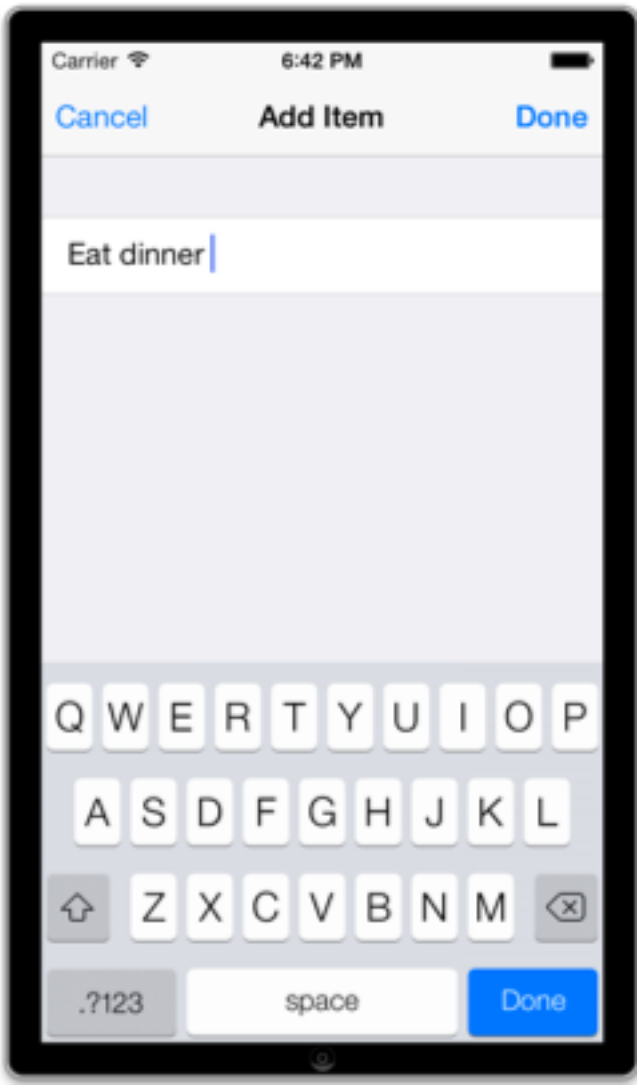
# 从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter8



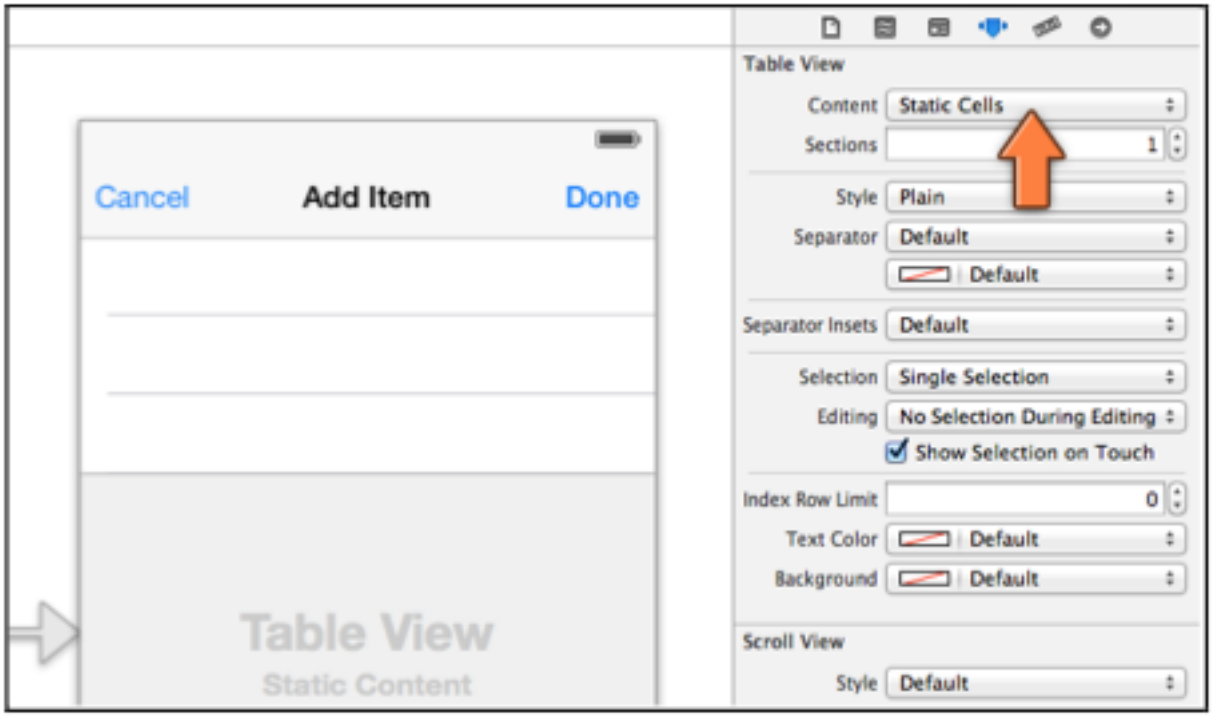
欢迎继续我们的学习。

在上一章的内容中我们学会了在storyboard中添加一个新的界面，同时还了解了segue和视图控制器容器的概念。似乎稍微有点抽象，不过我们将不断的重复，直到你可以完全理解。

现在Add Item界面中只是一个空白的表，然后在顶部有一个导航栏，但我们想要一个类似下面的界面：



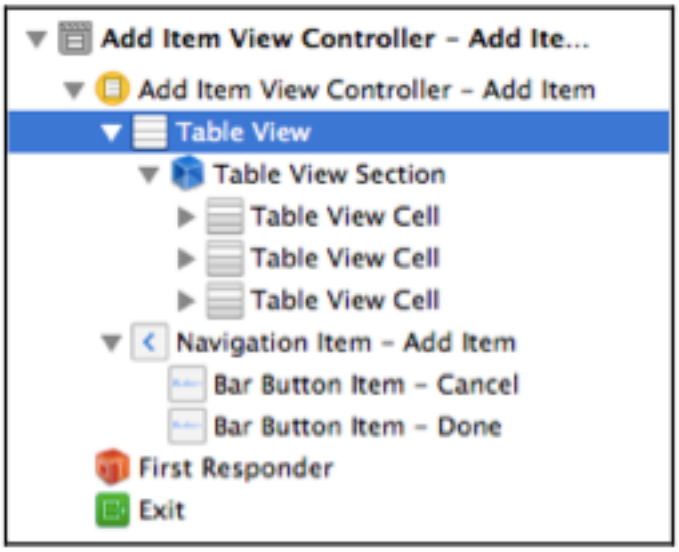

为此，在Xcode中切换到storyboard, 选中Add Item界面中的Table View对象，然后在Xcode右侧的面板中切换到Attributes inspector，将Content的设定从Dynamic Prototypes更改为Static Cells。



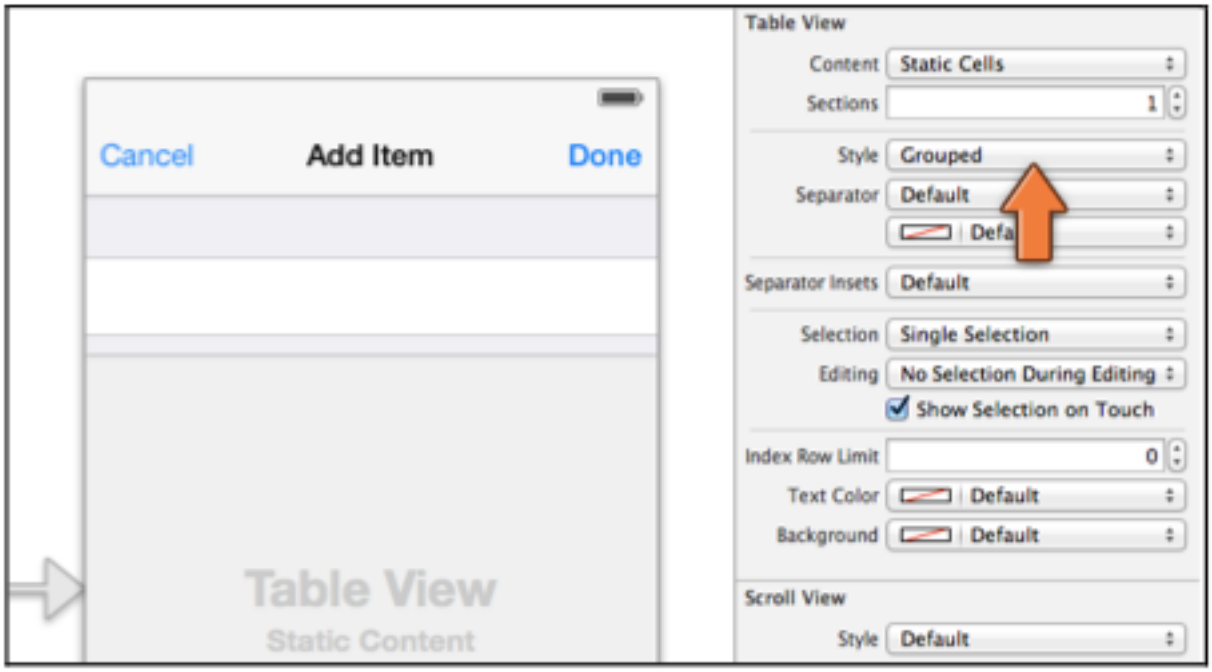

如果我们在设计的时候就知道在表视图中要放多少个section和行，就需要用到static cells。对于这种需要用户输入信息的界面，static cells尤其合适。我们可以在storyboard中直接设计行。对于带有static cells的行，我们无需提供数据源。而且我们可以直接降cell中的标签和其它控件和视图控制器中的属性关联在一起。

此时观察Xcode左侧面板，会看到Table View对象下面有了一个Table View Section，然后在这个Section里面有三个Table View Cell对象。

选中下面的两个Table View Cell对象，用delete键将其删除，因为我们只需要有一个cell就够了。

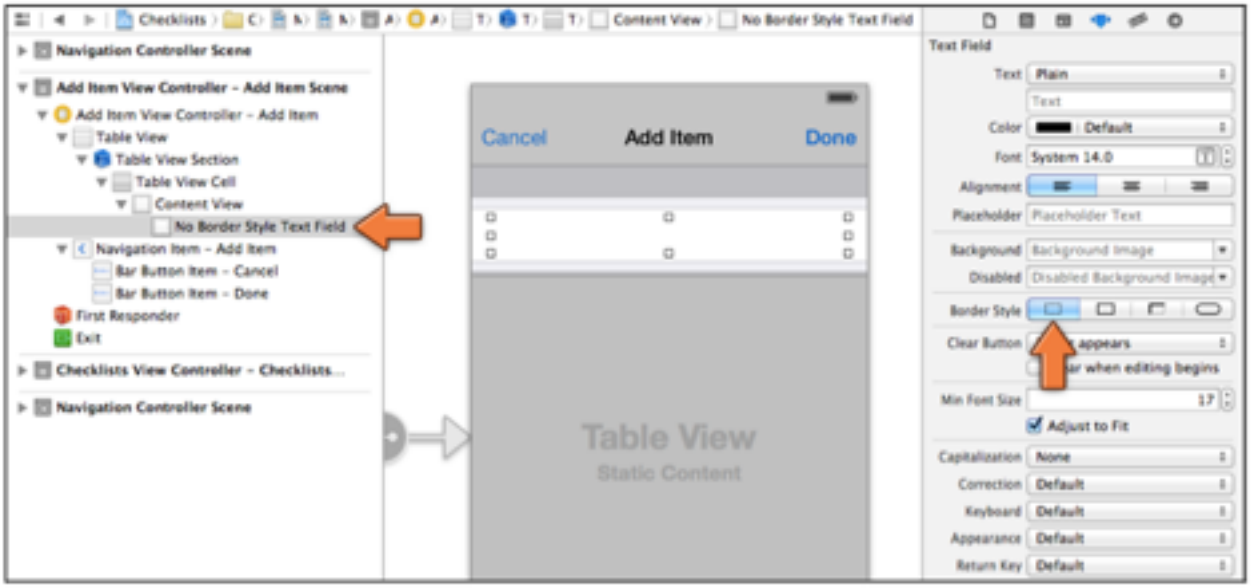



再次选中Table View对象，然后在Xcode右侧面板中切换到Attributes inspector,然后将Style属性更改为Grouped，这样就得到了我们希望的视觉效果。



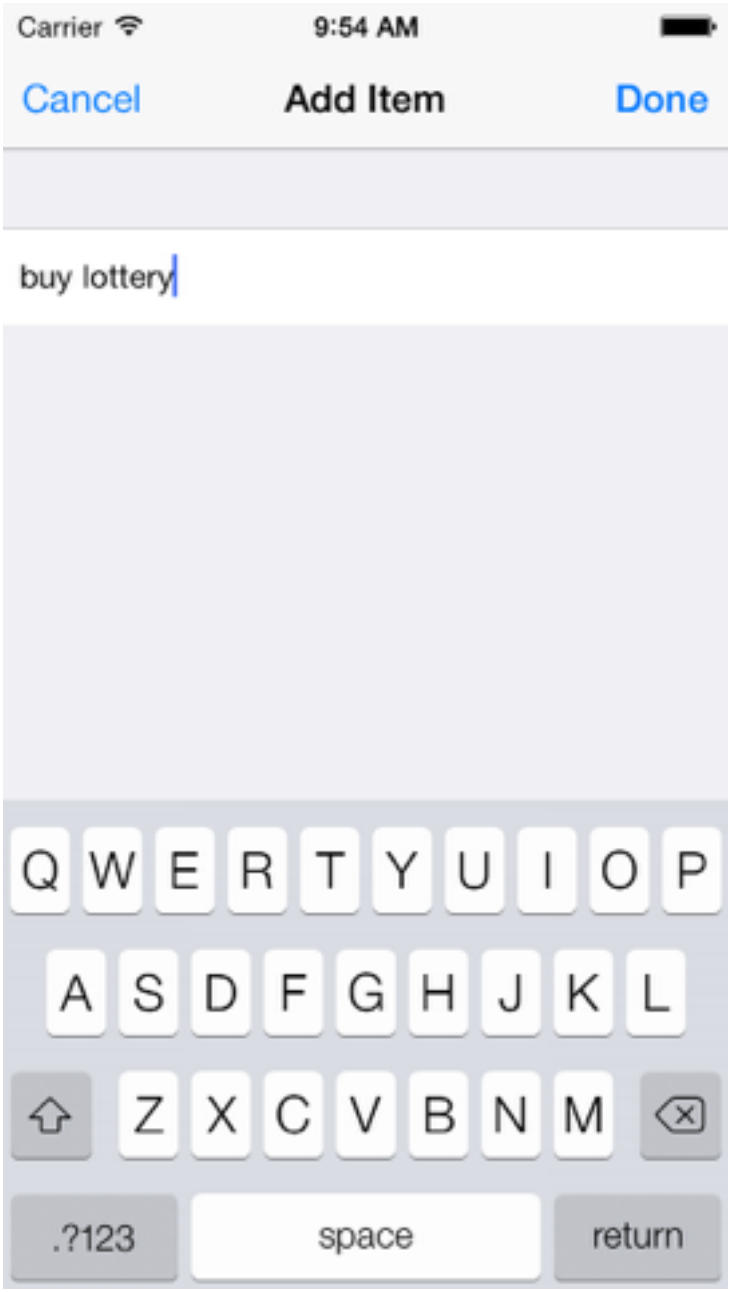

接下来我们需要在table view cell对象中添加一个text field（文本域），从而可以让用户在里面输入文字。

从Xcode右下的Object Library中拖出一个Text Field对象到cell中，然后根据需要调整它的大小。选中这个Text Field对象，在Xcode右侧面板中切换到Attributes inspector,然后将Border Style设置为none(也就是最左侧的选项）。

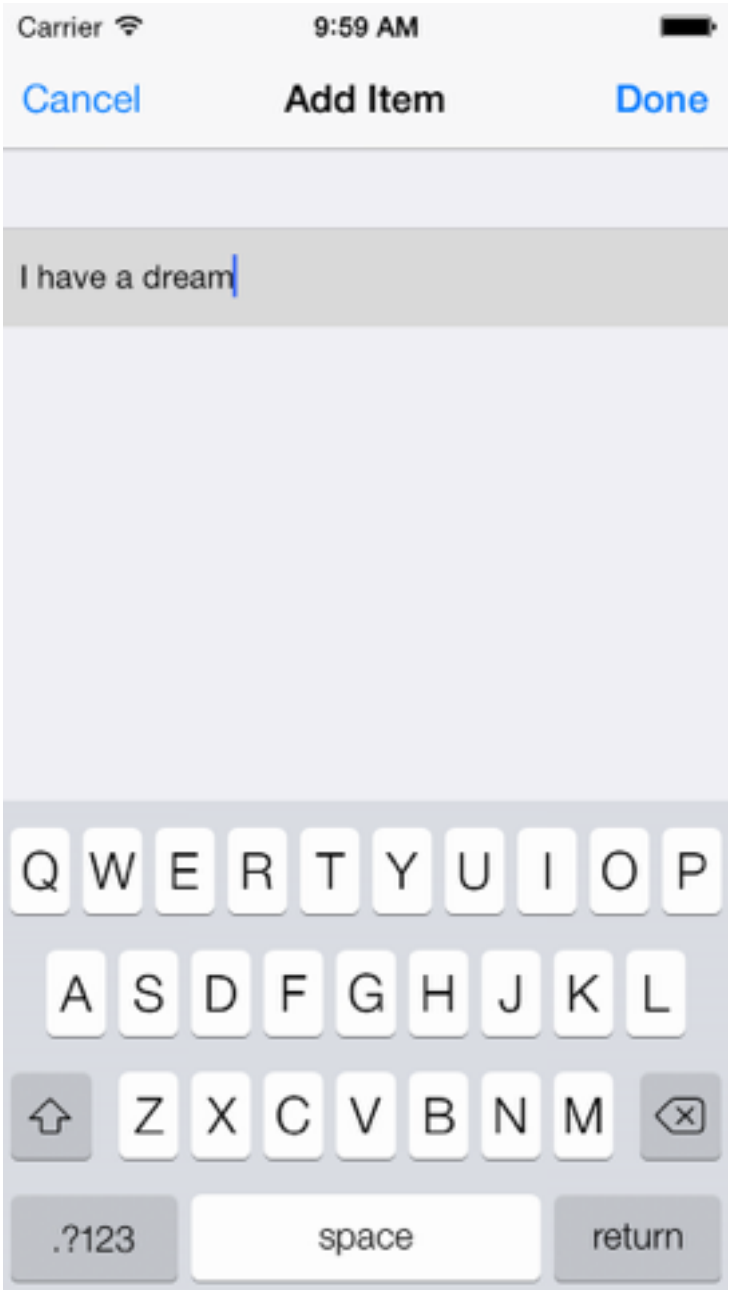



再次编译运行应用，然后触碰首界面的"+"按钮来打开Add Item 界面。触碰cell单元格，就可以看到底部出现一个虚拟键盘。

每当我们选中文本域的时候，虚拟键盘就会自动出现。我们可以输入自己希望的内容。




不过有一个小小的瑕疵，如果触碰文本框之外cell之内的区域，会发现该行变灰了。这是因为此时程序以为我们要选中当前行，实际上则不是我们想要的效果。




因此这里需要修补一下。
在Xcode中切换到AddItemViewController.m，然后在@end前添加一个方法如下：

```
-(NSIndexPath*)tableView:(UITableView *)tableView willSelectRowAtIndexPath:(NSIndexPath
*)indexPath{

    return nil;
}
```

这里我们再次认识到一个新的表视图代理方法。当用户触碰某一行的时候，表视图会向代理发送一条willSelectRowAtIndexPath消息：“代理你好，我现在要选中某一行了。”通过返回一个nil,代理对此的答复是：“不好意思，恐怕你没有这个权限这么做！”

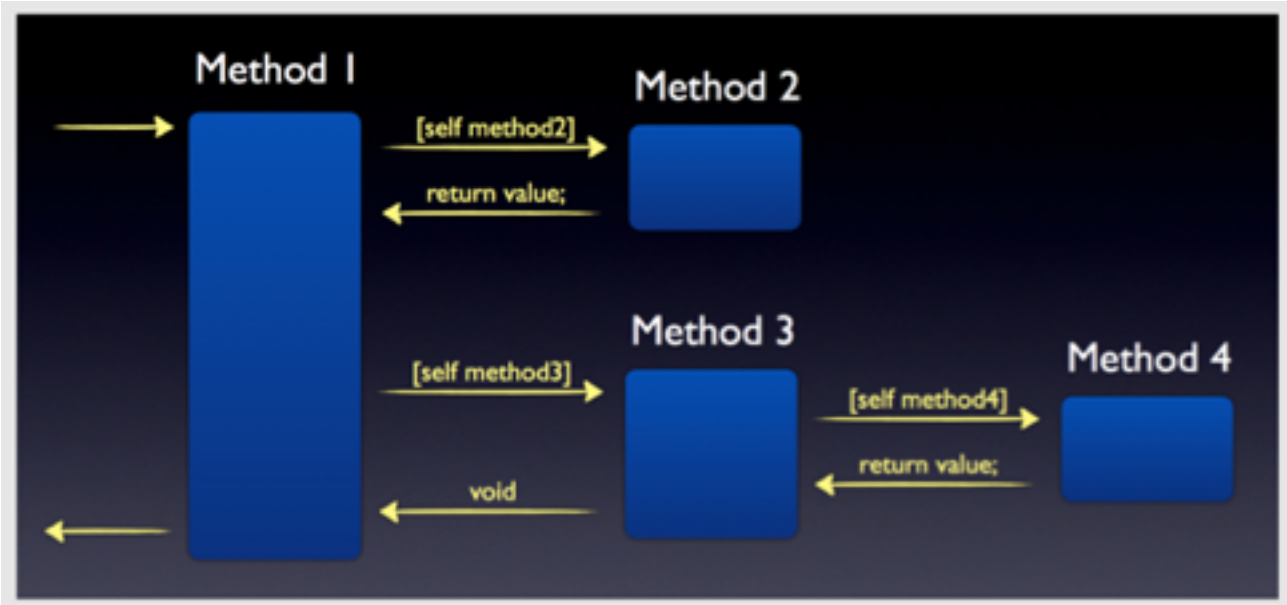
除此之外，为了防止这一行变灰，还需要做一件事。尽管我们现在已经禁止用户选中这一行，但cell却有一个Selection Color属性。在默认状态下被设置为Blue（因为历史的原因被命名为Blue,实际上却是灰色的）。记住row（行）和cell是两回事，对row禁止的事情cell可不在乎。

既然cell属于视觉元素，显然我们需要在storyboard中来设置。
切换到Main.storyboard，选中table view cell对象，然后在Xcode右侧面板中切换到Attributes inspector，将Selection 属性设置为None，就搞定了。

再次编译运行，就不会出现这种恼火的现象了。

稍事休息，让我们看看return 语句是怎么回事。

在之前的学习中我们已经几次接触到return 语句了。通常我们使用return 语句返回一个值到调用该方法的方法。看看下图：




某个方法会调用另一个方法，并接收所返回的值。你不能随意返回任意类型的值，所返回的值的数据类型必须是方法名称中所设置的。比如，tableView:numberOfRowsInSection:方法必须返回一个NSInteger类型的数值，因为方法名称之前有一个(NSInteger):

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
return 1;
}
```

NSInteger其实是int在Objective-C中的另一种名称，因此return 1就意味着返回了一个整数1.

不过你可以试着把代码改为：

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
return @"1";
}
```

毫无疑问编译器是会给出红色错误提示的，因为@"1"是一个字符串，而不是一个NSInteger整数。对于任何一个人类来说，两者看起来差不多，我们都可以理解。但Objective-C就不能忍受了。数据类型必须完全匹配，否则就会被给出红牌警告。

当然我们见到过类似下面的代码：

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
return [_items count];
}
```

这个返回语句是有效的，因为一个NSArray对象的count方法会返回一个NSUInteger类型的数值。在Objective-C中，我们经常会看到NSInteger,NSUInteger和int的混用，它们之前的区别不大，你也没有必要去深究，只需要知道这三种类型都同样代表整数。

tableView:cellForRowAtIndexPath:这个方法返回的是一个UITableViewCell对象：

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"ChecklistItem"]; ...
return cell;
}
```

返回一个NSInteger整数和返回一个UITableViewCell对象的区别在于，NSInteger不属于对象，而UITableViewCell则属于对象。从视觉上看，在方法名之前的括号中对于对象会有一个*，如果一个名称后面跟着一个*星号就表示你在和对象打交道。其它的被称为primitive types（基本数据类型）。
关于基本数据类型和对象之间的差别，我们会在下一系列的教程中涵盖，就目前来说，只需要盯住*就行了。

又比如，tableView:willSelectRowAtIndexPath:方法会返回一个NSIndexPath对象。但如果让其返回一个nil,就代表没有对象。

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

如果一个方法需要返回一个对象，我们可以返回一个对象，也可以返回nil。当然，返回nil是件有风险的事情。比如，如果你在cellForRowAtIndexPath方法中返回nil,那么应用就会崩溃。有些方法必须返回对象，有的则可以返回nil，那么苦逼程序猿怎么知道究竟是哪种情况呢？

最简单的方法，查查圣旨吧。圣旨？对，苹果的官方文档就是金科玉律和圣旨。一切你不能肯定的事情最好的方式就是通过苹果官方文档来确定。

关于willSelectRowAtIndexPath,方法圣旨中关于iOS的文档是这么宣布的：

“奉天承运皇帝诏曰：

Return Value: An index-path object that confirms or alters the selected row. Return an NSIndexPath object other than indexPath if you want another cell to be selected. Return nil if you don't want the row selected.

钦此。”

圣旨中说的清清楚楚明明白白真真切切：

1.我们可以返回和参数相同的index-path，这就意味着该行可以被选中。

2.我们可以返回另一个index-path，假如我们希望选中另一行。

3.我们可以返回nil，从而避免该行被选中。

而关于cellForRowAtIndexPath方法，圣旨中是这么宣布的：

“奉天承运皇帝诏曰：
Return Value: An object inheriting from UITableViewCell that the table view can use for the specified row. An assertion is raised if you return nil.

钦此”

上面说的很明白了，该方法必须返回一个恰当的table view cell对象。否则，An assertion is raised if you return nil. 如果你返回了nil，而不是一个有效的UITableViewCell对象，那么应用就会崩溃。因为你公然抗命，领旨不尊。
assertion是一种特殊的debug工具，用来检查确保你的代码中不会出现非法行为。如果出现了，应用就会崩溃，同时Xcode会帮忙提供一些错误信息。
关于bug和debug后面我们会逐渐感受到，现在先别害怕。Don’t panic.

除了返回对象和基本数据类型的方法，我们还碰到过另外一些不返回任何东东的方法，比如：

- (void)viewDidLoad
- (IBAction)cancel

这里的void意味着，该方法不会返回一个值到调用它的那一方。IBAction其实是void的同义词，只不过通过IBAction，Interface Builder就会知道它是和界面中某个视觉元素关联在一起的。因为IBAction和void是一回事，所以动作方法也不会返回数值。

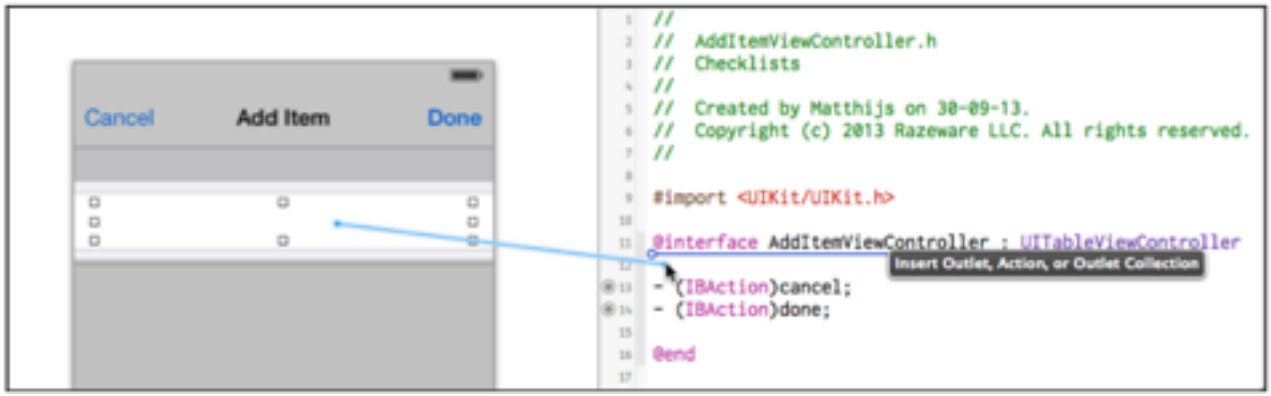
如果某个方法需要返回一个值，但你在代码中却没有这么做，那么Xcode就会给出提示：Control reaches end of non-void function。 应用或许会继续跑下去，但很有可能在某一个时空点就会崩盘，然后你就不知所措的枯坐一夜，举头望明月，对影成三人。

还是那句话，对任何警告和错误信息绝对不能轻易放过！

好了，脑补了这么久，又到了继续前进的时刻了。

现在用户已经可以往文本框里面输入文字了，接下来我们需要在用户触碰done按钮的时候，读取用户所输入的文字，并把它放到一个新建的ChecklistItem中，然后添加到我们的代办事务清单中。

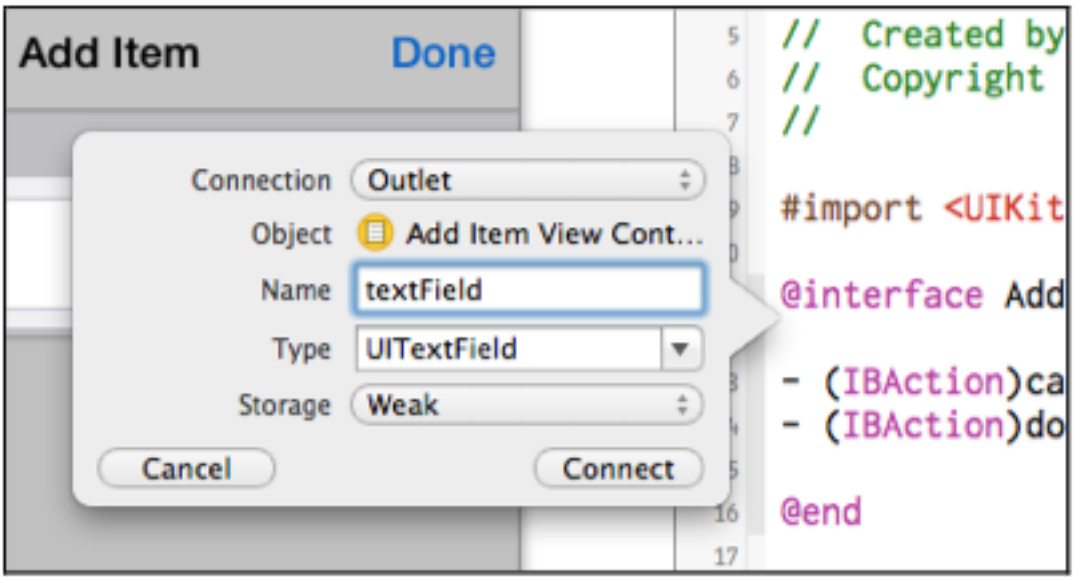
回到Xcode，打开Assistant editor，选中text field文本框对象，然后按住ctrl键拖出一条线到AddItemViewController.h的代码的@end前：




松开鼠标会看到一个弹出对话框，在里面输入相应信息：

Connection: 选择Outlet（注意不是action）
Name:输入textField
Type:选择UITextField
Storage:选择Weak

然后点击Connect就可以了。



这样我们就创建了一个和界面视觉元素关联的属性变量。

此时AddItemViewController.h的代码如下：

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

@interface AddItemViewController : UITableViewController
- (IBAction)cancel:(id)sender;
- (IBAction)done:(id)sender;
@property (weak, nonatomic) IBOutlet UITextField *textField;

@end
```

接下来需要修改和done按钮关联的动作方法内容。
切换到AddItemViewController.m，更改done方法的代码如下：

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

    NSLog(@"当前文本框中所输入的内容是：%@",self.textField.text);
    [self.presentingViewController dismissViewControllerAnimated:YES completion:nil];
}
```

注意：
在中英文输入切换的时候一定要小心，否则代码很容易出现低级错误！

编译运行应用，触碰"+"按钮，然后在新的界面中输入一些文本。触碰Done按钮，就会在Xcode的console中看到类似下面的消息：


好了，成功搞定！现在你对NSLog()这个函数应该已经相当熟悉了。

之前我们用过%d和%f格式符，分别代表整数和浮点数。而这里的%@格式符则可以用来打印一个对象的值，这里就是文本框中的文本内容。

在我们继续学习之前，先完善一下Add Item界面的设计。
首先，我们希望不必自己触碰文本框来打开虚拟键盘，而希望在该界面打开时就会自动打开虚拟键盘。

为此在Xcode中切换到AddItemViewController.m，然后添加一个新的方法如下：

```
-(void)viewWillAppear:(BOOL)animated{

    [super viewWillAppear:animated];

    [self.textField becomeFirstResponder];
}
```

viewWillAppear方法是iOS所提供的视图控制器方法，其作用是当界面跳转到当前界面但还没用显示出其中的内容时执行一些任务。通过给textField对象发送一个becomeFirstResponsder消息，我们通知它"成为当前的控制焦点"。在iOS术语中，当前的控件成为first responder（第一响应者）。

编译运行应用，现在一旦进入Add Item界面就会自动滑出一个虚拟键盘。

接下来对输入文本框的设计做一些微调。

在Xcode中切换到Main.storyboard，选中text field对象，然后在Xcode右侧的面板中切换到Attributes Inspector，然后设置以下属性：

Place holder:Name of the Item

Font: System 17

Adjust to Fit: 取消勾选

Capitalization: Sentences

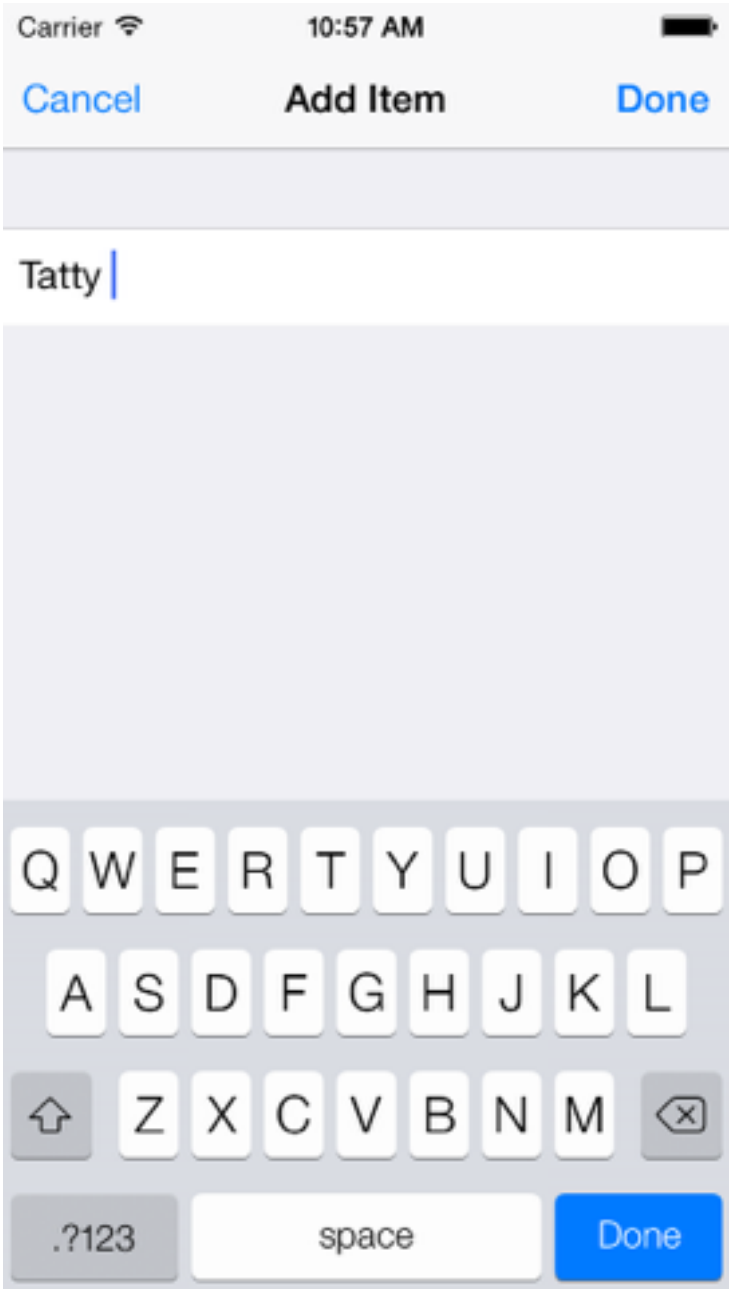
Return Key: Done

当然，还有一些其它选项可供设计的，比如如果你希望用户在这里只能输入数字，那么可以把Keyboard设置为Number Pad。这里我们用Default默认键盘就可以了。

此外，我们还更改了键盘上回车键的名称，默认显示return，这里将其设置为Done。不够这只是视觉上的显示，我们还需要添加动作方法。

确保选中text field对象，然后在右侧切换到Connection inspector。从Did End on Exit事件拖出一条线到视图控制器，然后选择done动作方法即可。

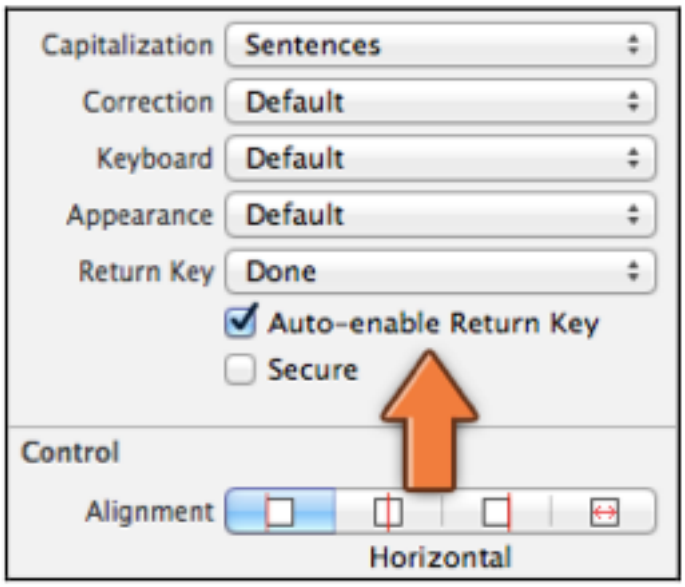
编译运行应用，触碰虚拟键盘上的Done键也会关闭界面，并在console中输出文本了。




但凡涉及到让用户输入信息的界面，我们都要注意验证的问题。比如，如果用户一打开界面就关闭了肿么办？我们不可能添加一个空的代办事务，为了避免这一点，需要在用户输入文字前禁用Done按钮。

当然，这里我们有两个done按钮，其中一个在虚拟键盘上，另一个在导航栏上。首先我们来对付虚拟键盘上的这个done按钮。

选中text field，在Xcode右侧切换到Attributes inspector，然后选中Auto-enable Return Key>好了，就这么简单。当我们编译运行应用的时候，如果没有输入内容，虚拟键盘上的done键就无效。试试看！




至于导航栏上的done键，要做的事情稍微多点。我们必须在用户的某一次按键后检查内容是否为空。如果是就禁用该按钮。用户随时可以触碰cancel按钮，但done按钮只有在有文字内容的情况下才可用。

为了随时监听文本框内容的变化-可能来自虚拟键盘的输入，也可能是拷贝/粘贴- 我们必须让视图控制器成为文本框的代理。文本框将会向代理发送事件通知，而代理（也就是AddItemViewController）将会对这些事件做出相应，并采取适当的行动。

之前在UITableView的使用中我们已经接触过代理。文本框对象，UITextField也有自己的代理。这样我们的视图控制器就分别扮演了两个对象的代理。

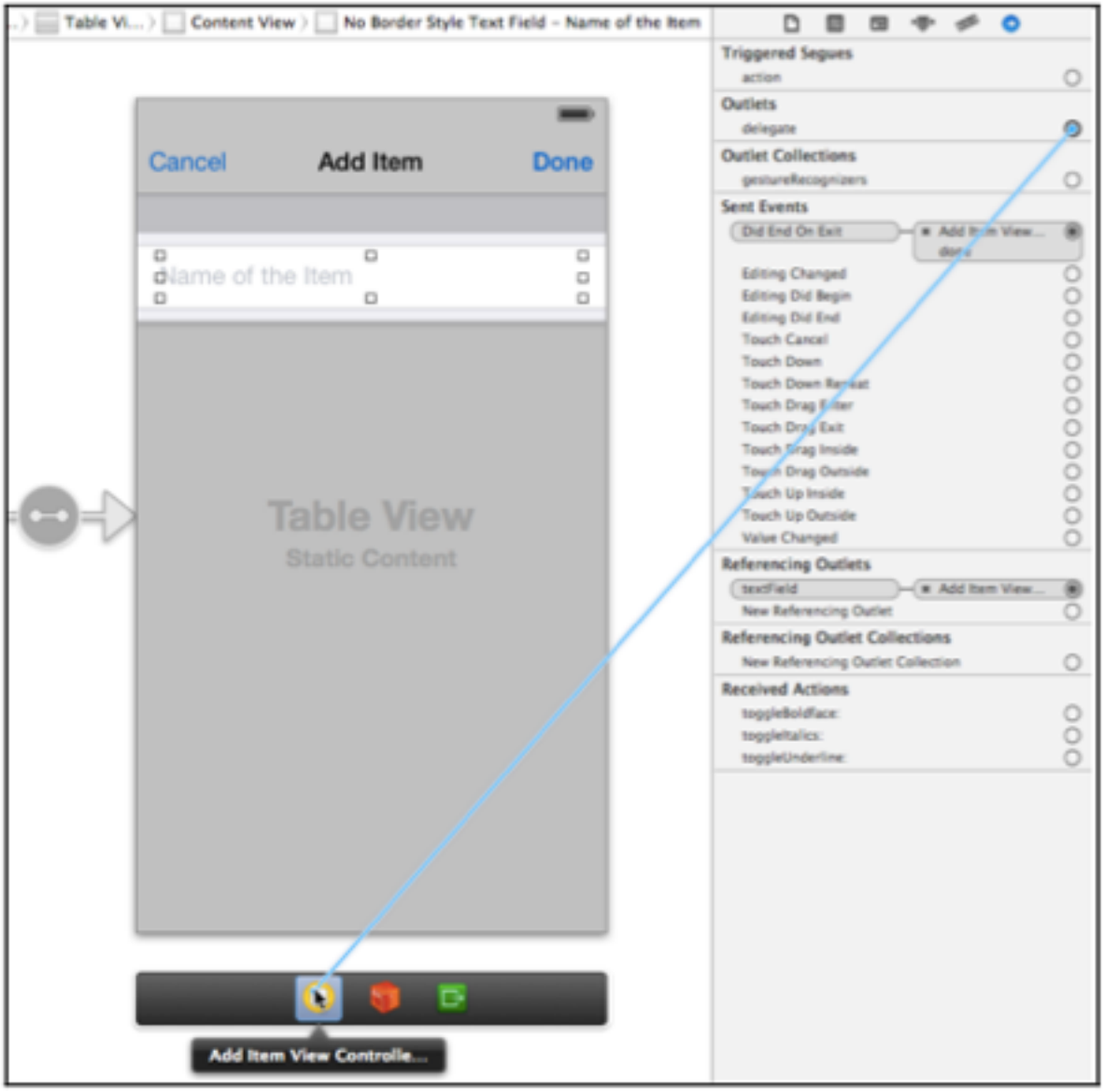
在Xcode中切换到AddItemViewController.h，然后在@interface部分更改相关代码如下：

@interface AddItemViewController : UITableViewController<UITextFieldDelegate>

通过添加这个尖括号中的内容，视图控制器向世人宣传：现在我是text field对象的代理了，有神马关于它的生意都来联系我吧~

当然，如果我们不告诉text field这一点，那么视图控制器也只不过是单相思。

回到Main.storyboard，选中text field，在Xcode右侧切换到connections inspector，然后从delegate后面的小圆圈拖出一条线到视图控制器的图标：




这样双方都知道这么回事了，AddItemViewController向世人宣布自己将承担text field的代理工作，而text field也通过storyboard了解了视图控制器的这个工作，以后就轻松了。

接下来打开Assistant editor,确保可以看到AddItemViewController.h的代码。然后按住ctrl键，从done按钮拖出一条线到AddItemViewController.h，松开，将新的outlet命名为doneBarButton。

这样就添加了一个新的IBOutlet属性：

```
@property (weak, nonatomic) IBOutlet UIBarButtonItem
 *doneBarButton;
```

注意之前我们为done按钮添加过一个IBAction的动作方法，而这里添加的则是一个属性变量，这是因为我们需要更改这个按钮的属性状态。

接着在Xcode中切换到AddItemViewController.m，在@end前添加一个新的方法：

```
-(BOOL)textField:(UITextField *)textField shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string{

   NSString *newText = [textField.text stringByReplacingCharactersInRange:range
withString:string];

   if([newText length] >0){

      self.doneBarButton.enabled = YES;
   }else{
      self.doneBarButton.enabled = NO;
   }

   return  YES;
}
```

以上我们用到了UITextField的一个代理方法之一。每当用户改变了文本框的内容时都会调用它，不管是通过虚拟键盘还是拷贝/粘贴。

仔细看看，首先我们需要获取新的文本内容：

```
   NSString *newText = [textField.text stringByReplacingCharactersInRange:range
withString:string];
```

stringByReplacingCharactersInRange这个代理方法并不是提供新的文本内容，而是文本内容中的哪一部分需要被替代（range），以及需要被替换成的文本(string)。因此我们需要根据文本框的文字和替代内容来获取最终的新文本内容。通过该方法，我们获取了一个保存在名为newText本地变量的新字符串对象。

接下来我们根据新文本内容的长度来判定是否为空，并根据该属性来开启或禁用done按钮：

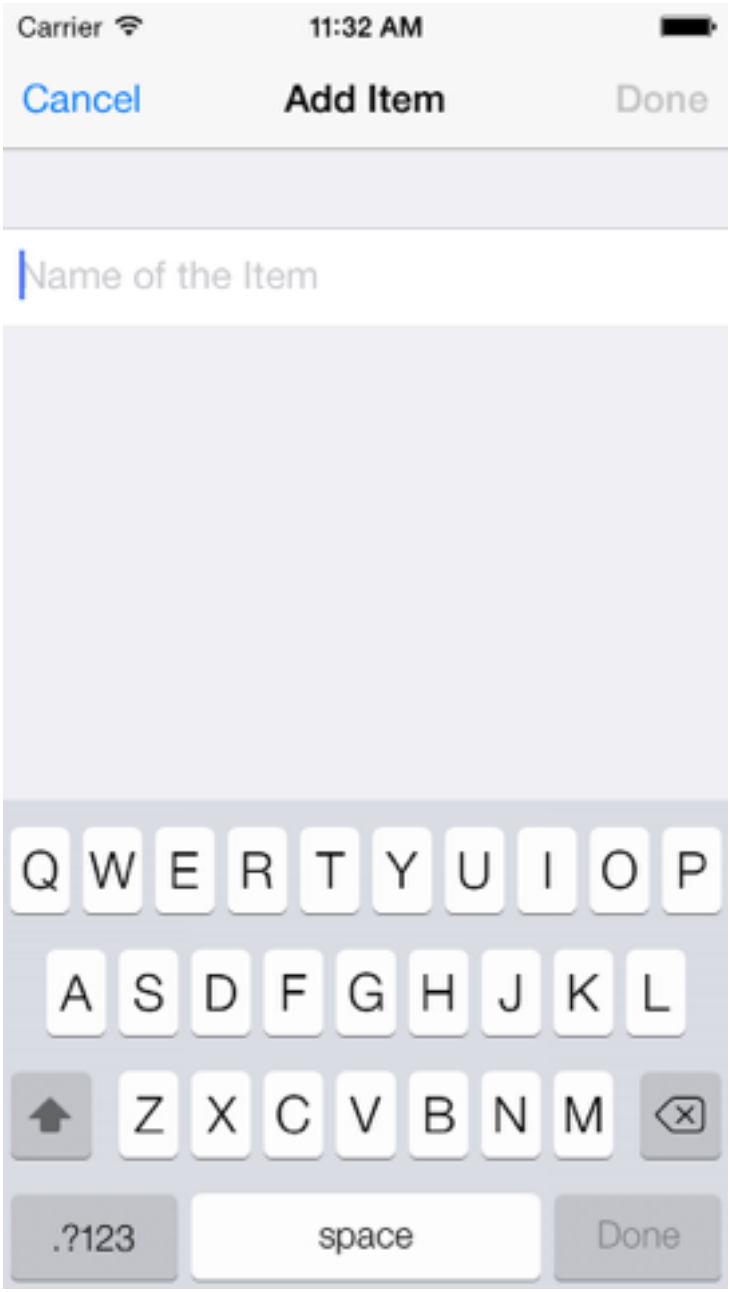
```
  if([newText length] >0){

      self.doneBarButton.enabled = YES;
   }else{
      self.doneBarButton.enabled = NO;
   }
```

编译运行应用，然后试着在文本框中输入一些文字。尝试删除这些文字，你会看到导航栏中的done按钮在文本框内容为空时会被禁用。

不过还有一个小瑕疵，初始状态下done按钮是被启用的，但此时文本框中却没有任何内容。

在storyboard中选择done按钮，然后切换到Attributes inspector，取消Enabled的勾选。

再次编译运行，此时初始状态下的done按钮就不可用了。




结束本章学习之前，我们最后来学习一个小的语法技巧：

上面的设置done按钮启用状态的方法可以改写为；

```
- (BOOL)textField:(UITextField *)theTextField shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
{
NSString *newText = [theTextField.text
stringByReplacingCharactersInRange:range withString:string];
self.doneBarButton.enabled = ([newText length] > 0);
return YES;
}
```

这里我们把if语句替换成了一行语句：

self.doneBarButton.enabled = ([newText length] > 0);

之前的写法是：
if ([newText length] > 0) {
// 如果长度大于0，那么
} else {
// 如果文本内容长度等于0，那么
}

最后了解下Objective-C中的关系运算符

> 大于
< 小于
>= 大于或等于
<= 小于或等于
== 等于
!= 不等于

千万要记住=和==的区别，==用来判断两个变量的值是否相同，而=则是赋值。

小技巧，每当你的代码类似下面这样
if (some condition) { something = YES;
} else { something = NO;
}
都可以用一行代码替代，

something = (some condition);

实际上我个人不推荐这种写法，但是如果看到别人这么写可千万别奇怪。

漫长的一章终于完成了，又到了发送福利时间~

送给妹子，华仔的军大衣淘宝同款畅销，至于XD们，你们还是不要随便跟风了。

妹子一张