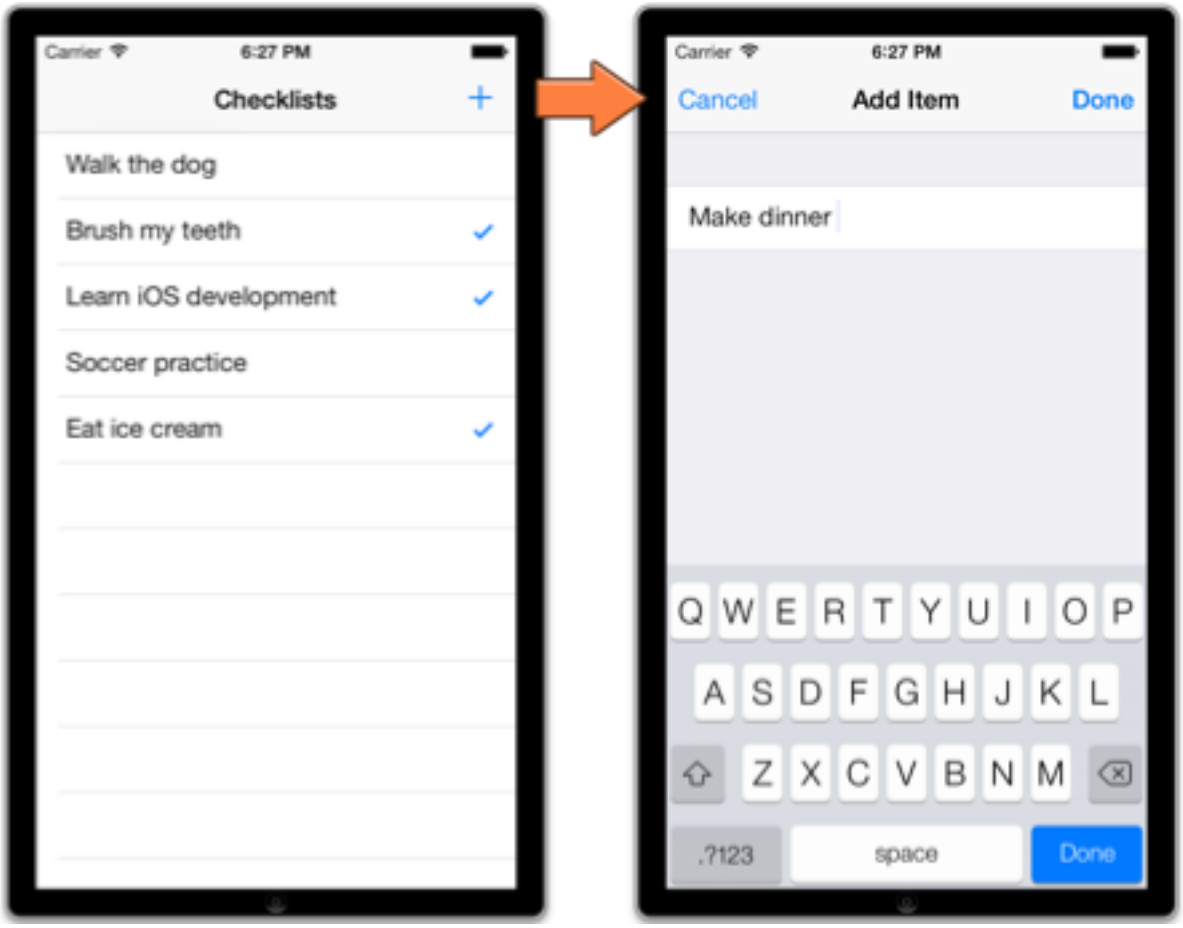
从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter6

欢迎继续我们的学习。

在之前的章节中，我们已经成功的往表视图中添加了一大堆数据，但遗憾的是这些数据都是程序默认提供的，没用户什么事儿。说白了，前5章我们都只不过是在自娱自乐而已，从这一章开始我们要赐予用户力量了。May the force be with you!

在本章学习中，我们首先要在界面顶部添加一个所谓的navigation bar（导航栏），这个bar上面有一个添加按钮（大的+号），用它可以开启新的界面，在其中可以让用户输入新的代办事务。当用户触碰Done时，新的代办事务就会出现在列表中。



在很多应用中都用到了这种模式，也就是打开一个新的界面来添加新项目。当你学会了这一点，你就可以自豪的说自己的iOS已经入门了。

好吧，看看我们要做哪些事情?
1.添加一个navigation controller
2.在导航栏上面放一个"+"添加按钮

3.当触碰”+“添加按钮的时候往列表中添加一个伪项目
4.使用swipe-to-delete（滑动删除）的方式来删除已有项目
5.打开Add Item界面让用户输入项目的文字描述。

当然，我们不可能把所有这些内容都在一章内完成，还是要一步步来。比如在放了”+“按钮之后，先可以写代码往列表中添加一个”伪“项目。当我们完成这一步之后，再来学习如何打开一个新的Add Item界面。

首先要接触一个新东西，就是和表视图控制器一样知名的--

Navigation controller（导航控制器）

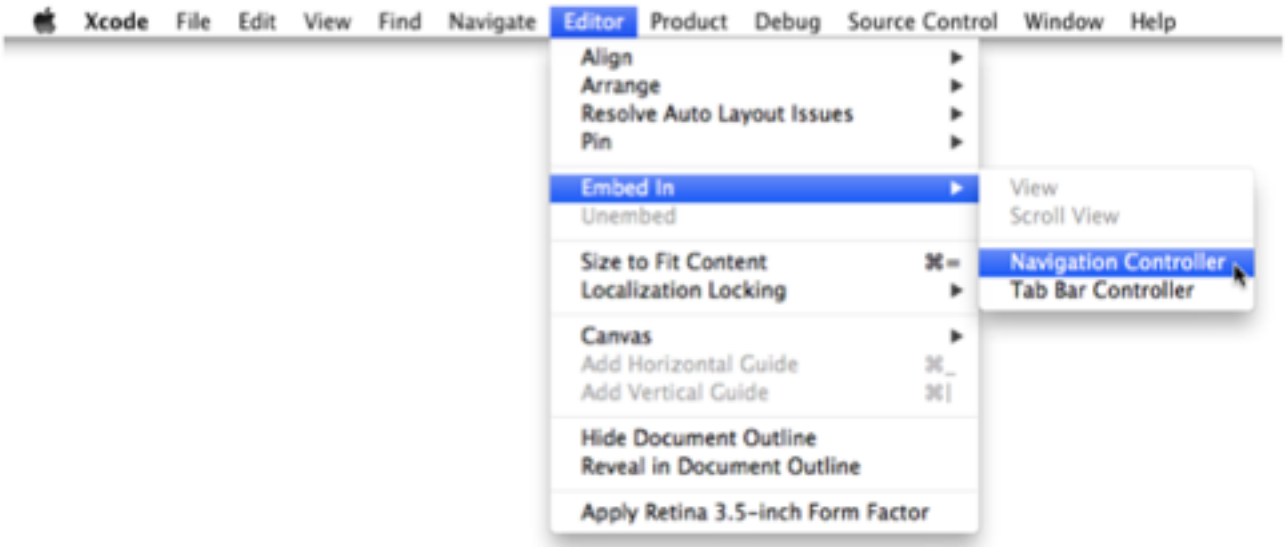
这里不多说理论方面的废话，先实际操作来感受下吧。
首先让我们来添加导航栏(navigation bar)，当然，如果你观察力不错的话，会发现在Object Library中有一个名为Navigation Bar的对象。你可以把它拖曳到视图里面。
不过这里我们暂时不这样做，这里的方法是把当前的表视图控制器embed（内置）在一个navigation controller（导航控制器）中。神马？这样也行？
Don’t panic，先让我们试试看吧。

实际上，在iOS界面开发中，导航控制器的使用频率仅次于表视图控制器。使用导航控制器，可以从一个界面跳转到另一个界面。除了导航控制器，还有一个UITabBarController的家伙也可以管理多个视图控制器，当然这里就不细说了。

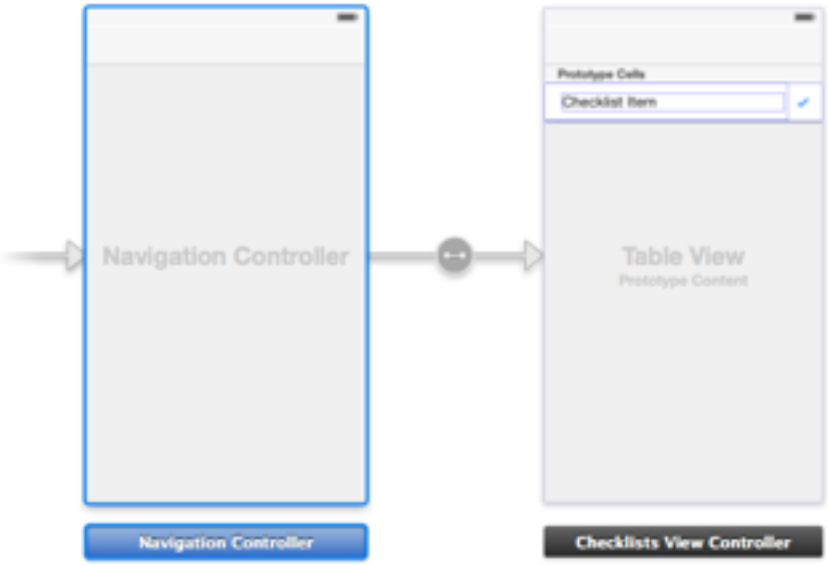
UINavigationController对象可以帮你轻松解决界面间跳转导航的事情，可以节省很多代码工作。通常在界面上方中间有一个title标题，以及一个用于返回上一个界面的"<"返回按钮。我们还可以在右侧放上自己的按钮。

添加导航控制器没有你想的那么复杂。

在Xcode中打开Main.storyboard，然后选中Checklists View Controller。然后在顶部菜单选择Editor-Embed In- Navigation Controller。



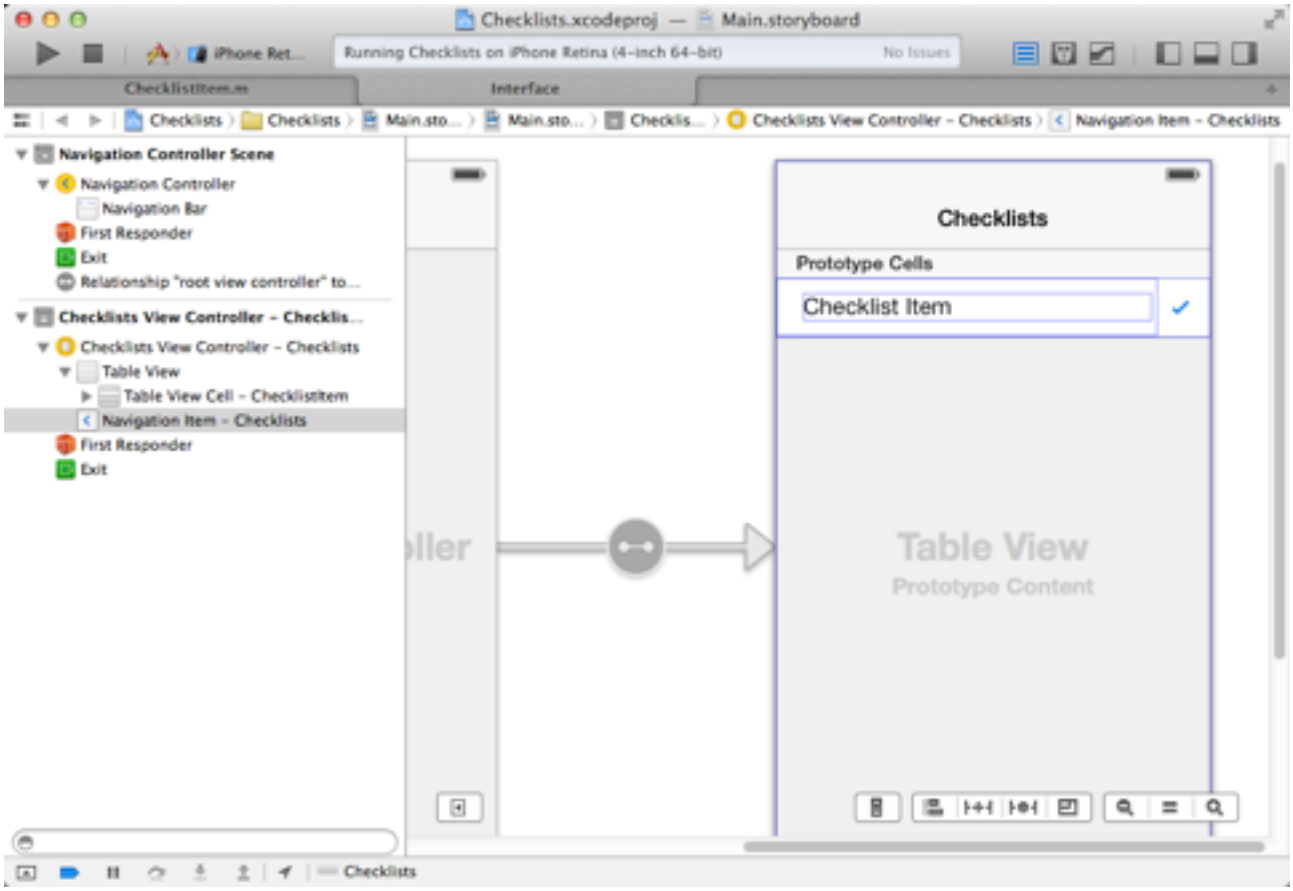
就这样搞定了。
现在Interface Builder已经添加了一个新的Navigation Controlller 界面，同时在导航控制器和之前的Checklists View Controller之间创建了关联。



编译运行项目，当启动应用的时候，Checklists View Controller会被自动放到一个导航控制器中。

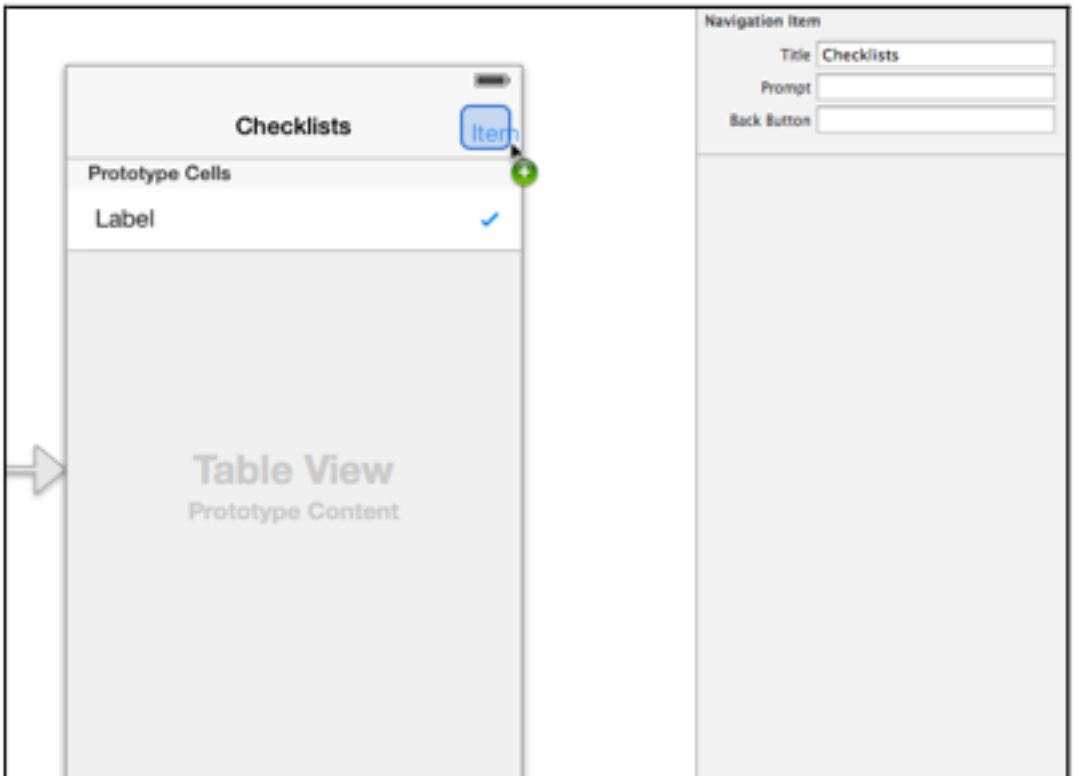
相比之前的界面，终于有了一丝变化，也就是顶部有了一个导航栏。

现在返回Xcode中的storyboard，双击Checklists View Controller中的navigation bar，将标题改为Checklists



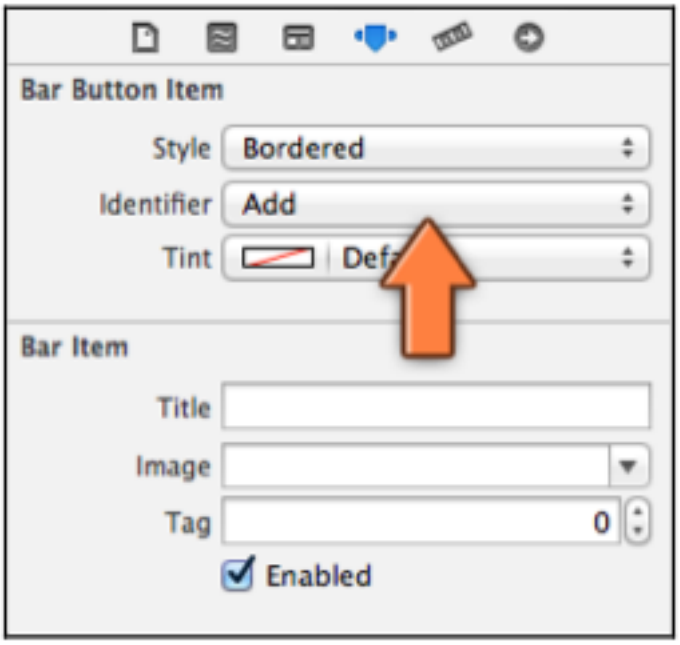
我们刚才做的事情就是更改自动添加到视图控制器中的Navigation Item对象的标题。
Navigation Item对象中包含了将在导航栏中显示的标题和按钮。
每个内置的视图控制器都有一个自己的Navigation Item，当导航控制器让一个新的视图控制器来接管界面时，就会替换该视图控制器的Navigation Item的导航栏内容。

接下来在Object Library中找到Bar Button Item，然后把它拖曳到导航栏的右侧位置。注意是Checklists View Controller里面的导航栏，而不是导航控制器里面的！！！



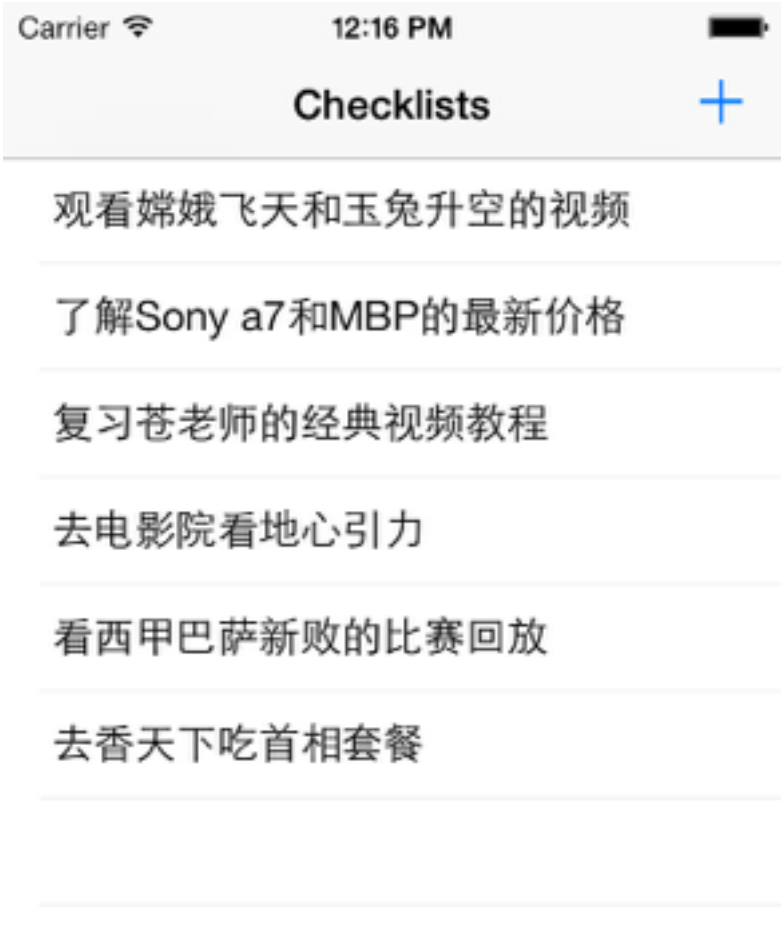
默认情况下这个新的按钮名为"Item"，不过我们这里需要把它改成一个"+"号标志。

选中这个bar button item,然后在右侧的Attributes inspector中选择Identifier: Add



仔细看看Identifier 列表里面的其它选项，你可以看到很多预定义的bar button 类型：Add,Compose,Reply,Camera等等。根据自己应用的需要，你可以选择合适的类型，选错了的话苹果可是会拒绝你的应用上线哦！

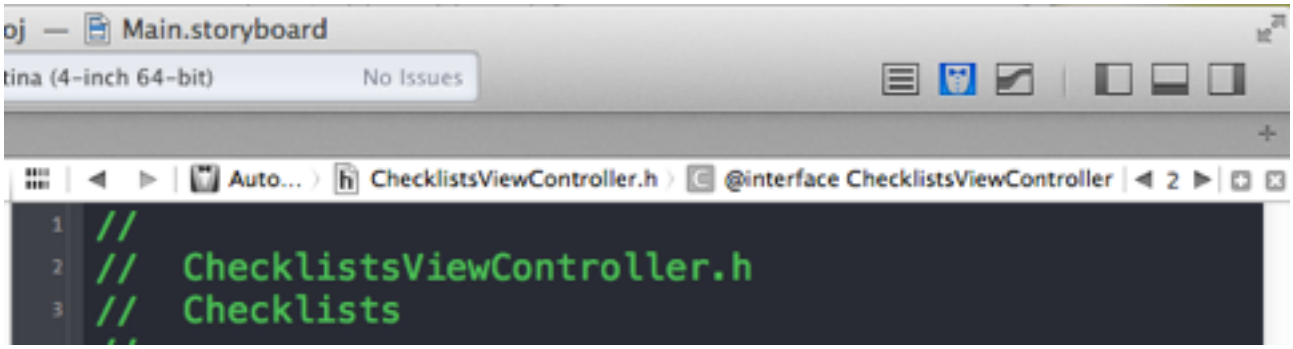
好了，现在我们已经成功的添加了一个按钮。编译运行应用，结果是这样的：



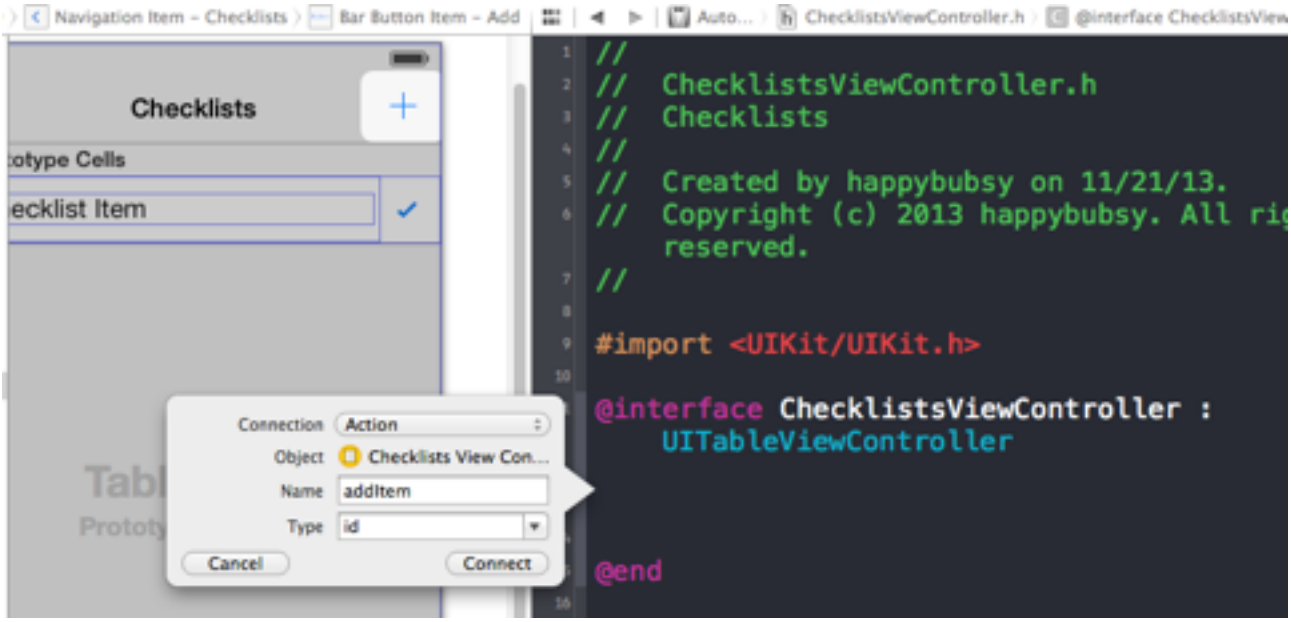
如果你已经迫不及待的按"+"按钮，会发现没有什么反应，这是因为我们并没有给触碰事件关联一个action动作方法。稍后我们会创建一个Add Item界面，然后当触碰按钮的时候显示新的界面。不过在此之前我们先学习如何在表中添加新的行。

首先把这个添加按钮和某个动作方法关联起来。这个你应该很熟悉了。

最简单直接的方法，在storyboard中选中这个按钮，在右上角打开Assistant Editor，确保看到Xcode界面的右侧出现ChecklistsViewController.h的代码。



按住ctrl键从"+"按钮拖曳一条线到@end前，然后在对话框里面把Connection类型设置为Action,Name那里填上addItem，点击Connect就好了。



接下来在Xcode中切换到ChecklistsViewController.m，替换addItem方法的内容如下：

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

    NSInteger newRowIndex = [_items count];

    ChecklistItem *item  =[[ChecklistItem alloc]init];

    item.text = @"我是新来的菜鸟，求照顾求虐";
    item.checked = NO;
    [_items addObject:item];

    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:newRowIndex inSection:0];
    NSArray *indexPaths = @[indexPath];

    [self.tableView insertRowsAtIndexPaths:indexPaths
withRowAnimation:UITableViewRowAnimationAutomatic];


}
```

这里看起来有一堆新东西，我们还是慢慢解释吧。

在这个方法中我们创建了一个新的ChecklistItem对象，然后把它添加到我们的数据模型中（也就是_items这个数组）。同时，我们还需要判断这个新对象的行编号，然后告诉表视图，“这里我要在这个index编号处添加一个新的行，别忘了更新一下你自己”。

来一行行看。

```
NSInteger newRowIndex = [_items count];
```

当我们启动应用的时候，在数组中已经有了6个项目，在界面上已经出现了6行数据。计算机从0开始计数，因此现有的行编号就是0，1，2，3，4和5。我们需要在数组的结尾添加新的行，所以新行的编号就是6。

换句话说，当我们在表中添加一行新数据的时候，新行的编号始终等于表中当前的项目数。这里的newRowIndex就是新行的编号。

接下来的几行代码大家就相当熟悉了。

```
   ChecklistItem *item  =[[ChecklistItem alloc]init];

    item.text = @"我是新来的菜鸟，求照顾求虐";
    item.checked = NO;
    [_items addObject:item];
```

这几行代码之前在viewDidLoad方法中看到过。我们创建了一个新的ChecklistItem对象，然后把它添加到数组的最后。现在数据模型的_items数组里面就有了7个ChecklistItem对象。注意现在

newRowIndex是6，而[items count]已经变成了7。这就是为什么我们需要读取[items count],，然后把它保存在newRowIndex这个本地变量中。

接下来的代码看起来就有点恼火了。

NSIndexPath *indexPath = [NSIndexPath indexPathForRow:newRowIndex inSection:0];

仅仅把新的ChecklistItem对象放到数据模型中还不够。我们还得告诉表视图有了一个新的行，这样它才会分配一个新的cell给这一行。之前我们学过，表视图通过index-paths来识别行，所以首先需要创建一个NSIndexPath对象指向新的行，而使用的参数则是newRowIndex这个行编号。这样一来，index-path对象就可以指向section 0的第5行了。

下一行代码创建了一个新的临时性的数组：

NSArray *indexPaths = @[indexPath];

由于接下来我们将使用表视图的insertRowsAtIndexPaths方法来插入新的行，不过看名字你也知道，我们会同时插入多个行。因此我们需要一个index-paths的数组，而不是一个单一的NSIndexPath对象。好吧，这样做不是很方便，但只能这样。幸运的是，现在我们可以使用@[indexPath]这种方式来创建一个包含单一index-path对象的数组。
@[]用于创建一个包含在方括号中所有对象的NSArray数组对象。

最后，我们通知表视图调用insertRowsAtIndexPaths方法来插入新的行，还带有一个动画效果：

```
  [self.tableView insertRowsAtIndexPaths:indexPaths
withRowAnimation:UITableViewRowAnimationAutomatic];
```

好了，重复一遍以上步骤：
1.创建一个新的ChecklistItem对象
2.把它添加到当前的数据模型中
3.在表视图中为该行数据插入一个新的cell

编译运行应用，现在我们可以随意添加新的行了。你还可以触碰这些新行来添加或去掉勾选标志。你可以随意上下滑动表，这些勾选标志的状态会被正确的保持。

特别需要强调的一点是，当你需要显示新的行时，需要往表和数据模型中同时添加新的行数据。当你向表视图发送insertRowAtIndexPaths消息时，其实就是说："那啥你好，我的数据模型里面有了不少新东西，麻烦你显示一下。"这太重要了！
如果你忘了通知表视图，或者你通知了表视图但却忘了给数据模型里面添加新数据。那么应用就会崩溃。这二者必须同步进行。

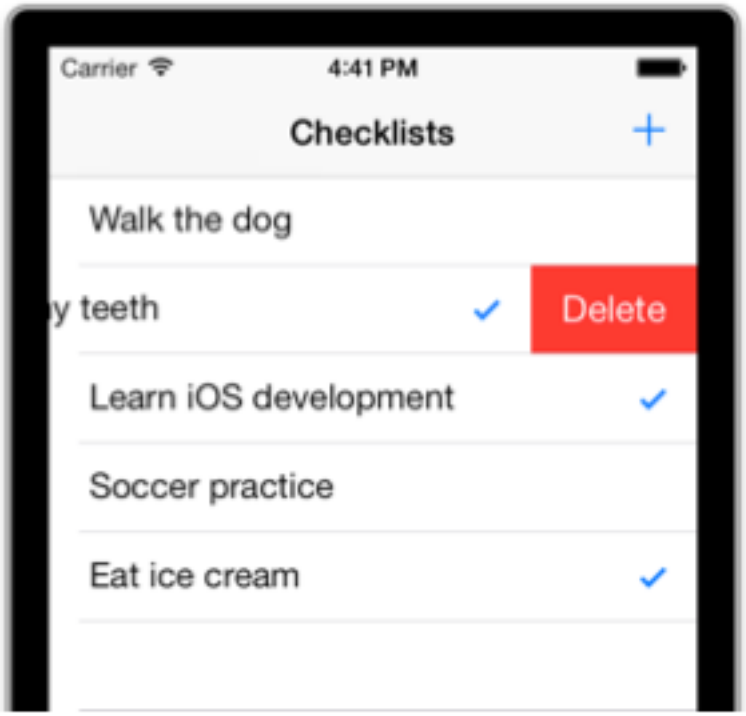
小练习：

让这些新添加的代办事务默认情况下被勾选。

在结束今天的学习之前，我们还要再实现一个新操作，就差不得圆满了。

那就是--
删除行数据

既然你允许用户添加新的行，显然也要允许他们删除旧的行。在iOS应用中，最常用的方式就是swipe-to-delete，滑动删除。简单来说，你在某一行上滑动手指，然后就会看到屏幕上出现一个红色的Delete按钮。如果你触碰这个Delete按钮，就确认删除，如果触碰了屏幕的其它地方就是取消该操作。



看起来很高大上，其实实现起来超级简单。

还是回到Xcode，切换到ChecklistsViewController.m，然后在@end前添加以下方法：

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

    [_items removeObjectAtIndex:indexPath.row];

    NSArray *indexPaths = @[indexPath];

    [tableView deleteRowsAtIndexPaths:indexPaths
withRowAnimation:UITableViewRowAnimationAutomatic];

}
```

当我们在视图控制器（表视图的数据源）中提供commitEditingStyle方法时，表视图就会自动启用swipe-to-delete的功能。我们要做的就是从数据模型中删除该项目。

```
 [_items removeObjectAtIndex:indexPath.row];
```

然后在表视图中删除对应的行：

```
   NSArray *indexPaths = @[indexPath];
```

```
    [tableView deleteRowsAtIndexPaths:indexPaths
withRowAnimation:UITableViewRowAnimationAutomatic];
```

以上代码看起来复杂，其实和之前在addItem方法中所做的类似。这里我们再次创建了一个只有一个index-path对象的NSArray对象，然后通知表视图以动画的形式删除了这些行（实际上是这一行）。

结束之前，再来点理论脑补。

销毁对象

为了让自己晚上不做噩梦，有时候我们希望可以清除自己的一些回忆，可惜目前还很难做到，至少不能精准的定点清除。
不过对于不想要的对象，我们就可以毫不客气了。

当调用{items removeObjectAtIndex]方法时，不仅从数组中删除了这个ChecklistItem对象，实际上也永久将其销毁了。

关于对象的销毁，后面的教程中还会提到，这里大概说一下。当程序中没有对某个对象的引用时，它就会被自动销毁。只要某个ChecklistItem对象还在数组中，那么数组就保持着一个到该对象的引用。但是当我们把这个ChecklistItem对象从该数组中拖出来后，引用没了，对象也会被销毁，或者用术语党的话叫deallocated（释放）。

那么一个对象被销毁意味着什么？每个对象都占用着计算机（或者iPhone)上的一小部分内存。当我们使用alloc来创建一个对象的时候，就会保留一小块的内存用来保存对象的值。当该对象被deallocate(释放）的时候，这部分内容就会再度开放，最终可能被其它新的对象所使用。当某个对象被销毁之后，该对象就不存在了，你再也没机会用到它。

很抽象？一个对象就是程序世界的一个生命，就好比芸芸众生的我们。当一个新生命诞生的时候，需要一个由分子原子组成的有机体来容纳它，这个有机体的形状、大小会随着生命的成长而发生变化，但始终为该有机体所用。直到有一天生命逝去，有机体会分解，重新化归泥土和分子原子，日后可能会被任何的其它生命或非生命所使用，这就是轮回?

在iOS5.0之前，iOS开发者需要手动管理内存，此后苹果推出了ARC（自动引用计数）技术，让iOS开发的内存管理不再那么晦涩难懂。关于ARC有点不好理解，这里就先不占用篇幅讲了。

好了，今天的学习到此结束，又到了福利时间~

世界杯快到了，赶紧换大屏幕电视吧，美女+足球+啤酒的享受时刻又来了！

哥最爱潘帕斯雄鹰和亚平宁的将士，你们呢？

附送值得一看的几场小组赛赛程安排，谁让我是球迷呢

北京时间	对阵球队	对阵球队
2014年6月13日 4：00	巴西	克罗地亚
2014年6月14日 4：00	西班牙	荷兰
2014年6月15日 6：00	英格兰	意大利
2014年6月16日 6：00	阿根廷	波黑
2014年6月17日 1：00	德国	葡萄牙
2014年6月18日 3：00	巴西	墨西哥
2014年6月19日 3：00	西班牙	智利
2014年6月20日 3：00	乌拉圭	英格兰
2014年6月25日 0：00	意大利	乌拉圭

其实还有很多场值得看，不过就看你熬夜功夫怎样了~据说世界杯往往是辞职高峰期，元芳你怎么看？