

## 从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter10

版权声明：

原文及示例代码来自raywenderlich store中的iOS Apprentice 系列2教程，经过翻译和改编。

版权归原作者所有，本系列教程仅供学习参考使用，感兴趣的朋友建议购买原教程(<http://www.raywenderlich.com/store/ios-apprentice>)。

欢迎继续我们的学习。

接连几天都在接触新的概念，可能你已经觉得有点无力推进的赶脚了。如果你在任何一处遇到困难了，有两种选择，一种是不纠缠在细节上继续前进，一种是从上一部分开始看。两种方式都可以，选择哪种取决于你的个人偏好和性格。个人的建议是先继续推进到看完全部，然后回过头来再看一遍。毕竟有些东西属于熟能生巧，第一遍看不懂实属正常，不要死磕。

到上一章内容为止，我们已经完成了这款应用的一大步，也就是让用户得以输入自己的待办事项。不过通常来说此类应用应该具备三种功能：

- 1.添加新的项目（已经搞定了！）
- 2.删除现有的项目（已经搞定了！）
- 3.编辑现有的项目（貌似还没有搞定。。。）

怎么来实现对现有项目的编辑呢？我们完全可以创建一个全新的Edit Item界面，不过这个界面的工作原理和现在的Add Item界面几乎完全相同。唯一的区别在于，一个在打开时内容时全空的，一个已经有现有的内容了。

因此，我们可以考虑重用Add Item界面，然后让它可以用来编辑现有的ChecklistItem对象。当用户触碰done按钮的时候，让应用更新对象的文本内容，并通知代理对象它需要更新相应table view cell的标签内容了。

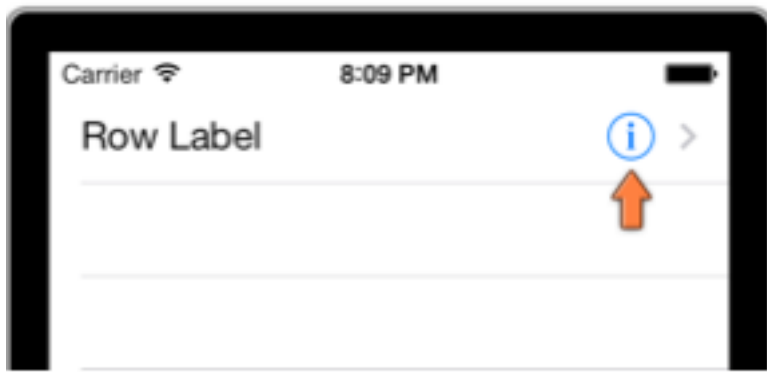
思考：如何让现在的Add Item界面可以编辑已有的项目？

答案：

- 1.该界面需要被重新命名为Edit Item
- 2.我们需要给它提供一个已有的checklistItem对象
- 3.我们需要在文本域中显示现有项目的文本内容
- 4.当用户触碰done的时候，更新现有项目的内容，而不是添加一个新的项目。

好吧，看起来有一大堆麻烦要解决。比如，用户如何打开Edit Item界面？在很多应用中，可以通过触碰项目所在的行来打开编辑界面，而这里触碰行的作用被设定为开启关闭勾选标志。为了解决这一问题，我们需要对UI做一个小小的调整。

如果某个行被赋予了两个功能，那么标准的做法就是，对第二个任务提供一个detail disclosure button（细节展示按钮）。



通过添加这个按钮，触碰行仍然执行其主要功能-开启关闭勾选标志。但此时如果触碰这个细节按钮就会打开Edit Item界面。

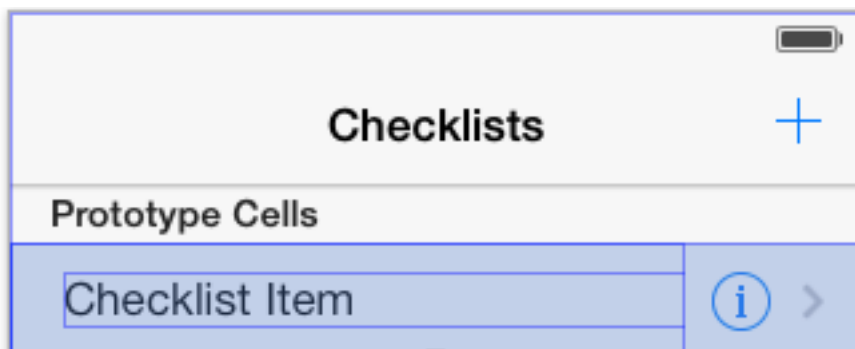
注意：

在苹果官方的Reminders应用中提供了另外一种解决方案。也就是让勾选标志显示在左侧，因此触碰行的最左侧会开启关闭勾选标志。而触碰行的其它位置则会打开Edit 界面。

让我们动作操作吧。

在Xcode中切换到storyboard，然后选中Checklists View controller上面的table view cell对象，在右侧面板中切换到Attributes inspector，然后将Accessory属性更改为Detail Disclosure.

现在之前的勾选标志不见了，取而代之的是类似下图的样子。



现在我们需要手动添加一个勾选标志。

从右下方的Objects Library中拖出一个Label到cell中，把它放置在文本标签的左侧。

在右侧面板的Attributes inspector中更改它的属性：

Text: 输入√(在mac电脑上用Alt/Option +V)

Font: System Bold, size 22

Tag: 1001

这里提供一个tag标志是为了方便后面找到它。

**Label**

Text

Color

Font

Alignment ☒ ☐ ☐

Lines

Behavior ☒ Enabled  
☐ Highlighted

Baseline

Line Breaks

Autoshrink

☐ Tighten Letter Spacing

Highlighted

Shadow

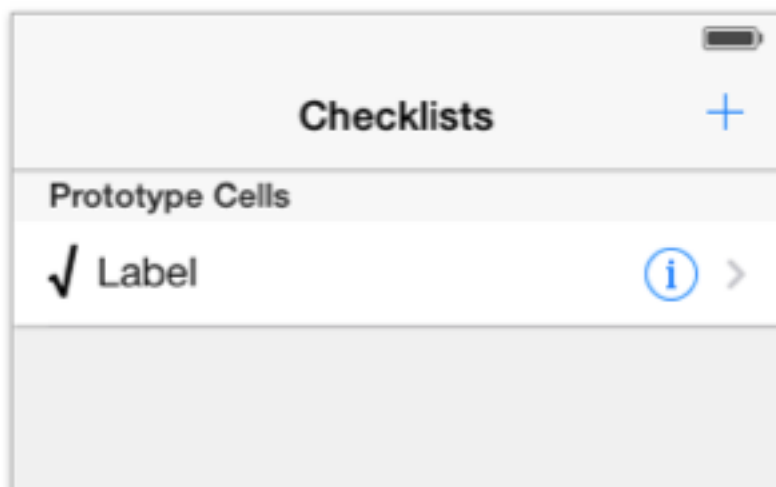
Shadow Offset    
Horizontal Vertical

**View**

Mode

Tag

设置完属性后，prototype cell现在会是下面的样子：



在Xcode中切换到ChecklistsViewController.m，更改configureCheckmarkForCell方法的代码为：

```

-(void)configureCheckmarkForCell:(UITableViewCell *)cell withChecklistItem:(ChecklistItem
*)item{

    UILabel *label = (UILabel *)[cell viewWithTag:1001];

    if(item.checked){
        label.text = @"√";
    }else{
        label.text = @"";
    }

}

```

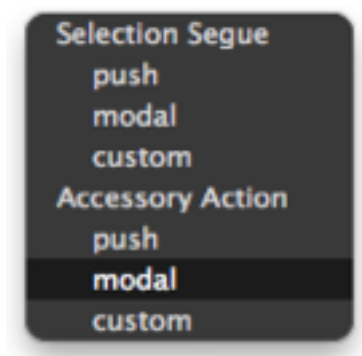
和之前不同的时，这里不再直接更改cell的accessoryType属性，而是通过更改新标签的文本内容来实现开启/关闭勾选标志。

编译运行项目，可以看到勾选标志出现在项目的左侧，而在右侧则是一个绿色的细节显示按钮。触碰某行仍然会开启关闭勾选标志，而触碰蓝色按钮则不会。



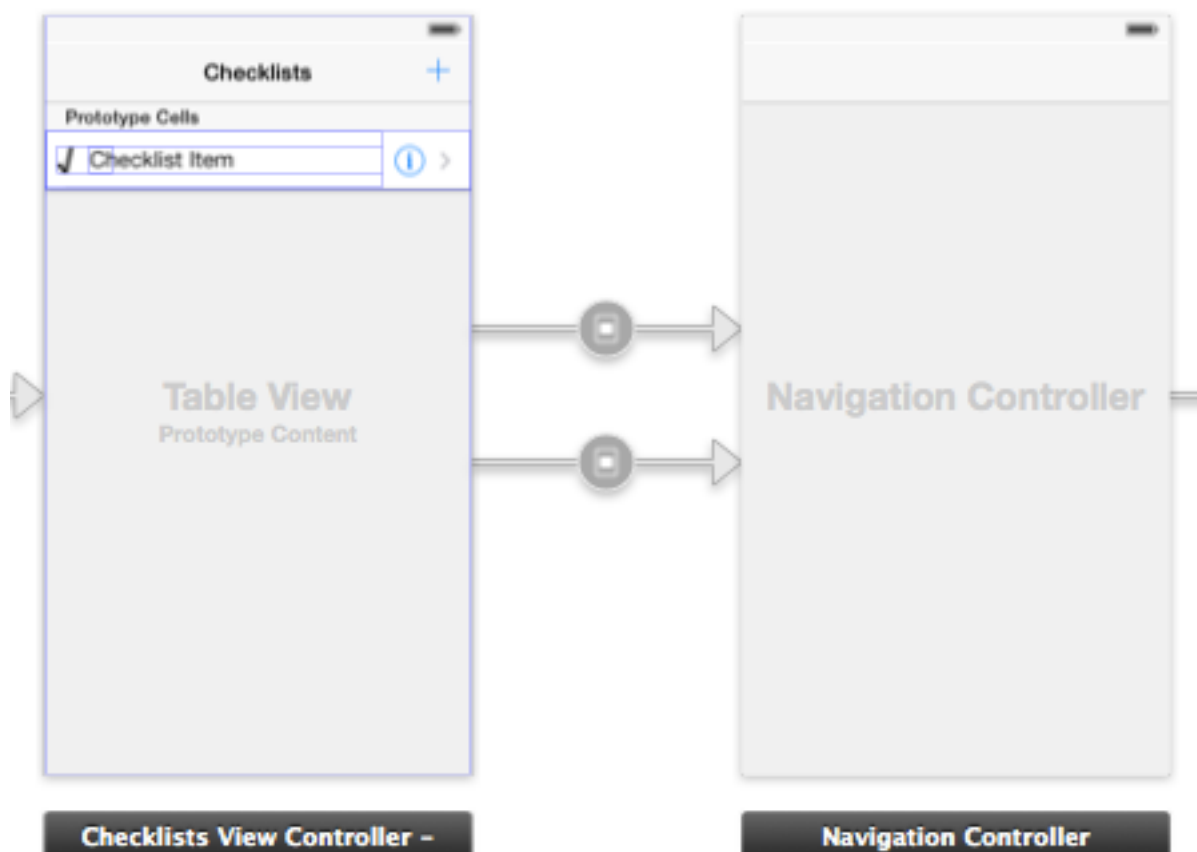
接下来要做的事情就是让细节显示按钮可以开启Add/Edit Item界面。这个倒是很简单，因为Interface Builder允许我们为细节显示按钮创建一个segue。

在Xcode中切换到storyboard，选中Checklists View Controller的table view cell，然后按住ctrl键，拖出一条线到旁边的导航控制器。从弹出菜单中选择Accessory Action部分的modal。注意不是Selection Segue里面的。



此时可以看到在Checklists View controller界面和旁边的导航控制器之间出现了两个segue。其中一个由“+”按钮触发，而另一个则是由prototype cell的细节显示按钮触发。显然，对两个segue必须提供各自不同的标识符。

选中这个新的segue,将其identifier属性设置为EditItem。



此时如果编译运行项目，触碰细节显示按钮会打开Add Item界面，但cancel和done两个按钮却毫无反应，因为我们还没有设置代理对象。此前我们在prepareForSegue中设置了和+按钮对应的AddItem 的代理，但是对于这个新的segue却没有设置。

在解决这个问题之前，我们先让Add/Edit Item界面具备编辑已有checklistItem对象的能力再说。

在Xcode中切换到AddItemViewController.h，然后添加一行属性变量声明：

```
@property(n nonatomic, strong) ChecklistItem *itemToEdit;
```

这个属性变量指向的就是用户想要编辑的已有的ChecklistItem对象。但是当我们添加一个新的to-do项目时，itemToEdit属性是nil。视图控制器据此来区分添加和编辑一个项目。

在Xcode中切换到AddItemViewController.m，然后更改viewDidLoad方法的代码如下：

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

    if(self.itemToEdit != nil){
        self.title = @"Edit Item";
        self.textField.text = self.itemToEdit.text;
    }
}
```

记住一点，当视图控制器被创建时还没有显示在界面上的时候系统会自动调用viewDidLoad方法。此时我们可以对用户界面做一些调整。在编辑模式下，self.itemToEdit不等于nil，因此我们极爱那个导航栏的标题更改为Edit Item。只需要修改self.title属性就可以实现这一点。导航控制器将查看该属性并自动修改导航栏中的标题。同时我们还根据item的文本属性来设置文本域中的文本内容。

现在AddItemViewController可以识别何时需要编辑项目了。如果其self.itemToEdit属性被提供了一个ChecklistItem对象，那么界面就会自动神奇的切换到Edit Item界面。

听起来挺不错的，不过我们在哪里填充itemToEdit属性呢？当然是在prepareForSegue里面！

这里是一个理想的在不同界面间传递属性值的地方。

在Xcode 中切换到ChecklistsViewController.m，然后更改prepareForSegue方法的内容如下；

```

-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{

    if([segue.identifier isEqualToString:@"AddItem"]){

        //1
        UINavigationController *navigationController = segue.destinationViewController;

        //2
        AddItemViewController *controller = (AddItemViewController*)
navigationController.topViewController;

        //3
        controller.delegate = self;
    }else if([segue.identifier isEqualToString:@"EditItem"]){

        UINavigationController *navigationController = segue.destinationViewController;

        AddItemViewController *controller = (AddItemViewController*)
navigationController.topViewController;

        controller.delegate = self;

        NSIndexPath * indexPath = [self.tableView indexPathForCell:sender];

        controller.itemToEdit = _items[indexPath.row];

    }
}

```

以上方法中主要是增加了当segue的标识符是EditItem的情况处理。

和之前一样，我们从storyboard中获取了导航控制器，并通过topViewController属性获取了到内置的AddItemViewController的引用。这里同样设置了controller的delegate属性，这样当用户触碰cancel或done按钮时会得到通知。

新的不同的地方在于：

```

NSIndexPath * indexPath = [self.tableView indexPathForCell:sender];

controller.itemToEdit = _items[indexPath.row];

```

这里的sender参数其实指的就是触发了该segue的控件，在这里就是table view cell中的细节显示按钮。通过它可以找到对应的index-path，然后获取要编辑的ChecklistItem对象的行编号。

在继续之前，先来点理论充电。

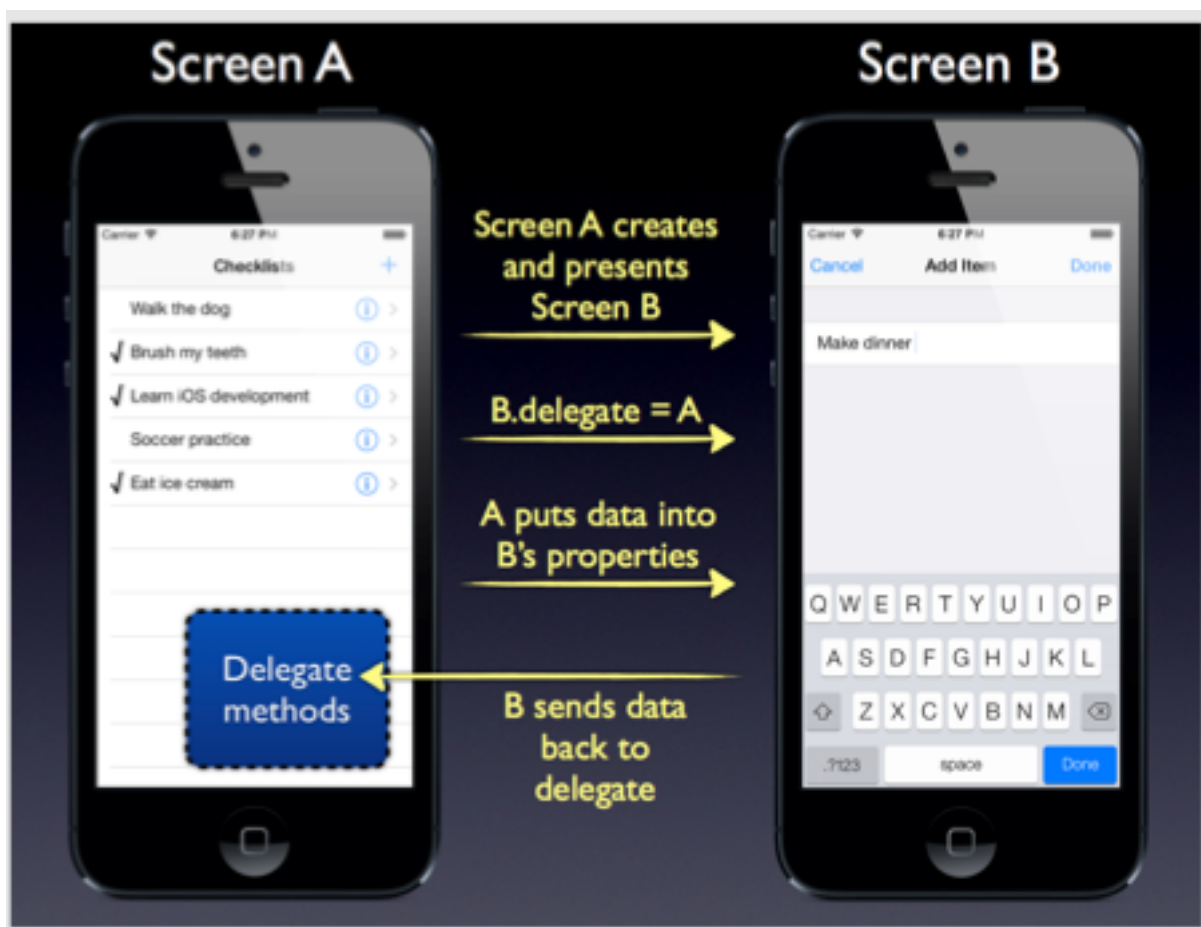
如何在视图控制器之间传递数据

之前我们曾经了解过界面B（比如Add/Item Item界面）如何向界面A（Checklists界面）回传数据，是通过代理实现的。和之前不同的是，这里我们是真真切切的传递了一个ChecklistItem对象。

此类数据传输通常以两种形式进行：

如果界面A打开了界面B，那么界面A可以直接向界面B提供所需的数据。我们只需要为该数据在界面B创建一个熟悉，然后界面A可以在界面B呈现之前将某些信息放到该属性中即可。

但如果想从界面B从界面A回传数据，最好就是通过代理来实现了。



在本篇教程中我们还将多次重复这个过程，所以不必太担心。

当以上步骤完成之后，可以编译运行应用。

当你触碰+按钮时，会和之前一样开启Add Item界面。

当你触碰附属的细节显示按钮时，会打开Edit Item界面，其中包含了项目的文本内容。





不过有一个小小的问题：之前导航栏上的done按钮初始状态下是被禁用的。这里当然要对此进行一下微调。

在Xcode中切换到AddItemViewController.m，更改viewDidLoad方法的代码如下：

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

    if(self.itemToEdit != nil){
        self.title = @"Edit Item";
        self.textField.text = self.itemToEdit.text;
        self.doneBarButton.enabled = YES;
    }
}
```

```
}
```

麻烦并没用到此结束。如果编译运行项目，触碰某个行来编辑它，然后点击done按钮。和我们想象的不同，程序并没保存我们对现有项目的修改，而是添加了一个新的项目。这是因为我们并没编写更新数据模型的代码，因此代理对象默认为还是在添加一个新的行。

为此我们需要在原有的代理协议中添加一个新方法。

在Xcode中切换到AddItemViewController.h，在@protocol部分添加一个新方法声明：

```
-(void)addItemViewController:(AddItemViewController*)controller  
    didFinishEditingItem:(ChecklistItem*)item;
```

完整的协议声明如下：

```
@protocol AddItemViewControllerDelegate <NSObject>  
  
-(void)addItemViewControllerDidCancel:(AddItemViewController*)controller;  
  
-(void)addItemViewController:(AddItemViewController*)controller  
    didFinishAddingItem:(ChecklistItem*)item;  
  
-(void)addItemViewController:(AddItemViewController*)controller  
    didFinishEditingItem:(ChecklistItem*)item;  
  
@end
```

现在协议中拥有了三个方法，一个是用户触碰cancel会调用的方法，而另外两个则是用户触碰done会调用的方法。当我们添加一个新的项目时，didFinishAddingItem方法会被调用，但如果是在编辑一个现有的项目，则会调用didFinishEditingItem方法。

通过使用不同的方法，代理对象（视图控制器）可以在不同情况下做出不同的处理。

回到AddItemViewController.m，更改done方法的代码如下：

```
-(IBAction)done:(id)sender {  
  
    if(self.itemToEdit == nil){  
        ChecklistItem *item = [[ChecklistItem alloc] init];  
        item.text = self.textField.text;  
        item.checked = NO;  
  
        [self.delegate addItemViewController:self didFinishAddingItem:item];  
    }else{  
  
        self.itemToEdit.text = self.textField.text;  
        [self.delegate addItemViewController:self didFinishEditingItem:self.itemToEdit];  
    }  
}
```

```
}  
}
```

这里首先检查itemToEdit属性中是否包含了一个对象，如果不是，那么用户就是在添加一个新项目。如果是，就是在编辑当前项目。

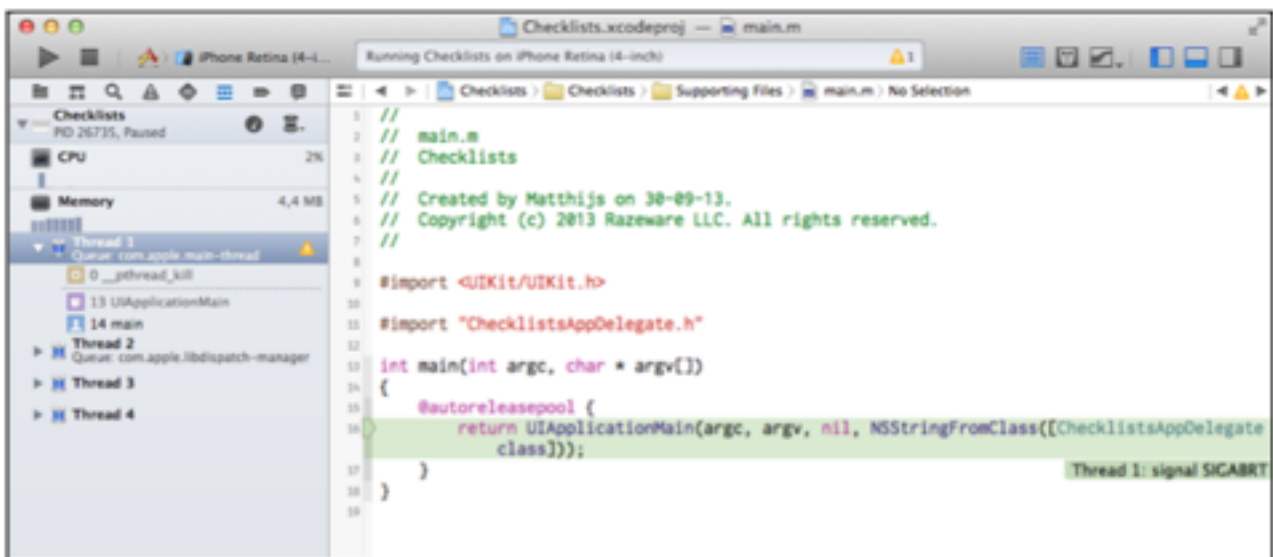
好吧，如果你认为现在已经大功告成的话，大可以一试。结果必然是-崩溃。  
当你触碰done按钮的时候，会看到提示出错：

```
*** Terminating app due to uncaught exception  
'NSInvalidArgumentException',  
reason: '-[ChecklistsViewController addItemViewController:
```

108

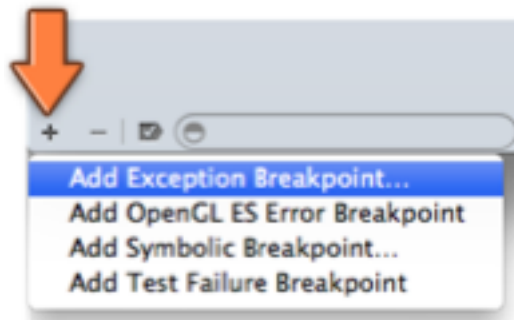
The iOS Apprentice (Second Edition) *Checklists*

```
didFinishEditingItem:]: unrecognized selector sent to  
instance 0xab04950'
```



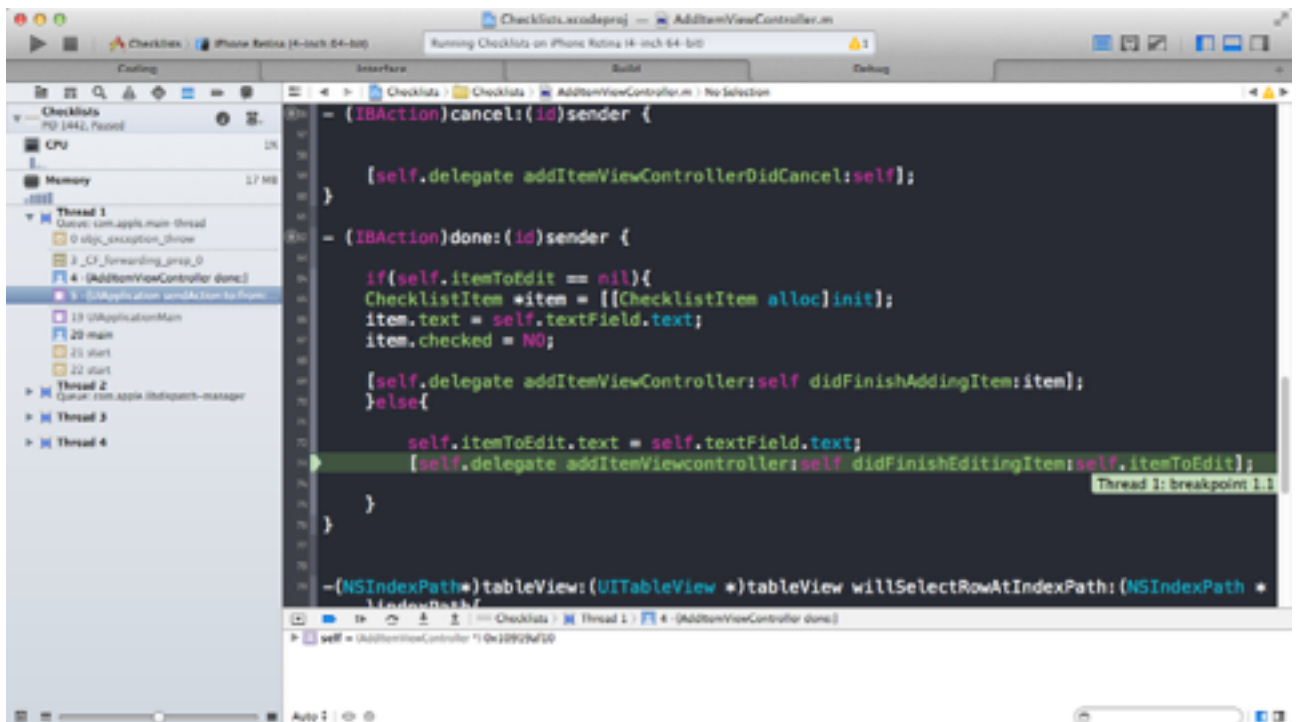
对于此次崩溃的原因，Xcode指向了main.m，但显然对你是个误导。为此我们需要启用Exception Breakpoint。

在Xcode左侧面板中切换到Breakpoint navigator，然后点击底部的+按钮，选择Add Exception Breakpoint...



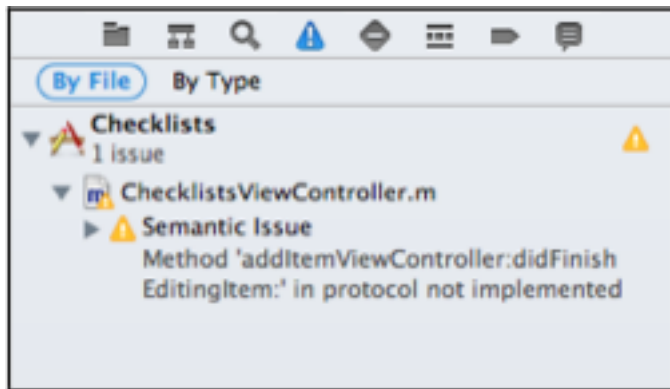
再次编译运行项目，编辑一个现有的项目，然后触碰done按钮。

现在Xcode将准确指向出问题的代码行：



关于代码调试，后续我们还会介绍很多相关的技巧。不过这里我们指出的其实是使用代理协议时常见的错误。

在Xcode左侧面板中切换到Issue navigator，会看到Xcode早就给你一个警告了。



Xcode在提醒你，ChecklistsViewController有一个协议方法并没有实现，这就是崩溃的原因：不能识别的selector。

在Objective-C的术语中，selector（有人翻译成选择器，很别扭，还不如直接用英文）代表方法的名称，因此这里的警告意味着：  
应用尝试调用一个名为addItemViewController:didFinishEditingItem:的方法，但我们却没有在任何地方实现它（虽然声明了）。

这一点毫不奇怪，我们虽然在代理协议中添加了这个方法声明，但并没有通知代理对象（视图控制器）这个方法究竟是干什么的。

为此，在Xcode中切换到ChecklistsViewController.m，然后添加一个方法如下：

```
-(void)addItemViewController:(AddItemViewController *)controller didFinishEditingItem:
(ChecklistItem *)item{
```

```
    NSInteger index = [_items indexOfObject:item];
```

```
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:index inSection:0];
```

```
    UITableViewCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
```

```
    [self configureTextForCell:cell withChecklistItem:item];
```

```
    [self dismissViewControllerAnimated:YES completion:nil];
```

```
}
```

好了，有了这个方法，代理对象就知道如何在编辑完成后进行相应的处理。虽然AddItemViewController已经更新了ChecklistItem对象的内容，但那只是数据模型的更新，我们仍然需要更新表视图中对应的内容。

再次编译运行应用，一切都应该如你所期待的那样正常工作了。

又是漫长的一章，辛苦了，发送福利的时刻到了。

如今是一寸山河一寸霾，还是让我们看看洗肺的美景吧。





接着还是养眼的mm



