iOS 5 Programming Cookbook

第二章

使用控制器和视图（下）

版本 1.0

翻译时间：2012-05-28

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun
DevDiv 校对： laigb kyelup
DevDiv 编辑： BeyondVincent

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问网站 www.devdiv.com 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3 和 xiebaochun 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译各章节汇总

目录

前言

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_前言

第 1 章 基础入门

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第一章_基础入门

第 2 章 使用控制器和视图(上)

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(上)

第 2 章 使用控制器和视图(下)

2.13. 使用 UILabel 显示静态文本

2.13.1. 问题

想要给用户显示静态文本，并且控制文本的字体和颜色。

2.13.2. 方案

使用 UILabel 类。

2.13.3. 讨论

在 iOS 中的任何地方都存在标签。几乎在每个应用程序中都能看到它们，除了那些使用 OpenGL ES（而不是在 iOS 构架）绘出来的游戏类标签。在 iPhone 里的设置应用中有一个标签的范例，如图 2-41 所示。

图 2-41.设置应用中的标签

可以看到设置应用中标签上显示的文本，比如：General,iCloud, Twitter, Phone, FaceTime 等。要创建一个

标签，需要实例化 UILabel 类的一个对象。可以通过这个标签的文本属性来设置（或获取）标签中的文本。先在视图控制的头文件中定义一个标签。如下所示：

```
#import <UIKit/UIKit.h>
@interface Displaying_Static_Text_with_UILabelViewController
: UIViewController
@property (nonatomic, strong) UILabel *myLabel;
@end
```

然后结合一下代码：

```
#import "Displaying_Static_Text_with_UILabelViewController.h"
@implementation Displaying_Static_Text_with_UILabelViewController
@synthesize myLabel;
...
```

在 viwDidLoad 中实例化这个标签，如下代码：

```
(void)viewDidLoad
{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
CGRect labelFrame = CGRectMake(0.0f,0.0f,100.0f,23.0f);
self.myLabel = [[UILabel alloc] initWithFrame:labelFrame];
self.myLabel.text = @"iOS 5 Programming Cookbook";
self.myLabel.font = [UIFont boldSystemFontOfSize:14.0f];
self.myLabel.center = self.view.center;
[self.view addSubview:self.myLabel];
}
```

现在运行此程序，看看会发生什么情况（如图 2-24 所示）。

图 2-24.标签太小不能将文本内容完全显示出来

可以看到，标签中的内容由于标签的宽度限制而被尾随的句号截断了。如果只是想着如何把宽度增大，那么为何不考虑一下增加高度呢？倘若将文本换到下一行呢？好的，那么就把高度 23.0f 改成 50.0f，如下所示。

```
CGRect labelFrame = CGRectMake(0.0f,
0.0f,
100.0f,
50.0f);
```

如果现在运行这个程序，将会得到图 2-42 所示的相同的结果。但是，你会问为什么增加了标签高度文本不会换到下一行呢？原来是因为 UILabel 类有一个叫 numberOFLInes 的属性，它需要调整到标签需要将文本换成的行数，防止它跑到水平线以外。假如把它设置成 3，它会告诉标签，如果文本在一行内不能完全显示就把文本转换成 3 行。如下所示。

```
self.myLabel.numberOfLines = 3;
```

假如现在运行这个程序，将会得到预期的结果。如图 2-43 所示。

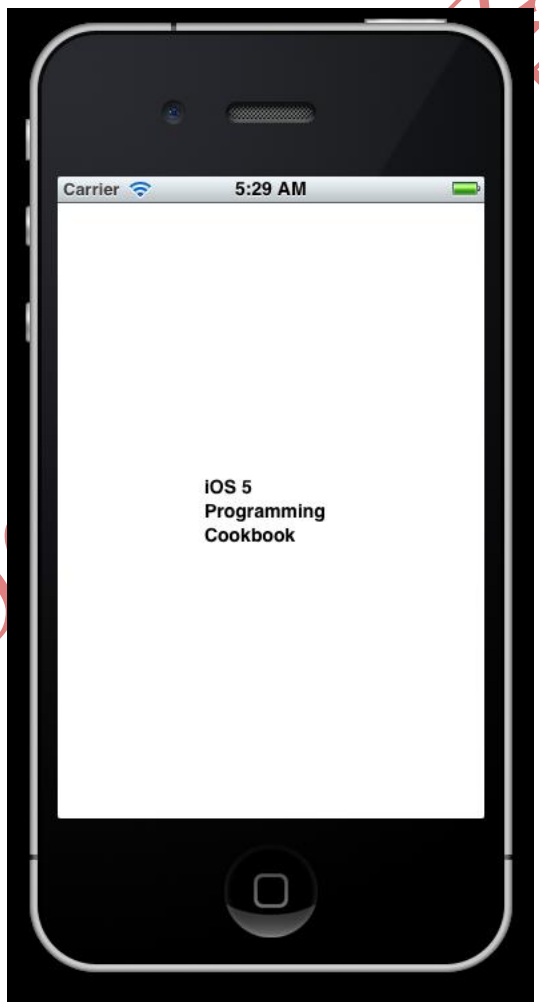


图 2-43 标签中的内容转换成 3 行。

在其它的情形下，也许不知道标签中的内容需要几行才能显示完全。在这种情况下，需要将 numberOFLInes 这个属性值设为 0。

假如想要使你的标签视图架保持静态的，并且标签里的字体能自动调整到适合标签的边界。这就需要将标签属性 adjustsFontSizeToWidth 设置成 YES。例如，假设标签高度是 23.0f（如图 2-42 所示的那样），我们可以将标签高度调整到适合边界的高度。如下代码所示。

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
CGRect labelFrame = CGRectMake(0.0f,
0.0f,
100.0f,
23.0f);
self.myLabel = [[UILabel alloc] initWithFrame:labelFrame];
self.myLabel.adjustsFontSizeToFitWidth = YES;
self.myLabel.text = @"iOS 5 Programming Cookbook";
self.myLabel.font = [UIFont boldSystemFontOfSize:14.0f];
self.myLabel.center = self.view.center;
[self.view addSubview:self.myLabel];
}
```

2.13.4. 参考

XXX

2.14. 使用 UITextField 接受用户文本输入

2.14.1. 问题

想要接受用户界面输入的文本

2.14.2. 方案

使用 UITextField 类

2.14.3. 讨论

文本视图很像一个标签，在文本视图里可以显示文本，但是一个文本视图也可以在运行时接受文本输入。这里有两个文本视图的一个例子，是在 iPhone 的设置程序里的 Twitter 部分。如图 2-44 所示。

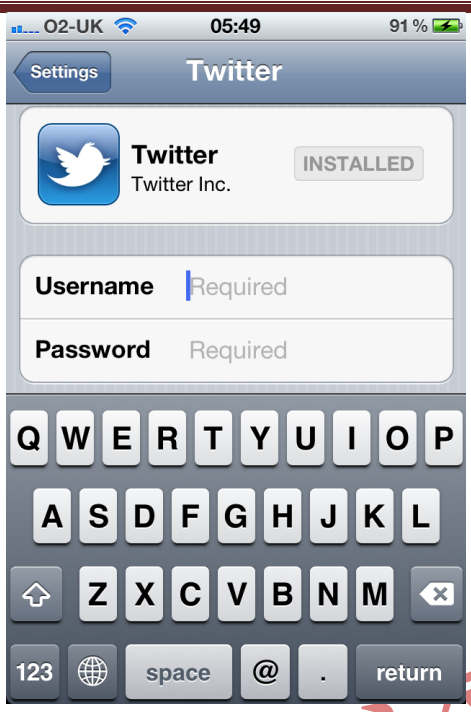


图 2-44 可输入的用户和密码文本视图

一个文本视图只允许一行文本的输入或显示，默认的文本视图高度为 31 像素。在 Interface Builder 里，文本视图是不能修改的，但是如果是在代码里创建的文本视图是可以修改的，但是不能改变这个文本视图的行数，默认行数是 1。

现在可以在视图控制器的头文件里定义文本视图了，如下所示。

```
#import <UIKit/UIKit.h>
@interface Accepting_User_Text_Input_with_UITextFieldViewController
: UIViewController
@property (nonatomic, strong) UITextField *myTextField;
@end
```

同时结合下面的 myTextField 属性代码：

```
#import "Accepting_User_Text_Input_with_UITextFieldViewController.h"
@implementation Accepting_User_Text_Input_with_UITextFieldViewController
@synthesize myTextField;
...
```

然后创建文本视图：

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
CGRect textFieldFrame = CGRectMake(0.0f,
0.0f,
200.0f,
31.0f);
self.myTextField = [[UITextField alloc]
initWithFrame:textFieldFrame];
self.myTextField.borderStyle = UITextBorderStyleRoundedRect;
self.myTextField.contentVerticalAlignment =
```

```
UIControlContentVerticalAlignmentCenter;
self.myTextField.textAlignment = UITextAlignmentCenter;
self.myTextField.text = @"Sir Richard Branson";
self.myTextField.center = self.view.center;
[self.view addSubview:self.myTextField];
}
```

先看看结果然后我们会看到代码详细的信息，如下图所示。

图 2-45 一个文本居中的文本视图

要创建文本视图，我们需要用到 UITextField 的各种属性。它们包括：

borderStyle

这个属性属于 UITextBorderStyle 类型，它能指定文本视图如何显示他的边视图。

ContentVerticalAlignment

这个属性属于 UIControlContenVerticalAlignment 类型，它会告诉文本如何在文本视图里纵向显示。假如我们不设置文本在文本视图里纵向居中，它将会默认显示在文本视图的左上角。

textAlignment

这个属性属于 UITextAlignment 类型，它能指定文本视图里的文本水平对齐。在这个例子中，我们使文本水平方向居中（垂直方向也是）。

text
这是一个可读写属性。你可以对它进行读取和写入操作。读取时它会返回文本视图里的当前文本，写入时它会使文本视图的文本变为你要写入的文本。

一个文本视图会发送一个委托信息给它的委托对象。当用户开始在文本视图里编辑文本，完成编辑，在文本视图里输入任何符号时，这些信息都会被发送。为了通知这些事件，我们需要给对象设置文本视图的 delegate 属性。这个文本视图的委托必须遵守 UITextFieldDelegate 协议。如下所示。

```
#import <UIKit/UIKit.h>
@interface Accepting_User_Text_Input_with_UITextFieldViewController
: UIViewController <UITextFieldDelegate>
@property (nonatomic, strong) UITextField *myTextField;
@end
```

按住电脑上的命令键并在 Xcode 里点击 UITextFieldDelegate 协议。将会看到这个协议提供的所有控制方法。下面是这些方法被调用是的描述。如下。

textFieldShouldBeginEditing：
这是返回一个 BOOL 值的方法，它会告诉文本视图（这个方法的参数）是否允许用户开始在文本视图编辑。假如不想让用户编辑这个文本视图则将返回 NO。

textFieldDidBeginin：
当用户开始编辑文本视图时这个方法将会被调用。

textFieldShouldEndEditing：
这个方法返回一个 BOOL 值，它将告诉文本视图是否结束当前的编辑进程。假如返回 NO，用户将不能终止文本的编辑。

textFieldDidEndEditing:
当文本视图的编辑进程终止时将会被调用。

textField：shouldChangCharacterInRange：replacementString：
任何时候文本视图里的文本被修改都会调动这个方法。方法返回的是一个布尔值。假如返回 YES，说明允许修改文本。如果返回 NO，文本视图里的文本的修改将不会被通知和发生。

textFieldShouldClear：
每个文本视图都有一个 Clear 按钮，通常是一个圆形 X 按钮。当用户按下这个按钮时，文本视图里的内容将会自动清除。但是我们需要人工启动清除按钮。如果已经启动了清除按钮并返回 NO 方法，它将会让用户感觉程序没有正常工作。所以想清楚你在做什么，因为用户看见一个清除按钮后按下它却看不见文本视图里的文本没有被清除。

textFieldShouldReturn：
当用户在键盘上按下 Return 或 Enter 键时将会调用这个方法，尝试隐藏键盘，你需要将这个文本视图注册为这个方法的第一个响应者。

现在结合本节和 2.13 节的内容 在文本视图里创建一个标签并将文本视图里的字符数显示在标签里。现在开始编写头文件，如下所示。

```
#import <UIKit/UIKit.h>
@interface Accepting_User_Text_Input_with_UITextFieldViewController
: UIViewController <UITextFieldDelegate>
@property (nonatomic, strong) UITextField *myTextField;
@property (nonatomic, strong) UILabel *labelCounter;
@end
```

结合下面的代码：

```
#import "Accepting_User_Text_Input_with_UITextFieldViewController.h"
@implementation Accepting_User_Text_Input_with_UITextFieldViewController
@synthesize myTextField;
@synthesize labelCounter;
...
```

然后创建文本视图，标签和我们需要的文本视图委托方法。我们跳过所有 UITextFieldDelegate 方法的编译，因为在这个例子里我们不需要它们的全部方法。如下所示。

```
- (void) calculateAndDisplayTextFieldLengthWithText:(NSString *)paramText{
NSString *characterOrCharacters = @"Characters";
if ([paramText length] == 1){
characterOrCharacters = @"Character";
}
self.labelCounter.text = [NSString stringWithFormat:@"%lu %@",
(unsigned long)[paramText length],
characterOrCharacters];
}
- (BOOL) textField:(UITextField *)textField
shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string{
BOOL result = YES;
if ([textField isEqual:self.myTextField]){
NSString *wholeText =
[textField.text stringByReplacingCharactersInRange:range
withString:string];
[self calculateAndDisplayTextFieldLengthWithText:wholeText];
}
return result;
}
- (BOOL)textFieldShouldReturn:(UITextField *)textField{
[textField resignFirstResponder];
return YES;
}
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
CGRect textFieldFrame = CGRectMake(38.0f,
30.0f,
220.0f,
31.0f);
self.myTextField = [[UITextField alloc]
initWithFrame:textFieldFrame];
self.myTextField.delegate = self;
self.myTextField.borderStyle = UITextBorderStyleRoundedRect;
self.myTextField.contentVerticalAlignment =
UIControlContentVerticalAlignmentCenter;
self.myTextField.textAlignment = UITextAlignmentCenter;
self.myTextField.text = @"Sir Richard Branson";
[self.view addSubview:self.myTextField];
CGRect labelCounterFrame = self.myTextField.frame;
labelCounterFrame.origin.y += textFieldFrame.size.height + 10;
self.labelCounter = [[UILabel alloc] initWithFrame:labelCounterFrame];
[self.view addSubview:self.labelCounter];
[self calculateAndDisplayTextFieldLengthWithText
[self calculateAndDisplayTextFieldLengthWithText:self.myTextField.text];
}
```

我们要特别注意的是我们是在 textField:shouldChangeCharactersInRange:replacementString:方法里进行的。这里，我们声明并使用了一个叫 wholeText 的变量。当这个方法被调用时，参数 replacementString 指定了用户在文本视图里已经输入的字符串。也许你会想，用户可以一次只输入一个字符，为什么这个参数是一个字符串呢？但是不要忘记了，用户可以粘贴一整块文本到文本视图里，所以这个参数必须是一个字符串。参数 shouldChangeCharactersInRange 指定用户在哪里输入文本（不过这是在文本视图里面的文本而言的）。所以使用这两个参数，我们将会在文本视图里创建一个第一次读取整个文本的字符串，并且在旧的文本里给出新文本所需空间的范围。有了这些，我们要在 textField:shouldChangeCharactersInRange:replacementString:这个方法返回 YES 之后想出在文本视图里需要显示的文本。如图 2-46 所示是运行的结果。

图 2-46 文本视图委托信息的响应

此外，一个文本视图还能显示一个占位符。一个占位符是用户还没输入任何文本之前在文本视图里显示的文本，但是文本视图的文本属性是空的。这可以是任何你想要的字符串，而且设置它有助于给用户提供关于到底文本视图什么的一个指示。许多开发者使用占位符来提示用户他们可以在文本视图输入何种类型的值。例如，在图 2-44，两个文本视图（用户名和密码）需要两个占位符。你可以使用文本视图的 placeholders 这个属性去设置或取得当前的占位符。这里有一个例子，如下所示。

```
self.myTextField = [[UITextField alloc]
initWithFrame:textFieldFrame];
self.myTextField.delegate = self;
self.myTextField.borderStyle = UITextBorderStyleRoundedRect;
self.myTextField.contentVerticalAlignment =
UIControlContentVerticalAlignmentCenter;
self.myTextField.textAlignment = UITextAlignmentCenter;
self.myTextField.placeholder = @"Enter text here...";
```

```
[self.view addSubview:self.myTextField];
```

下面是运行的结果。

图 2-47 当文本视图没有文本时显示一个占位符

文本视图有两个相对的属性，它们分别是 leftView 和 rightView。这两个属性是 UIView 类型并且是可读写的。根据他们的名字可以推断出，只要给要它们提供一个视图它们将会显示在文本视图的左边和右边。有一个地方你也许会用到左边的视图，例如，假如你要显示一个货币文本视图，在哪里你想在左视图里用 UILabel 显示用户当前国家的货币。下面是怎样完成这项工作的代码，如下所示。

```
UILabel *currencyLabel = [[UILabel alloc] initWithFrame:CGRectZero];
currencyLabel.text = [[[NSNumberFormatter alloc] init] currencySymbol];
currencyLabel.font = self.myTextField.font;
[currencyLabel sizeToFit];
self.myTextField.leftView = currencyLabel;
self.myTextField.leftViewMode = UITextFieldViewModeAlways;
```

假如我们只声明了一个视图一个文本视图的 leftView 或 rightView 属性，这些视图将默认不会中自动显示。这些视图是否显示在屏幕上是依据管理它们显示的模式决定的，可以使用 leftViewMode 和 rightViewMode 这两个属性控制这个模式。这个模式是 UITextFieldViewMode：类型。如下所示。

```
typedef enum {
UITextFieldViewModeNever,
```

```
UITextFieldViewModeWhileEditing,
UITextFieldViewModeUnlessEditing,
UITextFieldViewModeAlways
} UITextFieldViewMode;
```

例如，假如你把左视图模式设置成 UITextFieldViewModeWhileEditing，假如已经声明了一个值给它，当用户编辑这个文本视图时左视图将会显示。反过来，假如你把它设置成 UITextFieldViewModeUnlessEditing，当用户没有在文本视图里编辑文本时左视图将会显示，但是一旦开始编辑，做视图将会消失。下面看看这些代码在模拟器里运行的情况，如下图所示。

图 2-28 一个具有左视图的文本视图

2.14.4. 参考

xxx

2.15. 使用 UITextView 显示文本域

2.15.1. 问题

想在一个滑动视图里使用户界面显示多行文本。

2.15.2. 方案

使用 UITextView 类

2.15.3. 讨论

IUTextView 类可以显示多行文本而且具有可滑动内容，这意味着如果文本内容超过了文本视图的边界，文本视图的内部部件将会允许用户根据自己的需求上下滑动文本来查看文本的不同部分。在 iOS 应用程序中使用到的一个文本视图例子是 iPhone 里的 Notes 应用程序。如下所示。

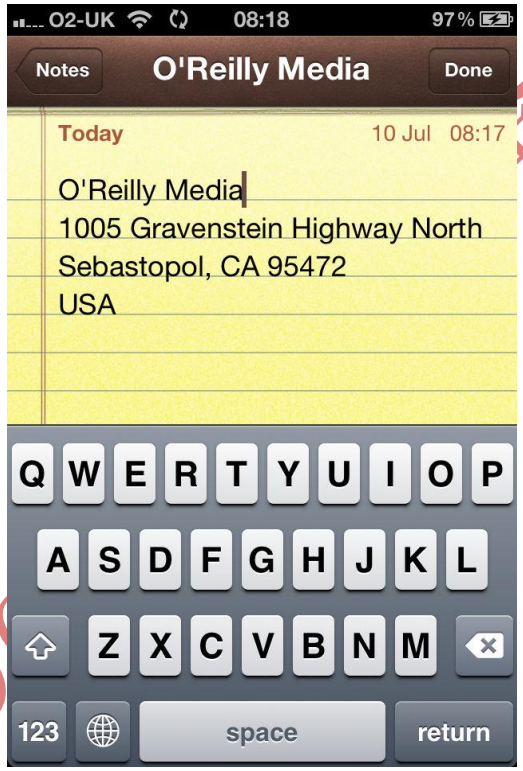


图 2-49 iPhone 里的 Notes 程序使用文本视图来提供文本

现在来创建一个文本视图并看看它是怎么样工作的。先在视图控制器的头文件里声明文本视图。如下所示。

```
#import <UIKit/UIKit.h>
@interface Displaying_Long_Lines_of_Text_with_UITextViewViewController
: UIViewController
@property (nonatomic, strong) UITextView *myTextView;
@end
```

接下来综合下面的文本视图，如下。

```
#import "Displaying_Long_Lines_of_Text_with_UITextViewViewController.h"
@implementation Displaying_Long_Lines_of_Text_with_UITextViewViewController
@synthesize myTextView;
```

...

现在是时候创建文本视图本身了。我们将要使文本视图跟视图控制器的视图大小一样大。

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
self.myTextView = [[UITextView alloc] initWithFrame:self.view.bounds];
self.myTextView.text = @"Some text here...";
self.myTextView.font = [UIFont systemFontOfSize:16.0f];
[self.view addSubview:self.myTextView];
}
```

现在在 iOS 模拟器里运行这个程序，看看结果如何。

图 2-50 文本视图占据了屏幕的全部边界

现在假如你点击文本视图，你会发现一个虚拟键盘将会从屏幕底部弹出来，遮盖了文本视图几乎一半的区域。这意味着,,假如用户开始输入文本并达到文本视图中间时，用户将会被会看不见的之后输入的文本。如下图所示。

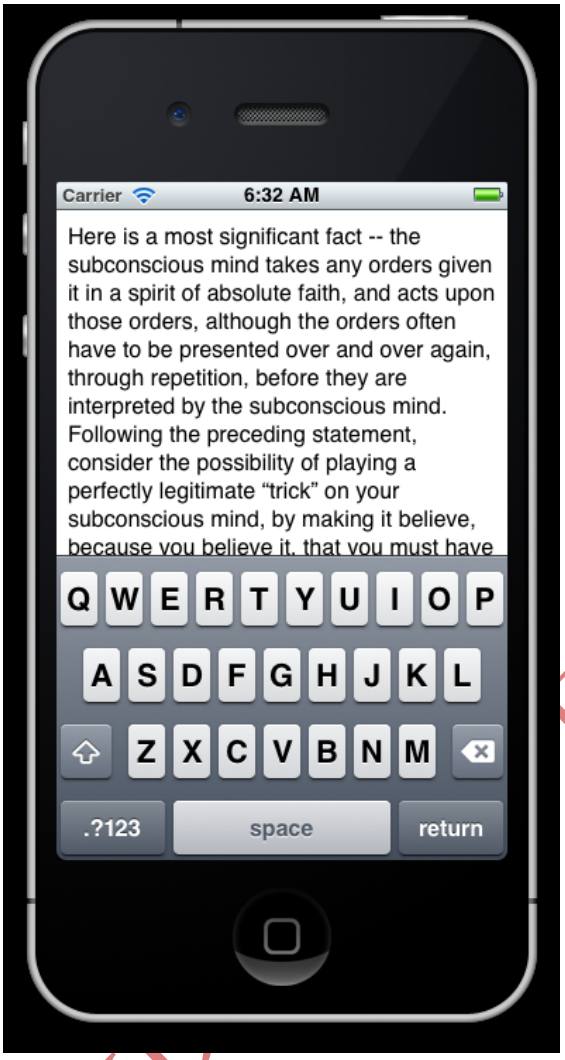


图 2-51 键盘占据了文本视图的一半范围

为了能弥补这个缺陷，我们需要监听一些待定通知，如下：

UIKeyboarWillShowNotification
当键盘正准备显示子屏幕上时这个通知将会被发送给任何部件，比如文本视图等等。

UIKeyboardDidShowNotification
当键盘已经显示在屏幕上时将会发送这个通知。

UIKeyboardWillHideNotification
当键盘即将被隐藏时将会发送这个通知。

UIKeyboardDidHideNotification
当键盘完全隐藏时将会发送这个通知。

所以我们的策略就是去发现键盘什么时候准备在屏幕上显示然后以何种方式调整视图。为了实现这个策略，我们将会用到 UITextView 的 contentInset 这个属性来指定文本内容的边沿到文本视图的顶部，左边，底部和右边的距离。

键盘通知包含了一个字典，可以通过 userInfo 这个属性使用它，他指定了键盘在屏幕上的边界。这个属性属于 NSDictionary 类型。在这个字典里，其中有一个名为 UIKeyboardFrameEndUserInfoKey 的关键字，它包含了一个 NSValue 类型的对象，这个对象包含一个当键盘完全显示时的矩形边界。这个矩形区域被标记为一个 CGRect。

```
- (void) handleKeyboardDidShow:(NSNotification *)paramNotification{
/* Get the frame of the keyboard */
NSValue *keyboardRectAsObject =
[[paramNotification userInfo]
objectForKey:UIKeyboardFrameEndUserInfoKey];
/* Place it in a CGRect */
CGRect keyboardRect;
[keyboardRectAsObject getValue:&keyboardRect];
/* Give a bottom margin to our text view as much
/* Give a bottom margin to our text view as much
as the height of the keyboard */
self.myTextView.contentInset =
UIEdgeInsetsMake(0.0f,
0.0f,
keyboardRect.size.height,
0.0f);
}
- (void) handleKeyboardWillHide:(NSNotification *)paramNotification{
/* Make the text view as big as the whole view again */
self.myTextView.contentInset = UIEdgeInsetsZero;
}
- (void) viewWillAppear:(BOOL)paramAnimated{
[super viewWillAppear:paramAnimated];
[[NSNotificationCenter defaultCenter]
addObserver:self
selector:@selector(handleKeyboardDidShow:)
name:UIKeyboardDidShowNotification
object:nil];
[[NSNotificationCenter defaultCenter]
addObserver:self
selector:@selector(handleKeyboardWillHide:)
name:UIKeyboardWillHideNotification
object:nil];
self.view.backgroundColor = [UIColor whiteColor];
self.myTextView = [[UITextView alloc] initWithFrame:self.view.bounds];
self.myTextView.text = @"Some text here...";
self.myTextView.font = [UIFont systemFontOfSize:16.0f];
[self.view addSubview:self.myTextView];
}
- (void) viewWillDisappear:(BOOL)paramAnimated{
[super viewWillDisappear:paramAnimated];
[[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

在这段代码中，在方法 viewWillAppear：开始监听键盘通知，在方法 viewWillDisappear 结束监听。其中背后的原因是假如在视图控件不在显示的时候，不移除作为监听键盘通知的视图控件，其他任何视图控件的部件发出的键盘通知将会导致通知中心也会发送这些通知到这个视图控制器。但是，当视图控件不显示在屏幕上时，我们没必要处理键盘通知，除非你有其他目的。我们已经看见过很多工程师不考虑这个简单的逻辑就直接

结束了他们的应用程序。

现在假如用户在视图空间里输入一些文本，键盘将会弹出并且我们将会取得键盘的高度及设置文本视图里的内容的底部边界空留值。这样就能使文本视图控件的内容更小，而且允许用户输入他们想要输入文本的量而不至于键盘挡住用户的视线。

2.15.4. 参考

XXX

2.16. 使用 UIButton 给用户界面添加按钮

2.16.1. 问题

想要给你的 UI 里添加一个按钮，并且给按钮设置一个触摸事件。

2.16.2. 方案

使用 UIButtonl 类。

2.16.3. 讨论

按钮是一些允许用户在应用程序里初始化一个动作的方法。例如，在设置程序里的 iCloud Setting 里显示了一个 Delete Account 按钮。假如你按下这个按钮，iCloud 程序将会产生一个动作。这个动作是与程序相关的，并不是用户按下任何程序里的 Delete 按钮都执行一样的动作。可以给按钮添加图片，待会我们将会看到。

图 2-52 一个叫 Delete Account 的按钮

一个按钮可以给不同的触发器注册动作。例如，当用户用手指按下一个按钮时按钮将发出一个动作，当手指离开按钮时又发出一个动作。这就变成了一个动作，执行这个动作的对象就成了目标对象。现在就开始在我们的视图控制器的头文件定义一个按钮吧，如下所示。

```
#import <UIKit/UIKit.h>
@interface Adding_Buttons_to_the_User_Interface_with_UIButtonViewController
: UIViewController
@property (nonatomic, strong) UIButton *myButton;
@end
```

UIButton 的默认高度是 37.0 像素。

接下来同时结合下面的属性：

```
#import "Adding_Buttons_to_the_User_Interface_with_UIButtonViewController.h"
@implementation
Adding_Buttons_to_the_User_Interface_with_UIButtonViewController
@synthesize myButton;
...
```

然后执行这个按钮：

```
- (void) buttonIsPressed:(UIButton *)paramSender{
NSLog(@"Button is pressed.");
}
- (void) buttonIsTapped:(UIButton *)paramSender{
NSLog(@"Button is tapped.");
}
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
self.myButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
self.myButton.frame = CGRectMake(110.0f,
200.0f,
100.0f,
37.0f);
[self.myButton setTitle:@"Press Me"
forState:UIControlStateNormal];
[self.myButton setTitle:@"I'm Pressed"
forState:UIControlStateHighlighted];
[self.myButton addTarget:self
action:@selector(buttonIsPressed:)
forControlEvents:UIControlEventTouchDown];
[self.myButton addTarget:self
action:@selector(buttonIsTapped:)
forControlEvents:UIControlEventTouchUpInside];
[self.view addSubview:self.myButton];
}
```

在这个示例代码中，我们使用 setTitle：forState：这个方法给按钮设置两个不同标题。这个标题是显示在按钮上的。一个按钮可以根据状态来显示标题，一个按钮有许多不同的状态，比如正常和高亮（按下时）。所以在这中情况下，当用户第一次看到按钮时，他会看到 Press Me。一旦他按下这个按钮，按钮的标题将会变成 I’m Pressed。

我们做了与按钮发出动作相似的事情。我们使用 addTarget：action：forControlEvents：这个方法为按钮指定两个动作。

当用户按下按钮一个动作将会产生。
当用户完成按下并使手指离开按钮时将会产生另一个动作。这就完成了一个 touch-up-inside 的动作.

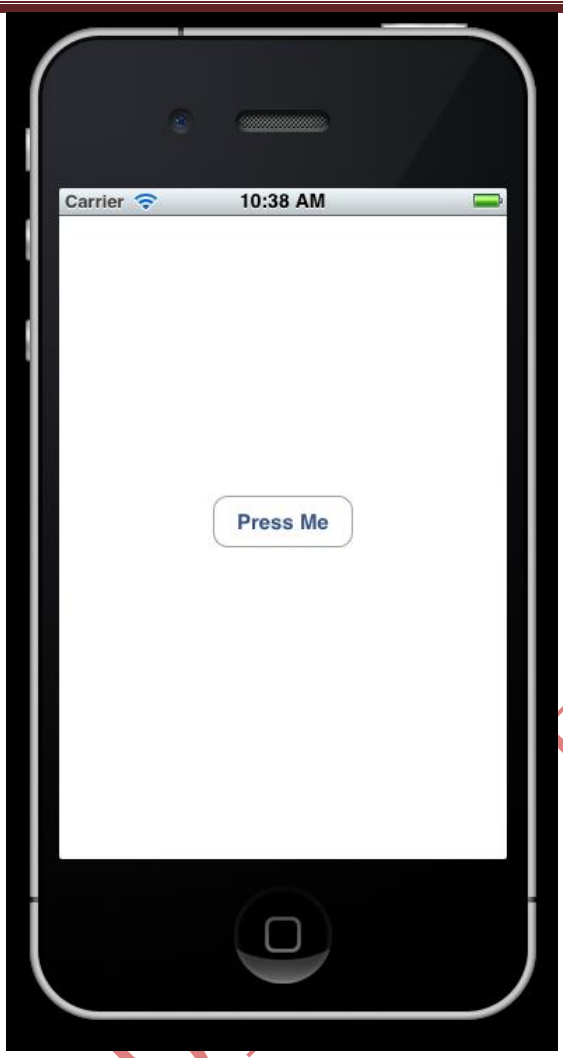


图 2-53 屏幕中间的一个圆边角矩形按钮

我们还要知道关于 UIButton 的另外一件事情，就是我们总是需要使用 buttonWithType 这个方法来初始化这它们，就像在示例代码中的那样。对于这个方法的参数，我们给它传递一个 UIButtonType 类型的值。如下。

```
 typedef enum {
UIButtonTypeCustom = 0,
UIButtonTypeRoundedRect,
UIButtonTypeDetailDisclosure,
UIButtonTypeInfoLight,
UIButtonTypeInfoDark,
UIButtonTypeContactAdd,
} UIButtonType;
```

一个按钮也可以提供一个图片。图片将会被改变按钮默认的外观。当你要给不同状态下按钮定义不同的图片时，要确保你的按钮是属于 UIBttonTypeCustom 类型的。这里准备了两张图片，一张是提供给正常按钮状态下的，一张是提供给高亮（按下）状态下的。现在创建一个自定义按钮并绑定这两张图片。一张是正常状态下的，一张是高亮状态下的。如下所示。

```
UIImage *normalImage = [UIImage imageNamed:@"NormalBlueButton.png"];
UIImage *highlightedImage = [UIImage imageNamed:@"HighlightedBlueButton"];
self.myButton = [UIButton buttonWithType:UIButtonTypeCustom];
self.myButton.frame = CGRectMake(110.0f,
200.0f,
100.0f,
```

```
37.0f);
[self.myButton setBackgroundImage:normalImage
forState:UIControlStateNormal];
[self.myButton setTitle:@"Normal"
forState:UIControlStateNormal];
[self.myButton setBackgroundImage:highlightedImage
forState:UIControlStateHighlighted];
[self.myButton setTitle:@"Pressed"
forState:UIControlStateHighlighted];
```

图 2-54 展示了这个程序在 iOS 模拟器下运行的情况。我们使用 setBackgroundImage：forState：这个方法设置背景图片。有了背景图片，我们依然可以使用 setTitle：forSate：这个方法在背景图上方显示文本。假如你的背景图片包含了文本，你就不必再给按钮设置标题了，你可以使用 setImage：forState：这个方法或者简单的将按钮上的标题删除。

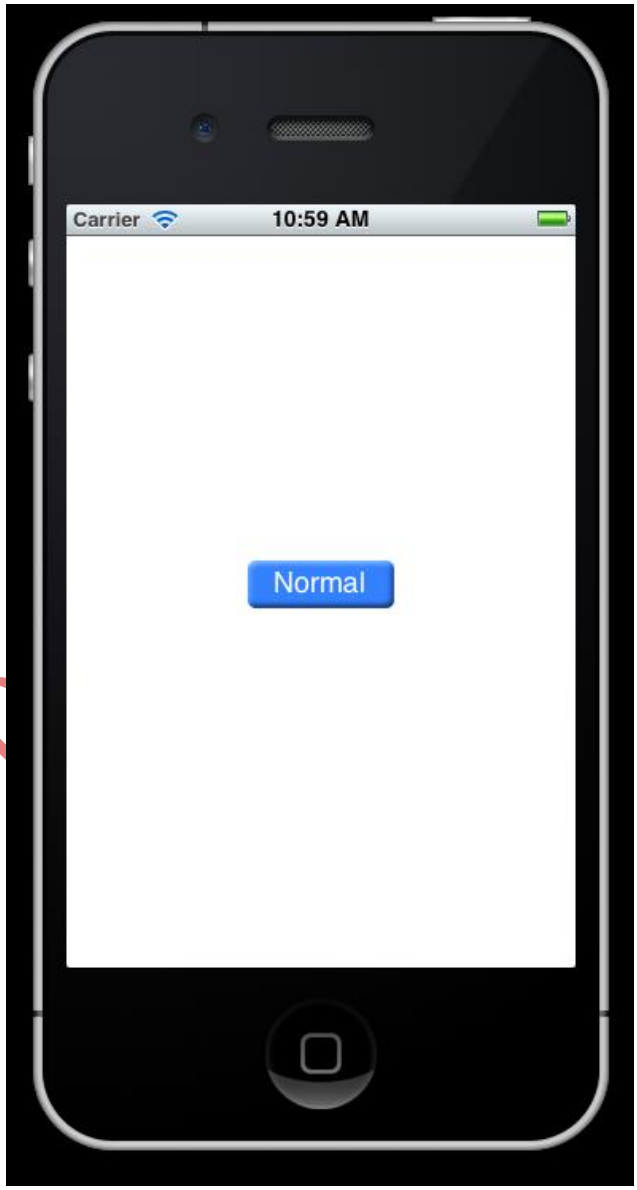


图 2-54 一个拥有背景图片的按钮

2.16.4. 参考

XXX

2.17. 使用 UIImageView 显示图片

2.17.1. 问题

想要在程序用户界面给用户显示图片。

2.17.2. 方案

使用 UIImageView 类。

2.17.3. 讨论

UIImageView 是 iOS SDK 中最复杂的类里的其中的一个类。请想想看，事实上没有一个类是复杂的，它们都是有据可查的。想必你也知道，图片视图是负责显示图片的。这里没有任何的提示或技巧在里面。你所要做的只是实例化一个 UIImageView 类型的对象并将它添加到视图里。就这么简单。现在这里有一张 Apple macBook 的图片，要将它显示在图片视图上。现在就开始在我们的视图控制器的头文件里编写代码吧，如下所示。

```
#import <UIKit/UIKit.h>
@interface Displaying_Images_with_UIImageViewViewController
: UIViewController
@property (nonatomic, strong) UIImageView *myImageView;
@end
```

然后在执行文件里结合这个属性：

```
#import "Displaying_Images_with_UIImageViewViewController.h"
@implementation Displaying_Images_with_UIImageViewViewController
@synthesize myImageView;
...
```

继续，实例化这个图片并将图片放置到这里：

```
- (void)viewDidLoad{
[super viewDidLoad];
UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir.png"];
self.myImageView = [[UIImageView alloc] initWithImage:macBookAir];
self.myImageView.center = self.view.center;
[self.view addSubview:self.myImageView];
}
```

现在如果运行这个程序，我们将会看来如下所示的结果：

图 2-55 一张图片因为太大而不适合屏幕

这里需要提醒一下，你看到的这张 MacBook Air 图片的规格是 980*519 像素的，当然不适合 iPhone 的屏幕。给怎么解决这个问题呢？首先，我们需要确保我们初始化图片用到的方法是 initWithFrame：而不是 initWithImage：，因为后者会以图片的实际宽和高显示在屏幕上。现在先修正一下，如下所示。

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir.png"];
self.myImageView = [[UIImageView alloc] initWithImage:macBookAir];
self.myImageView.center = self.view.center;
[self.view addSubview:self.myImageView];
}
```

现在程序运行的情况会如何呢？如下图所示。

图 2-56 图片的宽被压缩以至于能适合屏幕的宽

其实这并不是我们真正想要的结果，不是吗？当然我们我们正取得了这个图片的视图架，但是这个图片视图提供图片的方式不是很正确，那怎么办呢？我们可以通过设置图片视图的 contentMode 属性来改善这个问题。这个属性属于 UIContentMode 类型。如下所示。

```
typedef enum {
UIViewContentModeScaleToFill,
UIViewContentModeScaleAspectFit,
UIViewContentModeScaleAspectFill,
UIViewContentModeRedraw,
UIViewContentModeCenter,
UIViewContentModeTop,
UIViewContentModeBottom,
UIViewContentModeLeft,
UIViewContentModeRight,
UIViewContentModeTopLeft,
UIViewContentModeTopRight,
UIViewContentModeBottomLeft,
UIViewContentModeBottomRight,
} UIViewContentMode;
```

这里有 UIViewContentMode 枚举类列出来的一些最有用的值的解释，如下。

UIViewContentModeScaleToFill

这个值会将图片视图里的图片将图片视图填满。

UIViewContentModeScaleAspectFit
这个值会确保图片视图里的图片有正确的长宽比，并且会确保图片适应图片视图的边界。

UIViewContentModeScaleAspectFill
这个值会确保说图片视图里的图片有正确的长宽比，并且使图片充满整个图片视图的边界。为了能使这个值正常工作，，确保将 clipsToBounds 这个属性值设置为 YES。

所以确保图片能适应图片视图的边界并使具有合适的长宽比，我们需要使用 UIViewContentModeScaleAspectFit 内容模式：

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir.png"];
self.myImageView = [[UIImageView alloc] initWithFrame:self.view.bounds];
self.myImageView.image = macBookAir;
self.myImageView.center = self.view.center;
[self.view addSubview:self.myImageView];
}
```

图 2-57 图片视图的长宽比是绝对却到好处

2.17.4. 参考

XXX

2.18. 使用 UIScrollView 创建能滑动的内容。

2.18.1. 问题

有个内容需要显示在屏幕上，但是设备上的屏幕尺寸提供不了内容的实际所需的面积。

2.18.2. 方案

使用 UIScrollView 类

2.18.3. 讨论

滑动视图是使 iOS 操作系统更完美的其中一个特征。到处都有他们的身影。比如 Clock 和 Contacts 程序都使用了滑动视图。其中的文本内容可以上下滑动。这就是神奇的滑动视图。

我们真正要学习知识点是滑动视图里面内容的尺寸。其尺寸大小属于 CGSize 类型值，它确定了滑动视图内容的宽和高。由它的名字可以推断出，滑动视图是 UIView 的子类，所以可以使用 addSubview:这个方法给视图添加滑动视图。但是，必须确保有一个合适的滑动视图内容尺寸，否则里面的内容将不会滑动。

这里举个例子，找一个张图片并加载到图片视图里。这里就用图 2.17 里的 MacBook Air 这张图片。将它添加到一个图片视图里，并将这个图片视图放到一个滑动视图里。然后使用滑动视图的 contentSize 属性将滑动视图的内容尺寸设置成和
图片尺寸一样大小（宽和高都一样）。首先从视图控制器的头文件开始：

```
#import <UIKit/UIKit.h>
@interface Creating_Scrollable_Content_with_UIScrollViewViewController
: UIViewController
@property (nonatomic, strong) UIImageView *myImageView;
@property (nonatomic, strong) UIScrollView *myScrollView;
@end
//结合下面的属性：
#import "Creating_Scrollable_Content_with_UIScrollViewViewController.h"
@implementation Creating_Scrollable_Content_with_UIScrollViewViewController
@synthesize myImageView;
@synthesize myScrollView;
...
//将图片放置到滑动视图里
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
UIImage *imageToLoad = [UIImage imageNamed:@"MacBookAir.png"];
self.myImageView = [[UIImageView alloc] initWithImage:imageToLoad];
self.myScrollView = [[UIScrollView alloc] initWithFrame:self.view.bounds];
[self.myScrollView addSubview:self.myImageView];
self.myScrollView.contentSize = self.myImageView.bounds.size;
```

```
[self.view addSubview:self.myScrollView];
}
```

假如现在在 iOS 模拟器上运行这个程序，将会发现可以在水平和垂直方向滑动这张图片。因为图片的宽和高都超出了滑动视图的边界。但是如果图像的大小是 20*20 像素的话，这个滑动视图基本上没有什么作用。事实上，将 20*20 像素大小的图片放在滑动视图里是不合适的，因为图片的尺寸小于滑动视图的尺寸滑动视图将不起作用，滑动视图就多余了。

UIScrollView 有一个很有用的功能就是它支持委托。滑动视图能发送很重要的事件给他们的委托对象。滑动视图的任何一个委托都必须遵守 UIScrollViewDelegate 的协议。下面是一些定义在这个协议中的方法解释：

scrollViewDidScroll：
当滑动视图里的内容滑动时将调用这个方法。

scrollViewWillDeginDeceleratin:
当用户滑动滑动视图里的内容并且手指离开屏幕而且内容还在滑动的时候，将会调用这个方法。

scrollViewDidEndDecelerating:
当滑动视图里的内容结束滑动时将调用这个方法。

scrollViewDidEngDragging：willDecelerating:
当用户完成拖拽滑动视图里的内容时将会调用这个方法。这个方法跟 scrollViewDidEndDecelerating:方法非常相似，但是需要注意的是，用户可以不通过滑动内容而是直接拖拽内容。你可以将手指放到内容上，然后将手指滑动到屏幕的任何位置，当手指离开屏幕时，内容不会有滑动的惯性（停止滑动）。这种拖拽方式是相对于滑动的而言的。滑动和拖拽很相似，但是滑动是当用户的手指离开屏幕时会给滑动视图的内容一个惯性使它继续滑动，而不会等到手指离开屏幕之前就停止滑动了。这就像带档开车（拖拽）和空挡溜车（滑动）。

给我们的程序添加一些有意思的东西吧。现在的目标是：在用户开始滚动滑动视图的时候，将图片视图内图片的 ALPHA 等级设置为 0.50F（半透明），在用户完成滚动时，重设为 1.0F（不透明）。先以遵守 UIScrollViewDelegate 协议开始：

```
#import <UIKit/UIKit.h>
@interface Creating_Scrollable_Content_with_UIScrollViewViewController
: UIViewController <UIScrollViewDelegate>
@property (nonatomic, strong) UIImageView *myImageView;
@property (nonatomic, strong) UIScrollView *myScrollView;
@end
```

然后执行下面的功能：

```
- (void)scrollViewDidScroll:(UIScrollView *)scrollView{
/* Gets called when user scrolls or drags */
self.myScrollView.alpha = 0.50f;
}
- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView{
/* Gets called only after scrolling */
self.myScrollView.alpha = 1.0f;
}
- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView
willDecelerate:(BOOL)decelerate{
/* Make sure the alpha is reset even if the user is dragging */
self.myScrollView.alpha = 1.0f;
}
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
UIImage *imageToLoad = [UIImage imageNamed:@"MacBookAir.png"];
```

```
self.myImageView = [[UIImageView alloc] initWithImage:imageToLoad];
self.myScrollView = [[UIScrollView alloc] initWithFrame:self.view.bounds];
[self.myScrollView addSubview:self.myImageView];
self.myScrollView.contentSize = self.myImageView.bounds.size;
self.myScrollView.delegate = self;
[self.view addSubview:self.myScrollView];
}
```

也许你已经注意到了，滑动框有很多指示器。其中有一个指标是当滑动视图里的内容滑动时出现在滑动视图旁边的小追踪线。下面举个例子，如下图所示：

图 2-58 出现在滑动视图右下角的黑色指示器

指示器简单地显示了当前视图在内容上的大体位置。他跟浏览器访问互联网时的滑动条相似。可以改变 indicatorStyle 属性值控制指示器的外观。例如，下面的代码将滑动视图的指示器外观风格变成了白色：

```
self.myScrollView.indicatorStyle = UIScrollViewIndicatorStyleWhite;
```

滑动视图还有一个很有用的功能是它可以分页显示内容。分页和滑动一样，但是当转到下一页时将滑动功能锁定了。假如你使用过 iPhone 或 iPad 的图片应用程序，也许你就已经知道大概了。

当你查看图片时，你可以在图片之间来回刷。当一张图片显示在屏幕上时，刷它一下将会显示下一张或者上一张图片。你刷它一下并没有一直滑动到结束或者滑动到开始的位置。当滑动发生时，滑动视图将会查找下一张图片并将它显示在屏幕上，还会有滑动和反弹的的效果，然后停止滑动动画。这就是分页。假如你还没有尝试过，强烈建议你先体验一下，因为只有你用过图片浏览器之后，接下来的学习才有意义。

为了写一个示例代码，这里准备了三张图片：一张 iPhone 的图片，一张 iPad 的图片和一张 MacBook Air 的图片。并且已经将他们放置到了它们独立的图片视图里，然后将这些图片视图添加到滑动视图里。然后我们可以将滑动视图的 pagingEnabled 属性值设置成 Yes 来禁用分页：

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
UIImage *iPhone = [UIImage imageNamed:@"iPhone.png"];
UIImage *iPad = [UIImage imageNamed:@"iPad.png"];
UIImage *macBookAir = [UIImage imageNamed:@"MacBookAir.png"];
CGRect scrollViewRect = self.view.bounds;
self.myScrollView = [[UIScrollView alloc] initWithFrame:scrollViewRect];
self.myScrollView.pagingEnabled = YES;
self.myScrollView.contentSize = CGSizeMake(scrollViewRect.size.width * 3.0f,
scrollViewRect.size.height);
[self.view addSubview:self.myScrollView];
CGRect imageViewRect = self.view.bounds;
UIImageView *iPhoneImageView = [self newImageViewWithImage:iPhone
frame:imageViewRect];
[self.myScrollView addSubview:iPhoneImageView];
/* Go to next page by moving the x position of the next image view */
imageViewRect.origin.x += imageViewRect.size.width;
UIImageView *iPadImageView = [self newImageViewWithImage:iPad
frame:imageViewRect];
[self.myScrollView addSubview:iPadImageView];
/* Go to next page by moving the x position of the next image view */
imageViewRect.origin.x += imageViewRect.size.width;
UIImageView *macBookAirImageView =
[self newImageViewWithImage:macBookAir
frame:imageViewRect];
[self.myScrollView addSubview:macBookAirImageView];
}
```

现在可滑动内容里有三张图片了：

图 2-59 通过可滑动页面视图的页面来滑动图片

2.18.4. 参考

xxx

2.19. 使用 UIWebView 加载 Web 页面

2.19.1. 问题

想要在 iOS 程序里正确地动态加载 web 页面。

2.19.2. 方案

使用 UIWebView 类。

2.19.3. 讨论

网页视图是 iOS 浏览器加载网页内容的部件。你可以通过使用 UIWebView 类行使 iOS 上 Safari 的所有权限。你所需要做的就是将网页视图放置在 UI 上并使用其中的一个加载方法：

loadData：MIMEType：textEncodingName：baseURL：
加载一个 NSData 的实例到网页视图上。

loadHTMLString：baseURL:
这个方法是加载 NSString 的一个实例到页面视图上。这个实例必须是一个有效的 HTML，或者说是那些网页能提供的东西。

loadRequest：
加载一个 NSURLRequest 的实例。当你想要在应用程序的网页视图里加载远程的 URL 时，这个方法是很有用的。

下面看一个例子。先从视图控制器的头文件开始时：

```
#import <UIKit/UIKit.h>
@interface Loading_Web_Pages_with_UIWebViewViewController
: UIViewController
@property (nonatomic, strong) UIWebView *myWebView;
@end
```

很显然，还需要结合下面的属性：

```
#import "Loading_Web_Pages_with_UIWebViewViewController.h"
@implementation Loading_Web_Pages_with_UIWebViewViewController
@synthesize myWebView;
...
```

然后我想要将字符 ios 5 rogramming CookBookP 加载到网页上。为了证明这些工作达到了预期的效果并且网页视图能提供丰富的文本，我会将 Programming 部分设置成粗体字，其余部分不变：

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
self.myWebView = [[UIWebView alloc] initWithFrame:self.view.bounds];
[self.view addSubview:self.myWebView];
NSString *htmlString = @"iOS 5 Programming <strong>Cookbook</strong>";
[self.myWebView loadHTMLString:htmlString
baseURL:nil];
}
```

图 2-60 加载丰富的文本到网页视图里

另一个方法是将远程 URL 加载到网页视图里。为此，我们将会用到 loadRequest：这个方法。现在继续看一个例子，我们将会在 iOS 程序里的网页视图里加载 Apple 的主页面：

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
self.myWebView = [[UIWebView alloc] initWithFrame:self.view.bounds];
self.myWebView.scalesPageToFit = YES;
[self.view addSubview:self.myWebView];
NSURL *url = [NSURL URLWithString:@"http://www.apple.com"];
NSURLRequest *request = [NSURLRequest requestWithURL:url];
[self.myWebView loadRequest:request];
}
```

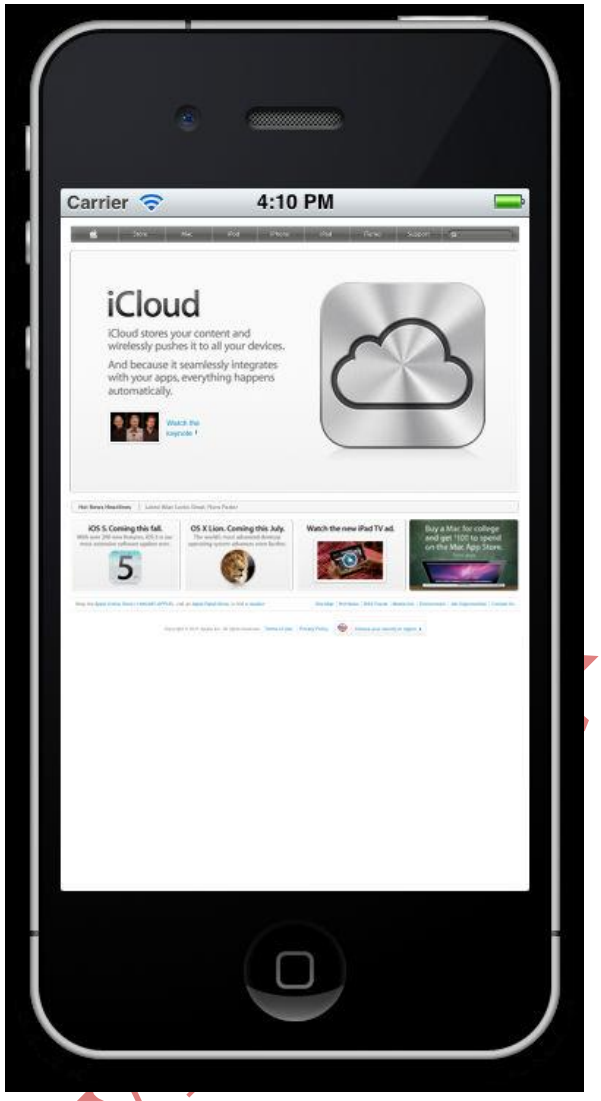


图 2-61 Apple 的首页加载到了一个网页视图里

将传送的内容加载到网页视图里需要花一定的时间。你可能已经发现，Safari 加载内容时在屏幕的左上角会出现一个活动的指示器，它会告诉你设备实际正在加载内容。这里有个例子：

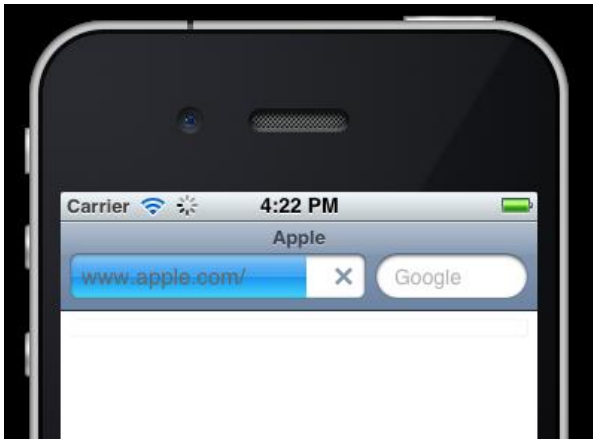


图 2-62 一个进度条指示着加载的进度

这种 iOS 的方式是通过委托来实现的。我们会订阅一个网页视图的委托，网页视图会告诉我们何时开始加载内容。当内容全部加载完时，我们将会得到网页视图关于这个的通知信息。我们通过网页视图的 delegate 属性完成这项工作。一个网页视图的委托必须遵守 UIWebViewDelegate 的协议。接下来继续并执行在视图控制器上的小活动指示器。请注意，这个活动指示器已经是程序的一部分了，我们必须先创建它。我们可以使用 UIApplication 的 setNetWorkActivityIndicatorVisible：这个方法来控制它。因此先从视图控制器头文件开始：

```
#import <UIKit/UIKit.h>
@interface Loading_Web_Pages_with_UIWebViewViewController
: UIViewController <UIWebViewDelegate>
@property (nonatomic, strong) UIWebView *myWebView;
@end
```

接下来开始执行，下面是声明在 UIWebViewDelegate 协议的三个方法：

webViewDidStartLoad:
　　当网页视图开始加载内容时将调用这个方法。
webViewDidFinishLoad:
　　当网页视图完成加载时将调用这个方法。
webView:didFailLoadWithError:
　　当因加载出错（例如：因网络问题而断开可连接）而导致停止加载时将调用这方法。

```
- (void)webViewDidStartLoad:(UIWebView *)webView{
[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:YES];
}
- (void)webViewDidFinishLoad:(UIWebView *)webView{
[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:NO];
}
- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error{
[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:NO];
}
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
self.myWebView = [[UIWebView alloc] initWithFrame:self.view.bounds];
self.myWebView.delegate = self;
self.myWebView.scalesPageToFit = YES;
[self.view addSubview:self.myWebView];
NSURL *url = [NSURL URLWithString:@"http://www.apple.com"];
NSURLRequest *request = [NSURLRequest requestWithURL:url];
[self.myWebView loadRequest:request];
}
```

2.19.4. 参考

XXX

2.20. 使用 UISplitViewController 显示 Master-Detail 视图

2.20.1. 问题

想要通过在一个视图控制器里显示两部分视图来充分体现 iPad 大屏幕优势。

2.20.2. 方案

使用 UISplitViewController 类。

2.20.3. 讨论

主从分离视图控制器只出现在 iPad 里。假如你使用过 iPad，很有可能已经看见它们了。只要在水平模式下打开设置程序就可以看到了。

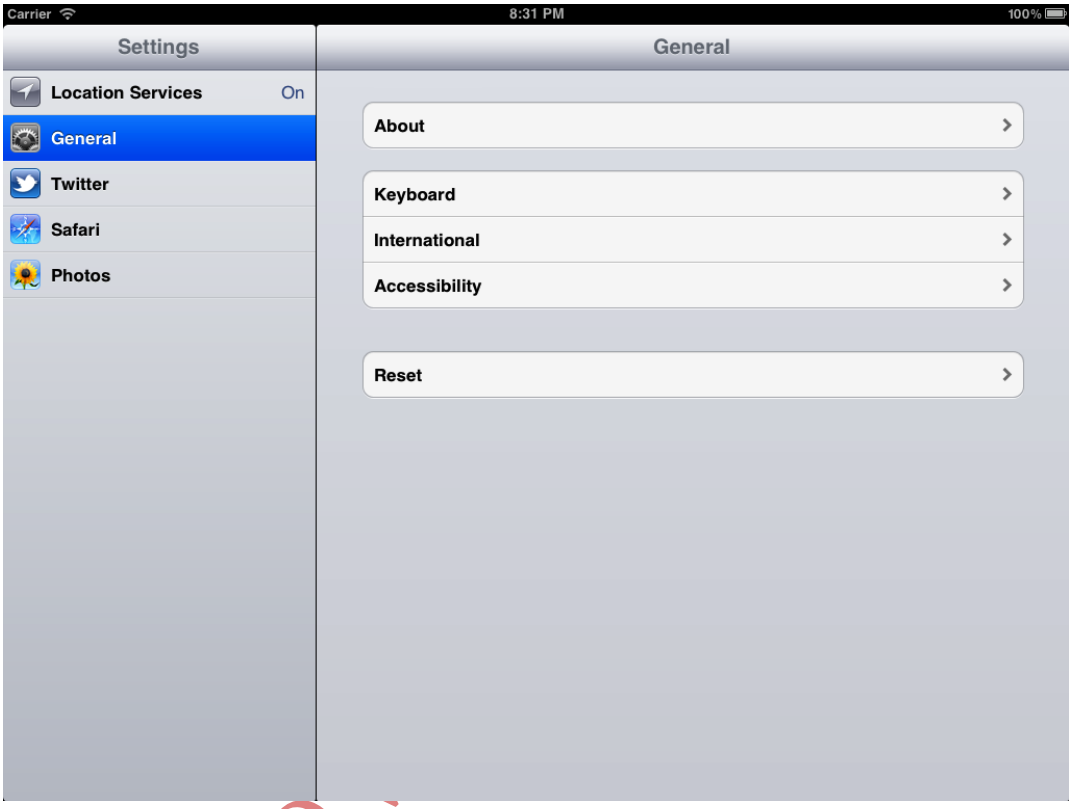


图 2-63 iPad 里设置应用里的主从分离视图控制器

在图 2-63 显示了主从分离视图控制器的样子。主从分离视图控制器分为左边和右边两个视图。左为主设置视图，点击其中一项将在右从视图显示具体设置项。

不要在设备上（除 iPad 外）尝试着实例化一个 UISplitViewController 类型的对象，否者将产生一个异常。

Apple 已经在应用程序上创建主从视图简单化了很多。只要按照下面的几步就可以在应用程序上创建主从视图：

在 Xcode 里，在 File 菜单里选择 New→New Project...创建一个新工程。

在 New Project 界面里，在左边选择 iOS 的 Application，然后选择 Master-detail Application（如图 2-64 所示），然后点击下一步。

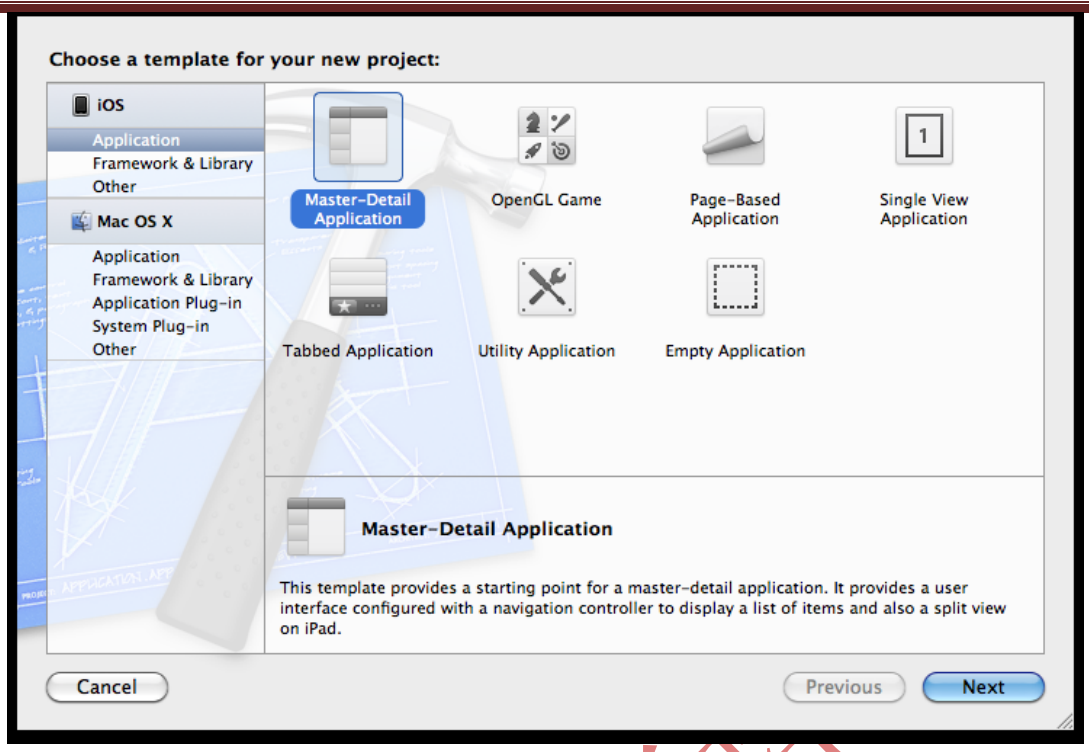


图 2-64 在 Xcode 里选择 Master-Detail AppApplication 工程模版

在这个界面里，选择你的产品名称并确保你的 Device 是 Universal。我们想要确保程序在 iPhone 和 iPad 上都能运行。完成后，按下一步。（见图 2-65）

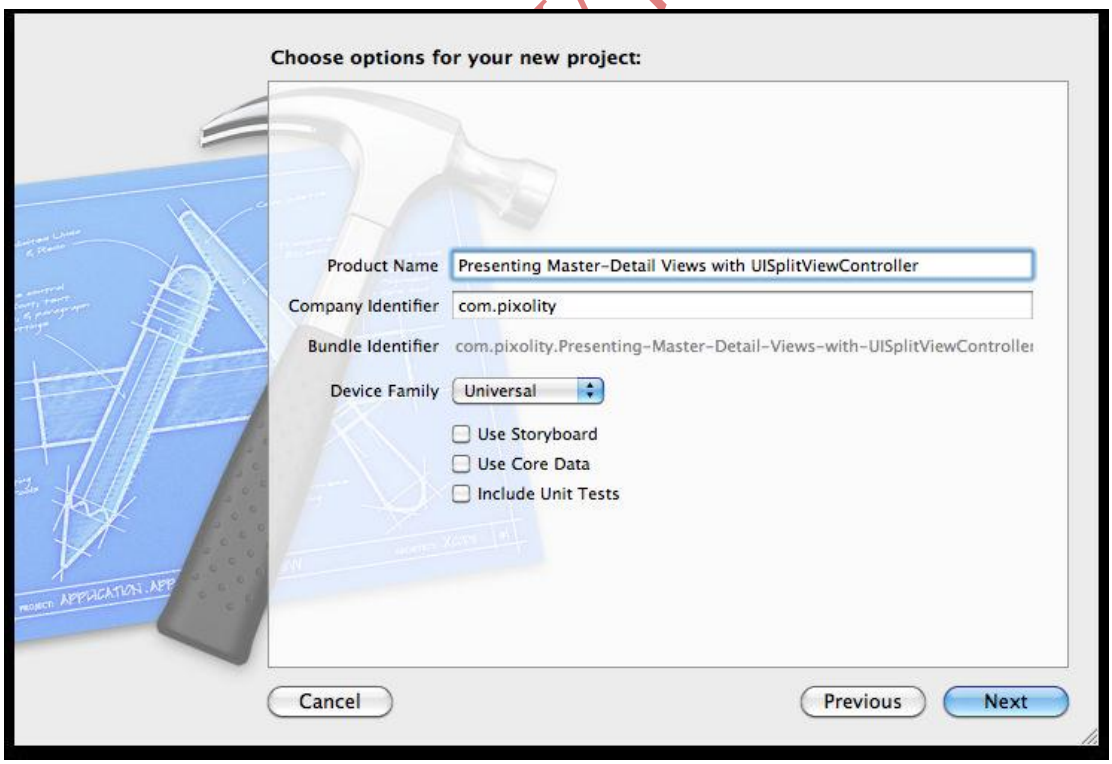


图 2-65 在 Xcode 里设置 master-detail 工程

现在选择你要保存工程的位置。完成后按下 Create 按钮（见图 2-66）

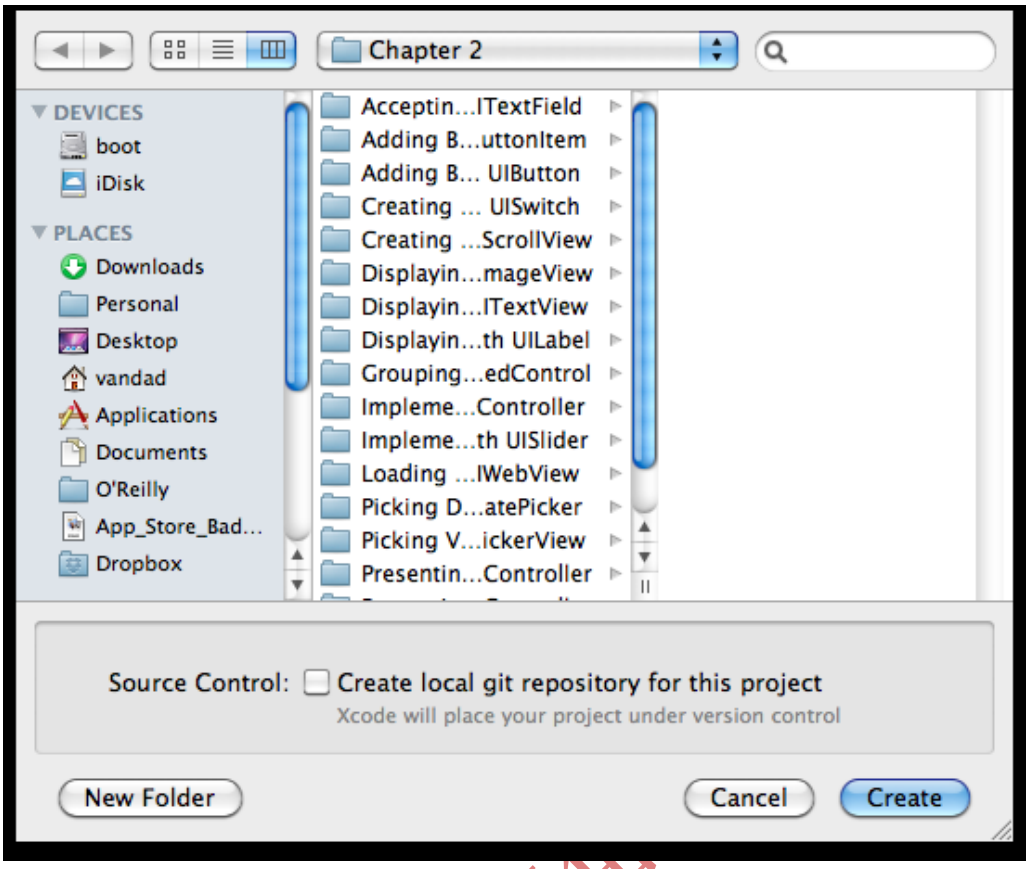


图 2-66 在硬盘里保存新建的住从视图程序

现在这个工程就建好了。在 Xcode 的左上角的计划痕迹导航按钮里，确保你的程序是在 iPad 模拟器上运行而不是 iPhone 模拟器上。假如你在 Xcode 里创建了一个普通的主从视图，Xcode 会确保这个程序也能在 iPhone 中运行。但是在 iPhone 上运行时，在程序的结构上会有一个导航控制器和一个视图控制器在里面，而运行在 iPad 上将会使用视图拆分控制器来包含两个视图在里面。

在主从视图控制器工程模版里有两个很重要的文件需要注意：

RootViewController

它是显示在 iPad 主从视图左边的主视图控制器，这是用户首先看到的视图控制器。

DetailViewController

它是显示在 iPad 主从视图右边的从视图控制器。在 iPhone 里，这是当用户点击主视图任何项目是将会出现的视图控制器。

主视图和从视图之间的信息交流是我们需要学习的知识。我们是要通过程序委托的方式去交流还是让主视图直接发送信息给从视图呢？这是取决于我们的。

假如你在 iPad 模拟器上运行这个程序，你会发现在水平模式下会看到主从视图的主视图和从视图。但是一旦我们将设备方向旋转为纵向，主视图将会消失，而是在从视图的左上角的一个主导航按钮所代替。虽然这样很不错，但是对比一下 iPad 的设置程序，我们将会排斥这种方式。假如你旋转 iPad 的方向为纵向，你还会看到主视图和从视图控制器。我们该怎么做呢？结果发现 Apple 提供了一个 API 给我们，通过它我们可以完成这个工作。只要在 DetailViewController.m 文件里执行这个方法即可：

```
- (BOOL) splitViewController:(UISplitViewController *)svc
shouldHideViewController:(UIViewController *)vc
inOrientation:(UIInterfaceOrientation)orientation{
```

```
return NO;
}
```

这个方法只在 iOS SDK 5.0 可用。

假如这个方法返回 NO，iOS 将不会在另一个方向上显示主视图，所以这是一个释放。既然已经执行了这个方法，我们将会在需要这两个方法：

```
- (void)splitViewController:(UISplitViewController *)svc
willHideViewController:(UIViewController *)aViewController
withBarButtonItem:(UIBarButtonItem *)barButtonItem
forPopoverController: (UIPopoverController *)pc{
barButtonItem.title = @"Master";
NSMutableArray *items = [[self.toolbar items] mutableCopy];
[items insertObject:barButtonItem atIndex:0];
[self.toolbar setItems:items animated:YES];
self.popoverController = pc;
}
- (void)splitViewController:(UISplitViewController *)svc
willShowViewController:(UIViewController *)aViewController
invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem{
NSMutableArray *items = [[self.toolbar items] mutableCopy];
[items removeObjectAtIndex:0];
[self.toolbar setItems:items animated:YES];
self.popoverController = nil;
}
```

这两个方法仅仅管理导航条按钮，但是现在我们将不会再用到它，我们可以摆脱这两个方法了。所以可以将它们注释掉或者从 DetailViewController.m 文件里删除。

同时，假如主视图不包含从视图时，主视图通过发送信息和从视图交流将不现实。我们将要改善它。这个程序的委托是这两个视图控件创建的对象，所以我们将程序的委托参照放在从视图控制器里并在主视图控制器里读取它。我们先给程序委托添加一个叫 detailViewController 的属性;

```
#import <UIKit/UIKit.h>
@class DetailViewController;
@interface Presenting_Master_Detail_Views_with_UISplitViewControllerAppDelegate
: UIResponder <UIApplicationDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong) UINavigationController *navigationController;
@property (nonatomic, strong) UISplitViewController *splitViewController;
@property (nonatomic, readonly, strong)
DetailViewController *detailViewController;
@end
```

现在在程序委托执行文件里，我们需要同时配合下面的新属性：

```
@synthesize detailViewController;
```

在程序的委托执行文件里。我们可以看到 application：didFinishLaunchingWithOption：的选择方法。找到

这么一行的代码方法：

```
chingWithOptions: selector. Find this line of code in that method:
DetailViewController *detailViewController =
[[DetailViewController alloc] initWithNibName:@"DetailViewController_iPad"
bundle:nil];
```

这里，程序将会一个 DetailViewController 类型的一个对象并将它放置在主从控制器里。但是，你可以看到，在代码里的 detailViewController 是一个本地变量。并且它会将同名的属性打上阴影，所以我们需要移除本地变量声明，可以这样修改代码：

```
detailViewController =
[[DetailViewController alloc] initWithNibName:@"DetailViewController_iPad"
bundle:nil];
```

一切正常，现在在主视图控制器里找到这样的一个方法：

```
- (void) tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
if ([[UIDevice currentDevice] userInterfaceIdiom]
== UIUserInterfaceIdiomPhone) {
DetailViewController *detailViewController =
[[DetailViewController alloc]
initWithNibName:@"DetailViewController_iPhone"
bundle:nil];
[self.navigationController pushViewController:detailViewController
animated:YES];
} else {
/* iPad */
}
}
```

任何时候用户只要点击主视图控制器上项目都将调用这个方法。你可以看到，假如是 iPad 这个逻辑关系是空白的，所以在这里，修改程序委托的参照到从视图控制器里：

```
- (void) tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
if ([[UIDevice currentDevice] userInterfaceIdiom]
== UIUserInterfaceIdiomPhone) {
DetailViewController *detailViewController =
[[DetailViewController alloc]
initWithNibName:@"DetailViewController_iPhone"
bundle:nil];
[self.navigationController pushViewController:detailViewController
animated:YES];
}
}
```

很好，现在我们有主从视图的一个参照了。使用这个参照，你可以发送信息到从视图上并可以根据用户在主视图控制器上的选择执行各种各样的任务。

2.20.4. 参考

XXX

2.21. 使用 UIPageViewController 启用分页

2.21.1. 问题

想要创建一个像 iBOOk 这样的程序，用户可以像现实中的书那样翻动书中页面，提供一个直观和真实的用户体验。

2.21.2. 方案

使用 UIPageVIewController 类。

2.21.3. 讨论

Xcode 提供了页面视图控制器的一个模版。在我们读它们的具体解释之前最好先看看它是怎样的外观。所以现在按照下面的步骤来创建一个具有页面视图控制器的程序：

页面视图控制器在 iPhone 和 iPad 上都能正常工作。

在 Xcode 里，选择 File 菜单然后选择 Mew→New Project…

在新工程窗口左边，确保你选择的是 iOS 的程序。完成后，在右边选择 Page-Based Application 模版并按下一步。如图 2-67 所示。

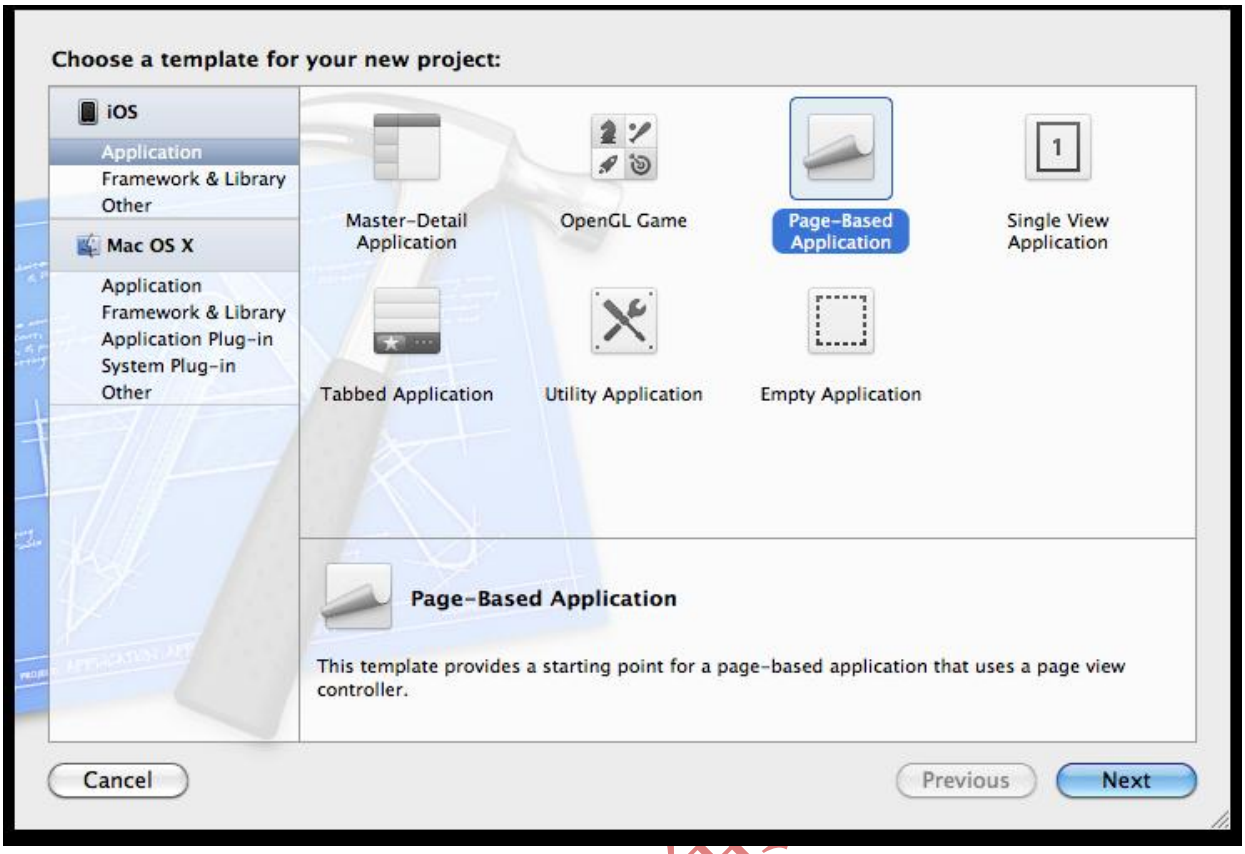


图 2-67 在 Xcode 里创建一个 Page-Based Application

现在选择产品名字并确保 Device Family 选择的是 Universal，因为我们要在 iPhone 和 iPad 上都能运行这个程序（见图 2-68）。完成后按下一步。

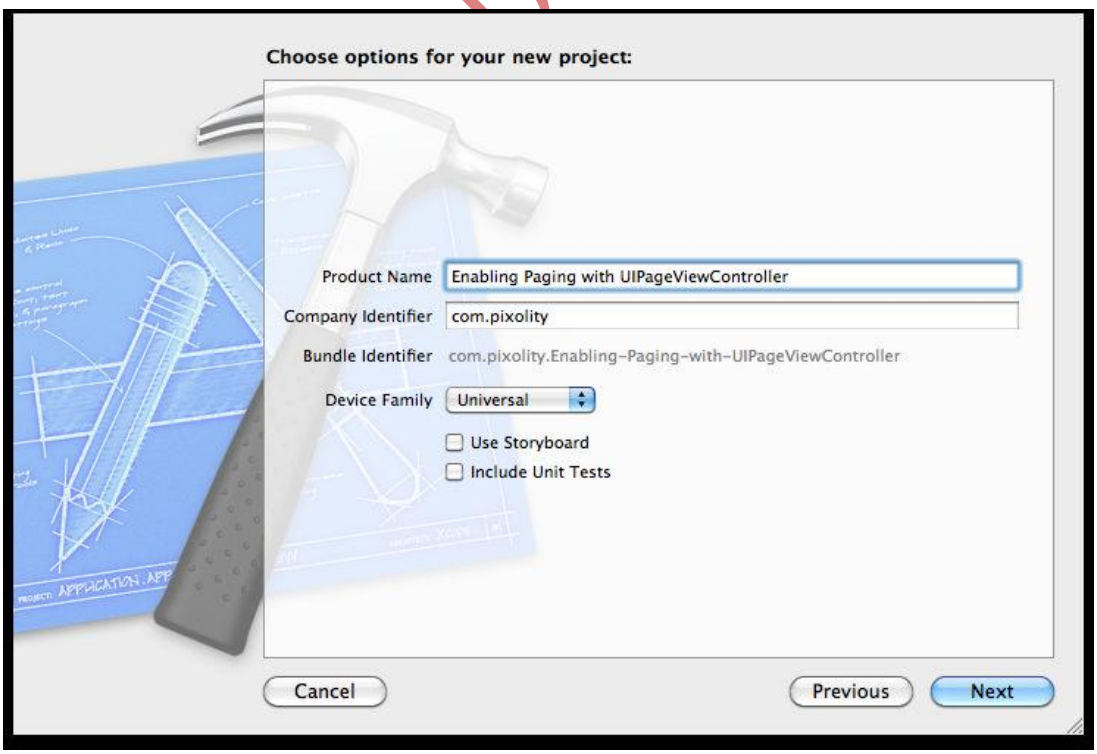


图 2-68 设置 page-based app 的工程配置

选择你要保存程序的位置。完成后，按下“Creat”按钮，现在就完成了工程的创建。

现在你可以在 Xcode 里看到工程里已经创建好的一些类。现在看看这些类的解释：

Delegate 类

这个程序委托简单地创建了一个 RootViewController 类的一个实例并展现给用户。这里的.xib 是供 iPad 使用的另一个是给 iPhone 用的。但是两个都是使用之前提到的类；只要有个映像就行。

RootViewController 类

创建 UIPageViewController 的一个实例并给这个实例添加一个视图控制器，因此这视图控制器的 UI 实际上是两个视图控制器的混合体，其中一个是它本身另一个是页面视图控制器。这个视图控制器同时也将自己设置成为页面视图控制器的委托，创建 ModelController 类的一个实例，并将它设置成页面视图控制器的数据资源。

DataViewController 类

在页面视图控制器的每个页面里都会提供这个类的一个实例给用户。这个类是 UIViewController 的一个子类。

ModelController 类

这仅仅是一个遵守 UIPageViewCintrollerDataSource 协议的一个 NSObject 的子类。这个类是页面视图控制器的数据资源类。

你可以看到一个页面是视图控制器具有一个委托和一个数据资源和 Xcode 默认的基于页面的程序模版，这个根视图控制器变成了页面视图控制器的委托，模式控制器变成了数据资源。为了理解一个页面视图控制器是怎么工作的，我们需要去理解这个委托和数据资源协议。现在先理解委托 UIPageViewControllerDelegate 这个协议。这个协议有两个重要的方法，如下：

```
- (void)pageViewController:(UIPageViewController *)pageViewController
didFinishAnimating:(BOOL)finished
previousViewControllers:(NSArray *)previousViewControllers
transitionCompleted:(BOOL)completed;
- (UIPageViewControllerSpineLocation)
pageViewController:(UIPageViewController *)pageViewController
spineLocationForInterfaceOrientation:(UIInterfaceOrientation)orientation;
```

当用户从一个页面转向下一个或者前一个页面，或者当用户开始从一个页面转向另一个页面的途中后悔了，并撤销返回到了之前的页面时，将会调用这个方法。假如成功跳转到另一个页面时，transitionCompleted 会被置成 YES，假如在跳转途中取消了跳转这个动作将会被置成 NO。

当设备的方向改变了将会调用第二个方法。你可以使用第二个方通过返回 UIPageViewControllerSpinelocation 类型的一个值来设定页面主键的位置：

```
enum {
UIPageViewControllerSpineLocationNone = 0,
UIPageViewControllerSpineLocationMin = 1,
UIPageViewControllerSpineLocationMid = 2,
UIPageViewControllerSpineLocationMax = 3
};
typedef NSInteger UIPageViewControllerSpineLocation;
```

这个可能对于你来说有点复杂，让我来给你演示一下。假如我们使用一个 UIpageViewControllerSpineLocationMin 值，页面视图将只会提供一个视图给用户，当转向另一个新页面时将提供一个新页面给用户。但是，当我们给 UIPageViewControllerSpineLocationMid 设置书脊时，我们需要同时提供两个视图。一个在左边一个在右边，中间放置一个书脊。在图 2-69 你可以看到一个页面视图控制器在水平模式下的例子，并设置了 UIpageViewControllerSpineLocationMin 的书脊：

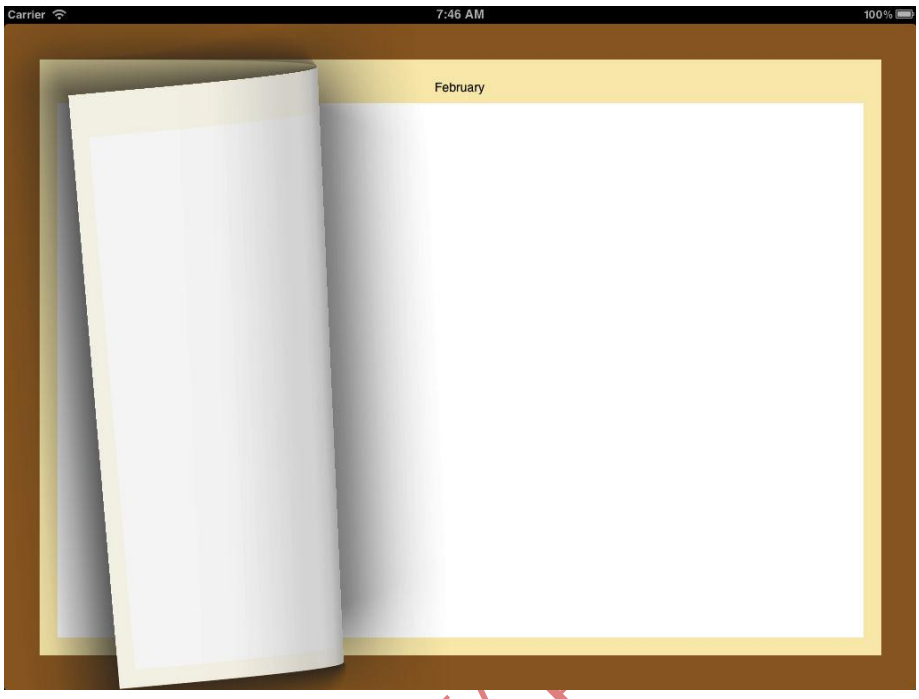


图 2-69 在水平模式下的页面视图控制器里的一个视图控制器

现在假如返回 UIPageViewControllerSpinelocationMid 的书脊，我们得到如下所示的结果：

图 2-70 水平模式下一个页面视图控制器里显示两个视图控制器

正如图 2-70 所示那样，书脊被放置在两个视图控制器的中间。一旦用户将页面从右边翻向左边，这个页面就停在了左边，而右边则显示一个新的视图控制器。整个逻辑关系都在下面的委托方法里，如下代码所示：

```
- (UIPageViewControllerSpineLocation)
pageViewController:(UIPageViewController *)pageViewController
spineLocationForInterfaceOrientation:(UIInterfaceOrientation)orientation;
```

我们现在讲了页面控制器的委托，那数据资源优势怎么一回事呢？一个页面视图控制器的数据资源必须遵守 UIPageViewControllerDataSource 协议。在协议里的两个很重要的方法是：

```
- (UIViewController *)
pageViewController:(UIPageViewController *)pageViewController
viewControllerBeforeViewController:(UIViewController *)viewController;
- (UIViewController *)
pageViewController:(UIPageViewController *)pageViewController
viewControllerAfterViewController:(UIViewController *)viewController;
```

当页面视图控制器已经有一个视图控制器在屏幕里并需要知道下一个即将要显示的视图控制器时将调用第一个方法。当用户决定翻转下一个页面时将发生此事件。当这个视图正在翻转的同时页面视图控制器想要判断哪一个视图控制器需要显示的时候将调用第二个方法。

正如你所看到的那样，Xcode 极大地简化了建立一个基于页面的应用程序。现在你真正需要做的就是给数据模型（Model Controller）提供内容。 继续，假如你要自定义视图控制器的颜色和图像，可以在 Interface Builder 里直接修改.xib 文件或者在每个视图控制器的执行文件里修改代码。

2.21.4. 参考

XXX

2.22. 使用 UIPopoverController 显示弹出画面（弹出框）

2.22.1. 问题

想要使内容不是全屏显示在 iPad 上。

2.22.2. 方案

使用弹出框。

2.22.3. 讨论

弹出框是用于在 iPad 屏幕上显示额外信息的。其中有一个例子是在 iPad 里的 Safari 程序。当用户点击书签按钮时，用户将会在屏幕上看到弹出框显示的书签内容（图 2-71）。

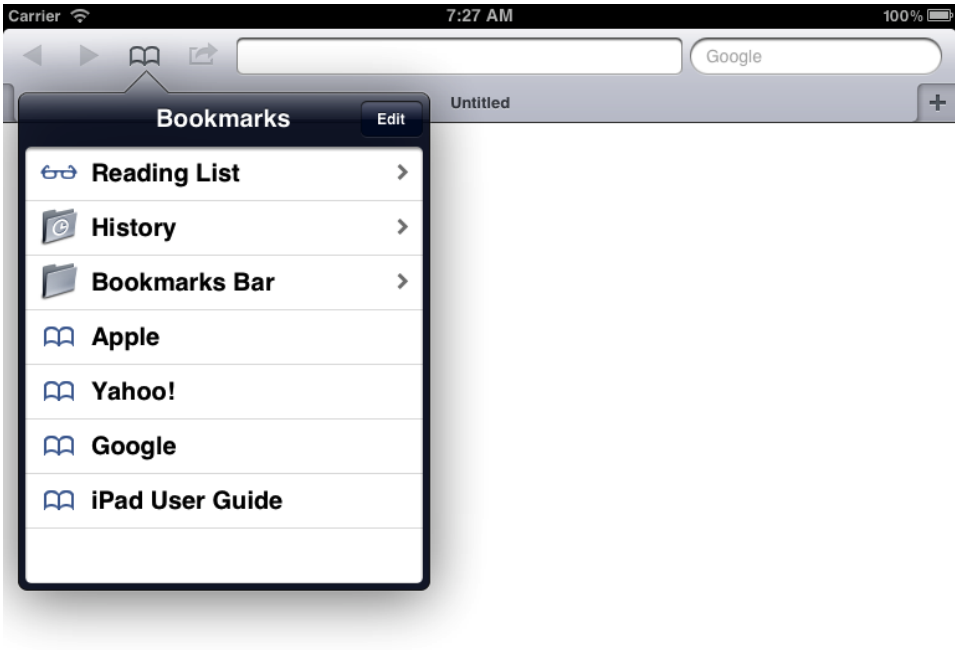


图 2-71 iPad 上的 Safari 程序上的书签弹出框。

弹出框的默认行为是当用户点击弹出框区域外的某个区域时，弹出框将自动消失。假如用户点击屏幕上的特定区域时，我们可以让弹出框不消失，待会我们将会看到。弹出框是通过视图控制器来显示内容的。注意，你也可以将弹出框里的导航控制器作为 UIViewController 导航控制器的子类。

这种弹出框只能用于 iPad 设备。假如有一个视图控制器的代码在 iPad 和 iPhone 上都是用的话，那么你得确定除了 iPad 外，在其他设备里没有实例化一个弹出框（popover）.

弹出框有两种方式显示给用户：

来自内部的一个导航按钮，UIBarButtonItem 的一个实例。
来自内部的一个视图里的矩形区域。

当设备方向改变了（这个设备是旋转的），弹出框不是消失就是暂时隐藏。你需要确保当方向改变稳定后重新显示弹出框时能给用户一个好的体验，假如可能的话。在某些情况下，当方向改变时弹出框也许会自动消失。例如，当用户在全屏幕模式下点击一个导航按钮在屏幕上将会显示一个弹出框。当设备方向是垂直的，那么你得程序将会被设计成另一种形式，导航条上的导航按钮将会被移除。现在，正确的用户体验是当设备的方向转向垂直方向时隐藏弹出框可与之关联的导航按钮。虽然在一些例子里，你需要发挥弹出框的一点特色给用户，因为不是所有的实例都像前面提到的案例那样处理设备的方向这样简单。

为了创建演示弹出框的程序，我们首先需要根据需求给出一个策略：我们想要建立一个在导航控制器里加载了一个视图控制器的这么一个程序。这个根视图控制器会在这个导航条上的右角显示+。当点击设备上的+按钮时，将会显示一个带两个按钮的弹出框。第一个按钮叫做“Photo”，第二个按钮叫做“Audio”。当这个导航按钮在 iPhone 设备上被点击时，将弹出一个带三个按钮的警告框。其中两个是前面提到过的加上一个取消按钮共三个按钮，因此用户可以按取消按钮来取消掉警告框。这些按钮不管在 iPhone 上的警告还是 iPad 上的弹出框被点击时，其实并没有做任何事。我们仅仅做的是取消警告和弹出框。

让我们继续并在 Xcode 上创建一个单视图的普通工程，并将新建工程命名为 Displaying_Popovers_with_UIPopoverControllerViewController。然后到程序的委托头文件并在文件里先定义一个导航控制器：

```
#import <UIKit/UIKit.h>
@class Displaying_Popovers_with_UIPopoverControllerViewController;
@interface Displaying_Popovers_with_UIPopoverControllerAppDelegate
: UIResponder <UIApplicationDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong)
Displaying_Popovers_with_UIPopoverControllerViewController *viewController;
@property (nonatomic, strong) UINavigationController *navigationController;
@end
```

接下来，我们会在程序委托的执行文件结合并实例化导航控制器而不是视图控制器，我们会将导航控制器展现给用户：

```
#import "Displaying_Popovers_with_UIPopoverControllerAppDelegate.h"
#import "Displaying_Popovers_with_UIPopoverControllerViewController.h"
@implementation Displaying_Popovers_with_UIPopoverControllerAppDelegate
@synthesize window = _window;
@synthesize viewController = _viewController;
@synthesize navigationController;
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
self.window = [[UIWindow alloc] initWithFrame:
[[UIScreen mainScreen] bounds]];
UIUserInterfaceIdiom uiIdiom = [[UIDevice currentDevice] userInterfaceIdiom];
NSString *viewControllerClass =
@"Displaying_Popovers_with_UIPopoverControllerViewController_iPad";
if (uiIdiom == UIUserInterfaceIdiomPhone) {
viewControllerClass =
@"Displaying_Popovers_with_UIPopoverControllerViewController_iPhone";
}
self.viewController =
[[Displaying_Popovers_with_UIPopoverControllerViewController alloc]
initWithNibName:viewControllerClass
bundle:nil];
self.navigationController = [[UINavigationController alloc]
initWithRootViewController:self.viewController];
self.window.
self.viewController =
[[Displaying_Popovers_with_UIPopoverControllerViewController alloc]
initWithNibName:viewControllerClass
bundle:nil];
self.navigationController = [[UINavigationController alloc]
initWithRootViewController:self.viewController];
self.window.rootViewController = self.navigationController;
[self.window makeKeyAndVisible];
return YES;
}
```

完成后，我们需要在视图控制器的定义文件里定义一个 UIPopoverController 类型的属性：

```
#import <UIKit/UIKit.h>
@interface Displaying_Popovers_with_UIPopoverControllerViewController
: UIViewController <UIAlertViewDelegate>
@property (nonatomic, strong) UIPopoverController *popoverController;
@property (nonatomic, strong) UIBarButtonItem *barButtonAdd;
```

```
@end
```

你可以看到我们同样也在视图控制器里定义了一个叫 barButtonAdd 的属性。这是我们要添加到导航条上的导航按钮，我们的计划是当用户点击这个按钮时显示这个弹出框（你可以在 2.11 节里了解更多的内容）。但是，但是要确保是在 iPad 里实例化这个弹出框的。在编译执行根视图控制器和导航按钮之前，我们需要先创建一个 UIViewController 的子类，名为 PopoverContViewController。之后将会在弹出框里的视图控制器里显示内容。参看 2.7 节获取更多的关于视图控制器并如何创建谈们的更多信息。

显示在弹出框里的内容视图控制器有两个按钮。但是，这个视图控制器需参照弹出控制器，目的是为了当用户点击任何一个按钮弹出框将会消失。为此，我们需要在内容控制器里定义一个属性来参照弹出框：

```
the popover:
#import <UIKit/UIKit.h>
@interface PopoverContentViewController : UIViewController
@property (nonatomic, strong) UIButton *buttonPhoto;
@property (nonatomic, strong) UIButton *buttonAudio;
/* We shouldn't define this as strong. That will create a retain cycle
between the popover controller and the content view controller since the
popover controller retains the content view controller and the view controller will
retain the popover controller */
@property (nonatomic, weak) UIPopoverController *popoverController;
@end
Now we will go and synthesize these properties in the implementation file of our content
view controller:
#import "PopoverContentViewController.h"
@implementation PopoverContentViewController
@synthesize buttonPhoto;
@synthesize buttonAudio;
@synthesize popoverController;
...
```

然后，我们需要在内容视图控制器里创建两个按钮，并连接它们对应的两个方法。这两个方法需要关注显示在视图控制器里的将要消失的弹出框。记住，弹出框控制器会负责将它本身签名到内容视图控制器的 popoverController 属性;

```
#import "PopoverContentViewController.h"
@implementation PopoverContentViewController
@synthesize buttonPhoto;
@synthesize buttonAudio;
@synthesize popoverController;
- (BOOL) isInPopover{
Class popoverClass = NSClassFromString(@"UIPopoverController");
if (popoverClass != nil &&
UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad &&
self.popoverController != nil){
return YES;
} else {
return NO;
}
}
- (void) gotoAppleWebsite:(id)paramSender{
if ([self isInPopover]){
/* Go to website and then dismiss popover */
[self.popoverController dismissPopoverAnimated:YES];
} else {
/* Handle case for iPhone */
```

```
}
}
- (void) gotoAppleStoreWebsite:(id)paramSender{
if ([self isInPopover]){
/* Go to website and then dismiss popover */
[self.popoverController dismissPopoverAnimated:YES];
} else {
/* Handle case for iPhone */
}
}
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
self.contentSizeForViewInPopover = CGSizeMake(200.0f, 125.0f);
CGRect buttonRect = CGRectMake(20.0f,
20.0f,
160.0f,
37.0f);
self.buttonPhoto = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[self.buttonPhoto setTitle:@"Photo"
forState:UIControlStateNormal];
[self.buttonPhoto addTarget:self
action:@selector(gotoAppleWebsite:)
forControlEvents:UIControlEventTouchUpInside];
self.buttonPhoto.frame = buttonRect;
[self.view addSubview:self.buttonPhoto];
buttonRect.origin.y += 50.0f;
self.buttonAudio = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[self.buttonAudio setTitle:@"Audio"
  forState:UIControlStateNormal];
[self.buttonAudio addTarget:self
action:@selector(gotoAppleStoreWebsite:)
forControlEvents:UIControlEventTouchUpInside];
self.buttonAudio.frame = buttonRect;
[self.view addSubview:self.buttonAudio];
}
- (void)viewDidUnload{
[super viewDidUnload];
self.buttonPhoto = nil;
self.buttonAudio = nil;
}
- (BOOL)shouldAutorotateToInterfaceOrientation
:(UIInterfaceOrientation)interfaceOrientation{
return YES;
}
@end
```

好的，现在回到根视图控制器并结合属性：

```
#import "Displaying_Popovers_with_UIPopoverControllerViewController.h"
#import "PopoverContentViewController.h"
@implementation Displaying_Popovers_with_UIPopoverControllerViewController
@synthesize popoverController;
@synthesize barButtonAdd;
...
```

现在在根视图控制器的 ViewDidLoad 方法里，我们会根据设备的类型创建导航按钮，当导航按钮被点击时，将会显示一个弹出框（在 iPad 里）或者一个警告视图（在 iPhone 里）：

```
- (void)viewDidLoad{
[super viewDidLoad];
/* See if this class exists on the iOS running the app */
Class popoverClass = NSClassFromString(@"UIPopoverController");
if (popoverClass != nil &&
UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad){
PopoverContentViewController *content =
[[PopoverContentViewController alloc] initWithNibName:nil
bundle:nil];
self.popoverController = [[UIPopoverController alloc]
initWithContentViewController:content];
content.popoverController = self.popoverController;
self.barButtonAdd = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
target:self
action:@selector(performAddWithPopover:)];
} else {
self.barButtonAdd = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
target:self
action:@selector(performAddWithAlertView:)];
}
[self.navigationItem setRightBarButtonItem:self.barButtonAdd
animated:NO];
}
- (void)viewDidUnload{
[super viewDidUnload];
self.barButtonAdd = nil;
}
- (BOOL)shouldAutorotateToInterfaceOrientation
:(UIInterfaceOrientation)interfaceOrientation{
r
```

弹出框在被实例化之后在内容视图控制器里设置一个他自己的参照。这很重要。一个弹出框控制器没有内容视图控制器不能被初始化。一旦弹出框在有内容视图控制器的情况下被初始化，你可以在弹出控制器改变内容视图控制器，但是不是在初始化的时候。

我们已经选择了 performAddWithPopover：这个方法作为调用方法，当+导航按条按钮在 iPad 被点击时。假如设备不是 iPad，我们将会令+导航条按钮去调用 performAddWithAlerView:这个方法，现在执行这些方法并注意警告视图的委托方法，现在在 iPhone 里，我们知道了用户点击的警告视图按钮了：

```
- (NSString *) photoButtonTitle{
return @"Photo";
}
- (NSString *) audioButtonTitle{
return @"Audio";
}
- (void) alertView:(UIAlertView *)alertView
didDismissWithButtonIndex:(NSInteger)buttonIndex{
NSString *buttonTitle = [alertView buttonTitleAtIndex:buttonIndex];
if ([buttonTitle isEqualToString:[self photoButtonTitle]]){
/* Adding a photo ... */
```

```
}
else if ([buttonTitle isEqualToString:[self audioButtonTitle]]){
/* Adding an audio... */
}
}
- (void) performAddWithAlertView:(id)paramSender{
[[[UIAlertView alloc] initWithTitle:nil
message:@"Add..."
delegate:self
cancelButtonTitle:@"Cancel"
otherButtonTitles:
[self photoButtonTitle],
[self audioButtonTitle], nil] show];
}
- (void) performAddWithPopover:(id)paramSender{
[self.popoverController
presentPopoverFromBarButtonItem:self.barButtonAdd
permittedArrowDirections:UIPopoverArrowDirectionAny
animated:YES];
}
```

假如你现在 iPad 模拟器及上运行这个程序，并在导航条点击+按钮，你将会看到如图 2-72 所示的一个接口。

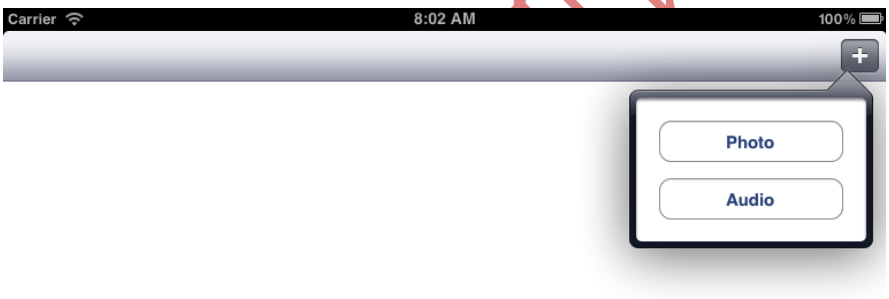


图 2-72 当点击了导航栏按钮显示 popover

假如在 iPhone 上运行这个程序并在导航条点击+按钮，你将会如图 2-73 所示的结果：

图 2-73 在通用应用程序中 alert view 代替了 popover

我们会使用内容视图控制器的一个重要属性，它是 contentSizeForViewInPopover 属性。当显示它的内容视图控制器时，它将会自动读取这个属性值并自动调整它的大小至适合。我们也会在根视图控制器使用属性的eingPopoverFromBarButtonItem:permittedArrowDirections:animated:方法在一个导航按钮上显示弹出框。这个方法的第一个参数是导航条按钮，弹出框在这里显示。第二个参数在显示时确定了弹出框，并关联这个弹出框的对象。例如，在图 2-72 你可以看到弹出框的箭头是指向导航条按钮的。你传递给这个参数的值必须是 UIPopoverArrowDirection 类型的：

```
enum {
UIPopoverArrowDirectionUp = 1UL << 0,
UIPopoverArrowDirectionDown = 1UL << 1,
UIPopoverArrowDirectionLeft = 1UL << 2,
UIPopoverArrowDirectionRight = 1UL << 3,
UIPopoverArrowDirectionAny = UIPopoverArrowDirectionUp |
UIPopoverArrowDirectionDown |
UIPopoverArrowDirectionLeft |
UIPopoverArrowDirectionRight,
UIPopoverArrowDirectionUnknown = NSUIntegerMax
};
typedef NSUInteger UIPopoverArrowDirection;
```

2.22.4. 参考

XXX

2.23. 使用 UIProgressView 显示进度条

2.23.1. 问题

你想在屏幕上显示一个进度条来描述某任务的进度。例如，从 URL 下载一个文件的进度条。

2.23.2. 方案

实例化一个 UIProgressView 类型的视图并将它放置在另一个视图里。

2.23.3. 讨论

一个进度视图是程序员所熟悉的进度条。一个例子是：进度视图描述一个数的到来。进度视图一般是给用户显示一个任务的进度，这个任务是已经明确了开始和终止点。例如，下载 30 个文件就是一个很明确的任务，它有确定的开始很终止点。当全部 30 文件下载完毕任务也就完成了。进度视图是 UIProgressView 的一个实例，并被此类设计好的初始化器初始化，使用 initWithProgressViewStyle：这个方法。这个方法是用来创建进度条风格的，通过参数的设置即可。这个参数是 UIPogressViewStyle 类型，并可以是一下所示的值：

UIProgressViewStyleDefault
这个进度视图的默认风格。其中有一个例子就是显示数字的倒计时的进度视图。

UIProgressViewStyleBar

跟 UIProgressViewStyleDefault 相似，但是这是使用在要被添加到工具条上的进度视图的参数。

UIProgressView 的一个实例定义了一个叫 progress 的属性（属于 float 类型值）。这个属性告示 iOS 如何在进度视图上显示进度条。这个值必须是在+0 到+1.0 之间。假如这个值是+0，那么将会显示一个未开始的进度条。这个只是+1.0 时，则显示一个进度是 100%的进度条。进度条显示在数值的到来是 0.5（或者是 50%）。为了创建一个进度视图，那么现在就开始创建一个跟之前看到的数值到来的相似的进度条吧。首先，我们需要给进度视图顶一个进度：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController
@property (nonatomic, strong) UIProgressView *progressView;
@end
```

结合下面的属性：

```
#import "ViewController.h"
@implementation ViewController
@synthesize progressView;
...
```

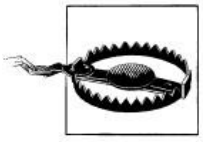
最后我们会实例化一个 UIProgressView 类型的对象：

```
- (void)viewDidLoad{
[super viewDidLoad];
self.view.backgroundColor = [UIColor whiteColor];
self.progressView = [[UIProgressView alloc]
initWithProgressViewStyle:UIProgressViewStyleBar];
self.progressView.center = self.view.center;
self.progressView.progress = 0.5f;
[self.view addSubview:self.progressView];
}
- (void)viewDidUnload{
[super viewDidUnload];
```

```
self.progressView = nil;
}
```

很明显，创建一个进度视图是很简单的。所需要注意的就是能够正确地显示进度，因为进度视图的 progress 属性值必须是在+0 到 1.0 之间（这是一个正常化的值），所以假如你有 30 个任务需要被监控，而目前已近完成了 20 个任务，你需要指定下面的方程来表示进展程度：

```
self.progressView.progress = 20.0f / 30.0f;
```



数值 20 和 30 需要以浮点型传递给方程式是为了告诉编译器这个除法是发生在浮点数值上面的，产生一个十进制的数值。但是，你提供的 20/30 的结果给编译器赋值给进度视图的进度里面，

你会从除法里得到 0，因为这个除法会变两个整数并清零，20 不能被 30 整除，所以结果会是 0.最终将进度视图的进度置为 0，这并不是你期望的结果。

2.23.4. 参考

XXX

2.24. 监听和响应键盘通知

2.24.1. 问题

允许用户在 UI 上输入文本，使用文本框或者文本视图或者其他需要键盘存在的组件。但是，当键盘从屏幕上弹出时将遮蔽一大块用户界面，显示这些是没用任何用处的。那么怎么解决这个问题呢？

2.24.2. 方案

监听键盘通知并将 UI 组件移动到上面或者下面，或者干脆改变这些组件至键盘遮住屏幕，给用户它们希望看见的视图。更多的关于键盘发出的通知，请参考这章的讨论部分。

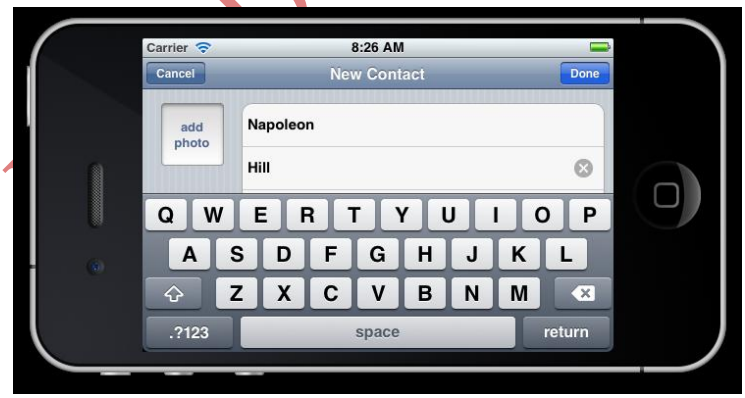
2.24.3. 讨论

iOS 设备没有实体的键盘。它们只有个虚拟键盘，当用户在输入框输入时将会弹出虚拟键盘，如文本框（更多关于 UITextField 的信息请看 2,.14 节）或者文本视图（UITextView，更多信息请看 2.15 节）。在 iPad 里，用户甚至可以滑动键盘使它向上向下移动。当你在设计用户界面时需要注意一些情况。你可以跟你们公司的 UI 设计师合作并让他们知道在 iPad 上是可以滑动键盘的。他们在做艺术工作是需要知道这些东西。我们将会在这个章节里讨论这些特殊情况。

先看看 iPhone 上的键盘吧。这个键盘可以在垂直和水平模式的情况下显示。在垂直情况下，iPhone 上的键盘如下图所示：

图 2-74 在 iPhone 中垂直模式下的键盘

iPhone 上的键盘在水平模式下如图 2-75 所示：



2-75 在 iPhone 中水平模式下的键盘

但是在 iPad 上，键盘就有点不一样了。其中最明显的不同就是 iPad 上的键盘比 iPhone 上的大得多，因为 iPad 的物理尺寸比较大。用户也可以滑动上面键盘的位置。这里有一个在垂直模式下的 iPad 上的键盘：

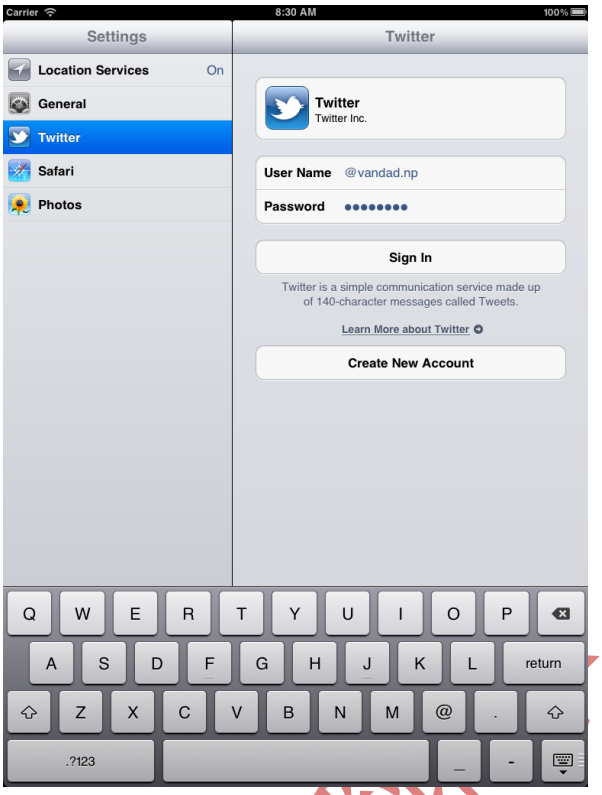


图 2-76 在 iPad 中垂直模式下的键盘

水平模式下的键盘里面的键和垂直下的键盘是一样的，就是在宽度上明显的宽：

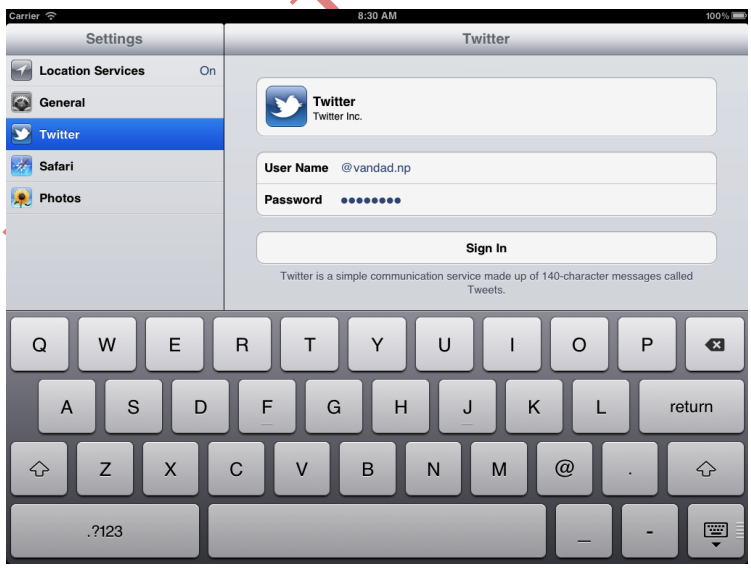


图 2-77 在 iPad 中水平模式下的键盘

这里有一个在 iPad 上滑动键盘的例子，在水平模式下的。（水平模式和垂直模式下都能滑动）：

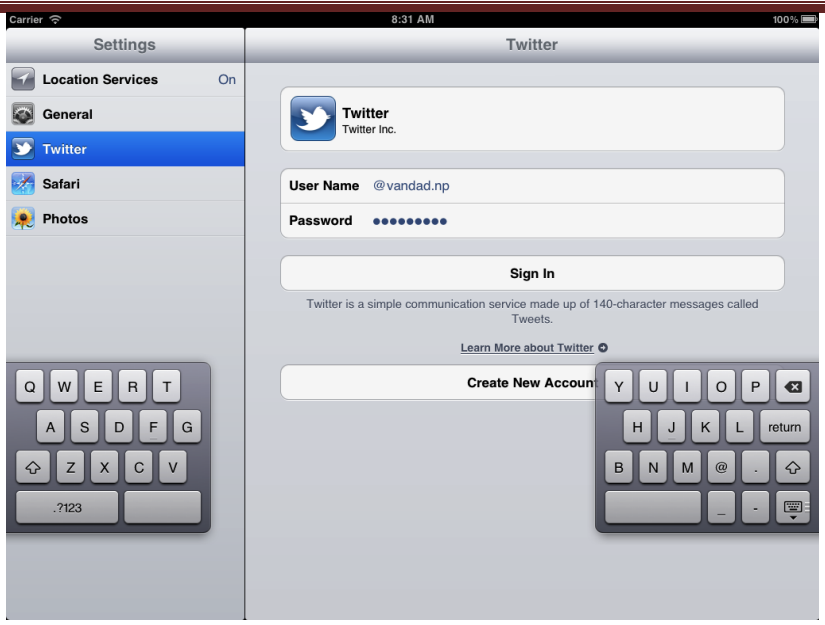


图 2-78 在 iPad 中水平模式下的拆分键盘

iOS 发布了很多关于屏幕上键盘的通知。下面列出了这些通知的简要解释：

UIKeyboardWillShowNotification

当键盘即将要显示的时候将会发出这个通知。这个通知包含了用户信息库，库里面包括了键盘的各种信息，键盘将以动画的形式显示在屏幕上。

UIKeyboardDidShowNotification

当键盘显示在屏幕上时将发出这个通知。

UIKeyboardWillHideNotification

当键盘将要从屏幕上移除时将会发出此通知。通知里包含了用户信息库，库里包括了各种关于键盘信息的详细信息，当键盘隐藏时的动画，动画的持续时间，等等。

UIKeyboardDidHideNotification

当键盘完全隐藏后将发出此通知。

正如前面所提到的，只有 UIKeyboardWillShowNotification 和 UIKeyboardWillHideNotification 这两个通知携带了用户信息。下面是这些信息的关键字，也许这对你有用。

UIKeyboardAnimationCurveUserInfoKey

这个关键字的值指明了动画的类型，用来显示和隐藏键盘。这个关键字包含了一个 NSNumber 类型的值，此类型包含了一个 NSUInteger 类型无符号整数

UIKeyboardAnimationDurationUserInfoKey

这个键值指明了键盘显示或隐藏的动画所用的时间。这个键包含一个 NSNumber 类型的值，此类包含一个 double 类型的双字节值。

UIKeyboardFrameBeginUserInfoKey

这个键值指明了键盘在动画之前的框架。假如键盘将要显示时，在显示之前将这个框架传递给这个动画。假如键盘已经显示了并即将要隐藏时，这个框架将会传递给这个隐藏动画，在键盘消失之前。这个键包含了一个 CGRect 类型的值。

UIKeyboardFrameEndUserInfoKey

这个键值指明了动画完成后的键盘框架。假如键盘即将要显示时，这个框架将会在键盘完全显示后传

递给键盘。。假如键盘已经完全显示，而且将要隐藏时，在完全隐藏后这个框架将会传递给键盘。这个键值包含了一个 CGRect 类型的值。

这个框架被作为 iOS 键盘以开始和结束来发布的，它不会考虑设备的方向。我们需要将 CGRect 值转换成一个关于坐标的值，接下来将会在此章节看到。

让我们看看一个简单的例子。在视图控制器里创建一个简单的桌面并在键盘显示之前改变它的外观属性（the margins from top, right, bottom and left side of the table view）。我们会给这个桌面分配 100 个单元格的大小，使它足够大而至填充整个屏幕，在 iPhone 和 iPad 上的普通程序都能适用。现在先从视图控制器的头文件开始：

```
  #import <UIKit/UIKit.h>
@interface ViewController : UIViewController
<UITableViewDelegate, UITableViewDataSource, UITextFieldDelegate>
@property (nonatomic, strong) UITableView *myTableView;
@end
```

现在我们需要结合视图控制器执行文件里的桌面属性：

```
#import "ViewController.h"
@implementation ViewController
@synthesize myTableView;
...
```

接下来，当视图被加载时实例化桌面视图，并且在 viewDidUnload 方法里设置权衡桌面视图的内存管理：

```
- (void)viewDidLoad{
[super viewDidLoad];
self.myTableView = [[UITableView alloc]
initWithFrame:self.view.bounds
style:UITableViewStyleGrouped];
self.myTableView.delegate = self;
self.myTableView.dataSource = self;
self.myTableView.autoresizingMask = UIViewAutoresizingFlexibleWidth |
UIViewAutoresizingFlexibleHeight;
[self.view addSubview:self.myTableView];
}
- (void)viewDidUnload{
[self setMyTableView:nil];
[super viewDidUnload];
}
```

完成后，需要将桌面视图填充 100 个单元格，在每个单元格里创建一个文本框作为附件视图。我们做着这些是为了让用户触发键盘使之弹出。假如没有这些文本框或者其他用户输入的方法，键盘将不会显示在屏幕上，所以需要做一下工作：

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField{
/* Make sure the Done button on the keyboard for each text field
(accessory views of each cell) dismisses the keyboard */
[textField resignFirstResponder];
return YES;
}
```

```
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView{
return 1;
}
- (NSInteger) tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
return 100;
    }
- (UITableViewCell *) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{
UITableViewCell *result = nil;
static NSString *CellIdentifier = @"CellIdentifier";
result = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
if (result == nil){
result = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier];
result.selectionStyle = UITableViewCellSelectionStyleNone;
}
result.textLabel.text = [NSString stringWithFormat:
@"Cell %ld", (long)indexPath.row];
CGRect accessoryRect = CGRectMake(0.0f,
0.0f,
150.0f,
31.0f);
UITextField *accesssory = [[UITextField alloc] initWithFrame:accessoryRect];
accesssory.borderStyle = UITextBorderStyleRoundedRect;
accesssory.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
accesssory.placeholder = @"Enter Text";
accesssory.delegate = self;
result.accessoryView = accesssory;
return result;
}
```

棒极了。假如现在在 iPhone 模拟器上运行这个程序，将会看到如下图所示的画面：

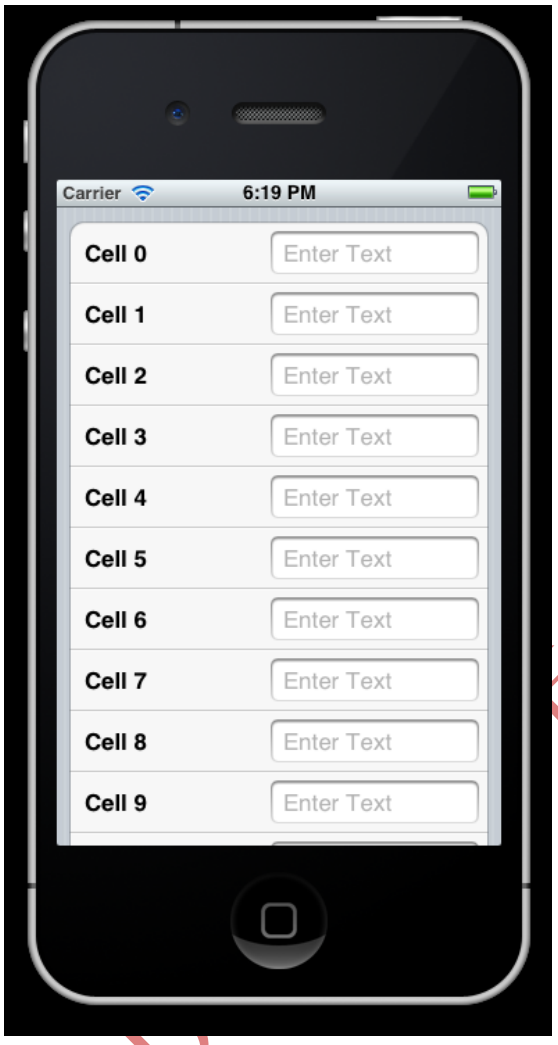


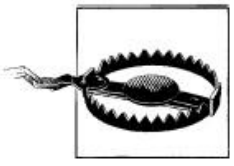
图 2-79 每个 cell 具有 text 控件的标示图 table view

现在点击第一个文本框（在第一个单元格）。然后滚动桌面视图直到最后一个单元格，看一下将会发生什么情况！你将看不见 5-6 单元格，对吧？在垂直模式下看这个程序如下图所示：

图 2-80 键盘遮住了 tableview 的下半部分

我们现在要做的就是监听 UIKeyboardWillShowNotification 和 UIKeyboardWillHideNotification 的通知并通过插入以下代码来调整桌面视图：

```
- (void) viewDidAppear:(BOOL)paramAnimated{
[super viewDidAppear:paramAnimated];
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
[center addObserver:self
selector:@selector(handleKeyboardWillShow:)
name:UIKeyboardWillShowNotification
object:nil];
[center addObserver:self
selector:@selector(handleKeyboardWillHide:)
name:UIKeyboardWillHideNotification
object:nil];
}
- (void) viewDidDisappear:(BOOL)paramAnimated{
[super viewDidDisappear:paramAnimated];
[[NSNotificationCenter defaultCenter] removeObserver:self];
}
```



一个很常见的错误是：即使某个视图控制器不在屏幕上，一些程序员还会保持监听键盘通知。他们在 viewDidLoad 方法里开始监听通知，当没有启用 ARC 时，观察者在

viewDidUnLoad 或者 dealloc 方法里移除这些通知。这是一个有问题的方法，因为当视图离开屏幕时，你不需要调整屏幕上的任何组件以适应其他视图控制器。记住，像其他任何通知一样，键盘通知是会给所有的观察者发出通知，所以你需要确保那些离开屏幕的视图不要对键盘通知再做出反应了。

既然已经开始监听键盘通知，我们可以执行已经在 NSNotifiacationCenter 注册的观察者方法。HandleKeyboardWillShow:这个方法负责将桌面视图插入到内容中：

```
- (void) handleKeyboardWillShow:(NSNotification *)paramNotification{
NSDictionary *userInfo = [paramNotification userInfo];
NSValue *animationCurveObject =
[userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey];
NSValue *animationDurationObject =
[userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey];
NSValue *keyboardEndRectObject =
[userInfo valueForKey:UIKeyboardFrameEndUserInfoKey];
NSUInteger animationCurve = 0;
double animationDuration = 0.0f;
CGRect keyboardEndRect = CGRectMake(0, 0, 0, 0);
[animationCurveObject getValue:&animationCurve];
[animationDurationObject getValue:&animationDuration];
[keyboardEndRectObject getValue:&keyboardEndRect];
[UIView beginAnimations:@"changeTableViewContentInset"
context:NULL];
[UIView setAnimationDuration:animationDuration];
[UIView setAnimationCurve:(UIViewAnimationCurve)animationCurve];
UIWindow *window = [[[UIApplication sharedApplication] delegate] window];
CGRect intersectionOfKeyboardRectAndWindowRect =
CGRectIntersection(window.frame, keyboardEndRect);
CGFloat bottomInset = intersectionOfKeyboardRectAndWindowRect.size.height;
self.myTableView.contentInset = UIEdgeInsetsMake(0.0f,
0.0f,
bottomInset,
0.0f);
NSIndexPath *indexPathOfOwnerCell = nil;
/* Also, make sure the selected text field is visible on the screen */
NSInteger numberOfCells = [self.myTableView.dataSource
tableView:self.myTableView
numberOfRowsInSection:0];
/* So let's go through all the cells and find their accessory text fields.
Once we have the refernece to those text fields, we can see which one of
them is the first responder (has the keyboard) and we will make a call
to the table view to make sure after the keyboard is displayed,
that specific cell is NOT obstructed by the keyboard */
for (NSInteger counter = 0;
counter < numberOfCells;
counter++){
NSIndexPath *indexPath = [NSIndexPath indexPathForRow:counter
inSection:0];
UITableViewCell *cell = [self.myTableView cellForRowAtIndexPath:indexPath];
UITextField *textField = (UITextField *)cell.accessoryView;
if ([textField isKindOfClass:[UITextField class]] == NO){
continue;
}
if ([textField isFirstResponder]){
indexPathOfOwnerCell = indexPath;
break;
}
}
```

```
[UIView commitAnimations];
if (indexPathOfOwnerCell != nil){
[self.myTableView scrollToRowAtIndexPath:indexPathOfOwnerCell
atScrollPosition:UITableViewScrollPositionMiddle
animated:YES];
}
}
```

下面是在这个方法里所做的工作，按顺序解释：

1、返回不同的键盘动画属性，包含动画时间，路径和键盘完全显示后的框架。使用 UIKeyboardWillShowNotification 通知的用户信息库来完成此工作。
2、然后创建一个动画模块，使用它改变桌面视图插图的内容。作此之前，我们需要知道桌面视图需要多大的面积来包含键盘。
3、使用 CGRectIntersection 函数，将返回视窗框架和当键盘完全显示后的框架之间的路口。使用这个技术，我们可以得到键盘完全显示后视窗需的面积大小，所以我们可以插入桌面视图设置底部的内容。
4、设置动画模块的属性，比如它的路径、时间和最终呈现出来的动画。当键盘弹出之后，之前提到的动画模块会改变桌面视图插图的内容来适应屏幕上可视界面。

CGRectIntersection 函数在 CoreGraphics 的框架被定义并执行。为了能编译这段代码，你需要确保你的程序和之前提到的框架关联起来。你可以通过在 Xcode 里选择程序的图标，然后在右手边的列表里选择你的目标。现在你需要在 Link Binary 的库里选择 Build Phases 并确保你的目标和框架关联起来了。假如没有，你可以在盒子里按+按钮将框架添加到你的程序已经关联起来的框架列表里。

现在我们需要转移到 handleKeyBoardWillHide：方法的执行文件：

```
- (void) handleKeyboardWillHide:(NSNotification *)paramNotification{
if (UIEdgeInsetsEqualToEdgeInsets(self.myTableView.contentInset,
UIEdgeInsetsZero)){
/* Our table view's content inset is intact so no need to reset it */
return;
}
NSDictionary *userInfo = [paramNotification userInfo];
NSValue *animationCurveObject =
[userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey];
NSValue *animationDurationObject =
[userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey];
NSValue *keyboardEndRectObject =
[userInfo valueForKey:UIKeyboardFrameEndUserInfoKey];
NSUInteger animationCurve = 0;
double animationDuration = 0.0f;
CGRect keyboardEndRect = CGRectMake(0, 0, 0, 0);
[animationCurveObject getValue:&animationCurve];
[animationDurationObject getValue:&animationDuration];
[keyboardEndRectObject getValue:&keyboardEndRect];
[UIView beginAnimations:@"changeTableViewContentInset"
context:NULL];
[UIView setAnimationDuration:animationDuration];
[UIView setAnimationCurve:(UIViewAnimationCurve)animationCurve];
self.myTableView.contentInset = UIEdgeInsetsZero;
[UIView commitAnimations];
}
```

在 handleKeyboardWillHide:方法里我们做了以下事情：

1、假如桌面视图的内容插图改变了，将会被发现。假如桌面视图的内容插图没有改变，我们实际上没有

做任何动作。

我们只是猜测我们去的是错误的通知或者其他视图控制器对象发出的通知。

2、然后使用 UIKeyboardWillHidNotification 通知的用户信息库取得键盘动画的时间和路径。使用这个信息，我们可以创建动画模块。

3、我们将视图内容插图设置成 UIEdgeInsetsZero 并呈现出动画。

正如之前提到的那样，当键盘通知被发送时，创建桌面视图的开始和结束框架时将不会考虑当前设备方向。例如，记录在 iPhone 垂直方向的视图控制器的 handleKeyboardWillShow：方法的键盘的结束框架，将会取得如下的值：

```
{{0, 264}, {320, 216}}
```

假如你现在将方向旋转到水平方向并重新记录这个值，你将会在控制窗口打印出如下的值：

```
{{0, 0}, {162, 480}}
```

很明显这些值是错误的。正如你看到的，键盘的 Y 坐标是 0，而我们都知道当键盘在 iPhone 水平模式下显示时，Y 坐标显然不是 0，宽度是整个屏幕的宽度，水平模式下这显然不是 162.0，高度是屏幕高度的一半，这里的 480 显然是错的。数据错误的原因是当 iOS 报告这些数据时是不考虑水平模式下的情况。报告到您的应用程序框架是在应用程序的主窗口的坐标系统。因此，要转换这些报告的框架窗口视图的坐标系统的坐标系，使用您认为的转换 tRect：fromView：方法和通过您的应用程序作为 fromView 的窗口参数：

现在，让我们使用 handleKeyboardWillShow：方法修改从窗口给我们视图的坐标系统的转换：

handleKeyboardWillHide：方法的执行不在用户信息字典中使用。在此方法中，我们始终假设键盘被隐藏，并且它的矩形坐标为（0，0，0，0）。

```
- (void) handleKeyboardWillShow:(NSNotification *)paramNotification{
NSDictionary *userInfo = [paramNotification userInfo];
NSValue *animationCurveObject =
[userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey];
NSValue *animationDurationObject =
[userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey];
NSValue *keyboardEndRectObject =
[userInfo valueForKey:UIKeyboardFrameEndUserInfoKey];
NSUInteger animationCurve = 0;
double animationDuration = 0.0f;
CGRect keyboardEndRect = CGRectMake(0, 0, 0, 0);
[animationCurveObject getValue:&animationCurve];
[animationDurationObject getValue:&animationDuration];
[keyboardEndRectObject getValue:&keyboardEndRect];
UIWindow *window = [[[UIApplication sharedApplication] delegate] window];
/* Convert the frame from window's coordinate system to
our view's coordinate system */
keyboardEndRect = [self.view convertRect:keyboardEndRect
fromView:window];
[UIView beginAnimations:@"changeTableViewContentInset"
context:NULL];
[UIView setAnimationDuration:animationDuration];
[UIView setAnimationCurve:(UIViewAnimationCurve)animationCurve];
CGRect intersectionOfKeyboardRectAndWindowRect =
CGRectIntersection(window.frame, keyboardEndRect);
CGFloat bottomInset = intersectionOfKeyboardRectAndWindowRect.size.height;
```

```
self.myTableView.contentInset = UIEdgeInsetsMake(0.0f,
0.0f,
bottomInset,
0.0f);
NSIndexPath *indexPathOfOwnerCell = nil;
/* Also, make sure the selected text field is visible on the screen */
NSInteger numberOfCells = [self.myTableView.dataSource
tableView:self.myTableView
numberOfRowsInSection:0];
/* So let's go through all the cells and find their accessory text fields.
Once we have the refernece to those text fields, we can see which one of
them is the first responder (has the keyboard) and we will make a call
to the table view to make sure after the keyboard is displayed,
that specific cell is NOT obstructed by the keyboard */
for (NSInteger counter = 0;
counter < numberOfCells;
counter++){
NSIndexPath *indexPath = [NSIndexPath indexPathForRow:counter
inSection:0];
UITableViewCell *cell = [self.myTableView cellForRowAtIndexPath:indexPath];
UITextField *textField = (UITextField *)cell.accessoryView;
if ([textField isKindOfClass:[UITextField class]] == NO){
continue;
}
if ([textField isFirstResponder]){
indexPathOfOwnerCell = indexPath;
break;
}
}
[UIView commitAnimations];
if (indexPathOfOwnerCell != nil){
[self.myTableView scrollToRowAtIndexPath:indexPathOfOwnerCell
atScrollPosition:UITableViewScrollPositionMiddle
animated:YES];
}
}
```

现在在 iPhone 模拟器上运行此程序，旋转模拟器到水平方向，点击其中一个文本框。当键盘显示后，滚动桌面视图到最后一项，确保内容正确显示，对于桌面视图要考虑坐标系系统的转换：

图 2-81 考虑键盘的坐标并正确计算显示的位置

2.24.4. 参考

XXX