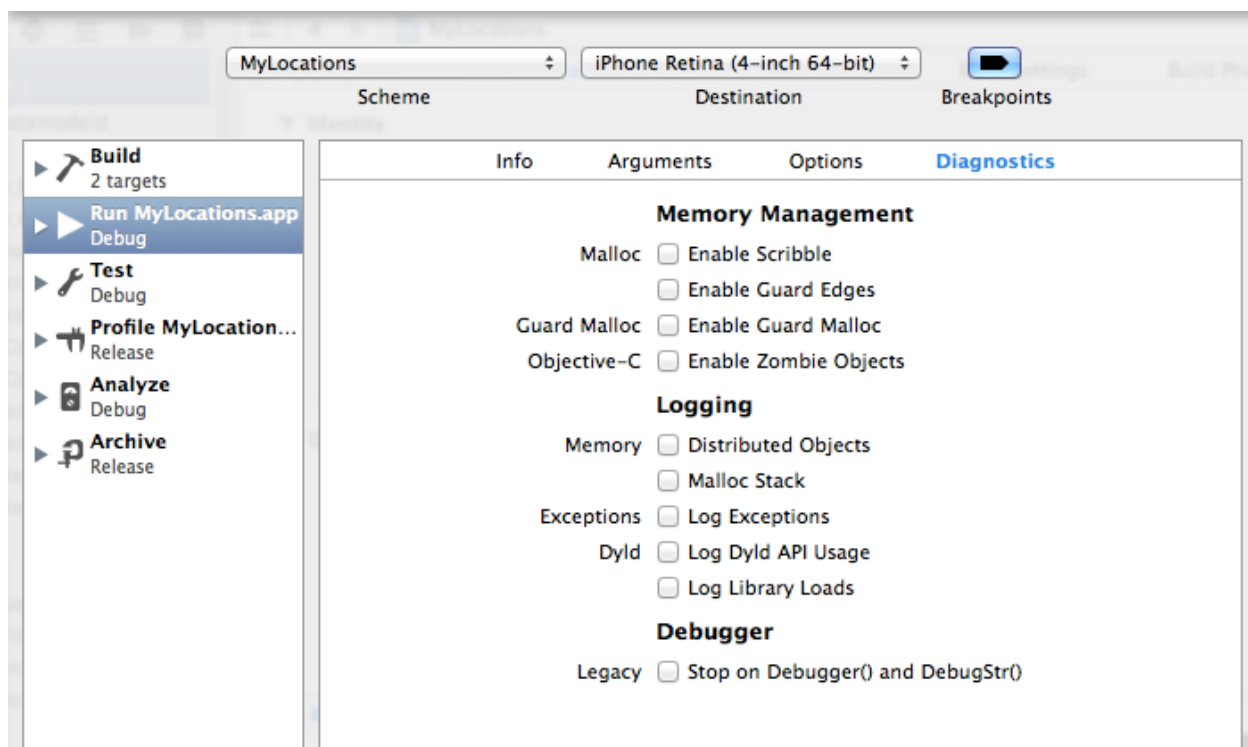
在Xcode中，我们使用Scheme对项目进行配置。

一个scheme(方案) 是用于编译运行应用的一系列设置。标准项目只有一种scheme，但是我们也可以添加其它scheme，当项目规模变大的时候会很有用。

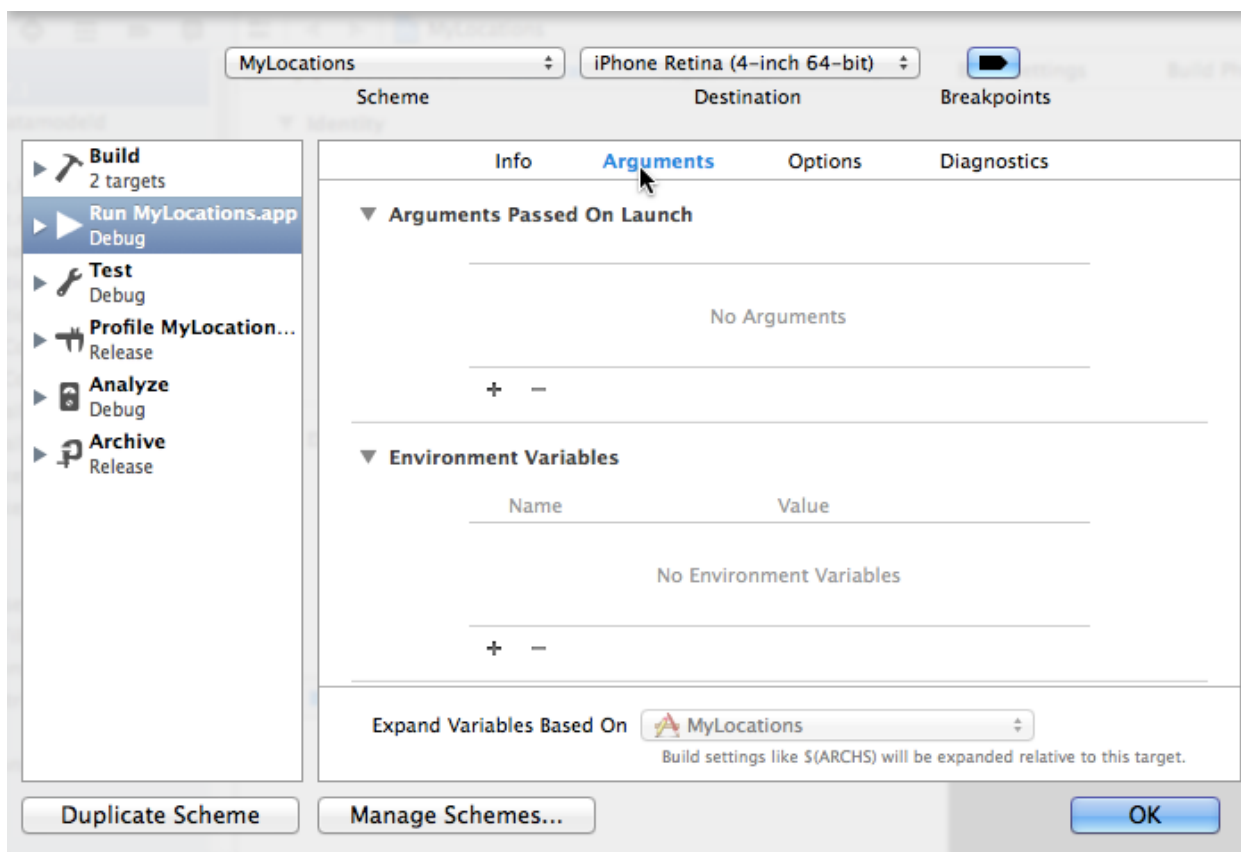
在Xcode中点击界面顶部MyLocations> iPhone Retina(4-inch)栏的左边，然后选择Edit Scheme...



此时会显示下面的界面：

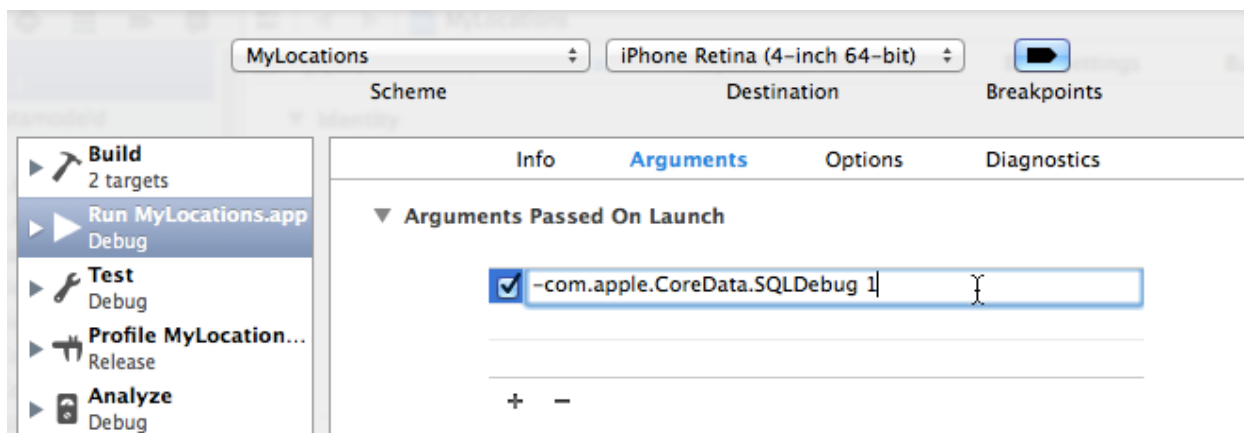


点击左侧的Run MyLocations选项，然后选择右侧选项卡中的Arguments选项：



在Arguments Passed On Launch部分，添加以下内容：

-com.apple.CoreData.SQLDebug 1



点击OK关闭此对话框，然后运行应用。

当应用启动后，可以在debug区域看到类似下面的信息提示：

```
2014-04-23 13:48:07.902 MyLocations[2395:60b] CoreData: annotation: Connecting to sqlite database file at "/Users/eseedo/Library/Application Support/iPhone Simulator/7.1-64/Applications/FBB83454-03B9-4BFA-93F1-E7B4E1047372/Documents/DataStore.sqlite"
2014-04-23 13:48:07.925 MyLocations[2395:60b] CoreData: sql: SELECT TBL_NAME FROM SQLITE_MASTER WHERE TBL_NAME = 'Z_METADATA'
2014-04-23 13:48:07.932 MyLocations[2395:60b] CoreData: sql: pragma journal_mode=wal
2014-04-23 13:48:07.933 MyLocations[2395:60b] CoreData: sql: pragma cache_size=200
2014-04-23 13:48:07.933 MyLocations[2395:60b] CoreData: sql: SELECT Z_VERSION, Z_UUID, Z_PLIST FROM Z_METADATA
```

以上信息就是来自Core Data的debug输出。这些信息的具体内容现在还不重要，不过至少表明Core Data已经成功的和数据库建立了连接。太棒了！

保存位置信息

我们已经成功的初始化了Core Data，并且将NSManagedObjectContext传递给了Tag Location这个界面。接下来我们需要在用户触碰了Done按钮的时候，让该界面将心的Location对象放到数据存储中。

在Xcode中切换到LocationDetailsViewController.m，在顶部添加以下代码：

```
#import "Location.h"
```

然后添加一个新的名为_date的实例变量：

```
@implementation LocationDetailsViewController
```

```
{
    NSString *_descriptionText;
    NSString *_categoryName;
    NSDate *_date;
}
```

之所以要添加这个变量，是因为我们需要将当期的日期保存到新的Location对象。

更改initWithCoder:方法的代码来初始化这个新的变量：

```
-(id)initWithCoder:(NSCoder *)aDecoder{
    if((self = [super initWithCoder:aDecoder])){
        _descriptionText = @"";
        _categoryName = @"No Category";
        _date = [NSDate date];
    }
    return self;
}
```

```
}
```

在viewDidLoad方法中，更改设置dateLabel文本的代码行为：

```
self.dateLabel.text = [self formatDate:_date];
```

接下来更改done:方法的代码为：

```
-(IBAction)done:(id)sender{

    HUDView *hudView = [HUDView hudInView:self.navigationController.view animated:YES];
    hudView.text = @"Tagged";

    //1
    Location *location = [NSEntityDescription insertNewObjectForEntityForName:@"Location"
inManagedObjectContext:self.managedObjectContext];
    //2
    location.locationDescription = _descriptionText;
    location.category = _categoryName;
    location.latitude = @(self.coordinate.latitude);
    location.longitude = @(self.coordinate.longitude);
    location.date = _date;
    location.placemark = self.placemark;

    //3
    NSError *error;
    if(![self.managedObjectContext save:&error]){
        NSLog(@"Error: %@",error);
        abort();
    }

    [self performSelector:@selector(closeScreen) withObject:nil afterDelay:0.6];
}
```

让我们看下以上代码的作用吧：

1.首先我们创建了一个新的Location对象。这和以前我们创建其它对象的方式有所不同。如果Location是个常规的NSObject，那么只需要使用类似[[Location alloc]init]的方式就可以创建一个新的实例了。不过此处它是一个Core Data管理的对象，因此需要用另一种方式来创建。

这里我们需要请求NSEntityDescription类为你的entity向managed object context中插入一个新的对象。看起来这里的代码有点奇怪，不过在Core Data中就是这么实现的。字符串@"Location"是之前我们在数据模型中所添加的entity名称。

2.一旦我们创建了Location对象，就可以像使用其它对象一样来使用它。

这里我们设置了它的多种属性。注意我们使用@()这种方式将经度和纬度信息转换成了NSNumber对象。对于CLPlacemark对象，我们不需要做任何特殊处理。

3.现在我们已经有了一个新的Location对象，其中的属性就是用户在界面中所输入的信息，不过如果此时我们查看数据存储，会发现其中没有任何对象。原因很简单，我们并没有将context保存

到其中。对于任何要添加到context中的对象，或是任何更改了其中内容的managed对象，除非我们将其保存到数据存储之中，其中的信息并不会持久化保存。

关于pass-by-reference

save:方法中使用了一个参数&error。这里的&意味着save:方法将把操作结果保存到一个NSError对象中，然后将该对象保存到error变量中。这就是所谓的output parameter或是pass-by-reference。

大多数情况下，参数的作用是将数据信息提供给一个方法，但output parameter的工作原理则是相反的。方法通常只能返回一个数值，但有时候我们需要向方法调用者返回超过一个的数值。在这种情况下，我们就需要用到output parameter

save方法会返回一个BOOL值来表示相关操作是否成功。如果不是，它会将错误信息保存到NSError对象中。这种方式的操作在iOS 开发中会经常碰到。

需要注意的是，在使用output parameter的时候，一定不要忘了&，否则Xcode就会给出类似下面的错误提示：

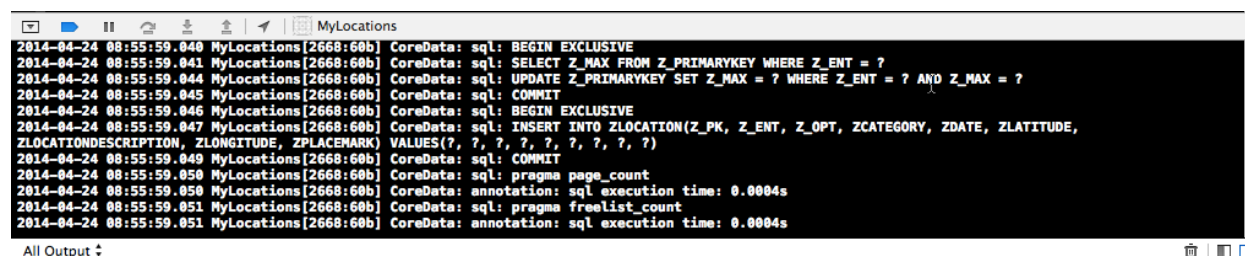
“Incompatible pointer types sending 'NSError *__strong' to parameter of type 'NSError *__autoreleasing *’”

实际上Xcode想说的是，你这个二货，连&都忘了敲。

&操作符实际上提供的是变量的地址，因此&error是指向error变量的指针，而error是指向NSError*的指针。换句话说，&error是指向一个指针的指针，或者说NSError**。好吧，你击败了我，指针！

编译运行应用，标记一个地址，输入描述信息，然后触碰Done按钮。

如果一切顺利的话，Core Data会友情提供以下反馈信息：



```
2014-04-24 08:55:59.040 MyLocations[2668:60b] CoreData: sql: BEGIN EXCLUSIVE
2014-04-24 08:55:59.041 MyLocations[2668:60b] CoreData: sql: SELECT Z_MAX FROM Z_PRIMARYKEY WHERE Z_ENT = ?
2014-04-24 08:55:59.044 MyLocations[2668:60b] CoreData: sql: UPDATE Z_PRIMARYKEY SET Z_MAX = ? WHERE Z_ENT = ? AND Z_MAX = ?
2014-04-24 08:55:59.045 MyLocations[2668:60b] CoreData: sql: COMMIT
2014-04-24 08:55:59.046 MyLocations[2668:60b] CoreData: sql: BEGIN EXCLUSIVE
2014-04-24 08:55:59.047 MyLocations[2668:60b] CoreData: sql: INSERT INTO ZLOCATION(Z_PK, Z_ENT, Z_OPT, ZCATEGORY, ZDATE, ZLATITUDE,
ZLOCATIONDESCRIPTION, ZLONGITUDE, ZPLACEMARK)
VALUES(?, ?, ?, ?, ?, ?, ?, ?)
2014-04-24 08:55:59.049 MyLocations[2668:60b] CoreData: sql: COMMIT
2014-04-24 08:55:59.050 MyLocations[2668:60b] CoreData: sql: pragma page_count
2014-04-24 08:55:59.050 MyLocations[2668:60b] CoreData: annotation: sql execution time: 0.0004s
2014-04-24 08:55:59.051 MyLocations[2668:60b] CoreData: sql: pragma freelist_count
2014-04-24 08:55:59.051 MyLocations[2668:60b] CoreData: annotation: sql execution time: 0.0004s
```

上面显示的就是Core Data用于向数据库中保存新的Location对象所使用的SQL语句。

再次打开Liya，然后更新ZLOCATIONS这个表的内容（点击Go按钮），可以看到类似下面的信息：

[illegible]

save:方法会返回NO，而我们会调用**abort()**函数。正如它的名字所暗示的，**abort()**函数会立即干掉我们的应用，然后让用户回到iPhone的主界面上。这一点对于PC用户来说似乎没什么，但是对移动应用的使用者来说简直就是晴天霹雳，因此我们是不推荐这样操作的。

好消息是，当我们保存未遂时，**Core Data**只会提供一个错误。当然，我们在开发过程中会消灭所有的bug，这样用户就不会感觉到了，这样不好吗？问题是，我们很难捕捉到所有的bug，有些bug非常狡猾，让人防不胜防。

不幸的是，如果**Core Data**出现了错误，除了崩溃外几乎没有其它解决方法。如果数据信息出错，我们就会被无效数据所困扰。因为如果应用继续进行，那么错误就会滚雪球一样被放大。即便冒着让应用崩溃的危险，我们也不应该让用户的数据受到污染。

那么，面对**Core Data**这个恼人的问题，我们究竟该如何处理呢？

且听下回分解~

虚拟现实技术是未来的一大热门，我已经订购了Oculus Rift，7月到货，到时候会跟大家分享虚拟现实应用开发的信息，请期待。



